Computer Science A

AP Review Book

2022

Note 0: the 2009 and 2004 multiple-choice sections were copied and scanned from actual paper copies of the tests, then manually copied-and-pasted into this book. In some cases, parts were moved closer together to save paper. I apologize for any sloppiness this caused.

Note 1: The tests include questions from the GridWorld case study. This used to be a part of the class which was provided by the College Board and which we studied in class. It is no longer part of the course.

Computer Science A – AP Review Book

Table of Contents

AP Computer Science Reference Sheet

Accessible methods from the Java library that may be included in the exam

Class Constructors and Methods	Explanation
String Class	
`String(String str)`	Constructs a new `String` object that represents the same sequence of characters as `str`
`int length()`	Returns the number of characters in a `String` object
`String substring(int from, int to)`	Returns the substring beginning at index `from` and ending at index `to - 1`
`String substring(int from)`	Returns `substring(from, length())`
`int indexOf(String str)`	Returns the index of the first occurrence of `str`; returns `-1` if not found
`boolean equals(String other)`	Returns `true` if `this` is equal to `other`; returns `false` otherwise
`int compareTo(String other)`	Returns a value `<0` if `this` is less than `other`; returns zero if `this` is equal to `other`; returns a value `>0` if `this` is greater than `other`
Integer Class	
`Integer(int value)`	Constructs a new `Integer` object that represents the specified `int` value
`Integer.MIN_VALUE`	The minimum value represented by an `int` or `Integer`
`Integer.MAX_VALUE`	The maximum value represented by an `int` or `Integer`
`int intValue()`	Returns the value of this `Integer` as an `int`
Double Class	
`Double(double value)`	Constructs a new `Double` object that represents the specified `double` value
`double doubleValue()`	Returns the value of this `Double` as a `double`
Math Class	
`static int abs(int x)`	Returns the absolute value of an `int` value
`static double abs(double x)`	Returns the absolute value of a `double` value
`static double pow(double base, double exponent)`	Returns the value of the first parameter raised to the power of the second parameter
`static double sqrt(double x)`	Returns the positive square root of a `double` value
`static double random()`	Returns a `double` value greater than or equal to `0.0` and less than `1.0`
ArrayList Class	
`int size()`	Returns the number of elements in the list
`boolean add(E obj)`	Appends `obj` to end of list; returns `true`
`void add(int index, E obj)`	Inserts `obj` at position `index` (`0 <= index <= size`), moving elements at position `index` and higher to the right (adds 1 to their indices) and adds 1 to size
`E get(int index)`	Returns the element at position `index` in the list
`E set(int index, E obj)`	Replaces the element at position `index` with `obj`; returns the element formerly at position `index`
`E remove(int index)`	Removes element from position `index`, moving elements at position `index + 1` and higher to the left (subtracts 1 from their indices) and subtracts 1 from size; returns the element formerly at position `index`
Object Class	
`boolean equals(Object other)`	
`String toString()`	

Multiple-Choice Sections

2009 Multiple Choice

COMPUTER SCIENCE A
SECTION I
Time—1 hour and 15 minutes
Number of questions—40
Percent of total grade—50

Directions: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratch work. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Assume that declarations of variables and methods appear within the context of an enclosing class.
- Assume that method calls that are not prefixed with an object or class name and are not shown within a complete class definition appear within the context of an enclosing class.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null`.

1. Consider the following code segment.

```
int value = 15;
while (value < 28)
{
  System.out.println(value);
  value++;
}
```

What are the first and last numbers output by the code segment?

	First	Last
(A)	15	27
(B)	15	28
(C)	16	27
(D)	16	28
(E)	16	29

2. A teacher put three bonus questions on a test and awarded 5 extra points to anyone who answered all three bonus questions correctly and no extra points otherwise. Assume that the `boolean` variables `bonusOne`, `bonusTwo`, and `bonusThree` indicate whether a student has answered the particular question correctly. Each variable was assigned `true` if the answer was correct and `false` if the answer was incorrect.

   Which of the following code segments will properly update the variable `grade` based on a student's performance on the bonus questions?

   I.
   ```
   if (bonusOne && bonusTwo && bonusThree)
     grade += 5;
   ```

   II.
   ```
   if (bonusOne || bonusTwo || bonusThree)
     grade += 5;
   ```

   III.
   ```
   if (bonusOne)
     grade += 5;
   if (bonusTwo)
     grade += 5;
   if (bonusThree)
     grade += 5;
   ```

   (A) I only

   (B) II only

   (C) III only

   (D) I and III

   (E) II and III

3. Assume that an array of integer values has been declared as follows and has been initialized.

   ```
   int[] arr = new int[10];
   ```

   Which of the following code segments correctly interchanges the value of `arr[0]` and `arr[5]` ?

   (A)
   ```
   arr[0] = 5;
   arr[5] = 0;
   ```

   (B)
   ```
   arr[0] = arr[5];
   arr[5] = arr[0];
   ```

   (C)
   ```
   int k = arr[5];
   arr[0] = arr[5];
   arr[5] = k;
   ```

   (D)
   ```
   int k = arr[0];
   arr[0] = arr[5];
   arr[5] = k;
   ```

   (E)
   ```
   int k = arr[5];
   arr[5] = arr[0];
   arr[0] = arr[5];
   ```

4. Consider the following code segment.

```
ArrayList<String> items = new ArrayList<String>();
items.add("A");
items.add("B");
items.add("C");
items.add(0, "D");
items.remove(3);
items.add(0, "E");
System.out.println(items);
```

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

(A) [A, B, C, E]

(B) [A, B, D, E]

(C) [E, D, A, B]

(D) [E, D, A, C]

(E) [E, D, C, B]

5. When designing a class hierarchy, which of the following should be true of a superclass?

(A) A superclass should contain the data and functionality that are common to all subclasses that inherit from the superclass.

(B) A superclass should be the largest, most complex class from which all other subclasses are derived.

(C) A superclass should contain the data and functionality that are only required for the most complex class.

(D) A superclass should have public data in order to provide access for the entire class hierarchy.

(E) A superclass should contain the most specific details of the class hierarchy.

Questions 6-7 refer to the following code segment.

```
int k = a random number such that 1 ≤ k ≤ n ;

for (int p = 2; p <= k; p++)
  for (int r = 1; r < k; r++)
    System.out.println("Hello");
```

6. What is the minimum number of times that `Hello` will be printed?

(A) 0

(B) 1

(C) 2

(D) $n - 1$

(E) $n - 2$

7. What is the maximum number of times that `Hello` will be printed?

   (A) 2

   (B) $n - 1$

   (C) $n - 2$

   (D) $(n - 1)^2$

   (E) n^2

8. Consider the following instance variable and incomplete method. The method `calcTotal` is intended to return the sum of all values in `vals`.

```
private int[] vals;

public int calcTotal()
{
  int total = 0;

  /* missing code */

  return total;
}
```

Which of the code segments shown below can be used to replace `/* missing code */` so that `calcTotal` will work as intended?

I.
```
for (int pos = 0; pos < vals.length; pos++)
{
  total += vals[pos];
}
```

II.
```
for (int pos = vals.length; pos > 0; pos--)
{
  total += vals[pos];
}
```

III.
```
int pos = 0;
while (pos < vals.length)
{
  total += vals[pos];
  pos++;
}
```

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

9. Consider the following code segment.

```
String str = "abcdef";
for (int rep = 0; rep < str.length() - 1; rep++)
{
  System.out.print(str.substring(rep, rep + 2));
}
```

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

(A) `abcdef`

(B) `aabbccddeeff`

(C) `abbccddeef`

(D) `abcbcdcdedef`

(E) Nothing is printed because an `IndexOutOfBoundsException` is thrown.

10. Consider the following method.

```
public void numberCheck(int maxNum)
{
  int typeA = 0;
  int typeB = 0;
  int typeC = 0;

  for (int k = 1; k <= maxNum; k++)
  {
    if (k % 2 == 0 && k % 5 == 0)
      typeA++;
    if (k % 2 == 0)
      typeB++;
    if (k % 5 == 0)
      typeC++;
  }

  System.out.println(typeA + " " + typeB + " " + typeC);
}
```

What is printed as a result of the call `numberCheck(50)` ?

(A) `5 20 5`

(B) `5 20 10`

(C) `5 25 5`

(D) `5 25 10`

(E) `30 25 10`

11. Consider the following method that is intended to modify its parameter `nameList` by replacing all occurrences of `name` with `newValue`.

```
public void replace(ArrayList<String> nameList,
                    String name, String newValue)
{
  for (int j = 0; j < nameList.size(); j++)
  {
    if ( /* expression */ )
    {
      nameList.set(j, newValue);
    }
  }
}
```

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

(A) `nameList.get(j).equals(name)`

(B) `nameList.get(j) == name`

(C) `nameList.remove(j)`

(D) `nameList[j] == name`

(E) `nameList[j].equals(name)`

12. Consider the following incomplete method.

```
public int someProcess(int n)
{
  /* body of someProcess */
}
```

The following table shows several examples of input values and the results that should be produced by calling `someProcess`.

Input Value `n`	Value Returned by `someProcess(n)`
3	30
6	60
7	7
8	80
9	90
11	11
12	120
14	14
16	160

Which of the following code segments could be used to replace `/* body of someProcess */` so that the method will produce the results shown in the table?

I.
```
if ((n % 3 == 0) && (n % 4 == 0))
  return n * 10;
else
  return n;
```

II.
```
if ((n % 3 == 0) || (n % 4 == 0))
  return n * 10;

return n;
```

III.
```
if (n % 3 == 0)
  if (n % 4 == 0)
    return n * 10;

return n;
```

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

13. Consider the following method.

```
// precondition: x >= 0
public void mystery(int x)
{
  if ((x / 10) != 0)
  {
    mystery(x / 10);
  }

  System.out.print(x % 10);
}
```

Which of the following is printed as a result of the call `mystery(123456)` ?

(A) `16`

(B) `56`

(C) `123456`

(D) `654321`

(E) Many digits are printed due to infinite recursion.

14. Consider the following instance variables and incomplete method that are part of a class that represents an item. The variables `years` and `months` are used to represent the age of the item, and the value for `months` is always between 0 and 11, inclusive. Method `updateAge` is used to update these variables based on the parameter `extraMonths` that represents the number of months to be added to the age.

```
private int years;
private int months;  // 0 <= months <= 11

// precondition: extraMonths >= 0
public void updateAge(int extraMonths)
{
  /* body of updateAge */
}
```

Which of the following code segments could be used to replace `/* body of updateAge */` so that the method will work as intended?

I.
```
int yrs = extraMonths % 12;
int mos = extraMonths / 12;
years = years + yrs;
months = months + mos;
```

II.
```
int totalMonths = years * 12 + months + extraMonths;
years = totalMonths / 12;
months = totalMonths % 12;
```

III.
```
int totalMonths = months + extraMonths;
years = years + totalMonths / 12;
months = totalMonths % 12;
```

(A) I only

(B) II only

(C) III only

(D) II and III only

(E) I, II, and III

15. Consider the following method.

```
public String inRangeMessage(int value)
{
  if (value < 0 || value > 100)
    return "Not in range";
  else
    return "In range";
}
```

Consider the following code segments that could be used to replace the body of `inRangeMessage`.

I.
```
if (value < 0)
{
  if (value > 100)
    return "Not in range";
  else
    return "In range";
}
else
  return "In range";
```

II.
```
if (value < 0)
  return "Not in range";
else if (value > 100)
  return "Not in range";
else
  return "In range";
```

III.
```
if (value >= 0)
  return "In range";
else if (value <= 100)
  return "In range";
else
  return "Not in range";
```

Which of the replacements will have the same behavior as the original version of `inRangeMessage` ?

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

16. 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);
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

17. The following incomplete method is intended to sort its array parameter `arr` in increasing order.

```
// postcondition: arr is sorted in increasing order
public static void sortArray(int[] arr)
{
  int j, k;

  for (j = arr.length - 1; j > 0; j--)
  {
    int pos = j;

    for ( /* missing code */ )
    {
      if (arr[k] > arr[pos])
      {
        pos = k;
      }
    }
    swap(arr, j, pos);
  }
}
```

Assume that `swap(arr, j, pos)` exchanges the values of `arr[j]` and `arr[pos]`. Which of the following could be used to replace `/* missing code */` so that executing the code segment sorts the values in array `arr`?

(A) `k = j - 1; k > 0; k--`

(B) `k = j - 1; k >= 0; k--`

(C) `k = 1; k < arr.length; k++`

(D) `k = 1; k > arr.length; k++`

(E) `k = 0; k <= arr.length; k++`

18. Assume that `x` and `y` are `boolean` variables and have been properly initialized.

```
(x && y) || !(x && y)
```

The result of evaluating the expression above is best described as

(A) always `true`

(B) always `false`

(C) `true` only when `x` is `true` and `y` is `true`

(D) `true` only when `x` and `y` have the same value

(E) `true` only when `x` and `y` have different values

19. Assume that the following variable declarations have been made.

```
double d = Math.random();
double r;
```

Which of the following assigns a value to `r` from the uniform distribution over the range $0.5 \le r < 5.5$?

(A) `r = d + 0.5;`

(B) `r = d + 0.5 * 5.0;`

(C) `r = d * 5.0;`

(D) `r = d * 5.0 + 0.5;`

(E) `r = d * 5.5;`

20. Consider the following instance variables and method that appear in a class representing student information.

```
private int assignmentsCompleted;
private double testAverage;

public boolean isPassing()
{ /* implementation not shown */ }
```

A student can pass a programming course if at least one of the following conditions is met.

- The student has a test average that is greater than or equal to 90.
- The student has a test average that is greater than or equal to 75 and has at least 4 completed assignments.

Consider the following proposed implementations of the `isPassing` method.

I.
```
if (testAverage >= 90)
  return true;
if (testAverage >= 75 && assignmentsCompleted >= 4)
  return true;
return false;
```

II.
```
boolean pass = false;
if (testAverage >= 90)
  pass = true;
if (testAverage >= 75 && assignmentsCompleted >= 4)
  pass = true;
return pass;
```

III.
```
return (testAverage >= 90) ||
       (testAverage >= 75 && assignmentsCompleted >= 4);
```

Which of the implementations will correctly implement method `isPassing` ?

(A) I only

(B) II only

(C) I and III only

(D) II and III only

(E) I, II, and III

Questions 21-25 refer to the code from the GridWorld case study. A copy of the code is provided in the Appendix.

21. Consider the following code segment.

```
Location loc1 = new Location(3, 3);
Location loc2 = new Location(3, 2);

if (loc1.equals(loc2.getAdjacentLocation(Location.EAST)))
  System.out.print("aaa");

if (loc1.getRow() == loc2.getRow())
  System.out.print("XXX");

if (loc1.getDirectionToward(loc2) == Location.EAST)
  System.out.print("555");
```

What will be printed as a result of executing the code segment?

(A) `aaaXXX555`

(B) `aaaXXX`

(C) `XXX555`

(D) `555`

(E) `aaa`

22. A `RightTurningBug` behaves like a `Bug`, except that when it turns, it turns 90 degrees to the right. The declaration for the `RightTurningBug` class is as follows.

```
public class RightTurningBug extends Bug
{
  public void turn()
  {
    /* missing implementation */
  }
}
```

Consider the following suggested replacements for `/* missing implementation */`.

I.
```
int desiredDirection = (getDirection() + Location.RIGHT)
                       % Location.FULL_CIRCLE;

while (getDirection() != desiredDirection)
{
  super.turn();
}
```

II.
```
super.turn();
super.turn();
```

III.
```
setDirection(getDirection() + Location.RIGHT);
```

Which of the replacements will produce the desired behavior?

(A) I only

(B) II only

(C) I and II only

(D) I and III only

(E) I, II, and III

23. Consider the following declarations.

```
Actor a = new Actor();
Bug b = new Bug();
Rock r = new Rock();
Critter c = new Critter();
```

Consider the following lines of code.

```
Line 1:  int dir1 = c.getDirection();
Line 2:  int dir2 = a.getDirection();
Line 3:  int dir3 = b.getDirection();
Line 4:  ArrayList<Location> rLoc = r.getMoveLocations();
Line 5:  ArrayList<Location> cLoc = c.getMoveLocations();
```

Which of the lines of code above will cause a compile time error?

(A) Line 1 only

(B) Lines 2 and 3 only

(C) Line 4 only

(D) Line 5 only

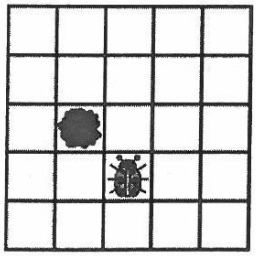
(E) Lines 4 and 5 only

24. Consider the following `TestBug` class declaration.

```
public class TestBug extends Bug
{
  public void act()
  {
    if (canMove())
    {
      move();
      if (canMove())
        move();
    }
    else
    {
      setDirection(getDirection() + Location.HALF_CIRCLE);
    }
  }
}
```

The following code segment will produce a grid that has a `Rock` object and a `TestBug` object placed as shown.

```
Grid<Actor> g = new BoundedGrid<Actor>(5, 5);
Rock r = new Rock();
r.putSelfInGrid(g, new Location(2, 1));
Bug t = new TestBug();
t.putSelfInGrid(g, new Location(3, 2));
```



Which of the following best describes what the `TestBug` object `t` does as a result of calling `t.act()` ?

(A) Moves forward two locations and remains facing current direction

(B) Moves forward two locations and turns 180 degrees

(C) Moves forward one location and remains facing current direction

(D) Moves forward one location and turns 180 degrees

(E) Stays in the same location and turns 180 degrees

25. A `DancingCritter` is a `Critter` that moves in the following manner. The `DancingCritter` makes a left turn if at least one of its neighbors is another `DancingCritter`. It then moves like a `Critter`. If none of its neighbors are `DancingCritter` objects, it moves like a `Critter` without making a left turn. In all other respects, a `DancingCritter` acts like a `Critter` by eating neighbors that are not rocks or critters. Consider the following implementations.

I.
```
public class DancingCritter extends Critter
{
  public ArrayList<Actor> getActors()
  {
    ArrayList<Actor> actors = new ArrayList<Actor>();
    for (Actor a : getGrid().getNeighbors(getLocation()))
    {
      if (a instanceof DancingCritter)
        actors.add(a);
    }
    return actors;
  }

  public void processActors(ArrayList<Actor> actors)
  {
    if (actors.size() > 0)
    {
      setDirection(getDirection() + Location.LEFT);
    }
    super.processActors(actors);
  }
}
```

II.
```
public class DancingCritter extends Critter
{
  public void processActors(ArrayList<Actor> actors)
  {
    boolean turning = false;
    for (Actor a : actors)
    {
      if (a instanceof DancingCritter)
        turning = true;
    }
    if (turning)
    {
      setDirection(getDirection() + Location.LEFT);
    }
  }
}
```

III.
```
public class DancingCritter extends Critter
{
  public void makeMove(Location loc)
  {
    boolean turning = false;
    for (Actor a : getGrid().getNeighbors(getLocation()))
    {
      if (a instanceof DancingCritter)
        turning = true;
    }
    if (turning)
    {
      setDirection(getDirection() + Location.LEFT);
    }
    super.makeMove(loc);
  }
}
```

Which of the proposed implementations will correctly implement the `DancingCritter` class?

(A) I only

(B) II only

(C) III only

(D) I and II only

(E) I, II, and III

26. Consider the following code segment.

```
int k = 0;
while (k < 10)
{
  System.out.print((k % 3) + "  ");
  if ((k % 3) == 0)
    k = k + 2;
  else
    k++;
}
```

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

(A) 0 2 1 0 2

(B) 0 2 0 2 0 2

(C) 0 2 1 0 2 1 0

(D) 0 2 0 2 0 2 0

(E) 0 1 2 1 2 1 2

27. Consider the following method. Method `allEven` is intended to return `true` if all elements in array `arr` are even numbers; otherwise, it should return `false`.

```
public boolean allEven(int[] arr)
{
  boolean isEven = /* expression */ ;

  for (int k = 0; k < arr.length; k++)
  {
    /* loop body */
  }

  return isEven;
}
```

Which of the following replacements for `/* expression */` and `/* loop body */` should be used so that method `allEven` will work as intended?

	/* expression */	/* loop body */
(A)	`false`	`if ((arr[k] % 2) == 0)` `  isEven = true;`

(B)	`false`	`if ((arr[k] % 2) != 0)`
		`  isEven = false;`
		`else`
		`  isEven = true;`

(C)	`true`	`if ((arr[k] % 2) != 0)` `  isEven = false;`

(D)	`true`	`if ((arr[k] % 2) != 0)`
		`  isEven = false;`
		`else`
		`  isEven = true;`

(E)	`true`	`if ((arr[k] % 2) == 0)`
		`  isEven = false;`
		`else`
		`  isEven = true;`

28. Consider the following code segment.

```
int x = /* some integer value */ ;
int y = /* some integer value */ ;

boolean result = (x < y);

result = ( (x >= y) && !result );
```

Which of the following best describes the conditions under which the value of `result` will be `true` after the code segment is executed?

(A) Only when `x < y`

(B) Only when `x >= y`

(C) Only when `x` and `y` are equal

(D) The value will always be `true`.

(E) The value will never be `true`.

29. Consider the following code segment.

```
for (int outer = 0; outer < n; outer++)
{
  for (int inner = 0; inner <= outer; inner++)
  {
    System.out.print(outer + " ");
  }
}
```

If `n` has been declared as an integer with the value 4, what is printed as a result of executing the code segment?

(A) `0 1 2 3`

(B) `0 0 1 0 1 2`

(C) `0 1 2 2 3 3 3`

(D) `0 1 1 2 2 2 3 3 3 3`

(E) `0 0 1 0 1 2 0 1 2 3`

30. Consider the following class declarations.

```
public class Base
{
  private int myVal;

  public Base()
  {  myVal = 0;  }

  public Base(int x)
  {  myVal = x;  }
}


public class Sub extends Base
{
  public Sub()
  {  super(0);  }
}
```

Which of the following statements will NOT compile?

(A) `Base b1 = new Base();`

(B) `Base b2 = new Base(5);`

(C) `Base s1 = new Sub();`

(D) `Sub s2 = new Sub();`

(E) `Sub s3 = new Sub(5);`

31. Assume that `a` and `b` are variables of type `int`. The expression

```
!(a < b) && !(a > b)
```

is equivalent to which of the following?

(A) `true`

(B) `false`

(C) `a == b`

(D) `a != b`

(E) `!(a < b) && (a > b)`

32. Consider the following code segment.

```
int a = 24;
int b = 30;
while (b != 0)
{
  int r = a % b;
  a = b;
  b = r;
}

System.out.println(a);
```

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

(A) 0

(B) 6

(C) 12

(D) 24

(E) 30

33. Consider the following method.

```
public int sol(int lim)
{
  int s = 0;

  for (int outer = 1; outer <= lim; outer++)
  {
    for (int inner = outer; inner <= lim; inner++)
    {
      s++;
    }
  }

  return s;
}
```

What value is returned as a result of the call `sol(10)` ?

