Test 1 Review: CompSci 6

Name (print): ________________________________

Honor Acknowledgment (signature): ____

DO NOT SPEND MORE THAN 15 MINUTES ON ANY OF THE QUESTIONS! If you do not see the solution to a problem right away, move on to another problem and come back to it later. The final page is a list of common methods of classes we have studied in class so that you do not need to memorize such details.

Before starting, make sure your test contains 16 pages.

If you think there is a syntax error, then ask.

	value	grade
Problem 1	6 pts.	
Problem 2	15 pts.	
Problem 3	15 pts.	
Problem 4	14 pts.	
TOTAL:	50 pts.	

PROBLEM 1 : (*Not, not, Who's There?* (6 points))

Part A: (2 points)

Each statement below is intended to print `number ok` if `number` is any value less than zero or greater than 100. No message should be printed for values in the range $0 \ldots 100$, inclusive.

Circle each implementation that correctly meets this specification.

1.
```
if (number < 0 || number > 100) {
    System.out.println("number ok");
}
```

2.
```
if (! (0 <= number && number <= 100) ) {
    System.out.println("number ok");
}
```

3.
```
if (0 > number && 100 > number) {
    System.out.println("number ok");
}
```

Part B: (4 points)

Three implementations of a method `longMonth` are shown below. The method is supposed to return true if the parameter `month` represents a month with 31 days, and false otherwise. The table below shows how many days each month has and the number used to represent each month in the code below. For example, January is represented by 0, February by 1, and so on with December being 11.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	1	2	3	4	5	6	7	8	9	10	11
31	28	31	30	31	30	31	31	30	31	30	31

Assume that all methods compile and run, you are evaluating them for correctness in terms of logic.

Circle each implementation that correctly meets this specification.

1.
```
public boolean longMonth (int month) {
    if (month == 1 || month == 3 || month == 5 ||
        month == 8 || month == 10) {
        return false;
    }
    else {
        return true;
    }
}
```

2.
```
public boolean longMonth (int month) {
    int[] lm = { 0, 2, 4, 6, 7, 9, 11 };
    for (int k = 0; k < lm.length; k++) {
        if (lm[k] == month) return true;
    }
    return false;
}
```

3.
```
public boolean longMonth (int month) {
    String shortMonths = "1:3:5:8:10:";
    return shortMonths.indexOf(month+":") < 0;
}
```

4.
```
public boolean longMonth (int month) {
    return month == 0 || month == 2 || month == 4 ||
           month == 6 || month == 7 || month == 9 ||
           month == 11;
}
```

PROBLEM 2 : (*Glory Days:* (15 pts))

Implement a class that represents a music jukebox. To support its use in as many diners as possible, your class should read its possible songs from a data file, one answer per line. Each time the jukebox is requested to play a song, a random one from those read from the file should be returned.

For example, below is the first five lines of a file that holds some of my favorite songs in no particular order:

```
Kool and The Gang - Ladies Night
Tom Jones - Its Not Unusual
Bee Gees - Night Fever
Captain & Tennille - Love Will Keep Us Together
Donna Summer - Dim All The Lights
```

For example, the code below represents how your class might be used in a program.

```
JukeBox box = new JukeBox(new Scanner(new File("old_songs.txt")));
while (box.numSongsPlayed() < 5)
{
    System.out.println(box.playSong());
}
```

Below is one possible sample output from the program above given the data file above.

```
Captain & Tennille - Love Will Keep Us Together
Tom Jones - Its Not Unusual
Donna Summer - Dim All The Lights
Kool and The Gang - Ladies Night
Bee Gees - Night Fever
```

(problem continued on next page ⇒)

Part A: (3 points)

Complete the declaration of the class' instance variables below. You should add any private state (instance variables) you think your JukeBox class needs in order to implement the constructor and other two methods of the class so that they function as shown in the example above.

```
public class JukeBox
{
    // Part A: declare instance variables here









    public JukeBox (Scanner input)
    {
        // implemented in Part B
    }

    public int numSongsPlayed ()
    {
        // implemented in Part B
    }

    public String playSong ()
    {
        // implemented in Part B
    }
}
```

Part B: (12 points)

Write the implementation for the class member functions previously declared below. Note, you may lose points on Part A for instance variables that are declared there, but not used in this Part.

```
/**
 * All songs from the list passed in are added to another
 * list kept by this class.  All other instance variables
 * are initialized.
 */
public JukeBox (Scanner input)
{
    input.useDelimiters("\n");








}

/**
 * @returns number of songs that have been played (i.e.,
 *          the number of times playSong has been called)
 */
public int numSongsPlayed ()
{



}

/**
 * @returns one random string from possible songs
 */
public String playSong ()
{







}
```

PROBLEM 3 : (*Form letters:* (15 points))

Different people want different prizes when they enter a sweepstakes. To accomodate this, you are to write several classes that inherit from a generic *Winner* class, but differ only in the prize that is offered to the contestant.

