

# Programming II

## Object Oriented Programming (OOP) Object and Class

Dr/ Abeer Amer

## Objects and Classes:

- Classes: where objects come from
  - A class is code that describes a particular type of object. It specifies the data that an object can hold (the object's fields), and the actions that an object can perform (the object's methods).
  - You can think of a class as a code “blueprint” that can be used to create a particular type of object.
- When a program is running, it can use the class to create, in memory, as many objects of a specific type as needed.
- Each object that is created from a class is called an **instance** of the class.

# Objects and Classes:

## Example:

This expression creates a Scanner object in memory.

```
Scanner input = new Scanner(System.in);
```

The object's memory address  
is assigned to the Input  
variable.



# Objects and Classes:

## Example:

This expression creates a Random object in memory.

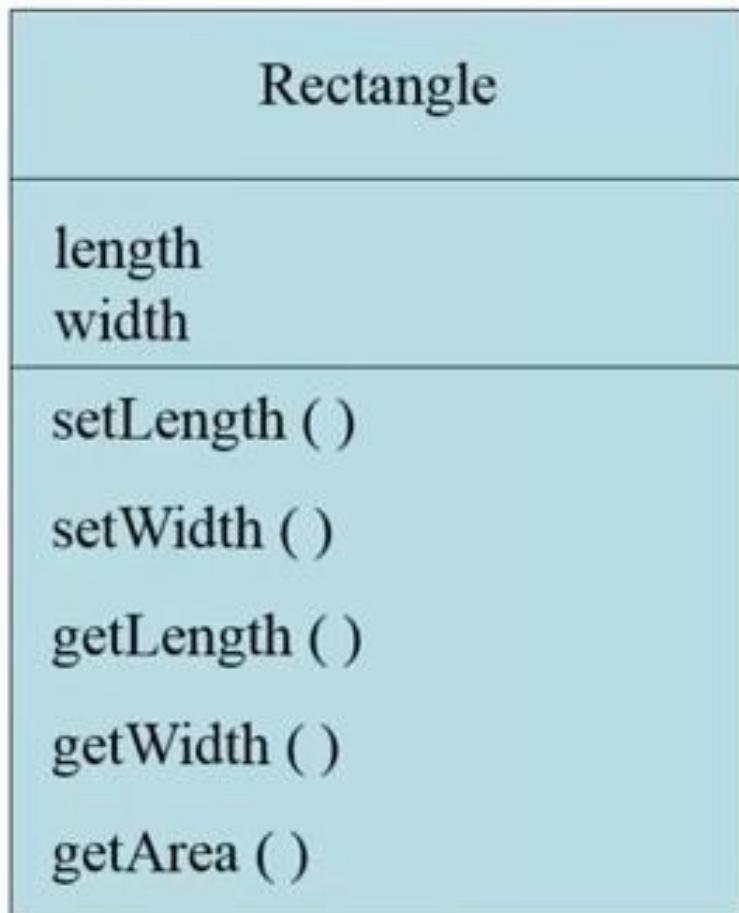
```
Random rand = new Random();
```

The object's memory address is assigned to the rand variable.



## Writing a class, step by step:

- A Rectangle class will have the following fields:



- UML Class Diagram

## Writing the code:

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

Rectangle
length width
setLength() setWidth() getLength() getWidth() getArea()

## **Access Modifiers:**

- 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 (field or method inside the class), the member can be accessed by the code inside the class or outside.

### **Private:**

- When the private access modifier is applied to a class member, the member cannot be accessed by the code outside the class. The member can be accessed only by methods are members of the same class.

## Data Hiding:

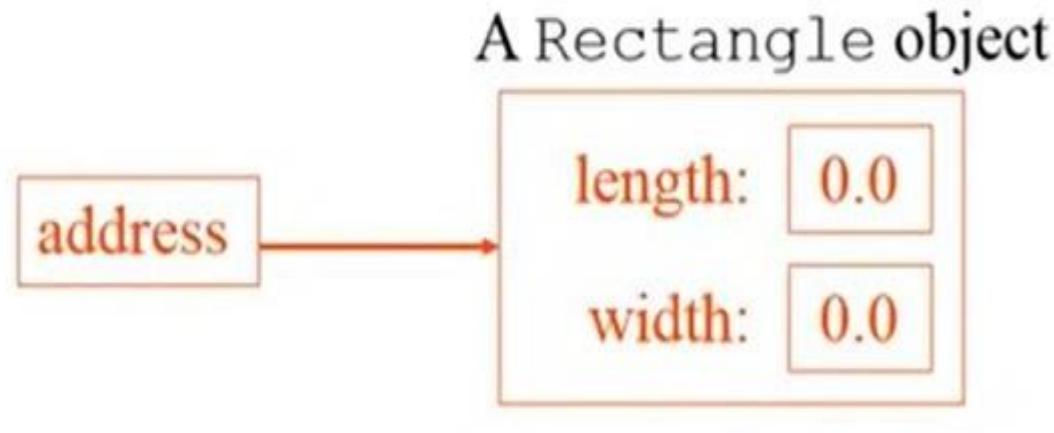
- An object hides its internal private fields from code that is outside the class that the object is an instance of.
- Only the class's methods may directly access and change the object's internal data.
- Code outside the class must use the class's public methods to operate on an object's private fields.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

