Test 1: CompSci 6

Susan Rodger

October 6, 2005

Name (print): _____

Community Standard acknowledgment (signature): _____

	value	grade
Problem 1	12 pts.	
Problem 2	6 pts.	
Problem 3	14 pts.	
Problem 4	8 pts.	
Problem 5	25 pts.	
TOTAL:	65 pts.	

This test has 11 pages, be sure your test has them all. In addition to the test itself, you should have a list of common methods of the classes and the complete code for the class `Mover` that we have studied in class so that you do not need to memorize such details.

Do **not** spend too much time on one question — remember that this class lasts only 75 minutes and there are 5 problems on the exam. That means you should spend no more than *15 minutes per problem.*

Don't panic. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious.

In writing code you do not need to worry about specifying the proper `import` statements. Assume that all libraries and packages we have discussed are imported in any code you write.

If you think there is a syntax error or an ambiguity in a problem's specification, then please ask.

PROBLEM 1 : (*Short Answer:* (12 points))

For each of the following questions below, please answer *briefly* but completely.

1. For each of the visibility designations in Java — public, protected, and private — describe what classes are allowed to access the designated method or variable. For each, explain why you might want to declare a variable or method with such visibility.

   (a) private

   (b) protected

   (c) public

2. 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.

   (a) instance variable

   (b) parameter

   (c) local variable

PROBLEM 2 : (*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 `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 `isLeap` returns true if `year` is a leap year and false otherwise. Circle those that are correct. If a method is not correct, provide a value for `year` for which it returns the wrong value.

```
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;
}
```

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

```
public boolean isLeap (int year)
{
    if (year % 100 == 0) return false;
    if (year % 400 == 0 || year % 4 == 0) return true;
    return false;
}
```

PROBLEM 3 : (***What's the count? (14 points)***)

Part A (6 points)

The method `isAscending` given below is supposed to determine if the digits of the given number are in ascending order, i.e., it should only return true if each digit from the most to the least significant is strictly larger than the one before. For example, it correctly returns `true` for the numbers 123, 23567, and 2469. And it correctly returns `false` for the numbers 1243, 2413, and 2143. However, it should also return `false` for the numbers 2145, 4215, and 4125, but instead it returns `true`.

```
public boolean isAscending (int number)
{
    int lastDigit = number % 10;

    number /= 10;
    while (number > 0)
    {
        int currentDigit = number % 10;
        if (currentDigit >= lastDigit)
        {
            return false;
        }
        number /= 10;
    }

    return true;
}
```

Explain why the method does not work as intended and then modify the code above by adding or changing as few lines as possible to fix the problem.

Part B (8 points)

Write the method `countAscending` that returns a count of the numbers in a file whose digits are in ascending order. Each number in the file is separated by a single space.

Complete the method `countAscending` below. Your implementation **must** call the method `isAscending` given above at least once and use its result in determining the result of this function. In this part, you may assume that the method works correctly no matter what you wrote in Part A.

```
public int countAscending (Scanner input)
{




}
```

PROBLEM 4 : (*It seems so close:* (10 points))

Complete the sub-class of `Mover` called `MarkClosest` such that it finds another `Mover` whose center is closest to this one and draws a line from that one to this one. The class will be constructed with a collection containing the other `Mover` objects being animated. Each time the `move` method is called, it should determine the closest other `Mover` object in the collection and store that object in the instance variable `myClosest`. If more than one object is equally close, any of the close objects may be stored. Be careful, not to choose this `Mover` as the closest because its distance is always 0, the smallest possible.

Complete the method `move` of the class `MarkClosest` below. You *must* use the `distance` method of the `Point` class to determine the distance between the centers. Do not calculate the value directly.

```
public class MarkClosest extends Mover
{
    private ArrayList<Mover> myMovers;
    private Mover myClosest;

    public MarkClosest (Point center, Dimension size, Point velocity, Color color,
                        ArrayList<Mover> movers)
    {
        super(center, size, velocity, color);
        myMovers = movers;
        myClosest = this;   // something to give it a value, will be changed in move
    }

    public void paint (Graphics pen)
    {
        // draw line from this Mover to the closest one
        pen.setColor(Color.BLACK);
        pen.drawLine(getCenter().x, getCenter().y,
                     myClosest.getCenter().x, myClosest.getCenter().y);
        // rest of implementation not shown
    }

    public void move (Dimension bounds)
    {









    }
}
```

PROBLEM 5 : (*Above average?:* (25 points))

Consider the following superclass declaration used for this problem.

```
/**
 * A Shape represents a generic geometric shape that fits within a rectangle.
 */
public abstract class Shape
{
    private double myWidth, myHeight;  // dimensions

    public Shape (double width, double height)
    {
        myWidth = width;
        myHeight = height;
    }

    // returns specific dimension asked for
    public double getWidth ()
    {
        return myWidth;
    }

    public double getHeight ()
    {
        return myHeight;
    }

    // returns area of the geometric shape
    public abstract double area ();
}
```

Part A (6 points)

Briefly answer the following questions regarding the class above.

1. What does it mean for the class to be `abstract`?

2. What is the purpose of the class' constructor and why are the two assignment statements necessary?

3. Is the `Shape` class mutable or immutable? Defend your choice.

Part B (4 points)

The class `Circle` below extends `Shape` and overrides the `area` method to return the area for only a circle shape. Although the formula for computing the area of a circle is correct, the implementation of the `area` method given below has two problems with its use of the instance variables `myWidth` and `myHeight`: one is a compilation error and the other a logic error.

```
/**
 * A Circle represents a geometric circle.
 */
public class Circle extends Shape
{
    private double myWidth;

    public Circle (double diameter)
    {
        super(diameter, diameter);
    }

    // returns area of circle
    public double area ()
    {
        return Math.PI * (myWidth / 2) * (myHeight / 2);
    }
}
```

Explain the reason for each error and then modify the code above to fix each one.

1. syntax

2. logic

Part C (10 points)

Write the class `Rectangle` that extends `Shape` and overrides the `area` method to return its $width * height$. You should provide only those constructors and methods that are required in order for the class to compile.

Part D (7 points)

Complete the method, `readShapes`, that reads a data file and returns a collection of shapes. Assume that the text file with shape information being read is in the format shown below. Data about each shape is given on a separate line. The first word will either be `circle` or `rect`, indicating that either a `Circle` or `Rectangle` subclass should be created. If it is the former, the rest of the line will contain only one number, the diameter of the circle. If it is the latter, the rest of the line will contain two numbers, the width and height of the rectangle.

A sample data file representing five different shapes is shown below:

```
circle 100
rect 100 100
circle 13
circle 25
rect 10 20
```

Complete the method `readShapes` below:

```
public ArrayList<Shape> readShapes (Scanner input)
{
    final String CIRCLE = "circle";
    final String RECT = "rect";









}
```