Unit2_objectsPartAJLC9_18_2023_Questions.pdf

1. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

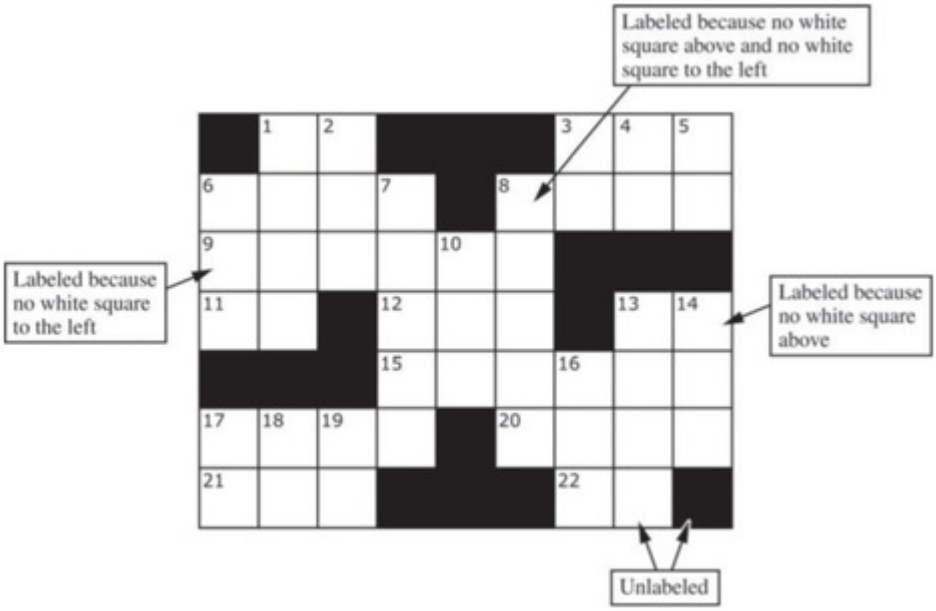
The crossword labeling rule identifies squares to be labeled with a positive number as follows.

A square is labeled with a positive number if and only if

- the square is white and
- the square does not have a white square immediately above it, or it does not have a white square immediately to its left, or both.

The squares identified by these criteria are labeled with consecutive numbers in row-major order, starting at 1.

The following diagram shows a crossword puzzle grid and the labeling of the squares according to the crossword labeling rule.



This question uses two classes, a Square class that represents an individual square in the puzzle and a Crossword class that represents a crossword puzzle grid. A partial declaration of the Square class is shown below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class Square
{
    /** Constructs one square of a crossword puzzle grid.
     *  Postcondition:
     *    - The square is black if and only if isBlack is true.
     *    - The square has number num.
     */
    public Square(boolean isBlack, int num)
    {  /* implementation not shown */  }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

A partial declaration of the Crossword class is shown below. You will implement one method and the constructor in the Crossword class.

```
public class Crossword
{
    /** Each element is a Square object with a color (black or white) and a number.
     *  puzzle[r][c] represents the square in row r, column c.
     *  There is at least one row in the puzzle.
     */
    private Square[][] puzzle;

    /** Constructs a crossword puzzle grid.
     *  Precondition: There is at least one row in blackSquares.
     *  Postcondition:
     *    - The crossword puzzle grid has the same dimensions as blackSquares.
     *    - The Square object at row r, column c in the crossword puzzle grid is black
     *      if and only if blackSquares[r][c] is true.
     *    - The squares in the puzzle are labeled according to the crossword labeling rule.
     */
    public Crossword(boolean[][] blackSquares)
    {  /* to be implemented in part (b) */  }

    /** Returns true if the square at row r, column c should be labeled with a positive number;
     *          false otherwise.
     *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
     *  Precondition: r and c are valid indexes in blackSquares.
     */
    private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
    {  /* to be implemented in part (a) */  }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the Crossword method toBeLabeled. The method returns true if the square indexed by row r, column c in a crossword puzzle grid should be labeled with a positive number according to the crossword labeling rule; otherwise it returns false. The parameter blackSquares indicates which squares in the crossword puzzle grid are black.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

Complete method toBeLabeled below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

b. Write the Crossword constructor. The constructor should initialize the crossword puzzle grid to have the same dimensions as the parameter blackSquares. Each element of the puzzle grid should be initialized with a reference to a Square object with the appropriate color and number. The number is positive if the square is labeled and 0 if the square is not labeled.

Class information for this question

```
public class Square

public Square(boolean isBlack, int num)

public class Crossword

private Square[][] puzzle

public Crossword(boolean[][] blackSquares)
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

Assume that toBeLabeled works as specified, regardless of what you wrote in part (a). You must use toBeLabeled appropriately to receive full credit.

Complete the Crossword constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    - The crossword puzzle grid has the same dimensions as blackSquares.
 *    - The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    - The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
```

2. A *filter* is an object that examines strings and accepts those that meet a particular criterion. The Filter interface is defined as follows.

```
public interface Filter
{
  /** @param text a string to consider for acceptance
   *  @return true if this Filter accepts text; false otherwise
   */
  boolean accept(String text);
}
```

In this question, you will implement one type of Filter and write a method to build a variety of filters. Many different filters can implement the Filter interface. The SimpleFilter and NotFilter are both implementations of the Filter interface and are described as follows:

The SimpleFilter class accepts any string that contains a particular substring. It has one constructor that takes a single parameter that contains the string to be found in the text. The NotFilter is constructed with a single Filter parameter and accepts text if and only if the Filter it was constructed with rejects the text.

For example, the following code segment creates a filter veggieFilter that accepts all strings that contain the substring "vegetable" and another filter noVeggie that accepts all strings that do NOT contain the substring "vegetable".

- Filter veggieFilter = new SimpleFilter("vegetable");
- Filter noVeggie = new NotFilter(veggieFilter);

The following table illustrates the results of several calls to the veggieFilter accept method and the noVeggie accept method.

Method Call	Result
`veggieFilter.accept("vegetable soup")`	`true`
`veggieFilter.accept("fruit salad")`	`false`
`noVeggie.accept("vegetable soup")`	`false`
`noVeggie.accept("fruit salad")`	`true`

a. An OrFilter is a Filter that contains two or more objects that implement the Filter interface (such as a SimpleFilter, an OrFilter, a NotFilter, or other types of Filter objects). The OrFilter accept method accepts a given string if and only if one or more of the filters contained in the OrFilter accepts the string. An OrFilter is constructed with two filters. More filters can be included in the OrFilter by calling the add method. The following code segment illustrates a filter that accepts all strings that contain either one or both of the substrings "vegetable" and "fruit" or that do not contain the string "poison".

```
Filter veggieFilter = new SimpleFilter("vegetable");
Filter fruitFilter = new SimpleFilter("fruit");
Filter noPoison = new NotFilter(new SimpleFilter("poison"));

OrFilter healthyFood = new OrFilter(veggieFilter, fruitFilter);
  //  healthyFood will accept strings containing "vegetable" or "fruit"

healthyFood.add(noPoison);
  //  healthyFood will accept strings containing "vegetable" or "fruit"
  //  or strings without "poison"
```

The following table illustrates the results of several calls to the healthyFood accept method.

Method Call	Result
`healthyFood.accept("vegetable soup is not poison")`	`true`
`healthyFood.accept("fruit salad")`	`true`
`healthyFood.accept("a poisoned apple")`	`false`
`healthyFood.accept("salad")`	`true`

Write the OrFilter class below.

b. Write the buildFilter method that can be used to build filters that meet a set of criteria as described below. The buildFilter method is part of an unrelated class.

You may assume the existence of an AndFilter class that is similar to the OrFilter class except that it accepts a string only if all of its component filters accept the string. Consider the following code segment.

```
Filter appleFilter = new SimpleFilter("apple");
Filter peachFilter = new SimpleFilter("peach");
Filter fruits = new AndFilter(appleFilter, peachFilter);

boolean b1 = fruits.accept("peach cobbler");      // b1 is set to false
boolean b2 = fruits.accept("peaches and apples"); // b2 is set to true
```

The method buildFilter takes two parameters: desirable, a list of two or more strings, and notAllowed, a string. The filter returned by buildFilter should accept all strings that contain at least one string from the desirable array and do not contain the notAllowed string.

The following code segment and table show an example of using the buildFilter method.

String[] primary = {"red", "blue", "yellow"};

Filter primaryColors = buildFilter(primary, "green");

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

Method Call	Result
`primaryColors.accept("Roses are red, violets are blue")`	`true`
`primaryColors.accept("blue grass is really green")`	`false`
`primaryColors.accept("The rainbow has many colors")`	`false`

In writing buildFilter, you may use any of the four filters described (SimpleFilter, NotFilter, OrFilter, and AndFilter). Assume that these classes work as specified, regardless of what you wrote in part (a).

Complete method buildFilter below.

```
/** @param desirable contains strings that are allowed
 *         Precondition: desirable.length > 1
 *  @param notAllowed the string that is not allowed
 *  @return a Filter that accepts strings that contain at least one string
 *          in desirable and do not contain notAllowed.
 */
public static Filter buildFilter(String[] desirable,
                                 String notAllowed)
```

3. A school district would like to get some statistics on its students' standardized test scores. Scores will be represented as objects of the following ScoreInfo class. Each ScoreInfo object contains a score value and the number of students who earned that score.

```
public class ScoreInfo
{
  private int score;
  private int numStudents;

  public ScoreInfo(int aScore)
  {
    score = aScore;
    numStudents = 1;
  }

  /** adds 1 to the number of students who earned this score
   */
  public void increment()
  {  numStudents++;  }

  /** @return this score
   */
  public int getScore()
  {  return score;  }

  /** @return the number of students who earned this score
   */
  public int getFrequency()
  {  return numStudents;  }
}
```

The following Stats class creates and maintains a database of student score information. The scores are stored in sorted order in the database.

```
public class Stats
{
  private ArrayList<ScoreInfo> scoreList;
    // listed in increasing score order; no two ScoreInfo objects contain the same score

  /** Records a score in the database, keeping the database in increasing score order. If no other
   *  ScoreInfo object represents score, a new ScoreInfo object representing score
   *  is added to the database; otherwise, the frequency in the ScoreInfo object representing
   *  score is incremented.
   *  @param score a score to be recorded in the list
   *  @return true if a new ScoreInfo object representing score was added to the list;
   *          false otherwise
   */
  public boolean record(int score)
  { /*  to be implemented in part (a)  */}

  /** Records all scores in stuScores in the database, keeping the database in increasing score order
   *  @param stuScores an array of student test scores
   */
  public void recordScores(int[] stuScores)
  {  /*  to be implemented in part (b)  */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the Stats method record that takes a test score and records that score in the database. If the score already exists in the database, the frequency of that score is updated. If the score does not exist in the database, a new ScoreInfo object is created and inserted in the appropriate position so that the database is maintained in increasing score order. The method returns true if a new ScoreInfo object was added to the database; otherwise, it returns false.
Complete method record below.

```
/** Records a score in the database, keeping the database in increasing score order. If no other
 *  ScoreInfo object represents score, a new ScoreInfo object representing score
 *  is added to the database; otherwise, the frequency in the ScoreInfo object representing
 *  score is incremented.
 *  @param score a score to be recorded in the list
 *  @return true if a new ScoreInfo object representing score was added to the list;
 *          false otherwise
 */
public boolean record(int score)
```

(b) Write the Stats method recordScores that takes an array of test scores and records them in the database. The database contains at most one ScoreInfo object per unique score value. Each ScoreInfo object contains a score and an associated frequency. The database is maintained in increasing order based on the score.

In writing recordScores, assume that record works as specified, regardless of what you wrote in part (a).

Complete method recordScores below.

```
/** Records all scores in stuScores in the database, keeping the database in increasing score order
 *  @param stuScores an array of student test scores
 */
public void recordScores(int[] stuScores)
```

4. Consider the following class.

```
public class SomeMethods

{
public void one(int first)

{ / * implementation not shown * / }

public void one(int first, int second)

{ / * implementation not shown * / }

public void one(int first, String second)

{ / * implementation not shown * / }

}
```

Which of the following methods can be added to the SomeMethods class without causing a compile-time error?

I. public void one(int value)
{ / * implementation not shown * / }

II. public void one (String first, int second)

{ / * implementation not shown * / }

III. public void one (int first, int second, int third)

{ / * implementation not shown * / }

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

(A) I only
(B) I and II only
(C) I and III only
(D) II and III only
(E) I, II, and III

5. An appointment scheduling system is represented by the following three classes: TimeInterval, Appointment, and DailySchedule. In this question, you will implement one method in the Appointment class and two methods in the DailySchedule class.

A TimeInterval object represents a period of time. The TimeInterval class provides a method to determine if another time interval overlaps with the time interval represented by the current TimeInterval object. An Appointment object contains a time interval for the appointment and a method that determines if there is a time conflict between the current appointment and another appointment. The declarations of the TimeInterval and Appointment classes are shown below.

```
public class TimeInterval
{
  // returns true if interval overlaps with this TimeInterval;
  // otherwise, returns false
  public boolean overlapsWith(TimeInterval interval)
  {  /* implementation not shown */  }

  // There may be fields, constructors, and methods that are not shown.
}


public class Appointment
{
  // returns the time interval of this Appointment
  public TimeInterval getTime()
  {  /* implementation not shown */  }

  // returns true if the time interval of this Appointment
  // overlaps with the time interval of other;
  // otherwise, returns false
  public boolean conflictsWith(Appointment other)
  {  /* to be implemented in part (a) */  }

  // There may be fields, constructors, and methods that are not shown.
}
```

a. Write the Appointment method conflictsWith. If the time interval of the current appointment overlaps with the time interval of the appointment other, method conflictsWith should return true, otherwise, it should return false.

   Complete method conflictsWith below.

```
// returns true if the time interval of this Appointment
// overlaps with the time interval of other;
// otherwise, returns false
public boolean conflictsWith(Appointment other)
```

b. A DailySchedule object contains a list of nonoverlapping Appointment objects. The DailySchedule class contains methods to clear all appointments that conflict with a given appointment and to add an appointment to the schedule.

```
public class DailySchedule
{
  // contains Appointment objects, no two Appointments overlap
  private ArrayList apptList;

  public DailySchedule()
  {  apptList = new ArrayList();  }

  // removes all appointments that overlap the given Appointment
  // postcondition: all appointments that have a time conflict with
  //                appt have been removed from this DailySchedule
  public void clearConflicts(Appointment appt)
  {  /* to be implemented in part (b) */  }

  // if emergency is true, clears any overlapping appointments and adds
  // appt to this DailySchedule; otherwise, if there are no conflicting
  // appointments, adds appt to this DailySchedule;
  // returns true if the appointment was added;
  // otherwise, returns false
  public boolean addAppt(Appointment appt, boolean emergency)
  {  /* to be implemented in part (c) */  }

  // There may be fields, constructors, and methods that are not shown.
}
```

Write the DailySchedule method clearConflicts. Method clearConflicts removes all appointments that conflict with the given appointment.

In writing method clearConflicts, you may assume that conflictsWith works as specified, regardless of what you wrote in part (a).

Complete method clearConflicts below.

```
// removes all appointments that overlap the given Appointment
// postcondition: all appointments that have a time conflict with
//                appt have been removed from this DailySchedule
public void clearConflicts(Appointment appt)
```

c. Write the DailySchedule method addAppt. The parameters to method addAppt are an appointment and a boolean value that indicates whether the appointment to be added is an emergency. If the appointment is an emergency, the schedule is cleared of all appointments that have a time conflict with the given appointment and the appointment is added to the schedule. If the appointment is not an emergency, the schedule is checked for any conflicting appointments. If there are no conflicting appointments, the given appointment is added to the schedule. Method addAppt returns true if the appointment was added to the schedule; otherwise, it returns false.

   In writing method addAppt, you may assume that conflictsWith and clearConflicts work as specified, regardless of what you wrote in parts (a) and (b).

   Complete method addAppt below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
// if emergency is true, clears any overlapping appointments and adds
// appt to this DailySchedule; otherwise, if there are no conflicting
// appointments, adds appt to this DailySchedule;
// returns true if the appointment was added;
// otherwise, returns false
public boolean addAppt(Appointment appt, boolean emergency)
```

6. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

1. This question involves managing the costs associated with animal adoption at an animal shelter. Animals are defined by the following `Animal` class.

```
public class Animal
{
 /** Constructs an Animal object with name n, type t, age a, and cost c
 */
 public Animal(String n, String t, double a, int c)
 {  /* implementation not shown */  }

 /** Returns the type of the animal (for example, "dog" or "cat") */
 public String getType()
 {  /* implementation not shown */  }

 /** Returns the age of the animal when it arrived at the shelter, in
 years */
 public double getAge()
 {  /* implementation not shown */  }

 /** Returns the cost, in dollars, of adopting the animal */
 public int getCost()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

Information about an animal shelter is stored in an `AnimalShelter` object, which contains a list of the animals at the shelter. You will write two methods of the `AnimalShelter` class.

```
public class AnimalShelter
{
 /** A list of animals at the shelter, sorted by the age of the animal
```

```
    when it arrived at the
      *  shelter, from least to greatest
      *   Guaranteed not to be null and to contain only non-null entries
     */
    private ArrayList<Animal> allAnimals;

    /** Creates and returns a new Animal object, as described in part (a)
    */
    private Animal createNewAnimal(String name, String type,
          double age)
    {  /* to be implemented in part (a) */  }

    /** Adds an animal to the list allAnimals, as described in part (b) */
    public void addAnimal(String name, String type, double age)
    {  /* to be implemented in part (b) */  }

    // There may be instance variables, constructors, and methods that are
    not shown.
}
```

(a) Write the `AnimalShelter` method `createNewAnimal`. The method creates and returns a new `Animal` object. The animal's name is given by the `name` parameter, the animal's type is given by the `type` parameter, and the animal's age when it enters the shelter is given by the `age` parameter. As animals come into the shelter, the adoption cost, in dollars, is determined by the age of the animal and the number of animals of the same type, according to the following rules.

- If the age of the animal is less than one year and the shelter has fewer than 5 animals with the given `type`, the cost is $25.
- If the age of the animal is less than one year and the shelter has 5 or more animals with the given `type`, the cost is $20.
- Otherwise, the cost is $15.

Complete method `createNewAnimal`.

```
/** Creates and returns a new Animal object, as described in part (a) */
private Animal createNewAnimal(String name, String type,
      double age)
```

(b) Write the `AnimalShelter` method `addAnimal`. The method creates a new `Animal` object with name `name`, type `type`, age `age`, and the adoption cost calculated in part (a). The new animal is added to the `ArrayList allAnimals` so that the sorted order of the list is maintained. The list is ordered by animal age, from least to greatest.

Assume that `createNewAnimal` works as specified, regardless of what you wrote for part (a). You must use `createNewAnimal` appropriately to receive full credit.

Complete method `addAnimal`.

```
/** Adds an animal to the list allAnimals, as described in part (b) */
public void addAnimal(String name, String type,
      double age)
```

(c) The programmer would like to add a method called `getAnimalWithMostInterest`, which returns the animal that has been shown the greatest interest from possible adopters. Interest shown to an animal is measured as the number of people who have asked questions about the animal since the animal arrived at the shelter.

Write a description of how you would change the `Animal` and `AnimalShelter` classes in order to support this modification.

Make sure to include the following in your response.

- Write the method header for the `getAnimalWithMostInterest` method.
- Identify any new or modified variables, constructors, or methods aside from the `getAnimalWithMostInterest` method. **Do not write the program code for this change.**
- Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the `getAnimalWithMostInterest` method. **Do not write the program code for this change.**

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

7. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

3. This question involves a system to maintain a babysitting service. Babysitters are represented by the following `Babysitter` class.

```
public class Babysitter
{
 /** Constructs a new Babysitter object */
 public Babysitter(String name, int exp)
 {  /* implementation not shown */  }

 /** Returns the name of this babysitter */
 public String getName()
 {  /* implementation not shown */  }

 /** Returns the number of years of experience of this babysitter */
 public int getYears()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

A babysitting service manages babysitters through a `BabysitterService` object, which contains a list of babysitters. You will write two methods in the `BabysitterService` class.

```
public class BabysitterService
{
 /** A list of the babysitters available through this service
   *   The list is guaranteed not to be null and to contain only
 non-null entries.
  */
 private ArrayList<Babysitter> sitterList;
```

```
    /** Adds new babysitters to the list of babysitters, avoiding
    duplicates, as described in
     *   part (a)
     */
    public void addNewSitters(String[] names)
    {  /* to be implemented in part (a) */  }

    /** Returns a list of babysitters for whom the babysitting charge for
    the given number of
      *   children and hours is within the given budget, as described in
    part (b)
     */
    public ArrayList<Babysitter> affordableSitters(double budget,
          int numChildren, int hours)
    {  /* to be implemented in part (b) */  }

    // There may be instance variables, constructors, and methods that are
    not shown.
}
```

(a) Write the `BabysitterService` method `addNewSitters`, which has as a parameter an array of strings representing the names of new babysitters to be added to the service. The following actions are taken for each name in the array of babysitter names.

- If a babysitter with the name already appears in `sitterList`, no action is taken.
- If a babysitter with the name is not found in `sitterList`, a new `Babysitter` object is created and added to `sitterList`. The new babysitter has zero years of experience.

For example, assume that `smartSitter` has been declared as a `BabysitterService` object and `ArrayList sitterList` contains the following babysitters.

"Joe"	"Holly"	"Jayna"
3	1	2

Assume that `newNames` is a `String` array containing `"Cora"` and `"Joe"`. Then the `ArrayList sitterList` after the method call `smartSitter.addNewSitters(newNames)` should contain the following babysitters.

"Joe"	"Holly"	"Jayna"	"Cora"
3	1	2	0

Complete method `addNewSitters`.

```
/** Adds new babysitters to the list of babysitters, avoiding duplicates,
```

```
as described in part (a) */
public void addNewSitters(String[] names)
```

(b) Write the `BabysitterService` method `affordableSitters`. The method returns an `ArrayList` of all babysitters for whom the babysitting charge would be an amount less than or equal to the specified budget, for the given number of children and the given number of hours. The amount charged for babysitting services is calculated according to the following rules.

- A babysitter with no years of experience charges $\$2$ per child per hour.
- For every year of experience that a babysitter has up to four years of experience, the babysitter charges an additional $\$1$ per child per hour. There is a maximum total charge of $\$6$ per child per hour.

The following table provides some examples of the calculation of babysitting charges.

Years of Experience	Number of Children	Number of Hours	Total Charge
0	3	2	$\$2 \times 3$ children $\times$ 2 hours $= \$12$
3	3	2	$\$5 \times 3$ children $\times$ 2 hours $= \$30$
5	3	2	$\$6 \times 3$ children $\times$ 2 hours $= \$36$

Complete method `affordableSitters`.

