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 10 pages.

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

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL:</strong></td>
<td><strong>60 pts.</strong></td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1:  (Short Answer: (6 points))

Describe the differences between an instance variable, a parameter, and a local variable. For each, explain where it is declared and why you might want to use it.

1. instance variable

2. parameter

3. local variable
PROBLEM 2:  \textit{(Leap of faith: (6 pts))}

A year is a leap year when it has 366 days instead of 365 days. In the international Gregorian calendar a year is a leap year according to the following, correct but perhaps poorly worded, rules as obtained from \url{www.timeanddate.com/date/leapyear.html}

1. Every year divisible by 4 is a leap year
2. But every year divisible by 100 is NOT a leap year
3. Unless the year is also divisible by 400, then it is still a leap year

This means 1800, 1900, and 2100 are not leap years but 2000 and 2004 are leap years.

Which of the following implementations of a method \texttt{isLeap} returns true if \texttt{year} is a leap year and false otherwise. Circle those that are correct. If a method is not correct, provide a value for \texttt{year} for which it returns the wrong value.

\begin{verbatim}
public boolean isLeap (int year)
{
    if (year % 400 == 0) return true;
    if (year % 100 == 0) return false;
    if (year % 4 == 0) return true;
    return false;
}
\end{verbatim}

\begin{verbatim}
public boolean isLeap (int year)
{
    return year % 400 == 0 || (year % 4 == 0 && year % 100 != 0);
}
\end{verbatim}

\begin{verbatim}
public boolean isLeap (int year)
{
    if (year % 100 == 0) return false;
    if (year % 400 == 0 || year % 4 == 0) return true;
    return false;
}
\end{verbatim}
PROBLEM 3:  

(Run it Off: (8 points))

Part A (6 points)

Arnie likes to jog in the morning. As he jogs, he counts the number of strides he makes in the first minute of his jogging and in the last minute. Arnie then averages these two values, calling it the average number of strides he makes in a minute while he was jogging. Arnie uses this average, the length of his stride, and the total time he spent jogging to determine the distance he has jogged in miles (by converting hours and minutes to total minutes, then using the formula $distance = rate \times time$, and then converting feet to miles).

Complete the method `computeDistance` that calculates how far Arnie jogged in miles based on four numbers (number of strides in first minute, number of strides in last minute, number of hours spent jogging, and number of minutes spent jogging). Assume that Arnie’s stride is 2.5 feet and that there are 5280 feet in a mile.

Complete the method `computeDistance` below that returns the number of miles Arnie jogged.

```java
public double computeDistance (int firstStrideCount, int lastStrideCount,
                               int hours, int minutes)
{ 
```
Consider the following class declaration used for this problem.

```java
/**
 * A Line is represented by the two points which define its ends.
 */
public class Line
{
    private Point myStart;
    private Point myEnd;

    public Line (Point start, Point end)
    {
        // completed in Part A
    }

    public double length ()
    {
        // completed in Part B
    }

    public void scale (double factor)
    {
        // completed in Part C
    }
}
```

Problem continued on next page ...
Part A: (4 points)
Complete the constructor for the Line class that takes two points representing its end points.

```java
public Line (Point start, Point end)
{
}
```

Part B: (6 points)
Complete the method length which returns the length of the line as the distance between those end points. Recall that the distance between two points is the square root of the sum of the squares of the distance between each point’s coordinates.

```java
public double length ()
{
}
```
Part C: (10 points)

Complete the method `scale` which, given an amount by which to change length of the line, changes the line's starting and ending points so that the line still has the same center point, but the new length. Thus the line appears not to have moved (its center is the same), but its end points have each been moved so that the length of the line has changed.

For example, if the line's oringally has the endpoints (10, 10) and (20, 10) and `scale` is called with a factor of two, to double the length of the line, then the new computed endpoints would be (5, 10) and (25, 10). Before the call, the center point of the line is (15, 10) and the length is 10. After the call, the center point of the line is (15, 10) but the length is now 20.

```java
public void scale (double factor)
{
}
```
PROBLEM 5 :  \textit{(It never stops: (4 points))}

For the following problems, you will be completing the class \texttt{WrappingSquares} that is intended to be animated in the \texttt{Canvas} class we have been developing in class. In the first part, you will complete the constructor and declare any additional instance variables if you need them. In the remaining parts, you will complete the \texttt{paint} and \texttt{update} methods of the class to draw and animate the shape.

\textbf{Part A} (4 points)

Complete the constructor for the class \texttt{WrappingSquares} that will behave as described on the following pages.

```java
public class WrappingSquares
{
    Point myCenter;
    Point myVelocity;
    Dimension mySize;
    Color myColor;
    // add any additional instance variables you may need

    public WrappingSquares (Point center, Dimension size, Point velocity, Color color)
    {
        // initialize instance variables here
    }
}
```
Part B (6 points)
Write the update method of the class WrappingSquares such that instead of changing its velocity when its center position encounters a wall, it should change the center position to the other side of the given bounds.

For example, if the width of the bounds is 600, then when the x-coordinate of the shape’s center becomes greater than 600, it should be reset to 0 (the other side of the bounds). Likewise, if the x-coordinate were to become negative, it should be reset to 600, the width of the bounds. In this way, the shape will appear to "pass through" or wrap around each edge of the panel so it does not disappear from view.

Complete the method update started below. You may refer to any instance variables you declared in the previous part:

```java
public void update (Dimension bounds) {
```
Part C (10 points)

Write the paint method of the class WrappingSquares such that it draws four squares that together form a larger square. Each of the smaller squares should be one-half the size given for the total square and arranged around the given center of the shape such that each is touching the center point with one of its corners. Thus they are organized much like the cartesian coordinate system, with each square representing on quadrant and the center of the shape representing the origin. The upper-left square then should have the color given in the constructor, with each successive quadrant having a darker color going around in clockwise order (so the lower-left square has the darkest color).

For example, if the shape is centered at (400, 200) and sized at (200, 200) then, going clockwise, the first square should be centered at (350, 150), the second at (450, 150), the third at (450, 250), and the fourth at (350, 250). All four squares should be sized at (100, 100). If the given color is Color.RED, then each of the squares should be a darker share of red, so the entire shape has four different colors.

Complete the method paint started below. You may refer to any instance variables you declared in the previous part:

```java
public void paint (Graphics pen)
{
...
```

}