(A) 20

(B) 45

(C) 55

(D) 100

(E) 385

34. Consider the following incomplete method. Method `findNext` is intended to return the index of the first occurrence of the value `val` beyond the position `start` in array `arr`.

```
// returns index of first occurrence of val in arr
// after position start;
// returns arr.length if val is not found
public int findNext(int[] arr, int val, int start)
{
  int pos = start + 1;

  while ( /* condition */ )
    pos++;

  return pos;
}
```

For example, consider the following code segment.

```
int[] arr = {11, 22, 100, 33, 100, 11, 44, 100};

System.out.println(findNext(arr, 100, 2));
```

The execution of the code segment should result in the value 4 being printed.

Which of the following expressions could be used to replace `/* condition */` so that `findNext` will work as intended?

(A) `(pos < arr.length) && (arr[pos] != val)`

(B) `(arr[pos] != val) && (pos < arr.length)`

(C) `(pos < arr.length) || (arr[pos] != val)`

(D) `(arr[pos] == val) && (pos < arr.length)`

(E) `(pos < arr.length) || (arr[pos] == val)`

35. Consider the following code segments.

I.
```
int k = 1;
while (k < 20)
{
  if (k % 3 == 1)
    System.out.print( k + "  ");

  k = k + 3;
}
```

II.
```
for (int k = 1; k < 20; k++)
{
  if (k % 3 == 1)
    System.out.print( k + "  ");
}
```

III.
```
for (int k = 1; k < 20; k = k + 3)
{
  System.out.print( k + "  ");
}
```

Which of the code segments above will produce the following output?

```
1  4  7  10  13  16  19
```

(A) I only

(B) II only

(C) I and II only

(D) II and III only

(E) I, II, and III

36. Consider the following two methods that appear within a single class.

```
public void changeIt(int[] list, int num)
{
  list = new int[5];
  num = 0;

  for (int x = 0; x < list.length; x++)
    list[x] = 0;
}


public void start()
{
  int[] nums = {1, 2, 3, 4, 5};
  int value = 6;

  changeIt(nums, value);

  for (int k = 0; k < nums.length; k++)
    System.out.print(nums[k] + " ");

  System.out.print(value);
}
```

What is printed as a result of the call `start()` ?

(A) `0 0 0 0 0 0`

(B) `0 0 0 0 0 6`

(C) `1 2 3 4 5 6`

(D) `1 2 3 4 5 0`

(E) `changeIt` will throw an exception.

37. Consider the following declaration of the class `NumSequence`, which has a constructor that is intended to initialize the instance variable `seq` to an `ArrayList` of `numberOfValues` random floating-point values in the range [0.0, 1.0).

```
public class NumSequence
{
  private ArrayList<Double> seq;

  // precondition:  numberOfValues > 0
  // postcondition: seq has been initialized to an ArrayList of
  //                length numberOfValues; each element of seq
  //                contains a random Double in the range [0.0, 1.0)
  public NumSequence(int numberOfValues)
  {
    /* missing code */
  }
}
```

Which of the following code segments could be used to replace `/* missing code */` so that the constructor will work as intended?

I.
```
ArrayList<Double> seq = new ArrayList<Double>();
for (int k = 0; k < numberOfValues; k++)
  seq.add(new Double(Math.random()));
```

II.
```
seq = new ArrayList<Double>();
for (int k = 0; k < numberOfValues; k++)
  seq.add(new Double(Math.random()));
```

III.
```
ArrayList<Double> temp = new ArrayList<Double>();
for (int k = 0; k < numberOfValues; k++)
  temp.add(new Double(Math.random()));

seq = temp;
```

(A) II only

(B) III only

(C) I and II

(D) I and III

(E) II and III

38. Consider the following code segment.

```
double a = 1.1;
double b = 1.2;

if ((a + b) * (a - b) != (a * a) - (b * b))
{
  System.out.println("Mathematical error!");
}
```

Which of the following best describes why the phrase `"Mathematical error!"` would be printed? (Remember that mathematically $(a + b) * (a - b) = a^2 - b^2$.)

(A) Precedence rules make the `if` condition true.

(B) Associativity rules make the `if` condition true.

(C) Roundoff error makes the `if` condition true.

(D) Overflow makes the `if` condition true.

(E) A compiler bug or hardware error has occurred.

39. Consider the following recursive method.

```
public static String recur(int val)
{
  String dig = "" + (val % 3);

  if (val / 3 > 0)
    return dig + recur(val / 3);

  return dig;
}
```

What is printed as a result of executing the following statement?

```
System.out.println(recur(32));
```

(A) 20

(B) 102

(C) 210

(D) 1020

(E) 2101

40. Consider the following method.

```
public String goAgain(String str, int index)
{
  if (index >= str.length())
    return str;

  return str + goAgain(str.substring(index), index + 1);
}
```

What is printed as a result of executing the following statement?

```
System.out.println(goAgain("today", 1));
```

(A) today

(B) todayto

(C) todayoday

(D) todayodayay

(E) todayodaydayayy

2004 Multiple Choice

1. Assume that `a`, `b`, and `c` are variables of type `int`. Consider the following three conditions.

   I. `(a == b) && (a == c) && (b == c)`

   II. `(a == b) || (a == c) || (b == c)`

   III. `((a - b) * (a - c) * (b - c)) == 0`

   Assume that subtraction and multiplication never overflow. Which of the conditions above is (are) always true if at least two of `a`, `b`, and `c` are equal?

   (A) I only

   (B) II only

   (C) III only

   (D) I and II

   (E) II and III

2. Consider the following two data structures for storing several million words.

   I. An array of words, not in any particular order

   II. An array of words, sorted in alphabetical order

   Which of the following statements most accurately describes the time needed for operations on these data structures?

   (A) Inserting a word is faster in II than in I.

   (B) Finding a given word is faster in I than in II.

   (C) Finding a given word is faster in II than in I.

   (D) Finding the longest word is faster in II than in I.

   (E) Finding the first word in alphabetical order is faster in I than in II.

3. Which of the following is a reason to use an `ArrayList` instead of an array?

   (A) An `ArrayList` allows faster access to its *k*th item than an array does.

   (B) An `ArrayList` always uses less memory than an array does.

   (C) An `ArrayList` can store objects and an array can only store primitive types.

   (D) An `ArrayList` resizes itself as necessary when items are added, but an array does not.

   (E) An `ArrayList` provides access to the number of items it stores, but an array does not.

4. Consider the following method, `between`, which is intended to return `true` if `x` is between `lower` and `upper`, inclusive, and `false` otherwise.

```
// precondition:  lower <= upper
// postcondition: returns true if x is between lower and upper,
//                inclusive; otherwise, returns false
public boolean between(int x, int lower, int upper)
{
  /* missing code */
}
```

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

(A) `return (x <= lower) && (x >= upper);`

(B) `return (x <= lower) || (x >= upper);`

(C) `return lower <= x <= upper;`

(D) `return (x >= lower) && (x <= upper);`

(E) `return (x >= lower) || (x <= upper);`

5. Consider the following class definitions.

```
public class Data
{
  private int x;

  public void setX(int n)
  {
    x = n;
  }

  // ... other methods not shown

}

public class EnhancedData extends Data
{
  private int y;

  public void setY(int n)
  {
    y = n;
  }

  // ... other methods not shown

}
```

Assume that the following declaration appears in a client program.

```
EnhancedData item = new EnhancedData();
```

Which of the following statements would be valid?

I. `item.y = 16;`

II. `item.setY(16);`

III. `item.setX(25);`

(A) I only

(B) II only

(C) I and II only

(D) II and III only

(E) I, II, and III

Questions 6-7 refer to the following data field and method.

```
private int[] arr;

// precondition: arr.length > 0
public void mystery()
{
  int s1 = 0;
  int s2 = 0;

  for (int k = 0; k < arr.length; k++)
  {
    int num = arr[k];

    if ((num > 0) && (num % 2 == 0))
      s1 += num;
    else if (num < 0)
      s2 += num;
  }

  System.out.println(s1);
  System.out.println(s2);
}
```

6. Which of the following best describes the value of `s1` output by the method `mystery` ?

   (A) The sum of all positive values in `arr`

   (B) The sum of all positive even values in `arr`

   (C) The sum of all positive odd values in `arr`

   (D) The sum of all values greater than 2 in `arr`

   (E) The sum of all values less than 2 in `arr`

7. Which of the following best describes the value of `s2` output by the method `mystery` ?

   (A) The sum of all positive values in `arr`

   (B) The sum of all positive even values in `arr`

   (C) The sum of all negative values in `arr`

   (D) The sum of all negative even values in `arr`

   (E) The sum of all negative odd values in `arr`

8. When designing classes, which of the following would be the best reason to use inheritance?

   (A) Inheritance allows you to write applications that require fewer base and super classes.

   (B) Inheritance allows the creation of a subclass that can use the methods of its superclass without rewriting the code for those methods.

   (C) Inheritance allows for data encapsulation, while noninherited classes do not allow for data encapsulation.

   (D) Inheritance reduces the number of polymorphic structures encapsulated in applications.

   (E) Inheritance guarantees that the applications will compile and execute much more quickly.

9. Consider the following method.

```
public void conditionalTest(int a, int b)
{
  if ((a > 0) && (b > 0))
  {
    if (a > b)
      System.out.println("A");
    else
      System.out.println("B");
  }
  else if ((b < 0) || (a < 0))
    System.out.println("C");
  else
    System.out.println("D");
}
```

What is printed as a result of the call `conditionalTest(3, -2)`?

(A) A

(B) B

(C) C

(D) D

(E) Nothing is printed.

10. Consider the following class declaration.

```
public class IntCell
{
  private int myStoredValue;

  // constructor not shown

  public int getValue()
  {
    return myStoredValue;
  }

  public String toString()
  {
    return "" + myStoredValue;
  }
}
```

Assume that the following declaration appears in a client class.

```
IntCell m = new IntCell();
```

Which of these statements can be used in the client class?

I. `System.out.println(m.getValue());`

II. `System.out.println(m.myStoredValue);`

III. `System.out.println(m);`

(A) I only

(B) II only

(C) III only

(D) I and II

(E) I and III

11. Consider the following static method.

```
public static int calculate(int x)
{
  x = x + x;
  x = x + x;
  x = x + x;

  return x;
}
```

Which of the following can be used to replace the body of `calculate` so that the modified version of `calculate` will return the same result as the original version for all `x` ?

(A) `return 3 + x;`

(B) `return 3 * x;`

(C) `return 4 * x;`

(D) `return 6 * x;`

(E) `return 8 * x;`

12. Consider the following code segment.

```
int k = 1;

while (k < 20)
{
  if ((k % 3) == 1)
    System.out.print(k + "  ");

  k++;
}
```

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

(A) `2  5  8  11  14  17`

(B) `3  6  9  12  15  18`

(C) `1  4  7  10  13  16  19`

(D) `1  3  5  7  9  11  13  15  17  19`

(E) `2  4  6  8  10  12  14  16  18  20`

13. Consider the following code segment. The code is intended to read nonnegative numbers and compute their product until a negative number is read; however, it does not work as intended. (Assume that the `readInt` method correctly reads the next number from the input stream.)

```
int k = 0;
int prod = 1;

while (k >= 0)
{
  System.out.print("enter a number: ");
  k = readInt(); // readInt reads the next number from input
  prod = prod * k;
}

System.out.println("product: " + prod);
```

Which of the following best describes the error in the program?

(A) The variable `prod` is incorrectly initialized.

(B) The while condition always evaluates to `false`.

(C) The while condition always evaluates to `true`.

(D) The negative number entered to signal no more input is included in the product.

(E) If the user enters a zero, the computation of the product will be terminated prematurely.

14. Consider the following declarations.

```
int valueOne, valueTwo;
```

Assume that `valueOne` and `valueTwo` have been initialized. Which of the following evaluates to `true` if `valueOne` and `valueTwo` contain the same value?

(A) `valueOne.equals((Object) valueTwo)`

(B) `valueOne == valueTwo`

(C) `valueOne.compareTo((Object) valueTwo) == 0`

(D) `valueOne.compareTo(valueTwo) == 0`

(E) `valueOne.equals(valueTwo)`

15. Consider the following method.

```
public int someCode(int a, int b, int c)
{
  if ((a < b) && (b < c))
    return a;
  if ((a >= b) && (b >= c))
    return b;
  if ((a == b) || (a == c) || (b == c))
    return c;
}
```

Which of the following best describes why this method does not compile?

(A) The reserved word `return` cannot be used in the body of an `if` statement.

(B) It is possible to reach the end of the method without returning a value.

(C) The `if` statements must have `else` parts when they contain `return` statements.

(D) Methods cannot have multiple `return` statements.

(E) The third `if` statement is not reachable.

16. Consider the following code segment.

```
int num1 = 0;
int num2 = 3;

while ((num2 != 0) && ((num1 / num2) >= 0))
{
  num1 = num1 + 2;
  num2 = num2 - 1;
}
```

What are the values of `num1` and `num2` after the while loop completes its execution?

(A) `num1 = 0, num2 = 3`

(B) `num1 = 8, num2 = -1`

(C) `num1 = 4, num2 = 1`

(D) `num1 = 6, num2 = 0`

(E) The loop will never complete its execution because a division by zero will generate an `ArithmeticException`.

17. A bear is an animal and a zoo contains many animals, including bears. Three classes `Animal`, `Bear`, and `Zoo` are declared to represent animal, bear, and zoo objects. Which of the following is the most appropriate set of declarations?

(A)
```
public class Animal extends Bear
{
  ...
}

public class Zoo
{
  private Animal[] myAnimals;
  ...
}
```

(B)
```
public class Bear extends Animal
{
  ...
}

public class Zoo
{
  private Animal[] myAnimals;
  ...
}
```

(C)
```
public class Animal extends Zoo
{
  private Bear myBear;
  ...
}
```

(D)
```
public class Bear extends Animal, Zoo
{
  ...
}
```

(E)
```
public class Bear extends Animal implements Zoo
{
  ...
}
```

18. Consider the static method `selectSort` shown below. Method `selectSort` is intended to sort an array into increasing order; however, it does not always work as intended.

```
// precondition:  numbers.length > 0
// postcondition: numbers is sorted in increasing order
public static void selectSort(int[] numbers)
{
  int temp;
Line 1:  for (int j = 0; j < numbers.length - 1; j++)
  {
Line 2:    int pos = 0;
Line 3:    for (int k = j + 1; k < numbers.length; k++)
    {
Line 4:      if (numbers[k] < numbers[pos])
      {
Line 5:        pos = k;
      }
    }
    temp = numbers[j];
    numbers[j] = numbers[pos];
    numbers[pos] = temp;
  }
}
```

Which of the following changes should be made so that `selectSort` will work as intended?

(A) Line 1 should be changed to
```
for (int j = 0; j < numbers.length - 2; j++)
```

(B) Line 2 should be changed to
```
int pos = j;
```

(C) Line 3 should be changed to
```
for (int k = 0; k < numbers.length; k++)
```

(D) Line 4 should be changed to
```
if (numbers[k] > numbers[pos])
```

(E) Line 5 should be changed to
```
k = pos;
```

19. Consider the following two static methods, where `f2` is intended to be the iterative version of `f1`.

```
public static int f1(int n)
{
  if (n < 0)
  {
    return 0;
  }
  else
  {
    return (f1(n - 1) + n * 10);
  }
}

public static int f2(int n)
{
  int answer = 0;

  while (n > 0)
  {
    answer = answer + n * 10;
    n--;
  }

  return answer;
}
```

The method `f2` will always produce the same results as `f1` under which of the following conditions?

I. $n < 0$

II. $n = 0$

III. $n > 0$

(A) I only

(B) II only

(C) III only

(D) II and III only

(E) I, II, and III

20. Consider the following class definitions.

```
public class A
{
  private int a1;

  public void methodA()
  {
    methodB();          // Statement I
  }
}

public class B extends A
{
  public void methodB()
  {
    methodA();          // Statement II
    a1 = 0;             // Statement III
  }
}
```

Which of the labeled statements in the methods shown above will cause a compile-time error?

(A) I only

(B) III only

(C) I and II

(D) I and III

(E) II and III

26. Assume that methods `f` and `g` are defined as follows.

```
public int f(int x)
{
  if (x <= 0)
  {
    return 0;
  }
  else
  {
    return g(x - 1);
  }
}

public int g(int x)
{
  if (x <= 0)
  {
    return 0;
  }
  else
  {
    return (f(x - 1) + x);
  }
}
```

What value is returned as a result of the call `f(6)` ?

(A) 0

(B) 3

(C) 6

(D) 9

(E) 12

27. Assume that `class Vehicle` contains the following method.

```
public void setPrice(double price)
{ /* implementation not shown */ }
```

Also assume that `class Car extends Vehicle` and contains the following method.

```
public void setPrice(double price)
{ /* implementation not shown */ }
```

Assume `Vehicle v` is initialized as follows.

```
Vehicle v = new Car();
v.setPrice(1000.0);
```

Which of the following is true?

(A) The code above will cause a compile-time error, because a subclass cannot have a method with the same name and the same signature as its superclass.

(B) The code above will cause a run-time error, because a subclass cannot have a method with the same name and the same signature as its superclass.

(C) The code above will cause a compile-time error because of type mismatch.

(D) The code `v.setPrice(1000.0);` will cause the `setPrice` method of the `Car` class to be called.

(E) The code `v.setPrice(1000.0);` will cause the `setPrice` method of the `Vehicle` class to be called.

28. Consider the following static method.

```
private static void recur(int n)
{
  if (n != 0)
  {
    recur(n - 2);
    System.out.print(n + " ");
  }
}
```

What numbers will be printed as a result of the call `recur(7)` ?

(A) `-1  1  3  5  7`

(B) `1  3  5  7`

(C) `7  5  3  1`

(D) Many numbers will be printed because of infinite recursion.

(E) No numbers will be printed because of infinite recursion.

29. Integers can be represented using different bases. Base 10 (decimal), and base 16 (hexadecimal) are indicated with the subscripts $_{dec}$ and $_{hex}$, respectively. For example, the decimal number 23 can also be represented in base 16 as shown below.

$$23_{dec} = 17_{hex}$$

Which of the following is equal to $100_{hex} - 10_{hex}$?

(A) 15_{dec}

(B) 90_{dec}

(C) 144_{dec}

(D) 240_{dec}

(E) 256_{dec}

Questions 30-31 are based on the following incomplete declaration of the class `BoundedIntArray` and its constructor definitions.

A `BoundedIntArray` represents an indexed list of integers. In a `BoundedIntArray` the user can specify a size, in which case the indices range from `0` to `size - 1`. The user can also specify the lowest index, `low`, in which case the indices can range from `low` to `low + size - 1`.

```
public class BoundedIntArray
{
  private int[] myItems;        // storage for the list
  private int myLowIndex;       // lowest index

  public BoundedIntArray(int size)
  {
    myItems = new int[size];
    myLowIndex = 0;
  }

  public BoundedIntArray(int size, int low)
  {
    myItems = new int[size];
    myLowIndex = low;
  }

  // other methods not shown
}
```

30. Which of the following is the best reason for declaring the data fields `myItems` and `myLowIndex` to be private rather than public?

(A) This permits `BoundedIntArray` objects to be initialized and modified.

(B) This permits `BoundedIntArray` methods to be written and tested before code that uses a `BoundedIntArray` is written.

(C) This helps to prevent clients of the `BoundedIntArray` class from writing code that would need to be modified if the implementation of `BoundedIntArray` were changed.

(D) This prevents compile-time errors whenever public methods are called that access the private data fields.

(E) This prevents run-time errors whenever public methods are called that access the private data fields.

31. Consider the following statements.

```
BoundedIntArray arr1 = new BoundedIntArray(100, 5);
BoundedIntArray arr2 = new BoundedIntArray(100);
```

Which of the following best describes `arr1` and `arr2` after these statements?

(A) `arr1` and `arr2` both represent lists of integers indexed from `0` to `99`.

(B) `arr1` and `arr2` both represent lists of integers indexed from `5` to `104`.

(C) `arr1` represents a list of integers indexed from `0` to `104`, and `arr2` represents a list of integers indexed from `0` to `99`.

(D) `arr1` represents a list of integers indexed from `5` to `99`, and `arr2` represents a list of integers indexed from `0` to `99`.

(E) `arr1` represents a list of integers indexed from `5` to `104`, and `arr2` represents a list of integers indexed from `0` to `99`.

32. Which of the following is (are) true of an interface?

I. An interface can contain a constructor.

II. An interface can be instantiated.

III. All methods in an interface are abstract.

(A) I only

(B) II only

(C) III only

(D) I and II only

(E) I, II, and III

33. Consider the following three declarations.

I. `Integer obj1 = new Integer(7);`

II. `Comparable obj2 = new Integer(7);`

III. `Comparable obj3 = new Comparable(7);`

Which of these declarations is (are) legal?

(A) I only

(B) I and II only

(C) I and III only

(D) II and III only

(E) I, II, and III

Questions 34-35 refer to the following declarations.

```
public class Point
{
  private double myX;
  private double myY;

  // postcondition: this Point has coordinates (0,0)
  public Point()
  { /* implementation not shown */ }

  // postcondition: this Point has coordinates (x,y)
  public Point(double x, double y)
  { /* implementation not shown */ }

  // other methods not shown
}

public class Circle
{
  private Point myCenter;
  private double myRadius;

  // postcondition: this Circle has center at (0,0) and radius 0.0
  public Circle()
  { /* implementation not shown */ }

  // postcondition: this Circle has the given center and radius
  public Circle(Point center, double radius)
  { /* implementation not shown */ }

  // other methods not shown
}
```

34. In a client program which of the following correctly declares and initializes `Circle circ` with center at (29.5, 33.0) and radius 10.0 ?

(A)
```
Circle circ = new Circle(29.5, 33.0, 10.0);
```

(B)
```
Circle circ = new Circle((29.5, 33.0), 10.0);
```

(C)
```
Circle circ = new Circle(new Point(29.5, 33.0), 10.0);
```

(D)
```
Circle circ = new Circle();
circ.myCenter = new Point(29.5, 33.0);
circ.myRadius = 10.0;
```

(E)
```
Circle circ = new Circle();
circ.myCenter = new Point();
circ.myCenter.myX = 29.5;
circ.myCenter.myY = 33.0;
circ.myRadius = 10.0;
```

35. Which of the following would be the best specification for a `Circle` method `isInside` that determines whether a `Point` lies inside this `Circle`?

(A)
```
public boolean isInside()
```

(B)
```
public void isInside(boolean found)
```

(C)
```
public boolean isInside(Point p)
```

(D)
```
public void isInside(Point p, boolean found)
```

(E)
```
public boolean isInside(Point p, Point center, double radius)
```

36. Consider the following method.

```
public static void sort(String[] arr)
{
  for (int pass = arr.length - 1; pass >= 1; pass--)
  {
    String large = arr[0];
    int index = 0;

    for (int k = 0; k <= pass; k++)
    {
      if ((arr[k].compareTo(large)) > 0)
      {
        large = arr[k];
        index = k;
      }
    }

    arr[index] = arr[pass];
    arr[pass] = large;
  }
}
```

Assume `arr` is the following array.

