# Lecture 04

## Object-Oriented Programming

{OOP} Inheritance

Prepared by **Hok Tin**

# Content

- Array List
- Modifier Types
- Inheritance

# ArrayList

- **ArrayList** is a ***resizable array***, which can be found in the **`java.util`** package

```java
class Student {
    int id;
    String name;
    int age;

    void setValues(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }
    void display() {
        System.out.println(id + " " + name + " " + age);
    }
}
```

**Usage:**

```java
ArrayList<Student> stuArr = new ArrayList<Student>();
Student stu1 = new Student();
stu1.setValues(11, "Tola", 18);
stuArr.add(stu1); // add 1st student

Student stu2 = new Student();
stu2.setValues(22, "Makara", 17);
stuArr.add(stu2); // add 2nd student

Student stu3 = new Student();
stu3.setValues(33, "Kompheak", 23);
stuArr.add(stu3); // add 3rd student

// Check array size
System.out.println("Size: " + stuArr.size());

// Access by index (0)
Student firstStu = stuArr.get(0);
firstStu.display();

stuArr.remove(0); // remove by index (0)
stuArr.get(0).display();
System.out.println("Size: " + stuArr.size());

stuArr.clear();
System.out.println("Size: " + stuArr.size());
```

**Output:**

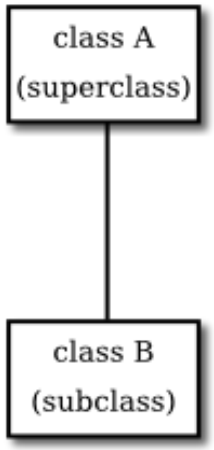
```
Size: 3
11 Tola 18
22 Makara 17
Size: 2
Size: 0
```

# Inheritance

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

✍ Terms used in Inheritance

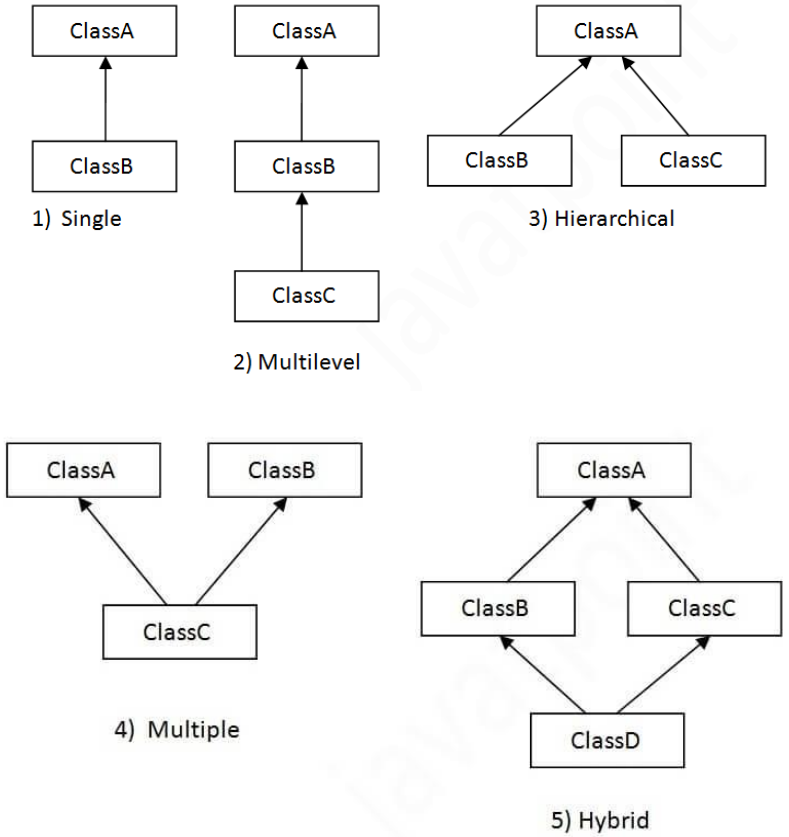
- **Class:** A class is a group of objects which have common properties. It is a *template* or *blueprint* from which objects are created
- **Sub Class/Child Class:** Subclass is a class which *inherits the other class*. It is also called a derived class, extended class, or child class
- **Super Class/Parent Class:** Superclass is the class from *where a subclass inherits the features*. It is also called a base class or a parent class
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you *to reuse the fields and methods of the existing class when you create a new class*. You can use the same fields and methods already defined in the previous class



