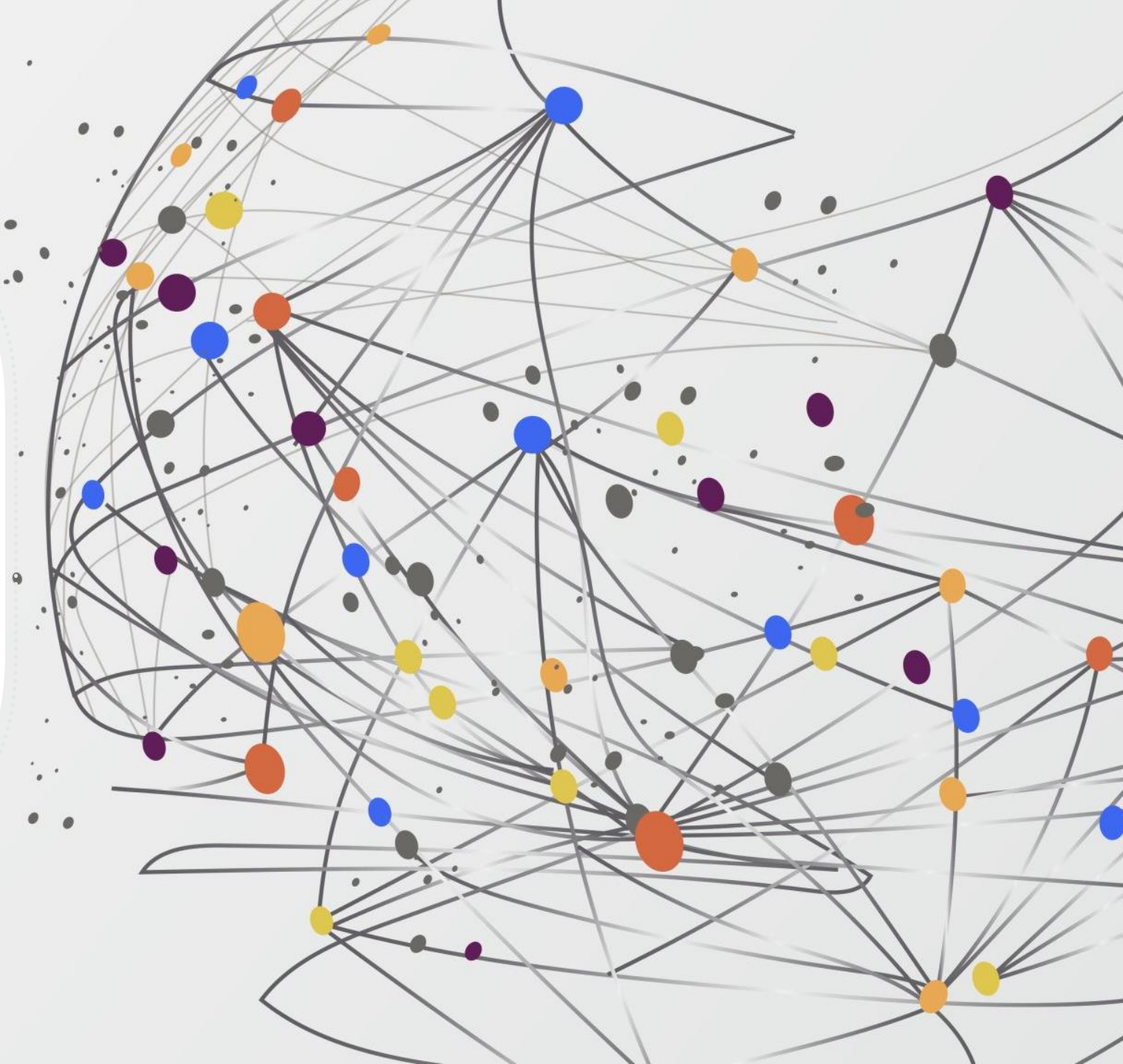
Inheritance in Java

DR. KRISHNENDU GUHA
ASSISTANT PROFESSOR/ LECTURER
SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY COLLEGE CORK

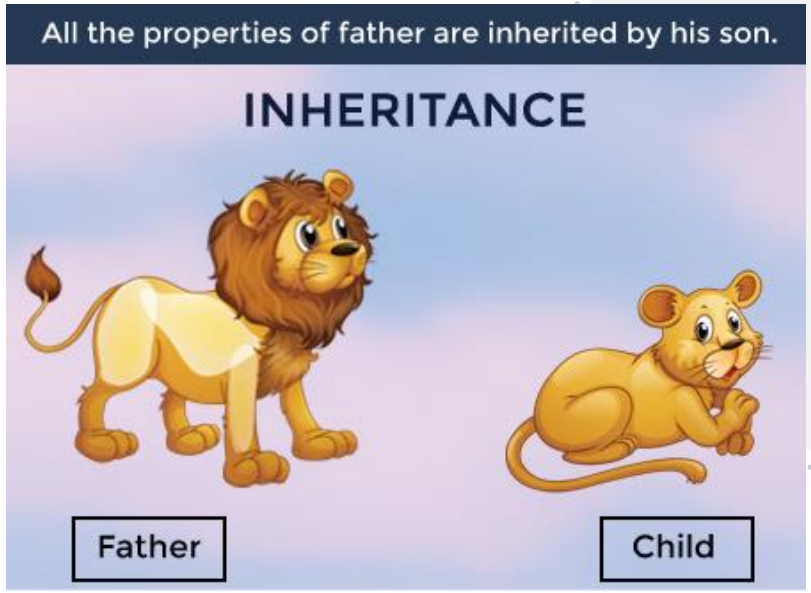


Inheritance

- **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).

- The idea behind inheritance in Java is that you can **create new classes that are built upon existing classes**.
- When you inherit from an existing class, you **can reuse methods and fields of the parent class.**

- Moreover, you **can add new methods and fields in your current class also**.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.



- 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.

Why use inheritance in java?

For Method Overriding (so runtime polymorphism can be achieved).

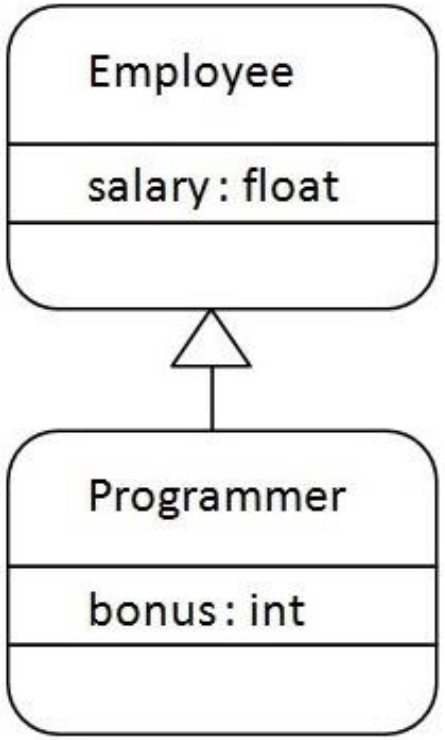
For Code Reusability.

The syntax of Java Inheritance

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

The **extends keyword** indicates that you are **making a new class that derives from an existing class**. The meaning of "extends" is to increase the functionality.

•**subclass** (child) - the class that inherits from another class
•**superclass** (parent) - the class being inherited from



Programmer is the subclass and Employee is the superclass.
The relationship between the two classes is **Programmer IS-A Employee**.
It means that Programmer is a type of Employee.

```
class Employee{
        float salary=40000;
}

class Programmer extends Employee{
        int bonus=10000;
        public static void main(String args[]){
        Programmer p=new Programmer();
        System.out.println("Programmer salary is:"+p.salary);
        System.out.println("Bonus of Programmer is:"+p.bonus);
        }
}
```

