APCS Final Exam – Fall 2013

Time: 1 hour, 30 minutes.

This exam consists of two parts: a multiple choice section and a free response section. In addition to provided scrap paper, you may write on the test paper. However, only answer sheets will be graded.

Part 1: Multiple Choice – Bubble in the *best* answer for each in **pencil** on answer sheet.

1. What is the value of n after the following code is executed?

```
int n=2013;
for (int i=0;i<42;i++)
    n = (n+5)/3;
```

(A) 0 **(B)** 1 **(C)** 2 **(D)** 3 **(E)** 65

2. What is the value of a[1] after the following code is executed?

```
int[] a = {0,2,4,1,3};
for (int i=0;i<a.length;i++)
    a[i] = a[(a[i]+3)%a.length];
```

(A) 0 **(B)** 1 **(C)** 2 **(D)** 3 **(E)** 4

3. What is the output of the following code:

```
List nums = new ArrayList<Integer>(3);
nums.add(new Integer(1));
nums.add(new Integer(2));
nums.add(0,nums.get(1));

Object x = nums.get(0);
Object y = nums.get(1);
if (x == y)
     System.out.println(x + " is equal to " + y);
else
     System.out.println(x + " is NOT equal to " + y);
```

(A) 1 is equal to 2 **(B)** 1 is NOT equal to 2 **(C)** 2 is equal to 2

(D) 2 is NOT equal to 1 **(E)** IndexOutOfBoundsException

4. What is the output of this code?

```
List cities = new ArrayList();
cities.add("Atlanta");
cities.add("Boston");
for (int i=1; i < cities.size() ; i++)
    cities.add(i,"+");
System.out.println(cities);
```

(A) `[Atlanta, Boston]` **(B)** `[Atlanta, +, Boston]` **(C)** `[Atlanta, Boston, +]`

(D) `[Atlanta, +, Boston, +]` **(E)** No output because the program goes into an infinite loop

The next two questions refer to the code below:

```
public interface InterfaceA { void methodA(); }
public interface InterfaceB extends InterfaceA { void methodB(); }
public class ClassA implements InterfaceA {
     public void methodA() { }
     public void methodB() { }
}
public class ClassB extends ClassA implements InterfaceB {
     public ClassB() { }
     // other methods not shown
}
```

5. Which methods, at a minimum, must be defined in **ClassB** for it to compile with no errors?

(A) methodA **(B)** methodB **(C)** methodA and methodB

(D) methodA, methodB, and toString **(E)** No particular methods are required

6. Which of the following statements causes a syntax error?

(A) `InterfaceA obj = new ClassA();`
(B) `InterfaceB obj = new ClassA();`
(C) `InterfaceA obj = new ClassB();`
(D) `InterfaceB obj = new ClassB();`
(E) `ClassA obj = new ClassB();`

7. A car dealership needs a program to store information about the cars for sale. For each car, they want to keep track of the following information: number of doors (2 or 4), whether the car has air conditioning, and its average number of miles per gallon. Which of the following is the best design?

(A) Use one class, Car, which has three data fields:
int numDoors, boolean hasAir, and double milesPerGallon.
(B) Use four unrelated classes:
Car, Doors, AirConditioning, and MilesPerGallon.
(C) Use a class Car, which has three subclasses:
Doors, AirConditioning, and MilesPerGallon.
(D) Use a class Car, which has a subclass Doors, with a subclass AirConditioning, with a subclass MilesPerGallon.
(E) Use three classes: Doors, AirConditioning, and MilesPerGallon, each with a subclass Car.

8. Consider the following declarations.

```
public interface Comparable {
     int compareTo(Object other);
}
public class SomeClass implements Comparable {
      // ...other methods not shown
}
```

Which of the following method signatures of compareTo will satisfy the Comparable interface requirement?

I `public int compareTo(Object other)`
II `public int compareTo(SomeClass other)`
III `public boolean compareTo(Object other)`

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

The next two questions refer to the following information:

Consider the following instance variable and method findLongest with line numbers added for reference. Method findLongest is intended to find the longest consecutive block of the value target occurring in the array nums; however, findLongest does not work as intended.

For example, if the array nums contains the values
[7 , 10 , 10 , 15 , 15 , 15 , 15 , 10 , 10 , 10 , 15 , 10 , 10],

the call findLongest(10) should return 3, the length of the longest consecutive block of 10's.

```
 private int[] nums;
 public int findLongest(int target) {
      int lenCount = 0;
      int maxLen = 0;
Line 1:    for ( int val : nums )
Line 2:    {
Line 3:        if (val == target)
Line 4:        {
Line 5:               lenCount++;
Line 6:        }
Line 7:        else
Line 8:        {
Line 9:            if (lenCount > maxLen)
Line 10:           {
Line 11:               maxLen = lenCount;
Line 12:           }
Line 13:       }
Line 14:   }
Line 15:   if (lenCount > maxLen)
Line 16:   {
Line 17:      maxLen = lenCount;
Line 18:   }
Line 19:   return maxLen;
}
```

9. The method **findLongest** does not work as intended. Which of the following best describes the value returned by a call to **findLongest**?

