

Abstract and interface

IQRA UNIVERSITY

- **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 OOP (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.
- **Why use inheritance in java**
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

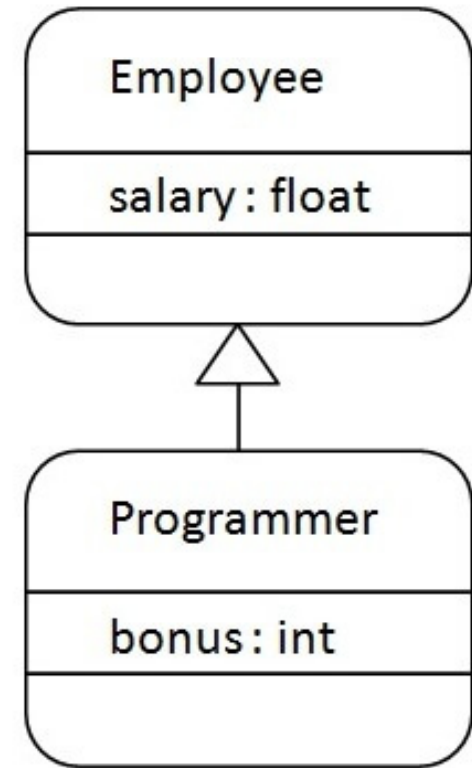
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent 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  
}
```

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.

Java Inheritance Example

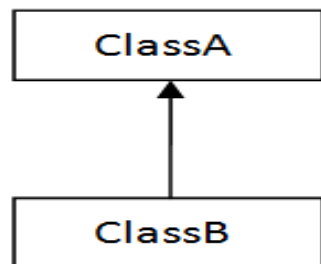
```
class Employee{
float salary=40000;
}
class Programmer extends Employee{
int bonus=10000;
public static void main(String args[]){
Programmer p=new Programmer();
System.out.println("Programmer salary is:"+p.salary);
System.out.println("Bonus of Programmer is:"+p.bonus); }
}
```



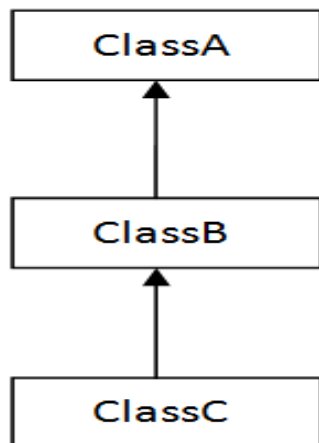
Programmer salary is:40000.0
Bonus of programmer is:10000

Types of inheritance in java

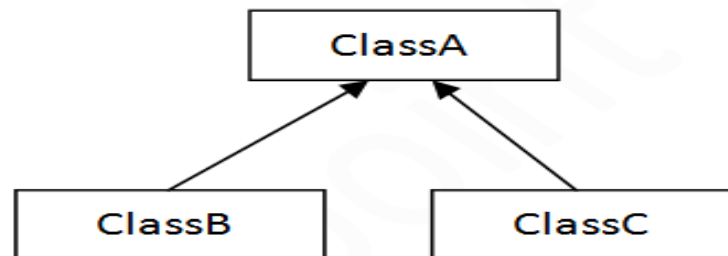
- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



1) Single

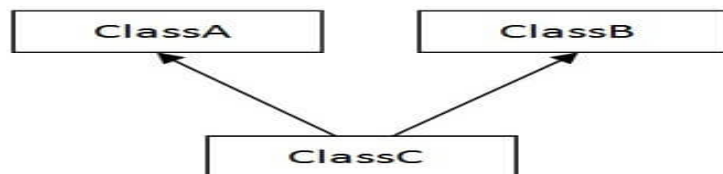


2) Multilevel

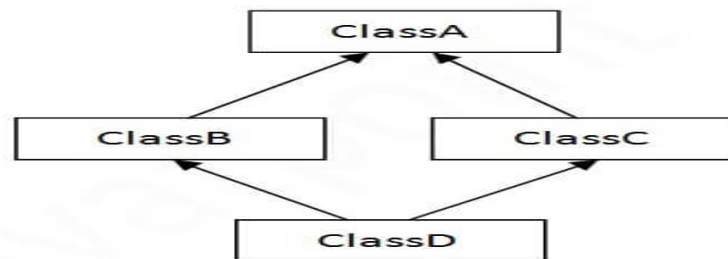


3) Hierarchical

- In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



4) Multiple



5) Hybrid

Single Inheritance Example

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 Animal{  
void eat(){System.out.println("eating...");}  
}  
class Dog extends Animal{  
void bark(){System.out.println("barking...");}  
}  
class TestInheritance2{  
public static void main(String args[]){  
Dog d=new Dog();  
d.bark();  
d.eat();  
}}
```