```
/** Returns a list of babysitters for whom the babysitting charge for the
given number of children
  *   and hours is within the given budget, as described in part (b)
 */
public ArrayList<Babysitter> affordableSitters(double budget,
     int numChildren, int hours)
```

(c) A programmer would like to add a method called `getHighDemandSitters`, which returns a list of babysitters who are in high demand.

Write a description of how you would change the `Babysitter` and `BabysitterService` classes in order to support this modification.

Make sure to include the following in your response.

- Write the method header for the `getHighDemandSitters` method.
- Identify any new or modified variables, constructors, or methods aside from the `getHighDemandSitters` method. **Do not write the program code for this change.**
- Describe, for each new or revised variable, constructor, or method, how it would change or be

implemented, including visibility and type. You do not need to describe the implementation of the `getHighDemandSitters` method. **Do not write the program code for this change.**

8. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

1. A bicycle shop maintains information about bicycles it has in stock. A bicycle is represented by the following `Bicycle` class.

```
public class Bicycle
{
 /** Returns the type of bicycle (for example, "road" or "mountain") */
 public String getType()
 {  /* implementation not shown */  }

 /** Returns true if the bicycle is assembled and returns false
 otherwise */
 public boolean isAssembled()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

Information about the inventory of bicycles at the shop is stored in a `BicycleInventory` class, which contains a list of the bicycles in stock at the shop. You will write two methods of the `BicycleInventory` class.

```
public class BicycleInventory
{
 /** A list of the bicycles the shop has in stock
  *    Guaranteed to contain at least one Bicycle and all entries are
 non-null
  */
 private ArrayList<Bicycle> bicycleList;

 /** Returns an array of bicycles, as described in part (a)
  *  Precondition: n > 0
  */
```

```
  public Bicycle[] getChoices(int n, String type, boolean assembled)
  {  /* to be implemented in part (a) */  }

  /** Returns a randomly selected bicycle, as described in part (b)
   *  Precondition: n > 0
   */
  public Bicycle chooseOne(int n, String type, boolean assembled)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are
  not shown.
}
```

(a) Write the `BicycleInventory` method `getChoices`. The method returns an array of at most `n` bicycles from those in stock where the bicycle type is equal to the `type` parameter and the bicycle assembly status is equal to the `assembled` parameter. If fewer than `n` bicycles meet the criteria, the unused array elements should be `null`.

For example, assume that `classicBike` has been declared as a `BicycleInventory` object and `bike1`, `bike2`, `bike3`, `bike4`, and `bike5` are properly declared and initialized `Bicycle` objects. The following table represents the contents of `bicycleList`.

`Bicycle` **Object**	**Bicycle Type**	**Bicycle Is Assembled**
bike1	"road"	true
bike2	"mountain"	false
bike3	"road"	false
bike4	"road"	true
bike5	"mountain"	false

The following table shows the results of several calls to the `getChoices` method.

Method Call	Return Value
`classicBike.getChoices(2, "mountain", false)`	`{bike2, bike5}` or `{bike5, bike2}`
`classicBike.getChoices(3, "road", false)`	`{bike3, null, null}`
`classicBike.getChoices(1, "road", true)`	`{bike1}` or `{bike4}`

Note that the bicycles can be in any order in the array, but the `null` values must always be at the end of the array.

Write method `getChoices`.

```
/** Returns an array of bicycles, as described in part (a)
 *  Precondition: n > 0
 */
public Bicycle[] getChoices(int n, String type, boolean assembled)
```

(b) Write the `BicycleInventory` method `chooseOne`. The method randomly selects one bicycle from an array of at most `n` bicycles for which the bicycle type is equal to the `type` parameter and the bicycle assembly status is equal to the `assembled` parameter.

The method returns the randomly selected bicycle if the array contains at least one bicycle matching the criteria or returns `null` if there are no matching bicycles. Each matching bicycle in the array must have an equal chance of being selected.

Assume that `getChoices` works as specified, regardless of what you wrote for part (a). You must use `getChoices` appropriately to receive full credit.

Write method `chooseOne`.

```
/** Returns a randomly selected bicycle, as described in part (b)
 *  Precondition: n > 0
 */
public Bicycle chooseOne(int n, String type, boolean assembled)
```

(c) A programmer would like to add a method called `getBicyclesByWheelSize`, which returns an array of

all bicycles in the inventory with a specified wheel size.

Write a description of how you would change the `Bicycle` and `BicycleInventory` classes in order to support this modification.

Make sure to include the following in your response.

The method header for the `getBicyclesByWheelSize` method
Identify any new or modified variables, constructors, or methods aside from the `getBicyclesByWheelSize` method. **Do not write the program code for this change.**
Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the `getBicyclesByWheelSize` method. **Do not write the program code for this change.**

9. Consider the following methods, which appear in the same class.

```
public void slope(int x1, int y1, int x2, int y2)
{
     int xChange = x2 - x1;
     int yChange = y2 - y1;
     printFraction(yChange, xChange);
}

public void printFraction(int numerator, int denominator)
{
     System.out.print(numerator + "/" + denominator);
}
```

Assume that the method call `slope(1, 2, 5, 10)` appears in a method in the same class. What is printed as a result of the method call?

(A) `8/4`

(B) `5/1`

(C) `4/8`

(D) `2/1`

(E) `1/5`

10. Consider the following method.

```
public double myMethod(int a, boolean b)
{  /* implementation not shown */  }
```

Which of the following lines of code, if located in a method in the same class as `myMethod`, will compile without error?

(A) `int result = myMethod(2, false);`

(B) `int result = myMethod(2.5, true);`

(C) `double result = myMethod(0, false);`

(D) `double result = myMethod(true, 10);`

(E) `double result = myMethod(2.5, true);`

11. Consider the following class declaration.

```
public class Sample
{
     private int a;
     private double b;

     public Sample(int x, double y)
     {
         a = x;
         b = y;
     }

     // No other constructors
}
```

The following method appears in a class other than `Sample`.

```
public static void test()
{
     Sample object = new /* missing constructor call */ ;
}
```

Which of the following could be used to replace `/*` *missing constructor call* `*/` so that the method will compile without error?

(A) `Sample()`

(B) `Sample(int x = 10, double y = 6.2)`

(C) `Sample(int x, double y)`

(D) `Sample(10, 6.2)`

(E) `Sample(6.2, 10)`

12. Consider the following class declaration

```
public class SomeClass
{
 private int num;

 public SomeClass(int n)
 {
  num = n;
 }

 public void increment(int more)
 {
  num = num + more;
 }

 public int getNum()
 {
  return num;
 }
}
```

The following code segment appears in another class.

```
SomeClass one = new SomeClass(100);
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
SomeClass two = new SomeClass(100);
SomeClass three = one;

one.increment(200);

System.out.println(one.getNum() + " " + two.getNum() + " " +
                   three.getNum());
```

What is printed as a result of executing the code segment?

(A) 100 100 100
(B) 300 100 100
(C) 300 100 300
(D) 300 300 100
(E) 300 300 300

13. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the appendices have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

A mountain climbing club maintains a record of the climbs that its members have made. Information about a climb includes the name of the mountain peak and the amount of time it took to reach the top. The information is contained in the ClimbInfo class as declared below.

```
public class ClimbInfo
{
  /** Creates a ClimbInfo object with name peakName and time climbTime.
   *  @param peakName the name of the mountain peak
   *  @param climbTime the number of minutes taken to complete the climb
   */
  public ClimbInfo(String peakName, int climbTime)
  {  /* implementation not shown */  }

  /** @return the name of the mountain peak
   */
  public String getName()
  {  /* implementation not shown */  }

  /** @return the number of minutes taken to complete the climb
   */
  public int getTime()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

The ClimbingClub class maintains a list of the climbs made by members of the club. The declaration of the ClimbingClub class is shown below. You will write two different implementations of the addClimb method. You will also answer two questions about an implementation of the distinctPeakNames method.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class ClimbingClub
{
  /** The list of climbs completed by members of the club.
   *  Guaranteed not to be null. Contains only non-null references.
   */
  private List<ClimbInfo> climbList;

  /** Creates a new ClimbingClub object. */
  public ClimbingClub()
  {  climbList = new ArrayList<ClimbInfo>();  }

  /** Adds a new climb with name peakName and time climbTime to the list of climbs.
   *  @param peakName the name of the mountain peak climbed
   *  @param climbTime the number of minutes taken to complete the climb
   */
  public void addClimb(String peakName, int climbTime)
  {  /* to be implemented in part (a) with ClimbInfo objects in the order they were added  */
     /* to be implemented in part (b) with ClimbInfo objects in alphabetical order by name */
  }

  /** @return the number of distinct names in the list of climbs */
  public int distinctPeakNames()
  {  /* implementation shown in part (c) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write an implementation of the ClimbingClub method addClimb that stores the ClimbInfo objects in the order they were added. This implementation of addClimb should create a new ClimbInfo object with the given name and time. It appends a reference to that object to the end of climbList. For example, consider the following code segment.

   - ClimbingClub hikerClub = new ClimbingClub();
   - hikerClub.addClimb("Monadnock", 274);
   - hikerClub.addClimb("Whiteface", 301);
   - hikerClub.addClimb("Algonquin", 225);
   - hikerClub.addClimb("Monadnock", 344);

   When the code segment has completed executing, the instance variable climbList would contain the following entries

Peak Name	"Monadnock"	"Whiteface"	"Algonquin"	"Monadnock"
Climb Time	274	301	225	344

Information repeated from the beginning of the question

```
public class ClimbInfo

public ClimbInfo(String peakName, int climbTime)
public String getName()
public int getTime()


public class ClimbingClub

private List<ClimbInfo> climbList
public void addClimb(String peakName, int climbTime)
public int distinctPeakNames()
```

Complete method addClimb below.

```
/** Adds a new climb with name peakName and time climbTime to the list of climbs.
 *  @param peakName the name of the mountain peak climbed
 *  @param climbTime the number of minutes taken to complete the climb
 *  Postcondition: The new entry is at the end of climbList;
 *                 The order of the remaining entries is unchanged.
 */
public void addClimb(String peakName, int climbTime)
```

b. Write an implementation of the ClimbingClub method addClimb that stores the elements of climbList in alphabetical order by name (as determined by the compareTo method of the String class). This implementation of addClimb should create a new ClimbInfo object with the given name and time and then insert the object into the appropriate position in climbList. Entries that have the same name will be grouped together and can appear in any order within the group. For example, consider the following code segment.

   - ClimbingClub hikerClub = new ClimbingClub();
   - hikerClub.addClimb("Monadnock", 274);
   - hikerClub.addClimb("Whiteface", 301);
   - hikerClub.addClimb("Algonquin", 225);
   - hikerClub.addClimb("Monadnock", 344);

   When the code segment has completed execution, the instance variable climbList would contain the following entries in either of the orders shown below.

Peak Name	"Algonquin"	"Monadnock"	"Monadnock"	"Whiteface"
Climb Time	225	344	274	301

OR

Peak Name	"Algonquin"	"Monadnock"	"Monadnock"	"Whiteface"
Climb Time	225	274	344	301

You may assume that climbList is in alphabetical order by name when the method is called. When the method has completed execution, climbList should still be in alphabetical order by name.

Information repeated from the beginning of the question

```
public class ClimbInfo

public ClimbInfo(String peakName, int climbTime)
public String getName()
public int getTime()


public class ClimbingClub

private List<ClimbInfo> climbList
public void addClimb(String peakName, int climbTime)
public int distinctPeakNames()
```

Complete method addClimb below.

```
/** Adds a new climb with name peakName and time climbTime to the list of climbs.
 *  Alphabetical order is determined by the compareTo method of the String class.
 *  @param peakName the name of the mountain peak climbed
 *  @param climbTime the number of minutes taken to complete the climb
 *  Precondition: entries in climbList are in alphabetical order by name.
 *  Postcondition: entries in climbList are in alphabetical order by name.
 */
public void addClimb(String peakName, int climbTime)
```

c. The ClimbingClub method distinctPeakNames is intended to return the number of different names in climbList. For example, after the following code segment has completed execution, the value of the variable numNames would be 3.

- ClimbingClub hikerClub = new ClimbingClub();
- hikerClub.addClimb("Monadnock", 274);
- hikerClub.addClimb("Whiteface", 301);

- hikerClub.addClimb("Algonquin", 225);
- hikerClub.addClimb("Monadnock", 344);
- int numNames = hikerClub.distinctPeakNames();

Consider the following implementation of method distinctPeakNames.

```
/** @return the number of distinct names in the list of climbs */
public int distinctPeakNames()
{
  if (climbList.size() == 0)
  {
    return 0;
  }

  ClimbInfo currInfo = climbList.get(0);
  String prevName = currInfo.getName();
  String currName = null;
  int numNames = 1;

  for (int k = 1; k < climbList.size(); k++)
  {
    currInfo = climbList.get(k);
    currName = currInfo.getName();
    if (prevName.compareTo(currName) != 0)
    {
       numNames++;
       prevName = currName;
    }
  }
  return numNames;
}
```

Assume that addClimb works as specified, regardless of what you wrote in parts (a) and (b).

i. Does this implementation of the distinctPeakNames method work as intended when the addClimb method stores the ClimbInfo objects in the order they were added as described in part (a)?

   Circle one of the answers below.

   YES NO

ii. Does this implementation of the distinctPeakNames method work as intended when the addClimb method stores the ClimbInfo objects in alphabetical order by name as described in part (b)?

   Circle one of the answers below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

YES NO

14. Consider the following incomplete class that stores information about a customer, which includes a name and unique ID (a positive integer). To facilitate sorting, customers are ordered alphabetically by name. If two or more customers have the same name, they are further ordered by ID number. A particular customer is "greater than" another customer if that particular customer appears later in the ordering than the other customer.

```
public class Customer
{
  // constructs a Customer with given name and ID number
  public Customer(String name, int idNum)
  {  /* implementation not shown */  }

  // returns the customer's name
  public String getName()
  {  /* implementation not shown */  }

  // returns the customer's id
  public int getID()
  {  /* implementation not shown */  }


  // returns 0 when this customer is equal to other;
  //   a positive integer when this customer is greater than other;
  //   a negative integer when this customer is less than other
  public int compareCustomer(Customer other)
  {  /* to be implemented in part (a) */  }

  // There may be fields, constructors, and methods that are not shown.
}
```

a. Write the Customer method compareCustomer, which compares this customer to a given customer, other. Customers are ordered alphabetically by name, using the compareTo method of the String class. If the names of the two customers are the same, then the customers are ordered by ID number. Method compareCustomer should return a positive integer if this customer is greater than other, a negative integer if this customer is less than other, and 0 if they are the same.

   For example, suppose we have the following Customer objects.

   - Customer c1 = new Customer("Smith", 1001);
   - Customer c2 = new Customer("Anderson", 1002);
   - Customer c3 = new Customer("Smith", 1003);

   The following table shows the result of several calls to compareCustomer.

Method Call	Result
`c1.compareCustomer(c1)`	0
`c1.compareCustomer(c2)`	a positive integer
`c1.compareCustomer(c3)`	a negative integer

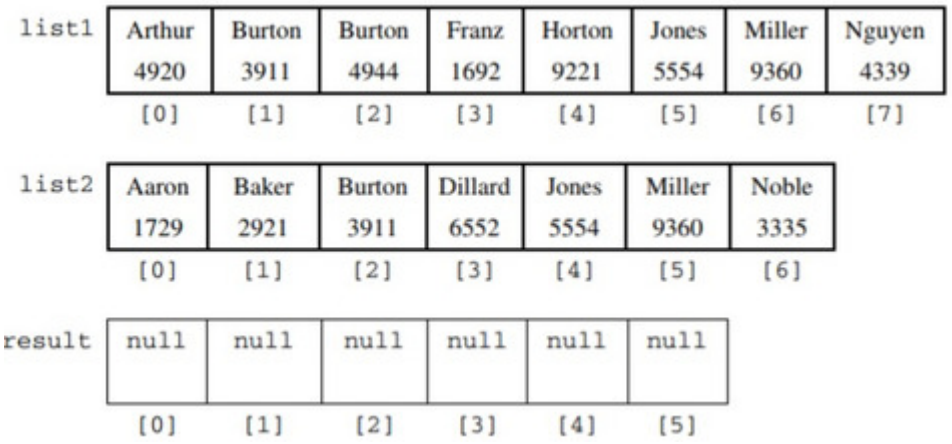
   Complete method compareCustomer below.

```
// returns 0 when this customer is equal to other;
//   a positive integer when this customer is greater than other;
//   a negative integer when this customer is less than other
public int compareCustomer(Customer other)
```

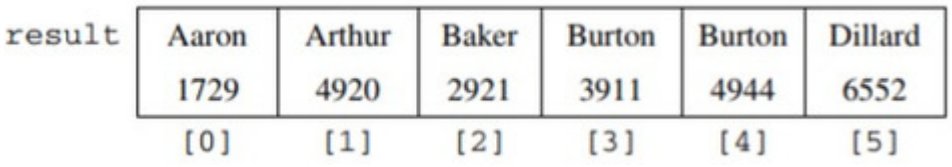
b. A company maintains customer lists where each list is a sorted array of customers stored in ascending order by customer. A customer may appear in more than one list, but will not appear more than once in the same list.

Write method prefixMerge, which takes three array parameters. The first two arrays, list1 and list2, represent existing customer lists. It is possible that some customers are in both arrays. The third array, result, has been instantiated to a length that is no longer than either of the other two arrays and initially contains null values. Method prefixMerge uses an algorithm similar to the merge step of a Mergesort to fill the array result. Customers are copied into result from the beginning of list1 and list2, merging them in ascending order until all positions of result have been filled. Customers who appear in both list1 and list2 will appear at most once in result.

For example, assume that three arrays have been initialized as shown below.



In this example, the array result must contain the following values after the call prefixMerge(list1, list2, result).



In writing prefixMerge, you may assume that compareCustomer works as specified, regardless of what you wrote in part (a). Solutions that create any additional data structures holding multiple objects (e.g., arrays, ArrayLists, etc.) will not receive full credit.

15. Consider a system for processing student test scores. The following class will be used as part of this system and contains a student's name and the student's answers for a multiple-choice test. The answers are represented as strings of length one with an omitted answer being represented by a string containing a single question mark ("?"). These answers are stored in an ArrayList in which the position of the answer corresponds to the question number on the test (question numbers start at 0). A student's score on the test is computed by comparing the student's answers with the corresponding answers in the answer key for the test. One point is awarded for each correct answer and ¼ of a point is deducted for each incorrect answer. Omitted answers (indicated by "?") do not change the student's score.

```
public class StudentAnswerSheet
{
  private ArrayList<String> answers;   // the list of the student's answers

  /** @param key the list of correct answers, represented as strings of length one
   *          Precondition: key.size() is equal to the number of answers in this answer sheet
   *  @return this student's test score
   */
  public double getScore(ArrayList<String> key)
  {  /* to be implemented in part (a) */  }

  /** @return the name of the student
   */
  public String getName()
  {  /* implementation not shown */  }

  // There may be fields, constructors, and methods that are not shown.
}
```

The following table shows an example of an answer key, a student's answers, and the corresponding point values that would be awarded for the student's answers. In this example, there are six correct answers, three incorrect answers, and one omitted answer. The student's score is ((6 * 1) - (3 * 0.25)) = 5.25 .

Question number	0	1	2	3	4	5	6	7	8	9
key	"A"	"C"	"D"	"E"	"B"	"C"	"E"	"B"	"B"	"C"
answers	"A"	"B"	"D"	"E"	"A"	"C"	"?"	"B"	"D"	"C"
Points awarded	1	−0.25	1	1	−0.25	1	0	1	−0.25	1

a. Write the StudentAnswerSheet method getScore. The parameter passed to method getScore is an ArrayList of strings representing the correct answer key for the test being scored. The method computes and returns a double that represents the score for the student's test answers when compared with the answer key. One point is awarded for each correct answer and ¼ of a point is deducted for each incorrect answer. Omitted answers (indicated by "?") do not change the student's score.

Complete method getScore below.

```
/** @param key the list of correct answers, represented as strings of length one
 *          Precondition: key.size() is equal to the number of answers in this answer sheet
 *  @return this student's test score
 */
public double getScore(ArrayList<String> key)
```

b. Consider the following class that represents the test results of a group of students that took a multiple-choice test.

```
public class TestResults
{
  private ArrayList<StudentAnswerSheet> sheets;

  /** Precondition: sheets.size() > 0;
   *                all answer sheets in sheets have the same number of answers
   *  @param key the list of correct answers represented as strings of length one
   *           Precondition: key.size() is equal to the number of answers
   *                         in each of the answer sheets in sheets
   *  @return the name of the student with the highest score
   */
  public String highestScoringStudent(ArrayList<String> key)
  {  /* to be implemented in part (b) */  }

  // There may be fields, constructors, and methods that are not shown.
}
```

Write the TestResults method highestScoringStudent, which returns the name of the student who received the highest score on the test represented by the parameter key. If there is more than one student with the highest score, the name of any one of these highest-scoring students may be returned. You may assume that the size of each answer sheet represented in the ArrayList sheets is equal to the size of the ArrayList key.

In writing highestScoringStudent, assume that getScore works as specified, regardless of what you wrote in part (a).

Complete method highestScoringStudent below

```
/** Precondition: sheets.size() > 0;
 *                all answer sheets in sheets have the same number of answers
 *  @param key the list of correct answers represented as strings of length one
 *           Precondition: key.size() is equal to the number of answers
 *                         in each of the answer sheets in sheets
 *  @return the name of the student with the highest score
 */
public String highestScoringStudent(ArrayList<String> key)
```

16. Consider the following class declaration.

```
public class Person
{
  private String myName;
  private int myYearOfBirth;

  public Person(String name, int yearOfBirth)
  {
    myName = name;
    myYearOfBirth = yearOfBirth;
  }

  public String getName()
  {  return myName;  }

  public void setName(String name)
  {  myName = name;  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

Assume that the following declaration has been made.
Person student = new Person ("Thomas", 1995);
Which of the following statements is the most appropriate for changing the name of student from "Thomas" to "Tom" ?

(A) student = new Person ("Tom", 1995);

(B) student.myName = "Tom";

(C) student.getName ("Tom");

(D) student.setName ("Tom");

(E) Person.setName ("Tom");

17. Consider the following class declaration.

```
public class Student
{
  private String myName;
  private int myAge;

  public Student()
  {  /* implementation not shown */  }

  public Student(String name, int age)
  {  /* implementation not shown */  }

  // No other constructors
}
```

Which of the following declarations will compile without error?

