CS1110 Lab 08. Abstract classes Fall 2008

Name _____ Section time _____ Section instructor _____

This lab introduces you to the concepts of an "abstract class" and "abstract method". The topic is covered in Section 4.7 of the class text and on lesson page 4-5 of the ProgramLive CD. Your lab instructor will present the concepts to you if you want (which are quite simple) —ask them. You will obtain a few Java classes from the course website, load them into Java, and change one of the classes from a normal class to an abstract class. You will be asked to look at the classes and modify them a bit. You will need a sheet of paper to write information about this lab. Show it to your instructor when you have finished. Because you probably don't have text with you, we summarize the material here.

Problem 1. In some situations, we don't want a programmer to instantiate (create instances of) a class. The class is there only to provide a superclass of other classes. But there is no way to prevent programmers from instantiating the class.

Solution. Change the class to an **abstract class**, because abstract classes cannot be instantiated. To do this, change the first line of the class, say class `C`, from

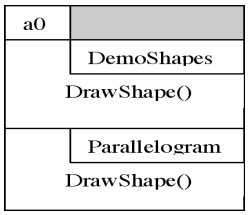
```
public class C {
```

to

```
public abstract class C {
```

Purpose of making a class abstract. Make a class abstract so that it cannot be instantiated (one cannot create an instance of it).

Problem 2. In abstract class `DemoShapes`, to the right, method `DrawShapes` is defined ONLY so that it can be overridden. We don't want programmers to call this method; they should call only the overriding methods in the subclass. But we can't force them to override the method, and if they don't, the one in `DemoShapes` will be called..



Solution. Change `DrawShapes` to an abstract method, because abstract methods cannot be called (but can be overridden). To do this, change `DrawShapes` as shown below. There are two changes: (1) keyword **`abstract`** is inserted and (2) the method body is replaced by a semicolon.

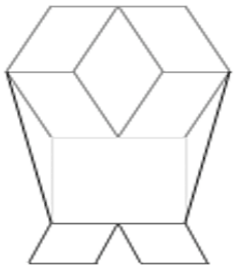
```
public void DrawShapes(...) { ...}
```

to

```
public abstract DrawShapes(...) ;
```

Purpose of making a method abstract. Make a method in an abstract class abstract so that it cannot be called.

Step 1. Open some files in DrJava. Start a new directory on your hard drive. Download these five files into the directory: DemoShapes.java Shape.java Parallelogram.java Rhombus.java Square.java. You can also obtain them by opening the course web page in a browser and clicking "Labs" in the lefthand column; this opens a page that has links to these files.



Open files DemoShapes and Shape in DrJava and compile. In the Interactions pane, create an instance of DemoShape:

```
d= DemoShapes();
```

A figure like that on the right (above) should appear. The output in the Java console is a description of seven shapes that are drawn in the window.

Step 2. Make class `Shape` abstract

(a) Open file `DemoShapes.java` and place the following statement in method `paint`, just before the declaration (and initialization) of variable `h`:

```
Shape s0= new Shape(5,5);
```

Execute the program; it should still run. On your paper, write what this statement does.

(b) Open file `Shape.java` and place keyword `abstract` just before keyword `class` in the class definition, so that the third line of the file looks like

```
public abstract class Shape {
```

You have made the class into an abstract class. Try executing the program again. Do you get an error message? Write down the error message and explain in a few words why it is an error. Now delete the statement that you placed in file `DemoShapes.java` in part (a) and run the program again. You should no longer have an error message.

Step 3. Make method `drawShape` of class `Shape` abstract

In file `Shape.java`, change method `drawShape` to:

```
public abstract void drawShape(Graphics g);
```

Note that the body `{}` is replaced by a semicolon. You have made this method into an abstract method. Execute the program; it should still execute.

Open file `Parallelogram.java` and comment out method `drawShape` (put `/*` before the method and `*/` after the method). Try to execute the program. Do you get error messages? Write on your paper the error message that deals with class `Parallelogram`. Write a few words explaining what the error is.

Remove the comment symbols, so that `drawShapes` is again defined in `Parallelogram`. Execute the program again just to be sure that you removed them correctly.

Step 4 Add Arms. Class `Shape` is designed to be the root of all classes that draw a shape. We have the following hierarchy: `Object` -> `Shape` -> `Parallelogram` -> `Rhombus` -> `Square`, because a square is a rhombus with angle 90 degrees, a rhombus is a parallelogram all of whose sides are equal, and a parallelogram is a shape.

The shape that appears when the program is executed looks almost like a person. It is drawn using methods of instance `g` of class `Graphics` that is attached to the `Frame` that opens. The only method you need from Graphics is setColor and getColor. You will do most of your work in this step 4 using the Shape classes.

You will give the person arms. All the changes you will make will be in class `DemoShapes`. Read through method `paint` of `DemoShapes`.

First, comment out the code that produces the two black lines (in DemoShapes). Hint: look for where the color is set to black.

Each arm is a green rectangle that is 60 pixels long and 20 pixels high. Its leaning factor (the third parameter of the Parallelogram constructor) is 0, which means that it is a rectangle. The leaning factor is defined on Lesson page 4.4 of the ProgramLive CD (see also the comment at the beginning of class Parallelogram), but you really don't have to read about it. Later, when you get the program going with leaning factor 0, you can try a different leaning factor, say 15, and see what it looks like.

The arms should be attached at the top right and top left of the square that makes up the body. The tops of the arms should be parallel to the top line of the square.

In writing the code that draws these rectangles, use the variables that are defined at the top of method `paint`. Also, use variables to contain all the constants that you need, as we did in method `paint`. You may have to move the whole figure to the right (by changing the value of variable `x`) so that you can see the whole picture. You must use class `Parallelogram`; you may not use method `drawRect` in class `Graphics`.

Hint: to figure out the coordinates for the arms, look at the positioning of the green square.

When you are finished, write on your paper the sequence of statements that you added to method `paint`.