Lab Assignment # 4

Creating Your Own Classes

Every class can contain data declarations and method declarations. Instance data is what we call the data that will be stored in every object of the class. The methods determine actions or behaviors of the objects.

In chapter 4 of the textbook, we're looking at creating classes to model objects in the real world. One of the examples is the Die class, which models the important data and behavior for a typical 6-sided die. (See example programs Die.java and RollingDice.java.)

The Die class includes two instance variables and 5 methods.

Encapsulation

Whenever possible, a class should be self-governing. It should control how other programs access its data and methods. Instance data of an object should only be modified by that object. That's why instance variables should be declared private. Only methods within the class can examine or change private variables.

Constructors

A constructor is a special method that is invoked when an object is created (instantiated). Most commonly a constructor is used to initialize the variables for an object.

A constructor method must be public. It has no return type, not even void. The name of a constructor method must exactly match the name of the class. Here is the Die class constructor

```
public Die ( ) {
   faceValue = 1;
}
```

This is considered a default constructor because it has no parameters. Constructors may have as many parameters as needed.

Always provide a default contructor for your class. If you need other constructors, add them after you have written the default constructor.

Accessor Methods

An accessor method provides read-only access to a particular instance variable. By convention, accessor method names begin with "get" followed by the name of the variable. Here is the accessor from the Die class.

```
public int getFaceValue( ) {
      return faceValue;
}
```

Mutator Methods

A mutator method is used to change the value of a particular instance variable. By convention, mutator method names begin with "set" followed by the name of the variable. Here is the mutator method from the Die class.

```
public void setFaceValue( int value ) {
      faceValue = value;
}
```

The toString Method

The purpose of the toString method is to create a String representation of an object. With the toString method, your class can control how objects look when they are printed. Here is the way the authors of our textbook wrote the toString method of the Die class.

```
public String toString( ) {
   String result = Integer.toString(faceValue);
   return result;
}
```

When you create a toString method for your class, use the method header exactly as shown above. The String that you return can be anything that you want. Think about how you want your objects to print.

Program # 1: WindTurbine.java

1. Create a class called WindTurbine. The WindTurbine class should have 3 instance variables and methods as follows:

- instance variables for powerOutput (double), bladeSpeed (int), and orientation (String).
  - powerOutput is the maximum output of the turbine, in kilowatts. Allowed values for powerOutput are from 1.0 to 10000.0. Default value is 100.
  - bladeSpeed is the current "speed" of rotation, in rpm (actually, it's a frequency not a speed). Allowed values for bladeSpeed are from 0 to 300. Default value is 0.
  - orientation is either "Vertical" or "Horizontal". No other values are allowed. Default value is "Horizontal".
  - All instance variables are private.
- a default constructor that initializes powerOutput to 100, bladeSpeed to 0, and orientation to "Horizontal".
- another constructor that accepts parameters for powerOutput, bladeSpeed, and orientation. This constructor must ensure that only allowed values are stored in the instance variables. If an invalid value is passed to this constructor, the corresponding instance variable should be set to its default value.
- an accessor "get" method for each instance variable
- a mutator "set" method for each instance variable. Mutators must ensure that only allowed values are stored in the instance variables. If an invalid value is passed to a mutator, the mutator should <u>not</u> change the instance variable.
- a toString method that returns a String containing a description of the WindTurbine. Example: "WindTurbine: powerOutput = 100 kW bladeSpeed = 80 orientation = Horizontal"

2. Save and compile the class. Debug as needed before going on to the next program.

You can compile WindTurbine.java by itself to debug any syntax errors. However, you cannot run WindTurbine by itself because it doesn't contain a main method.

Program # 2: TestLab4.java

1. Create a test (driver) program called TestLab4.java with a main( ) method to test the methods you created in the WindTurbine class. You should have TestLab4.java stored in the same directory as WindTurbine.java, so it is not necessary to use an import statement.

The main method in TestLab4.java should implement at least these steps:

- Create WindTurbine **wtOne** using the default constructor. Then, set the powerOutput to 200, bladeSpeed to 10, and orientation to "Vertical" using the appropriate mutator methods.

- Print **wtOne** with System.out.println. This should automatically call the toString method you have written.
- Change the powerOutput of **wtOne** to 150.
- Print the powerOutput of **wtOne.**
- Attempt to change the bladeSpeed of **wtOne** to -10.
- Print the bladeSpeed of **wtOne.**

- Create WindTurbine **wtTwo** using the constructor that accepts 3 parameters. Pass actual parameter values (800, 90, "Vertical").
- Print (on separate lines) the powerOutput of **wtTwo,** the bladeSpeed of **wtTwo**, and the orientation of **wtTwo**.
- Attempt to change the powerOutput of **wtTwo** to 0.
- Print the powerOutput of **wtTwo**.
- Attempt to change the orientation of **wtTwo** to "Diagonal".
- Print the orientation of **wtTwo.**
- Print **wtTwo** using System.out.println.

2. Save TestLab4.java and compile it.

In JGrasp, make sure that TestLab4.java is visible before you click the compile button.
This will automatically compile both programs.
Also, make sure that TestLab4.java is visible when you click the run button.

3. Debug your programs as needed.

Grading Note ☑

The TA will check to see if you have implemented each of the steps above. The TA will also run another test program with your WindTurbine.java to make sure that all the methods work correctly.

Submit: WindTurbine.java and TestLab4.java on Canvas.