



# **COMPUTER SCIENCE DEPARTMENT FACULTY OF ENGINEERING AND TECHNOLOGY**

**ADVANCED PROGRAMMING COMP231**

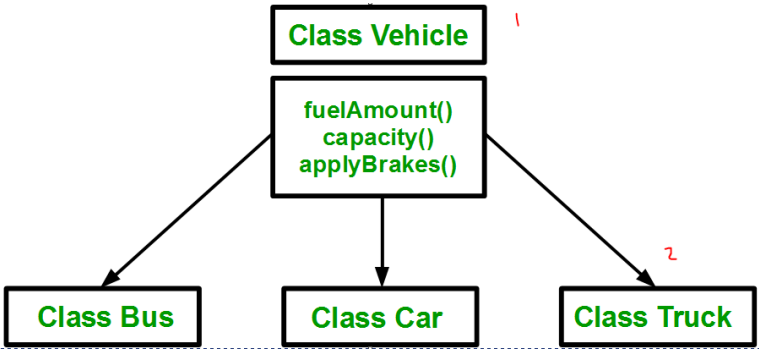
**Lecturer :Farid Mohammad**

# Inheritance



# Inheritance

Object-oriented programming allows you to define new classes from existing classes. This is called *inheritance*.




The new class that is created is known as **subclass** (child or derived class) and the existing class from where the child class is derived is known as **superclass** (parent or base class).

The **extends** keyword is used to perform inheritance in Java.

extends

## How we do it in Java

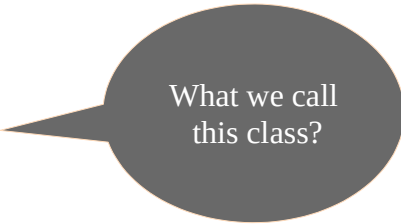
```
public class Vehicle {  
  
    private double fuel;  
    private boolean brake;  
  
    public Vehicle() {  
    }  
  
    public double fuelamount() {  
        return fuel;  
    }  
  
    public void applyBrakes() {  
        brake=true;  
    }  
}
```



What we call  
this class?

## First subclass

```
public class Car extends Vehicle {  
  
    public Car() {  
    }  
  
}
```



What we call  
this class?

## others subs

```
public class Bus extends Vehicle {  
  
    public Bus() {  
    }  
  
}
```

```
public class Truck extends Vehicle {  
  
    private double load;  
  
    public Truck() {  
        super();  
        System.out.println("Truck created");  
    }  
  
    public void carryLargeLoad(double load)  
    {  
        this.load=load;  
    }  
  
}
```

```
public class MainClass {  
  
    public static void main(String[] args) {  
        Car c=new Car();  
  
        c.applyBrakes();  
  
    }  
  
}
```



## is-a relationship

### is-a relationship

In Java, inheritance is an **is-a** relationship. That is, we use **inheritance only if there exists an is-a relationship** between two classes. For example,

- **Car** is a **Vehicle**
- **Orange** is a **Fruit**
- **Surgeon** is a **Doctor**
- **Dog** is an **Animal**

Here, **Car** can inherit from **Vehicle**, **Orange** can inherit from **Fruit**, and so on.

## private fields

Private data fields in a superclass **are not accessible** outside the class.

Therefore, they cannot be used directly in a subclass.

They can, however, be accessed/mutated through public getters/setters if defined in the superclass.



## Not all is-a

**Not** all is-a relationships should be **modeled** using **inheritance**.

For example, a square is a rectangle,  
but you **should not** extend a **Square** class from a **Rectangle** class,

because the **width** and **height** properties are **not** appropriate for a square.

## One class only

Some programming languages allow you to derive a subclass from several classes.

This capability is known as *multiple inheritance*.

**Java, however, does not allow multiple inheritance.**

A Java class may inherit directly from only one superclass.

This restriction is known as *single inheritance*.

Nevertheless, multiple inheritance can be achieved through **interfaces**, which will be introduced in Section 13.4.

## Using the **super** Keyword

The keyword **super** refers to the superclass and can be used to invoke the superclass's **methods** and **constructors**.

A subclass inherits accessible data fields and methods from its superclass.

Does it inherit constructors? Can the superclass's constructors be invoked from a subclass?

- To call a superclass constructor.
- To call a superclass method.

## Calling Superclass Constructors

A constructor is used to construct an instance of a class.

Unlike properties and methods, the **constructors of a superclass are not inherited** by a subclass. They can only be **invoked**

**from** the constructors of the subclasses using the keyword **super**.

The syntax to call a superclass's constructor is: **super()**, or **super(parameters)**;

```
public class SimpleGeometricObject {  
    private String color = "white";  
    private boolean filled;  
    private java.util.Date dateCreated;
```