I. Student a = new Student();

II. Student b = new Student("Juan", 15);

III. Student c = new Student("Juan", "15");

(A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

Directions: Select the choice that best fits each statement. The following question(s) refer to the following incomplete class declaration.

```
public class TimeRecord
{
  private int hours;
  private int minutes; // 0 ≤ minutes < 60
  /** Constructs a TimeRecord object.
   *  @param h the number of hours
   *         Precondition: h ≥ 0
   *  @param m the number of minutes
   *         Precondition: 0 ≤ m < 60
   */
  public TimeRecord(int h, int m)
  {
    hours = h;
    minutes = m;
  }

  /** @return the number of hours
   */
  public int getHours()
  { /* implementation not shown */ }

  /** @return the number of minutes
   *  Postcondition: 0 ≤ minutes < 60
   */
  public int getMinutes()
  { /* implementation not shown */ }

  /** Adds h hours and m minutes to this TimeRecord.
   *  @param h the number of hours
   *         Precondition: h ≥ 0
   *  @param m the number of minutes
   *         Precondition: m ≥ 0
   */
  public void advance(int h, int m)
  {
    hours = hours + h;
    minutes = minutes + m;
    /* missing code */
  }
  // Other methods not shown
}
```

18. Consider the following declaration that appears in a class other than TimeRecord.

TimeRecord [ ] timeCards = new TimeRecord [100] ;

Assume that timeCards has been initialized with TimeRecord objects. Consider the following code segment that is intended to compute the total of all the times stored in timeCards.

```
TimeRecord total = new TimeRecord(0,0);
for (int k = 0; k < timeCards.length; k++)
{
  /* missing expression */ ;
}
```

Which of the following can be used to replace / * *missing expression* * / so that the code segment will work as intended?

(A) timeCards [k] .advance ( )

(B) total += timeCards [k] .advance ( )

(C)
```
total.advance(timeCards[k].hours,
              timeCards[k].minutes)
```

(D)
```
total.advance(timeCards[k].getHours(),
              timeCards[k].getMinutes())
```

(E)
```
timeCards[k].advance(timeCards[k].getHours(),
                     timeCards[k].getMinutes())
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

The following questions refer to the code in the GridWorld case study. A copy of the code is provided below.

Consider the design of a Grasshopper class that extends Bug. When asked to move, a Grasshopper moves to a randomly chosen empty adjacent location that is within the grid. If there is no empty adjacent location that is within the grid, the Grasshopper does not move, but turns 45 degrees to the right without changing its location.

Appendix B — Testable API

info.gridworld.grid.Location class (implements Comparable)

public Location(int r, int c)

constructs a location with given row and column coordinates

public int getRow()

returns the row of this location

public int getCol()

returns the column of this location

public Location getAdjacentLocation(int direction)

returns the adjacent location in the direction that is closest to direction

public int getDirectionToward(Location target)

returns the closest compass direction from this location toward target

public boolean equals(Object other)

returns true if other is a Location with the same row and column as this location; false otherwise

public int hashCode()

returns a hash code for this location

public int compareTo(Object other)

returns a negative integer if this location is less than other, zero if the two locations are equal, or a positive integer if this location is greater than other. Locations are ordered in row-major order.

Precondition: other is a Location object.

public String toString()

returns a string with the row and column of this location, in the format (row, col)

Compass directions:

public static final int NORTH = 0;

public static final int EAST = 90;

public static final int SOUTH = 180;

public static final int WEST = 270;

public static final int NORTHEAST = 45;

public static final int SOUTHEAST = 135;

public static final int SOUTHWEST = 225;

public static final int NORTHWEST = 315;

Turn angles:

public static final int LEFT = -90;

public static final int RIGHT = 90;

public static final int HALF_LEFT = -45;

public static final int HALF_RIGHT = 45;

public static final int FULL_CIRCLE = 360;

public static final int HALF_CIRCLE = 180;

public static final int AHEAD = 0;

info.gridworld.grid.Grid<E> interface

int getNumRows()

returns the number of rows, or -1 if this grid is unbounded

int getNumCols()

returns the number of columns, or -1 if this grid is unbounded

boolean isValid(Location loc)

returns true if loc is valid in this grid, false otherwise

Precondition: loc is not null

E put(Location loc, E obj)

puts obj at location loc in this grid and returns the object previously at that location (or null if the

location was previously unoccupied).

Precondition: (1) loc is valid in this grid (2) obj is not null

E remove(Location loc)

removes the object at location loc from this grid and returns the object that was removed (or null if the location is unoccupied)

Precondition: loc is valid in this grid

E get(Location loc)

returns the object at location loc (or null if the location is unoccupied)

Precondition: loc is valid in this grid

ArrayList<Location> getOccupiedLocations()

returns an array list of all occupied locations in this grid

ArrayList<Location> getValidAdjacentLocations(Location loc)

returns an array list of the valid locations adjacent to loc in this grid

Precondition: loc is valid in this grid

ArrayList<Location> getEmptyAdjacentLocations(Location loc)

returns an array list of the valid empty locations adjacent to loc in this grid

Precondition: loc is valid in this grid

ArrayList<Location> getOccupiedAdjacentLocations(Location loc)

returns an array list of the valid occupied locations adjacent to loc in this grid

Precondition: loc is valid in this grid

ArrayList<E> getNeighbors(Location loc)

returns an array list of the objects in the occupied locations adjacent to loc in this grid

Precondition: loc is valid in this grid

info.gridworld.actor.Actor class

public Actor()

constructs a blue actor that is facing north

public Color getColor()

returns the color of this actor

public void setColor(Color newColor)

sets the color of this actor to newColor

public int getDirection()

returns the direction of this actor, an angle between 0 and 359 degrees

public void setDirection(int newDirection)

sets the direction of this actor to the angle between 0 and 359 degrees that is equivalent to newDirection

public Grid<Actor> getGrid()

returns the grid of this actor, or null if this actor is not contained in a grid

public Location getLocation()

returns the location of this actor, or null if this actor is not contained in a grid

public void putSelfInGrid(Grid<Actor> gr, Location loc)

puts this actor into location loc of grid gr. If there is another actor at loc, it is removed.

Precondition: (1) This actor is not contained in a grid (2) loc is valid in gr

public void removeSelfFromGrid()

removes this actor from its grid.

Precondition: this actor is contained in a grid

public void moveTo(Location newLocation)

moves this actor to newLocation. If there is another actor at newLocation, it is removed.

Precondition: (1) This actor is contained in a grid (2) newLocation is valid in the grid of this actor

public void act()

reverses the direction of this actor. Override this method in subclasses of Actor to define types of actors with different behavior

public String toString()

returns a string with the location, direction, and color of this actor

info.gridworld.actor.Rock class (extends Actor)

public Rock()

constructs a black rock

public Rock(Color rockColor)

constructs a rock with color rockColor

public void act()

overrides the act method in the Actor class to do nothing

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

info.gridworld.actor.Flower class (extends Actor)

public Flower()

constructs a pink flower

public Flower(Color initialColor)

constructs a flower with color initialColor

public void act()

causes the color of this flower to darken

19. Consider the following method that is intended to return an ArrayList of all the locations in grd that contain actors facing in direction dir.

```
public ArrayList<Location> findLocsFacingDir(int dir, Grid<Actor> grd)

{

ArrayList<Location> desiredLocs = new ArrayList<Location>();

for (Location loc : grd.getOccupiedLocations())

{

if ( /* expression */ == dir )

desiredLocs.add(loc);

}

return desiredLocs;

}
```

Which of the following can be used to replace /* *expression* */ so that findLocsFacingDir will work as intended?

(A) loc.getDirection()
(B) getDirection(loc)
(C) ((Actor) loc).getDirection()
(D) grd(loc).getDirection()
(E) grd.get(loc).getDirection()

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

The following questions refer to the code from the GridWorld case study. A copy of the code is provided below.

Appendix B — Testable API

info.gridworld.grid.Location class (implements Comparable)

public Location(int r, int c)

constructs a location with given row and column coordinates

public int getRow()

returns the row of this location

public int getCol()

returns the column of this location

public Location getAdjacentLocation(int direction)

returns the adjacent location in the direction that is closest to direction

public int getDirectionToward(Location target)

returns the closest compass direction from this location toward target

public boolean equals(Object other)

returns true if other is a Location with the same row and column as this location; false otherwise

public int hashCode()

returns a hash code for this location

public int compareTo(Object other)

returns a negative integer if this location is less than other, zero if the two locations are equal, or a positive integer if this location is greater than other. Locations are ordered in row-major order.

Precondition: other is a Location object.

public String toString()

returns a string with the row and column of this location, in the format (row, col)

Compass directions:

public static final int NORTH = 0;

public static final int EAST = 90;

public static final int SOUTH = 180;

public static final int WEST = 270;

public static final int NORTHEAST = 45;

public static final int SOUTHEAST = 135;

public static final int SOUTHWEST = 225;

public static final int NORTHWEST = 315;

Turn angles:

public static final int LEFT = -90;

public static final int RIGHT = 90;

public static final int HALF_LEFT = -45;

public static final int HALF_RIGHT = 45;

public static final int FULL_CIRCLE = 360;

public static final int HALF_CIRCLE = 180;

public static final int AHEAD = 0;

info.gridworld.grid.Grid<E> interface

int getNumRows()

returns the number of rows, or -1 if this grid is unbounded

int getNumCols()

returns the number of columns, or -1 if this grid is unbounded

boolean isValid(Location loc)

returns true if loc is valid in this grid, false otherwise

Precondition: loc is not null

E put(Location loc, E obj)

puts obj at location loc in this grid and returns the object previously at that location (or null if the location was previously unoccupied).

Precondition: (1) loc is valid in this grid (2) obj is not null

E remove(Location loc)

removes the object at location loc from this grid and returns the object that was removed (or null if the location is unoccupied)

Precondition: loc is valid in this grid

E get(Location loc)

returns the object at location loc (or null if the location is unoccupied)

Precondition: loc is valid in this grid

ArrayList<Location> getOccupiedLocations()

returns an array list of all occupied locations in this grid

ArrayList<Location> getValidAdjacentLocations(Location loc)

returns an array list of the valid locations adjacent to loc in this grid

Precondition: loc is valid in this grid

ArrayList<Location> getEmptyAdjacentLocations(Location loc)

returns an array list of the valid empty locations adjacent to loc in this grid

Precondition: loc is valid in this grid

ArrayList<Location> getOccupiedAdjacentLocations(Location loc)

returns an array list of the valid occupied locations adjacent to loc in this grid

Precondition: loc is valid in this grid

ArrayList<E> getNeighbors(Location loc)

returns an array list of the objects in the occupied locations adjacent to loc in this grid

Precondition: loc is valid in this grid

info.gridworld.actor.Actor class

public Actor()

constructs a blue actor that is facing north

public Color getColor()

returns the color of this actor

public void setColor(Color newColor)

sets the color of this actor to newColor

public int getDirection()

returns the direction of this actor, an angle between 0 and 359 degrees

public void setDirection(int newDirection)

sets the direction of this actor to the angle between 0 and 359 degrees that is equivalent to newDirection

public Grid<Actor> getGrid()

returns the grid of this actor, or null if this actor is not contained in a grid

public Location getLocation()

returns the location of this actor, or null if this actor is not contained in a grid

public void putSelfInGrid(Grid<Actor> gr, Location loc)

puts this actor into location loc of grid gr. If there is another actor at loc, it is removed.

Precondition: (1) This actor is not contained in a grid (2) loc is valid in gr

public void removeSelfFromGrid()

removes this actor from its grid.

Precondition: this actor is contained in a grid

public void moveTo(Location newLocation)

moves this actor to newLocation. If there is another actor at newLocation, it is removed.

Precondition: (1) This actor is contained in a grid (2) newLocation is valid in the grid of this actor

public void act()

reverses the direction of this actor. Override this method in subclasses of Actor to define types of actors with different behavior

public String toString()

returns a string with the location, direction, and color of this actor

info.gridworld.actor.Rock class (extends Actor)

public Rock()

constructs a black rock

public Rock(Color rockColor)

constructs a rock with color rockColor

public void act()

overrides the act method in the Actor class to do nothing

info.gridworld.actor.Flower class (extends Actor)

public Flower()

constructs a pink flower

public Flower(Color initialColor)

constructs a flower with color initialColor

public void act()

causes the color of this flower to darken

20. Consider the following method that is intended to move the parameter anActor to a different grid that is referred to by the parameter newGrid. The location of anActor in newGrid should be the same as the location that anActor had occupied in its original grid.

```
/** Moves anActor to newGrid in the same location it occupied in its original grid.
* @param anActor the actor to be moved
* @param newGrid the grid in which the actor is to be placed
*/
public void moveActorToNewGrid(Actor anActor, Grid<Actor> newGrid)
{
Grid<Actor> oldGrid = anActor.getGrid();
Location loc = anActor.getLocation();
/* missing code */
}
```

Which of the following can be used to replace /* *missing code* */ so that moveActorToNewGrid will work as intended?

(A)
```
anActor.putSelfInGrid(newGrid, loc);
anActor.removeSelfFromGrid();
```

(B)
```
oldGrid.remove(loc);
anActor.putSelfInGrid(newGrid, loc);
```

(C)
```
anActor.removeSelfFromGrid();
anActor.putSelfInGrid(newGrid, loc);
```

(D)
```
oldGrid.remove(loc);
newGrid.put(anActor, loc);
```

(E)
```
newGrid.put(anActor, loc);
oldGrid.remove(loc);
```

---

21. Consider the following method.

```
public int pick(boolean test, int x, int y)

{

if (test)

return x;

else

return y;

}
```

What value is returned by the following method call?

```
pick(false, pick(true, 0, 1), pick(true, 6, 7))
```

(A) 0
(B) 1
(C) 3
(D) 6
(E) 7

22. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

3. This question involves analyzing a restaurant's menu information. A restaurant stores information about menu items that customers can order in the `MenuItem` class.

```
public class MenuItem
{
 /** Returns the name, as shown on the menu, of the item */
 public String getName()
 {  /* implementation not shown */  }

 /** Returns the price of the item */
 public double getPrice()
 {  /* implementation not shown */  }

 /** Returns true if the item is an entree and returns false otherwise
 */
 public boolean isEntree()
 {  /* implementation not shown */  }

 /** Returns true if the item is a daily special menu item and returns
 false
  *  otherwise
  */
 public boolean isDailySpecial()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

The `CustomerCheck` class is used to represent an itemized check for one or more customers. You will write two methods of the `CustomerCheck` class.

```
public class CustomerCheck
```

```
{
 /** The check for a customer or group of customers
   *  Guaranteed to contain at least one MenuItem and all entries are
 non-null
  */
 private ArrayList<MenuItem> check;

 /** Returns the total of all MenuItem prices on the check */
 public double totalPrices()
 {  /* implementation not shown */  }

 /** Returns true if the restaurant's coupon offer can be applied to
 this check and
   *  returns false otherwise, as described in part (a)
  */
 public boolean couponApplies()
 {  /* to be implemented in part (a) */  }

 /** Calculates the final cost of this check, as described in part (b)
 */
 public double calculateCheck()
 {  /* to be implemented in part (b) */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

(a) The `couponApplies` method returns `true` if the restaurant's discount coupon offer can be applied to this check and returns `false` otherwise. The following rules are used to determine if the coupon offer can be applied.

- The coupon offer does not apply if any of the items on the check are daily special menu items.
- The coupon offer does not apply if the total of all `MenuItem` prices is less than 40 dollars.

A helper method, `totalPrices`, has been provided. It returns the total of all `MenuItem` prices on `check`. You must use `totalPrices` appropriately in order to receive full credit.

Complete method `couponApplies`.

```
/** Returns true if the restaurant's coupon offer can be applied to this
check and
  *  returns false otherwise, as described in part (a)
 */
public boolean couponApplies()
```

(b) The `calculateCheck` method calculates the final cost for a check. The original check is the sum of all `MenuItem` prices. The final cost is the original check decreased by a possible coupon discount and increased by a possible tip, based on the following rules.

- If the discount coupon offer can be applied, the final cost is reduced by 25 percent of the original check.
- The number of customers on the check can be determined by counting the number of entrees included on the check. If there are at least six customers in the group, the final cost is increased by a tip that is equal to 20 percent of the original check.

The following table shows examples of the final cost calculation.

Original Check	Number of Customers	Discount Applies	Final Cost Calculation	Explanation
$\$100$	8	yes	$\$100 + \$20 - \$25 = \$95$	Tip automatically added
$\$50$	5	yes	$\$50 - \$12.50 = \$37.50$	Tip not automatically added
$\$80$	6	no	$\$80 + \$16 = \$96$	Tip automatically added
$\$40$	2	no	$\$40$	Tip not automatically added

Assume that `couponApplies` works as specified, regardless of what you wrote in part (a). You must use `couponApplies` and `totalPrices` appropriately in order to receive full credit.

Complete method `calculateCheck`.

```
/** Calculates the final cost of this check, as described in part (b) */
public double calculateCheck()
```

(c) A programmer would like to add a method called `hasPremiumEntrees`, which returns `true` if at least one entree on the check is considered premium and returns `false` otherwise.

Write a description of how you would change the `MenuItem` and `CustomerCheck` classes in order to support this modification.

Make sure to include the following in your response.

- Write the method header for the `hasPremiumEntrees` method.

- Identify any new or modified variables, constructors, or methods aside from the `hasPremiumEntrees` method. **Do not write the program code for this change.**
- Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the `hasPremiumEntrees` method. **Do not write the program code for this change.**

23. The `Student` class has been defined to store and manipulate grades for an individual student. The following methods have been defined for the class.

```
/* Returns the sum of all of the student's grades */
public double sumOfGrades()
{  /* implementation not shown */  }
/* Returns the total number of grades the student has received */
public int numberOfGrades()
{  /* implementation not shown */  }
/* Returns the lowest grade the student has received */
public double lowestGrade()
{  /* implementation not shown */  }
```

Which of the following statements, if located in a method in the `Student` class, will determine the average of all of the student's grades except for the lowest grade and store the result in the `double` variable `newAverage`?

(A) `newAverage = sumOfGrades() / numberOfGrades() - 1;`

(B) `newAverage = sumOfGrades() / (numberOfGrades() - 1);`

(C) `newAverage = sumOfGrades() - lowestGrade() / (numberOfGrades() - 1);`

(D) `newAverage = (sumOfGrades() - lowestGrade()) / numberOfGrades() - 1;`

(E) `newAverage = (sumOfGrades() - lowestGrade()) / (numberOfGrades() - 1);`

24. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

A two-dimensional array of integers in which most elements are zero is called a sparse array. Because most elements have a value of zero, memory can be saved by storing only the non-zero values along with their row and column indexes. The following complete SparseArrayEntry class is used to represent non-zero elements in a sparse array. A SparseArrayEntry object cannot be modified after it has been constructed.

```
public class SparseArrayEntry
{
  /** The row index and column index for this entry in the sparse array */
  private int row;
  private int col;

  /** The value of this entry in the sparse array */
  private int value;

  /** Constructs a SparseArrayEntry object that represents a sparse array element
   *  with row index r and column index c, containing value v.
   */
  public SparseArrayEntry(int r, int c, int v)
  {
    row = r;
    col = c;
    value = v;
  }

  /** Returns the row index of this sparse array element. */
  public int getRow()
  { return row; }

  /** Returns the column index of this sparse array element. */
  public int getCol()
  { return col; }

  /** Returns the value of this sparse array element. */
  public int getValue()
  { return value; }
}
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

The SparseArray class represents a sparse array. It contains a list of SparseArrayEntry objects, each of which represents one of the non-zero elements in the array. The entries representing the non-zero elements are stored in the list in no particular order. Each non-zero element is represented by exactly one entry in the list.

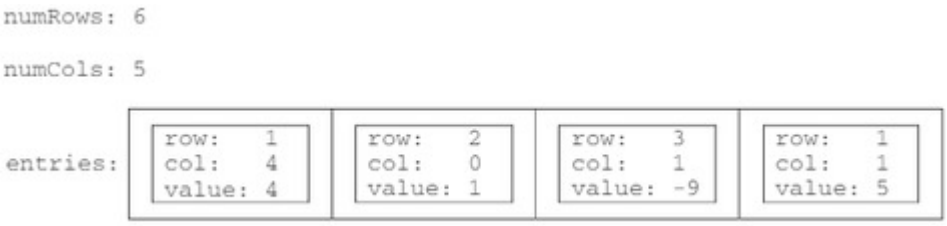
```
public class SparseArray
{
  /** The number of rows and columns in the sparse array. */
  private int numRows;
  private int numCols;

  /** The list of entries representing the non-zero elements of the sparse array. Entries are stored in the
   *  list in no particular order. Each non-zero element is represented by exactly one entry in the list.
   */
  private List<SparseArrayEntry> entries;

  /** Constructs an empty SparseArray. */
  public SparseArray()
  {  entries = new ArrayList<SparseArrayEntry>();  }

  /** Returns the number of rows in the sparse array. */
  public int getNumRows()
  {  return numRows;  }

  /** Returns the number of columns in the sparse array. */
  public int getNumCols()
  {  return numCols;  }

  /** Returns the value of the element at row index row and column index col in the sparse array.
   *  Precondition: 0 ≤ row < getNumRows()
   *                0 ≤ col < getNumCols()
   */
  public int getValueAt(int row, int col)
  {  /* to be implemented in part (a) */  }

  /** Removes the column col from the sparse array.
   *  Precondition: 0 ≤ col < getNumCols()
   */
  public void removeColumn(int col)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

The following table shows an example of a two-dimensional sparse array. Empty cells in the table indicate zero values.

	0	1	2	3	4
0					
1		5			4
2	1				
3		-9			
4					
5					

The sample array can be represented by a SparseArray object, sparse, with the following instance variable values. The items in entries are in no particular order; one possible ordering is shown below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf



(a) Write the SparseArray method getValueAt. The method returns the value of the sparse array element at a given row and column in the sparse array. If the list entries contains an entry with the specified row and column, the value associated with the entry is returned. If there is no entry in entries corresponding to the specified row and column, 0 is returned.
In the example above, the call sparse.getValueAt(3, 1) would return -9, and sparse.getValueAt(3, 3) would return 0.

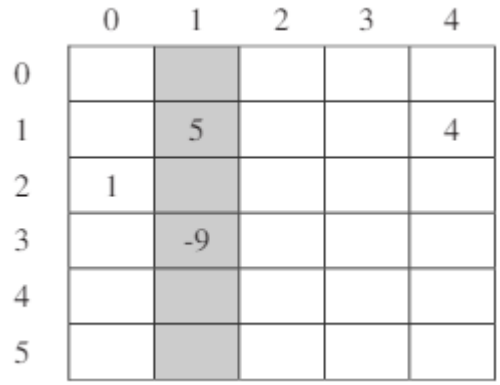
Complete method getValueAt below.

