Defining Classes for Objects

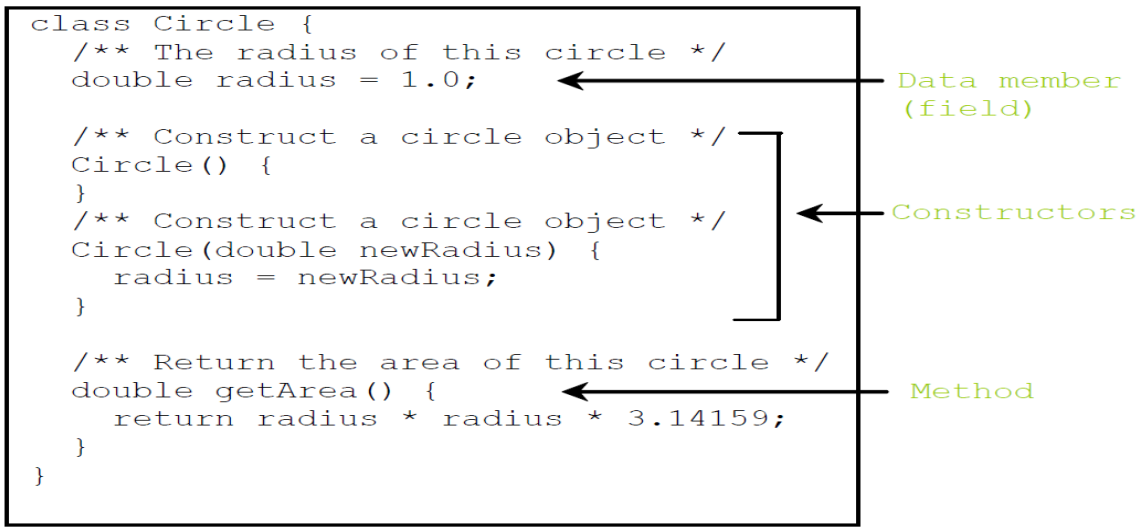
Object-oriented programming (OOP) involves programming using objects. An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.

An object has a unique identity, state, and behavior.

- The *state* of an object (also known as its *properties* or *attributes*) is represented by *data fields* with their current values.
  A circle object, for example, has a data field **radius**, which is the property that characterizes a circle.
  A rectangle object has data fields **width** and **height**, which are the properties that characterize a rectangle.

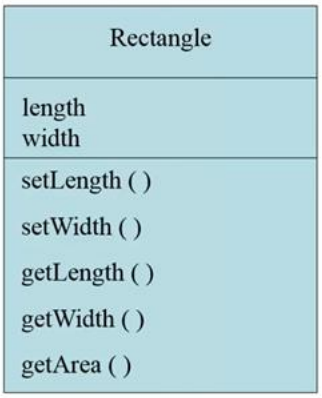
- The *behavior* of an object (also known as its *actions*) is defined by methods. To invoke a method on an object is to ask the object to perform an action. For example, you may define a method named **getArea()** for circle objects. A circle object may invoke **getArea()** to return its area.

A Java class uses variables to define data fields and methods to define actions. Additionally, a class provides methods of a special type, known as constructors, which are invoked to create a new object. A constructor can perform any action, but constructors are designed to perform initializing actions, such as initializing the data fields of objects.



Write a Class, Step By Step

- A Rectangle object will have the following fields:



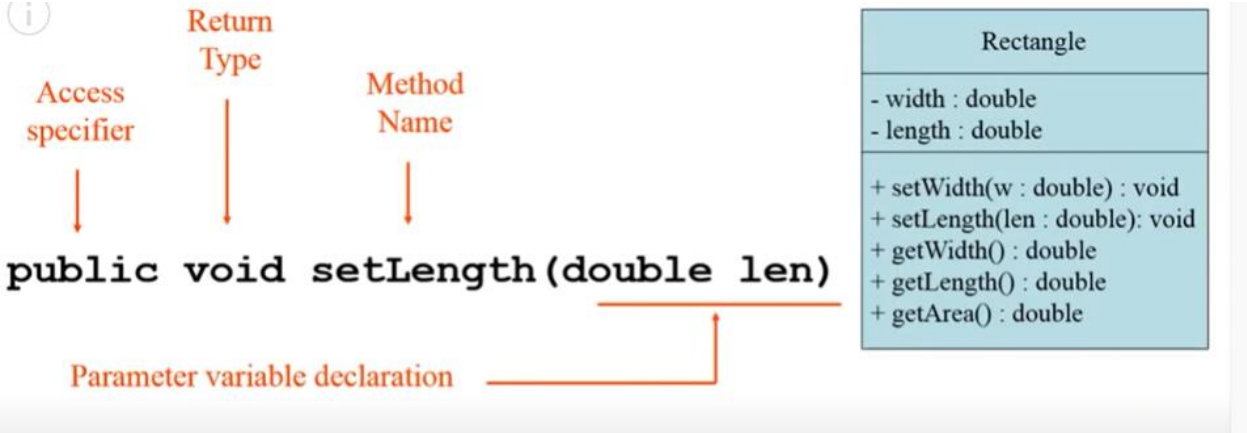

```
public class Rectangle {
        private double length;
         private double width;

}
```