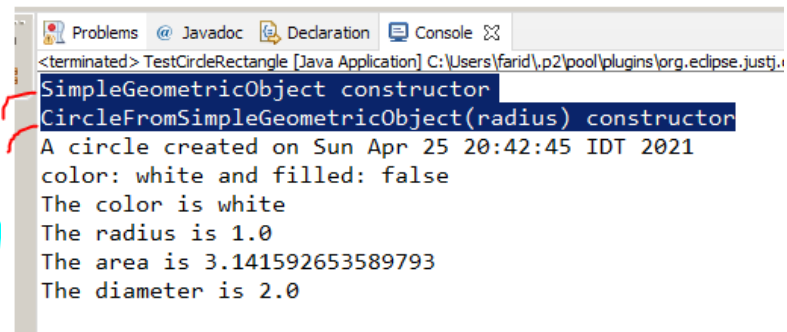
```
    /** Construct a default geometric object */
```

```
    public SimpleGeometricObject() {  
        dateCreated = new java.util.Date();  
    }
```

```
    public class CircleFromSimpleGeometricObject extends
```

```
SimpleGeometricObject {  
    private double radius;  
    public CircleFromSimpleGeometricObject(double radius, String color,  
    boolean filled) {  
        super(color, filled);  
        this.radius = radius;  
    }  
}
```

superclass constructors  
can be invoked using  
**super()**

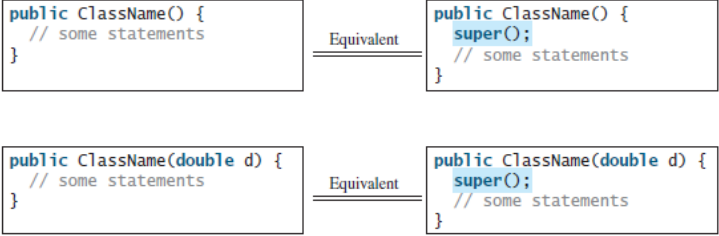


```
Problems Javadoc Declaration Console  
<terminated> TestCirdeRectangle [Java Application] C:\Users\farid\p2\pool\plugins\org.eclipse.justj...  
SimpleGeometricObject constructor  
CircleFromSimpleGeometricObject(radius) constructor  
A circle created on Sun Apr 25 20:42:45 IDT 2021  
color: white and filled: false  
The color is white  
The radius is 1.0  
The area is 3.141592653589793  
The diameter is 2.0
```

# Constructor Chaining

A constructor may invoke an overloaded constructor or its superclass constructor.

If **neither is invoked explicitly, the compiler automatically puts `super()` as the first statement in the constructor.** For example:



```

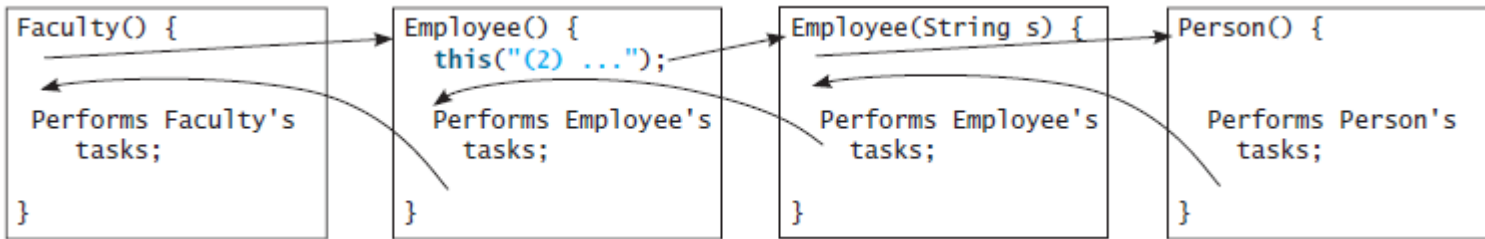
public class Faculty extends Employee {
    public static void main(String[] args) {
        new Faculty();
    }

    public Faculty() {
        System.out.println("(4) Performs Faculty's tasks");
    }
}

class Employee extends Person {
    public Employee() {
        this("(2) Invoke Employee's overloaded constructor");
        System.out.println("(3) Performs Employee's tasks ");
    }

    public Employee(String s) {
        System.out.println(s);
    }
}

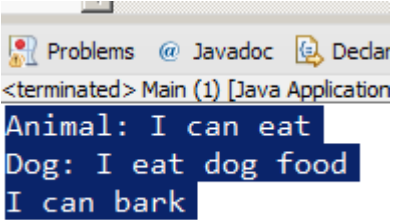
class Person {
    public Person() {
        System.out.println("(1) Performs Person's tasks");
    }
}
    
```



# super methods

You can call super methods from subclass

```
class Animal {  
  
    // method in the superclass  
    public void eat() {  
        System.out.println("I can eat");  
    }  
}  
  
// Dog inherits Animal  
class Dog extends Animal {  
  
    // overriding the eat() method  
    @Override  
    public void eat() {  
  
        // call method of superclass  
        super.eat();  
        System.out.println("I eat dog food");  
    }  
  
    // new method in subclass  
    public void bark() {  
        System.out.println("I can bark");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
  
        // create an object of the subclass  
        Dog labrador = new Dog();  
  
        // call the eat() method  
        labrador.eat();  
        labrador.bark();  
    }  
}
```



The screenshot shows a terminal window titled "<terminated> Main (1) [Java Application]". The output of the program is displayed as follows:

```
Animal: I can eat  
Dog: I eat dog food  
I can bark
```