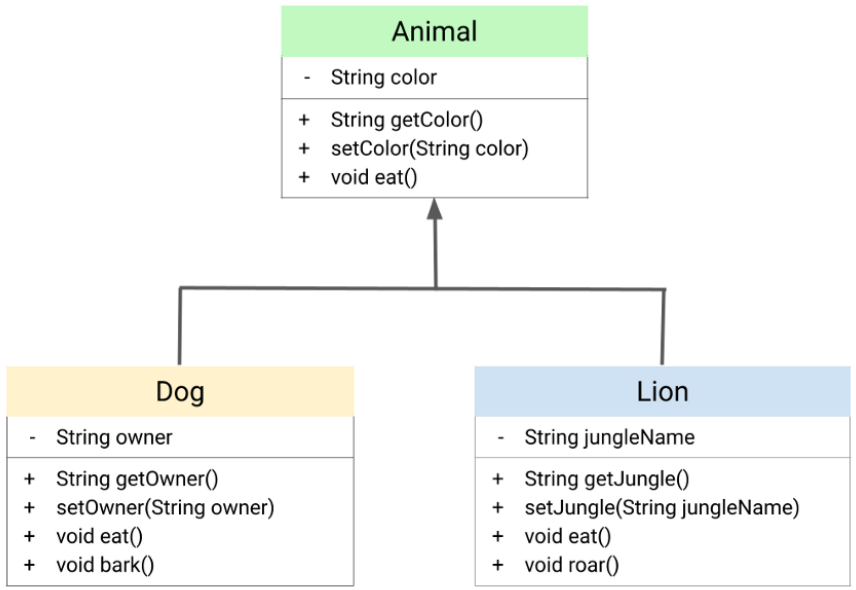

```
class Subclass-name extends Superclass-name {
    //methods and fields
}
```

# Inheritance

## Types of inheritance in java




## Java Inheritance Example



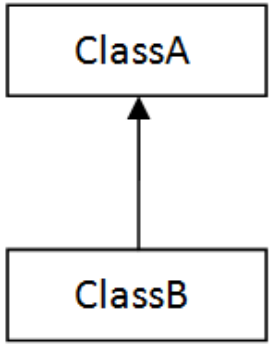

✍ Note: Multiple inheritance is not supported in Java through class

# Single Inheritance

When a class inherits another class, it is known as a single inheritance.

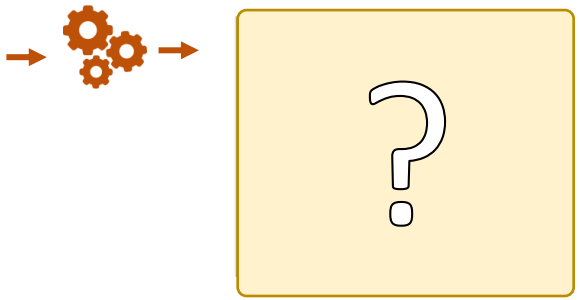
MainTest.java

```java
class Animal {
    void eat() {
        System.out.println("eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("barking...");
    }
}

class MainTest {
    public static void main(String args[]) {
        Dog d=new Dog();
        d.bark();
        d.eat();
    }
}
```




1) Single

Output

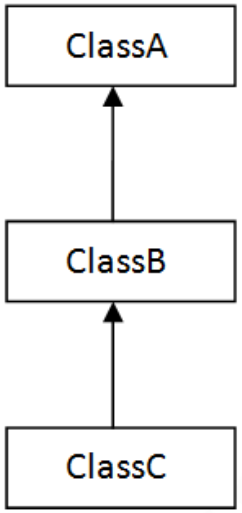


# Multilevel Inheritance

When there is a chain of inheritance, it is known as multilevel inheritance

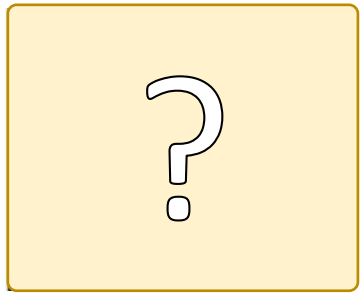
MainTest.java

```java
class Animal {
    void eat() {
        System.out.println("eating...");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("barking...");
    }
}
class BabyDog extends Dog {
    void weep() {
        System.out.println("weeping...");
    }
}
class MainTest {
    public static void main(String args[]) {
        BabyDog d=new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}
```