Access Modifier

- An access modifier is a Java keyword that indicates how a field or method can be accessed.
- **public**
  - When the `public` access modifier is applied to a class member, the member can be accessed by code inside the class or outside.
- **private**
  - When the `private` access modifier is applied to a class member, the member cannot be accessed by code outside the class. The member can be accessed only by methods that are members of the same class.

data hiding

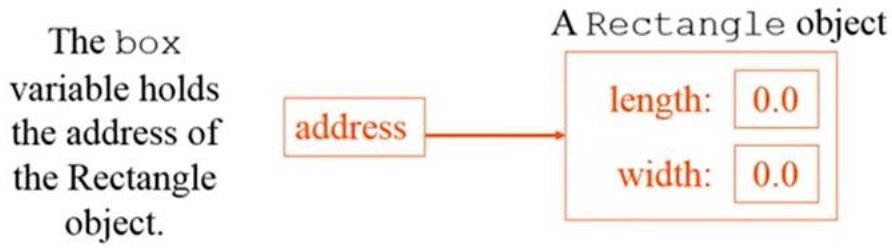
- An object hides its internal, private fields from code that is outside the class that the object is an instance of.
- Only the class's methods may directly access and change the object's internal data.
- Code outside the class must use the class's public methods to operate on an object's private fields.
- Data hiding is important because classes are typically used as components in large software systems, involving a team of programmers.




```
public class Rectangle {
        private double length;
         private double width;
  public void setLength (double l)
{
length=l;
}
public void setWidth (double w)
{
width=w;
}
}
```

Creating a Rectangle object

```
Rectangle box = new Rectangle ();
```



Calling setLength method

```
box.setLength(10.0);
```

This is the state of the box object after the `setLength` method executes.

Accessors and Mutators

```
public class Rectangle
{
    private double width;
    private double length;

    public void setWidth(double w)
    {       width = w;
    }
    public void setLength(double len)
    {       length = len;
    }
    public double getWidth()
    {       return width;
    }
    public double getLength()
    {       return length;
    }
    public double getArea()
    {       return length * width;
    }
}
```

Uninitialized Local Reference Variables

- Reference variables can be declared without being initialized.

```
Rectangle box;
```

- This statement does not create a `Rectangle` object, so it is an uninitialized local reference variable.
- A local reference variable must reference an object before it can be used, otherwise a compiler error will occur.

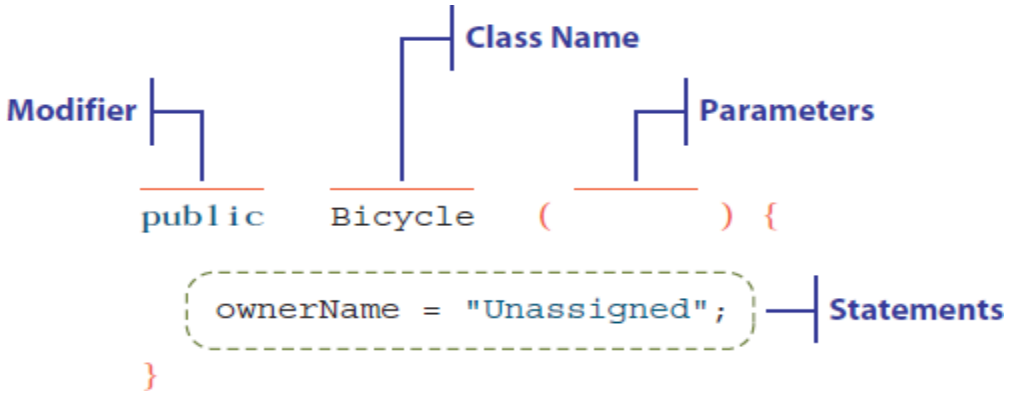
```
box = new Rectangle();
```

where <class name> is the name of the class to which this constructor belongs. The following diagram shows the constructor of the Bicycle class:

Constructors

- Classes can have special methods called *constructors*.
- A constructor is a method that is automatically called when an object is created.
- Constructors are used to perform operations at the time an object is created.
- Constructors typically initialize instance fields and perform other object

where <class name> is the name of the class to which this constructor belongs. The following diagram shows the constructor of the Bicycle class:



Notice that a constructor does not have a **return type** and,consequently, will never include a **return** statement.

The modifier of a constructor does not have to be public, but non-public constructors are rarely used.

The purpose of the constructor is to initialize the (data field) data memberand perform any other initialization tasks.

The Default Constructor

- When an object is created, its constructor is always called.
- If you do not write a constructor, Java provides one when the class is compiled. The constructor that Java provides is known as the *default constructor*.
  - It sets all of the object's numeric fields to 0.
  - It sets all of the object's `boolean` fields to `false`.
  - It sets all of the object's reference variables to the special value *null*.

Ex :Define class have two constructors and a method to display name ,id and salary of employ

```
class Emp {

private String name;
private String id;
private int salary;
        Emp(){                      // default constructor. No argument list
        name = "Ahmed";
        id = "1234";
        salary = 100;
}

public Emp(String n, String i, int s) {              // non-default constructor
        name = n;
        id = i;
        salary = s;
}


void display()  {
        System.out.println("\nEmploye Info ");
        System.out.println("Name "+ name);
        System.out.println("ID "+ id);
        System.out.println("Salary "+ salary);
        }
}

class ExEmp  {
        public static void main(String [] args){
            Emp e1 = new Emp();
            e1.display();
           Emp e4 = new Emp("Ali","98745", 400);
           E4.display();
        }    }
```