Test 1: CompSci 6

75 Minute Exam

February 4, 2009

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.

Before starting, make sure your test contains 13 pages. The last page is blank and can be used as scratch paper, but **all pages must be turned in**.

None of the Java programs or program segments should have syntax errors unless stated. If you think there is a syntax error or you do not understand what a question is asking, then please ask. Import statements may not always be shown.

Do **not** discuss this test **with anyone** until the test is handed back.

	value	grade
Problem 1	8 points	
Problem 2	10 points	
Problem 3	8 points	
Problem 4	24 points	
Problem 5	20 points	
Extra Credit	5 points	
TOTAL:	70 points	

PROBLEM 1 : (*Understanding Java:* (8 points))

For each of the following questions below, please answer briefly in complete sentences.

1. Describe the purpose of the constructor method of a class and two differences between the constructor and any other class method.

2. Describe the conceptual differences between an (object) class and an (object) instance. Give two examples of each that emphasize those differences.

PROBLEM 2 : (*Maximum impact:* 10 points)

Part A: (5 points)

Complete the method `allDifferent` that returns true only if all three of the components of the given `Color` have the different values. For example, a call to `allDifferent` with a color that has red, green, and blue component values `(3, 128, 255)` should return true; while a color with component values of `(128, 200, 128)` should return false.

```
public boolean allDifferent (Color c)
{









}
```

Part B: (5 points)

Complete the method `max3` that takes three numbers and returns the one with the largest value. Your implementation may not use any conditionals or loops and **must** call the method `Math.max` at least once and use its result in determining the result of this function.

```
public int max3 (int a, int b, int c)
{









}
```

PROBLEM 3 : (*Timeless ... (8 points)*)

The method `addTime` below is intended to print the total time in hours, minutes, and seconds reprented by the given pair of minutes and seconds. However, as written, it contains both a syntax error (found by the compiler) and a logic error (not found by the compiler). Note, the line numbers below are not part of the code and none of the errors in the code have to do with line 5 that prints out the results of the calculations.

```
0   public void addTime (int min1, int sec1, int min2, int sec2)
1   {
2       int secSum = sec1 + sec2 % 60;
3       int minSum = ((min1 + min2) + ((sec1 + sec2) / 60)) % 60;
4       int hrSum = ((min1 + min2) + ((sec1 + sec2) / 60) / 60;
5       System.out.println(hrSum + ":" + minSum + ":" + secSum);
6   }
```

Part A (4 points)

As written `addTime` generates the *compilation error* below when compiled.

```
TimeCalculator.java:4: Insert ')' to complete expression
```

Describe in one sentence below why this error is generated and then show how to fix the problem by changing the code above to add the necessary closing parenthesis.

Part B (4 points)

After fixing the error in Part A, the code should run, but not print the correct results. For the following calls, it produces the following output (note, the last output is incorrect):

call	**printed results**
`addTime( 5, 30,  3, 15)`	0 : 8 : 45
`addTime(30, 30, 30, 60)`	1 : 1 : 30
`addTime(35, 30, 35, 45)`	1 : 11 : 75

Describe in one sentence below why these calculations are wrong and then show how to fix the problem by changing the code above.

PROBLEM 4 : (*Everythings included. (20 points)*)

Consider the class given below that models a rectanglular region as a `Point` and a `Dimension`. The point represents the top, left coordinate of the rectangle and the dimension represents its width and height. You will complete its methods as part of this problem.

```
public class Rectangle
{
    private Point myTopLeft;
    private Dimension mySize;

    public Rectangle (Point topLeft, Point bottomRight) {
        // to be implemented in Part A
    }

    public int getLeft () {
        return myTopLeft.x;
    }

    public int getRight () {
        return myTopLeft.x + mySize.width;
    }

    public int getTop () {
        return myTopLeft.y;
    }

    public int getBottom () {
        return myTopLeft.y + mySize.height;
    }

    public void translate (int xChange, int yChange) {
        // to be implemented in Part B
    }

    public void includePoint (Point pt) {
        // to be implemented in Part C
    }

    public Rectangle union (Rectangle other) {
        // to be implemented in Part D
    }
}
```

Problem continued on next page ...

Part A (5 points)

Wrtie the constructor for the `Rectangle` class that takes two points representing its oppostite corners. Do not create any additional instance variables, instead make the necessary calculations to compute the proper size of the rectangle.

```
public Rectangle (Point topLeft, Point bottomRight)
{









}
```

Part B (3 points)

Write the `Rectangle` method `translate` that takes two integer parameters, representing the number of pixels to move a rectangle horizontally and vertically, and moves the rectangle by changing its left and top coordinate. This method should return nothing, but instead change the state of the rectangle.

```
public void translate (int xChange, int yChange)
{









}
```

Part C (8 points)

Write the `Rectangle` method `includePoint` that takes a `Point` parameter, representing a point on the screen, and expands the rectangle minimally such that it contains the new point. This method should return nothing, but instead change the state of the rectangle.

