CIT 149: Java I
Chapter 8
Lab1

In this lab we will complete #3 on page 648. This will include several classes, including one tester class. When the tester class is run it will display information about the left arrow and at the end display the arrow. Let's get started:

The ShapeInterface

1. This will create an interface that will be used by another class, which will implement it. Its purpose is to draw simple shapes.
2. Open a new document window in TextPad and save the file as ShapeInterface.java.
3. First we will type the interface header and opening brace. Type:

```
public interface ShapeInterface
{
```

4. The interface will include several methods that have no body whatsoever. Comments should be included to designate the purpose of each. Type:

```
  /**
      Sets the offset for the shape.
 */

public void setOffset(int newOffset);

 /**
      Returns the offset for the shape.
 */

public int getOffset();

 /**
      Draws the shape at lineNumber lines down from the current line.
*/

public void drawAt(int lineNumber);

 /**
```

```
        Draws the shape at the current line.
    */

    public void drawHere();
```

5. Interfaces have specific methods that must be included within any class that implements the interface.
6. Close the interface and compile the program.

The ShapeBase class

This class is abstract. Recall that with abstract classes, you cannot instantiate the class. The class will contain some abstract methods. Abstract methods do not have a body. Any class that extends this class, must have methods with the same name as the abstract methods.

1. Open a new document window in TextPad and save the file as ShapeBase.java. This class will implement the ShapeInterface.
2. Create the class header and opening brace by typing:

```
public abstract class ShapeBase implements ShapeInterface
{
```

   - Notice that this class is abstract. This means that you cannot create ShapeBase objects. This class can be used only as a base class.

3. We have a single variable which represents the offsetting of the shape. Type:

```
private int offset;
```

4. Since this class is an abstract class, it must have a minimum of one abstract method. In this case we have only one abstract method. Abstract methods do not have a body. Type:

```
public abstract void drawHere();
```

5. The remainder of the code in this class are simply methods that should be able to understand by this time. Type:

```
public ShapeBase()
 {
    offset = 0;
}

 public ShapeBase(int theOffset)
```

```
{
    offset = theOffset;
}

 public void drawAt(int lineNumber)
 {
     for (int count = 0; count < lineNumber; count++)  //no braces are included
        System.out.println();                          //because it handles only one line of code

    drawHere();
}

public void setOffset(int newOffset)
{
    offset = newOffset;
 }

public int getOffset()
{
    return offset;
 }
```

6. Close the class, compile the program, fix any errors if necessary.

LeftArrow class

This class will extend, inherit, from the ShapeBase class.

1. Open a new document window in TextPad and save the file as LeftArrow.java.
2. Type the class header and opening brace, having the class extend the ShapeBase class.
3. We will declare two, private integers for the tail and base of the arrow. Type:

```
private int tail;
private int base;
```

4. Create your default constructor method. I want you to try creating this one on your own. Within the method:
    - invoke the constructor method of the superclass
    - set the default value of the variable *tail* to zero

- set the default value of the variable *base* to 3

5. Next we will overload the constructor method. The overloaded method will have parameters of three integers. Type:

```
public LeftArrow(int theOffset, int theTail, int theBase)
{
   super(theOffset);
   tail = theTail;
   if(theBase < 3)
      base = 3;

   else if(theBase % 2 == 0)  // Remainder = 0 is an even number
    {
      base = theBase + 1; // Add 1 to make it odd
      System.out.println("Added 1 to make the base odd");
    }

   else
      base = theBase;
}
```

- we first invoke a constructor method of the superclass. ShapeBase must have a constructor method with parameters of a single integer.
- we set the variable *tail* to equal the value of a variable passed to this constructor method.
- we check to see what the value of the variable theBase is. The value of this variable was passed to the constructor method.
    i. if it is less than three, change the value of *base* to 3
    ii. else if the remainder of the variable, after being divided by 2, equals 0 then set *base* to equal its value plus 1
    iii. else set *base* to equal its value

6. We overload the constructor method once again by typing:

```
public LeftArrow(int theTail, int theBase)

  {

     super();

     tail = theTail;
```

```
        if(theBase < 3)
            base = 3;

        else if(theBase % 2 == 0)  // Remainder = 0 is an even number
        {
            base = theBase + 1; // Add 1 to make it odd
            System.out.println("Added 1 to make the base odd");
        }

        else
            base = theBase;
    }
```

7. The set method will change values for the three parameters. Type:

```
    public void set(int newOffset, int newTail, int newBase)
    {
        setOffset(newOffset);
        tail = newTail;
        if(newBase < 3)
            base = 3;

        else if(newBase % 2 == 0)  // Remainder = 0 is an even number
        {
            base = newBase + 1; // Add 1 to make it odd
            System.out.println("Added 1 to make the base odd");
        }
```

```
    else
        base = newBase;
  }
```