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

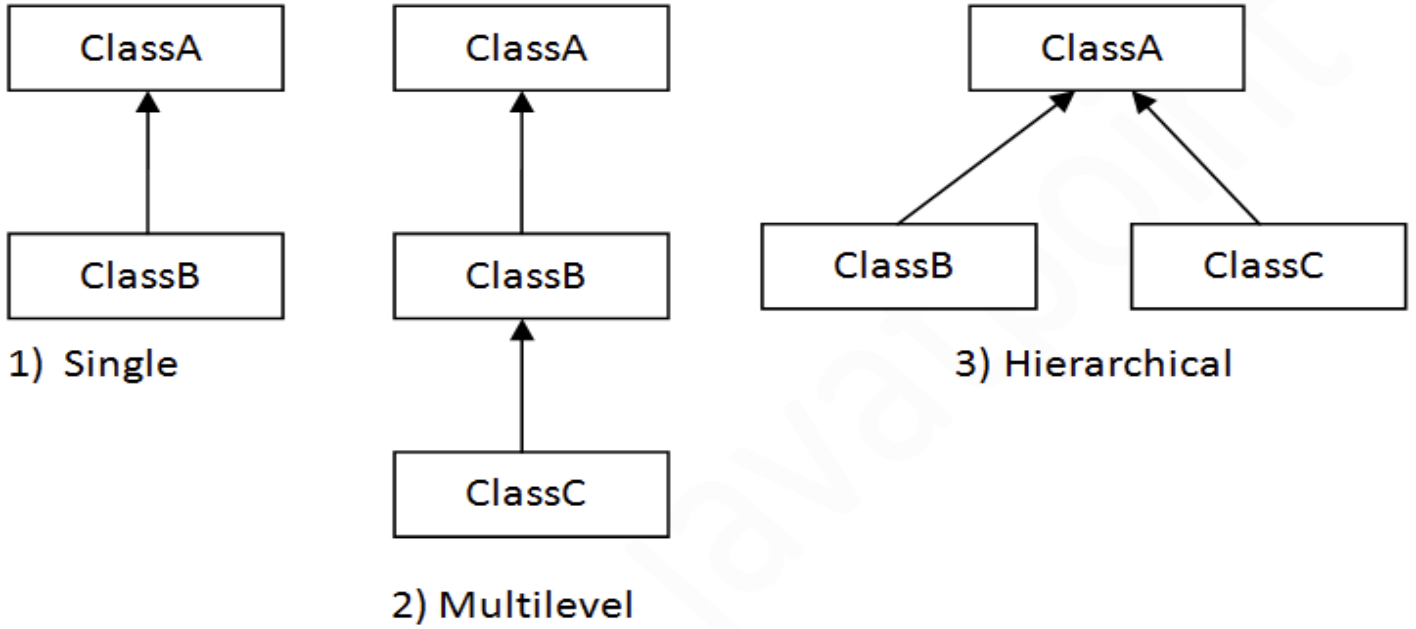
Types of Inheritance in Java

- On the basis of class, there can be three types of inheritance in java:

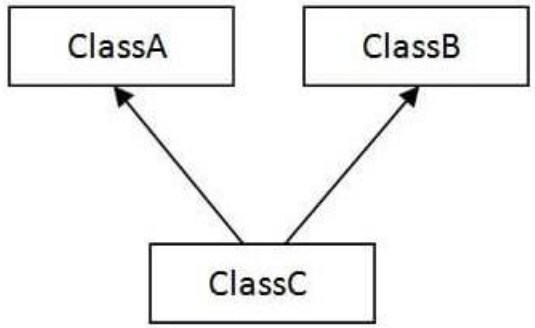
  single,

  multilevel

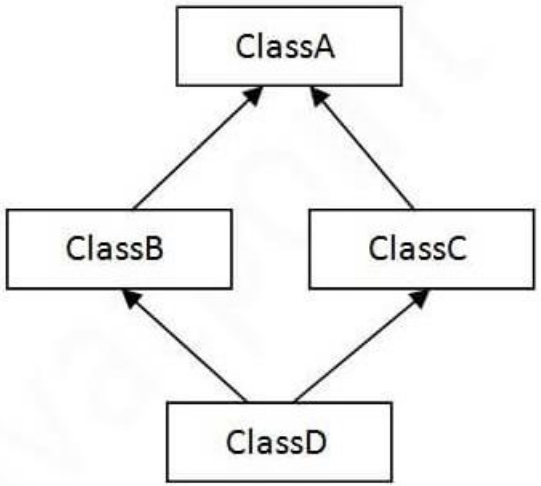
  hierarchical.



