COMPUTER SCIENCE A
SECTION II
Time—1 hour and 45 minutes
Number of questions—4
Percent of total score—50

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.

1. 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.

```
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.
}
```

Part (a) begins on page 4.

(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()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (b) begins on page 6.

(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)
```

Part (c) begins on page 8.

(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.

YES NO

3. 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.

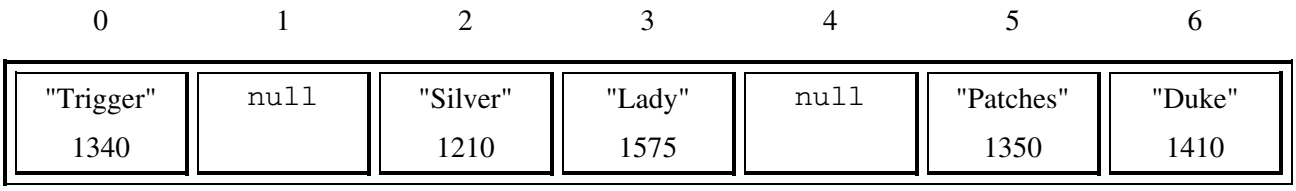
The declaration of the `HorseBarn` class is shown below. You will write two unrelated methods of the `HorseBarn` class.

```
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.
}
```

Part (a) begins on page 14.

(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()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

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)
```

Part (b) begins on page 16.

(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()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `consolidate` below.

```
/** 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()
```

4. A grayscale image is represented by a 2-dimensional rectangular array of pixels (picture elements). A pixel is an integer value that represents a shade of gray. In this question, pixel values can be in the range from 0 through 255, inclusive. A black pixel is represented by 0, and a white pixel is represented by 255.

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

```
public class GrayImage
{
  public static final int BLACK = 0;
  public static final int WHITE = 255;


  /** The 2-dimensional representation of this image. Guaranteed not to be null.
   *  All values in the array are within the range [BLACK, WHITE], inclusive.
   */
  private int[][] pixelValues;


  /** @return the total number of white pixels in this image.
   *  Postcondition: this image has not been changed.
   */
  public int countWhitePixels()
  {  /* to be implemented in part (a) */  }


  /** Processes this image in row-major order and decreases the value of each pixel at
   *  position (row, col) by the value of the pixel at position (row + 2, col + 2) if it exists.
   *  Resulting values that would be less than BLACK are replaced by BLACK.
   *  Pixels for which there is no pixel at position (row + 2, col + 2) are unchanged.
   */
  public void processImage()
  {  /* to be implemented in part (b) */  }
}
```

   (a) Write the method `countWhitePixels` that returns the number of pixels in the image that contain the value `WHITE`. For example, assume that `pixelValues` contains the following image.

	0	1	2	3	4
0	**255**	184	178	84	129
1	84	**255**	**255**	130	84
2	78	**255**	0	0	78
3	84	130	**255**	130	84

   A call to `countWhitePixels` method would return 5 because there are 5 entries (shown in boldface) that have the value `WHITE`.

Complete method `countWhitePixels` below.

```
/** @return the total number of white pixels in this image.
 *  Postcondition: this image has not been changed.
 */
public int countWhitePixels()
```

Part (b) begins on page 20.

(b) Write the method `processImage` that modifies the image by changing the values in the instance variable `pixelValues` according to the following description. The pixels in the image are processed one at a time in row-major order. Row-major order processes the first row in the array from left to right and then processes the second row from left to right, continuing until all rows are processed from left to right. The first index of `pixelValues` represents the row number, and the second index represents the column number.

The pixel value at position (row, col) is decreased by the value at position (row + 2, col + 2) if such a position exists. If the result of the subtraction is less than the value `BLACK`, the pixel is assigned the value of `BLACK`. The values of the pixels for which there is no pixel at position (row + 2, col + 2) remain unchanged. You may assume that all the original values in the array are within the range [`BLACK`, `WHITE`], inclusive.

The following diagram shows the contents of the instance variable `pixelValues` before and after a call to `processImage`. The values shown in boldface represent the pixels that could be modified in a grayscale image with 4 rows and 5 columns.

Before Call to `processImage`

	0	1	2	3	4
0	**221**	**184**	**178**	84	135
1	**84**	**255**	**255**	130	84
2	78	255	0	0	78
3	84	130	255	130	84

After Call to `processImage`

	0	1	2	3	4
0	**221**	**184**	**100**	84	135
1	**0**	**125**	**171**	130	84
2	78	255	0	0	78
3	84	130	255	130	84

Information repeated from the beginning of the question

```
public class GrayImage

public static final int BLACK = 0
public static final int WHITE = 255
private int[][] pixelValues
public int countWhitePixels()
public void processImage()
```

WRITE YOUR SOLUTION ON THE NEXT PAGE.

Complete method `processImage` below.

```
/** Processes this image in row-major order and decreases the value of each pixel at
 *  position (row, col) by the value of the pixel at position (row + 2, col + 2) if it exists.
 *  Resulting values that would be less than BLACK are replaced by BLACK.
 *  Pixels for which there is no pixel at position (row + 2, col + 2) are unchanged.
 */
public void processImage()
```

STOP

END OF EXAM