CMSC 113 – COMPUTER SCIENCE 1 (Prof. Kumar)
Lab#10-Creating Data Types/Objects & APIs

In this lab you will practice creating data types and their APIs

Task#1: Country – A data type
Let us design a new data type. The data file:
`~dkumar/CMSC113/LabPrograms/Lab10/countries.csv` contains information about the countries in the United Nations. For each country, the following information is provided:

- Country code: a three-digit UN code
- The 2-Letter country abbreviation
- The 3-letter ISO abbreviation
- The name of the country
- The capital of the country

For example, here are some sample entries:

```
056,BE,BEL,Belgium,Brussels
068,BO,BOL,Bolivia,La Paz
180,CD,COD,The Democratic Republic of the Congo,Kinshasa
398,KZ,KAZ,Kazakhstan,Astana
```

The first line of the data file contains the number of entries. Let us design a **`Country`** data type to load and store information about all the countries. Here is the design of the data type

---

```
public class Country

Country(String abbrev, String name, String capital) – Constructor

String getISO()         – Return the 3-letter ISO abbreviation
String getName()        – Returns the name of the country
String getCapital()     – Returns the name of the capital

String toString()       – prints out the object in the format:
                          "KAZ Kazakhstan (Astana)"
```

Given the **`Country`** data type API above, we can write a program to read the data from the data file and store it in an array. Then we can print out information about any given country by finding it first. Here is a starter program:

```
public static void main(String[] args) {
    Country usa = new Country("USA", "United States", "Washington");

    StdOut.println(usa);

} // main()
```

Implement the Country data type as defined above. For starters, **only implement the constructor, and the print method**. Then, you should be able to use the **`main()`** above to test the **`Country`** data type. Complete, write, compile, and test the program before proceeding.

Task#2: Reading in the data.

Modify the **`main()`** function from above to include:

```
// read in data
In infile = new In("/home/dkumar/CMSC113/LabPrograms/Lab10/countries.csv");

int n = Integer.parseInt(infile.readLine()); // read the # of entries in file

Country[] countries = new Country[n];     // Create array of size, n

for (int i=0; i < n; i++) {   // Read from file, one line at a time
    String s = infile.readLine();

    // Split the line (it is a comma separated file)
    String[] fields = s.split(",");

    // Extract the relevant fields (3-letter abbrev, name, and capital)
    String abbrev3 = fields[2];
    String name = fields[3];
    String capital = fields[4];

    // Create and store
    countries[i] = new Country(abbrev3, name, capital);
}

for (int i=0; i < countries; i++)   // Print out the data
    StdOut.println(countries[i]);
```

The above code reads in the data from the data file and stores the relevant information in an array of **`Country`** objects (**`countries[]`**). The **`In`** data type is described on page 354 of your text.

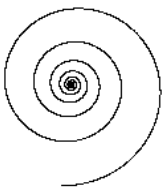
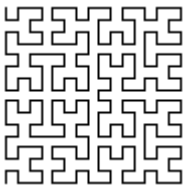
You now have enough to complete the definition of the **Country** data type.

If you have time, or at your leisure, try and build an application that plays the game of capitals. That is, the program prints out the name of the country, and the user must enter its capital. Have fun!

Optional Practice for later…for fun!

The Turtle Data Type

Next, we will design a simple system called Turtle Graphics. Imagine a turtle with a pen so that whenever it moves, the pen attached to it draws or traces its path. In our design, we will allow the turtle to simply move forward some steps, or turn. This simple functionality can then be used to create series of turtle movements that result in drawings that can range from simple shapes to some intriguing drawings as shown below:


Let us first model the turtle as an object (data type). Here are some more details about the turtle:

- The turtle lives in a 1.0x1.0 canvas (default canvas size).
- At any given time, the turtle has a position (x, y coordinates in the drawing canvas), and an orientation (angle in which it is facing, in degrees).
- To begin, a turtle must be created by specifying its location and orientation angle.
- Given an angle, say delta, the turtle can be asked to turn left delta degrees.
- Given a step amount, the turtle can be asked to move forward, in the direction it is facing.

Based on the above, we can design a class (data type) called **`Turtle`**. Its API is shown below:

`public class Turtle`	
`Turtle(double x0, double y0, double a0)`	*create a new turtle at (`x0`, `y0`) facing `a0` degrees counterclockwise from the x-axis*
`void turnLeft(double delta)`	*rotate `delta` degrees counterclockwise*
`void goForward(double step)`	*move distance `step`, drawing a line*

The above data type is already available to you (see Section 3.2 of your text). If you like, you can first experiment with the **`Turtle`** data type already provided. Below, we will implement our own.

