CIS 255 • Java Programming
Laboratory Assignment 8: Classes and Objects

In this laboratory assignment you will work with Java class syntax to create a class to store and retrieve information about a circle; you will then instantiate this class in three different circle objects.

Background

The example class in the textbook stores the length and width of a rectangle and allows a programmer to obtain the length, width, and area of that rectangle. For a circle, the only attribute needed to be stored is the radius; a programmer can then use methods to determine the diameter, area, and circumference of the circle.

Step 1: Launch TextPad and Save a New Document

You'll be using the starter file when you instantiate the class later on, but for now, you need to launch TextPad (or a similar Java code editor) and start a new document. Remember that TextPad highlights syntax based on the file extension, so you need to save the file as a Java file before you start typing. Since the name of your class will be `Circle`, save the file as **Circle.java** where you want the file to be stored.

Step 2: Write the Class Header and the Field Declaration

Start your file with comments at the top that include your name, the course number, the laboratory assignment or project number, and a brief description of the program's contents. You may also wish to include the date (perhaps after the laboratory assignment number). As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire class definition to add comments.

This file will not require any `import` statements, so the next item you should write is the class header.

When you write an object-oriented class definition, the name of the class (and of the file) will also be the type used to declare variables from this definition. Name your classes carefully!

Write the two brackets for the class body. After the opening curly bracket, you'll need to declare one field: a `private` variable of type `double` named `radius`. Remember: fields are declared inside the class brackets alone, not inside a method.

Step 3: Write the Constructor Definitions

You will write two constructors for this class: a no-arg constructor and a constructor with a single parameter from which `radius` will take its value. The no-arg constructor is a `public` method with no return type, a method name identical to the class name, and empty parentheses.

The header of a constructor method should not contain a return type!

In the body of this first constructor, assign `radius` a "default" value of 1.0 (`Circle` objects will begin with a `radius` of 1.0 unless otherwise specified).

The second constructor is a `public` method with no return type, a method name identical to the class name, and a parameter list in parentheses containing a variable of type `double`. Name this parameter something other than `radius` so the compiler won't be confused.

> In an object-oriented class definition, be careful not to give a parameter or other local variable the same name as one of the fields! Java will not have access to the field in such a method.

In the body of this constructor, use an `if` statement to see if the parameter's value is positive. If so, assign the parameter to `radius`; if not, display an error message in the console, and assign the "default" value of 1.0 to `radius`.

> Whenever a method could assign a parameter's value to a field, validating the parameter ensures that the field won't be assigned an invalid or illogical value. In a constructor, there should be a "default" alternative value to assign to the field when the parameter's value cannot be used.

Step 4: Write the Mutator Method Definition

Since the class contains only one field, `radius`, only one mutator method is needed. This method should have a return type of `void`, its name should begin with `set`, and it should have a single parameter that matches the type of the field (`double`) but has a different name. In the body of this method, if the parameter's value is positive, assign it to the field; otherwise, display an error message in the console letting the user know that the existing value of the field will be maintained.

> In a mutator method, don't replace the existing value of a field when the parameter's value cannot be used. The existing value may be more meaningful than the default value.

Step 5: Write the Accessor Method Definitions

The class needs one accessor method for the field `radius` and three additional accessor methods for the calculated values diameter, area, and circumference. Each method should have a return type matching the type of the value to be returned (`double`), a name that begins with `get`, and an empty parameter list.

> Don't declare a method's local variables in the parentheses of its header! Only declare variables in the parentheses (parameters) if they must receive arguments from the calling method.

The body of the accessor method for `radius` should simply return the value of the field. The body of the accessor methods for the calculated values should return the values from these formulas:

- Diameter: 2 * radius
- Area: π * radius * radius (or π * radius^2)
- Circumference: π * diameter (or π * 2 * radius)

> Java has a built-in constant for π (`Math.PI`). Also, since the circumference is based on the diameter, you could have the accessor method for circumference call the accessor method for diameter.

You may use temporary work variables to store the calculated quantities before you return them, or you may place the formulas inline within the `return` statement.

Don’t declare fields for calculated values! Since diameter, area, and circumference are dependent on the value of radius, they should be calculated when the methods are called and not stored in fields within the objects.

Step 6: Compile Your Class Definition

To determine if there are any syntax errors in your class definition, and to generate the byte code once your source code is free of syntax errors, you’ll need to select the item **Compile Java** from the **External Tools** submenu of the **Tools** menu. The results of the build process will appear in the **Tool Output** area below your code. If the Tool Output area indicates that there are syntax errors in your code, you may double-click the line that describes each syntax error to see the line that contains the error (or, in some cases, the line after the error).

Once you receive a message in the Tool Output area that the tool completed successfully, your class may be free of errors, but you cannot execute an object-oriented class definition that does not contain a `main` method. You’ll write your `main` method in a separate source code file.