(A) It is the length of the shortest consecutive block of the value **target** in **nums**.
(B) It is the length of the array **nums**.
(C) It is the number of occurrences of the value **target** in **nums**.
(D) It is the length of the first consecutive block of the value **target** in **nums**.
(E) It is the length of the last consecutive block of the value **target** in **nums**.

10. Which of these changes should be made so that method **findLongest** will work as intended?

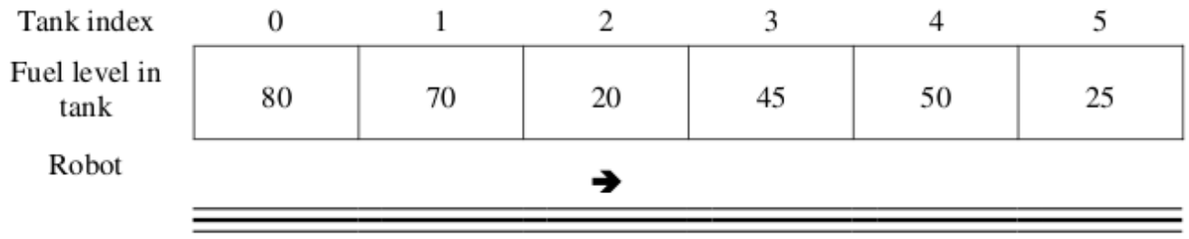
(A) Insert the statement `lenCount = 0;` between lines 2 and 3.
(B) Insert the statement `lenCount = 0;` between lines 8 and 9.
(C) Insert the statement `lenCount = 0;` between lines 10 and 11.
(D) Insert the statement `lenCount = 0;` between lines 11 and 12.
(E) Insert the statement `lenCount = 0;` between lines 17 and 18.

Part 2: Free Response – Write answers on **answer sheet** (pen or pencil).

11.
A fuel depot has a number of fuel tanks arranged in a line and a robot that moves a filling mechanism back and forth along the line so that the tanks can be filled. A fuel tank is specified by the FuelTank interface below.

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

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



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

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


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


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


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

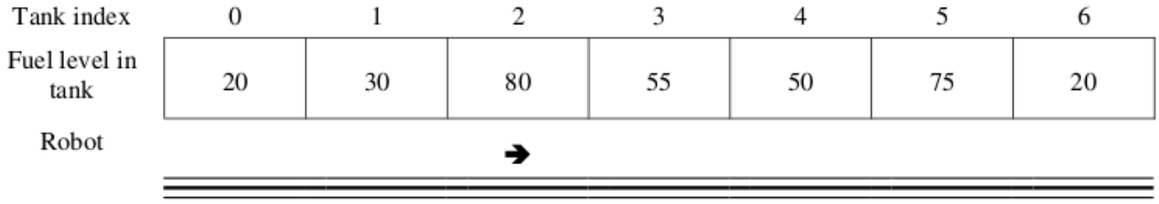
11. (*continued*)
A fuel depot is represented by the FuelDepot class as shown in the following class declaration.

```
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.
}
```

11a.
Write the FuelDepot method nextTankToFill that returns the index of the next tank to be filled.
The index for the next tank to be filled is determined according to the following rules:

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

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



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

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

Complete method nextTankToFill using this specification:

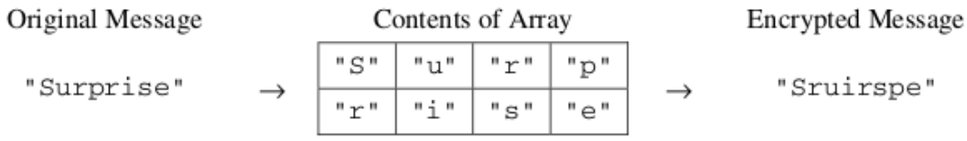
```
/** Determines and returns the index of the next tank to be filled.
 * @param threshold tanks with a fuel level <= threshold may be filled
 * @return index of the location of the next tank to be filled
 * Postcondition: the state of the robot has not changed
 */
public int nextTankToFill(int threshold)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

11b. 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 using this specification:

```
/* Moves the robot to location locIndex.
 * @param locIndex 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)
```

12.
In this question you will write two methods for a class RouteCipher that encrypts (puts into a coded form) a message by changing the order of the characters in the message. The route cipher fills a two-dimensional array with single-character substrings of the original message in row-major order, encrypting the message by retrieving the single-character substrings in column-major order.

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



An incomplete implementation of the RouteCipher class is shown below.

```
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.
}
```

12a. 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.

12. (*continued*)
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 using this specification:

```
/** 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)
```

12b. 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 the following specification:

```
/** 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)
```