"Ann"	"Mike"	"Walt"	"Lisa"	"Shari"	"Jose"	"Mary"	"Bill"

What is the intermediate value of `arr` after two iterations of the outer `for` loop in the call `sort(arr)` ?

(A)

"Ann"	"Mike"	"Walt"	"Lisa"	"Shari"	"Jose"	"Mary"	"Bill"

(B)

"Ann"	"Mike"	"Lisa"	"Shari"	"Jose"	"Mary"	"Bill"	"Walt"

(C)

"Ann"	"Bill"	"Jose"	"Lisa"	"Mary"	"Mike"	"Shari"	"Walt"

(D)

"Ann"	"Mike"	"Bill"	"Lisa"	"Mary"	"Jose"	"Shari"	"Walt"

(E)

"Walt"	"Shari"	"Ann"	"Lisa"	"Mike"	"Jose"	"Mary"	"Bill"

37. The following incomplete method is intended to return the largest integer in the array `numbers`.

```
// precondition:  numbers.length > 0
public static int findMax(int[] numbers)
{
  int posOfMax = 0;

  for (int index = 1; index < numbers.length; index++)
  {
    if ( /* condition */ )
    {
      /* statement */
    }
  }
  return numbers[posOfMax];
}
```

Which of the following can be used to replace `/* condition */` and `/* statement */` so that `findMax` will work as intended?

	`/* condition */`	`/* statement */`
(A)	`numbers[index] > numbers[posOfMax]`	`posOfMax = numbers[index];`
(B)	`numbers[index] > numbers[posOfMax]`	`posOfMax = index;`
(C)	`numbers[index] > posOfMax`	`posOfMax = numbers[index];`
(D)	`numbers[index] < posOfMax`	`posOfMax = numbers[index];`
(E)	`numbers[index] < numbers[posOfMax]`	`posOfMax = index;`

Questions 38-39 refer to the following information.

Consider the following data field and method. The method `removeDups` is intended to remove all adjacent duplicate numbers from `myData`, but does not work as intended.

```
private ArrayList myData;

public void removeDups()
{
  int k = 1;
  while (k < myData.size())
  {
    if (myData.get(k).equals(myData.get(k - 1)))
    {
      myData.remove(k);
    }
    k++;
  }
}
```

For example, if `myData` has the values `3 3 4 4 4 8 7 7 7`, after calling `removeDups`, `myData` should have the values `3 4 8 7`.

38. Assume that `myData` has the following values.

```
2 7 5 5 5 5 6 6 3 3 3
```

Which of the following represents `myData` after the incorrect `removeDups` is executed?

(A) `2 7 5 6 3`

(B) `2 7 5 6 3 3`

(C) `2 7 5 5 6 3 3`

(D) `2 7 5 5 5 6 3 3`

(E) `2 7 5 5 5 5 6 6 3 3`

39. Which of the following best describes how to fix the error so that `removeDups` works as intended?

(A) `k` should be initialized to 0 at the beginning of the method.

(B) The `while` condition should be `(k < myData.size() - 1)`.

(C) The `if` test should be `(myData.get(k).equals(myData.get(k + 1)))`.

(D) The body of the `if` statement should be: `myData.remove(k - 1);`

(E) There should be an `else` before the statement `k++;`

40. Consider the following output.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

Which of the following code segments will produce the output shown above?

(A)
```
for (int j = 1; j <= 6; j++)
{
  for (int k = 1; k < j; k++)
    System.out.print(" " + k);
  System.out.println();
}
```

(B)
```
for (int j = 1; j <= 6; j++)
{
  for (int k = 1; k <= j; k++)
    System.out.print(" " + j);
  System.out.println();
}
```

(C)
```
for (int j = 1; j <= 6; j++)
{
  for (int k = 1; k <= j; k++)
    System.out.print(" " + k);
  System.out.println();
}
```

(D)
```
for (int j = 1; j < 6; j++)
{
  for (int k = 1; k <= j; k++)
    System.out.print(" " + k);
  System.out.println();
}
```

(E)
```
for (int j = 1; j < 6; j++)
{
  for (int k = 1; k < j; k++)
    System.out.print(" " + k);
  System.out.println();
}
```

Sample Computer Science Exam (provided by the College Board)

1. Consider the following method.

```
public static int mystery(int[] arr)
{
  int x = 0;

  for (int k = 0; k < arr.length; k = k + 2)
    x = x + arr[k];

  return x;
}
```

Assume that the array `nums` has been declared and initialized as follows.

```
int[] nums = {3, 6, 1, 0, 1, 4, 2};
```

What value will be returned as a result of the call `mystery(nums)` ?

(A) 5

(B) 6

(C) 7

(D) 10

(E) 17

Questions 2-3 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;  }
}
```

2. 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);
```

3. Which of the following changes to `SomeClass` will allow other classes to access but not modify the value of `myC` ?

(A) Make `myC` public.

(B) Include the method:
```
public int getC()
{  return myC;  }
```

(C) Include the method:
```
private int getC()
{  return myC;  }
```

(D) Include the method:
```
public void getC(int x)
{  x = myC;  }
```

(E) Include the method:
```
private void getC(int x)
{  x = myC;  }
```

4. Consider the following code segment.

```
int x = 7;
int y = 3;

if ((x < 10) && (y < 0))
  System.out.println("Value is: " + x * y);
else
  System.out.println("Value is: " + x / y);
```

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

(A) `Value is: 21`

(B) `Value is: 2.3333333`

(C) `Value is: 2`

(D) `Value is: 0`

(E) `Value is: 1`

5. Consider the following method.

```
public ArrayList<Integer> mystery(int n)
{
  ArrayList<Integer> seq = new ArrayList<Integer>();

  for (int k = 1; k <= n; k++)
    seq.add(new Integer(k * k + 3));

  return seq;
}
```

Which of the following is printed as a result of executing the following statement?

```
System.out.println(mystery(6));
```

(A) `[3, 4, 7, 12, 19, 28]`

(B) `[3, 4, 7, 12, 19, 28, 39]`

(C) `[4, 7, 12, 19, 28, 39]`

(D) `[39, 28, 19, 12, 7, 4]`

(E) `[39, 28, 19, 12, 7, 4, 3]`

6. Consider the following method that is intended to determine if the `double` values `d1` and `d2` are close enough to be considered equal. For example, given a `tolerance` of `0.001`, the values `54.32271` and `54.32294` would be considered equal.

```
/** @return true if d1 and d2 are within the specified tolerance,
 *          false otherwise
 */
public boolean almostEqual(double d1, double d2, double tolerance)
{
  /* missing code */
}
```

Which of the following should replace `/* missing code */` so that `almostEqual` will work as intended?

(A) `return (d1 - d2) <= tolerance;`

(B) `return ((d1 + d2) / 2) <= tolerance;`

(C) `return (d1 - d2) >= tolerance;`

(D) `return ((d1 + d2) / 2) >= tolerance;`

(E) `return Math.abs(d1 - d2) <= tolerance;`

7. 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");`

8. 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

9. Consider the following method that is intended to return the sum of the elements in the array `key`.

```
public static int sumArray(int[] key)
{
  int sum = 0;

  for (int i = 1; i <= key.length; i++)
  {
    /* missing code */
  }

  return sum;
}
```

Which of the following statements should be used to replace `/* missing code */` so that `sumArray` will work as intended?

(A) `sum = key[i];`

(B) `sum += key[i - 1];`

(C) `sum += key[i];`

(D) `sum += sum + key[i - 1];`

(E) `sum += sum + key[i];`

Questions 10-11 refer to the following information.

Consider the following instance variable and methods. You may assume that `data` has been initialized with length > 0. The methods are intended to return the index of an array element equal to `target`, or -1 if no such element exists.

```
private int[] data;

public int seqSearchRec(int target)
{
  return seqSearchRecHelper(target, data.length - 1);
}


private int seqSearchRecHelper(int target, int last)
{
  // Line 1

  if (data[last] == target)
    return last;
  else
    return seqSearchRecHelper(target, last - 1);
}
```

10. For which of the following test cases will the call `seqSearchRec(5)` always result in an error?

    I. `data` contains only one element.

    II. `data` does not contain the value 5.

    III. `data` contains the value 5 multiple times.

    (A) I only

    (B) II only

    (C) III only

    (D) I and II only

    (E) I, II, and III

11. Which of the following should be used to replace `// Line 1` in `seqSearchRecHelper` so that `seqSearchRec` will work as intended?

    (A)
    ```
    if (last <= 0)
      return -1;
    ```

    (B)
    ```
    if (last < 0)
      return -1;
    ```

    (C)
    ```
    if (last < data.length)
      return -1;
    ```

    (D)
    ```
    while (last < data.length)
    ```

    (E)
    ```
    while (last >= 0)
    ```

12. Consider the following method.

```
public String mystery(String input)
{
  String output = "";

  for (int k = 1; k < input.length(); k = k + 2)
  {
    output += input.substring(k, k + 1);
  }

  return output;
}
```

What is returned as a result of the call `mystery("computer")` ?

(A) `"computer"`

(B) `"cmue"`

(C) `"optr"`

(D) `"ompute"`

(E) Nothing is returned because an `IndexOutOfBoundsException` is thrown.

13. Consider the following code segment.

```
int[] arr = {7, 2, 5, 3, 0, 10};
for (int k = 0; k < arr.length - 1; k++)
{
  if (arr[k] > arr[k + 1])
    System.out.print(k + "  " + arr[k] + "  ");
}
```

What will be printed as a result of executing the code segment?

(A) `0  2  2  3  3  0`

(B) `0  7  2  5  3  3`

(C) `0  7  2  5  5  10`

(D) `1  7  3  5  4  3`

(E) `7  2  5  3  3  0`

14. Consider the following interface and class declarations.

```
public interface Vehicle
{
  /** @return the mileage traveled by this Vehicle
   */
  double getMileage();
}


public class Fleet
{
  private ArrayList<Vehicle> myVehicles;

  /** @return the mileage traveled by all vehicles in this Fleet
   */
  public double getTotalMileage()
  {
    double sum = 0.0;

    for (Vehicle v : myVehicles)
    {
      sum += /* expression */ ;
    }

    return sum;
  }

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

Which of the following can be used to replace `/* expression */` so that `getTotalMileage` returns the total of the miles traveled for all vehicles in the fleet?

(A) `getMileage(v)`

(B) `myVehicles[v].getMileage()`

(C) `Vehicle.get(v).getMileage()`

(D) `myVehicles.get(v).getMileage()`

(E) `v.getMileage()`

15. Consider the following method, `isSorted`, which is intended to return `true` if an array of integers is sorted in nondecreasing order and to return `false` otherwise.

```
/** @param data an array of integers
 *  @return true if the values in the array appear in sorted (nondecreasing) order
 */
public static boolean isSorted(int[] data)
{
  /* missing code */
}
```

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

I.
```
for (int k = 1; k < data.length; k++)
{
  if (data[k - 1] > data[k])
    return false;
}
return true;
```

II.
```
for (int k = 0; k < data.length; k++)
{
  if (data[k] > data[k + 1])
    return false;
}
return true;
```

III.
```
for (int k = 0; k < data.length - 1; k++)
{
  if (data[k] > data[k + 1])
    return false;
  else
    return true;
}
return true;
```

(A) I only

(B) II only

(C) III only

(D) I and II only

(E) I and III only

16. Consider the following incomplete method that is intended to return an array that contains the contents of its first array parameter followed by the contents of its second array parameter.

```
public static int[] append(int[] a1, int[] a2)
{
  int[] result = new int[a1.length + a2.length];

  for (int j = 0; j < a1.length; j++)
    result[j] = a1[j];

  for (int k = 0; k < a2.length; k++)
    result[ /* index */ ] = a2[k];

  return result;
}
```

Which of the following expressions can be used to replace `/* index */` so that `append` will work as intended?

(A) `j`

(B) `k`

(C) `k + a1.length - 1`

(D) `k + a1.length`

(E) `k + a1.length + 1`

17. Consider the following code segment.

```
int[] arr = {1, 2, 3, 4, 5, 6, 7};

for (int k = 3; k < arr.length - 1; k++)
  arr[k] = arr[k + 1];
```

Which of the following represents the contents of `arr` as a result of executing the code segment?

(A) `{1, 2, 3, 4, 5, 6, 7}`

(B) `{1, 2, 3, 5, 6, 7}`

(C) `{1, 2, 3, 5, 6, 7, 7}`

(D) `{1, 2, 3, 5, 6, 7, 8}`

(E) `{2, 3, 4, 5, 6, 7, 7}`

18. Assume that `myList` is an `ArrayList` that has been correctly constructed and populated with objects. Which of the following expressions produces a valid random index for `myList`?

(A) `(int)( Math.random() * myList.size() ) - 1`

(B) `(int)( Math.random() * myList.size() )`

(C) `(int)( Math.random() * myList.size() ) + 1`

(D) `(int)( Math.random() * (myList.size() + 1) )`

(E) `Math.random(myList.size())`

19. Assume that `a` and `b` have been defined and initialized as `int` values. The expression

```
!(!(a != b ) && (b > 7))
```

is equivalent to which of the following?

(A) `(a != b) || (b < 7)`

(B) `(a != b) || (b <= 7)`

(C) `(a == b) || (b <= 7)`

(D) `(a != b) && (b <= 7)`

(E) `(a == b) && (b > 7)`

20. Consider the following method.

```
public static void arrayMethod(int nums[])
{
  int j = 0;
  int k = nums.length - 1;

  while (j < k)
  {
    int x = nums[j];
    nums[j] = nums[k];
    nums[k] = x;
    j++;
    k--;
  }
}
```

Which of the following describes what the method `arrayMethod()` does to the array `nums`?

(A) The array `nums` is unchanged.

(B) The first value in `nums` is copied to every location in the array.

(C) The last value in `nums` is copied to every location in the array.

(D) The method generates an `ArrayIndexOutOfBoundsException`.

(E) The contents of the array `nums` are reversed.

Questions 21-25 refer to the code from the GridWorld case study. A copy of the code is provided in the Appendix.

21. 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.

    Which method(s) of the `Bug` class should the `Grasshopper` class override so that a `Grasshopper` can behave as described above?

    I. `act()`

    II. `move()`

    III. `canMove()`

    (A) I only

    (B) II only

    (C) I and II only

    (D) II and III only

    (E) I, II, and III

22. Assume that `gus` has been defined and initialized as a `Bug` object in a class that contains the following code segment.

```
int numTurnsMade = 0;
for (int k = 1; k <= 100; k++)
{
  int dir = gus.getDirection();
  int dirTurn = dir + Location.HALF_RIGHT;
  gus.act();
  if ( /* expression */ )
    numTurnsMade++;
}
```

    Which of the following could be used to replace `/* expression */` so that the variable `numTurnsMade` accurately stores the number of times that `gus` turns 45 degrees to the right?

    (A) `dir == dirTurn`

    (B) `dir == gus.getDirection()`

    (C) `dirTurn == Location.HALF_RIGHT`

    (D) `dirTurn == gus.getDirection()`

    (E) `Location.HALF_RIGHT == gus.getDirection()`

23. 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()`

24. A `ColorChangingCritter` behaves like a `ChameleonCritter` but does not turn when it moves. A partial declaration for the `ColorChangingCritter` class is as follows.

```
public class ColorChangingCritter extends ChameleonCritter
{
  public void makeMove(Location loc)
  {  /* missing code */  }
}
```

Which of the following replacements for `/* missing code */` will correctly implement the desired behavior?

I. `moveTo(loc);`

II. `super.super.makeMove(loc);`

III.
```
int dir = getDirection();
super.makeMove(loc);
setDirection(dir);
```

(A) I only

(B) III only

(C) I and II only

(D) I and III only

(E) I, II, and III

25. A `MunchingCritter` acts by selecting one adjacent actor of any type, eating it (removing it from the grid), and moving to occupy its location. If there is no adjacent actor, the `MunchingCritter` moves like a normal critter. Consider the following three implementations of `MunchingCritter`.

Implementation I

```
public class MunchingCritter extends Critter
{
  private Location eatLoc;  // Remember location of critter that was eaten

  public void processActors(ArrayList<Actor> actors)
  {
    if (actors.size() == 0)
      eatLoc = null;
    else
    {
      Actor selected = actors.get(0);
      eatLoc = selected.getLocation();
      selected.removeSelfFromGrid();
    }
  }

  public Location selectMoveLocation(ArrayList<Location> locs)
  {
    if (eatLoc == null)
      return super.selectMoveLocation(locs);
    else
      return eatLoc;
  }
}
```

Implementation II

```
public class MunchingCritter extends Critter
{
  private Location eatLoc;  // Remember location of critter that was eaten

  public void processActors(ArrayList<Actor> actors)
  {
    if (actors.size() == 0)
      eatLoc = null;
    else
    {
      Actor selected = actors.get(0);
      eatLoc = selected.getLocation();
      selected.removeSelfFromGrid();
    }
  }

  public void makeMove(Location loc)
  {
    if (eatLoc == null)
      moveTo(loc);
    else
      moveTo(eatLoc);
  }
}
```

Implementation III

```
public class MunchingCritter extends Critter
{
  private boolean hasEaten;  // Remember if this critter ate something during this step

  public void processActors(ArrayList<Actor> actors)
  {
    if (actors.size() == 0)
      hasEaten = false;
    else
    {
      Actor selected = actors.get(0);
      Location moveLoc = selected.getLocation();
      selected.removeSelfFromGrid();
      moveTo(moveLoc);
      hasEaten = true;
    }
  }

  public void makeMove(Location loc)
  {
    if (!hasEaten)
      moveTo(loc);
  }
}
```

Which of the implementations would be considered to be well designed, in that they satisfy the postconditions in `Critter.java` ?

(A) I only

(B) II only

(C) III only

(D) I and II only

(E) I, II, and III

26. Assume that the array `arr` has been defined and initialized as follows.

```
int[] arr = /* initial values for the array */ ;
```

Which of the following will correctly print all of the odd integers contained in `arr` but none of the even integers contained in `arr` ?

(A)
```
for (int x : arr)
  if (x % 2 == 1)
    System.out.println(x);
```

(B)
```
for (int k = 1; k < arr.length; k++)
  if (arr[k] % 2 == 1)
    System.out.println(arr[k]);
```

(C)
```
for (int x : arr)
  if (x % 2 == 1)
    System.out.println(arr[x]);
```

(D)
```
for (int k = 0; k < arr.length; k++)
  if (arr[k] % 2 == 1)
    System.out.println(k);
```

(E)
```
for (int x : arr)
  if (arr[x] % 2 == 1)
    System.out.println(arr[x]);
```

Questions 27-28 refer to the following method.

```
public static int mystery(int n)
{
  int x = 1;
  int y = 1;

  // Point A

  while (n > 2)
  {
    x = x + y;

    // Point B

    y = x - y;
    n--;
  }

  // Point C

  return x;
}
```

27. What value is returned as a result of the call `mystery(6)` ?

(A) 1

(B) 5

(C) 6

(D) 8

(E) 13

28. Which of the following is true of method `mystery` ?

(A) `x` will sometimes be `1` at `// Point B`.

(B) `x` will never be `1` at `// Point C`.

(C) `n` will never be greater than `2` at `// Point A`.

(D) `n` will sometimes be greater than `2` at `// Point C`.

(E) `n` will always be greater than `2` at `// Point B`.

29. Consider the following code segment.

```
for (int k = 1; k <= 100; k++)
  if ((k % 4) == 0)
    System.out.println(k);
```

Which of the following code segments will produce the same output as the code segment above?

(A)
```
for (int k = 1; k <= 25; k++)
  System.out.println(k);
```

(B)
```
for (int k = 1; k <= 100; k = k + 4)
  System.out.println(k);
```

(C)
```
for (int k = 1; k <= 100; k++)
  System.out.println(k % 4);
```

(D)
```
for (int k = 4; k <= 25; k = 4 * k)
  System.out.println(k);
```

(E)
```
for (int k = 4; k <= 100; k = k + 4)
  System.out.println(k);
```

30. Consider the following method.

```
public static String scramble(String word, int howFar)
{
  return word.substring(howFar + 1, word.length()) +
         word.substring(0, howFar);
}
```

What value is returned as a result of the call `scramble("compiler", 3)`?

(A) `"compiler"`

(B) `"pilercom"`

(C) `"ilercom"`

(D) `"ilercomp"`

(E) No value is returned because an `IndexOutOfBoundsException` will be thrown.

31. Consider the following method.

```
public void mystery(int[] data)
{
  for (int k = 0; k < data.length - 1; k++)
    data[k + 1] = data[k] + data[k + 1];
}
```

The following code segment appears in another method in the same class.

```
int[] values = {5, 2, 1, 3, 8};
mystery(values);
for (int v : values)
  System.out.print(v + " ");
System.out.println();
```

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

(A) `5 2 1 3 8`

(B) `5 7 3 4 11`

(C) `5 7 8 11 19`

(D) `7 3 4 11 8`

(E) Nothing is printed because an `ArrayIndexOutOfBoundsException` is thrown during the execution of method `mystery`.

32. Consider the following method.

```
public int compute(int n, int k)
{
  int answer = 1;

  for (int i = 1; i <= k; i++)
    answer *= n;

  return answer;
}
```

Which of the following represents the value returned as a result of the call `compute(n, k)` ?

(A) $n*k$

(B) $n!$

(C) n^k

(D) 2^k

(E) k^n

33. Consider the following code segment.

```
int sum = 0;
int k = 1;
while (sum < 12 || k < 4)
  sum += k;

System.out.println(sum);
```

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

(A) 6

(B) 10

(C) 12

(D) 15

(E) Nothing is printed due to an infinite loop.

34. Consider the following class declarations.

```
public class Point
{
  private double x;   // x-coordinate
  private double y;   // y-coordinate

  public Point()
  {
    x = 0;
    y = 0;
  }

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

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


public class Circle
{
  private Point center;
  private double radius;

  /** Constructs a circle where (a, b) is the center and r is the radius.
   */
  public Circle(double a, double b, double r)
  {
    /* missing code */
  }
}
```

Which of the following replacements for `/* missing code */` will correctly implement the `Circle` constructor?

I.
```
center = new Point();
radius = r;
```

II.
```
center = new Point(a, b);
radius = r;
```

III.
```
center = new Point();
center.x = a;
center.y = b;
radius = r;
```

(A) I only

(B) II only

(C) III only

(D) II and III only

(E) I, II, and III

35. Consider the following code segment.

```
int num = 2574;
int result = 0;

while (num > 0)
{
  result = result * 10 + num % 10;
  num /= 10;
}
System.out.println(result);
```

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

(A) 2

(B) 4

(C) 18

(D) 2574

(E) 4752

36. Consider the following method.

```
public void test(int x)
{
  int y;

  if (x % 2 == 0)
    y = 3;
  else if (x > 9)
    y = 5;
  else
    y = 1;

  System.out.println("y = " + y);
}
```

Which of the following test data sets would test each possible output for the method?

(A) 8, 9, 12

(B) 7, 9, 11

(C) 8, 9, 11

(D) 8, 11, 13

(E) 7, 9, 10

37. Consider the following code segment.