Step 7: Download the Starter File and Open It in TextPad

On Blackboard, click the link for the starter file to download it. Make sure you insert your first initial and last name where indicated (**FLast_Lab08.java**) and that you save the file in the same location where `Circle.java` is saved. Once you have downloaded the file, open it in TextPad or a similar Java code editor.

Step 8: Customize Your Code and Add `import` Statements

Edit the comments at the top of the code to include your name, the course number, the laboratory assignment or project number, and a brief description of the program’s contents. You may also wish to include the date (perhaps after the laboratory assignment number). Also, modify the class name to match your file name. As you continue your code, you may go ahead and add comments to the statements or groups of statements that you believe deserve additional explanation, or you may wait until after you have written your entire program to add comments.

This program will deal with interactive input, so you will need to decide whether to use the console or dialogs; include the `import` statement that corresponds to your choice.

Step 9: Declare Variables in `main` for Primitive Values and Input / Output

You’ll need at least one variable to store the user’s input for the radius of a `Circle` before it is passed to a constructor or mutator. If you are using the console for input and output, you will need to declare a `Scanner` object for `System.in`; if you are using dialogs, you will need a temporary `String` variable to hold the value returned by `showInputDialog` before it is converted to the proper type (if necessary).

Step 10: Create Three Instances of the Class Using Constructors

Your program should contain **THREE** `Circle` objects. The first object should be created using the no-arg constructor. The second object should be created using a hard-coded value for the radius (your choice, as long as the value doesn't match the default value of 1.0).

Don't insert the entire definitions of the constructors here! Write a statement for each object that declares a variable of the class type and assigns a value from a constructor call.

Don't give an object the same name as one of its fields. The object contains the fields but is not the same as its fields, and you don't have to declare a separate object for each field when the class contains multiple fields.

To use the no-arg constructor, leave the parentheses empty.

Before you create the third `Circle` object, prompt the user for its radius. Then, pass the variable storing the user's input as the argument to the constructor.

The argument to a constructor call can be a literal value, a variable, or an expression as long as the data type of the argument matches the data type of the parameter.

Step 11: Display the Objects' Field and Calculated Values Using Accessors

For each of the three `Circle` objects you have created, display the radius, diameter, area, and circumference using the accessor methods (a total of **twelve** lines of output). The calls to the accessors may be placed within the output statements so that you are not required to store them in an intermediate work variable. You may wish to use `printf` or `DecimalFormat` to limit the number of digits shown after the decimal point.

You must use the accessor methods to obtain the values of the fields out of the objects. Using the object name by itself in an output statement displays the address of the object instead of the values of the fields.

Example:
```
Circle #1:
Radius:  1.0
Diameter:  2.0
Area:  3.1415927
Circumference:  6.2831853
```

It's a good idea to label the output for each object separately (e.g., `Circle #1`). This isn't done automatically, so you should include this as a separate line of output in the console or the dialog box for that object.

Step 12: Modify an Object's Field Using a Mutator and Display the Changes

Prompt the user for another radius. Then, pass the variable storing the user's input as the argument to the mutator method invoked on one of your three `Circle` objects.

Do not call the constructor here! This step instructs you to call the mutator, which is a method that modifies an existing object.

Display the new radius, diameter, area, and circumference of this modified object after the mutator call, using the accessors as in step 11.

When the program's execution is complete, the user should have seen the set of `Circle` values (radius, diameter, area, and circumference) a total of four times (three from the original constructor values, once after the mutator has been called).

Step 13: Compile Your Code and Execute the Byte Code

When you compile a program that makes use of another class definition, the code in this program will be validated against the syntax of the class definition. You will receive errors if you attempt to use a constructor, mutator, or accessor method in a way that is inconsistent with its definition within the other class. Compile and correct these errors until you have successfully removed any errors that exist.

If the compiled version of your class (the `.class` file) is not in the same folder as the program that instantiates the class, Java will not be able to find it, and you will receive multiple syntax errors.

In the **External Tools** submenu of the **Tools** menu, select **Run Java Application**. Enter some test values to verify that the results match what you believe they should be. To test multiple sets of input, you'll need to run the program multiple times. Make sure that you test some invalid (non-positive) values for the radius passed to the constructor and to the mutator so that you can verify the program's behavior.

Step 14: Add Inline Comments and Submit Your Code

Make sure that **each file** has **three or more inline comments** (a total of **six comments**) near the statements that you believe warrant the additional description (especially those that deal with classes and objects). These can be simple line comments at the end of a line of code that needs explanation, or block comments where each precedes a series of statements described in the comment.

Follow the instructions on the course website to attach and submit your source code.

Make sure you complete the submission process! If you attach your file without submitting the assignment, your file will not be received!