```
/** Returns the value of the element at row index row and column index col in the sparse array.
 *  Precondition: 0 ≤ row < getNumRows()
 *                0 ≤ col < getNumCols()
 */
```

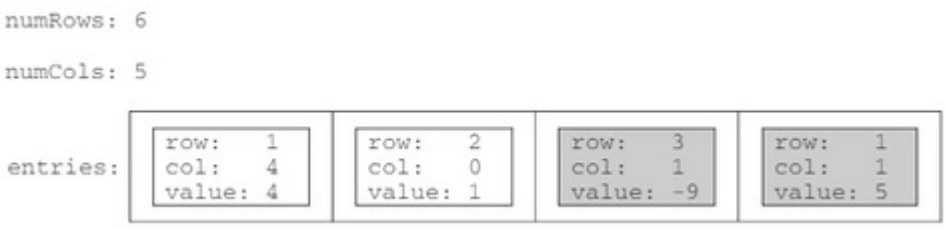
(b) Write the SparseArray method removeColumn. After removing a specified column from a sparsearray:

- All entries in the list entries with column indexes matching col are removed from the list.
- All entries in the list entries with column indexes greater than col are replaced by entries with column indexes that are decremented by one (moved one column to the left).
- The number of columns in the sparse array is adjusted to reflect the column removed.

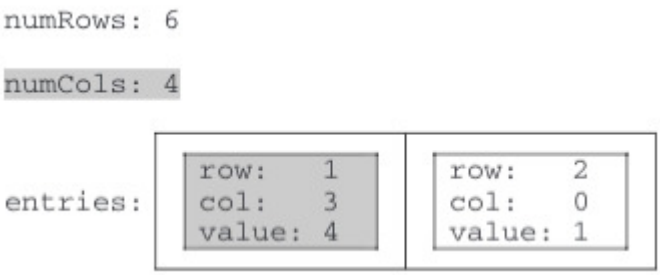
The sample object sparse from the beginning of the question is repeated for your convenience.



The shaded entries in entries, below, correspond to the shaded column above.



When sparse has the state shown above, the call sparse.removeColumn(1) could result insparse having the following values in its instance variables (since entries is in no particular order, itwould be equally valid to reverse the order of its two items). The shaded areas below show the changes.



Class information repeated from the beginning of the question

```
public class SparseArrayEntry

public SparseArrayEntry(int r, int c, int v)
public int getRow()
public int getCol()
public int getValue()

public class SparseArray

private int numRows
private int numCols
private List<SparseArrayEntry> entries
public int getNumRows()
public int getNumCols()
public int getValueAt(int row, int col)
public void removeColumn(int col)
```

Complete method `removeColumn` below.

```
/** Removes the column col from the sparse array.
 *  Precondition: 0 ≤ col < getNumCols()
 */
public void removeColumn(int col)
```

25. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

Consider the following partial declaration for a WordScrambler class. The constructor for theWordScrambler class takes an even-length array of String objects and initializes the instance variable scrambledWords.

```
public class WordScrambler
{
  private String[] scrambledWords;

  /** @param wordArr an array of String objects
   *          Precondition: wordArr.length is even
   */
  public WordScrambler(String[] wordArr)
  {
    scrambledWords = mixedWords(wordArr);
  }


  /** @param word1 a String of characters
   *  @param word2 a String of characters
   *  @return a String that contains the first half of word1 and the second half of word2
   */
  private String recombine(String word1, String word2)
  {  /* to be implemented in part (a) */  }


  /** @param words an array of String objects
   *          Precondition: words.length is even
   *  @return an array of String objects created by recombining pairs of strings in array words
   *  Postcondition: the length of the returned array is words.length
   */
  private String[] mixedWords(String[] words)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the WordScrambler method recombine. This method returns a String created from its two String

parameters as follows.

- take the first half of word1
- take the second half of word2
- concatenate the two halves and return the new string.

For example, the following table shows some results of calling recombine. Note that if a word has an odd number of letters, the second half of the word contains the extra letter.

word1	word2	recombine(word1, word2)
"apple"	"pear"	"apar"
"pear"	"apple"	"peple"

Complete method recombine below.

```
/** @param word1 a String of characters
 *  @param word2 a String of characters
 *  @return a String that contains the first half of word1 and the second half of word2
 */
private String recombine(String word1, String word2)
```

(b) Write the WordScrambler method mixedWords. This method creates and returns a new array of String objects as follows.

It takes the first pair of strings in words and combines them to produce a pair of strings to be included in the array returned by the method. If this pair of strings consists of w1 and w2, the method should include the result of calling recombine with w1 and w2 as arguments and should also include the result of calling recombine with w2 and w1 as arguments. The next two strings, if they exist, would form the next pair to be processed by this method. The method should continue until all the strings in words have been processed in this way and the new array has been filled. For example, if the array words contains the following elements:

{"apple", "pear", "this", "cat"}

then the call mixedWords(words) should return the following array.

{"apar", "peple", "that", "cis"}

In writing mixedWords, you may call recombine. Assume that recombine works as specified, regardless of what

you wrote in part (a).

Complete method mixedWords below.

```
/** @param words an array of String objects
 *          Precondition: words.length is even
 *  @return an array of String objects created by recombining pairs of strings in array words
 *  Postcondition: the length of the returned array is words.length
 */
private String[] mixedWords(String[] words)
```

26. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

```
public class TemperatureGrid
{
  /** A two-dimensional array of temperature values, initialized in the constructor (not shown)
   *  Guaranteed not to be null
   */
  private double[][] temps;

  /** Computes and returns a new temperature value for the given location.
   *  @param row a valid row index in temps
   *  @param col a valid column index in temps
   *  @return the new temperature for temps[row][col]
   *          - The new temperature for any element in the border of the array is the
   *            same as the old temperature.
   *          - Otherwise, the new temperature is the average of the four adjacent entries.
   *  Postcondition: temps is unchanged.
   */
  private double computeTemp(int row, int col)
  {  /* to be implemented in part (a) */  }


  /** Updates all values in temps and returns a boolean that indicates whether or not all the
   *  new values were within tolerance of the original values.
   *  @param tolerance a double value >= 0
   *  @return true if all updated temperatures are within tolerance of the original values;
   *          false otherwise.
   *  Postcondition: Each value in temps has been updated with a new value based on the
   *                 corresponding call to computeTemp.
   */
  public boolean updateAllTemps(double tolerance)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

A two-dimensional array of temperatures is represented in the above class.

a) Write method computeTemp, which computes and returns the new temperature for a given element of temps according to the following rules.

1. If the element is in the border of the array (in the first row or last row or first column or last column), the new temperature is the same as the old temperature.

2. Otherwise, the new temperature is the average (arithmetic mean) of the temperatures of the four adjacent values in the table (located above, below, to the left, and to the right of the element).

If temps is the table shown below, temps.length is 5 and temps[0].length is 6.

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	50.0	50.0	50.0	50.0	0.0
2	0.0	40.0	40.0	40.0	40.0	0.0
3	0.0	20.0	20.0	20.0	20.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

The following table shows the results of several calls to computeTemp.

Function Call	Result
`computeTemp(2, 3)`	`37.5` (the average of the values 50.0, 20.0, 40.0, and 40.0)
`computeTemp(1, 1)`	`47.5` (the average of the values 100.0, 40.0, 0.0, and 50.0)
`computeTemp(0, 2)`	`100.0` (the same as the old value)
`computeTemp(1, 3)`	`60.0` (the average of the values 100.0, 40.0, 50.0, and 50.0)

Complete method `computeTemp` below.

```
/** Computes and returns a new temperature value for the given location.
 *  @param row a valid row index in temps
 *  @param col a valid column index in temps
 *  @return the new temperature for temps[row][col]
 *          - The new temperature for any element in the border of the array is the
 *            same as the old temperature.
 *          - Otherwise, the new temperature is the average of the four adjacent entries.
 *  Postcondition: temps is unchanged.
 */
private double computeTemp(int row, int col)
```

(b) Write method updateAllTemps, which computes the new temperature for every element of temps.The new

values should be based on the original values, so it will be necessary to create another two-dimensional array in which to store the new values. Once all the computations are complete, the new values should replace the corresponding positions of temps. Method updateAllTemps also determines whether every new temperature is within tolerance of the corresponding old temperature (i.e., the absolute value of the difference between the old temperature and the new temperature is less than or equal to tolerance). If so, it returns true; otherwise, it returns false.

If temps contains the values shown in the first table below, then the call updateAllTemps(0.01) should update temps as shown in the second table.

temps before the call updateAllTemps(0.01)

	0	1	2	3	4	5
0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	50.0	50.0	50.0	50.0	0.0
2	0.0	40.0	40.0	40.0	40.0	0.0
3	0.0	20.0	20.0	20.0	20.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

temps after the call updateAllTemps(0.01)

0	95.5	100.0	100.0	100.0	100.0	110.3
1	0.0	47.5	60.0	60.0	47.5	0.0
2	0.0	27.5	37.5	37.5	27.5	0.0
3	0.0	15.0	20.0	20.0	15.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

In the example shown, the call updateAllTemps (0.01) should return false because there are several new temperatures that are not within the given tolerance of the corresponding old temperature. For example, the updated value in temps[2][3] is 37.5, the original value in temps[2][3] was 40.0, and the absolute value of (37.5 - 40.0) is greater than the value of tolerance (0.01).

Assume that computeTemp works as specified, regardless of what you wrote in part (a).

Complete method `updateAllTemps` below.

```
/** Updates all values in temps and returns a boolean that indicates whether or not all the
 *  new values were within tolerance of the original values.
 *  @param tolerance a double value >= 0
 *  @return true if all updated temperatures are within tolerance of the original values;
 *          false otherwise.
 *  Postcondition: Each value in temps has been updated with a new value based on the
 *                 corresponding call to computeTemp.
 */
public boolean updateAllTemps(double tolerance)
```

27. **Directions**: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN Java.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

Consider a grade-averaging scheme in which the final average of a student's scores is computed differently from the traditional average if the scores have "improved." Scores have improved if each score is greater than or equal to the previous score. The final average of the scores is computed as follows.

A student has n scores indexed from 0 to n-1. If the scores have improved, only those scores with indexes greater than or equal to n/2 are averaged. If the scores have not improved, all the scores are averaged.

The following table shows several lists of scores and how they would be averaged using the scheme described above.

Student Scores	Improved?	Final Average
50, 50, 20, 80, 53	No	(50 + 50 + 20 + 80 + 53) / 5.0 = 50.6
20, 50, 50, 53, 80	Yes	(50 + 53 + 80) / 3.0 = 61.0
20, 50, 50, 80	Yes	(50 + 80) / 2.0 = 65.0

Consider the following incomplete StudentRecord class declaration. Each StudentRecord object stores a list of that student's scores and contains methods to compute that student's final average.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class StudentRecord
{
  private int[] scores; // contains scores.length values
                        // scores.length > 1

  // constructors and other data fields not shown


  // returns the average (arithmetic mean) of the values in scores
  // whose subscripts are between first and last, inclusive
  // precondition:  0 <= first <= last < scores.length
  private double average(int first, int last)
  {  /* to be implemented in part (a) */  }


  // returns true if each successive value in scores is greater
  // than or equal to the previous value;
  // otherwise, returns false
  private boolean hasImproved()
  {  /* to be implemented in part (b) */  }


  // if the values in scores have improved, returns the average
  // of the elements in scores with indexes greater than or equal
  // to scores.length/2;
  // otherwise, returns the average of all of the values in scores
  public double finalAverage()
  {  /* to be implemented in part (c) */  }

}
```

a. Write the StudentRecord method average. This method returns the average of the values in scores given a starting and an ending index.

   Complete method average below.

```
// returns the average (arithmetic mean) of the values in scores
// whose subscripts are between first and last, inclusive
// precondition:  0 <= first <= last < scores.length
private double average(int first, int last)
```

b. Write the StudentRecord method hasImproved.

   Complete method hasImproved below

```
// returns true if each successive value in scores is greater
// than or equal to the previous value;
// otherwise, returns false
private boolean hasImproved()
```

c. Write the StudentRecord method finalAverage.

   In writing finalAverage, you must call the methods defined in parts (a) and (b). Assume that these methods work as specified, regardless of what you wrote in parts (a) and (b).

   Complete method finalAverage below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
// if the values in scores have improved, returns the average
// of the elements in scores with indexes greater than or equal
// to scores.length/2;
// otherwise, returns the average of all of the values in scores
public double finalAverage()
```

28. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

A travel agency maintains a list of information about airline flights. Flight information includes a departure time and an arrival time. You may assume that the two times occur on the same day. These times are represented by objects of the Time class.

The declaration for the Time class is shown below. It includes a method minutesUntil that returns the difference (in minutes) between the current Time object and another Time object.

```
public class Time
{
  /** @return difference, in minutes, between this time and other;
   *          difference is negative if other is earlier than this time
   */
  public int minutesUntil(Time other)
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

For example, assume that t1 and t2 are Time objects where t1 represents 1:00 P.M. and t2 represents 2:15 P.M. The call t1.minutesUntil(t2) will return 75 and the call t2.minutesUntil(t1) will return -75.

The declaration for the Flight class is shown below. It has methods to access the departure time and the arrival time of a flight. You may assume that the departure time of a flight is earlier than its arrival time.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class Flight
{
    /** @return time at which the flight departs
     */
    public Time getDepartureTime()
    { /* implementation not shown */ }

    /** @return time at which the flight arrives
     */
    public Time getArrivalTime()
    { /* implementation not shown */ }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

A trip consists of a sequence of flights and is represented by the Trip class. The Trip class contains an ArrayList of Flight objects that are stored in chronological order. You may assume that for each flight after the first flight in the list, the departure time of the flight is later than the arrival time of the preceding flight in the list. A partial declaration of the Trip class is shown below. You will write two methods for the Trip class.

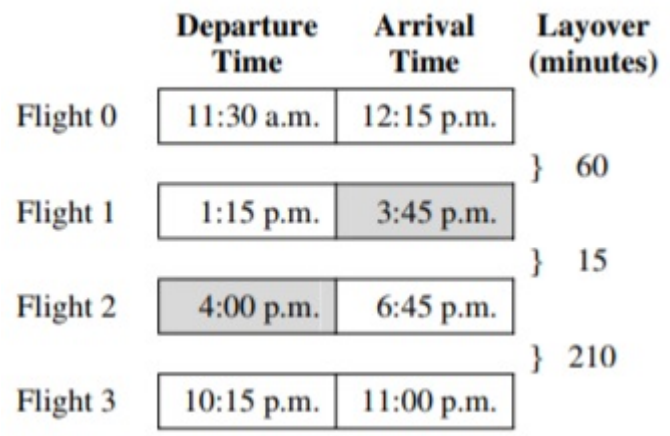
```
public class Trip
{
  private ArrayList<Flight> flights;
    // stores the flights (if any) in chronological order

  /** @return the number of minutes from the departure of the first flight to the arrival
   *          of the last flight if there are one or more flights in the trip;
   *          0, if there are no flights in the trip
   */
  public int getDuration()
  {  /* to be implemented in part (a) */  }

  /** Precondition: the departure time for each flight is later than the arrival time of its
   *                preceding flight
   *  @return the smallest number of minutes between the arrival of a flight and the departure
   *          of the flight immediately after it, if there are two or more flights in the trip;
   *          -1, if there are fewer than two flights in the trip
   */
  public int getShortestLayover()
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Complete method getDuration below.

```
/** @return the number of minutes from the departure of the first flight to the arrival
 *          of the last flight if there are one or more flights in the trip;
 *          0, if there are no flights in the trip
 */
public int getDuration()
```

b. Write the Trip method getShortestLayover. A layover is the number of minutes from the arrival of one flight in a trip to the departure of the flight immediately after it. If there are two or more flights in the trip, the method should return the shortest layover of the trip; otherwise, it should return -1.

For example, assume that the instance variable flights of a Trip object vacation contains the following flight information.



The call vacation.getShortestLayover() should return 15.

Complete method getShortestLayover below.

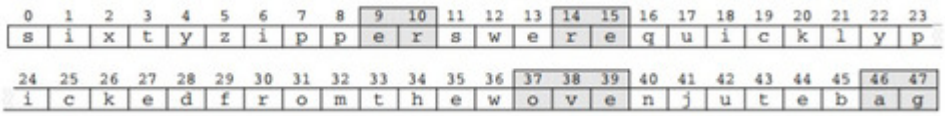
```
/** Precondition: the departure time for each flight is later than the arrival time of its
 *                preceding flight
 *  @return the smallest number of minutes between the arrival of a flight and the departure
 *          of the flight immediately after it, if there are two or more flights in the trip;
 *          -1, if there are fewer than two flights in the trip
 */
public int getShortestLayover()
```

29. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

Consider a method of encoding and decoding words that is based on a master string. This master string will contain all the letters of the alphabet, some possibly more than once. An example of a master string is "sixtyzipperswerequicklypickedfromthewovenjutebag". This string and its indexes are shown below.



An encoded string is defined by a list of string parts. A string part is defined by its starting index in the master string and its length. For example, the string "overeager" is encoded as the list of string parts [(37, 3), (14, 2), (46, 2), (9, 2)] denoting the substrings "ove", "re", "ag", and "er".

String parts will be represented by the StringPart class shown below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class StringPart
{
  /** @param start  the starting position of the substring in a master string
   *  @param length  the length of the substring in a master string
   */
  public StringPart(int start, int length)
  {  /* implementation not shown */  }

  /** @return  the starting position of the substring in a master string
   */
  public int getStart()
  {  /* implementation not shown */  }

  /** @return  the length of the substring in a master string
   */
  public int getLength()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

The class StringCoder provides methods to encode and decode words using a given master string. When encoding, there may be multiple matching string parts of the master string. The helper method findPart is provided to choose a string part within the master string that matches the beginning of a given string.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class StringCoder
{
  private String masterString;

  /** @param master the master string for the StringCoder
   *         Precondition: the master string contains all the letters of the alphabet
   */
  public StringCoder(String master)
  {  masterString = master;  }

  /** @param parts an ArrayList of string parts that are valid in the master string
   *         Precondition: parts.size() > 0
   *  @return the string obtained by concatenating the parts of the master string
   */
  public String decodeString(ArrayList<StringPart> parts)
  {  /* to be implemented in part (a) */  }

  /** @param str the string to encode using the master string
   *         Precondition: all of the characters in str appear in the master string;
   *                       str.length() > 0
   *  @return a string part in the master string that matches the beginning of str.
   *          The returned string part has length at least 1.
   */
  private StringPart findPart(String str)
  {  /* implementation not shown */  }

  /** @param word the string to be encoded
   *         Precondition: all of the characters in word appear in the master string;
   *                       word.length() > 0
   *  @return an ArrayList of string parts of the master string that can be combined
   *          to create word
   */
  public ArrayList<StringPart> encodeString(String word)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the StringCoder method decodeString. This method retrieves the substrings in the master string represented by each of the StringPart objects in parts, concatenates them in the order in which they appear in parts, and returns the result.

   Complete method decodeString below.

```
/** @param parts an ArrayList of string parts that are valid in the master string
 *         Precondition: parts.size() > 0
 *  @return the string obtained by concatenating the parts of the master string
 */
public String decodeString(ArrayList<StringPart> parts)
```

b. Write the StringCoder method encodeString. A string is encoded by determining the substrings in the master string that can be combined to generate the given string. The encoding starts with a string part that matches the beginning of the word, followed by a string part that matches the beginning of the rest of the word, and so on. The string parts are returned in an array list in the order in which they appear in word.

   The helper method findPart must be used to choose matching string parts in the master string.

   Complete method encodeString below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
/** @param word the string to be encoded
 *         Precondition: all of the characters in word appear in the master string;
 *                       word.length() > 0
 *  @return an ArrayList of string parts of the master string that can be combined
 *          to create word
 */
public ArrayList<StringPart> encodeString(String word)
```

30. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

A statistician is studying sequences of numbers obtained by repeatedly tossing a six-sided number cube. On each side of the number cube is a single number in the range of 1 to 6, inclusive, and no number is repeated on the cube. The statistician is particularly interested in runs of numbers. A run occurs when two or more consecutive tosses of the cube produce the same value. For example, in the following sequence of cube tosses, there are runs starting at positions 1, 6, 12, and 14

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Result	1	5	5	4	3	1	2	2	2	2	6	1	3	3	5	5	5	5

The number cube is represented by the following class.

```
public class NumberCube
{
  /** @return an integer value between 1 and 6, inclusive
   */
  public int toss()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

You will implement a method that collects the results of several tosses of a number cube and another method that calculates the longest run found in a sequence of tosses.

a. Write the method getCubeTosses that takes a number cube and a number of tosses as parameters. The method should return an array of the values produced by tossing the number cube the given number of times.

Complete method getCubeTosses below.

```
/** Returns an array of the values obtained by tossing a number cube numTosses times.
 *  @param cube a NumberCube
 *  @param numTosses the number of tosses to be recorded
 *         Precondition: numTosses > 0
 *  @return an array of numTosses values
 */
public static int[] getCubeTosses(NumberCube cube, int numTosses)
```

b. Write the method getLongestRun that takes as its parameter an array of integer values representing a series of number cube tosses. The method returns the starting index in the array of a run of maximum size. A run is defined as the repeated occurrence of the same value in two or more consecutive positions in the array.

For example, the following array contains two runs of length 4, one starting at index 6 and another starting at index 14. The method may return either of those starting indexes.

If there are no runs of any value, the method returns -1.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Result	1	**5**	**5**	4	3	1	**2**	**2**	**2**	**2**	6	1	**3**	**3**	**5**	**5**	**5**	**5**

Complete method getLongestRun below.

```
/** Returns the starting index of a longest run of two or more consecutive repeated values
 *  in the array values.
 *  @param values an array of integer values representing a series of number cube tosses
 *         Precondition: values.length > 0
 *  @return the starting index of a run of maximum size;
 *          -1 if there is no run
 */
public static int getLongestRun(int[] values)
```

31. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

An electric car that runs on batteries must be periodically recharged for a certain number of hours. The battery technology in the car requires that the charge time not be interrupted.

The cost for charging is based on the hour(s) during which the charging occurs. A rate table lists the 24 one-hour periods, numbered from 0 to 23, and the corresponding hourly cost for each period. The same rate table is used for each day. Each hourly cost is a positive integer. A sample rate table is given below.

Hour	Cost
0	50
1	60
2	160
3	60
4	80
5	100
6	100
7	120

Hour	Cost
8	150
9	150
10	150
11	200
12	40
13	240
14	220
15	220

Hour	Cost
16	200
17	200
18	180
19	180
20	140
21	100
22	80
23	60

The class BatteryCharger below uses a rate table to determine the most economic time to charge the battery. You will write two of the methods for the BatteryCharger class.

```
public class BatteryCharger
{
  /** rateTable has 24 entries representing the charging costs for hours 0 through 23. */
  private int[] rateTable;

  /** Determines the total cost to charge the battery starting at the beginning of startHour.
   *  @param startHour the hour at which the charge period begins
   *         Precondition: 0 ≤ startHour ≤ 23
   *  @param chargeTime the number of hours the battery needs to be charged
   *         Precondition: chargeTime > 0
   *  @return the total cost to charge the battery
   */
  private int getChargingCost(int startHour, int chargeTime)
  {  /* to be implemented in part (a) */  }

  /** Determines start time to charge the battery at the lowest cost for the given charge time.
   *  @param chargeTime the number of hours the battery needs to be charged
   *         Precondition: chargeTime > 0
   *  @return an optimal start time, with 0 ≤ returned value ≤ 23
   */
  public int getChargeStartTime(int chargeTime)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the BatteryCharger method getChargingCost that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

   For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

   Note that a charge period consists of consecutive hours that may extend over more than one day.