For example, if the rectangle, `r`, has the top-left coordinate $(100, 100)$ and a size of $50x50$, then the following calls to `includePoint` result in the successive changes to the `r`:

Function call	**New State of Rectangle `r`**
`r.includePoint(new Point(105, 115))`	$(100, 100)$ $50x50$
`r.includePoint(new Point(155, 115))`	$(100, 100)$ $55x50$
`r.includePoint(new Point( 95, 115))`	$(95, 100)$ $60x50$
`r.includePoint(new Point(105, 155))`	$(95, 100)$ $60x55$
`r.includePoint(new Point(105,  95))`	$(95, 95)$ $60x60$

Complete the `Rectangle` method `includePoint` below:

```
public void includePoint (Point pt)
{









}
```

Part D (8 points)

Write the `Rectangle` method `union` that takes a `Rectangle` as a parameter, representing another rectangle, and returns a new rectangle that is minimally positioned and sized such that it covers the space that contains both rectangles. This method should not modify the state of the current rectangle or the rectangle passed as a parameter.

For example, if the rectangle, `r`, has the top-left coordinate $(100, 100)$ and a size of $50x50$, and another rectangle, `s`, has the top-left coordinate $(90, 110)$ and a size of $70x30$, then the union of these two rectangles is a rectangle with the top-left coordinate $(90, 100)$ and a size of $70x50$. Additionally, if one of the rectangles completely covers the other, the result is a new rectangle with the same values as the larger rectangle.

Complete the method `union` below:

```
public Rectangle union (Rectangle other)
{









}
```

PROBLEM 5 : (*Right on Target:* (24 points))

For the following problems, you will be completing the class `Target` that is intended to be animated in the `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 `paint` and `update` methods of the class to draw and animate the shape.

Part A: (4 points)

Complete the constructor for the class `Target` that will behave as described on the following pages.

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



    public Target (Point center, Dimension size, Point velocity, Color color)
    {









    }
```

Part B: (8 points)

Write the `paint` method of the class `Target` such that it draws itself as three 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 distance between each should be equal and each oval should be brighter than the one drawn before it, starting from the shape's given color.

For example, the largest oval's size should be 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.

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

```
public void paint (Graphics pen)
{









}
```

Part C: (8 points)

Write the `update` method of the class `Target` such that it expands its size based on its velocity instead of changing its position until its size causes one of the edges of the target to go outside of the given `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 target's edges hit the bounds. Thus if the target were positioned in the center of the bounds, it would stop growing when the dimensions grew to be larger than the `bounds` itself. If it were not centered, then it would stop sooner (when one of the four extreme points of the oval went outside the bounds).

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

```
public void update (Dimension bounds)
{









}
```

PROBLEM 6 : (*Find the Counterfeit* (extra credit - 5 points))

Part A: (2 points)

You are given nine gold coins but you suspect one of the coins is counterfeit and is lighter than real gold. To prove your suspicion, you decide to weigh the coins. However, looking around you find that you only have a standard balance, i.e., one that compares the weights of two quantities. There is an easy algorithm to determine which coin is fake by comparing each coin against another using, in the worst case, four separate balance measurements. Describe an algorithm that allows you to find the fake coin with only two measurements.

Part B: (3 points)

Now, you are given six bags of coins, each bag with a large number of coins in it. However, only one of the bags has genuine gold coins in it. The other bags have counterfeit coins in them. It is known that each coin of false gold weighs 100 grams and that each real coin weighs 110 grams apiece. This time, you find that you have only a simple scale that displays the weight in grams. Again, there is a easy algorithm to determine the real bag of gold that requires five weighings in the worst case. Describe an algorithm to find the bag of real gold with only one weighing?

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)
}
```

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 Color {
    // 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 ()
}
```

Math

```
public class Math {
  // Returns greater of the two given values
  public int max (int a, int b)
  // Returns lesser of the two given values
  public int min (int a, int b)

  // Returns sin of the given angle
  public double sin (double angleInRadians)
  // Returns cos of the given angle
  public double cos (double angleInRadians)

  // Returns smallest integer greater than the given value
  public double ceil (double value)
  // Returns largest integer smaller than the given angle
  public double floor (double value)
}

// do not create a Math object to call Math methods:
   int greater = Math.max(10, 2);
```

Graphics

```
public class Graphics {
    // Sets current color to the given color
    public void setColor (Color c)

    // Fills the specified rectangle with the current color
    public void fillRect (int x, int y, int width, int height)

    // Fills an oval bounded by the specified rectangle with
    // the current color
    public void fillOval (int x, int y, int width, int height)

    // Fills an elliptical arc covering arcAngle and bounded
    // by the specified rectangle with the current color
    public void fillArc (int x, int y, int width, int height,
                         int startAngle, int arcAngle)
}
```