- In java programming, multiple and hybrid inheritance is supported through interface only.
- Multiple inheritance is not supported in Java through class.



4) Multiple



5) Hybrid

Single Inheritance

- When a class inherits another class, it is known as a *single inheritance*.
- In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class University{
        void study(){System.out.println(“This is UCC");}
}
class Csit extends University{
        void learn(){
                System.out.println(“This is CSIT department");}
}
class TestInheritance{
        public static void main(String args[]){
        Csit c=new Csit();
        c.learn();
        c.study();
}}
```

Output
This is CSIT department
This is UCC

Multilevel Inheritance

- When there is a **chain of inheritance**, it is known as *multilevel inheritance*.
- As you can see in the example given below, Java class inherits the Csit class which again inherits the University class, so there is a multilevel inheritance.

```
class University{
      void study(){
            System.out.println(“This is UCC");}
}
class Csit extends University{
      void learn(){
            System.out.println(“This is CSIT department");}
}
class Java extends Csit{
      void module(){
            System.out.println(“This is Java Class");}
}
class TestInheritance2{
      public static void main(String args[]){
      Java d=new Java();
      d.study();
      d.learn();
      d.module();
      }
}
```

Output

This is UCC
This is CSIT department
This is Java Class

Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.
- In the example given below, Ucc and Mtu classes inherits the University class, so there is hierarchical inheritance.

```
class University{
void study(){System.out.println(“Study in University");}
}
class Ucc extends University{
void learn(){System.out.println(“Study in UCC");}
}
class Mtu extends University{
void discuss(){System.out.println(“Study in MTU");}
}
class TestInheritance3{
public static void main(String args[]){
Ucc c=new Ucc();
c.learn();
c.study();
//c.discuss();//C.T.Error
}}
```

Output
Study in UCC
Study in University

Why multiple inheritance is not supported in java?

- To **reduce the complexity and simplify the language**, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes. Then C class inherits A and B classes.
- If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.
- Since **compile-time errors are better than runtime errors**, Java renders compile-time error if you inherit 2 classes. So, whether you have same method or different, there will be compile time error.

```
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?
}
}
```

```
Compile Time Error
```

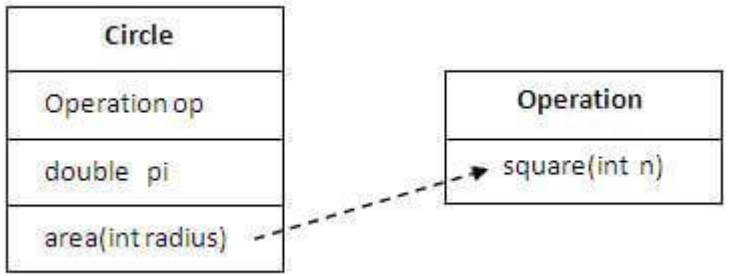
Aggregation in Java

- If a **class have an entity reference, it is known as Aggregation**.
- **Aggregation represents HAS-A relationship**.

- Consider a situation, **Employee object** contains many information such as *id*, *name*, *email Id etc.*
- It contains ***one more object named address***, which contains its own information such as city, state, country, zipcode etc. as given below.
- In such case, **Employee has an entity reference address**, so relationship is Employee HAS-A address.

- Why use Aggregation?
- For Code Reusability.