## Example 1:

### Creating a Rectangle object:

**Rectangle r1 = new Rectangle();**

The r1 variable holds the address of the Rectangle object.



## Example 1:

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

```
public class Main  
{  
    public static void main(String[] args) {  
        Rectangle r1 = new Rectangle ();  
        r1.width = 10.5;  
        r1.length = 60;  
    }  
}
```

```
Main.java:14: error: length has private access in Rectangle  
        r1.length = 60;  
                           ^  
1 error
```

## Example 1:

```
public class Main
{
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle ();
        r1.width = -7;

    }
}
```

## Example 1:

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

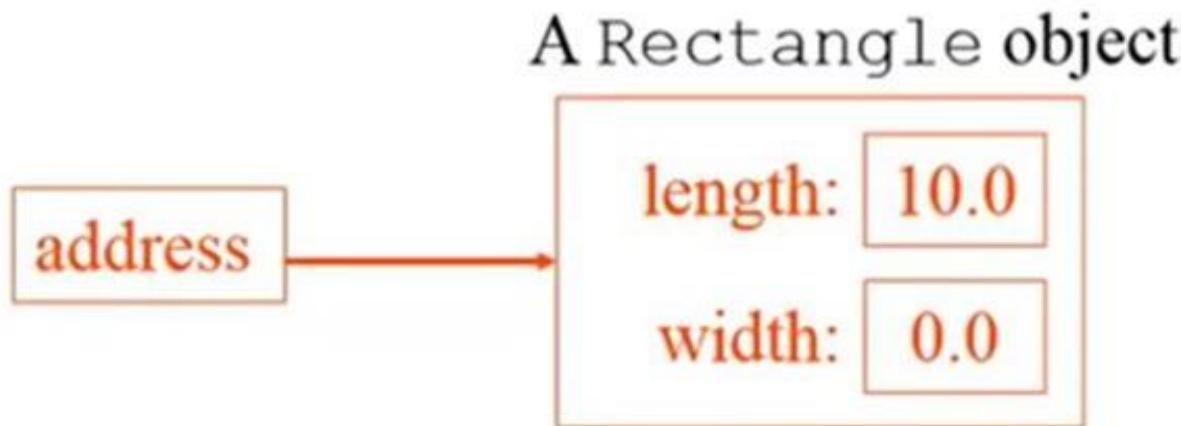
```
public class Main  
{  
    public static void main(String[] args) {  
        Rectangle r1 = new Rectangle ();  
        r1.setlength(10);  
        r1.setWidth(12.5);  
    }  
}
```

## Calling the setlength Method:

**r1.setlength(10);**

- This is the state of the r1 object after the **setlength** method executes.

The r1 variable holds the address of the Rectangle object.



## Example 1:

```
public class Rectangle{  
    private double length;  
    private double width;  
  
    public void setlength (double l)  
    {  
        length = l;  
    }  
    public void setwidth (double w)  
    {  
        width = w;  
    }  
    public double getlength()  
    {  
        return length;  
    }  
    public double getwidth()  
    {  
        return width;  
    }  
    public double getarea()  
    {  
        return length*width;  
    }  
}
```

Rectangle

length

width

setLength ()

setWidth ()

getLength ()

getWidth ()

getArea ()

## Example 1:

```
public class Main
{
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle ();
        r1.setlength(10);
        r1.setwidth(12.5);
        Rectangle r2 = new Rectangle ();
        r2.setlength(18);
        r2.setwidth(20);
        System.out.println (r1.getLength());
        System.out.println (r2.getArea());
    }
}
```