   Complete method getChargingCost below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
```

b. Write the BatteryCharger method getChargeStartTime that returns the start time that will allow the battery to be charged at minimal cost. If there is more than one possible start time that produces the

minimal cost, any of those start times can be returned.

For example, using the rate table given at the beginning of the question, the following table shows the resulting minimal costs and optimal starting hour of several possible charges.

Hours of Charge Time	Minimum Cost	Start Hour of Charge	Last Hour of Charge
1	40	12	12
2	110	0 **or** 23	1 **or** 0 (the next day)
7	550	22	4 (the next day)
30	3,710	22	3 (two days later)

Assume that getChargingCost works as specified, regardless of what you wrote in part (a).

Complete method getChargeStartTime below.

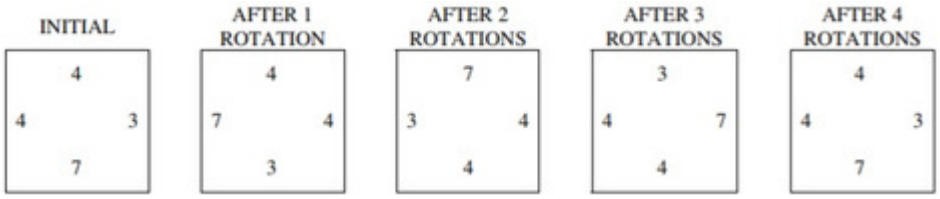
```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
```

32. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

A game uses square tiles that have numbers on their sides. Each tile is labeled with a number on each of its four sides and may be rotated clockwise, as illustrated below.

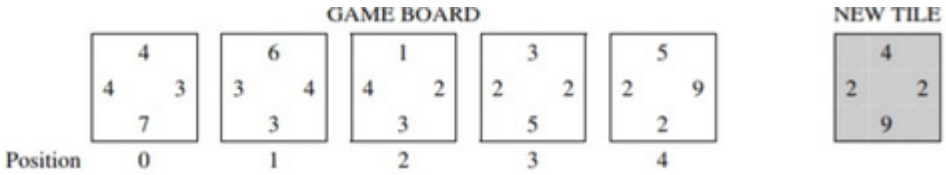


The tiles are represented by the NumberTile class, as given below.

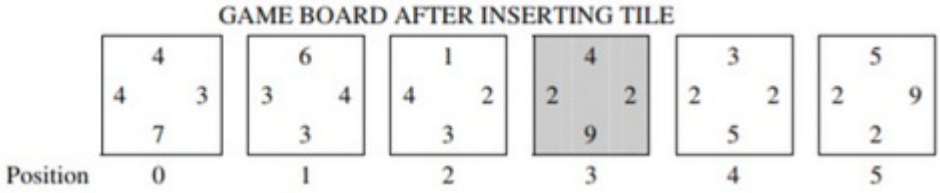
```
public class NumberTile
{
  /** Rotates the tile 90 degrees clockwise
   */
  public void rotate()
  {  /* implementation not shown */  }

  /** @return value at left edge of tile
   */
  public int getLeft()
  {  /* implementation not shown */  }

  /** @return value at right edge of tile
   */
  public int getRight()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

Tiles are placed on a game board so that the adjoining sides of adjacent tiles have the same number. The following figure illustrates an arrangement of tiles and shows a new tile that is to be placed on the game board.



In its original orientation, the new tile can be inserted between the tiles at positions 2 and 3 or between the tiles at positions 3 and 4. If the new tile is rotated once, it can be inserted before the tile at position 0 (the first tile) or after the tile at position 4 (the last tile). Assume that the new tile, in its original orientation, is inserted between the tiles at positions 2 and 3. As a result of the insertion, the tiles at positions 3 and 4 are moved one location to the right, and the new tile is inserted at position 3, as shown below.

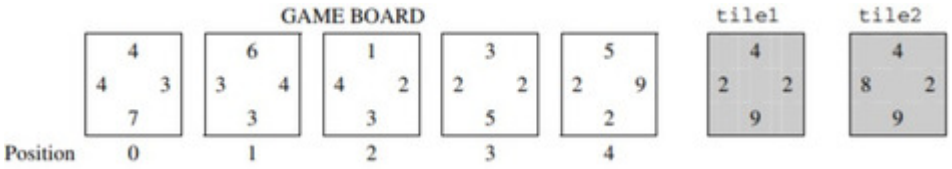


A partial definition of the TileGame class is given below.

```
public class TileGame
{
  /** represents the game board; guaranteed never to be null */
  private ArrayList<NumberTile> board;

  public TileGame()
  { board = new ArrayList<NumberTile>(); }

  /** Determines where to insert tile, in its current orientation, into game board
   *  @param tile the tile to be placed on the game board
   *  @return the position of tile where tile is to be inserted:
   *              0 if the board is empty;
   *             -1 if tile does not fit in front, at end, or between any existing tiles;
   *             otherwise, 0 ≤ position returned ≤ board.size()
   */
  private int getIndexForFit(NumberTile tile)
  {  /* to be implemented in part (a) */  }

  /** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
   *  If there are no tiles on the game board, the tile is placed at position 0.
   *  The tile should be placed at most 1 time.
   *  Precondition: board is not null
   *  @param tile the tile to be placed on the game board
   *  @return true if tile is placed successfully; false otherwise
   *  Postcondition: the orientations of the other tiles on the board are not changed
   *  Postcondition: the order of the other tiles on the board relative to each other is not changed
   */
  public boolean insertTile(NumberTile tile)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the TileGame method getIndexForFit that determines where a given tile, in its current orientation, fits on the game board. A tile can be inserted at either end of a game board or between two existing tiles if the side(s) of the new tile match the adjacent side(s) of the tile(s) currently on the game board. If there are no tiles on the game board, the position for the insert is 0. The method returns the position that the new tile will occupy on the game board after it has been inserted. If there are multiple possible positions for the tile, the method will return any one of them. If the given tile does not fit anywhere on the game board, the method returns -1.

For example, the following diagram shows a game board and two potential tiles to be placed. The call getIndexForFit(tile1) can return either 3 or 4 because tile1 can be inserted between the tiles at positions 2 and 3, or between the tiles at positions 3 and 4. The call getIndexForFit(tile2) returns -1 because tile2, in its current orientation, does not fit anywhere on the game board.



Complete method getIndexForFit below.

```
/** Determines where to insert tile, in its current orientation, into game board
 *  @param tile the tile to be placed on the game board
 *  @return the position of tile where tile is to be inserted:
 *            0 if the board is empty;
 *           -1 if tile does not fit in front, at end, or between any existing tiles;
 *           otherwise, 0 ≤ position returned ≤ board.size()
 */
private int getIndexForFit(NumberTile tile)
```

b. Write the TileGame method insertTile that attempts to insert the given tile on the game board. The method returns true if the tile is inserted successfully and false only if the tile cannot be placed on the board in any orientation.

Assume that getIndexForFit works as specified, regardless of what you wrote in part (a).

Complete method insertTile below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
/** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
 *  If there are no tiles on the game board, the tile is placed at position 0.
 *  The tile should be placed at most 1 time.
 *  Precondition: board is not null
 *  @param tile the tile to be placed on the game board
 *  @return true if tile is placed successfully; false otherwise
 *  Postcondition: the orientations of the other tiles on the board are not changed
 *  Postcondition: the order of the other tiles on the board relative to each other is not changed
 */
public boolean insertTile(NumberTile tile)
```

33. **Directions**: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

An organization raises money by selling boxes of cookies. A cookie order specifies the variety of cookie and the number of boxes ordered. The declaration of the CookieOrder class is shown below.

```
public class CookieOrder
{
  /** Constructs a new CookieOrder object. */
  public CookieOrder(String variety, int numBoxes)
  {  /* implementation not shown */  }

  /** @return the variety of cookie being ordered
   */
  public String getVariety()
  {  /* implementation not shown */  }

  /** @return the number of boxes being ordered
   */
  public int getNumBoxes()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

The MasterOrder class maintains a list of the cookies to be purchased. The declaration of the MasterOrder class is shown below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class MasterOrder
{
  /** The list of all cookie orders */
  private List<CookieOrder> orders;

  /** Constructs a new MasterOrder object. */
  public MasterOrder()
  {  orders = new ArrayList<CookieOrder>();  }

  /** Adds theOrder to the master order.
   *  @param theOrder the cookie order to add to the master order
   */
  public void addOrder(CookieOrder theOrder)
  {  orders.add(theOrder);  }

  /** @return the sum of the number of boxes of all of the cookie orders
   */
  public int getTotalBoxes()
  {  /* to be implemented in part (a) */  }

  /** Removes all cookie orders from the master order that have the same variety of
   *  cookie as cookieVar and returns the total number of boxes that were removed.
   *  @param cookieVar the variety of cookies to remove from the master order
   *  @return the total number of boxes of cookieVar in the cookie orders removed
   */
  public int removeVariety(String cookieVar)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. The getTotalBoxes method computes and returns the sum of the number of boxes of all cookie orders. If there are no cookie orders in the master order, the method returns 0.

   Complete method getTotalBoxes below.

```
/** @return the sum of the number of boxes of all of the cookie orders
 */
public int getTotalBoxes()
```

b. The removeVariety method updates the master order by removing all of the cookie orders in which the variety of cookie matches the parameter cookieVar. The master order may contain zero or more cookie orders with the same variety as cookieVar. The method returns the total number of boxes removed from the master order.

   For example, consider the following code segment.

```
MasterOrder goodies = new MasterOrder();
goodies.addOrder(new CookieOrder("Chocolate Chip", 1));
goodies.addOrder(new CookieOrder("Shortbread", 5));
goodies.addOrder(new CookieOrder("Macaroon", 2));
goodies.addOrder(new CookieOrder("Chocolate Chip", 3));
```

   After the code segment has executed, the contents of the master order are as shown in the following table.

"Chocolate Chip" 1	"Shortbread" 5	"Macaroon" 2	"Chocolate Chip" 3

The method call goodies.removeVariety("Chocolate Chip") returns 4 because there were two Chocolate Chip cookie orders totaling 4 boxes. The master order is modified as shown below.

"Shortbread" 5	"Macaroon" 2

The method call goodies.removeVariety("Brownie") returns 0 and does not change the master order.

Complete method removeVariety below.

```
/** Removes all cookie orders from the master order that have the same variety of
 *  cookie as cookieVar and returns the total number of boxes that were removed.
 *  @param cookieVar the variety of cookies to remove from the master order
 *  @return the total number of boxes of cookieVar in the cookie orders removed
 */
public int removeVariety(String cookieVar)
```

34. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

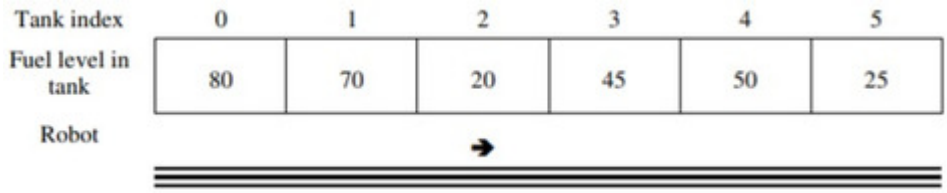
Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit. Digital sounds can be represented as an array of integer values. For this question, you will write two unrelated methods of the Sound class.

A fuel depot has a number of fuel tanks arranged in a line and a robot that moves a filling mechanism back and forth along the line so that the tanks can be filled. A fuel tank is specified by the FuelTank interface below.

```
public interface FuelTank
{
  /** @return an integer value that ranges from 0 (empty) to 100 (full) */
  int getFuelLevel();
}
```

A fuel depot keeps track of the fuel tanks and the robot. The following figure represents the tanks and the robot in a fuel depot. The robot, indicated by the arrow, is currently at index 2 and is facing to the right.



The state of the robot includes the index of its location and the direction in which it is facing (to the right or to the left). This information is specified in the FuelRobot interface as shown in the following declaration.

```
public interface FuelRobot
{
  /** @return the index of the current location of the robot */
  int getCurrentIndex();

  /** Determine whether the robot is currently facing to the right
   *  @return true  if the robot is facing to the right (toward tanks with larger indexes)
   *          false if the robot is facing to the left (toward tanks with smaller indexes)
   */
  boolean isFacingRight();

  /** Changes the current direction of the robot */
  void changeDirection();

  /** Moves the robot in its current direction by the number of locations specified.
   *  @param numLocs the number of locations to move. A value of 1 moves
   *                 the robot to the next location in the current direction.
   *          Precondition: numLocs > 0
   */
  void moveForward(int numLocs);
}
```

A fuel depot is represented by the FuelDepot class as shown in the following class declaration.

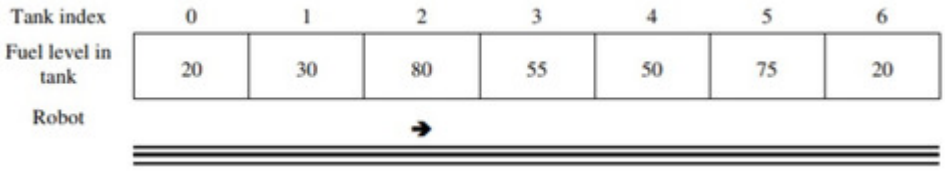
```
public class FuelDepot
{
  /** The robot used to move the filling mechanism */
  private FuelRobot filler;

  /** The list of fuel tanks */
  private List<FuelTank> tanks;

  /** Determines and returns the index of the next tank to be filled.
   *  @param threshold fuel tanks with a fuel level ≤ threshold may be filled
   *  @return index of the location of the next tank to be filled
   *  Postcondition: the state of the robot has not changed
   */
  public int nextTankToFill(int threshold)
  {  /* to be implemented in part (a) */  }

  /** Moves the robot to location locIndex.
   *  @param locIndex the index of the location of the tank to move to
   *          Precondition: 0 ≤ locIndex < tanks.size()
   *  Postcondition: the current location of the robot is locIndex
   */
  public void moveToLocation(int locIndex)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the FuelDepot method nextTankToFill that returns the index of the next tank to be filled.

   The index for the next tank to be filled is determined according to the following rules:

- Return the index of a tank with the lowest fuel level that is less than or equal to a given threshold. If there is more than one fuel tank with the same lowest fuel level, any of their indexes can be returned.
- If there are no tanks with a fuel level less than or equal to the threshold, return the robot's current index.

For example, suppose the tanks contain the fuel levels shown in the following figure.



The following table shows the results of several independent calls to nextTankToFill.

threshold	Return Value	Rationale
50	0 or 6	20 is the lowest fuel level, so either 0 or 6 can be returned.
15	2	There are no tanks with a fuel level ≤ threshold, so the robot's current index is returned.

Complete method nextTankToFill below.

```
/** Determines and returns the index of the next tank to be filled.
 *  @param threshold fuel tanks with a fuel level ≤ threshold may be filled
 *  @return index of the location of the next tank to be filled
 *  Postcondition: the state of the robot has not changed
 */
public int nextTankToFill(int threshold)
```

b. Write the FuelDepot method moveToLocation that will move the robot to the given tank location. Because the robot can only move forward, it may be necessary to change the direction of the robot before having it move. Do not move the robot past the end of the line of fuel tanks.

Complete method moveToLocation below.

```
/** Moves the robot to location locIndex.
 *  @param locIndex the index of the location of the tank to move to
 *         Precondition: 0 ≤ locIndex < tanks.size()
 *  Postcondition: the current location of the robot is locIndex
 */
public void moveToLocation(int locIndex)
```

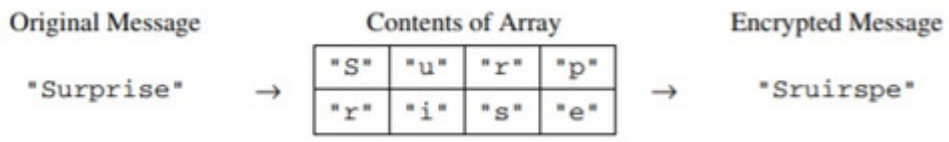
35. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit. Digital sounds can be represented as an array of integer values. For this question, you will write two unrelated methods of the Sound class.

In this question you will write two methods for a class RouteCipher that encrypts (puts into a coded form) a message by changing the order of the characters in the message. The route cipher fills a two-dimensional array with single-character substrings of the original message in row-major order, encrypting the message by retrieving the single-character substrings in column-major order.

For example, the word "Surprise" can be encrypted using a 2-row, 4-column array as follows.



An incomplete implementation of the RouteCipher class is shown below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class RouteCipher
{
  /** A two-dimensional array of single-character strings, instantiated in the constructor */
  private String[][] letterBlock;

  /** The number of rows of letterBlock, set by the constructor */
  private int numRows;

  /** The number of columns of letterBlock, set by the constructor */
  private int numCols;

  /** Places a string into letterBlock in row-major order.
   *  @param str the string to be processed
   *  Postcondition:
   *    if str.length() < numRows * numCols, "A" is placed in each unfilled cell
   *    if str.length() > numRows * numCols, trailing characters are ignored
   */
  private void fillBlock(String str)
  {  /* to be implemented in part (a) */  }

  /** Extracts encrypted string from letterBlock in column-major order.
   *  Precondition: letterBlock has been filled
   *  @return the encrypted string from letterBlock
   */
  private String encryptBlock()
  {  /* implementation not shown */  }

  /** Encrypts a message.
   *  @param message the string to be encrypted
   *  @return the encrypted message;
   *          if message is the empty string, returns the empty string
   */
  public String encryptMessage(String message)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the method fillBlock that fills the two-dimensional array letterBlock with one-character strings from the string passed as parameter str.

   The array must be filled in row-major order—the first row is filled from left to right, then the second row is filled from left to right, and so on, until all rows are filled.

   If the length of the parameter str is smaller than the number of elements of the array, the string "A" is placed in each of the unfilled cells. If the length of str is larger than the number of elements in the array, the trailing characters are ignored.

   For example, if letterBlock has 3 rows and 5 columns and str is the string "Meet at noon", the resulting contents of letterBlock would be as shown in the following table.

"M"	"e"	"e"	"t"	" "
"a"	"t"	" "	"n"	"o"
"o"	"n"	"A"	"A"	"A"

   If letterBlock has 3 rows and 5 columns and str is the string "Meet at midnight", the resulting contents of letterBlock would be as shown in the following table.

"M"	"e"	"e"	"t"	" "
"a"	"t"	" "	"m"	"i"
"d"	"n"	"i"	"g"	"h"

The following expression may be used to obtain a single-character string at position k of the string str.

str.substring(k, k + 1)

Complete method fillBlock below.

```
/** Places a string into letterBlock in row-major order.
 *  @param str the string to be processed
 *  Postcondition:
 *    if str.length() < numRows * numCols, "A" is placed in each unfilled cell
 *    if str.length() > numRows * numCols, trailing characters are ignored
 */
private void fillBlock(String str)
```

b. Write the method encryptMessage that encrypts its string parameter message. The method builds an encrypted version of message by repeatedly calling fillBlock with consecutive, nonoverlapping substrings of message and concatenating the results returned by a call to encryptBlock after each call to fillBlock. When all of message has been processed, the concatenated string is returned. Note that if message is the empty string, encryptMessage returns an empty string.

The following example shows the process carried out if letterBlock has 2 rows and 3 columns and encryptMessage("Meet at midnight") is executed.

Substring	letterBlock after Call to fillBlock	Value Returned by encryptBlock	Concatenated String
"Meet a"	"M" "e" "e" "t" " " "a"	"Mte ea"	"Mte ea"
"t midn"	"t" " " "m" "i" "d" "n"	"ti dmn"	"Mte eati dmn"
"ight"	"i" "g" "h" "t" "A" "A"	"itgAhA"	"Mte eati dmnitgAhA"

In this example, the method returns the string "Mte eati dmnitgAhA".

Assume that fillBlock and encryptBlock methods work as specified. Solutions that reimplement the functionality of one or both of these methods will not receive full credit.

Complete method encryptMessage below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
/** Encrypts a message.
 *  @param message the string to be encrypted
 *  @return the encrypted message;
 *          if message is the empty string, returns the empty string
 */
public String encryptMessage(String message)
```

36. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the appendices have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

Consider a software system that models a horse barn. Classes that represent horses implement the following interface.

```
public interface Horse
{
  /** @return the horse's name */
  String getName();

  /** @return the horse's weight */
  int getWeight();

  // There may be methods that are not shown.
}
```

A horse barn consists of N numbered spaces. Each space can hold at most one horse. The spaces are indexed starting from 0; the index of the last space is $N - 1$. No two horses in the barn have the same name.

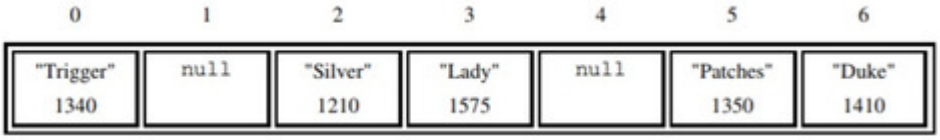
The declaration of the HorseBarn class is shown below. You will write two unrelated methods of the HorseBarn class.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class HorseBarn
{
  /** The spaces in the barn. Each array element holds a reference to the horse
   *  that is currently occupying the space. A null value indicates an empty space.
   */
  private Horse[] spaces;

  /** Returns the index of the space that contains the horse with the specified name.
   *  Precondition: No two horses in the barn have the same name.
   *  @param name the name of the horse to find
   *  @return the index of the space containing the horse with the specified name;
   *          -1 if no horse with the specified name is in the barn.
   */
  public int findHorseSpace(String name)
  {  /* to be implemented in part (a) */  }

  /** Consolidates the barn by moving horses so that the horses are in adjacent spaces,
   *  starting at index 0, with no empty space between any two horses.
   *  Postcondition: The order of the horses is the same as before the consolidation.
   */
  public void consolidate()
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the HorseBarn method findHorseSpace. This method returns the index of the space in which the horse with the specified name is located. If there is no horse with the specified name in the barn, the method returns -1.

For example, assume a HorseBarn object called sweetHome has horses in the following spaces.



The following table shows the results of several calls to the findHorseSpace method.

Method Call	Value Returned	Reason
`sweetHome.findHorseSpace("Trigger")`	0	A horse named Trigger is in space 0.
`sweetHome.findHorseSpace("Silver")`	2	A horse named Silver is in space 2.
`sweetHome.findHorseSpace("Coco")`	-1	A horse named Coco is not in the barn.

Information repeated from the beginning of the question

```
public interface Horse

String getName()
int getWeight()

public class HorseBarn

private Horse[] spaces
public int findHorseSpace(String name)
public void consolidate()
```

Complete method findHorseSpace below.