```
class Employee{
int id;
String name;
Address address;//Address is a class
...
}
```




```
class Operation{
   int square(int n){
   return n*n;
   }
}

class Circle{
   Operation op;//aggregation
   double pi=3.14;

   double area(int radius){
   op=new Operation();
   int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
   return pi*rsquare;
   }

   public static void main(String args[]){
   Circle c=new Circle();
   double result=c.area(5);
   System.out.println(result);
   }
}
```

```
Output:78.5
```

- When use Aggregation?

- Code reuse is also best achieved by aggregation **when there is no is-a relationship**.
- Inheritance should be used only if the *relationship is-a is maintained throughout the lifetime of the objects involved*; otherwise, ***aggregation is the best choice***.

Understanding meaningful example of Aggregation

In this example, Employee has an object of Address, address object contains its own information such as city, state, country etc.

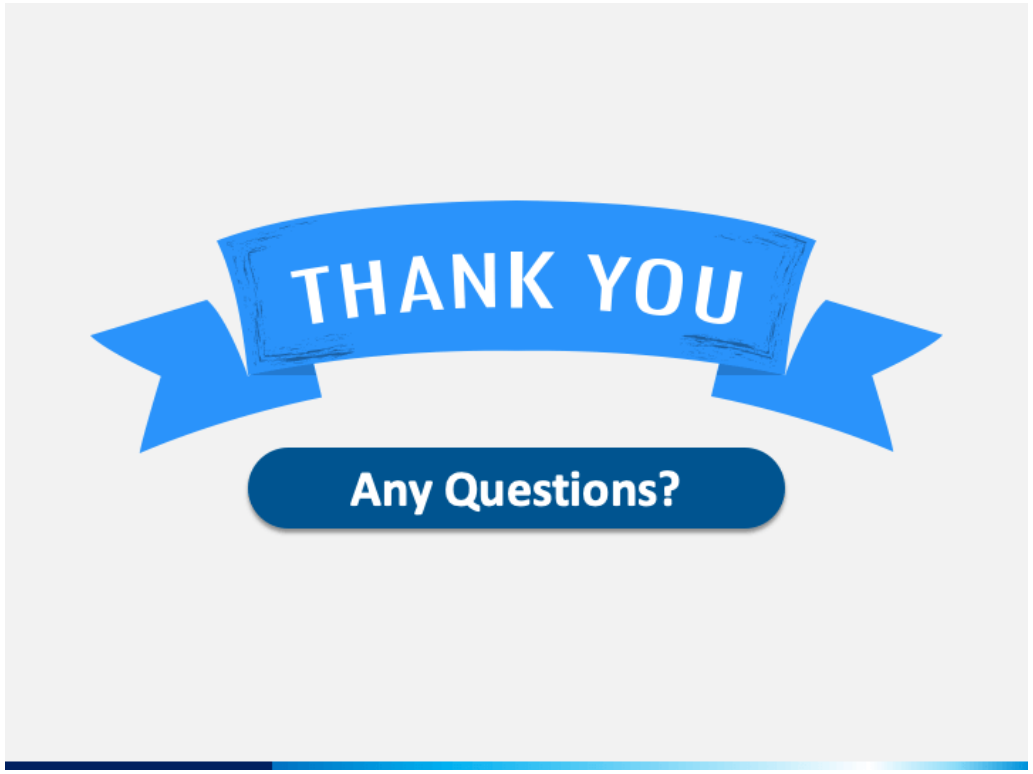
In such case relationship is Employee HAS-A address.

Address.java

```
public class Address {
String city,state,country;

public Address(String city, String state, String country) {
    this.city = city;
    this.state = state;
    this.country = country;
}

}
```

Emp.java

```
public class Emp {
int id;
String name;
Address address;

public Emp(int id, String name,Address address) {
   this.id = id;
   this.name = name;
   this.address=address;
}

void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+" "+address.country);
}

public static void main(String[] args) {
Address address1=new Address("Cork","Cork","Ireland");
Address address2=new Address("Miami","Florida","USA");

Emp e=new Emp(111,“Giselle",address1);
Emp e2=new Emp(112,“Mark",address2);

e.display();
e2.display();

}
}
```


THANK YOU
Any Questions?