2) Multilevel

Output

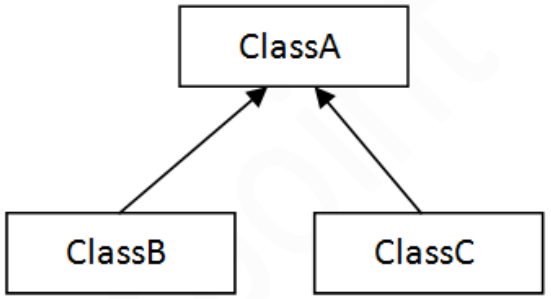


# Hierarchical Inheritance

When two or more classes inherits a single class, it is known as hierarchical inheritance

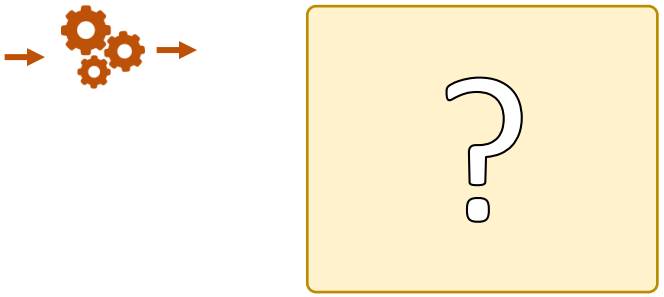
MainInheritance3.java

```java
class Animal {
    void eat() {
        System.out.println("eating...");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("barking...");
    }
}
class Cat extends Animal {
    void meow() {
        System.out.println("meowing...");
    }
}
class TestInheritance3 {
    public static void main(String args[]) {
        Cat c=new Cat();
        c.meow();
        c.eat();
        //c.bark();   //C.T.Error
    }
}
```




3) Hierarchical

Output



# Multiple inheritance

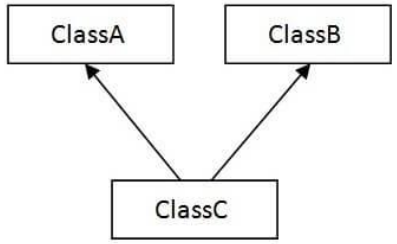
Q) Why multiple inheritance is not supported in java?

✔ To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

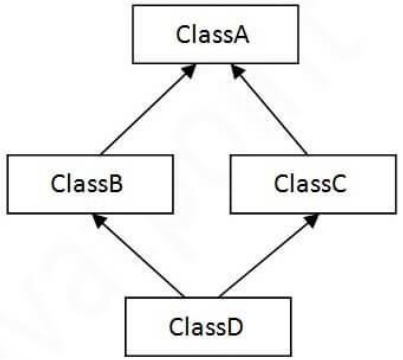
Consider the below scenario:

C.java

```
class A {
    void msg() {
        System.out.println("Hello");
    }
}
class B {
    void msg() {
        System.out.println("Welcome");
    }
}
class C extends A,B { //suppose if it were

    public static void main(String args[]) {
        C obj=new C();
        obj.msg();//Now which msg() method would be invoked? ✕
    }
}
```




4) Multiple




5) Hybrid

# Modifier Types

## Access Control Modifiers

- **Private**: the access level of a private modifier is only within the class. It cannot be accessed from outside the class
- **Default**: the access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default
- **Public**: the access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class/subclasses. If you do not make the child class, it cannot be accessed from outside the package

| Access Modifier | within class | within package | outside package by subclass only | outside package | |
|---|---|---|---|---|---|
| **Private** | Y | N | N | N | **(Class property level)** |
| **Default** | Y | Y | N | N | **(Class & property level, Default keyword not used)** |
| **Protected** | Y | Y | Y | N | **(Class properties level)** |
| **Public** | Y | Y | Y | Y | **(Class & property level)** |

# Private access modifier

The **private** access modifier is accessible only within the class.