```
/** Returns the index of the space that contains the horse with the specified name.
 *  Precondition: No two horses in the barn have the same name.
 *  @param name the name of the horse to find
 *  @return the index of the space containing the horse with the specified name;
 *          -1 if no horse with the specified name is in the barn.
 */
public int findHorseSpace(String name)
```

b. Write the HorseBarn method consolidate. This method consolidates the barn by moving horses so that the horses are in adjacent spaces, starting at index 0, with no empty spaces between any two horses. After the barn is consolidated, the horses are in the same order as they were before the consolidation.

For example, assume a barn has horses in the following spaces.

0	1	2	3	4	5	6
"Trigger" 1340	null	"Silver" 1210	null	null	"Patches" 1350	"Duke" 1410

The following table shows the arrangement of the horses after consolidate is called.

0	1	2	3	4	5	6
"Trigger" 1340	"Silver" 1210	"Patches" 1350	"Duke" 1410	null	null	null

Information repeated from the beginning of the question

```
public interface Horse

String getName()
int getWeight()

public class HorseBarn

private Horse[] spaces
public int findHorseSpace(String name)
public void consolidate()
```

Complete method consolidate below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
/** Consolidates the barn by moving horses so that the horses are in adjacent spaces,
 *  starting at index 0, with no empty space between any two horses.
 *  Postcondition: The order of the horses is the same as before the consolidation.
 */
public void consolidate()
```

37. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the appendices have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

A music Web site keeps track of downloaded music. For each download, the site uses a DownloadInfo object to store a song's title and the number of times it has been downloaded. A partial declaration for the DownloadInfo class is shown below.

```
public class DownloadInfo
{
  /** Creates a new instance with the given unique title and sets the
   *  number of times downloaded to 1.
   *  @param title the unique title of the downloaded song
   */
  public DownloadInfo(String title)
  {  /* implementation not shown */  }

  /** @return the title */
  public String getTitle()
  {  /* implementation not shown */  }

  /** Increment the number times downloaded by 1 */
  public void incrementTimesDownloaded()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

The list of downloaded information is stored in a MusicDownloads object. A partial declaration for the MusicDownloads class is shown below.

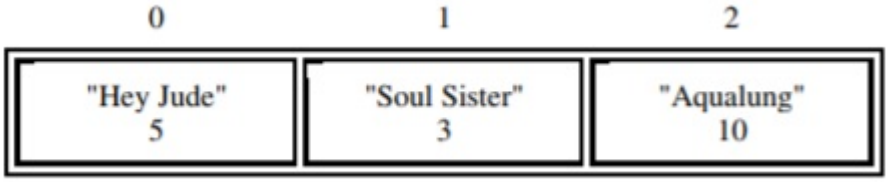
Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
public class MusicDownloads
{
  /** The list of downloaded information.
   *  Guaranteed not to be null and not to contain duplicate titles.
   */
  private List<DownloadInfo> downloadList;

  /** Creates the list of downloaded information. */
  public MusicDownloads()
  {  downloadList = new ArrayList<DownloadInfo>();  }

  /** Returns a reference to the DownloadInfo object with the requested title if it exists.
   *  @param title the requested title
   *  @return a reference to the DownloadInfo object with the
   *          title that matches the parameter title if it exists in the list;
   *          null otherwise.
   *  Postcondition:
   *    - no changes were made to downloadList.
   */
  public DownloadInfo getDownloadInfo(String title)
  {  /* to be implemented in part (a) */  }

  /** Updates downloadList with information from titles.
   *  @param titles a list of song titles
   *  Postcondition:
   *    - there are no duplicate titles in downloadList.
   *    - no entries were removed from downloadList.
   *    - all songs in titles are represented in downloadList.
   *    - for each existing entry in downloadList, the download count is increased by
   *        the number of times its title appeared in titles.
   *    - the order of the existing entries in downloadList is not changed.
   *    - the first time an object with a title from titles is added to downloadList, it
   *        is added to the end of the list.
   *    - new entries in downloadList appear in the same order
   *        in which they first appear in titles.
   *    - for each new entry in downloadList, the download count is equal to
   *        the number of times its title appeared in titles.
   */
  public void updateDownloads(List<String> titles)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

a. Write the MusicDownloads method getDownloadInfo, which returns a reference to a DownloadInfo object if an object with a title that matches the parameter title exists in the downloadList. If no song in downloadList has a title that matches the parameter title, the method returns null.

   For example, suppose variable webMusicA refers to an instance of MusicDownloads and that the table below represents the contents of downloadList. The list contains three DownloadInfo objects. The object at position 0 has a title of "Hey Jude" and a download count of 5. The object at position 1 has a title of

"Soul Sister" and a download count of 3. The object at position 2 has a title of "Aqualung" and a download count of 10.



The call webMusicA.getDownloadInfo("Aqualung") returns a reference to the object in position 2 of the list.

The call webMusicA.getDownloadInfo("Happy Birthday") returns null because there are no DownloadInfo objects with that title in the list.

Class information repeated from the beginning of the question

```
public class DownloadInfo

public DownloadInfo(String title)
public String getTitle()
public void incrementTimesDownloaded()

public class MusicDownloads

private List<DownloadInfo> downloadList
public DownloadInfo getDownloadInfo(String title)
public void updateDownloads(List<String> titles)
```

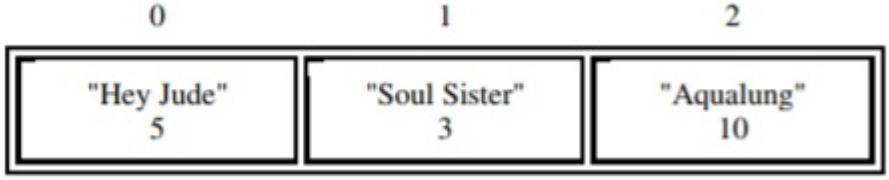
Complete method getDownloadInfo below.

```
/** Returns a reference to the DownloadInfo object with the requested title if it exists.
 *  @param title the requested title
 *  @return a reference to the DownloadInfo object with the
 *          title that matches the parameter title if it exists in the list;
 *          null otherwise.
 *  Postcondition:
 *    - no changes were made to downloadList.
 */
public DownloadInfo getDownloadInfo(String title)
```

b. Write the MusicDownloads method updateDownloads, which takes a list of song titles as a parameter. For each title in the list, the method updates downloadList, either by incrementing the download count if a DownloadInfo object with the same title exists, or by adding a new DownloadInfo object with that title and a download count of 1 to the end of the list. When a new DownloadInfo object is added to the end of the list, the order of the already existing entries in downloadList remains unchanged.

For example, suppose variable webMusicB refers to an instance of MusicDownloads and that the table below represents the contents of the instance variable downloadList.

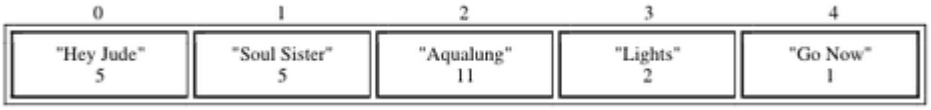
Unit2_objectsPartAJLC9_18_2023_Questions.pdf



Assume that the variable List songTitles has been defined and contains the following entries.

{"Lights", "Aqualung", "Soul Sister", "Go Now", "Lights", "Soul Sister"}

The call webMusicB.updateDownloads(songTitles) results in the following downloadList with incremented download counts for the objects with titles of "Soul Sister" and "Aqualung". It also has a new DownloadInfo object with a title of "Lights" and a download count of 2, and another DownloadInfo object with a title of "Go Now" and a download count of 1. The order of the already existing entries remains unchanged.



Class information repeated from the beginning of the question

```
public class DownloadInfo

public DownloadInfo(String title)
public String getTitle()
public void incrementTimesDownloaded()

public class MusicDownloads

private List<DownloadInfo> downloadList
public DownloadInfo getDownloadInfo(String title)
public void updateDownloads(List<String> titles)
```

In writing your solution, you must use the getDownloadInfo method. Assume that getDownloadInfo works as specified, regardless of what you wrote for part (a).

Complete method updateDownloads below.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
/** Updates downloadList with information from titles.
 *  @param titles a list of song titles
 *  Postcondition:
 *    - there are no duplicate titles in downloadList.
 *    - no entries were removed from downloadList.
 *    - all songs in titles are represented in downloadList.
 *    - for each existing entry in downloadList, the download count is increased by
 *          the number of times its title appeared in titles.
 *    - the order of the existing entries in downloadList is not changed.
 *    - the first time an object with a title from titles is added to downloadList, it
 *          is added to the end of the list.
 *    - new entries in downloadList appear in the same order
 *          in which they first appear in titles.
 *    - for each new entry in downloadList, the download count is equal to
 *          the number of times its title appeared in titles.
 */
public void updateDownloads(List<String> titles )
```

38. **Directions:** SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

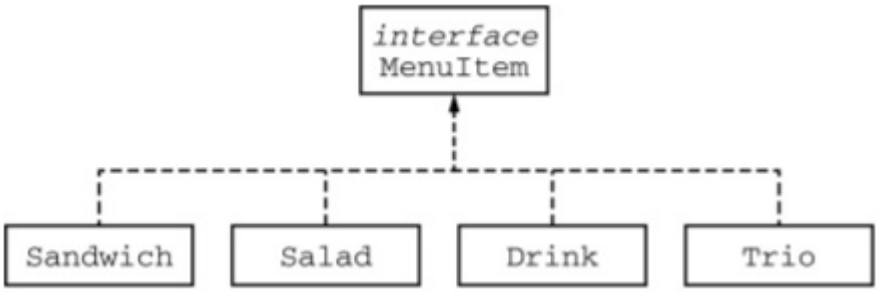
- Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
- In writing solutions, you may use any of the accessible methods that are listed in classes defined in the question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

The menu at a lunch counter includes a variety of sandwiches, salads, and drinks. The menu also allows a customer to create a "trio," which consists of three menu items: a sandwich, a salad, and a drink. The price of the trio is the sum of the two highest-priced menu items in the trio; one item with the lowest price is free.

Each menu item has a name and a price. The four types of menu items are represented by the four classes Sandwich, Salad, Drink, and Trio. All four classes implement the following MenuItem interface.

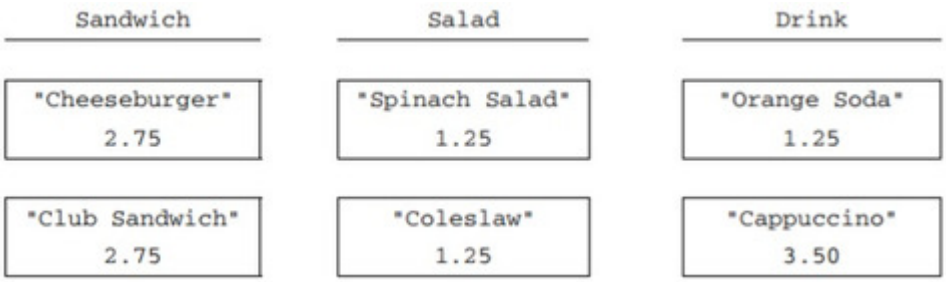
```
public interface MenuItem
{
  /** @return the name of the menu item */
  String getName();

  /** @return the price of the menu item */
  double getPrice();
}
```

The following diagram shows the relationship between the MenuItem interface and the Sandwich, Salad, Drink, and Trio classes.



For example, assume that the menu includes the following items. The objects listed under each heading are instances of the class indicated by the heading.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf



The menu allows customers to create Trio menu items, each of which includes a sandwich, a salad, and a drink. The name of the Trio consists of the names of the sandwich, salad, and drink, in that order, each separated by "/" and followed by a space and then "Trio". The price of the Trio is the sum of the two highest-priced items in the Trio; one item with the lowest price is free.

A trio consisting of a cheeseburger, spinach salad, and an orange soda would have the name "Cheeseburger/Spinach Salad/Orange Soda Trio" and a price of $4.00 (the two highest prices are $2.75 and $1.25). Similarly, a trio consisting of a club sandwich, coleslaw, and a cappuccino would have the name "Club Sandwich/Coleslaw/Cappuccino Trio" and a price of $6.25 (the two highest prices are $2.75 and $3.50).

Write the Trio class that implements the MenuItem interface. Your implementation must include a constructor that takes three parameters representing a sandwich, salad, and drink. The following code segment should have the indicated behavior.

```
Sandwich sandwich;
Salad salad;
Drink drink;
/* Code that initializes sandwich, salad, and drink */

Trio trio = new Trio(sandwich, salad, drink);   // Compiles without error

Trio trio1 = new Trio(salad, sandwich, drink); // Compile-time error
Trio trio2 = new Trio(sandwich, salad, salad); // Compile-time error
```

Write the complete Trio class below.

39. Consider the following method.

```
public void doSomething()
{
     System.out.println("Something has been done");
}
```

Each of the following statements appears in a method in the same class as `doSomething`. Which of the following statements are valid uses of the method `doSomething` ?

I. `doSomething();`
II. `String output = doSomething();`
III. `System.out.println(doSomething());`

(A) I only
(B) II only
(C) I and II only
(D) I and III only
(E) I, II, and III

40. Consider the following class definition.

```
public class ExamScore
{
     private String studentId;
     private double score;
     public ExamScore(String sid, double s)
     {
         studentId = sid;
         score = s;
     }
     public double getScore()
     {
         return score;
     }
     public void bonus(int b)
     {
         score += score * b/100.0;
     }
}
```

Assume that the following code segment appears in a class other than `ExamScore`.

```
ExamScore es = new ExamScore("12345", 80.0);
es.bonus(5);
System.out.println(es.getScore());
```

What is printed as a result of executing the code segment?

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

(A) `4.0`
(B) `5.0`
(C) `80.0`
(D) `84.0`
(E) `85.0`

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

41. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

3. This question involves a system to analyze the inventory of different types of flowers sold by a florist. Flower types are represented by the following `Flower` class.

```
public class Flower
{
 /** Returns the name of the flower (e.g., "rose", "tulip", etc.) */
 public String getName()
 {  /* implementation not shown */  }

 /** Returns the quantity in stock of the flower */
 public int getQuantity()
 {  /* implementation not shown */  }

 /** Sets the quantity in stock of the flower to newQ */
 public void setQuantity(int newQ)
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

In the following `FlowerShop` class, the array `flowerInventory` contains information about the flowers sold at a flower shop. You will implement two methods in the `FlowerShop` class.

```
public class FlowerShop
{
 /** An array of the flowers sold at the flower shop, sorted
 alphabetically by flower name
   *   Guaranteed to be nonempty and to contain only non-null entries
  */
 private Flower[] flowerInventory;
```

```
/** Returns a copy of Flower[] arr in which the array elements have
been sorted in
  *  order from highest quantity to lowest quantity
 */
public Flower[] sortByQuantity(Flower[] arr)
{  /* implementation not shown */  }

/** Updates the Flower objects contained in flowerInventory, as
described in
 *   part (a)
  *   Precondition: newInventory has the same flower names in the same
positions
 *             as flowerInventory.
 *  Postcondition: newInventory is unchanged.
 */
public void updateInventory(Flower[] newInventory)
{  /* to be implemented in part (a) */  }

/** Returns true if the top n flowers by quantity in stock at the
flower shop are the
  *  same as the top n flowers by quantity in stock in otherInventory,
as described
 *  in part (b)
  *  Precondition: n is less than or equal to the lengths of
flowerInventory and
 *            otherInventory.
 *  Postcondition: flowerInventory and otherInventory are unchanged.
 */
public boolean topNSame(int n, Flower[] otherInventory)
{  /* to be implemented in part (b) */  }

// There may be instance variables, constructors, and methods that are
not shown.
}
```

(a) Write the `FlowerShop` method `updateInventory`, which increases the quantities of the flowers in `flowerInventory` by the quantities of the flowers in the array parameter `newInventory`. Each quantity of a flower in `newInventory` is added to the quantity of the corresponding flower in `flowerInventory`. The arrays `flowerInventory` and `newInventory` are the same length and contain the same flower names in the same positions.

For example, assume that `sunnyFlowers` has been declared as a `FlowerShop` object and `newFlowers` is a `Flower` array.

The array `flowerInventory` before the call `sunnyFlowers.updateInventory(newFlowers)`:

"daffodil"	"rose"	"tulip"
225	550	419

The array `newFlowers`:

"daffodil"	"rose"	"tulip"
200	50	150

The array `flowerInventory` after the call `sunnyFlowers.updateInventory(newFlowers)`:

"daffodil"	"rose"	"tulip"
425	600	569

Complete method `updateInventory`.

```
/** Updates the Flower objects contained in flowerInventory, as described
in
 *   part (a)
  *   Precondition: newInventory has the same flower names in the same
positions
 *             as flowerInventory.
 *  Postcondition: newInventory is unchanged.
 */
public void updateInventory(Flower[] newInventory)
```

(b) Write the `FlowerShop` method `topNSame`, which compares the flowers that appear in the first `n` positions of `flowerInventory` with the flowers that appear in the first `n` positions of `otherInventory` when both arrays are sorted in decreasing order by quantity in stock. The method returns `true` if the first `n` `Flower` objects in both arrays are the same and appear in the same position in both sorted arrays and returns `false` otherwise.

A helper method, `sortByQuantity`, has been provided. It returns a copy of its `Flower[]` parameter sorted in decreasing order by quantity in stock.

For example, assume that `sunnyFlowers` has been declared as a `FlowerShop` object and that `stockB` is an array of `Flower` objects.

The array `flowerInventory` in `sunnyFlowers`:

"carnation"	"daffodil"	"daisy"	"rose"	"tulip"
50	320	375	550	419

The array `stockB`:

`"carnation"` `250`	`"daffodil"` `220`	`"daisy"` `500`	`"rose"` `525`	`"tulip"` `310`

The method call `sunnyFlowers.topNSame(3, stockB)` should return `false`. For `sunnyFlowers`, the top three flowers by quantity (in order) are `"rose"`, `"tulip"`, and `"daisy"`. However, for `stockB`, the top three flowers by quantity (in order) are `"rose"`, `"daisy"`, and `"tulip"`.

Complete method `topNSame`. You must use `sortByQuantity` appropriately to receive full credit.

```
/** Returns true if the top n flowers by quantity in stock at the flower
shop are the
  *   same as the top n flowers by quantity in stock in otherInventory,
as described
  *  in part (b)
  *  Precondition: n is less than or equal to the lengths of
flowerInventory and
  *           otherInventory.
  *  Postcondition: flowerInventory and otherInventory are unchanged.
 */
public boolean topNSame(int n, Flower[] otherInventory)
```

(c) A programmer would like to add a method called `getMostExpensiveFlower`, which returns the name of the flower with the highest price.

Write a description of how you would change the `Flower` and `FlowerShop` classes in order to support this modification.

Make sure to include the following in your response.

- Write the method header for the `getMostExpensiveFlower` method.
- Identify any new or modified variables, constructors, or methods aside from the `getMostExpensiveFlower` method. **Do not write the program code for this change.**
- Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the implementation of the `getMostExpensiveFlower` method. **Do not write the program code for this change.**

42. This question involves reasoning about a simulation of a frog hopping in a straight line. The frog attempts to hop to a goal within a specified number of hops. The simulation is encapsulated in the following FrogSimulation class. You will write two of the methods in this class.

```
public class FrogSimulation
{
   /** Distance, in inches, from the starting position to the goal. */
   private int goalDistance;

   /** Maximum number of hops allowed to reach the goal. */
   private int maxHops;


   /** Constructs a FrogSimulation where dist is the distance, in inches, from the starting
    *  position to the goal, and numHops is the maximum number of hops allowed to reach the goal.
    *  Precondition: dist > 0; numHops > 0
    */
   public FrogSimulation(int dist, int numHops)
   {
      goalDistance = dist;
      maxHops = numHops;
   }


   /** Returns an integer representing the distance, in inches, to be moved when the frog hops.
    */
   private int hopDistance()
   {  /* implementation not shown */  }


   /** Simulates a frog attempting to reach the goal as described in part (a).
    *  Returns true if the frog successfully reached or passed the goal during the simulation;
    *          false otherwise.
    */
   public boolean simulate()
   {  /* to be implemented in part (a) */  }


   /** Runs num simulations and returns the proportion of simulations in which the frog
    *  successfully reached or passed the goal.
    *  Precondition: num > 0
    */
   public double runSimulations(int num)
   {  /* to be implemented in part (b) */  }
}
```

a. Write the simulate method, which simulates the frog attempting to hop in a straight line to a goal from the frog's starting position of 0 within a maximum number of hops. The method returns true if the frog successfully reached the goal within the maximum number of hops; otherwise, the method returns false.

   The FrogSimulation class provides a method called hopDistance that returns an integer representing the distance (positive or negative) to be moved when the frog hops. A positive distance represents a move toward the goal. A negative distance represents a move away from the goal. The returned distance may vary from call to call. Each time the frog hops, its position is adjusted by the value returned by a call to the hopDistance method.

   The frog hops until one of the following conditions becomes true:

   • The frog has reached or passed the goal.

   • The frog has reached a negative position.

   • The frog has taken the maximum number of hops without reaching the goal.

The following example shows a declaration of a FrogSimulation object for which the goal distance is 24 inches and the maximum number of hops is 5. The table shows some possible outcomes of calling the simulate method.

FrogSimulation sim = new FrogSimulation(24, 5);

	Values returned by `hopDistance()`	Final position of frog	Return value of `sim.simulate()`
Example 1	`5, 7, -2, 8, 6`	24	`true`
Example 2	`6, 7, 6, 6`	25	`true`
Example 3	`6, -6, 31`	31	`true`
Example 4	`4, 2, -8`	-2	`false`
Example 5	`5, 4, 2, 4, 3`	18	`false`

Class information for this question

```
public class FrogSimulation

private int goalDistance
private int maxHops

private int hopDistance()
public boolean simulate()
public double runSimulations(int num)
```

Complete method simulate below. You must use hopDistance appropriately to receive full credit.

/** Simulates a frog attempting to reach the goal as described in part (a). * Returns true if the frog successfully reached or passed the goal during the simulation; * false otherwise. */

public boolean simulate()

b. Write the runSimulations method, which performs a given number of simulations and returns the proportion of simulations in which the frog successfully reached or passed the goal. For example, if the parameter passed to runSimulations is 400, and 100 of the 400 simulate method calls returned true, then the runSimulations method should return 0.25.

Complete method runSimulations below. Assume that simulate works as specified, regardless of what you wrote in part (a). You must use simulate appropriately to receive full credit.

```
/** Runs num simulations and returns the proportion of simulations in which the frog
 *  successfully reached or passed the goal.
 *  Precondition: num > 0
 */
public double runSimulations(int num)
```

43. Consider the following methods, which appear in the same class.

```
public int function1(int i, int j)
{
     return i + j;
}

public int function2(int i, int j)
{
     return j - i;
}
```

Which of the following statements, if located in a method in the same class, will initialize the variable `x` to `11`?

(A) `int x = function2(4, 5) + function1(1, 3);`

(B) `int x = function1(4, 5) + function2(1, 3);`

(C) `int x = function1(4, 5) + function2(3, 1);`

(D) `int x = function1(3, 1) + function2(4, 5);`

(E) `int x = function2(3, 1) + function1(4, 5);`

44. Consider the following class declaration.

