Creating a Class

Introduction to OO Programming 2

Introduction

- Object oriented programming languages allow the use of both ***primitive*** and ***object*** types.
- A Java programmer can design and implement their own classes, or can instantiate objects from classes defined by others.
- So far we have been instantiating objects from classes defined in the Java library – `String` and `Rectangle`
- The next step is to define our own Class and to instantiate objects of that Class.

Using the String class

File `String.java`
(In package java.lang)

```
public class String
{

// instance variables


// methods defined here


}
```

File `NameLength.java`

```
public class NameLength
{

public static void main(String[] args)
{

   // declare variables
   String myName = new String();
   myName = "Walter White";

   // obtain number of characters
   System.out.print(myName.length());

}
```

Using the Rectangle class

File `Rectangle.java`
(In package java.awt)

```
public class Rectangle
{

// instance variables


// methods defined here


}
```

File `RectangleTest.java`

```
import java.awt.Rectangle;

public class RectangleTest
{

public static void main(String[] args)
{
// declare an object of type Rectangle
Rectangle box = new Rectangle();
box.setLocation(10, 14);

System.out.println("x: " + box.getX());
System.out.println("y: " + box.getY());

}
```

Using Java Classes

- When we use Java classes, we look at the API to inspect the ***public interface*** of the class
- We only need to know about
  - methods of the class
  - Instance variables of the class
- We do not see the code or implementation detail of the class

Steps in Designing a Class

1. Identify the object / objects that are required by the program
2. For each object
   a. Identify the properties of that object (instance variables)
   b. Identify the behaviours of that object (methods)
3. Create a UML class diagram
4. Implement the class (i.e. write the code for the class)
5. Write a tester containing the `main` method, where objects will be instantiated and tested

Syntax : Class Definition

```
accessSpecifier class ClassName
 {
    //instance fields or variables
    //constructors  //will deal with later
    //methods
 }
```

Example:

```
 public class Oblong
{
   // instance variables or the attributes
   private double length;
   private double height;

   // methods
   public void setLength(double lengthIn)
   {...}
```

Purpose:

To define a class, its public interface, and its implementation details

Objects

- An object represents an entity that can be distinctly identified
  - A student, a circle, a postcard, a loan, an oblong, …
- An object has state and behaviours
  - The ***state*** of an object is represented by the values held in its data fields (properties or instance variables)
  - The ***behaviour*** of an object is defined by a set of ***methods***. To invoke or call a method on an object is to ask the object to perform a task.

Oblong

- An Oblong is a four sided rectangular shape which is defined by its height and width.
- Similar to the Rectangle, but does not store information on its location.
- We will define what an Oblong is in a file called `Oblong.java`.
- Once we have done that, we can instantiate or create as many Oblong objects as we want.

Design a class `Oblong`

- **An Oblong is a four sided rectangular shape which is defined by its height and width.**
- What *properties* (or data) should an object of type oblong have?
  - Identify these as instance variables
  - What data type are they?
- An oblong should store its own width and height
  Should they be `int` or `double`?

Design a class `Oblong`

- **An Oblong is a four sided rectangular shape which is defined by its height and width.**
- What *operations* (or methods) should we need to apply to an object of type oblong?
  - Identify these as methods
    - Should the method return a value?
    - Should the method take an argument?

Design a class `Oblong`

- Identify some simple operations such as
  - get the value of height
  - get the value of width
  - set the height to some value
  - set the width to some value
  - calculate the area

Think about getters and setters

- We may identify other methods later in practical

Design a class `Oblong`

- Think of appropriate method names for each
  - get the value of height — `getHeight()`
  - get the value of width — `getWidth()`
  - Set the height to some value — `setHeight()`
  - Set the width to some value — `setWidth()`
  - Calculate the area — `calculateArea()`

getters and setters

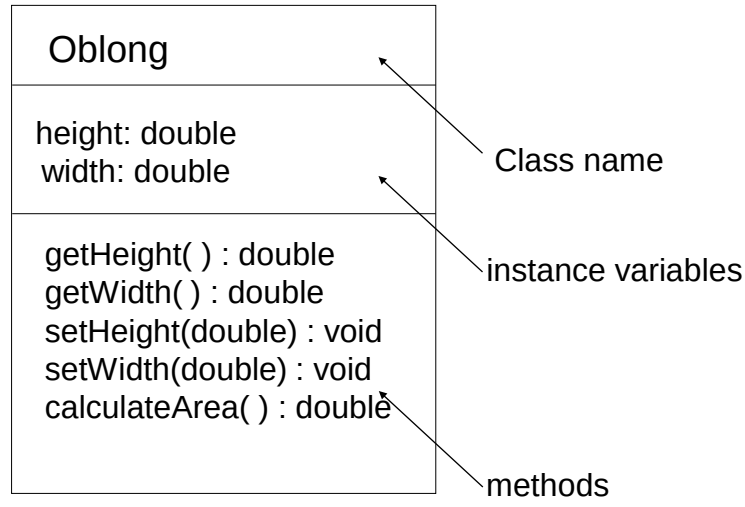
For each method think about returnType (output) and argument list (input)

Method Name	Input (argument list)	Output (return type)
`getHeight()`	None	`double` (the value of height)
`getWidth()`	None	`double` (the value of width)
`setHeight()`	`double heightIn`	none
`setWidth()`	`double widthIn`	none
`calculateArea()`	none	double (the area)

UML

- Unified Modelling Language is a methodology used in designing and documenting object oriented applications
- UML includes various diagrams for modelling OO applications
- A UML class diagram depicts a class using a box with three sections:
  - The *Class name* is placed in the top
  - *Instance variables* are placed in the middle
  - *Methods* (including *Constructors*) are placed in the bottom