10.0  
360.0

# Setters (Mutators) and Getters (Accessors):

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

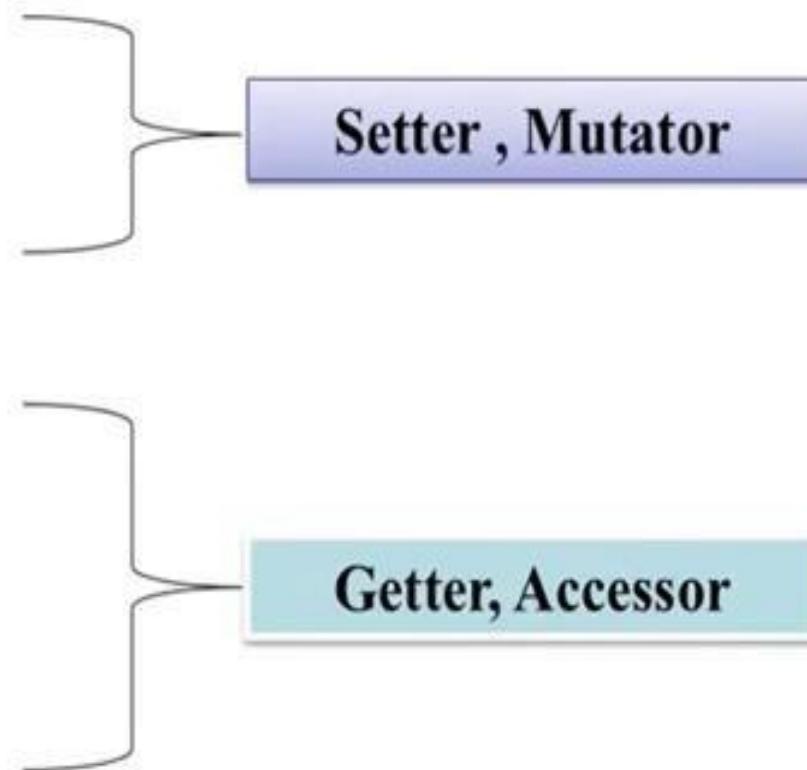
    public void setWidth(double w)
    {
        width = w;
    }

    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();



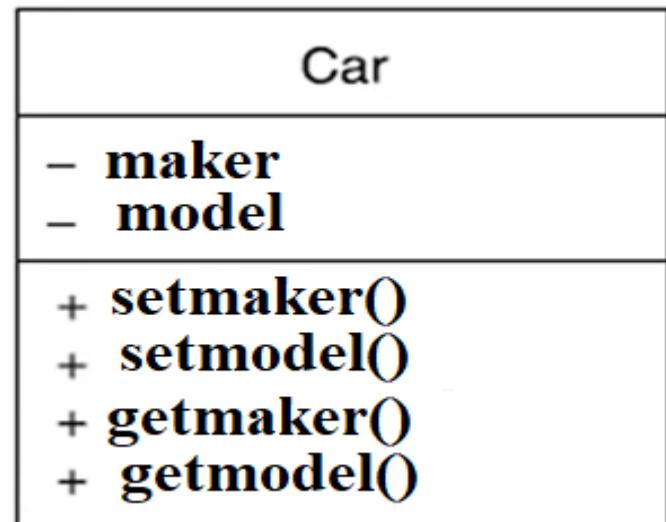
Rectangle box;

box = new Rectangle();

=      Rectangle box = new Rectangle();

## Example 2:

```
public class Car{  
    private String maker;  
    private int model;  
    public void setmaker (String m)  
    { maker = m;  
    }  
    public void setmodel (int year)  
    { model = year;  
    }  
    public String getmaker()  
    { return maker;  
    }  
    public int getmodel()  
    { return model;  
    }  
}
```



## Example 2:

```
public class Main
{
    public static void main(String[] args) {
        Car c1;
        c1 = new Car();
        Car c2 = new Car();
        c1.setmaker("Honda");
        c1.setmodel(2016);
        c2.setmaker("Toyota");
        c2.setmodel (2019);
        System.out.println(c1.getmaker());
        System.out.println(c2.getmodel());
    }
}
```

Honda  
2019

## Example 2(Data Hiding)

```
public class Car{  
    private String maker;  
    private int model;  
    public void setmaker (String m)  
    {   if (m == "Toyota" || m=="Honda" || m=="Merceds")  
        maker = m;  
    else  
        System.out.println("invalid maker");  
    }  
    public void setmodel (int year)  
    {   if (year > 2012)  
        model = year;  
    else  
        System.out.println("invalid model");  
    }  
    public String getmaker()  
    { return maker;  
    }  
    public int getmodel()  
    { return model;  
    }  
}
```

## Example 2(Data Hiding)

```
public class Main
{
    public static void main(String[] args) {
        Car c1;
        c1 = new Car();
        Car c2 = new Car();
        c1.setmaker("Honda");
        c1.setmodel(2016);
        c2.setmaker("Toyota");
        c2.setmodel (2008);
        System.out.println(c1.getmaker());
    }
}
```

invalid model  
Honda