Writing Classes in Java

Simple Classes I

1. Create a **Circle class**. It represents a circle of a given radius. The radius should be a decimal (double). The Circle has a getArea() command.
2. Create a **Shape2D interface** that represents the typical things you can do with a 2D shape. In this case, just create a getArea() method. Later you might add other abilities.
3. Create a **Rectangle class** that represents a rectangle with a given width and height. The Rectangle has a getArea() command.
4. [**Computer**]
   4a. Create a Monitor class that knows the width and height of a computer monitor. (Make this a subclass of Rectangle!)
   4b. Create a Hard Drive class that is initialized with a fixed (integer) capacity in gigabytes. It starts out empty.
   4b(i). The hard drive class has a method to useSpace(int howMuchSpace) to use up more space.
   4b(ii). It should also have a getAvailableSpace() method to tell how much remaining space is available on the drive.
   4b(iii). A reformat() method erases all of the data, emptying the hard drive.

   4c. Create a Computer class that has a monitor, a hard drive, and a clock speed (in GHz, typically a decimal like 2.2).
   4c(i). The computer has getMonitor(), getHardDrive(), and getClockSpeed() methods.
   4c(ii). It also has a getQuality() method which returns the sum of width*height of monitor plus hard drive size in GB plus clock speed to the third power.

Simple Classes II

1. Everything has a name, which is a String saying what it is commonly called. For example, bananas have a name "banana".
   1a. Write a Named interface that specifies that the object can understand the "String getName()" command.
   1b. Have all of your Fruits below implement this interface.
2. All Fruit has a calorie value that is retrieved by getCalories(). All Fruit also has a pH measurement (from Chemistry, usually 0.0-14.0) retrieved by getPH().
   2a. Write the Fruit class.
   2b. Write an Apple class with the typical food value for a medium apple 93 cal and a pH of 3.4. The constructor for a typical apple should take no arguments.
   2c. Add a constructor to the Apple class for different size apples, allowing you to specify the calories but not the pH of the apple.
   2d. Write a Banana class. pH = 4.8. calories = 120. Typical length = 8 inches.
   One constructor gives the generic banana with the specs just given.
   2e. Add a constructor that accepts a length argument and guesses the number of calories by assuming the ratio calories/length (inches) is constant for all bananas.
3. A Smoothie can be made by using a variety of fruits. For the purposes of this assignment, assume there are exactly two fruits used in the smoothie, but they can be the same or different types of fruit. The Smoothie object should know what two fruits it contains. The Smoothie can report its own getCalories() amount by summing the calories of the ingredients.

Medium Complexity Classes

1. A FruitList contains from zero to two fruits, inclusive (0<=count<=2). If you add more than two fruits to a list, the last fruit in the list is forgotten. A FruitList knows how many fruits are in the list.
   1a. A FruitList must start empty, but can have any name it wants (see Simple Classes II, #1). Write a logical constructor for a FruitList.
   1b. Write the method used below:

```
FruitList basket = new FruitList("granny's shopping bag");
Fruit newFruit = new Banana();
Fruit oldFruit = basket.addFruit(newFruit);
```

   This is a somewhat interesting idiom where if a fruit is pushed off the end of the two fruit list, it is returned by the function. Otherwise the function returns null.[1]
   1c. A FruitList has a "print()" method that prints the name of all three Fruits in the list using System.out.println(...).[2]

---

[1] If you learned from Karel the Robot, it is better to start off with a list full of NullFruit, so that the object that you return can still respond to method calls.
[2] Advanced: this is usually done by the "public String toString()" method instead. You need to know how to append strings using plus (+).

Simple Classes III

1. Create a Painter interface with functions: a void paint() function, an int getColor(), and a void setColor(int color), where color is an integer from 0 to 255.
2. Create a BasicPainter class that implements the Painter interface with ordinary getter and setter methods.
3. Create an AllBlackPainter class that always paints in black (0).
4. Create a BinaryPainter class that paints in black if asked to paint in a color less than 127, and white (255) if asked to paint in a color >= 127.
5. Create an AbstractArtPainter class that picks a random color (0-255) and every time it paints it just says System.out.println(current color). The getColor function returns the current random color. The setColor function ignores its argument and picks a new random color.
6. Create a PainterSquad class that makes a team of three painters and like a Choreographer just tells each painter what to do.
7. Subclass Banana to create a BananaPainter that implements the Painter interface. The BananaPainter's paint routine prints "BananaPaint #", where # is the banana's length. It ignores attempts to set its color. An interface allows you to do this.

HOW TO DO RANDOM NUMBERS

There are two choices to get a random number from 0-99. Both start with

```
double randomDecimal = Math.random() * 100;
```

Then in order to get a random integer 0<= number <= 99 you can do either of these:

1. Math.floor (randomDecimal); // OR
2. (int) (randomDecimal);