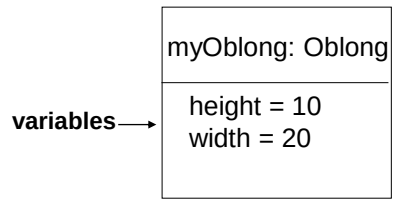
UML Class Diagram - Oblong



An Oblong object

```
//create an Oblong object called myOblong
Oblong myOblong = new Oblong();

myOblong.setHeight(10);
myOblong.setWidth(20);
System.out.println(myOblong.calculateArea());
```




```
// Oblong Class
// Class representing a four sided rectangular shape

public class Oblong
{
   // instance variables
   private double height;
   private double width;


   // methods
   public double getHeight()
   {
      return height;
   }


   public double getWidth()
   {
      return width;
   }
```

```
// Oblong Class continued


   public void setWidth(double widthIn)
   {
      width = widthIn;
   }

   public void setHeight(double heightIn)
   {
      height = heightIn;
   }

   public double calculateArea()
   {
      return width * height;
   }

}// end of class
```

```
// OblongTester.java
// Tester class for Oblong

public class OblongTester
{
   public static void main(String [] args)
   {
     // create an instance of type Oblong
     Oblong myOblong = new Oblong();

     // use methods to set instance variables
     myOblong.setHeight(10);
     myOblong.setWidth(20);

     // use methods to retrieve values of instance variables
     System.out.println("Width: " + myOblong.getWidth());
     System.out.println("Height: " + myOblong.getHeight());
     System.out.println("Area: " + myOblong.calculateArea());

   }
}
```

Designing the Class

```
public class Oblong
{ …
  …
}
```

- The class has no **`main( )`** method because it will never be run. Oblong is a template from which objects of type Oblong will be created.
- This class is saved in a file called `Oblong.java`. This will be saved in the same directory as any tester files.

Instance Variables

```
//accessSpecifier type name;
// instance variables
private double height;
private double width;
```

- Every instance of an Oblong will have its own height and width. These are the ***instance variables*** and are of type **`double`**.
- The keyword **`private`** indicates that the instance variables are only accessible through the public methods of that Class.
- This protects them from outside interference, and is part of an important Object Oriented concept called ***encapsulation***.

Instance Fields/Variables

- An *instance field* declaration

```
accessSpecifier type name;
```

  - access specifier
    - normally `private`
  - type of variable (primitive or class)
    - such as `double` or `int` or `String`
  - name of variable
    - such as `height` or `balance`
- Each object of a class has its own set of instance fields/variables
- declare all instance fields as private

Methods

```
public double getHeight()
{
   return height;
}
```

- Each method defines an operation that objects of the class can perform, usually by accessing or manipulating its instance variables
- This method returns a **`double`** representing the value of the **`height`** instance variable of an instance of the oblong class.

Method signature

```
public double getHeight()
```

- The first line of code is known as the **method signature**.
- The method signature defines how it is invoked or called. This includes information on:
  - Method name `getHeight()`
  - Input parameters `none`
  - Return type `double`
  - Access specifier `public`

Method input parameters

```
public void setHeight(double heightIn)
{
   height = heightIn;
}
```

- Input parameters will appear inside the brackets in the method signature. This tells the programmer what extra information a method needs to perform its task.
- This method needs one input parameter of type `double`.

```
myOblong.setHeight(10);  //method call
```

- When the method is called the value 10 is passed to the parameter variable **`heightIn`**.
- The value of **`heightIn`** is then assigned to the instance variable **`height`** for **`myOblong`**.

Method return type

```
public double getHeight()
{
   return height;
}
```

- This method does not need any input parameters, as indicated by the empty brackets after the method name.
- The method returns the value of the **`height`** instance variable for a particular object. This value is returned as a **`double`**.

```
System.out.println("Height: " + myOblong.getHeight());
```

Method access specifier

```
public double getHeight()
{
   return height;
}
```

- A method's access specifier defines how other Java files can call on it.
- Here, the access specifier is **`public`**, which means that it can be called from other classes.
- Methods are generally defined as **`public`**.
- Instance variables are usually defined as **`private`**.

Method body

```
public double getHeight()
{
   //method body
   return height;
}
```

- A method's body is the code which appears between the curly brackets.
- This code is usually brief, as methods are set up to perform one specific task.
- Here, the method uses the `return` keyword to return the value of the **`height`** instance variable of that object.

Method body

```
public void setHeight(double heightIn)
{
   height = heightIn;
}
```

- Here, the method sets the value of the `height` ***instance variable***.
- A value (10) is passed to the ***parameter variable*** `heightIn` in the method call

```
myOblong.setHeight(10);
```

- This results in the setting of the **`height`** instance variable of **`myOblong`** to 10.

Using the Oblong class

- The code which defines the Oblong is saved in a file called **`Oblong.java`**.
- This file will have no **`main( )`** method as it will never be run.
- The code which instantiates and uses **`Oblong`** objects will be declared in a separate Java file e.g. **`OblongTester.java`**
- This file **will** have a **`main( )`** method.

```
// OblongTester.java
// Tester class for Oblong

public class OblongTester
{
   public static void main(String [] args)
   {
     // create an instance of type Oblong
     Oblong myOblong = new Oblong();

     // use methods to set instance variables
     myOblong.setHeight(10);
     myOblong.setWidth(20);

     // use methods to retrieve values of instance variables
     System.out.println("Width: " + myOblong.getWidth());
     System.out.println("Height: " + myOblong.getHeight());
     System.out.println("Area: " + myOblong.calculateArea());

   }
}
```