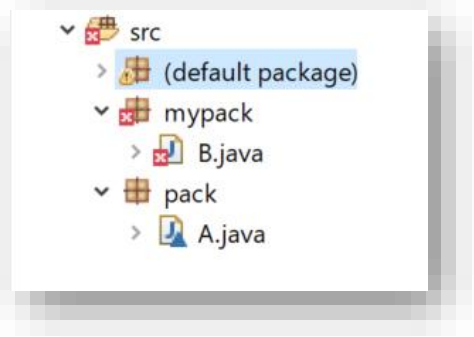
```java
class A {
    private int data = 40;
    private void msg() {
        System.out.println("Hello java");
    }
}

public class Simple {
    public static void main(String args[]) {
        A obj = new A();
        System.out.println(obj.data); ✕
        obj.msg(); ✕
    }
}
```

Terminal

```
Simple.java:11: error: data has private access in A
        System.out.println(obj.data);
                                 ^
Simple.java:12: error: msg() has private access in A
        obj.msg();
          ^
2 errors
```

# Default access modifier

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package.




A.java

```java
package pack;
class A {
    void msg() {
        System.out.println("Hello");
    }
}
```

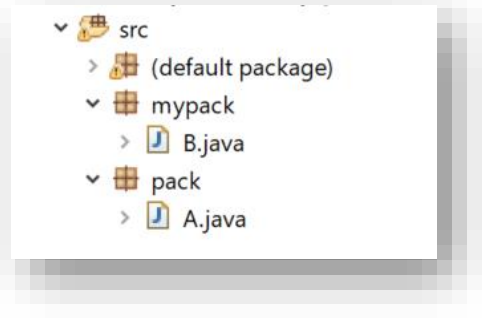
B.java

```java
package mypack;
import pack.*;
class B {
    public static void main(String args[]) {
        A obj = new A();//Compile Time Error ✕
        obj.msg();//Compile Time Error ✕
    }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The type A is not visible
        The type A is not visible
        The type A is not visible

        at mypack.B.main(B.java:6)
```

# Public access modifier

The **public** access modifier is is accessible everywhere. It has the widest scope among all other modifiers.

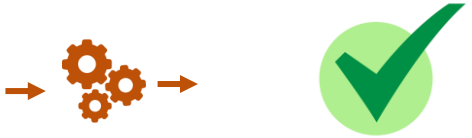



A.java

```java
package pack;
public class A {
    public void msg() {
        System.out.println("Hello");
    }
}
```

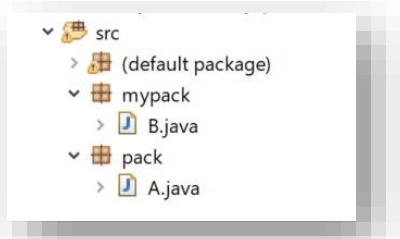
B.java

```java
package mypack;
import pack.*;

class B {
    public static void main(String args[]) {
        A obj = new A();
        obj.msg();
    }
}
```



# Protected access modifier

The **protected** access modifier is accessible within package and outside the package but through inheritance only.

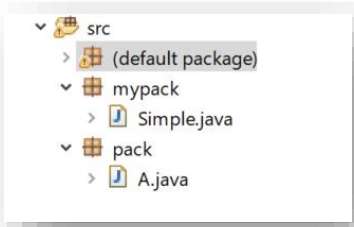



A.java

```java
package pack;
public class A {
    protected void msg() {
        System.out.println("Hello");
    }
}
```

B.java

```java
package mypack;
import pack.*;

class B extends A {
    public static void main(String args[]) {
        B obj = new B();
        obj.msg();
    }
}
```




A.java

```java
package pack;
public class A {
    protected void msg() {
        System.out.println("Hello");
    }
}
```

Simple.java

```java
package mypack;
import pack.*;

public class Simple extends A {
    void msg() {
        System.out.println("Hello java");   //C.T.Error
    }
    public static void main(String args[]) {
        Simple obj=new Simple();
        obj.msg();
    }
}
```

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

# Good luck 👌

😉 **References:**


https://www.javatpoint.com/access-modifiers

https://www.javatpoint.com/inheritance-in-java

https://www.javatpoint.com/method-overriding-in-java

https://www.tutorialspoint.com/java/java_inheritance.htm

https://dev.java/oop/