8. The next two methods are mutator method. Type:

```
  /**
    Method to change the length of the tail.
   */
   public void setTail(int newTail)
   {
      tail = newTail;
   }

   /**
    Method to change the size of the base of the arrowhead.
   */
   public void setBase(int newBase)
   {
      if(newBase < 3)
         base = 3;

      else if(newBase % 2 == 0)  // Remainder = 0 is an even number
      {
         base = newBase + 1; // Add 1 to make it odd
         System.out.println("Added 1 to make the base odd");
```

```
        }

        else
            base = newBase;
    }
```

9. The writeOutput() method will simply invoke several other methods which are responsible for writing different parts of the arrow. Type:

```
    /**
       Method to write the values of the three parameters.
    */
    public void writeOutput()
    {
       writeOffset();
       writeTail();
       writeBase();
    }
```

10. The next three methods are the methods invoked by the writeOutput method. All three are void. Type:

```
    /**
       Method to write the value of offset.
    */
    public void writeOffset()
    {
       System.out.println("Offset = " + getOffset());
    }
```

```
/**
 Method to write the value of tail.
*/
public void writeTail()
{
   System.out.println("Tail   = " + getTail());
}


/**
 Method to write the value of base.
*/
public void writeBase()
{
   System.out.println("Base   = " + getBase());
}
```

11. Next we have our accessor methods which obtain the current value of the variables defined within this class, and return that value to wherever the method(s) are invoked. Type:

```
/**
   Method to return the value of tail.
*/
public int getTail()
{
   return tail;
}
```

```
/**
 Method to return the value of base.
*/
public int getBase()
{
   return base;
}
```

12. The drawHere method invokes several methods responsible for drawing the arrow. Type:

```
/**
 Draws the left arrow at the current line.
*/
public void drawHere()
{
   drawTop();
   drawTail();
   drawBottom();
}
```

13. The drawTop method draws the top of the arrow. Type:

```
/**
 Draws the top at the current line.
*/
public void drawTop()
{
```

```
    int linecount = getBase() / 2;
    int numberOfSpaces = getOffset() + linecount * 2;
    spaces(numberOfSpaces);
    System.out.println("*");
    int count;
    int insideWidth = 1;
    for(count = 1; count < linecount; ++count)
    {
      numberOfSpaces = numberOfSpaces - 2;
      spaces(numberOfSpaces);
      System.out.print("*");
      spaces(insideWidth);
      System.out.println("*");
      insideWidth = insideWidth + 2;
    }
  }
```

- By this time you should be able to read through the code and understand what is going on. If you have questions, please ask.

14. The next two methods draw the remainder of the arrow. Type:

```
  /**
    Draws the tail at the current line.
   */
   public void drawTail()
   {
     spaces(getOffset());
```

```
   System.out.print("*");
   int insideWidth = (int)(getBase() / 2) * 2 - 1;
   spaces(insideWidth);
   int count;
   for(count = 0; count <= tail; ++count)
      System.out.print("*");
   System.out.println();
}

/**
 Draws the bottom at the current line.
*/
public void drawBottom()
{
   int startOfLine = getOffset() + 2;
   int count;
   int linecount = getBase() / 2;
   int insideWidth = (linecount - 1) * 2 - 1;
   for(count = 1; count < linecount; ++count)
   {
      spaces(startOfLine);
      System.out.print("*");
      spaces(insideWidth);
      System.out.println("*");
      startOfLine = startOfLine + 2;
```

```
            insideWidth = insideWidth - 2;
        }
        // Print the last line

        spaces(startOfLine);
        System.out.println("*");
    }
```

15. The final method writes spaces that are required. Type:

```
    /**
        Method to write the indicated number of spaces.
    */
    public static void spaces(int number)
    {
        int count;
        for(count = 0; count < number; ++count)
            System.out.print(' '); //there is a space between two single quotes
    }
```

16. Close the class, compile the program, and fix errors as needed.

LeftArrowTest class

This class will test the other classes.