```
int x = 1;
while ( /* missing code */ )
{
  System.out.print(x + " ");
  x = x + 2;
}
```

Consider the following possible replacements for `/* missing code */`.

I. `x < 6`

II. `x != 6`

III. `x < 7`

Which of the proposed replacements for `/* missing code */` will cause the code segment to print only the values `1 3 5` ?

(A) I only

(B) II only

(C) I and II only

(D) I and III only

(E) I, II, and III

38. Assume that `x` and `y` have been declared and initialized with `int` values. Consider the following Java expression.

```
(y > 10000) || (x > 1000 && x < 1500)
```

Which of the following is equivalent to the expression given above?

(A) `(y > 10000 || x > 1000) && (y > 10000 || x < 1500)`

(B) `(y > 10000 || x > 1000) || (y > 10000 || x < 1500)`

(C) `(y > 10000) && (x > 1000 || x < 1500)`

(D) `(y > 10000 && x > 1000) || (y > 10000 && x < 1500)`

(E) `(y > 10000 && x > 1000) && (y > 10000 && x < 1500)`

39. Consider the following recursive method.

```
public int recur(int n)
{
  if (n <= 10)
    return n * 2;
  else
    return recur(recur(n / 3));
}
```

What value is returned as a result of the call `recur(27)` ?

(A) 8

(B) 9

(C) 12

(D) 16

(E) 18

40. Consider the following recursive method.

```
public static void whatsItDo(String str)
{
  int len = str.length();
  if (len > 1)
  {
    String temp = str.substring(0, len - 1);
    whatsItDo(temp);
    System.out.println(temp);
  }
}
```

What is printed as a result of the call `whatsItDo("WATCH")` ?

(A)
```
WATC
WAT
WA
W
```

(B)
```
WATCH
WATC
WAT
WA
```

(C)
```
W
WA
WAT
WATC
```

(D)
```
W
WA
WAT
WATC
WATCH
```

(E)
```
WATCH
WATC
WAT
WA
W
WA
WAT
WATC
WATCH
```

Free-Response Sections

2021 Free Response Questions

1. This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
public class WordMatch
{
    /** The secret string. */
    private String secret;

    /** Constructs a WordMatch object with the given secret string of lowercase letters. */
    public WordMatch(String word)
    {
        /* implementation not shown */
    }

    /** Returns a score for guess, as described in part (a).
     *  Precondition: 0 < guess.length() <= secret.length()
     */
    public int scoreGuess(String guess)
    {  /* to be implemented in part (a) */  }

    /** Returns the better of two guesses, as determined by scoreGuess and the rules for a
     *  tie-breaker that are described in part (b).
     *  Precondition: guess1 and guess2 contain all lowercase letters.
     *                guess1 is not the same as guess2.
     */
    public String findBetterGuess(String guess1, String guess2)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.

Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.

The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: **(Number of Substring Occurrences)** x **(Square of the Length of** `guess`**)**	**Return Value of** `game.scoreGuess(guess)`
`"i"`	4	`4 * 1 * 1 = 4`	`4`
`"iss"`	2	`2 * 3 * 3 = 18`	`18`
`"issipp"`	1	`1 * 6 * 6 = 36`	`36`
`"mississippi"`	1	`1 * 11 * 11 = 121`	`121`

```
WordMatch game = new WordMatch("aaaabb");
```

Value of `guess`	**Number of Substring Occurrences**	**Score Calculation**: **(Number of Substring Occurrences)** x **(Square of the Length of** `guess`**)**	**Return Value of** `game.scoreGuess(guess)`
`"a"`	4	`4 * 1 * 1 = 4`	`4`
`"aa"`	3	`3 * 2 * 2 = 12`	`12`
`"aaa"`	2	`2 * 3 * 3 = 18`	`18`
`"aabb"`	1	`1 * 4 * 4 = 16`	`16`
`"c"`	0	`0 * 1 * 1 = 0`	`0`

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

Complete the `scoreGuess` method.

```
/** Returns a score for guess, as described in part (a).
 *  Precondition: 0 < guess.length() <= secret.length()
 */
public int scoreGuess(String guess)
```

(b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two `String` parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.

The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
`game.scoreGuess("ten");`	`9`	`1 * 3 * 3`
`game.scoreGuess("nation");`	`36`	`1 * 6 * 6`
`game.findBetterGuess("ten", "nation");`	`"nation"`	Since `scoreGuess` returns `36` for `"nation"` and `9` for `"ten"`, the guess with the greater score, `"nation"`, is returned.
`game.scoreGuess("con");`	`9`	`1 * 3 * 3`
`game.scoreGuess("cat");`	`9`	`1 * 3 * 3`
`game.findBetterGuess("con", "cat");`	`"con"`	Since `scoreGuess` returns `9` for both `"con"` and `"cat"`, the alphabetically greater guess, `"con"`, is returned.

Class information for this question

```
public class WordMatch

private String secret

public WordMatch(String word)
public int scoreGuess(String guess)
public String findBetterGuess(String guess1, String guess2)
```

Complete method `findBetterGuess`.

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

```
/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
 *  tie-breaker that are described in part (b).
 *  Precondition: guess1 and guess2 contain all lowercase letters.
 *                guess1 is not the same as guess2.
 */
public String findBetterGuess(String guess1, String guess2)
```

2. 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 `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.

3. A high school club maintains information about its members in a `MemberInfo` object. A `MemberInfo` object stores a club member's name, year of graduation, and whether or not the club member is in *good standing*. A member who is in good standing has fulfilled all the responsibilities of club membership.

   A partial declaration of the `MemberInfo` class is shown below.

```
public class MemberInfo
{
    /** Constructs a MemberInfo object for the club member with name name,
     *  graduation year gradYear, and standing hasGoodStanding.
     */
    public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
    {  /* implementation not shown */  }

    /** Returns the graduation year of the club member. */
    public int getGradYear()
    {  /* implementation not shown */  }

    /** Returns true if the member is in good standing and false otherwise. */
    public boolean inGoodStanding()
    {  /* implementation not shown */  }

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

   The `ClubMembers` class maintains a list of current club members. The declaration of the `ClubMembers` class is shown below.

```
public class ClubMembers
{
   private ArrayList<MemberInfo> memberList;

    /** Adds new club members to memberList, as described in part (a).
     *  Precondition: names is a non-empty array.
     */
    public void addMembers(String[] names, int gradYear)
    {  /* to be implemented in part (a) */  }

    /** Removes members who have graduated and returns a list of members who have graduated
     *  and are in good standing, as described in part (b).
     */
    public ArrayList<MemberInfo> removeMembers(int year)
    {  /* to be implemented in part (b) */  }

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

(a) Write the `ClubMembers` method `addMembers`, which takes two parameters. The first parameter is a `String` array containing the names of new club members to be added. The second parameter is the graduation year of all the new club members. The method adds the new members to the `memberList` instance variable. The names can be added in any order. All members added are initially in good standing and share the same graduation year, `gradYear`.

Complete the `addMembers` method.

```
/** Adds new club members to memberList, as described in part (a).
 *  Precondition: names is a non-empty array.
 */
public void addMembers(String[] names, int gradYear)
```

(b) Write the `ClubMembers` method `removeMembers`, which takes the following actions.

- Returns a list of all students who have graduated and are in good standing. A member has graduated if the member's graduation year is less than or equal to the method's `year` parameter. If no members meet these criteria, an empty list is returned.
- Removes from `memberList` all members who have graduated, regardless of whether or not they are in good standing.

The following example illustrates the results of a call to `removeMembers`.

The `ArrayList memberList` before the method call `removeMembers(2018)`:

"SMITH, JANE"	"FOX, STEVE"	"XIN, MICHAEL"	"GARCIA, MARIA"
2019	2018	2017	2020
false	true	false	true

The `ArrayList memberList` after the method call `removeMembers(2018)`:

"SMITH, JANE"	"GARCIA, MARIA"
2019	2020
false	true

The `ArrayList` returned by the method call `removeMembers(2018)`:

"FOX, STEVE"
2018
true

Class information for this question

```
public class MemberInfo

public MemberInfo(String name, int gradYear, boolean hasGoodStanding)
public int getGradYear()
public boolean inGoodStanding()

public class ClubMembers

private ArrayList<MemberInfo> memberList

public void addMembers(String[] names, int gradYear)
public ArrayList<MemberInfo> removeMembers(int year)
```

Complete the `removeMembers` method.

```
/** Removes members who have graduated and returns a list of members who have graduated and are
 *  in good standing, as described in part (b).
 */
public ArrayList<MemberInfo> removeMembers(int year)
```

4. This question involves manipulating a two-dimensional array of integers. You will write two `static` methods of the `ArrayResizer` class, which is shown below.

```
public class ArrayResizer
{
    /** Returns true if and only if every value in row r of array2D is non-zero.
     *  Precondition: r is a valid row index in array2D.
     *  Postcondition: array2D is unchanged.
     */
    public static boolean isNonZeroRow(int[][] array2D, int r)
    {  /* to be implemented in part (a) */  }

    /** Returns the number of rows in array2D that contain all non-zero values.
     *  Postcondition: array2D is unchanged.
     */
    public static int numNonZeroRows(int[][] array2D)
    {  /* implementation not shown */  }

    /** Returns a new, possibly smaller, two-dimensional array that contains only rows
     *  from array2D with no zeros, as described in part (b).
     *  Precondition: array2D contains at least one column and at least one row with no zeros.
     *  Postcondition: array2D is unchanged.
     */
    public static int[][] resize(int[][] array2D)
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the method `isNonZeroRow`, which returns `true` if and only if all elements in row `r` of a two-dimensional array `array2D` are not equal to zero.

For example, consider the following statement, which initializes a two-dimensional array.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
```

Sample calls to `isNonZeroRow` are shown below.

Call to `isNonZeroRow`	Value Returned	Explanation
`ArrayResizer.isNonZeroRow(arr, 0)`	`false`	At least one value in row `0` is zero.
`ArrayResizer.isNonZeroRow(arr, 1)`	`true`	All values in row `1` are non-zero.
`ArrayResizer.isNonZeroRow(arr, 2)`	`false`	At least one value in row `2` is zero.
`ArrayResizer.isNonZeroRow(arr, 3)`	`true`	All values in row `3` are non-zero.

Complete the `isNonZeroRow` method.

```
/** Returns true if and only if every value in row r of array2D is non-zero.
 *  Precondition: r is a valid row index in array2D.
 *  Postcondition: array2D is unchanged.
 */
public static boolean isNonZeroRow(int[][] array2D, int r)
```

Class information for this question

```
public class ArrayResizer

public static boolean isNonZeroRow(int[][] array2D, int r)
public static int numNonZeroRows(int[][] array2D)
public static int[][] resize(int[][] array2D)
```

(b) Write the method `resize`, which returns a new two-dimensional array containing only rows from `array2D` with all non-zero values. The elements in the new array should appear in the same order as the order in which they appeared in the original array.

The following code segment initializes a two-dimensional array and calls the `resize` method.

```
int[][] arr = {{2, 1, 0},
               {1, 3, 2},
               {0, 0, 0},
               {4, 5, 6}};
int[][] smaller = ArrayResizer.resize(arr);
```

When the code segment completes, the following will be the contents of `smaller`.

```
{{1, 3, 2}, {4, 5, 6}}
```

A helper method, `numNonZeroRows`, has been provided for you. The method returns the number of rows in its two-dimensional array parameter that contain no zero values.

Complete the `resize` method. Assume that `isNonZeroRow` works as specified, regardless of what you wrote in part (a). You must use `numNonZeroRows` and `isNonZeroRow` appropriately to receive full credit.

```
/** Returns a new, possibly smaller, two-dimensional array that contains only rows from array2D
 *  with no zeros, as described in part (b).
 *  Precondition: array2D contains at least one column and at least one row with no zeros.
 *  Postcondition: array2D is unchanged.
 */
public static int[][] resize(int[][] array2D)
```

2019 Free Response Questions

1. The `APCalendar` class contains methods used to calculate information about a calendar. You will write two methods of the class.

```
public class APCalendar
{

    /** Returns true if year is a leap year and false otherwise. */
    private static boolean isLeapYear(int year)
    {  /* implementation not shown */  }

    /** Returns the number of leap years between year1 and year2, inclusive.
     *  Precondition: 0 <= year1 <= year2
     */
    public static int numberOfLeapYears(int year1, int year2)
    {  /* to be implemented in part (a) */  }

    /** Returns the value representing the day of the week for the first day of year,
     *  where 0 denotes Sunday, 1 denotes Monday, ..., and 6 denotes Saturday.
     */
    private static int firstDayOfYear(int year)
    {  /* implementation not shown */  }

    /** Returns n, where month, day, and year specify the nth day of the year.
     *  Returns 1 for January 1 (month = 1, day = 1) of any year.
     *  Precondition: The date represented by month, day, year is a valid date.
     */
    private static int dayOfYear(int month, int day, int year)
    {  /* implementation not shown */  }

    /** Returns the value representing the day of the week for the given date
     *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
     *  and 6 denotes Saturday.
     *  Precondition: The date represented by month, day, year is a valid date.
     */
    public static int dayOfWeek(int month, int day, int year)
    {  /* to be implemented in part (b) */  }

    // There may be instance variables, constructors, and other methods not shown.

}
```

(a) Write the static method `numberOfLeapYears`, which returns the number of leap years between `year1` and `year2`, inclusive.

In order to calculate this value, a helper method is provided for you.

- `isLeapYear(year)` returns `true` if `year` is a leap year and `false` otherwise.

Complete method `numberOfLeapYears` below. You must use `isLeapYear` appropriately to receive full credit.

```
/** Returns the number of leap years between year1 and year2, inclusive.
 *  Precondition: 0 <= year1 <= year2
 */
public static int numberOfLeapYears(int year1, int year2)
```

(b) Write the static method `dayOfWeek`, which returns the integer value representing the day of the week for the given date (`month`, `day`, `year`), where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, 2019 began on a Tuesday, and January 5 is the fifth day of 2019. As a result, January 5, 2019, fell on a Saturday, and the method call `dayOfWeek(1, 5, 2019)` returns `6`.

As another example, January 10 is the tenth day of 2019. As a result, January 10, 2019, fell on a Thursday, and the method call `dayOfWeek(1, 10, 2019)` returns `4`.

In order to calculate this value, two helper methods are provided for you.

- `firstDayOfYear(year)` returns the integer value representing the day of the week for the first day of `year`, where `0` denotes Sunday, `1` denotes Monday, ..., and `6` denotes Saturday. For example, since 2019 began on a Tuesday, `firstDayOfYear(2019)` returns `2`.
- `dayOfYear(month, day, year)` returns *n*, where `month`, `day`, and `year` specify the *n*th day of the year. For the first day of the year, January 1 (`month = 1, day = 1`), the value `1` is returned. This method accounts for whether `year` is a leap year. For example, `dayOfYear(3, 1, 2017)` returns `60`, since 2017 is not a leap year, while `dayOfYear(3, 1, 2016)` returns `61`, since 2016 is a leap year.

Class information for this question

```
public class APCalendar

private static boolean isLeapYear(int year)
public static int numberOfLeapYears(int year1, int year2)
private static int firstDayOfYear(int year)
private static int dayOfYear(int month, int day, int year)
public static int dayOfWeek(int month, int day, int year)
```

Complete method `dayOfWeek` below. You must use `firstDayOfYear` and `dayOfYear` appropriately to receive full credit.

```
/** Returns the value representing the day of the week for the given date
 *  (month, day, year), where 0 denotes Sunday, 1 denotes Monday, ...,
 *  and 6 denotes Saturday.
 *  Precondition: The date represented by month, day, year is a valid date.
 */
public static int dayOfWeek(int month, int day, int year)
```

2. This question involves the implementation of a fitness tracking system that is represented by the `StepTracker` class. A `StepTracker` object is created with a parameter that defines the minimum number of steps that must be taken for a day to be considered *active*.

   The `StepTracker` class provides a constructor and the following methods.

   - `addDailySteps`, which accumulates information about steps, in readings taken once per day
   - `activeDays`, which returns the number of active days
   - `averageSteps`, which returns the average number of steps per day, calculated by dividing the total number of steps taken by the number of days tracked

   The following table contains a sample code execution sequence and the corresponding results.

Statements and Expressions	Value Returned (blank if no value)	Comment
`StepTracker tr = new StepTracker(10000);`		Days with at least 10,000 steps are considered active. Assume that the parameter is positive.
`tr.activeDays();`	`0`	No data have been recorded yet.
`tr.averageSteps();`	`0.0`	When no step data have been recorded, the `averageSteps` method returns `0.0`.
`tr.addDailySteps(9000);`		This is too few steps for the day to be considered active.
`tr.addDailySteps(5000);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`0`	No day had at least 10,000 steps.
`tr.averageSteps();`	`7000.0`	The average number of steps per day is (14000 / 2).
`tr.addDailySteps(13000);`		This represents an active day.
`tr.activeDays();`	`1`	Of the three days for which step data were entered, one day had at least 10,000 steps.
`tr.averageSteps();`	`9000.0`	The average number of steps per day is (27000 / 3).
`tr.addDailySteps(23000);`		This represents an active day.
`tr.addDailySteps(1111);`		This is too few steps for the day to be considered active.
`tr.activeDays();`	`2`	Of the five days for which step data were entered, two days had at least 10,000 steps.
`tr.averageSteps();`	`10222.2`	The average number of steps per day is (51111 / 5).

Write the complete `StepTracker` class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications and conform to the example.

3. Many encoded strings contain *delimiters*. A delimiter is a non-empty string that acts as a boundary between different parts of a larger string. The delimiters involved in this question occur in pairs that must be *balanced*, with each pair having an open delimiter and a close delimiter. There will be only one type of delimiter for each string. The following are examples of delimiters.

   Example 1
   Expressions in mathematics use open parentheses `"("` and close parentheses `")"` as delimiters. For each open parenthesis, there must be a matching close parenthesis.

`(x + y) * 5`	is a valid mathematical expression.
`(x + (y)`	is NOT a valid mathematical expression because there are more open delimiters than close delimiters.

   Example 2
   HTML uses `<B>` and `</B>` as delimiters. For each open delimiter `<B>`, there must be a matching close delimiter `</B>`.

`<B> Make this text bold </B>`	is valid HTML.
`<B> Make this text bold </UB>`	is NOT valid HTML because there is one open delimiter and no matching close delimiter.

In this question, you will write two methods in the following `Delimiters` class.

```
public class Delimiters
{

    /** The open and close delimiters. */
    private String openDel;
    private String closeDel;

    /** Constructs a Delimiters object where open is the open delimiter and close is the
     *  close delimiter.
     *  Precondition: open and close are non-empty strings.
     */
    public Delimiters(String open, String close)
    {
       openDel = open;
       closeDel = close;
    }

    /** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
    public ArrayList<String> getDelimitersList(String[] tokens)
    {  /* to be implemented in part (a) */  }

    /** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
     *  Precondition: delimiters contains only valid open and close delimiters.
     */
    public boolean isBalanced(ArrayList<String> delimiters)
    {  /* to be implemented in part (b) */  }

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

}
```

(a) A string containing text and possibly delimiters has been split into *tokens* and stored in `String[] tokens`. Each token is either an open delimiter, a close delimiter, or a substring that is not a delimiter. You will write the method `getDelimitersList`, which returns an `ArrayList` containing all the open and close delimiters found in `tokens` in their original order.

The following examples show the contents of an `ArrayList` returned by `getDelimitersList` for different open and close delimiters and different `tokens` arrays.

Example 1

```
openDel:  "("

closeDel: ")"
```

tokens:	"("	"x + y"	")"	" * 5"

ArrayList of delimiters:	"("	")"

Example 2

```
openDel:  "<q>"

closeDel: "</q>"
```

tokens:	"<q>"	"yy"	"</q>"	"zz"	"</q>"

ArrayList of delimiters:	"<q>"	"</q>"	"</q>"

Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `getDelimitersList` below.

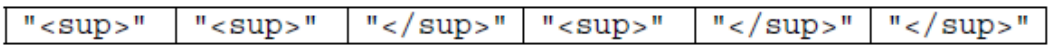
```
/** Returns an ArrayList of delimiters from the array tokens, as described in part (a). */
public ArrayList<String> getDelimitersList(String[] tokens)
```

(b) Write the method `isBalanced`, which returns `true` when the delimiters are balanced and returns `false` otherwise. The delimiters are balanced when both of the following conditions are satisfied; otherwise, they are not balanced.

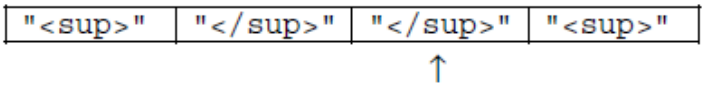
1. When traversing the `ArrayList` from the first element to the last element, there is no point at which there are more close delimiters than open delimiters at or before that point.
2. The total number of open delimiters is equal to the total number of close delimiters.

Consider a `Delimiters` object for which `openDel` is `"<sup>"` and `closeDel` is `"</sup>"`. The examples below show different `ArrayList` objects that could be returned by calls to `getDelimitersList` and the value that would be returned by a call to `isBalanced`.

Example 1
The following example shows an `ArrayList` for which `isBalanced` returns `true`. As tokens are examined from first to last, the number of open delimiters is always greater than or equal to the number of close delimiters. After examining all tokens, there are an equal number of open and close delimiters.

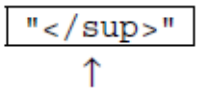


Example 2
The following example shows an `ArrayList` for which `isBalanced` returns `false`.



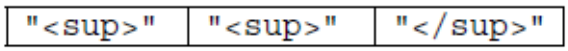
When starting from the left, at this point, condition 1 is violated.

Example 3
The following example shows an `ArrayList` for which `isBalanced` returns `false`.



At this point, condition 1 is violated.

Example 4
The following example shows an `ArrayList` for which `isBalanced` returns `false` because the second condition is violated. After examining all tokens, there are not an equal number of open and close delimiters.



Class information for this question

```
public class Delimiters

private String openDel
private String closeDel

public Delimiters(String open, String close)
public ArrayList<String> getDelimitersList(String[] tokens)
public boolean isBalanced(ArrayList<String> delimiters)
```

Complete method `isBalanced` below.

```
/** Returns true if the delimiters are balanced and false otherwise, as described in part (b).
 *  Precondition: delimiters contains only valid open and close delimiters.
 */
public boolean isBalanced(ArrayList<String> delimiters)
```

4. The `LightBoard` class models a two-dimensional display of lights, where each light is either on or off, as represented by a Boolean value. You will implement a constructor to initialize the display and a method to evaluate a light.

```
public class LightBoard
{
    /** The lights on the board, where true represents on and false represents off.
     */
    private boolean[][] lights;

    /** Constructs a LightBoard object having numRows rows and numCols columns.
     *  Precondition: numRows > 0, numCols > 0
     *  Postcondition: each light has a 40% probability of being set to on.
     */
    public LightBoard(int numRows, int numCols)
    {  /* to be implemented in part (a) */  }

    /** Evaluates a light in row index row and column index col and returns a status
     *  as described in part (b).
     *  Precondition: row and col are valid indexes in lights.
     */
    public boolean evaluateLight(int row, int col)
    {  /* to be implemented in part (b) */  }

    // There may be additional instance variables, constructors, and methods not shown.