Task#3: Implementing the Turtle class.
Based on the design above, define a new Java program file, **Turtle.java**. The turtle will require three private instance variables: **`x`**, **`y`** and **`angle`** (in degrees). It will need a constructor (as shown above), and two instance methods. Additionally, we will also add a print method: **`toString()`** with the following print representation of the turtle:

```
<Turtle at (X, Y) facing THETA degrees>
```

For example, a turtle object at (0.5, 0.5) facing North will be printed as:

```
<Turtle at (0.5, 0.5) facing 90 degrees>
```

Let us implement all this in Java:

```
// File: Turtle.java

// Purpose: To implement the turtle as a data type for doing turtle graphics.
public class Turtle {
    // Instance variables
    private double x, y;     // turtle is at (x, y)
    private double angle;    // degrees counterclockwise from x-axis

    // Constructor:
    // start at (x0, y0), facing a0 degrees counterclockwise from the x-axis
    public Turtle(double x0, double y0, double a0) {
        x = x0;
        y = y0;
        angle = a0;
    } // Turtle()

    // Instance Methods
```

```
    // rotate orientation delta degrees counterclockwise
    public void turnLeft(double delta) {
        angle += delta;
    } // turnLeft()

    // move forward the given amount, step
    public void goForward(double step) {
        double oldx = x;
        double oldy = y;
        x += step * Math.cos(Math.toRadians(angle));
        y += step * Math.sin(Math.toRadians(angle));
        StdDraw.line(oldx, oldy, x, y);
    } // goForward()

    // Print Method: <Turtle at (X, Y) facing THETA degrees>
    public String toString() {
        return "<Turtle at (" + x + ", " + Y + ") facing " + angle + " degrees>";
    } // toString()

    // Test Suite
    // goes here…
} // Turtle
```

READ the program above carefully. While doing so, read over the design shown on the previous page. It is very important that you understand how the design was formulated, and coded. Before proceeding, be sure to ask your instructor if you have ANY questions and/or clarifications.

Next, enter and compile the program shown above. When there are no compiler errors, you are ready to proceed to the next step.

Task#4: Testing the turtle

Now that you have implemented the **`Turtle`** API, it is time to test it. Test programs serve the purpose of running through all possible cases of the design to ensure that the implementation is correct in all aspects. Designing tests can be challenging when the thing you are testing is quite complex. In this exercise, we will give you a flavor of this process.

Let us write a set of commands for the turtle to draw a simple equilateral triangle. This will test the **`Turtle`** class. Typically, in Java such test "programs" are written in the **`main()`** function of the class defining the data type. This way, the tests stay with the API and can be run by any one at any time. This is what we will do. Below, we give an example test suite:

```
public static void main(String[] args) {
    // Declare and create a turtle
    Turtle koopa = new Turtle(0.5, 0.2, 60.0);  // A new turtle at (0.5, 0.2)
                                                // facing 60 degree
    System.out.println(koopa);      // Use the toString() method

    // Draw triangle
    double step = 0.5;
    koopa.goForward(step);
    koopa.tunLeft(120.0);
```

```
    koopa.goForward(step);
    koopa.turnLeft(120.0);

    koopa.goForward(step);
    koopa.turnLeft(120.0);
} // main()
```

Go ahead, enter the above in your **`Turtle`** class, compile, and run it. You should see a triangle (drawn by the turtle (**koopa**).

Task#5: Drawing Spirals with the turtle.
The program below, uses the **`Turtle`** API to draw logarithmic spirals. It takes an integer *n* and a decay factor as command-line arguments and instructs the turtle to alternately step and turn until it has wound around itself 10 times. This produces a geometric shape known as a logarithmic spiral, which arise frequently in nature. Define it in a file: **`Spiral.java`**

```
public class Spiral {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);  // # sides if decay = 1.0
        double decay  = Double.parseDouble(args[1]);   // decay factor

        double angle  = 360.0 / n;
        double step   = Math.sin(Math.toRadians(angle/2.0));
        Turtle turtle = new Turtle(0.5, 0.0, angle/2.0);
        for (int i = 0; i < 10*360; i++) {
            step /= decay;
            turtle.goForward(step);
            turtle.turnLeft(angle);
        }
    } // main()
} // Spiral
```

Try the program for values: (3, 1.0), (3, 1.2), (6, 1.2), (400, 1.0004), and (400, 1.0008) using the command:

```
$ java-introcs Spiral 3 1.0
```

See pages 394-402 of your text as well as Exercises at the end of Section 3.2 for more ideas on creating turtle graphics.