1. Open a new document window in TextPad and save the file as LeftArrowTest.java
2. Write the import statement that will import ALL classes from the java.util package.
3. Type the class header, opening brace, main method header and the opening brace for the main method.
4. First we will declare a variable of character type and construct a new Scanner object. Type:

```
char repeat;
 Scanner keyboard = new Scanner(System.in);
```

5. Within a do/while loop we will test constructor methods from the LeftArrow class. First we will test the default constructor. Type:

```
do  // Repeat if user says 'yes'
{
   // Test the three constructors (uses writeValues method)

   LeftArrow a1 = new LeftArrow();
   System.out.println("Using default constructor:");
   a1.writeOutput();
   System.out.println();
   System.out.println("==============================");
```

6. Next we test the other two constructor methods. Type:

```
LeftArrow a2 = new LeftArrow(3, 5, 7);
   System.out.println("Using fully specified constructor:");
   a2.writeOutput();
   System.out.println();
   System.out.println("==============================");

   LeftArrow a3 = new LeftArrow(9, 11);
   System.out.println("Constructor with just tail & base:");
   a3.writeOutput();
   System.out.println();
```

```
System.out.println("==============================");
```

7. We also want to test other methods. Type:

```
// Test the methods to change, return, and write values


System.out.println("Changing value of offset from 0 to 1:");
System.out.println();
System.out.println("Before:");
a1.writeOutput();
System.out.println();
a1.setOffset(1);
System.out.println("After:");
a1.writeOutput();
System.out.println();
System.out.println("==============================");


System.out.println("Changing value of tail from 0 to 2:");
System.out.println();
System.out.println("Before:");
a1.writeOutput();
System.out.println();
a1.setTail(2);
System.out.println("After:");
a1.writeOutput();
System.out.println();
System.out.println("==============================");
```

```
System.out.println("Changing value of base from 3 to 10:");
System.out.println("It should automatically be changed to 11"
   + " to make it odd.");
System.out.println();
System.out.println("Before:");
a1.writeOutput();
System.out.println();
a1.setBase(10);
System.out.println("After:");
a1.writeOutput();
System.out.println();
System.out.println("================================");

System.out.println();
System.out.println("Testing write offset:");
a1.writeOffset();
System.out.println();
System.out.println("Testing returnOffset:");
System.out.println("The offset returned = " + a1.getOffset());
System.out.println();
System.out.println("================================");

System.out.println();
System.out.println("Testing write tail:");
```

```
a1.writeTail();
System.out.println();
System.out.println("Testing returnTail:");
System.out.println("The tail returned = " + a1.getTail());
System.out.println();
System.out.println("==============================");

System.out.println();
System.out.println("Testing write base:");
a1.writeBase();
System.out.println();
System.out.println("Testing returnBase:");
System.out.println("The base returned = " + a1.getBase());
System.out.println();
System.out.println("==============================");

// Test the actual drawing of left arrows
// using the drawHere method

System.out.println();
System.out.println("RESET to");
System.out.println("Offset = 0, tail = 3, base = 5:");
// Note: the base should be changed automatically to 7
// to make it odd.
a1.set(0, 3, 5);
```

```
a1.writeOutput();
System.out.println();
a1.drawHere();
System.out.println();
System.out.println("================================");

System.out.println();
System.out.println("RESET to");
System.out.println("Offset = 1, tail = 4, base = 7:");
a1.set(1, 4, 7);
a1.writeOutput();
System.out.println();
a1.drawHere();
System.out.println();
System.out.println("================================");

System.out.println();
System.out.println("RESET to");
System.out.println("Offset = 3, tail = 5, base = 6:");
a1.set(3, 5, 6);
a1.writeOutput();
System.out.println();
a1.drawHere();
System.out.println();
System.out.println("================================");
```

```
// Now test the drawing of a left arrow
// using the drawAt method.

System.out.println();
System.out.println("The arrow parameters are:");
a1.writeOutput();
System.out.println();
System.out.print("This arrow should display ");
System.out.print("five lines down.");
a1.drawAt(5);
System.out.println();
System.out.println("==============================");
```

- I recommend that you type each section separately.

8. We give users the option of whether that want to test things again. Type:

```
System.out.println("Do again? (Y for Yes, or N for No)");
repeat = keyboard.next().charAt(0);
```

- The charAt() method checks the character that was typed at a specific position. The position of 0 is the first character typed, all other characters will be ignored.

9. We close the do/while loop, specifying as to whether the loop will be repeated. Type:

```
}while((repeat == 'y') || (repeat == 'Y'));
```

10. Close the main method and the class.
11. Compile the program, fix any errors if necessary.
12. Run the program to make certain it runs properly. It should display information about the arrow, followed by displaying the arrow shown on page 648, except that it will point in the opposite direction.

13. Compress the following files into a single compressed file and submit to the appropriate drop box:

    ShapeBaseInterface.java
    ShapeBaseInterface.class
    ShapeBase.java
    ShapeBase.class
    LeftArrow.java
    LeftArrow.class
    LeftArrowTest.java
    LeftArrowTest.class