```
public class GameClass
{
     private int numPlayers;
     private boolean gameOver;

     public Game()
     {
         numPlayers = 1;
         gameOver = false;
     }

     public void addPlayer()
     {
         numPlayers++;
     }

     public void endGame()
     {
         gameOver = true;
     }
}
```

Assume that the `GameClass` object `game` has been properly declared and initialized in a method in a class other than `GameClass`. Which of the following statements are valid?

I. `game.numPlayers++;`
II. `game.addPlayer();`
III. `game.gameOver();`
IV. `game.endGame();`

(A) IV only

(B) I and III only

(C) I and IV only

(D) II and IV only

(E) II, III, and IV only

45. In this question, you will complete methods in classes that can be used to represent a multi-player game. You will be able to implement these methods without knowing the specific game or the players' strategies.

The GameState interface describes the current state of the game. Different implementations of the interface can be used to play different games. For example, the state of a checkers game would include the positions of all the pieces on the board and which player should make the next move.

The GameState interface specifies these methods. The Player class will be described in part (a).

```
public interface GameState
{
  /** @return true if the game is in an ending state;
   *          false otherwise
   */
  boolean isGameOver();

  /** Precondition: isGameOver() returns true
   *  @return the player that won the game or null if there was no winner
   */
  Player getWinner();

  /** Precondition: isGameOver() returns false
   *  @return the player who is to make the next move
   */
  Player getCurrentPlayer();

  /** @return a list of valid moves for the current player;
   *          the size of the returned list is 0 if there are no valid moves.
   */
  ArrayList<String> getCurrentMoves();

  /** Updates game state to reflect the effect of the specified move.
   *  @param move a description of the move to be made
   */
  void makeMove(String move);

  /** @return a string representing the current GameState
   */
  String toString();
}
```

The makeMove method makes the move specified, updating the state of the game being played. Its parameter is a String that describes the move. The format of the string depends on the game. In tic-tac-toe, for example, the move might be something like "X-1-1", indicating an X is put in the position (1, 1).

a. The Player class provides a method for selecting the next move. By extending this class, different playing strategies can be modeled.

```
public class Player
{
  private String name;    // name of this player

  public Player(String aName)
  {  name = aName;  }

  public String getName()
  {  return name;  }

  /** This implementation chooses the first valid move.
   *  Override this method in subclasses to define players with other strategies.
   *  @param state the current state of the game; its current player is this player.
   *  @return a string representing the move chosen;
   *          "no move" if no valid moves for the current player.
   */
  public String getNextMove(GameState state)
  {  /* implementation not shown */  }
}
```

The method getNextMove returns the next move to be made as a string, using the same format as that used by makeMove in GameState. Depending on how the getNextMove method is implemented, a player can exhibit different game-playing strategies.

Write the complete class declaration for a RandomPlayer class that is a subclass of Player. The class should have a constructor whose String parameter is the player's name. It should override the getNextMove method to randomly select one of the valid moves in the given game state. If there are no valid moves available for the player, the string "no move" should be returned.

b. The GameDriver class is used to manage the state of the game during game play. The GameDriver class can be written without knowing details about the game being played

```
public class GameDriver
{
  private GameState state;  // the current state of the game

  public GameDriver(GameState initial)
  {  state = initial;  }

  /** Plays an entire game, as described in the problem description
   */
  public void play()
  {  /* to be implemented in part (b) */  }

  // There may be fields, constructors, and methods that are not shown.
}
```

Write the GameDriver method play. This method should first print the initial state of the game. It should then repeatedly determine the current player and that player's next move, print both the player's name and

the chosen move, and make the move. When the game is over, it should stop making moves and print either the name of the winner and the word "wins" or the message "Game ends in a draw" if there is no winner. You may assume that the GameState makeMove method has been implemented so that it will properly handle any move description returned by the Player getNextMove method, including the string "no move".

Complete method play below

```
/** Plays an entire game, as described in the problem description
 */
public void play()
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.
In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

The `Gizmo` class represents gadgets that people purchase. Some `Gizmo` objects are electronic and others are not. A partial definition of the `Gizmo` class is shown below.

```
public class Gizmo
{
     /** Returns the name of the manufacturer of this Gizmo. */
     public String getMaker()
     {
         /* implementation not shown */
     }

     /** Returns true if this Gizmo is electronic, and false otherwise. */
     public boolean isElectronic()
     {
         /* implementation not shown */
     }

     /** Returns true if this Gizmo is equivalent to the Gizmo object
   represented by the
      * parameter, and false otherwise.
      */
     public boolean equals(Object other)
     {
         /* implementation not shown */
     }

     // There may be instance variables, constructors, and methods not shown.
}
```

The `OnlinePurchaseManager` class manages a sequence of `Gizmo` objects that an individual has purchased from an online vendor. You will write two methods of the `OnlinePurchaseManager` class. A partial definition of the `OnlinePurchaseManager` class is shown below.

```
public class OnlinePurchaseManager
{
     /** An ArrayList of purchased Gizmo objects, instantiated in the
   constructor. */
     private ArrayList<Gizmo> purchases;

     /** Returns the number of purchased Gizmo objects that are electronic and
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
  are
      * manufactured by maker, as described in part (a).
      */
     public int countElectronicsByMaker(String maker)
     {
         /* to be implemented in part (a) */
     }

     /** Returns true if any pair of adjacent purchased Gizmo objects are
  equivalent, and
      * false otherwise, as described in part (b).
      */
     public boolean hasAdjacentEqualPair()
     {
         /* to be implemented in part (b) */
     }

     // There may be instance variables, constructors, and methods not shown.
}
```

46. **(a)** Write the `countElectronicsByMaker` method. The method examines the `ArrayList` instance variable `purchases` to determine how many `Gizmo` objects purchased are electronic and are manufactured by `maker`.

Assume that the `OnlinePurchaseManager` object `opm` has been declared and initialized so that the `ArrayList purchases` contains `Gizmo` objects as represented in the following table.

Index in `purchases`	0	1	2	3	4	5
Value returned by method call `isElectronic()`	`true`	`false`	`true`	`false`	`true`	`false`
Value returned by method call `getMaker()`	`"ABC"`	`"ABC"`	`"XYZ"`	`"lmnop"`	`"ABC"`	`"ABC"`

The following table shows the value returned by some calls to `countElectronicsByMaker`.

Method Call	Return Value
`opm.countElectronicsByMaker("ABC")`	2
`opm.countElectronicsByMaker("lmnop")`	0
`opm.countElectronicsByMaker("XYZ")`	1
`opm.countElectronicsByMaker("QRP")`	0

Complete method `countElectronicsByMaker` below.

```
/** Returns the number of purchased Gizmo objects that are electronic and
 * whose manufacturer is maker, as described in part (a).
 */
public int countElectronicsByMaker(String maker)
```

(b) When purchasing items online, users occasionally purchase two identical items in rapid succession without intending to do so (e.g., by clicking a purchase button twice). A vendor may want to check a user's purchase history to detect such occurrences and request confirmation.

Write the `hasAdjacentEqualPair` method. The method detects whether two adjacent `Gizmo` objects in `purchases` are equivalent, using the `equals` method of the `Gizmo` class. If an adjacent equivalent pair is found, the `hasAdjacentEqualPair` method returns `true`. If no such pair is found, or if `purchases` has fewer than two elements, the method returns `false`.

Complete method `hasAdjacentEqualPair` below.

```
/** Returns true if any pair of adjacent purchased Gizmo objects are
equivalent, and
 * false otherwise, as described in part (b).
 */
public boolean hasAdjacentEqualPair()
```

47. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

A mathematical sequence is an ordered list of numbers. This question involves a sequence called a *hailstone sequence*. If n is the value of a term in the sequence, then the following rules are used to find the next term, if one exists.

- If n is 1, the sequence terminates.
- If n is even, then the next term is $\frac{n}{2}$.
- If n is odd, then the next term is $3n + 1$.

For this question, assume that when the rules are applied, the sequence will eventually terminate with the term $n = 1$.

The following are examples of hailstone sequences.

Example 1: 5, 16, 8, 4, 2, 1

- The first term is 5, so the second term is $5 * 3 + 1 = 16$.
- The second term is 16, so the third term is $\frac{16}{2} = 8$.
- The third term is 8, so the fourth term is $\frac{8}{2} = 4$.
- The fourth term is 4, so the fifth term is $\frac{4}{2} = 2$.
- The fifth term is 2, so the sixth term is $\frac{2}{2} = 1$.
- Since the sixth term is 1, the sequence terminates.

Example 2: 8, 4, 2, 1

- The first term is 8, so the second term is $\frac{8}{2} = 4$.
- The second term is 4, so the third term is $\frac{4}{2} = 2$.
- The third term is 2, so the fourth term is $\frac{2}{2} = 1$.
- Since the fourth term is 1, the sequence terminates.

The `Hailstone` class, shown below, is used to represent a hailstone sequence. You will write three methods in the `Hailstone` class.

```
public class Hailstone
{
     /** Returns the length of a hailstone sequence that starts with n,
       * as described in part (a).
       * Precondition: n > 0
       */
     public static int hailstoneLength(int n)
```

```
    { /* to be implemented in part (a) */ }
  /** Returns true if the hailstone sequence that starts with n is
considered long
      * and false otherwise, as described in part (b).
      * Precondition: n > 0
      */
    public static boolean isLongSeq(int n)
    { /* to be implemented in part (b) */ }
  /** Returns the proportion of the first n hailstone sequences that are
considered long,
      * as described in part (c).
      * Precondition: n > 0
      */
    public static double propLong(int n)
    { /* to be implemented in part (c) */ }
  // There may be instance variables, constructors, and methods not
shown.
}
```

(a) The length of a hailstone sequence is the number of terms it contains. For example, the hailstone sequence in example 1 (5, 16, 8, 4, 2, 1) has a length of 6 and the hailstone sequence in example 2 (8, 4, 2, 1) has a length of 4.

Write the method `hailstoneLength(int n)`, which returns the length of the hailstone sequence that starts with `n`.

```
/** Returns the length of a hailstone sequence that starts with n,
 * as described in part (a).
 * Precondition: n > 0
 */
public static int hailstoneLength(int n)
```

Class information for this question

```
public class Hailstone
public static int hailstoneLength(int n)
public static boolean isLongSeq(int n)
public static double propLong(int n)
```

(b) A hailstone sequence is considered long if its length is greater than its starting value. For example, the hailstone sequence in example 1 (5, 16, 8, 4, 2, 1) is considered long because its length (6) is greater than its starting value (5). The hailstone sequence in example 2 (8, 4, 2, 1) is not considered long because its length (4) is less than or equal to its starting value (8).

Write the method `isLongSeq(int n)`, which returns `true` if the hailstone sequence starting with `n` is considered long and returns `false` otherwise. Assume that `hailstoneLength` works as intended, regardless of what you wrote in part (a). You must use `hailstoneLength` appropriately to receive full credit.

```
/** Returns true if the hailstone sequence that starts with n is
considered long
 * and false otherwise, as described in part (b).
 * Precondition: n > 0
 */
public static boolean isLongSeq(int n)
```

(c) The method `propLong(int n)` returns the proportion of long hailstone sequences with starting values between 1 and `n`, inclusive.

Consider the following table, which provides data about the hailstone sequences with starting values between 1 and 10, inclusive.

Starting Value	Terms in the Sequence	Length of the Sequence	Long?
1	1	1	No
2	2, 1	2	No
3	3, 10, 5, 16, 8, 4, 2, 1	8	Yes
4	4, 2, 1	3	No
5	5, 16, 8, 4, 2, 1	6	Yes
6	6, 3, 10, 5, 16, 8, 4, 2, 1	9	Yes
7	7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1	17	Yes
8	8, 4, 2, 1	4	No
9	9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1	20	Yes
10	10, 5, 16, 8, 4, 2, 1	7	No

The method call `Hailstone.propLong(10)` returns `0.5`, since 5 of the 10 hailstone sequences shown in the table are considered long.

Write the `propLong` method. Assume that `hailstoneLength` and `isLongSeq` work as intended, regardless of what you wrote in parts (a) and (b). You must use `isLongSeq` appropriately to receive full credit.

```
/** Returns the proportion of the first n hailstone sequences that are
considered long,
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

```
 * as described in part (c).
 * Precondition: n > 0
 */
public static double propLong(int n)
```

Class information for this question

```
public class Hailstone
public static int hailstoneLength(int n)
public static boolean isLongSeq(int n)
public static double propLong(int n)
```

48. In this question, you will implement two methods for a class Hotel that is part of a hotel reservation system. The Hotel class uses the Reservation class shown below. A Reservation is for the person and room number specified when the Reservation is constructed.

```
public class Reservation
{
  public Reservation(String guestName, int roomNumber)
  { /* implementation not shown */ }

  public int getRoomNumber()
  { /* implementation not shown */ }

  // private data and other methods not shown
}
```

An incomplete declaration for the Hotel class is shown below. Each hotel in the hotel reservation system has rooms numbered 0, 1, 2, . . . , up to the last room number in the hotel. For example, a hotel with 100 rooms would have rooms numbered 0, 1, 2, . . . , 99.

```
public class Hotel
{
  private Reservation[] rooms;
    // each element corresponds to a room in the hotel;
    // if rooms[index] is null, the room is empty;
    // otherwise, it contains a reference to the Reservation
    // for that room, such that
    // rooms[index].getRoomNumber() returns index

  private ArrayList waitList;
    // contains names of guests who have not yet been
    // assigned a room because all rooms are full


  // if there are any empty rooms (rooms with no reservation),
  // then create a reservation for an empty room for the
  // specified guest and return the new Reservation;
  // otherwise, add the guest to the end of waitList
  // and return null
  public Reservation requestRoom(String guestName)
  { /* to be implemented in part (a) */ }


  // release the room associated with parameter res, effectively
  // canceling the reservation;
  // if any names are stored in waitList, remove the first name
  // and create a Reservation for this person in the room
  // reserved by res; return that new Reservation;
  // if waitList is empty, mark the room specified by res as empty and
  // return null
  // precondition:  res is a valid Reservation for some room
  //                in this hotel
  public Reservation cancelAndReassign(Reservation res)
  { /* to be implemented in part (b) */ }

  // constructors and other methods not shown
}
```

a. Write the Hotel method requestRoom. Method requestRoom attempts to reserve a room in the hotel for a given guest. If there are any empty rooms in the hotel, one of them will be assigned to the named guest and the newly created reservation is returned. If there are no empty rooms, the guest is added to the end of the waiting list and null is returned.

Complete method requestRoom below.

```
// if there are any empty rooms (rooms with no reservation),
// then create a reservation for an empty room for the
// specified guest and return the new Reservation;
// otherwise, add the guest to the end of waitList
// and return null
public Reservation requestRoom(String guestName)
```

b. Write the Hotel method cancelAndReassign. Method cancelAndReassign releases a previous reservation. If the waiting list for the hotel contains any names, the vacated room is reassigned to the first person at the beginning of the list. That person is then removed from the waiting list and the newly created reservation is returned. If no one is waiting, the room is marked as empty and null is returned.

In writing cancelAndReassign you may call any accessible methods in the Reservation and Hotel classes. Assume that these methods work as specified.

Complete method cancelAndReassign below.

```
// release the room associated with parameter res, effectively
// canceling the reservation;
// if any names are stored in waitList, remove the first name
// and create a Reservation for this person in the room
// reserved by res; return that new Reservation;
// if waitList is empty, mark the room specified by res as empty and
// return null
// precondition:  res is a valid Reservation for some room
//                in this hotel
public Reservation cancelAndReassign(Reservation res)
```

49. Consider the following class definition.

```
public class Bird
{
     private String species;
     private String color;
     private boolean canFly;
     public Bird(String str, String col, boolean cf)
     {
         species = str;
         color = col;
         canFly = cf;
     }
}
```

Which of the following constructors, if added to the `Bird` class, will cause a compilation error?

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

(A)
```
public Bird()
{
     species = "unknown";
     color = "unknown";
     canFly = false;
}
```

(B)
```
public Bird(boolean cf)
{
     species = "unknown";
     color = "unknown";
     canFly = cf;
}
```

(C)
```
public Bird(String col, String str)
{
     species = str;
     color = col;
     canFly = false;
}
```

(D)
```
public Bird(boolean cf, String str, String col)
{
     species = str;
     color = col;
     canFly = cf;
}
```

(E)
```
public Bird(String col, String str, boolean cf)
{
     species = str;
     color = col;
     canFly = cf;
}
```

50. A student has created a `Car` class. The class contains variables to represent the following.

A `String` variable called `color` to represent the color of the car
An `int` variable called `year` to represent the year the car was made
A `String` variable called `make` to represent the manufacturer of the car
A `String` variable called `model` to represent the model of the car

The object `vehicle` will be declared as type `Car`.

Which of the following descriptions is accurate?

(A) An instance of the `vehicle` class is `Car`.

(B) An instance of the `Car` object is `vehicle`.

(C) An attribute of the `year` object is `int`.

(D) An attribute of the `vehicle` object is `color`.

(E) An attribute of the `Car` instance is `vehicle`.

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

51. A student has created an `OrderedPair` class to represent points on an xy-plane. The class contains the following.

An `int` variable called `x` to represent an x-coordinate.
An `int` variable called `y` to represent a y-coordinate.
A method called `printXY` that will print the values of `x` and `y`.

The object `origin` will be declared as type `OrderedPair`.

Which of the following descriptions is accurate?

(A) `origin` is an instance of the `printXY` method.

(B) `origin` is an instance of the `OrderedPair` class.

(C) `origin` is an instance of two `int` objects.

(D) `OrderedPair` is an instance of the `origin` object.

(E) `printXY` is an instance of the `OrderedPair` class.

52. Consider the following attempts at method overloading.

I.

```
public class Overload
{
     public int average(int x, int y)
     { /* implementation not shown */ }

     public int average(int value1, int value2)
     { /* implementation not shown */ }

     // There may be instance variables, constructors,
     // and methods that are not shown.
}
```

II.

```
public class Overload
{
     public int average(int x, int y)
     { /* implementation not shown */ }

     public int average(int x, int y, int z)
     { /* implementation not shown */ }

     // There may be instance variables, constructors
     // and methods that are not shown.
}
```

III.

```
public class Overload
{
     public int average(int x, int y)
     { /* implementation not shown */ }

     public int average(double x, double y)
     { /* implementation not shown */ }

     // There may be instance variables, constructors,
     // and methods that are not shown.
}
```

Which of the attempts at method overloading will compile without error?

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

53. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

3. This question involves information about songs, which are represented by the `Song` class shown below.

```
public class Song
{
 /** Returns the length of a song in seconds */
 public int getLength()
 {  /* implementation not shown */  }

 /** Returns true if the song has been played recently and false
 otherwise */
 public boolean recentlyPlayed()
 {  /* implementation not shown */  }

 /** Marks a song as played today */
 public void setPlayedToday()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

The `Playlist` class, shown below, contains methods that organize songs into playlists. You will write two methods of the `Playlist` class.

```
public class Playlist
{
 /** A list of songs in the playlist
   *   Guaranteed not to be null and to contain only non-null entries
  */
 private ArrayList<Song> songList;

 /** Returns the next song to be played, as described in part (a)
```

```
   *   Precondition: There is at least one song in songList that has not
 been played
  *             recently.
  */
 public Song getNextSong()
 {  /* to be implemented in part (a) */  }

 /** Removes all songs from songList that are within 30  seconds in
 length of the next
   *   song to be played, as described in part (b)
  */
 public void removeLong()
 {  /* to be implemented in part (b) */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

(a) The `getNextSong` method returns the next song in `songList` to be played. The next song to be played is the longest song found in the list that has not been played recently. Whether or not a song has been played recently is determined by the `recentlyPlayed` method.

The `getNextSong` method marks the selected song as having been played today.

Write the `getNextSong` method.

```
/** Returns the next song to be played, as described in part (a)
  *   Precondition: There is at least one song in songList that has not
been played recently.
 */
public Song getNextSong()
```

(b) The `removeLong` method removes from `songList` all songs with a length that is within 30 seconds of the length of the song returned by `getNextSong`. For example, if the song identified by the `getNextSong` method has a length of `200` seconds, songs with lengths of `220` and `170` seconds would be removed, but songs with lengths of `231` and `165` seconds would be retained. Note that songs are removed based on their lengths, regardless of whether or not they have been played recently.

Write the `removeLong` method. You must use `getNextSong` appropriately in order to receive full credit.

```
/** Removes all songs from songList that are within 30 seconds in length
of the next song
  *  to be played, as described in part (b)
 */
public void removeLong()
```

(c) A programmer would like to add a method called `getShortestByArtist`, which returns the shortest song on the playlist by a given artist.

Write a description of how you would change the `Song` and `Playlist` classes in order to support this modification.

Make sure to include the following in your response.

The method header for the `getShortestByArtist` method
Identify any new or modified variables, constructors, or methods aside from `getShortestByArtist`. **Do not write the program code for this change.**
Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the `getShortestByArtist` method. **Do not write the program code for this change.**

54. Consider the following `Point2D` class.

```
public class Point2D
{
     private double xCoord;
     private double yCoord;

     public Point2D(double x, double y)
     {
         xCoord = x;
         yCoord = y;
     }
}
```

Which of the following code segments, appearing in a class other than `Point2D`, will correctly create an instance of a `Point2D` object?

(A) `Point2D p = (3.0, 4.0);`

(B) `Point2D p = Point2D(3.0, 4.0);`

(C) `new p = Point2D(3.0, 4.0);`

(D) `new Point2D = p(3.0, 4.0);`

(E) `Point2D p = new Point2D(3.0, 4.0);`

55. Consider the following class definition.

```
public class Points
{
     private double num1;
     private double num2;
     public Points(int n1, int n2)            // Line 6
     {
         num1 = n1;                            // Line 8
         num2 = n2;                            // Line 9
     }
     public void incrementPoints(int value)   // Line 12
     {
         n1 += value;                          // Line 14
         n2 += value;                          // Line 15
     }
}
```

The class does not compile. Which of the following identifies the error in the class definition?

(A) In line 6, the `Points` constructor must have a `void` return type.

(B) In lines 8 and 9, `int` values cannot be assigned to `double` variables.

(C) In line 12, the `incrementPoints` method must have a non-`void` return type.

(D) In lines 14 and 15, the variables `n1` and `n2` are not defined.

(E) In lines 14 and 15, the variable `value` is not defined.

56. Consider the following methods, which appear in the same class.

```
public void printSum(int x, double y)
{
     System.out.println(x + y);
}

public void printProduct(double x, int y)
{
     System.out.println(x * y);
}
```

Consider the following code segment, which appears in a method in the same class as `printSum` and `printProduct`.

```
int num1 = 5;
double num2 = 10.0;
printSum(num1, num2);
printProduct(num1, num2);
```

What, if anything, is printed as a result of executing the code segment?

(A)
```
15
50
```

(B)
```
15
50.0
```

(C)
```
15.0
50
```

(D)
```
15.0
50.0
```

(E) Nothing is printed because the code does not compile.

Directions: Select the choice that best fits each statement. The following question(s) refer to the following information.

Consider the following partial class declaration.