}
```

(a) Write the constructor for the `LightBoard` class, which initializes `lights` so that each light is set to on with a 40% probability. The notation `lights[r][c]` represents the array element at row `r` and column `c`.

Complete the `LightBoard` constructor below.

```
/** Constructs a LightBoard object having numRows rows and numCols columns.
 *  Precondition: numRows > 0, numCols > 0
 *  Postcondition: each light has a 40% probability of being set to on.
 */
public LightBoard(int numRows, int numCols)
```

(b) Write the method `evaluateLight`, which computes and returns the status of a light at a given row and column based on the following rules.

1. If the light is on, return `false` if the number of lights in its column that are on is even, including the current light.
2. If the light is off, return `true` if the number of lights in its column that are on is divisible by three.
3. Otherwise, return the light's current status.

For example, suppose that `LightBoard sim = new LightBoard(7, 5)` creates a light board with the initial state shown below, where `true` represents a light that is on and `false` represents a light that is off. Lights that are off are shaded.

`lights`

	0	1	2	3	4
0	true	true	false	true	true
1	true	false	false	true	false
2	true	false	false	true	true
3	true	false	false	false	true
4	true	false	false	false	true
5	true	true	false	true	true
6	false	false	false	false	false

Sample calls to `evaluateLight` are shown below.

Call to `evaluateLight`	Value Returned	Explanation
`sim.evaluateLight(0, 3);`	`false`	The light is on, and the number of lights that are on in its column is even.
`sim.evaluateLight(6, 0);`	`true`	The light is off, and the number of lights that are on in its column is divisible by 3.
`sim.evaluateLight(4, 1);`	`false`	Returns the light's current status.
`sim.evaluateLight(5, 4);`	`true`	Returns the light's current status.

Class information for this question

```
public class LightBoard
private boolean[][] lights
public LightBoard(int numRows, int numCols)
public boolean evaluateLight(int row, int col)
```

Complete the `evaluateLight` method below.

```
/** Evaluates a light in row index row and column index col and returns a status
 *  as described in part (b).
 *  Precondition: row and col are valid indexes in lights.
 */
public boolean evaluateLight(int row, int col)
```

2018 Free Response Questions

1. 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)
```

2. 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 $0 \le i < j <$ `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()
```

3. The `StringChecker` interface describes classes that check if strings are valid, according to some criterion.

```
public interface StringChecker
{
   /** Returns true if str is valid. */
   boolean isValid(String str);
}
```

A `CodeWordChecker` is a `StringChecker`. A `CodeWordChecker` object can be constructed with three parameters: two integers and a string. The first two parameters specify the minimum and maximum code word lengths, respectively, and the third parameter specifies a string that must not occur in the code word. A `CodeWordChecker` object can also be constructed with a single parameter that specifies a string that must not occur in the code word; in this case the minimum and maximum lengths will default to 6 and 20, respectively.

The following examples illustrate the behavior of `CodeWordChecker` objects.

Example 1

```
StringChecker sc1 = new CodeWordChecker(5, 8, "$");
```

Valid code words have 5 to 8 characters and must not include the string `"$"`.

Method call	Return value	Explanation
`sc1.isValid("happy")`	`true`	The code word is valid.
`sc1.isValid("happy$")`	`false`	The code word contains `"$"`.
`sc1.isValid("Code")`	`false`	The code word is too short.
`sc1.isValid("happyCode")`	`false`	The code word is too long.

Example 2

```
StringChecker sc2 = new CodeWordChecker("pass");
```

Valid code words must not include the string `"pass"`. Because the bounds are not specified, the length bounds are 6 and 20, inclusive.

Method call	Return value	Explanation
`sc2.isValid("MyPass")`	`true`	The code word is valid.
`sc2.isValid("Mypassport")`	`false`	The code word contains `"pass"`.
`sc2.isValid("happy")`	`false`	The code word is too short.
`sc2.isValid("1,000,000,000,000,000")`	`false`	The code word is too long.

Write the complete `CodeWordChecker` class. Your implementation must meet all specifications and conform to all examples.

4. This question involves reasoning about arrays of integers. You will write two static methods, both of which are in a class named `ArrayTester`.

```
public class ArrayTester
{

    /** Returns an array containing the elements of column c of arr2D in the same order as
     *  they appear in arr2D.
     *  Precondition: c is a valid column index in arr2D.
     *  Postcondition: arr2D is unchanged.
     */
    public static int[] getColumn(int[][] arr2D, int c)
    {  /* to be implemented in part (a) */  }


    /** Returns true if and only if every value in arr1 appears in arr2.
     *  Precondition: arr1 and arr2 have the same length.
     *  Postcondition: arr1 and arr2 are unchanged.
     */
    public static boolean hasAllValues(int[] arr1, int[] arr2)
    {  /* implementation not shown */  }


    /** Returns true if arr contains any duplicate values;
     *          false otherwise.
     */
    public static boolean containsDuplicates(int[] arr)
    {  /* implementation not shown */  }


    /** Returns true if square is a Latin square as described in part (b);
     *          false otherwise.
     *  Precondition: square has an equal number of rows and columns.
     *                square has at least one row.
     */
    public static boolean isLatin(int[][] square)
    {  /* to be implemented in part (b) */  }

}
```

(a) Write a static method `getColumn`, which returns a one-dimensional array containing the elements of a single column in a two-dimensional array. The elements in the returned array should be in the same order as they appear in the given column. The notation `arr2D[r][c]` represents the array element at row `r` and column `c`.

The following code segment initializes an array and calls the `getColumn` method.

```
int[][] arr2D = { { 0, 1, 2 },
                  { 3, 4, 5 },
                  { 6, 7, 8 },
                  { 9, 5, 3 } };

int[] result = ArrayTester.getColumn(arr2D, 1);
```

When the code segment has completed execution, the variable `result` will have the following contents.

```
result: {1, 4, 7, 5}
```

Complete method `getColumn` below.

```
/** Returns an array containing the elements of column c of arr2D in the same order as they
 *  appear in arr2D.
 *  Precondition: c is a valid column index in arr2D.
 *  Postcondition: arr2D is unchanged.
 */
public static int[] getColumn(int[][] arr2D, int c)
```

(b) Write the static method `isLatin`, which returns `true` if a given two-dimensional square array is a *Latin square*, and otherwise, returns `false`.

A two-dimensional square array of integers is a Latin square if the following conditions are true.

- The first row has no duplicate values.
- All values in the first row of the square appear in each row of the square.
- All values in the first row of the square appear in each column of the square.

Examples of Latin Squares

1	2	3
2	3	1
3	1	2

10	30	20	0
0	20	30	10
30	0	10	20
20	10	0	30

Examples that are NOT Latin Squares

1	2	1
2	1	1
1	1	2

Not a Latin square because the first row contains duplicate values

1	2	3
3	1	2
7	8	9

Not a Latin square because the elements of the first row do not all appear in the third row

1	2
1	2

Not a Latin square because the elements of the first row do not all appear in either column

The `ArrayTester` class provides two helper methods: `containsDuplicates` and `hasAllValues`. The method `containsDuplicates` returns `true` if the given one-dimensional array `arr` contains any duplicate values and `false` otherwise. The method `hasAllValues` returns `true` if and only if every value in `arr1` appears in `arr2`. You do not need to write the code for these methods.

Class information for this question

```
public class ArrayTester

public static int[] getColumn(int[][] arr2D, int c)
public static boolean hasAllValues(int[] arr1, int[] arr2)
public static boolean containsDuplicates(int[] arr)
public static boolean isLatin(int[][] square)
```

Complete method `isLatin` below. Assume that `getColumn` works as specified, regardless of what you wrote in part (a). You must use `getColumn`, `hasAllValues`, and `containsDuplicates` appropriately to receive full credit.

```
/** Returns true if square is a Latin square as described in part (b);
 *          false otherwise.
 *  Precondition: square has an equal number of rows and columns.
 *                square has at least one row.
 */
public static boolean isLatin(int[][] square)
```

2017 Free Response Questions

1. This question involves identifying and processing the digits of a non-negative integer. The declaration of the `Digits` class is shown below. You will write the constructor and one method for the `Digits` class.

```
public class Digits
{
    /** The list of digits from the number used to construct this object.
     *  The digits appear in the list in the same order in which they appear in the original number.
     */
    private ArrayList<Integer> digitList;

    /** Constructs a Digits object that represents num.
     *  Precondition: num >= 0
     */
    public Digits(int num)
    {  /* to be implemented in part (a) */  }

    /** Returns true if the digits in this Digits object are in strictly increasing order;
     *          false otherwise.
     */
    public boolean isStrictlyIncreasing()
    {  /* to be implemented in part (b) */  }
}
```

(a) Write the constructor for the `Digits` class. The constructor initializes and fills `digitList` with the digits from the non-negative integer `num`. The elements in `digitList` must be `Integer` objects representing single digits, and appear in the same order as the digits in `num`. Each of the following examples shows the declaration of a `Digits` object and the contents of `digitList` as initialized by the constructor.

Example 1

```
Digits d1 = new Digits(15704);
```

d1:

	0	1	2	3	4
digitList:	1	5	7	0	4

Example 2

```
Digits d2 = new Digits(0);
```

d2:

	0
digitList:	0

Complete the `Digits` constructor below.

```
/** Constructs a Digits object that represents num.
 *  Precondition: num >= 0
 */
public Digits(int num)
```

(b) Write the `Digits` method `isStrictlyIncreasing`. The method returns `true` if the elements of `digitList` appear in strictly increasing order; otherwise, it returns `false`. A list is considered strictly increasing if each element after the first is greater than (but not equal to) the preceding element.

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

Method call	Value returned
`new Digits(7).isStrictlyIncreasing()`	`true`
`new Digits(1356).isStrictlyIncreasing()`	`true`
`new Digits(1336).isStrictlyIncreasing()`	`false`
`new Digits(1536).isStrictlyIncreasing()`	`false`
`new Digits(65310).isStrictlyIncreasing()`	`false`

Complete method `isStrictlyIncreasing` below.

```
/** Returns true if the digits in this Digits object are in strictly increasing order;
 *         false otherwise.
 */
public boolean isStrictlyIncreasing()
```

2. This question involves the design of a class that will be used to produce practice problems. The following `StudyPractice` interface represents practice problems that can be used to study some subject.

```
public interface StudyPractice
{
    /** Returns the current practice problem. */
    String getProblem();

    /** Changes to the next practice problem. */
    void nextProblem();
}
```

The `MultPractice` class is a `StudyPractice` that produces multiplication practice problems. A `MultPractice` object is constructed with two integer values: *first integer* and *initial second integer*. The first integer is a value that remains constant and is used as the first integer in every practice problem. The initial second integer is used as the starting value for the second integer in the practice problems. This second value is incremented for each additional practice problem that is produced by the class.

For example, a `MultPractice` object created with the call `new MultPractice(7, 3)` would be used to create the practice problems `"7 TIMES 3"`, `"7 TIMES 4"`, `"7 TIMES 5"`, and so on.

In the `MultPractice` class, the `getProblem` method returns a string in the format of *"first integer* `TIMES` *second integer"*. The `nextProblem` method updates the state of the `MultPractice` object to represent the next practice problem.

The following examples illustrate the behavior of the `MultPractice` class. Each table shows a code segment and the output that would be produced as the code is executed.

Example 1

Code segment	Output produced
`StudyPractice p1 = new MultPractice(7, 3);` `System.out.println(p1.getProblem());`	`7 TIMES 3`
`p1.nextProblem();` `System.out.println(p1.getProblem());`	`7 TIMES 4`
`p1.nextProblem();` `System.out.println(p1.getProblem());`	`7 TIMES 5`
`p1.nextProblem();` `System.out.println(p1.getProblem());`	`7 TIMES 6`

Example 2

Code segment	Output produced
`StudyPractice p2 = new MultPractice(4, 12);`	
`p2.nextProblem();`	
`System.out.println(p2.getProblem());`	`4 TIMES 13`
`System.out.println(p2.getProblem());`	`4 TIMES 13`
`p2.nextProblem();`	
`p2.nextProblem();`	
`System.out.println(p2.getProblem());`	`4 TIMES 15`
`p2.nextProblem();`	
`System.out.println(p2.getProblem());`	`4 TIMES 16`

Interface information for this question

```
public interface StudyPractice

String getProblem()
void nextProblem()
```

Write the complete `MultPractice` class. Your implementation must be consistent with the specifications and the given examples.

3. This question involves analyzing and modifying a string. The following `Phrase` class maintains a phrase in an instance variable and has methods that access and make changes to the phrase. You will write two methods of the `Phrase` class.

```
public class Phrase
{
   private String currentPhrase;

   /** Constructs a new Phrase object. */
   public Phrase(String p)
   {  currentPhrase = p;  }


   /** Returns the index of the nth occurrence of str in the current phrase;
    *  returns -1 if the nth occurrence does not exist.
    *  Precondition: str.length() > 0 and n > 0
    *  Postcondition: the current phrase is not modified.
    */
   public int findNthOccurrence(String str, int n)
   {  /* implementation not shown */  }

/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 *  If the nth occurrence does not exist, the current phrase is unchanged.
 *  Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
{  /* to be implemented in part (a) */  }


/** Returns the index of the last occurrence of str in the current phrase;
 *  returns -1 if str is not found.
 *  Precondition: str.length() > 0
 *  Postcondition: the current phrase is not modified.
 */
public int findLastOccurrence(String str)
{  /* to be implemented in part (b) */  }


/** Returns a string containing the current phrase. */
public String toString()
{  return currentPhrase;  }
}
```

(a) Write the `Phrase` method `replaceNthOccurrence`, which will replace the `nth` occurrence of the string `str` with the string `repl`. If the `nth` occurrence does not exist, `currentPhrase` remains unchanged.

Several examples of the behavior of the method `replaceNthOccurrence` are shown below.

Code segments	Output produced

`Phrase phrase1 = new Phrase("A cat ate late.");`
`phrase1.replaceNthOccurrence("at", 1, "rane");`
`System.out.println(phrase1);`
	`A crane ate late.`

`Phrase phrase2 = new Phrase("A cat ate late.");`
`phrase2.replaceNthOccurrence("at", 6, "xx");`
`System.out.println(phrase2);`
	`A cat ate late.`

`Phrase phrase3 = new Phrase("A cat ate late.");`
`phrase3.replaceNthOccurrence("bat", 2, "xx");`
`System.out.println(phrase3);`
	`A cat ate late.`

`Phrase phrase4 = new Phrase("aaaa");`
`phrase4.replaceNthOccurrence("aa", 1, "xx");`
`System.out.println(phrase4);`
	`xxaa`

`Phrase phrase5 = new Phrase("aaaa");`
`phrase5.replaceNthOccurrence("aa", 2, "bbb");`
`System.out.println(phrase5);`
	`abbba`

Class information for this question

```
public class Phrase

private String currentPhrase
public Phrase(String p)
public int findNthOccurrence(String str, int n)
public void replaceNthOccurrence(String str, int n, String repl)
public int findLastOccurrence(String str)
public String toString()
```

The `Phrase` class includes the method `findNthOccurrence`, which returns the `nth` occurrence of a given string. You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `replaceNthOccurrence` below.

```
/** Modifies the current phrase by replacing the nth occurrence of str with repl.
 *  If the nth occurrence does not exist, the current phrase is unchanged.
 *  Precondition: str.length() > 0 and n > 0
 */
public void replaceNthOccurrence(String str, int n, String repl)
```

(b) Write the `Phrase` method `findLastOccurrence`. This method finds and returns the index of the last occurrence of a given string in `currentPhrase`. If the given string is not found, `-1` is returned. The following tables show several examples of the behavior of the method `findLastOccurrence`.

```
Phrase phrase1 = new Phrase("A cat ate late.");
```

Method call	Value returned
`phrase1.findLastOccurrence("at")`	`11`
`phrase1.findLastOccurrence("cat")`	`2`
`phrase1.findLastOccurrence("bat")`	`-1`

Class information for this question

```
public class Phrase

private String currentPhrase
public Phrase(String p)
public int findNthOccurrence(String str, int n)
public void replaceNthOccurrence(String str, int n, String repl)
public int findLastOccurrence(String str)
public String toString()
```

You must use `findNthOccurrence` appropriately to receive full credit.

Complete method `findLastOccurrence` below.

```
/** Returns the index of the last occurrence of str in the current phrase;
 *  returns -1 if str is not found.
 *  Precondition: str.length() > 0
 *  Postcondition: the current phrase is not modified.
 */
public int findLastOccurrence(String str)
```

4. This question involves reasoning about a two-dimensional (2D) array of integers. You will write two static methods, both of which are in a single enclosing class named `Successors` (not shown). These methods process a 2D integer array that contains consecutive values. Each of these integers may be in any position in the 2D integer array. For example, the following 2D integer array with 3 rows and 4 columns contains the integers 5 through 16, inclusive.

2D Integer Array

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

The following `Position` class is used to represent positions in the integer array. The notation `(r,c)` will be used to refer to a `Position` object with row `r` and column `c`.

```
public class Position
{
    /** Constructs a Position object with row r and column c. */
    public Position(int r, int c)
    {  /* implementation not shown */  }

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

(a) Write a `static` method `findPosition` that takes an integer value and a 2D integer array and returns the position of the integer in the given 2D integer array. If the integer is not an element of the 2D integer array, the method returns `null`.

For example, assume that array `arr` is the 2D integer array shown at the beginning of the question.

- The call `findPosition(8, arr)` would return the `Position` object `(2,1)` because the value `8` appears in `arr` at row 2 and column 1.
- The call `findPosition(17, arr)` would return `null` because the value `17` does not appear in `arr`.

Complete method `findPosition` below.

```
/** Returns the position of num in intArr;
 *  returns null if no such element exists in intArr.
 *  Precondition: intArr contains at least one row.
 */
public static Position findPosition(int num, int[][] intArr)
```

(b) Write a `static` method `getSuccessorArray` that returns a 2D successor array of positions created from a given 2D integer array.

The *successor* of an integer value is the integer that is one greater than that value. For example, the successor of 8 is 9. A 2D *successor array* shows the position of the successor of each element in a given 2D integer array. The 2D successor array has the same dimensions as the given 2D integer array. Each element in the 2D successor array is the position (row, column) of the corresponding 2D integer array element's successor. The largest element in the 2D integer array does not have a successor in the 2D integer array, so its corresponding position in the 2D successor array is `null`.

The following diagram shows a 2D integer array and its corresponding 2D successor array. To illustrate the successor relationship, the values 8 and 9 in the 2D integer array are shaded. In the 2D successor array, the shaded element shows that the position of the successor of 8 is `(0,2)` in the 2D integer array. The largest value in the 2D integer array is 16, so its corresponding element in the 2D successor array is `null`.

2D Integer Array

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

2D Successor Array

	0	1	2	3
0	(1,1)	(1,3)	(0,3)	(1,2)
1	(2,2)	null	(1,0)	(2,3)
2	(0,0)	(0,2)	(2,0)	(2,1)

Class information for this question

```
public class Position

public Position(int r, int c)

public class Successors

public static Position findPosition(int num, int[][] intArr)
public static Position[][] getSuccessorArray(int[][] intArr)
```

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

Complete method `getSuccessorArray` below.

```
/** Returns a 2D successor array as described in part (b) constructed from intArr.
 *  Precondition: intArr contains at least one row and contains consecutive values.
 *                Each of these integers may be in any position in the 2D array.
 */
public static Position[][] getSuccessorArray(int[][] intArr)
```

2016 Free Response Questions

1. This question involves the implementation and extension of a `RandomStringChooser` class.

   (a) A `RandomStringChooser` object is constructed from an array of non-`null` `String` values. When the object is first constructed, all of the strings are considered available. The `RandomStringChooser` class has a `getNext` method, which has the following behavior. A call to `getNext` returns a randomly chosen string from the available strings in the object. Once a particular string has been returned from a call to `getNext`, it is no longer available to be returned from subsequent calls to `getNext`. If no strings are available to be returned, `getNext` returns `"NONE"`.

   The following code segment shows an example of the behavior of `RandomStringChooser`.

```
String[] wordArray = {"wheels", "on", "the", "bus"};
RandomStringChooser sChooser = new RandomStringChooser(wordArray);
for (int k = 0; k < 6; k++)
{
   System.out.print(sChooser.getNext() + " ");
}
```

   One possible output is shown below. Because `sChooser` has only four strings, the string `"NONE"` is printed twice.

```
bus the wheels on NONE NONE
```

Write the entire `RandomStringChooser` class. Your implementation must include an appropriate constructor and any necessary methods. Any instance variables must be `private`. The code segment in the example above should have the indicated behavior (that is, it must compile and produce a result like the possible output shown). Neither the constructor nor any of the methods should alter the parameter passed to the constructor, but your implementation may copy the contents of the array.

(b) The following partially completed `RandomLetterChooser` class is a subclass of the `RandomStringChooser` class. You will write the constructor for the `RandomLetterChooser` class.

```
public class RandomLetterChooser extends RandomStringChooser
{
    /** Constructs a random letter chooser using the given string str.
     *  Precondition: str contains only letters.
     */
    public RandomLetterChooser(String str)
    {  /* to be implemented in part (b) */  }

    /** Returns an array of single-letter strings.
     *  Each of these strings consists of a single letter from str. Element k
     *  of the returned array contains the single letter at position k of str.
     *  For example, getSingleLetters("cat") returns the
     *  array { "c", "a", "t" }.
     */
    public static String[] getSingleLetters(String str)
    {  /* implementation not shown */  }
}
```

The following code segment shows an example of using `RandomLetterChooser`.

```
RandomLetterChooser letterChooser = new RandomLetterChooser("cat");
for (int k = 0; k < 4; k++)
{
   System.out.print(letterChooser.getNext());
}
```

The code segment will print the three letters in `"cat"` in one of the possible orders. Because there are only three letters in the original string, the code segment prints `"NONE"` the fourth time through the loop. One possible output is shown below.

```
actNONE
```

Assume that the `RandomStringChooser` class that you wrote in part (a) has been implemented correctly and that `getSingleLetters` works as specified. You must use `getSingleLetters` appropriately to receive full credit.

Complete the `RandomLetterChooser` constructor below.

```
/** Constructs a random letter chooser using the given string str.
 *  Precondition: str contains only letters.
 */
public RandomLetterChooser(String str)
```

2. This question involves two classes that are used to process log messages. A list of sample log messages is given below.

```
CLIENT3:security alert - repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

Log messages have the format *machineId*:*description*, where *machineId* identifies the computer and *description* describes the event being logged. Exactly one colon (":") appears in a log message. There are no blanks either immediately before or immediately after the colon.

The following `LogMessage` class is used to represent a log message.

```
public class LogMessage
{
   private String machineId;
   private String description;

   /** Precondition: message is a valid log message. */
   public LogMessage(String message)
   {  /* to be implemented in part (a) */  }

   /** Returns true if the description in this log message properly contains keyword;
    *          false otherwise.
    */
   public boolean containsWord(String keyword)
   {  /* to be implemented in part (b) */  }

   public String getMachineId()
   {  return machineId;  }

   public String getDescription()
   {  return description;  }

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

(a) Write the constructor for the `LogMessage` class. It must initialize the private data of the object so that `getMachineId` returns the *machineId* part of the message and `getDescription` returns the *description* part of the message.