Consider the hierarchy of classes given below to solve this problem (their method implementations are not important yet, so they are not included below):

```
public abstract class Winner
{
  private string myFirstName;
  private string myLastName;
  private int myTicketNumber;

  public Winner (String firstName, String lastName)
  public void printLetter ()

  protected void printGreeting ()
  protected void printTicket ()
  protected void printPrize ()

  private void generateTicket (int numDigits)
}

public class CarWinner extends Winner
{
  private String myCar;

  public CarWinner (String firstName, String lastName, String car)

  protected void printGreeting ()
  protected void printPrize ()
}

class CruiseWinner extends Winner
{
  private String myDestination;

  public CruiseWinner (String firstName, String lastName, String destination)

  protected void printGreeting ()
  protected void printPrize ()
}
```

Part A: (2 points)

If an instance of *CarWinner* was created for the contestant "Amanda Harris" and an instance of *CruiseWinner* was created for "Michael Abernathy", the following output should be printed.

```
Amanda Harris, you've won a NEW CAR!
Amanda, that's what you will hear if your
ticket 387452289 matches the winning ticket.
Imagine, Amanda, you could be cruising
in a Mercedes down Main Street!

Michael Abernathy, you've won a 7-DAY CRUISE!
Michael, that's what you will hear if your
ticket 126755804 matches the winning ticket.
Imagine, Michael, you could be cruising
to beautiful Hawaii with your sweetheart!
```

Assumming that all the classes declared previously have been correctly implemented, complete the code started in *main* below such that the output above is printed.

```
public static void main (String[] args)
{
    Winner c1 = new CarWinner(                    ,                    ,                 );
    Winner c2 = new CruiseWinner(                    ,                    ,              );

    c1.printLetter();
    c2.printLetter();
}
```

Below the code for the base class *Winner* is given (except for the function *generateTicket* which you will complete in the final part of this problem). On the next page, you will complete the code for the two sub-classes of *Winner* so that they generate the output in the previous part.

In writing your subclasses, you should write as little new code as possible. In other, make use of calls to already implemented base class functions where possible.

```
public Winner (String firstName, String lastName)
{
    myFirstName = firstName;
    myLastName = lastName;
    myTicketNumber = generateTicket(9);
}

public void printLetter ()
{
    printGreeting();
    printTicket();
    printPrize();
    System.out.println();
}

protected void printGreeting ()
{
    System.out.print(myFirstName + " " + myLastName + ", ");
}

protected void printTicket ()
{
    System.out.println(myFirstName + ", that's what you will hear if your");
    System.out.println("ticket " + myTicketNumber + " matches the winning ticket.");
}

protected void printPrize ()
{
    System.out.println("Imagine, " + myFirstName + ", you could be cruising");
}
```

(problem continued on next page ⇒)

Part B: (5 points)

```
public CarWinner (String firstName, String lastName, String car)
{







}

protected void printGreeting ()
{







}

protected void printPrize ()
{







}
```

Part C: (5 points)

```
public CruiseWinner (String firstName, String lastName, String destination);
{







}

protected void printGreeting ()
{






}

protected void printPrize ()
{






}
```

Part D: (3 points)

Write the function *generateTicket* that takes a number of digits to generate and creates a random number guaranteed to have that many digits. Note, this means the number can have zeros within it, like the five-digit number 10020, but it cannot start with a zero. For example, the number 01234 is not a valid five-digit number.

```
private int generateTicket (int numDigits)
{
    int result = 0;
    Random rand = new Random();











    return result;
}
```

PROBLEM 4 : (*Right on Target:* (14 pts))

Part A: (8 points)

Write the `paint` method of a sub-class of `Mover` such that draws itself as a number of concentric ovals centered at the same position. They should be drawn from largest to smallest so that they are all visible and look like an archery target. The number of ovals drawn is passed as a parameter to the constructor and stored in an instance variable `myNumOvals` and the ovals should be drawn such that the distance between each is equal and each oval is brighter than the one drawn before it.