```
public class SomeClass
{
  private int myA;
  private int myB;
  private int myC;

  // Constructor(s) not shown

  public int getA()
  {  return myA;  }

  public void setB(int value)
  {  myB = value;  }
}
```

57. The following declaration appears in another class.
SomeClass obj = new SomeClass ( );
Which of the following code segments will compile without error?

(A) int x = obj.getA ( );

(B)
```
int x;
obj.getA (x);
```

(C) int x = obj.myA;

(D) int x = SomeClass.getA ( );

(E) int x = getA(obj);

---

58. A student has created a `Song` class. The class contains the following variables.

A `String` variable called `artist` to represent the artist name
A `String` variable called `title` to represent the song title
A `String` variable called `album` to represent the album title

The object `happyBirthday` will be declared as type `Song`.

Which of the following statements is true?

(A) `artist`, `title`, and `album` are instances of the `Song` class.

(B) `happyBirthday` is an instance of three `String` objects.

(C) `happyBirthday` is an instance of the `Song` class.

(D) `Song` is an instance of the `happyBirthday` object.

(E) `Song` is an instance of three `String` objects.

59. Consider the following class definition.

```
public class Student
{
     private int studentID;
     private int gradeLevel;
     private boolean honorRoll;

     public Student(int s, int g)
     {
         studentID = s;
         gradeLevel = g;
         honorRoll = false;
     }

     public Student(int s)
     {
         studentID = s;
         gradeLevel = 9;
         honorRoll = false;
     }
}
```

Which of the following code segments would successfully create a new `Student` object?

I.
```
Student one = new Student(328564, 11);
```
II.
```
Student two = new Student(238783);
```
III.
```
int id = 392349;
int grade = 11;
Student three = new Student(id, grade);
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I, II, and III

60. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

1. This question involves a system to manage a student's assignments in a course. Assignments are represented by the following `Assignment` class.

```
public class Assignment
{
 /** Returns the title of the assignment */
 public String getTitle()
 {  /* implementation not shown */  }

 /** Returns the grade earned for the assignment
   *  Postcondition: The value returned is greater than or equal to 0.
  */
 public int getGrade()
 {  /* implementation not shown */  }

 /** Sets the grade for the assignment to x */
 public void setGrade(int x)
 {  /* implementation not shown */  }

 /** Returns the category of the assignment */
 public String getCategory()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

A student's assignments for a course are stored in a `StudentAssignments` object, which contains a list of the student's assignments. You will implement two methods in this class.

```
public class StudentAssignments
{
```

```
  /** A list containing the student's assignments
    *  Guaranteed not to be null and to contain only non-null entries
   */
  private ArrayList<Assignment> assignmentList;

  /** Returns the number of assignments in a given category that have a
  grade greater than
    *  or equal to 90, as described in part (a)
   */
  public int getNumberOfAs(String cat)
  {  /* to be implemented in part (a) */  }

  /** Replaces a single assignment grade in a given category if the
  number of As in
    *  the category is greater than 5 but at least one score is below 70,
  as described in part (b)
   */
  public void updateGrade(String cat)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are
  not shown.
}
```

(a) Write the `StudentAssignments` method `getNumberOfAs`. The method returns the number of assignments in a given category that have a grade greater than or equal to `90`.

Complete method `getNumberOfAs`.

```
/** Returns the number of assignments in a given category that have a
grade greater than or
  *  equal to 90, as described in part (a)
 */
public int getNumberOfAs(String cat)
```

(b) Write the `StudentAssignments` method `updateGrade`. The method replaces one and only one grade below 70 in a given category, if any such grade exists and if the given category has more than five assignments with a grade of A. The identified grade is replaced with the value 70. If more than one such assignment exists, the grade of any one of those assignments may be replaced.

Assume that `getNumberOfAs` works as specified, regardless of what you wrote for part (a). You must use `getNumberOfAs` appropriately to receive full credit.

Complete method `updateGrade`.

```
/** Replaces a single assignment grade in a given category if the number
of As in
  *  the category is greater than 5 but at least one score is below 70,
as described in part (b)
 */
public void updateGrade(String cat)
```

(c) The programmer would like to add a method called `getAllGroupAssignments` that returns a list of assignments that have been designated as having parts where students worked in groups. A group assignment can appear in any category.

Write a description of how you would change the `Assignment` and `StudentAssignments` classes in order to support this modification.

Make sure to include the following in your response.

- Write the method header for the `getAllGroupAssignments` method.
- Identify any new or modified variables, constructors, or methods aside from the `getAllGroupAssignments` method. **Do not write the program code for this change.**
- Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the `getAllGroupAssignments` method. **Do not write the program code for this change.**

61. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

The class `SingleTable` represents a table at a restaurant.

```
public class SingleTable
{
     /** Returns the number of seats at this table. The value
       * is always greater than or equal to 4.
       */
     public int getNumSeats()
     { /* implementation not shown */ }

     /** Returns the height of this table in centimeters. */
     public int getHeight()
     { /* implementation not shown */ }

     /** Returns the quality of the view from this table. */
     public double getViewQuality()
     { /* implementation not shown */ }

     /** Sets the quality of the view from this table to value.
       */
     public void setViewQuality(double value)
     { /* implementation not shown */ }

     // There may be instance variables, constructors, and methods
     // that are not shown.
}
```

At the restaurant, customers can sit at tables that are composed of two single tables pushed together. You will write a class `CombinedTable` to represent the result of combining two `SingleTable` objects, based on the following rules and the examples in the chart that follows.

A `CombinedTable` can seat a number of customers that is two fewer than the total number of seats in its two `SingleTable` objects (to account for seats lost when the tables are pushed together).
A `CombinedTable` has a desirability that depends on the views and heights of the two single tables. If the two single tables of a `CombinedTable` object are the same height, the desirability of the `CombinedTable` object is the average of the view qualities of the two single tables.
If the two single tables of a `CombinedTable` object are not the same height, the desirability of the

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

`CombinedTable` object is 10 units less than the average of the view qualities of the two single tables.

Assume `SingleTable` objects `t1`, `t2`, and `t3` have been created as follows.

`SingleTable t1` has 4 seats, a view quality of 60.0, and a height of 74 centimeters.
`SingleTable t2` has 8 seats, a view quality of 70.0, and a height of 74 centimeters.
`SingleTable t3` has 12 seats, a view quality of 75.0, and a height of 76 centimeters.

The chart contains a sample code execution sequence and the corresponding results.

Statement	Value Returned (blank if no value)	Class Specification
`CombinedTable c1 = new CombinedTable(t1, t2);`		A `CombinedTable` is composed of two `SingleTable` objects.
`c1.canSeat(9);`	`true`	Since its two single tables have a total of 12 seats, `c1` can seat 10 or fewer people.
`c1.canSeat(11);`	`false`	`c1` cannot seat 11 people.
`c1.getDesirability();`	`65.0`	Because `c1`'s two single tables are the same height, its desirability is the average of `60.0` and `70.0`.
`CombinedTable c2 = new CombinedTable(t2, t3);`		A `CombinedTable` is composed of two `SingleTable` objects.
`c2.canSeat(18);`	`true`	Since its two single tables have a total of 20 seats, `c2` can seat 18 or fewer people.
`c2.getDesirability();`	`62.5`	Because `c2`'s two single tables are not the same height, its desirability is 10 units less than the average of `70.0` and `75.0`.
`t2.setViewQuality(80);`		Changing the view quality of one of the tables that makes up `c2` changes the desirability of `c2`, as illustrated in the next line of the chart. Since `setViewQuality` is a `SingleTable` method, you do not need to write it.
`c2.getDesirability();`	`67.5`	Because the view quality of `t2` changed, the desirability of `c2` has also changed.

The last line of the chart illustrates that when the characteristics of a `SingleTable` change, so do those of the

`CombinedTable` that contains it.

Write the complete `CombinedTable` class. Your implementation must meet all specifications and conform to the examples shown in the preceding chart.

62. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

A theater contains rows of seats with the same number of seats in each row. Some rows contain tier 1 seats, and the remaining rows contain tier 2 seats. Tier 1 seats are closer to the stage and are more desirable. All seats in a row share the same tier.

The `Seat` class, shown below, represents seats in the theater. The `boolean` instance variable `available` is `false` if a ticket for the seat has been sold (the seat is no longer available). The `int` instance variable `tier` indicates whether the seat is a tier 1 or tier 2 seat.

```
public class Seat
{
     private boolean available;
     private int tier;
     public Seat(boolean isAvail, int tierNum)
     {
         available = isAvail;
         tier = tierNum;
     }
     public boolean isAvailable()
     {  return available;  }
     public int getTier()
     {  return tier;  }
     public void setAvailability(boolean isAvail)
     {  available = isAvail;  }
}
```

The `Theater` class represents a theater of seats. The number of seats per row and the number of tier 1 and tier 2 rows are determined by the parameters of the `Theater` constructor. Row `0` of the `theaterSeats` array represents the row closest to the stage.

```
public class Theater
{
     private Seat[][] theaterSeats;
     /** Constructs a Theater object, as described in part (a).
       * Precondition: seatsPerRow > 0; tier1Rows > 0; tier2Rows >= 0
       */
```

```
    public Theater(int seatsPerRow, int tier1Rows, int tier2Rows)
    {  /* to be implemented in part (a) */  }
    /** Returns true if a seat holder was reassigned from the seat at
  fromRow, fromCol
     * to the seat at toRow, toCol; otherwise it returns false, as
   described in part (b).
     * Precondition: fromRow, fromCol, toRow, and toCol represent valid
   row and
     * column positions in the theater.
     * The seat at fromRow, fromCol is not available.
     */
    public boolean reassignSeat(int fromRow, int fromCol,
   int toRow, int toCol)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the constructor for the `Theater` class. The constructor takes three `int` parameters, representing the number of seats per row, the number of tier 1 rows, and the number of tier 2 rows, respectively. The constructor initializes the `theaterSeats` instance variable so that it has the given number of seats per row and the given number of tier 1 and tier 2 rows and all seats are available and have the appropriate tier designation.

Row `0` of the `theaterSeats` array represents the row closest to the stage. All tier 1 seats are closer to the stage than tier 2 seats.

Complete the `Theater` constructor.

```
/** Constructs a Theater object, as described in part (a).
 * Precondition: seatsPerRow > 0; tier1Rows > 0; tier2Rows >= 0
 */
public Theater(int seatsPerRow, int tier1Rows, int tier2Rows)
```

Class information for this question

```
public class Seat
private boolean available
private int tier
public Seat(boolean isAvail, int tierNum)
public boolean isAvailable()
public int getTier()
public void setAvailability(boolean isAvail)
public class Theater
private Seat[][] theaterSeats
public Theater(int seatsPerRow, int tier1Rows, int tier2Rows)
public boolean reassignSeat(int fromRow, int fromCol,
                    int toRow, int toCol)
```

(b) Write the `reassignSeat` method, which attempts to move a person from a source seat to a destination seat. The reassignment can be made if the destination seat is available and has the same or greater tier than the source seat (that is, it is equally or less desirable). For example, a person in a tier 1 seat can be moved to a different tier 1 seat or to a tier 2 seat, but a person in a tier 2 seat can only be moved to a different tier 2 seat.

The `reassignSeat` method has four `int` parameters representing the row and column indexes of the source ("from") and destination ("to") seats. If the reassignment is possible, the source seat becomes available, the destination seat becomes unavailable, and the method returns `true`. If the seat reassignment is not possible, no changes are made to either seat and the method returns `false`. Assume that the source seat is occupied when the method is called.

Complete method `reassignSeat`.

```
/** Returns true if a seat holder was reassigned from the seat at
fromRow, fromCol
 * to the seat at toRow, toCol; otherwise it returns false, as described
 in part (b).
 * Precondition: fromRow, fromCol, toRow, and toCol represent valid row
 and
 * column positions in the theater.
 * The seat at fromRow, fromCol is not available.
 */
public boolean reassignSeat(int fromRow, int fromCol,
                   int toRow, int toCol)
```

63. SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

Consider the following `Thing` class. Each `Thing` object has a name attribute, which can be set in the constructor or by using the `setName` method. The name of a `Thing` object can be returned by the `getName` method.

```
public class Thing
{
     // attributes not shown

     /** Constructs a new Thing named myName
       */
     public Thing(String myName)
     { /* implementation not shown */ }

     /** Returns this Thing's name
       */
     public String getName()
     { /* implementation not shown */ }

     /** Sets this Thing's name to newName
       */
     public void setName(String newName)
     { /* implementation not shown */ }

     /** Returns a message as described in part (b)
       */
     public void printMessage()
     { /* implementation not shown */ }
}
```

(a) Write a statement to create a new `Thing` object `snack` that has the name `"potato chip"`.

Write the statement below.

(b) The `Thing` method `printMessage` prints a string consisting of the name of the object followed by `"_is_great"`.

Suppose the name of the `Thing` object `favFood` is `"pizza"`. Write a statement that uses the `printMessage`

method to print the string "pizza_is_great".

Write the statement below.

(c) Write a code segment to change the name of the Thing object something such that the new name consists of the old name with one character removed at random. For example, if something has name "ABCD", its new name could be set to "ACD".

Write the code segment below.

64. Consider the following class definition.

```
public class Thing
{
     public void talk()
     {
         System.out.print("Hello ");
     }

     public void name()
     {
         System.out.print("my friend");
     }

     public void greet()
     {
         talk();
         name();
     }
     /* Constructors not shown */
}
```

Which of the following code segments, if located in a method in a class other than Thing, will cause the message "Hello my friend" to be printed?

(A)
```
Thing a = new Thing();
Thing.talk();
Thing.name();
```

(B)
```
Thing a = new Thing();
Thing.greet();
```

(C)
```
Thing a = new Thing();
a.talk();
```

(D)
```
Thing a = new Thing();
a.greet();
```

(E)
```
Thing a = new Thing();
a.name();
a.talk();
```

65. This question involves reasoning about pairs of words that are represented by the following WordPair class.

```
public class WordPair
{

    /** Constructs a WordPair object. */
    public WordPair(String first, String second)
    {  /* implementation not shown */  }


    /** Returns the first string of this WordPair object. */
    public String getFirst()
    {  /* implementation not shown */  }


    /** Returns the second string of this WordPair object. */
    public String getSecond()
    {  /* implementation not shown */  }

}
```

You will implement the constructor and another method for the following WordPairList class.

```
public class WordPairList
{

    /** The list of word pairs, initialized by the constructor. */
    private ArrayList<WordPair> allPairs;


    /** Constructs a WordPairList object as described in part (a).
     *  Precondition: words.length >= 2
     */
    public WordPairList(String[] words)
    { /* to be implemented in part (a) */ }


    /** Returns the number of matches as described in part (b).
     */
    public int numMatches()
    { /* to be implemented in part (b) */ }

}
```

a. Write the constructor for the WordPairList class. The constructor takes an array of strings words as a parameter and initializes the instance variable allPairs to an ArrayList of WordPair objects. A WordPair

object consists of a word from the array paired with a word that appears later in the array. The allPairs list contains WordPair objects (words[i], words[j]) for every i and j, where $0 \leq i < j <$ words.length . Each WordPair object is added exactly once to the list. The following examples illustrate two different WordPairList objects.

Example 1

```
String[] wordNums = {"one", "two", "three"};
WordPairList exampleOne = new WordPairList(wordNums);
```

After the code segment has executed, the allPairs instance variable of exampleOne will contain the following WordPair objects in some order

```
("one", "two"), ("one", "three"), ("two", "three")
```

Example 2

```
String[] phrase = {"the", "more", "the", "merrier"};
WordPairList exampleTwo = new WordPairList(phrase);
```

After the code segment has executed, the allPairs instance variable of exampleTwo will contain the following WordPair objects in some order.

```
("the", "more"), ("the", "the"), ("the", "merrier"),
("more", "the"), ("more", "merrier"), ("the", "merrier")
```

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete the WordPairList constructor below.

```
/** Constructs a WordPairList object as described in part (a).
 *  Precondition: words.length >= 2
 */
public WordPairList(String[] words)
```

b. Write the WordPairList method numMatches. This method returns the number of WordPair objects in allPairs for which the two strings match.

For example, the following code segment creates a WordPairList object.

```
String[] moreWords = {"the", "red", "fox", "the", "red"};
WordPairList exampleThree = new WordPairList(moreWords);
```

After the code segment has executed, the allPairs instance variable of exampleThree will contain the following WordPair objects in some order. The pairs in which the first string matches the second string are shaded for illustration.

```
("the", "red"), ("the", "fox"), ("the", "the"),
("the", "red"), ("red", "fox"), ("red", "the"),
("red", "red"), ("fox", "the"), ("fox", "red"),
("the", "red")
```

The call exampleThree.numMatches() should return 2.

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Complete method numMatches below.

```
/** Returns the number of matches as described in part (b).
 */
public int numMatches()
```

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

66. Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Java Quick Reference have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

1. This question involves analyzing the difficulty levels of trivia questions used in a trivia game. Trivia questions are represented by the following `TriviaQuestion` class.

```
public class TriviaQuestion
{
 /** Returns the difficulty level of the trivia question, which is a
 value between 0 and
  *  100, inclusive
  */
 public int getDifficultyLevel()
 {  /* implementation not shown */  }

 // There may be instance variables, constructors, and methods that are
 not shown.
}
```

An array of the trivia questions that make up a trivia game is stored in a `TriviaGame` object. A partial declaration of the `TriviaGame` class is shown below. You will write two methods of the `TriviaGame` class.

```
public class TriviaGame
{
 /** An array of all trivia questions in the game
   *   Guaranteed to have at least three entries and to contain only
 non-null entries
  */
 private TriviaQuestion[] allQuestions;

 /** Returns the greatest difficulty level that appears in any trivia
 question in the game */
 public int getMostDifficult()
 {  /* implementation not shown */  }
```

```
  /** Returns the least difficulty level that appears in any trivia
  question in the game */
  public int getLeastDifficult()
  {  /* implementation not shown */  }

  /** Returns a String representing the overall rating of the trivia game
  as
    *  "Beginner", "Intermediate", or "Expert", as described in part (a)
   */
  public String getRating()
  {  /* to be implemented in part (a) */  }

  /** Returns true if the trivia questions in allQuestions appear in
  order from least
    *   difficulty to greatest difficulty and returns false otherwise, as
  described in part (b)
   */
  public boolean isDifficultyIncreasing()
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are
  not shown.
}
```

(a) Write the `TriviaGame` method `getRating`, which returns a `String` representing the difficulty level of the trivia game based on the *adjusted average* of the difficulty levels of the trivia questions in the game.

The *adjusted average* is defined as the average of the difficulty levels with the least value and greatest value ignored. If there are multiple questions with the greatest difficulty level, only one is ignored. If there are multiple questions with the least difficulty level, only one is ignored. A trivia game consists of at least three trivia questions.

Based on the adjusted average, the `getRating` method should return the following values.

- `"Expert"` if the adjusted average is at least `70.0`
- `"Intermediate"` if the adjusted average is at least `50.0` but less than `70.0`
- `"Beginner"` if the adjusted average is less than `50.0`

For example, assume that `game1` has been declared as a `TriviaGame` object and that `q1`, `q2`, `q3`, and `q4` are properly declared and initialized `TriviaQuestion` objects. The following is an example of the contents of `allQuestions` in `game1`, showing each `TriviaQuestion` object and its difficulty level.

0	1	2	3
q1 Difficulty: 50	q2 Difficulty: 90	q3 Difficulty: 85	q4 Difficulty: 90

In computing the adjusted average for the `game1` object, the least value (`50`) and greatest value (`90`) are ignored. The adjusted average (`87.5`) is computed as the average of the remaining two values, `85` and `90`.

The method call `game1.getRating()` should return `"Expert"`.

Two helper methods, `getMostDifficult` and `getLeastDifficult`, are provided in the `TriviaGame` class. You must use `getMostDifficult` and `getLeastDifficult` appropriately to receive full credit.

Complete method `getRating`.

```
/** Returns a String representing the overall rating of the trivia game
as
  *  "Beginner", "Intermediate", or "Expert", as described in part (a)
 */
public String getRating()
```

(b) Write the `TriviaGame` method `isDifficultyIncreasing`, which returns `true` if the difficulty level of each question in `allQuestions` is less than or equal to the question that follows it, if any, and returns `false` otherwise.

For example, assume that `game1` and `game2` have been declared as `TriviaGame` objects and that `q1`, `q2`, `q3`, `q4`, `q5`, and `q6` are properly declared and initialized `TriviaQuestion` objects. The following example shows the contents of `allQuestions` for `game1` and `game2`. The `TriviaQuestion` object and its difficulty level are provided for each `allQuestions` element.

The array `allQuestions` for `game1` object:

0	1	2	3	4
q1 Difficulty: 50	q2 Difficulty: 60	q3 Difficulty: 60	q4 Difficulty: 70	q5 Difficulty: 80

The array `allQuestions` for `game2` object:

0	1	2	3
q1 Difficulty: 50	q4 Difficulty: 70	q3 Difficulty: 60	q6 Difficulty: 90

The method call `game1.isDifficultyIncreasing()` should return `true`, and the method call `game2.isDifficultyIncreasing()` should return `false`.

Complete method `isDifficultyIncreasing`.

```
/** Returns true if the trivia questions in allQuestions appear in order
from least
  *   difficulty to greatest difficulty and returns false otherwise, as
described in part (b)
 */
public boolean isDifficultyIncreasing()
```

(c) A programmer would like to add a method called `gameLength` to inform players of how long it is expected to take, in minutes, to play a trivia game. The expected length of a game is based on the expected time to answer each question in the game.

Write a description of how you would change the `TriviaQuestion` and `TriviaGame` classes in order to support this modification.

Make sure to include the following in your response.

- Write the method header for the `gameLength` method.
- Identify any new or modified instance variables, constructors, or methods aside from the `gameLength` method. **Do not write the program code for this change.**
- Describe, for each new or revised variable, constructor, or method, how it would change or be implemented, including visibility and type. You do not need to describe the implementation of the `gameLength` method. **Do not write the program code for this change.**

67. Consider the following class.

```
public class WindTurbine
{
     private double efficiencyRating;
     public WindTurbine()
     {
         efficiencyRating = 0.0;
     }
     public WindTurbine(double e)
     {
         efficiencyRating = e;
     }
}
```

Which of the following code segments, when placed in a method in a class other than `WindTurbine`, will construct a `WindTurbine` object `wt` with an `efficiencyRating` of `0.25` ?

Unit2_objectsPartAJLC9_18_2023_Questions.pdf

(A)
```
WindTurbine wt = new WindTurbine(0.25);
```

(B)
```
WindTurbine wt = 0.25;
```

(C)
```
WindTurbine wt = new WindTurbine();
wt = 0.25;
```

(D)
```
WindTurbine wt = new WindTurbine();
wt.efficiencyRating = 0.25;
```

(E)
```
new WindTurbine wt = 0.25;
```