CSCI 1227 Sample M2

1. **[10]** Declare and initialize the following variables. For example

   The number of students in this class.

   _int numStudents = 18;____

   (a) An object to represent the file named MyData.txt.

   ____

   (b) An object to read data from the file defined above.

   ____

   (c) An object to print information to the file named in part (a).

   ____

2. **[6]** Suppose the Scanner `in` has been declared and connected to the file `data.txt`. Write a loop to read and add up all the numbers in the file. (You do not need to print the sum; you may assume that there are no non-numerals in the file.)

3. **[4]** Consider the class and interface definitions on the **extra code page**, and the declarations below:

```
Rectangle r = new Rectangle(5, 20);
Measurable m = new Circle(25);
```

   Indicate which of the following methods calls are valid, and which will result in a compiler error.

a)	`r.getPerimeter()`	OK	Error
b)	`r.getHeight()`	OK	Error
c)	`m.getRadius()`	OK	Error
d)	`m.getArea()`	OK	Error

4. **[8]** Complete the following main method so that it creates a file named MyOutput.txt containing the line `This is my output file.`

```
import java.io.*;
public static void main(String[] args) {







}
```

5. **[6]** Revise the class below so that Collections.sort can sort a list of Thingy objects by value from smallest to largest.

```
public class Thingy
{
    private int value;

    public Thingy(int v)
    {
        value = v;
    }

    public int getValue()
    {
        return value;
    }




}
```

6. **[4]** This method ends the program when given a number less than zero. Revise it so that it throws an IllegalArgumentException instead. ~~Cross out~~ code that's not required, and write new code in the correct location(s).

```
public static void cheer(int n)

{

    if (n < 0)

    {

        System.err.println("No negatives allowed!");

        System.exit(1);

    }

    for (int i = 1; i <= n; i++)

    {

        System.out.println("Hip-");

    }

    System.out.println("Hooray!");

}
```

7. **[4]** Write the interface Usable so that the following method can compile.

```
public static void doThis(Usable u) {
    u.showValues();
    if (u.hasZeroValue()) {
        int zero = u.getZeroLocation();
        double val = u.solve(zero);
        System.out.println("solution @" + zero + " = " + val);
    }
}
```

8. **[4]** Match each of the Exceptions (left) with a time they might be thrown (right).

___ FileNotFoundException	a) addressed a question to a variable that's not referring to any object
___ IndexOutOfBoundsException	b) the user gave the wrong type of input
___ InputMismatchException	c) tried to access an element past the end of an ArrayList
___ NullPointerException	d) tried to read a file that doesn't exist
	e) tried to read from a Scanner with no more data

9. **[…]** Multiple Choice: select the *best available* answer from the options shown.

If the method `method` may result in the exception `MyException` being thrown, then the method header should be

a) `public void method() catch MyException`

b) `public void method() throw MyException`

c) `public void method() throw new MyException()`

d) `public void method() throws MyException`

e) `public void method() throws new MyException()`

If we don't close our file streams when we are done with them, then

a) eventually the operating system will not be able to open any more for us.

b) stuff we wrote to the stream may not appear in the file.

c) the files they are attached to may be unavailable for other programs to use.

d) we may not be able to open those files again later.

e) (all of the above)

If we have a Measurable variable m, and we want to check to see if it holds a Circle object, the way to do that is

f) if (Circle extends Measurable)

g) if (Circle implements Measurable)

h) if (Circle instanceof Measurable)

i) if (m extends Circle)

j) if (m instanceof Circle)

Which of the following is *not* a class that transfers data to/from a file?

a) File

b) FileInputStream

c) FileOutputStream

d) PrintWriter

e) Scanner

Extra Code Page

```
public interface Measurable {
    public double getArea();
    public double getPerimeter();
}

public class Circle implements Measurable {
    private double radius;
    public Circle(double r) {
        radius = r;
    }
    public double getRadius() {
        return radius;
    }
    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }
    @Override
    public double getPerimeter() {
        return getCircumference();
    }
    public double getCircumference() {
        return 2.0 * Math.PI * radius;
    }
}

public class Rectangle implements Measurable {
    private double width;
    private double height;
    public Rectangle(double w, double h) {
        width = w;
        height = h;
    }
    public double getHeight() {
        return height;
    }
    public double getWidth() {
        return width;
    }
    @Override
    public double getArea() {
        return width * height;
    }
    @Override
    public double getPerimeter() {
        return 2.0 * (width + height);
    }
}
```