Complete the `LogMessage` constructor below.

```
/** Precondition: message is a valid log message. */
public LogMessage(String message)
```

(b) Write the `LogMessage` method `containsWord`, which returns `true` if the description in the log message *properly contains* a given keyword and returns `false` otherwise.

A description *properly contains* a keyword if all three of the following conditions are true.

- the keyword is a substring of the description;
- the keyword is either at the beginning of the description or it is immediately preceded by a space;
- the keyword is either at the end of the description or it is immediately followed by a space.

The following tables show several examples. The descriptions in the left table properly contain the keyword `"disk"`. The descriptions in the right table do not properly contain the keyword `"disk"`.

Descriptions that properly contain `"disk"`

`"disk"`
`"error on disk"`
`"error on /dev/disk disk"`
`"error on disk DSK1"`

Descriptions that do not properly contain `"disk"`

`"DISK"`
`"error on disk3"`
`"error on /dev/disk"`
`"diskette"`

Assume that the `LogMessage` constructor works as specified, regardless of what you wrote in part (a). Complete method `containsWord` below.

```
/** Returns true if the description in this log message properly contains keyword;
 *          false otherwise.
 */
public boolean containsWord(String keyword)
```

(c) The `SystemLog` class represents a list of `LogMessage` objects and provides a method that removes and returns a list of all log messages (if any) that properly contain a given keyword. The messages in the returned list appear in the same order in which they originally appeared in the system log. If no message properly contains the keyword, an empty list is returned. The declaration of the `SystemLog` class is shown below.

```
public class SystemLog
{
    /** Contains all the entries in this system log.
     *  Guaranteed not to be null and to contain only non-null entries.
     */
    private List<LogMessage> messageList;

    /** Removes from the system log all entries whose descriptions properly contain keyword,
     *  and returns a list (possibly empty) containing the removed entries.
     *  Postcondition:
     *    – Entries in the returned list properly contain keyword and
     *      are in the order in which they appeared in the system log.
     *    – The remaining entries in the system log do not properly contain keyword and
     *      are in their original order.
     *    – The returned list is empty if no messages properly contain keyword.
     */
    public List<LogMessage> removeMessages(String keyword)
    {  /* to be implemented in part (c) */  }

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

Write the `SystemLog` method `removeMessages`, which removes from the system log all entries whose descriptions properly contain `keyword` and returns a list of the removed entries in their original order. For example, assume that `theLog` is a `SystemLog` object initially containing six `LogMessage` objects representing the following list of log messages.

```
CLIENT3:security alert - repeated login failures
Webserver:disk offline
SERVER1:file not found
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
Webserver:error on /dev/disk
```

The call `theLog.removeMessages("disk")` would return a list containing the `LogMessage` objects representing the following log messages.

```
Webserver:disk offline
SERVER2:read error on disk DSK1
SERVER1:write error on disk DSK2
```

After the call, `theLog` would contain the following log messages.

```
CLIENT3:security alert - repeated login failures
SERVER1:file not found
Webserver:error on /dev/disk
```

Assume that the `LogMessage` class works as specified, regardless of what you wrote in parts (a) and (b). You must use `containsWord` appropriately to receive full credit.

Complete method `removeMessages` below.

```
/** Removes from the system log all entries whose descriptions properly contain keyword,
 *  and returns a list (possibly empty) containing the removed entries.
 *  Postcondition:
 *    – Entries in the returned list properly contain keyword and
 *      are in the order in which they appeared in the system log.
 *    – The remaining entries in the system log do not properly contain keyword and
 *      are in their original order.
 *    – The returned list is empty if no messages properly contain keyword.
 */
public List<LogMessage> removeMessages(String keyword)
```

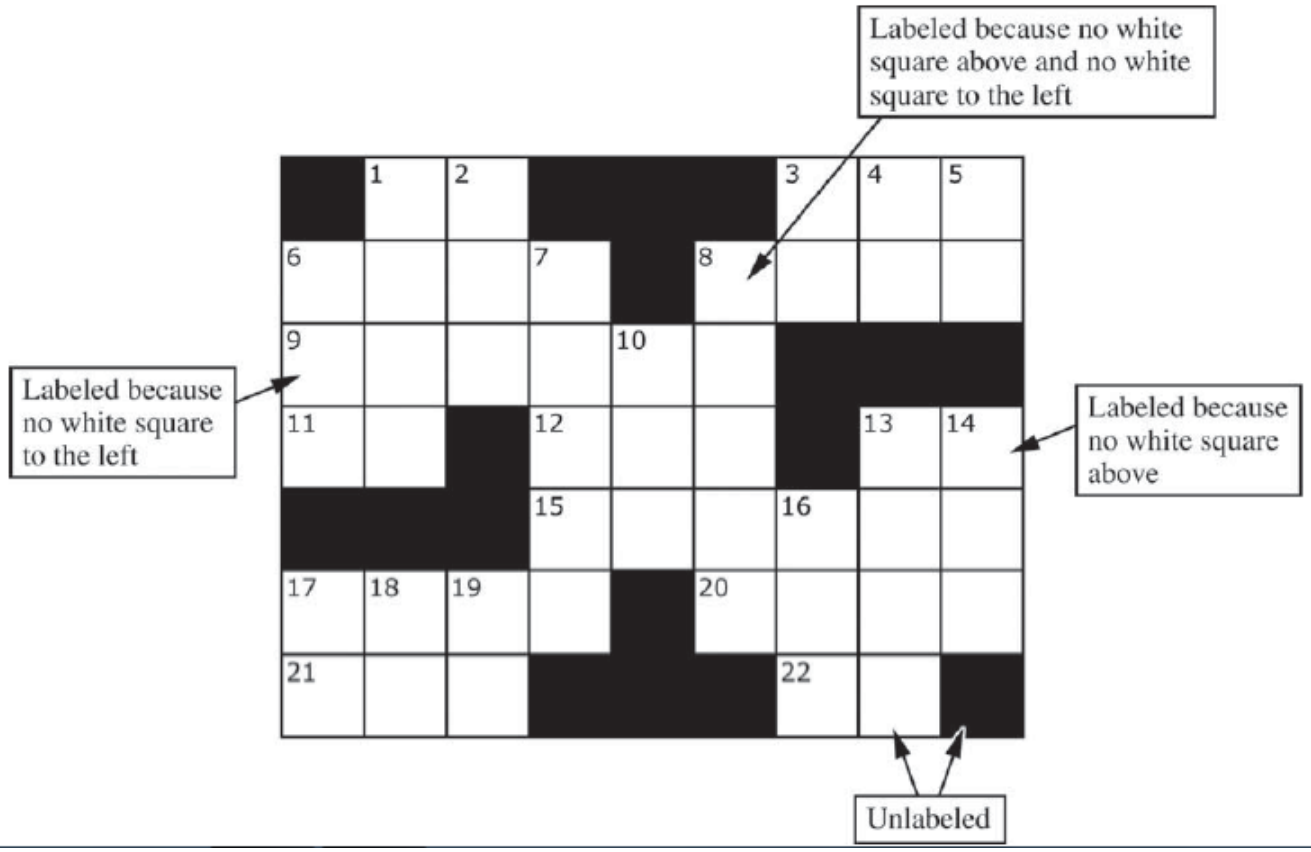
3. A crossword puzzle grid is a two-dimensional rectangular array of black and white squares. Some of the white squares are labeled with a positive number according to the *crossword labeling rule*.

   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.

```
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)
```

4. This question involves the process of taking a list of words, called `wordList`, and producing a formatted string of a specified length. The list `wordList` contains at least two words, consisting of letters only.

   When the formatted string is constructed, spaces are placed in the gaps between words so that as many spaces as possible are evenly distributed to each gap. The equal number of spaces inserted into each gap is referred to as the *basic gap width.* Any *leftover spaces* are inserted one at a time into the gaps from left to right until there are no more leftover spaces.

   The following three examples illustrate these concepts. In each example, the list of words is to be placed into a formatted string of length 20.

   Example 1: `wordList: ["AP", "COMP", "SCI", "ROCKS"]`

   Total number of letters in words: 14
   Number of gaps between words: 3
   Basic gap width: 2
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	P			C	O	M	P			S	C	I			R	O	C	K	S

   Example 2: `wordList: ["GREEN", "EGGS", "AND", "HAM"]`

   Total number of letters in words: 15
   Number of gaps between words: 3
   Basic gap width: 1
   Leftover spaces: 2

   The leftover spaces are inserted one at a time between the words from left to right until there are no more leftover spaces. In this example, the first two gaps get an extra space.

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
G	R	E	E	N			E	G	G	S			A	N	D		H	A	M

   Example 3: `wordList: ["BEACH", "BALL"]`

   Total number of letters in words: 9
   Number of gaps between words: 1
   Basic gap width: 11
   Leftover spaces: 0

   Formatted string:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B	E	A	C	H												B	A	L	L

You will implement three `static` methods in a class named `StringFormatter` that is not shown.

(a) Write the `StringFormatter` method `totalLetters`, which returns the total number of letters in the words in its parameter `wordList`. For example, if the variable `List<String> words` is `["A", "frog", "is"]`, then the call `StringFormatter.totalLetters(words)` returns `7`. You may assume that all words in `wordList` consist of one or more letters.

Complete method `totalLetters` below.

```
/** Returns the total number of letters in wordList.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 */
public static int totalLetters(List<String> wordList)
```

(b) Write the `StringFormatter` method `basicGapWidth`, which returns the basic gap width as defined earlier.

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

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

Complete method `basicGapWidth` below.

```
/** Returns the basic gap width when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
```

(c) Write the `StringFormatter` method `format`, which returns the formatted string as defined earlier. The `StringFormatter` class also contains a method called `leftoverSpaces`, which has already been implemented. This method returns the number of leftover spaces as defined earlier and is shown below.

```
/** Returns the number of leftover spaces when wordList is used to produce
 *  a formatted string of formattedLen characters.
 *  Precondition: wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 */
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
{  /* implementation not shown */  }
```

Class information for this question

```
public class StringFormatter

public static int totalLetters(List<String> wordList)
public static int basicGapWidth(List<String> wordList,
                                int formattedLen)
public static int leftoverSpaces(List<String> wordList,
                                 int formattedLen)
public static String format(List<String> wordList, int formattedLen)
```

Assume that `basicGapWidth` works as specified, regardless of what you wrote in part (b). You must use `basicGapWidth` and `leftoverSpaces` appropriately to receive full credit.

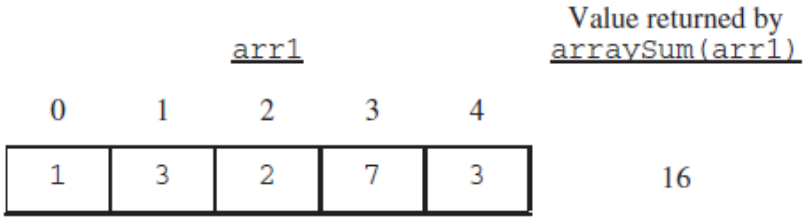
Complete method `format` below.

```
/** Returns a formatted string consisting of the words in wordList separated by spaces.
 *  Precondition: The wordList contains at least two words, consisting of letters only.
 *                formattedLen is large enough for all the words and gaps.
 *  Postcondition: All words in wordList appear in the formatted string.
 *    - The words appear in the same order as in wordList.
 *    - The number of spaces between words is determined by basicGapWidth and the
 *      distribution of leftoverSpaces from left to right, as described in the question.
 */
public static String format(List<String> wordList, int formattedLen)
```

2015 Free Response Questions

1. This question involves reasoning about one-dimensional and two-dimensional arrays of integers. You will write three static methods, all of which are in a single enclosing class, named `DiverseArray` (not shown). The first method returns the sum of the values of a one-dimensional array; the second method returns an array that represents the sums of the rows of a two-dimensional array; and the third method analyzes row sums.

   (a) Write a `static` method `arraySum` that calculates and returns the sum of the entries in a specified one-dimensional array. The following example shows an array `arr1` and the value returned by a call to `arraySum`.



Complete method `arraySum` below.

```
/** Returns the sum of the entries in the one-dimensional array arr.
 */
public static int arraySum(int[] arr)
```

(b) Write a `static` method `rowSums` that calculates the sums of each of the rows in a given two-dimensional array and returns these sums in a one-dimensional array. The method has one parameter, a two-dimensional array `arr2D` of `int` values. The array is in row-major order: `arr2D[r][c]` is the entry at row `r` and column `c`. The method returns a one-dimensional array with one entry for each row of `arr2D` such that each entry is the sum of the corresponding row in `arr2D`. As a reminder, each row of a two-dimensional array is a one-dimensional array.

For example, if `mat1` is the array represented by the following table, the call `rowSums(mat1)` returns the array `{16, 32, 28, 20}`.

`mat1`

	0	1	2	3	4
0	1	3	2	7	3
1	10	10	4	6	2
2	5	3	5	9	6
3	7	6	4	2	1

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

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

Complete method `rowSums` below.

```
/** Returns a one-dimensional array in which the entry at index k is the sum of
 *  the entries of row k of the two-dimensional array arr2D.
 */
public static int[] rowSums(int[][] arr2D)
```

(c) A two-dimensional array is *diverse* if no two of its rows have entries that sum to the same value. In the following examples, the array `mat1` is diverse because each row sum is different, but the array `mat2` is not diverse because the first and last rows have the same sum.

`mat1`

	0	1	2	3	4	Row sums
0	1	3	2	7	3	16
1	10	10	4	6	2	32
2	5	3	5	9	6	28
3	7	6	4	2	1	20

`mat2`

	0	1	2	3	4	Row sums
0	1	1	5	3	4	14
1	12	7	6	1	9	35
2	8	11	10	2	5	36
3	3	2	3	0	6	14

Write a `static` method `isDiverse` that determines whether or not a given two-dimensional array is diverse. The method has one parameter: a two-dimensional array `arr2D` of `int` values. The method should return `true` if all the row sums in the given array are unique; otherwise, it should return `false`. In the arrays shown above, the call `isDiverse(mat1)` returns `true` and the call `isDiverse(mat2)` returns `false`.

Methods written in this question

```
public static int arraySum(int[] arr)
public static int[] rowSums(int[][] arr2D)
public static boolean isDiverse(int[][] arr2D)
```

Assume that `arraySum` and `rowSums` work as specified, regardless of what you wrote in parts (a) and (b). You must use `rowSums` appropriately to receive full credit.

Complete method `isDiverse` below.

```
/** Returns true  if all rows in arr2D have different row sums;
 *          false otherwise.
 */
public static boolean isDiverse(int[][] arr2D)
```

2. Consider a guessing game in which a player tries to guess a hidden word. The hidden word contains only capital letters and has a length known to the player. A guess contains only capital letters and has the same length as the hidden word.

   After a guess is made, the player is given a hint that is based on a comparison between the hidden word and the guess. Each position in the hint contains a character that corresponds to the letter in the same position in the guess. The following rules determine the characters that appear in the hint.

If the letter in the guess is …	**the corresponding character in the hint is**
also in the same position in the hidden word,	the matching letter
also in the hidden word, but in a different position,	"+"
not in the hidden word,	"*"

   The `HiddenWord` class will be used to represent the hidden word in the game. The hidden word is passed to the constructor. The class contains a method, `getHint`, that takes a guess and produces a hint.

   For example, suppose the variable `puzzle` is declared as follows.

```
HiddenWord puzzle = new HiddenWord("HARPS");
```

The following table shows several guesses and the hints that would be produced.

Call to `getHint`	String returned
`puzzle.getHint("AAAAA")`	`"+A+++"`
`puzzle.getHint("HELLO")`	`"H****"`
`puzzle.getHint("HEART")`	`"H*++*"`
`puzzle.getHint("HARMS")`	`"HAR*S"`
`puzzle.getHint("HARPS")`	`"HARPS"`

Write the complete `HiddenWord` class, including any necessary instance variables, its constructor, and the method, `getHint`, described above. You may assume that the length of the guess is the same as the length of the hidden word.

3. 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;  }
}
```

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.

```
numRows: 6

numCols: 5

entries: [ row: 1 col: 4 value: 4 ] [ row: 2 col: 0 value: 1 ] [ row: 3 col: 1 value: -9 ] [ row: 1 col: 1 value: 5 ]
```

(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()
 */
public int getValueAt(int row, int col)
```

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

- 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.

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

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

```
numRows: 6

numCols: 5

entries: [ row: 1 col: 4 value: 4 ] [ row: 2 col: 0 value: 1 ] [ row: 3 col: 1 value: -9 ] [ row: 1 col: 1 value: 5 ]
```

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

```
numRows: 6

numCols: 4

entries: [ row: 1 col: 3 value: 4 ] [ row: 2 col: 0 value: 1 ]
```

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)
```

4. This question involves the design of an interface, writing a class that implements the interface, and writing a method that uses the interface.

   (a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

   Write an interface named `NumberGroup` that represents a group of integers. The interface should have a single `contains` method that determines if a given integer is in the group. For example, if `group1` is of type `NumberGroup`, and it contains only the two numbers `-5` and `3`, then `group1.contains(-5)` would return `true`, and `group1.contains(2)` would return `false`.

   Write the complete `NumberGroup` interface. It must have exactly one method.

(b) A *range* represents a number group that contains all (and only) the integers between a minimum value and a maximum value, inclusive.

Write the `Range` class, which is a `NumberGroup`. The `Range` class represents the group of `int` values that range from a given minimum value up through a given maximum value, inclusive. For example, the declaration

```
NumberGroup range1 = new Range(-3, 2);
```

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete `Range` class. Include all necessary instance variables and methods as well as a constructor that takes two `int` parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

(c) The `MultipleGroups` class (not shown) represents a collection of `NumberGroup` objects and is a `NumberGroup`. The `MultipleGroups` class stores the number groups in the instance variable `groupList` (shown below), which is initialized in the constructor.

```
private List<NumberGroup> groupList;
```

Write the `MultipleGroups` method `contains`. The method takes an integer and returns `true` if and only if the integer is contained in one or more of the number groups in `groupList`.

For example, suppose `multiple1` has been declared as an instance of `MultipleGroups` and consists of the three ranges created by the calls `new Range(5, 8)`, `new Range(10, 12)`, and `new Range(1, 6)`. The following table shows the results of several calls to `contains`.

Call	Result
`multiple1.contains(2)`	`true`
`multiple1.contains(9)`	`false`
`multiple1.contains(6)`	`true`

Complete method `contains` below.

```
/** Returns true if at least one of the number groups in this multiple group contains num;
 *          false otherwise.
 */
public boolean contains(int num)
```

2014 Free Response Questions

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. This question involves reasoning about strings made up of uppercase letters. You will implement two related methods that appear in the same class (not shown). The first method takes a single string parameter and returns a scrambled version of that string. The second method takes a list of strings and modifies the list by scrambling each entry in the list. Any entry that cannot be scrambled is removed from the list.

   (a) Write the method `scrambleWord`, which takes a given word and returns a string that contains a scrambled version of the word according to the following rules.

   - The scrambling process begins at the first letter of the word and continues from left to right.
   - If two consecutive letters consist of an "A" followed by a letter that is not an "A", then the two letters are swapped in the resulting string.
   - Once the letters in two adjacent positions have been swapped, neither of those two positions can be involved in a future swap.

The following table shows several examples of words and their scrambled versions.

`word`	Result returned by `scrambleWord(word)`
"TAN"	"TNA"
"ABRACADABRA"	"BARCADABARA"
"WHOA"	"WHOA"
"AARDVARK"	"ARADVRAK"
"EGGS"	"EGGS"
"A"	"A"
""	""

Complete method `scrambleWord` below.

```
/** Scrambles a given word.
 *  @param word the word to be scrambled
 *  @return the scrambled word (possibly equal to word)
 *  Precondition: word is either an empty string or contains only uppercase letters.
 *  Postcondition: the string returned was created from word as follows:
 *    – the word was scrambled, beginning at the first letter and continuing from left to right
 *    – two consecutive letters consisting of "A" followed by a letter that was not "A" were swapped
 *    – letters were swapped at most once
 */
public static String scrambleWord(String word)
```

(b) Write the method `scrambleOrRemove`, which replaces each word in the parameter `wordList` with its scrambled version and removes any words that are unchanged after scrambling. The relative ordering of the entries in `wordList` remains the same as before the call to `scrambleOrRemove`.

The following example shows how the contents of `wordList` would be modified as a result of calling `scrambleOrRemove`.

Before the call to `scrambleOrRemove`:

	0	1	2	3	4
`wordList`	"TAN"	"ABRACADABRA"	"WHOA"	"APPLE"	"EGGS"

After the call to `scrambleOrRemove`:

	0	1	2
`wordList`	"TNA"	"BARCADABARA"	"PAPLE"

Assume that `scrambleWord` is in the same class as `scrambleOrRemove` and works as specified, regardless of what you wrote in part (a).

Complete method `scrambleOrRemove` below.

```
/** Modifies wordList by replacing each word with its scrambled
 *  version, removing any words that are unchanged as a result of scrambling.
 *  @param wordList the list of words
 *  Precondition: wordList contains only non-null objects
 *  Postcondition:
 *    – all words unchanged by scrambling have been removed from wordList
 *    – each of the remaining words has been replaced by its scrambled version
 *    – the relative ordering of the entries in wordList is the same as it was
 *        before the method was called
 */
public static void scrambleOrRemove(List<String> wordList)
```

2. This question involves reasoning about the GridWorld case study. Reference materials are provided in the appendixes.

   A `Director` is a type of `Rock` that has the following characteristics.
   - A `Director` has an initial color of `Color.RED` and alternates between `Color.RED` and `Color.GREEN` each time it acts.
   - If the color of a `Director` is `Color.GREEN` when it begins to act, it will cause any `Actor` objects in its neighboring cells to turn 90 degrees to their right.

   Write the complete `Director` class, including the zero-parameter constructor and any necessary instance variables and methods. Assume that the `Color` class has been imported.

3. A student in a school is represented by the following class.

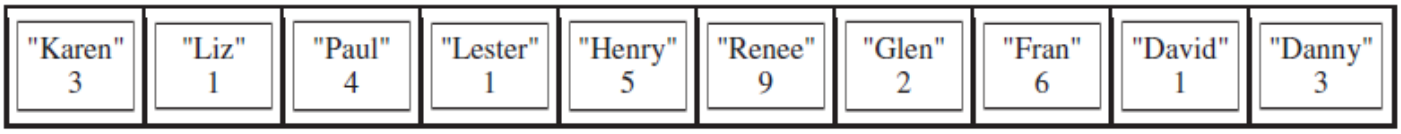
```
public class Student
{
  /** Returns the name of this Student. */
  public String getName()
  {  /* implementation not shown */  }

  /** Returns the number of times this Student has missed class. */
  public int getAbsenceCount()
  {  /* implementation not shown */  }

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

The class `SeatingChart`, shown below, uses a two-dimensional array to represent the seating arrangement of students in a classroom. The seats in the classroom are in a rectangular arrangement of rows and columns.