For example, a target of three ovals should be drawn such that largest oval's size is the same as that given for the entire target, the inner circle's size is two-thirds of that given for the target, and the center circle's size is one-third of that given for the target. Your `paint` method should work for any target with a positive number of ovals.

```
public class Target extends Mover
{
    private int myNumOvals;

    public Target (Point center, Dimension size, Point velocity, Color color,
                   int numRings)
    {
        super(center, size, velocity, color);
        myNumOvals = numRings;
    }

    public void paint (Graphics pen)
    {










    }

    public void move (Dimension bounds)
    {
        // implementation not shown
    }
 }
```

Part B: (6 points)

Write the `move` method of a sub-class of `Mover` such that it expands its size based on its velocity instead of changing its position until its size is the same or larger than that of the `bounds`.

For example, if its size was $100x50$ pixels and its velocity was $(2, 2)$, then its dimensions would double each time update was called. After the first call, its size would be $200x100$; after the second call, its size would be $400x200$; and on and on, until one of the dimensions grew to be larger than the corresponding `bounds` dimension.

Complete the class *Grower* started below.

```
public class Grower extends Mover
{
    // no instance variables needed

    public Grower (Point center, Dimension size, Point velocity, Color color)
    {
        super(center, size, velocity, color);
    }

    public void paint (Graphics pen)
    {
        // implementation not shown
    }

    public void move (Dimension bounds)
    {









    }
 }
```

Throughout this test, assume that the following classes and methods are available. These classes are taken directly from the material used in class. There should be no methods you have never seen before here.

Point

```
public class Point {
    // coordinates
    public int x;
    public int y;

    // Constructs and initializes a point at the
    // specified (x,y) location.
    public Point (int x, int y)

    // Returns distance from this point to given one
    public int distance (Point other)

    // Adds dx and dy to this point's coordinates
    public void translate (int dx, int dy)
}
```

Dimension

```
public class Dimension {
    // lengths
    public int width;
    public int height;

    // Constructs and initializes a dimension
    // with the specified (w, h) lengths.
    public Dimension (int w, int h)
}
```

Color

```
public class Dimension {
    // Constructs and initializes a color with
    // the specified (r, g, b) components.
    public Color (int red, int green, int blue)

    // Returns the red component
    public int getRed ()

    // Returns the green component
    public int getGreen ()

    // Returns the blue component
    public int getBlue ()

    // Returns a color that is brighter than
    // this one
    public Color brighter ()

    // Returns a color that is darker than
    // this one
    public Color darker ()
}
```

Random

```
public class Random {
    // Creates a new random number generator.
    public Random ()

    // Returns pseudorandom, uniformly distributed,
    // int value between 0 (inclusive) and the
    // specified value (exclusive)
    public int nextInt (int n)
}
```

ArrayList

```
public class ArrayList {
    // Constructs an empty list
    public ArrayList ()

    // Returns the number of elements in this list.
    public int size ()

    // Searches for the first occurence of the given
    // argument, returns  -1 if not found
    public int indexOf (Object itemm)

    // Returns element at index in this list.
    public Object get (int index)

    // Appends specified element to end of this list.
    public boolean add (Object o)
}
```

Scanner

```
public class Scanner
{
    // Create Scanner for that reads data from a file.
    public Scanner (File file)

    // Create Scanner for that reads data from a string.
    public Scanner (String str)

    // Change delimiters used to separate items
    public void useDelimiter (String characters)

    // Check if more items are available
    public boolean hasNext ()

    // Get next delimited item as a string
    public String next ()

    // Get next delimited item as an integer value
    public int nextInt ()
}
```

Integer

```
public class Integer {
    // The smallest value of type int
    public static final int MIN_VALUE

    // The largest value of type int
    public static final int MAX_VALUE

    // Returns the integer represented by the
    // argument as a decimal integer.
    public static int parseInt (String s)

    // Returns a new String object representing
    // the specified integer.
    public static String toString (int i)

    // Returns value of Integer object as an int
    public int intValue ()
}
```

String

```
public class String {
    // Returns length of this string.
    public int length ()

    // Returns index within this string of first
    // occurrence of the specified substring.
    // If str is not found, then returns -1
    public int indexOf (String str)

    // Returns index within this string of last
    // occurrence of the specified substring.
    // If str is not found, then returns -1
    public int lastIndexOf (String str)
}
```