```
public class SeatingChart
{
  /** seats[r][c] represents the Student in row r and column c in the classroom. */
  private Student[][] seats;

  /** Creates a seating chart with the given number of rows and columns from the students in
   *  studentList. Empty seats in the seating chart are represented by null.
   *  @param rows the number of rows of seats in the classroom
   *  @param cols the number of columns of seats in the classroom
   *  Precondition: rows > 0; cols > 0;
   *                rows * cols >= studentList.size()
   *  Postcondition:
   *    - Students appear in the seating chart in the same order as they appear
   *      in studentList, starting at seats[0][0].
   *    - seats is filled column by column from studentList, followed by any
   *      empty seats (represented by null).
   *    - studentList is unchanged.
   */
  public SeatingChart(List<Student> studentList,
                      int rows, int cols)
  {  /* to be implemented in part (a) */  }

  /** Removes students who have more than a given number of absences from the
   *  seating chart, replacing those entries in the seating chart with null
   *  and returns the number of students removed.
   *  @param allowedAbsences an integer >= 0
   *  @return number of students removed from seats
   *  Postcondition:
   *    - All students with allowedAbsences or fewer are in their original positions in seats.
   *    - No student in seats has more than allowedAbsences absences.
   *    - Entries without students contain null.
   */
  public int removeAbsentStudents(int allowedAbsences)
  {  /* to be implemented in part (b) */  }

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

(a) Write the constructor for the `SeatingChart` class. The constructor initializes the `seats` instance variable to a two-dimensional array with the given number of rows and columns. The students in `studentList` are copied into the seating chart in the order in which they appear in `studentList`. The students are assigned to consecutive locations in the array `seats`, starting at `seats[0][0]` and filling the array column by column. Empty seats in the seating chart are represented by `null`.

For example, suppose a variable `List<Student> roster` contains references to `Student` objects in the following order.



A `SeatingChart` object created with the call `new SeatingChart(roster, 3, 4)` would have `seats` initialized with the following values.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	`null`
2	"Paul" 4	"Renee" 9	"David" 1	`null`

Complete the `SeatingChart` constructor below.

```
/** Creates a seating chart with the given number of rows and columns from the students in
 *  studentList. Empty seats in the seating chart are represented by null.
 *  @param rows the number of rows of seats in the classroom
 *  @param cols the number of columns of seats in the classroom
 *  Precondition: rows > 0; cols > 0;
 *                rows * cols >= studentList.size()
 *  Postcondition:
 *    - Students appear in the seating chart in the same order as they appear
 *      in studentList, starting at seats[0][0].
 *    - seats is filled column by column from studentList, followed by any
 *      empty seats (represented by null).
 *    - studentList is unchanged.
 */
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
```

(b) Write the `removeAbsentStudents` method, which removes students who have more than a given number of absences from the seating chart and returns the number of students that were removed. When a student is removed from the seating chart, a `null` is placed in the entry for that student in the array `seats`. For example, suppose the variable `SeatingChart introCS` has been created such that the array `seats` contains the following entries showing both students and their number of absences.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	"Henry" 5	"Fran" 6	null
2	"Paul" 4	"Renee" 9	"David" 1	null

After the call `introCS.removeAbsentStudents(4)` has executed, the array `seats` would contain the following values and the method would return the value 3.

	0	1	2	3
0	"Karen" 3	"Lester" 1	"Glen" 2	"Danny" 3
1	"Liz" 1	**null**	**null**	null
2	"Paul" 4	**null**	"David" 1	null

Class information repeated from the beginning of the question:

```
public class Student

public String getName()
public int getAbsenceCount()

public class SeatingChart

private Student[][] seats
public SeatingChart(List<Student> studentList,
                    int rows, int cols)
public int removeAbsentStudents(int allowedAbsences)
```

Complete method `removeAbsentStudents` below.

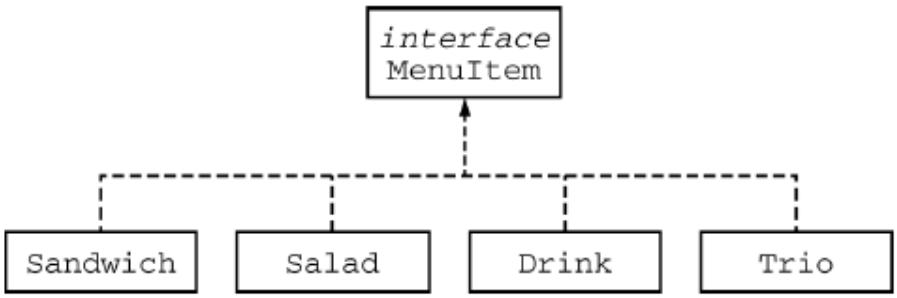
```
/** Removes students who have more than a given number of absences from the
 *  seating chart, replacing those entries in the seating chart with null
 *  and returns the number of students removed.
 *  @param allowedAbsences an integer >= 0
 *  @return number of students removed from seats
 *  Postcondition:
 *    - All students with allowedAbsences or fewer are in their original positions in seats.
 *    - No student in seats has more than allowedAbsences absences.
 *    - Entries without students contain null.
 */
public int removeAbsentStudents(int allowedAbsences)
```

4. 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.

Sandwich	Salad	Drink
"Cheeseburger" 2.75	"Spinach Salad" 1.25	"Orange Soda" 1.25
"Club Sandwich" 2.75	"Coleslaw" 1.25	"Cappuccino" 3.50

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.

2013 Free Response Questions

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 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.

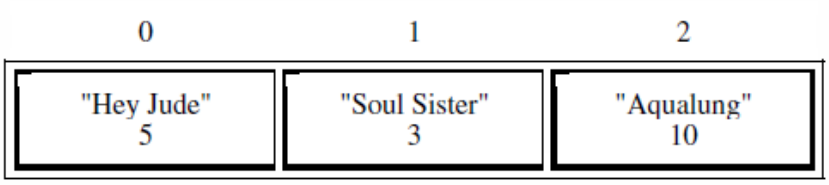
```
public class DownloadInfo
{
  /** Creates a new instance with the given unique title and sets the
   *  number of times downloaded to 1.
   *  @param title the unique title of the downloaded song
   */
  public DownloadInfo(String title)
  {  /* implementation not shown */  }

  /** @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.

```
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

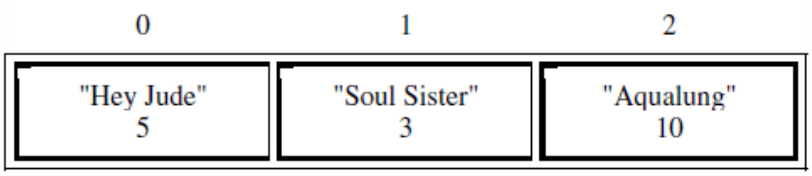
```
public class DownloadInfo

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.

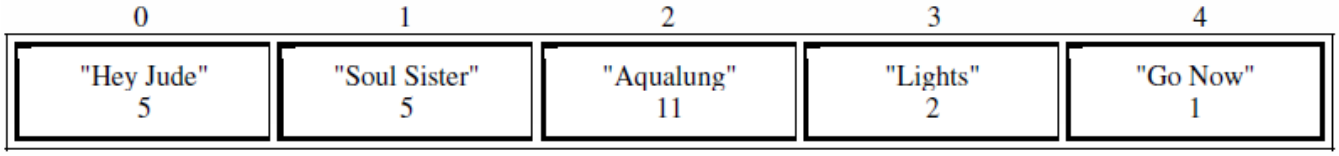
For example, suppose variable `webMusicB` refers to an instance of `MusicDownloads` and that the table below represents the contents of the instance variable `downloadList`.



Assume that the variable `List<String> 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.

```
/** 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 )
```

2. A multiplayer game called Token Pass has the following rules.

Each player begins with a random number of tokens (at least 1, but no more than 10) that are placed on a linear game board. There is one position on the game board for each player. After the game board has been filled, a player is randomly chosen to begin the game. Each position on the board is numbered, starting with 0.

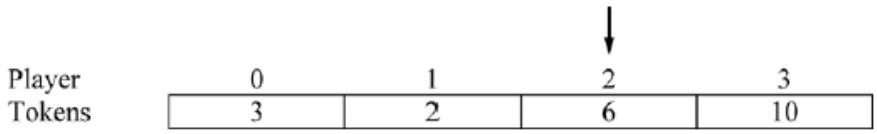
The following rules apply for a player's turn.

- The tokens are collected and removed from the game board at that player's position.
- The collected tokens are distributed one at a time, to each player, beginning with the next player in order of increasing position.
- If there are still tokens to distribute after the player at the highest position gets a token, the next token will be distributed to the player at position 0.
- The distribution of tokens continues until there are no more tokens to distribute.

The Token Pass game board is represented by an array of integers. The indexes of the array represent the player positions on the game board, and the corresponding values in the array represent the number of tokens that each player has. The following example illustrates one player's turn.

Example

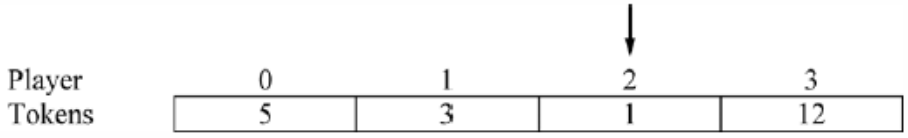
The following represents a game with 4 players. The player at position 2 was chosen to go first.



The tokens at position 2 are collected and distributed as follows.

1st token - to position 3 (The highest position is reached, so the next token goes to position 0.)
2nd token - to position 0
3rd token - to position 1
4th token - to position 2
5th token - to position 3 (The highest position is reached, so the next token goes to position 0.)
6th token - to position 0

After player 2's turn, the values in the array will be as follows.



The Token Pass game is represented by the `TokenPass` class.

```
public class TokenPass
{
  private int[] board;
  private int currentPlayer;


  /** Creates the board array to be of size playerCount and fills it with
   *  random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
   *  random integer value in the range between 0 and playerCount-1, inclusive.
   *  @param playerCount the number of players
   */
  public TokenPass(int playerCount)
  {  /* to be implemented in part (a) */  }



  /** Distributes the tokens from the current player's position one at a time to each player in
   *  the game. Distribution begins with the next position and continues until all the tokens
   *  have been distributed. If there are still tokens to distribute when the player at the
   *  highest position is reached, the next token will be distributed to the player at position 0.
   *  Precondition: the current player has at least one token.
   *  Postcondition: the current player has not changed.
   */
  public void distributeCurrentPlayerTokens()
  {  /* to be implemented in part (b) */  }

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

(a) Write the constructor for the `TokenPass` class. The parameter `playerCount` represents the number of players in the game. The constructor should create the `board` array to contain `playerCount` elements and fill the array with random numbers between 1 and 10, inclusive. The constructor should also initialize the instance variable `currentPlayer` to a random number between 0 and `playerCount-1`, inclusive.

Complete the `TokenPass` constructor below.

```
/** Creates the board array to be of size playerCount and fills it with
 *  random integer values from 1 to 10, inclusive. Initializes currentPlayer to a
 *  random integer value in the range between 0 and playerCount-1, inclusive.
 *  @param playerCount the number of players
 */
public TokenPass(int playerCount)
```

(b) Write the `distributeCurrentPlayerTokens` method.

The tokens are collected and removed from the game board at the current player's position. These tokens are distributed, one at a time, to each player, beginning with the next higher position, until there are no more tokens to distribute.

Class information repeated from the beginning of the question

```
public class TokenPass

private int[] board
private int currentPlayer
public TokenPass(int playerCount)
public void distributeCurrentPlayerTokens()
```

Complete method `distributeCurrentPlayerTokens` below.

```
/** Distributes the tokens from the current player's position one at a time to each player in
 *  the game. Distribution begins with the next position and continues until all the tokens
 *  have been distributed. If there are still tokens to distribute when the player at the
 *  highest position is reached, the next token will be distributed to the player at position 0.
 *  Precondition: the current player has at least one token.
 *  Postcondition: the current player has not changed.
 */
public void distributeCurrentPlayerTokens()
```

3. This question involves reasoning about the GridWorld case study. Reference materials are provided in the appendixes. In part (a) you will write a method to return an array list of all empty locations in a given grid. In part (b) you will write the class for a new type of `Critter`.

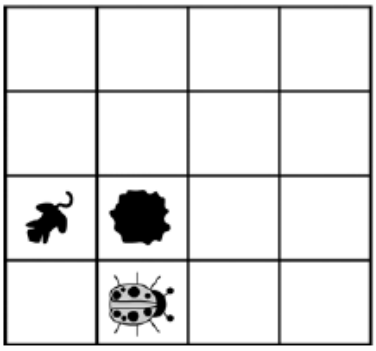
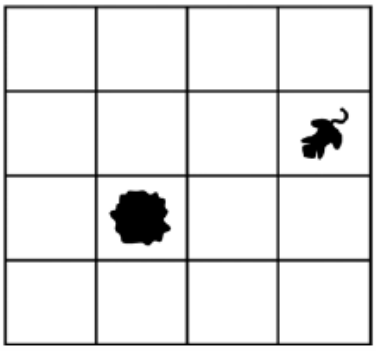
   (a) The `GridWorldUtilities` class contains `static` methods. A partial declaration of the `GridWorldUtilities` class is shown below.

```
public class GridWorldUtilities
{
   /** Gets all the locations in grid that do not contain objects.
    *  @param grid a reference to a BoundedGrid object
    *  @return an array list (possibly empty) of empty locations in grid.
    *          The size of the returned list is 0 if there are no empty locations in grid.
    *          Each empty location in grid should appear exactly once in the returned list.
    */
   public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
   {  /* to be implemented in part (a) */  }

   // There may be instance variables that are not shown.
}
```

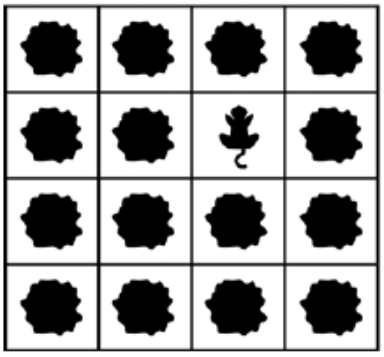
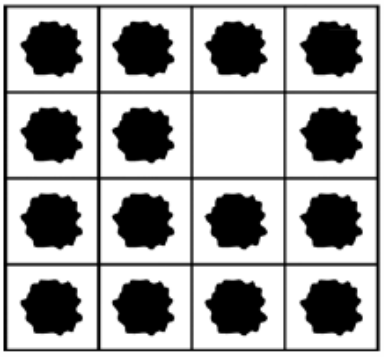
   Write the `GridWorldUtilities` method `getEmptyLocations`. If there are no empty locations in `grid`, the method returns an empty array list. Otherwise, it returns an array list of all empty locations in `grid`. Each empty location should appear exactly once in the array list.

   (b) A `JumpingCritter` acts like a `Critter`, except that it moves by jumping to a randomly selected empty location in its grid. If there are no empty locations, the `JumpingCritter` removes itself from the grid.

   The following diagram shows an example of a jumping critter that is able to move to an empty location. Example World #1 is shown below on the left. After the jumping critter at location (2, 0) acts, the world shown below on the right is one possible result.

EXAMPLE WORLD #1	POSSIBLE WORLD AFTER ACT
	
A jumping critter is in location (2, 0).	The jumping critter has eaten the bug that was in location (3, 1) and has moved to location (1, 3).

   Example World #2 is shown below on the left. After the jumping critter at location (1, 2) acts, the world shown below on the right is the result.

EXAMPLE WORLD #2	WORLD AFTER ACT
	
A jumping critter is in location (1, 2).	The jumping critter removed itself from the grid because no empty locations were available.

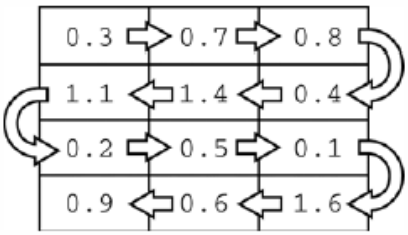
Class information repeated from the beginning of the question

```
public class GridWorldUtilities

public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
```

Assume that the `GridWorldUtilities` `getEmptyLocations` method works as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality of this method will not receive full credit.

Write the complete `JumpingCritter` class. Do NOT override the `act` method. Remember that your design must not violate the postconditions of the methods of the `Critter` class.

4. A telescope scans a rectangular area of the night sky and collects the data into a 1-dimensional array. Each data value scanned is a number representing the amount of light detected by the telescope. The telescope scans back and forth across the sky (alternating between left to right and right to left) in the pattern indicated below by the arrows. The back-and-forth ordering of the values received from the scan is called *telescope order*.



The telescope records the data in telescope order into a 1-dimensional array of `double` values. This 1-dimensional array of information received from a single scan will be transferred into a 2-dimensional array, which reconstructs the original view of the rectangular area of the sky. This 2-dimensional array is part of the `SkyView` class, shown below. In this question you will write a constructor and a method for this class.

```
public class SkyView
{
  /** A rectangular array that holds the data representing a rectangular area of the sky. */
  private double[][] view;

  /** Constructs a SkyView object from a 1-dimensional array of scan data.
   *  @param numRows the number of rows represented in the view
   *         Precondition: numRows > 0
   *  @param numCols the number of columns represented in the view
   *         Precondition: numCols > 0
   *  @param scanned the scan data received from the telescope, stored in telescope order
   *         Precondition: scanned.length == numRows * numCols
   *  Postcondition: view has been created as a rectangular 2-dimensional array
   *                 with numRows rows and numCols columns and the values in
   *                 scanned have been copied to view and are ordered as
   *                 in the original rectangular area of sky.
   */
  public SkyView(int numRows, int numCols, double[] scanned)
  {  /* to be implemented in part (a) */  }

  /** Returns the average of the values in a rectangular section of view.
   *  @param startRow the first row index of the section
   *  @param endRow the last row index of the section
   *  @param startCol the first column index of the section
   *  @param endCol the last column index of the section
   *  Precondition: 0 <= startRow <= endRow < view.length
   *  Precondition: 0 <= startCol <= endCol < view[0].length
   *  @return the average of the values in the specified section of view
   */
  public double getAverage(int startRow, int endRow,
                           int startCol, int endCol)
  {  /* to be implemented in part (b) */  }

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

(a) Write the constructor for the `SkyView` class. The constructor initializes the `view` instance variable to a 2-dimensional array with `numRows` rows and `numCols` columns. The information from `scanned`, which is stored in the telescope order, is copied into `view` to reconstruct the sky view as originally seen by the telescope. The information in `scanned` must be rearranged as it is stored into `view` so that the sky view is oriented properly.

For example, suppose `scanned` contains values, as shown in the following array.

	0	1	2	3	4	5	6	7	8	9	10	11
scanned	0.3	0.7	0.8	0.4	1.4	1.1	0.2	0.5	0.1	1.6	0.6	0.9

Using the `scanned` array above, a `SkyView` object created with `new SkyView(4, 3, scanned)`, would have `view` initialized with the following values.

view	0	1	2
0	0.3	0.7	0.8
1	1.1	1.4	0.4
2	0.2	0.5	0.1
3	0.9	0.6	1.6

For another example, suppose `scanned` contains the following values.

	0	1	2	3	4	5
scanned	0.3	0.7	0.8	0.4	1.4	1.1

A `SkyView` object created with `new SkyView(3, 2, scanned)`, would have `view` initialized with the following values.

view	0	1
0	0.3	0.7
1	0.4	0.8
2	1.4	1.1

Complete the `SkyView` constructor below.

```
/** Constructs a SkyView object from a 1-dimensional array of scan data.
 *  @param numRows the number of rows represented in the view
 *         Precondition: numRows > 0
 *  @param numCols the number of columns represented in the view
 *         Precondition: numCols > 0
 *  @param scanned the scan data received from the telescope, stored in telescope order
 *         Precondition: scanned.length == numRows * numCols
 *  Postcondition: view has been created as a rectangular 2-dimensional array
 *                 with numRows rows and numCols columns and the values in
 *                 scanned have been copied to view and are ordered as
 *                 in the original rectangular area of sky.
 */
public SkyView(int numRows, int numCols, double[] scanned)
```

(b) Write the `SkyView` method `getAverage`, which returns the average of the elements of the section of `view` with row indexes from `startRow` through `endRow`, inclusive, and column indexes from `startCol` through `endCol`, inclusive.

For example, if `nightSky` is a `SkyView` object where `view` contains the values shown below, the call `nightSky.getAverage(1, 2, 0, 1)` should return `0.8`. (The average is `(1.1 + 1.4 + 0.2 + 0.5) / 4`, which equals `0.8`). The section being averaged is indicated by the dark outline in the table below.

`view`

	0	1	2
0	0.3	0.7	0.8
1	1.1	1.4	0.4
2	0.2	0.5	0.1
3	0.9	0.6	1.6

Class information repeated from the beginning of the question

```
public class SkyView

private double[][] view
public SkyView(int numRows, int numCols, double[] scanned)
public double getAverage(int startRow, int endRow,
                         int startCol, int endCol)
```

Complete method `getAverage` below.

```
/** Returns the average of the values in a rectangular section of view.
 *  @param startRow the first row index of the section
 *  @param endRow the last row index of the section
 *  @param startCol the first column index of the section
 *  @param endCol the last column index of the section
 *  Precondition: 0 <= startRow <= endRow < view.length
 *  Precondition: 0 <= startCol <= endCol < view[0].length
 *  @return the average of the values in the specified section of view
 */
public double getAverage(int startRow, int endRow,
                         int startCol, int endCol)
```

2012 Free Response Questions

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.
}
```

(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.

YES NO

2. This question involves reasoning about the GridWorld case study. Reference materials are provided in the appendices.

A retro bug behaves like a regular bug. It also has the ability to revert to its previous location and direction. When a retro bug acts, it maintains information about its location and direction at the beginning of the act. The retro bug has a `restore` method that restores it to the location (if possible) and direction it faced at the beginning of its previous act. A retro bug only maintains information about its most recent act; therefore, multiple calls to `restore` that occur before its next act will use the same information. The `restore` method has no effect if it is called before a retro bug's first act.

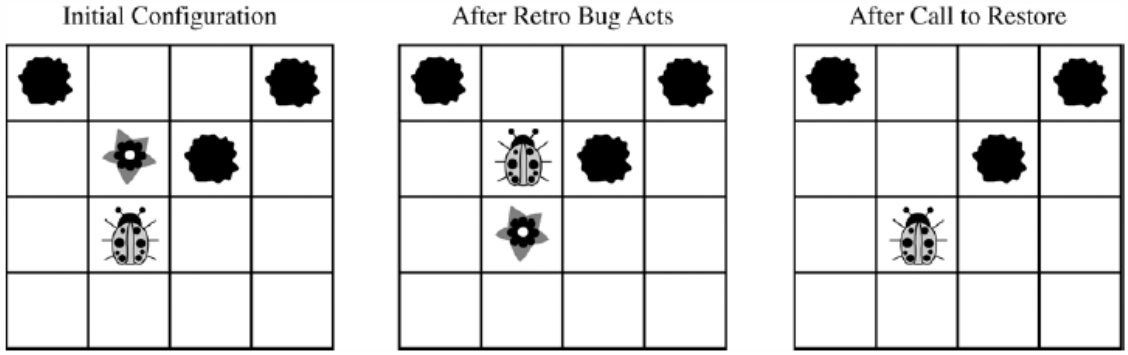
The `restore` method takes no parameters and does not return a value. The `restore` method has the following functionality.

- If the previous location of the retro bug is either unoccupied or contains a flower, the `restore` method places the retro bug in that previous location. The presence of any other type of actor in that location will prevent the retro bug from being placed in that location.
- The `restore` method always ends with the retro bug facing in the same direction that it had been facing at the beginning of its most recent act.

The following examples illustrate the behavior of the `restore` method.

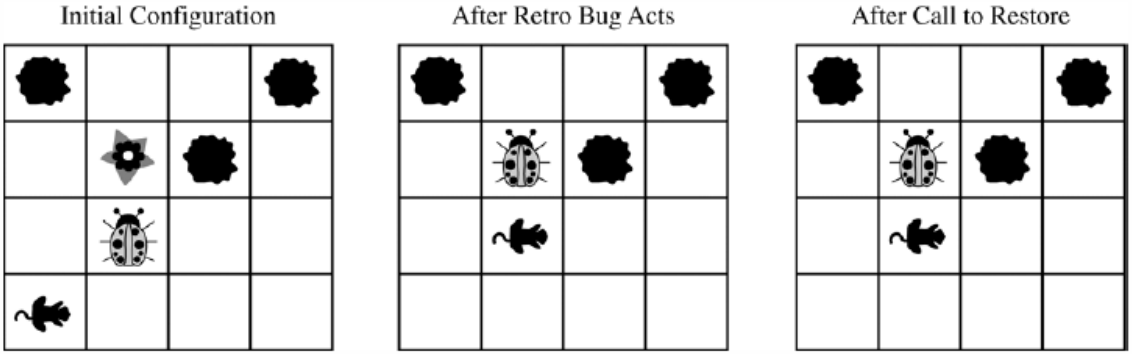
Example 1

The retro bug acts once and later calls `restore`. Note that the flower that was originally in front of the retro bug is not replaced as a result of the call to `restore`. The retro bug is returned to its previous direction, which, in this case, is the same as the current direction.



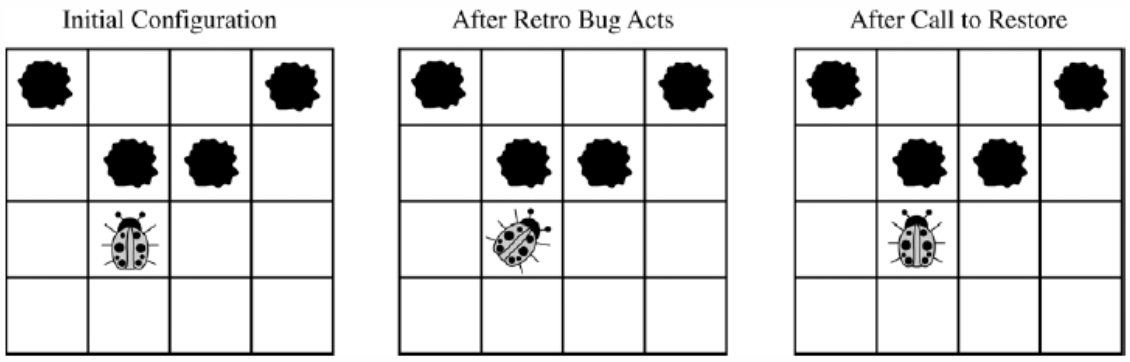
Example 2

The retro bug acts once and then some other actor moves into the location that the retro bug originally held. The call to `restore` results in the retro bug staying in its current location. The retro bug is returned to its previous direction (in this case it is the same as the current direction).



Example 3

The retro bug acts once and later calls `restore`. Because the retro bug is blocked from moving forward, it turns as its first act. The `restore` method results in the retro bug staying in its current location (the same as its previous location) and returning to its previous direction.



Write the entire `RetroBug` class, including all necessary instance variables and methods.

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.
}
```

(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.

0	1	2	3	4	5	6
"Trigger" 1340	null	"Silver" 1210	"Lady" 1575	null	"Patches" 1350	"Duke" 1410

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.

```
/** 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()
```

(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()
```

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()
```

2011 Free Response Questions

1. 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 partial declaration of the `Sound` class is shown below.

```
public class Sound
{
  /** the array of values in this sound; guaranteed not to be null */
  private int[] samples;

  /** Changes those values in this sound that have an amplitude greater than limit.
   *  Values greater than limit are changed to limit.
   *  Values less than -limit are changed to -limit.
   *  @param limit the amplitude limit
   *         Precondition: limit ≥ 0
   *  @return the number of values in this sound that this method changed
   */
  public int limitAmplitude(int limit)
  {  /* to be implemented in part (a) */  }

  /** Removes all silence from the beginning of this sound.
   *  Silence is represented by a value of 0.
   *  Precondition: samples contains at least one nonzero value
   *  Postcondition: the length of samples reflects the removal of starting silence
   */
  public void trimSilenceFromBeginning()
  {  /* to be implemented in part (b) */  }

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

(a) The volume of a sound depends on the amplitude of each value in the sound. The amplitude of a value is its absolute value. For example, the amplitude of -2300 is 2300 and the amplitude of 4000 is 4000.

Write the method `limitAmplitude` that will change any value that has an amplitude greater than the given limit. Values that are greater than `limit` are replaced with `limit`, and values that are less than `-limit` are replaced with `-limit`. The method returns the total number of values that were changed in the array. For example, assume that the array `samples` has been initialized with the following values.

40	2532	17	-2300	-17	-4000	2000	1048	-420	33	15	-32	2030	3223

When the statement

```
int numChanges = limitAmplitude(2000);
```

is executed, the value of `numChanges` will be 5, and the array `samples` will contain the following values.

40	2000	17	-2000	-17	-2000	2000	1048	-420	33	15	-32	2000	2000

Complete method `limitAmplitude` below.

```
/** Changes those values in this sound that have an amplitude greater than limit.
 *  Values greater than limit are changed to limit.
 *  Values less than -limit are changed to -limit.
 *  @param limit the amplitude limit
 *         Precondition: limit ≥ 0
 *  @return the number of values in this sound that this method changed
 */
public int limitAmplitude(int limit)
```

(b) Recorded sound often begins with silence. Silence in a sound is represented by a value of 0.

Write the method `trimSilenceFromBeginning` that removes the silence from the beginning of a sound. To remove starting silence, a new array of values is created that contains the same values as the original `samples` array in the same order but without the leading zeros. The instance variable `samples` is updated to refer to the new array. For example, suppose the instance variable `samples` refers to the following array.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	0	0	0	-14	0	-35	-39	0	-7	16	32	37	29	0	0

After `trimSilenceFromBeginning` has been called, the instance variable `samples` will refer to the following array.

Index	0	1	2	3	4	5	6	7	8	9	10	11
Value	-14	0	-35	-39	0	-7	16	32	37	29	0	0

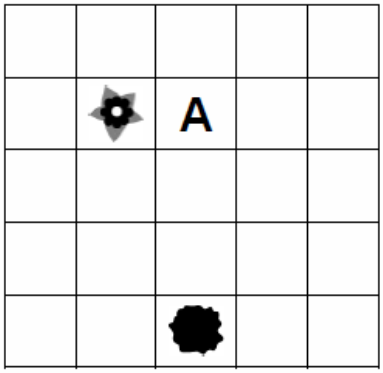
Complete method `trimSilenceFromBeginning` below.

```
/** Removes all silence from the beginning of this sound.
 *  Silence is represented by a value of 0.
 *  Precondition: samples contains at least one nonzero value
 *  Postcondition: the length of samples reflects the removal of starting silence
 */
public void trimSilenceFromBeginning()
```

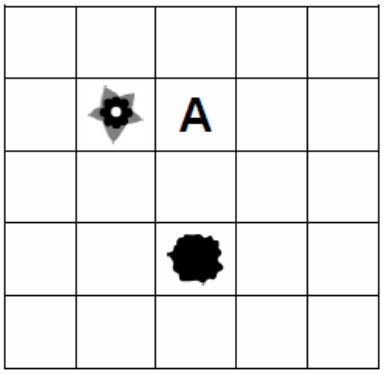
2. This question involves reasoning about the GridWorld case study. Reference materials are provided in the Appendix.

An attractive critter is a critter that processes other actors by attempting to relocate all of the other actors in the grid, including other attractive critters. The attractive critter attempts to move each other actor one grid cell closer to itself in the direction specified by `getDirectionToward`. An actor is relocated only if the location into which it would be relocated is empty. If an actor cannot be moved, it is left in its original position. After trying to move all other actors, the attractive critter moves like a critter.

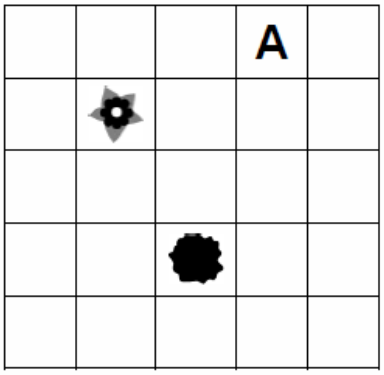
The following series of figures represents an example of how the attractive critter affects the other actors in the grid.



An attractive critter (indicated by the letter A) is in location (1, 2), a flower is in location (1, 1), and a rock is in location (4, 2).



When the attractive critter acts, the rock will be relocated to location (3, 2) because that location is one grid cell closer to the attractive critter and is empty. The flower will not be relocated because the grid cell that is one location closer to the attractive critter is already occupied.



After attempting to relocate all the other actors in the grid, the attractive critter then moves like a critter. In this example, the attractive critter moves to location (0, 3).

The order in which the actors in the grid are processed is not specified, making it possible to get different results from the same grid of actors.

Write the complete `AttractiveCritter` class, including all instance variables and required methods. Do NOT override the `act` method. Remember that your design must not violate the postconditions of the methods of the `Critter` class and that updating an object's instance variable changes the state of that object.

3. 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.

Tank index	0	1	2	3	4	5
Fuel level in tank	80	70	20	45	50	25
Robot			➔			

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.

Tank index	0	1	2	3	4	5	6
Fuel level in tank	20	30	80	55	50	75	20
Robot			➔				

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)
```

4. 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.

Original Message		Contents of Array					Encrypted Message
`"Surprise"`	→	`"S"`	`"u"`	`"r"`	`"p"`	→	`"Sruirspe"`
		`"r"`	`"i"`	`"s"`	`"e"`		

```
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.

```
/** 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)
```

2010 Free Response Questions

1. 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.

```
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)
```

2. An `APLine` is a line defined by the equation $ax + by + c = 0$, where a is not equal to zero, b is not equal to zero, and a, b, and c are all integers. The slope of an `APLine` is defined to be the `double` value $-a/b$. A point (represented by integers x and y) is on an `APLine` if the equation of the `APLine` is satisfied when those x and y values are substituted into the equation. That is, a point represented by x and y is on the line if $ax + by + c$ is equal to 0. Examples of two `APLine` equations are shown in the following table.

Equation	**Slope ($-a$ / b)**	**Is point (5, -2) on the line?**
$5x + 4y - 17 = 0$	-5 / 4 = -1.25	Yes, because $5(5) + 4(-2) + (-17) = 0$
$-25x + 40y + 30 = 0$	25 / 40 = 0.625	No, because $-25(5) + 40(-2) + 30 \neq 0$

Assume that the following code segment appears in a class other than `APLine`. The code segment shows an example of using the `APLine` class to represent the two equations shown in the table.

```
APLine line1 = new APLine(5, 4, -17);
double slope1 = line1.getSlope();          // slope1 is assigned -1.25
boolean onLine1 = line1.isOnLine(5, -2); // true because 5(5) + 4(-2) + (-17) = 0


APLine line2 = new APLine(-25, 40, 30);
double slope2 = line2.getSlope();          // slope2 is assigned 0.625
boolean onLine2 = line2.isOnLine(5, -2); // false because -25(5) + 40(-2) + 30 ≠ 0
```

Write the `APLine` class. Your implementation must include a constructor that has three integer parameters that represent a, b, and c, in that order. You may assume that the values of the parameters representing a and b are not zero. It must also include a method `getSlope` that calculates and returns the slope of the line, and a method `isOnLine` that returns `true` if the point represented by its two parameters (`x` and `y`, in that order) is on the `APLine` and returns `false` otherwise. Your class must produce the indicated results when invoked by the code segment given above. You may ignore any issues related to integer overflow.

3. A hiking trail has elevation markers posted at regular intervals along the trail. Elevation information about a trail can be stored in an array, where each element in the array represents the elevation at a marker. The elevation at the first marker will be stored at array index 0, the elevation at the second marker will be stored at array index 1, and so forth. Elevations between markers are ignored in this question. The graph below shows an example of trail elevations.

The table below contains the data represented in the graph.

Trail Elevation (meters)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Elevation	100	150	105	120	90	80	50	75	75	70	80	90	100

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

```
public class Trail
{
  /** Representation of the trail. The number of markers on the trail is markers.length. */
  private int[] markers;

  /** Determines if a trail segment is level. A trail segment is defined by a starting marker,
   *  an ending marker, and all markers between those two markers.
   *  A trail segment is level if it has a difference between the maximum elevation
   *  and minimum elevation that is less than or equal to 10 meters.
   *  @param start the index of the starting marker
   *  @param end the index of the ending marker
   *         Precondition: 0 <= start < end <= markers.length - 1
   *  @return true if the difference between the maximum and minimum
   *          elevation on this segment of the trail is less than or equal to 10 meters;
   *          false otherwise.
   */
  public boolean isLevelTrailSegment(int start, int end)
  {  /* to be implemented in part (a) */  }

  /** Determines if this trail is rated difficult. A trail is rated by counting the number of changes in
   *  elevation that are at least 30 meters (up or down) between two consecutive markers. A trail
   *  with 3 or more such changes is rated difficult.
   *  @return true if the trail is rated difficult; false otherwise.
   */
  public boolean isDifficult()
  {  /* to be implemented in part (b) */  }

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

(a) Write the `Trail` method `isLevelTrailSegment`. A trail segment is defined by a starting marker, an ending marker, and all markers between those two markers. The parameters of the method are the index of the starting marker and the index of the ending marker. The method will return `true` if the difference between the maximum elevation and the minimum elevation in the trail segment is less than or equal to 10 meters.

For the trail shown at the beginning of the question, the trail segment starting at marker 7 and ending at marker 10 has elevations ranging between 70 and 80 meters. Because the difference between 80 and 70 is equal to 10, the trail segment is considered level.

The trail segment starting at marker 2 and ending at marker 12 has elevations ranging between 50 and 120 meters. Because the difference between 120 and 50 is greater than 10, this trail segment is not considered level.

Complete method `isLevelTrailSegment` below.

```
/** Determines if a trail segment is level. A trail segment is defined by a starting marker,
 *  an ending marker, and all markers between those two markers.
 *  A trail segment is level if it has a difference between the maximum elevation
 *  and minimum elevation that is less than or equal to 10 meters.
 *  @param start the index of the starting marker
 *  @param end the index of the ending marker
 *         Precondition: 0 <= start < end <= markers.length - 1
 *  @return true if the difference between the maximum and minimum
 *          elevation on this segment of the trail is less than or equal to 10 meters;
 *          false otherwise.
 */
public boolean isLevelTrailSegment(int start, int end)
```

(b) Write the `Trail` method `isDifficult`. A trail is rated by counting the number of changes in elevation that are at least 30 meters (up or down) between two consecutive markers. A trail with 3 or more such changes is rated difficult. The following table shows trail elevation data and the elevation changes between consecutive trail markers.

Trail Elevation (meters)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Elevation	100	150	105	120	90	80	50	75	75	70	80	90	100
	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	\ /	
Elevation change	50	-45	15	-30	-10	-30	25	0	-5	10	10	10	

This trail is rated difficult because it has 4 changes in elevation that are 30 meters or more (between markers 0 and 1, between markers 1 and 2, between markers 3 and 4, and between markers 5 and 6).

Complete method `isDifficult` below.

```
/** Determines if this trail is difficult. A trail is rated by counting the number of changes in
 *  elevation that are at least 30 meters (up or down) between two consecutive markers. A trail
 *  with 3 or more such changes is rated difficult.
 *  @return true if the trail is rated difficult; false otherwise.
 */
public boolean isDifficult()
```

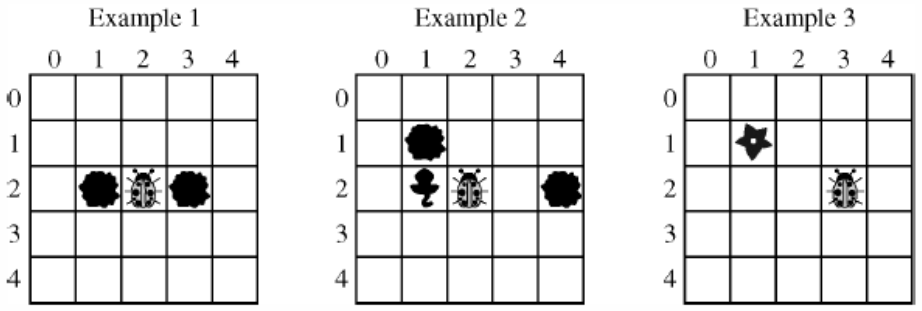
4. This question involves reasoning about the GridWorld case study. Reference materials are provided in the Appendix.

In this question, you will write two unrelated methods of the `GridChecker` class that will process a `BoundedGrid<Actor>` object. Recall that the `BoundedGrid` class implements the `Grid` interface. Also note that the methods in the `Grid` interface that return an array list will return an empty array list when no objects meet the return criteria.

The declaration of the `GridChecker` class is shown below.

```
public class GridChecker
{
  /**  The grid to check; guaranteed never to be null  */
  private BoundedGrid<Actor> gr;

  /** @return an Actor in the grid gr with the most neighbors; null if no actors in the grid.
   */
  public Actor actorWithMostNeighbors()
  {  /* to be implemented in part (a) */  }

  /** Returns a list of all occupied locations in the grid gr that are within 2 rows
   *  and 2 columns of loc. The object references in the returned list may appear in any order.
   *  @param loc a valid location in the grid gr
   *  @return a list of all occupied locations in the grid gr that are within 2 rows
   *          and 2 columns of loc.
   */
  public List<Location> getOccupiedWithinTwo(Location loc)
  {  /* to be implemented in part (b) */  }

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

(a) The method `actorWithMostNeighbors` returns an `Actor` in the grid `gr` that has the most neighbors. A neighbor of a given actor is another actor that occupies any of the given actor's 8 adjacent locations. Consider the following examples.

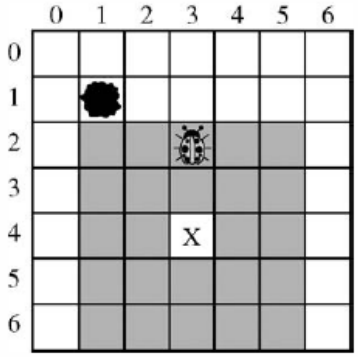


In Example 1, the method `actorWithMostNeighbors` will return the `Actor` (in this case a bug) in location (2, 2). In Example 2, there are three `Actor` objects that have the same largest number of neighbors—the rock in location (1, 1), the critter in location (2, 1), and the bug in location (2, 2). In this case, any one of those three `Actor` objects may be returned. Similarly, either of the `Actor` objects shown in Example 3 could be returned. If there are no `Actor` objects in the grid, the method returns `null`.

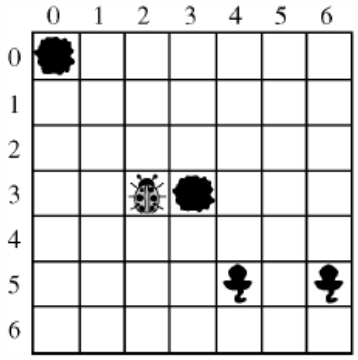
Complete method `actorWithMostNeighbors` below.

```
/** @return an Actor in the grid gr with the most neighbors; null if no actors in the grid.
 */
public Actor actorWithMostNeighbors()
```

(b) The method `getOccupiedWithinTwo` returns a list containing all occupied locations in the grid `gr` that are within 2 rows and 2 columns of the parameter `loc`. The location `loc` is not included in the returned list, even if that location is occupied. The object references in the returned list may appear in any order. The shaded area in the following diagram shows the group of locations that are within 2 rows and 2 columns of the location labeled X.



For example, consider the following grid.



The table below shows the results of several calls to the method `getOccupiedWithinTwo`.

`loc`	`getOccupiedWithinTwo(loc)`
`(1, 1)`	`[(0, 0), (3, 2), (3, 3)]`
`(0, 0)`	An empty list is returned.
`(3, 3)`	`[(3, 2), (5, 4)]`
`(5, 4)`	`[(3, 2), (3, 3), (5, 6)]`
`(5, 6)`	`[(5, 4)]`

Complete method `getOccupiedWithinTwo` below.

```
/** Returns a list of all occupied locations in the grid gr that are within 2 rows
 *  and 2 columns of loc. The object references in the returned list may appear in any order.
 *  @param loc a valid location in the grid gr
 *  @return a list of all occupied locations in the grid gr that are within 2 rows
 *          and 2 columns of loc.
 */
public List<Location> getOccupiedWithinTwo(Location loc)
```

Appendix

AP® COMPUTER SCIENCE A
2012 GENERAL SCORING GUIDELINES

Apply the question-specific rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question-specific rubric. No part of a question — (a), (b), or (c) — may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in different parts of that question.

1-Point Penalty

(w) Extraneous code that causes a side effect or prevents earning points in the rubric (*e.g., information written to output*)

(x) Local variables used but none declared

(y) Destruction of persistent data *(e.g., changing value referenced by parameter)*

(z) `Void` method or constructor that returns a value

No Penalty

- Extraneous code that causes no side effect
- Extraneous code that is unreachable and would not have earned points in rubric
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared, provided that other variables are declared in some part
- `private` qualifier on local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (x • ÷ ≤ ≥ < > ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` (and vice versa)
- Array/collection element access confusion (`[]` vs. `get` for r-values)
- Array/collection element modification confusion (`[]` vs. `set` for l-values)
- `length/size` confusion for array, `String`, and `ArrayList`, with or without `()`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, (*e.g.,* `int[size] nums = new int[size];`)
- Missing `;` provided that line breaks and indentation clearly convey intent
- Missing `{ }` where indentation clearly conveys intent and `{ }` are used elsewhere
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if/while` conditions
- Use of local variable outside declared scope (must be within same method body)
- Failure to cast object retrieved from nongeneric collection

** Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be* **unambiguously** *inferred from context; for example, "*`ArayList`*" instead of "*`ArrayList`*". As a counterexample, note that if the code declares "*`Bug bug;`*" and then uses "*`Bug.move()`*" instead of "*`bug.move()`*", the context does* **not** *allow for the reader to assume the object instead of the class.*