Multilevel Inheritance Example

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

```
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 TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Weeping...
barking...
eating...

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
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
}}
```

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. The 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?
}
}
```

Introduction

- As per dictionary, **abstraction** is the quality of dealing with ideas rather than events. For example, when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user. Therefore, to send an e-mail you just need to type the content, mention the address of the receiver, and click send.

- Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.
- In Java, abstraction is achieved using Abstract classes and interfaces.

- A class which contains the **abstract** keyword in its declaration is known as abstract class.
- Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class **must** be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

```

/* File name : Employee.java */
public abstract class Employee {
    private String name;
    private String address;
    private int number;

    public Employee(String name, String address, int number) {
        System.out.println("Constructing an Employee");
        this.name = name;
        this.address = address;
        this.number = number;
    }

    public double computePay() {
        System.out.println("Inside Employee computePay");
        return 0.0;
    }

    public void mailCheck() {
        System.out.println("Mailing a check to " + this.name + " " + this.address);
    }

    public String toString() {
        return name + " " + address + " " + number;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String newAddress) {
        address = newAddress;
    }

    public int getNumber() {
        return number;
    }
}

```

- You can observe that except abstract methods the Employee class is same as normal class in Java. The class is now abstract, but it still has three fields, seven methods, and one constructor.
- Now you can try to instantiate the Employee class in the following way –

```
/* File name : AbstractDemo.java */
public class AbstractDemo {

    public static void main(String [] args) {
        /* Following is not allowed and would raise error */
        Employee e = new Employee("George W.", "Houston, TX", 43);
        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}
```

When you compile the above class, it gives you the following error –

```
Employee.java:46: Employee is abstract; cannot be instantiated
    Employee e = new Employee("George W.", "Houston, TX", 43);
                  ^
1 error
```


- We can inherit the properties of Employee class just like concrete class in the following way –

Example

```
/* File name : Salary.java */
public class Salary extends Employee {
    private double salary;    // Annual salary

    public Salary(String name, String address, int number, double salary) {
        super(name, address, number);
        setSalary(salary);
    }

    public void mailCheck() {
        System.out.println("Within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName() + " with salary " + salary);
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
            salary = newSalary;
        }
    }

    public double computePay() {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }
}
```

Here, you cannot instantiate the Employee class, but you can instantiate the Salary Class, and using this instance you can access all the three fields and seven methods of Employee class as shown below.

```
/* File name : AbstractDemo.java */
public class AbstractDemo {

    public static void main(String [] args) {
        Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
        Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
        System.out.println("Call mailCheck using Salary reference --");
        s.mailCheck();
        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}
```

This produces the following result – Output

```
Constructing an Employee
Constructing an Employee
Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```

- If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.
- **abstract** keyword is used to declare the method as abstract.
- You have to place the **abstract** keyword before the method name in the method declaration.
- An abstract method contains a method signature, but no method body.
- Instead of curly braces, an abstract method will have a semi colon (;) at the end.

```
public abstract class Employee {  
    private String name;  
    private String address;  
    private int number;  
  
    public abstract double computePay();  
    // Remainder of class definition  
}
```

Declaring a method as abstract has two consequences –

- The class containing it must be declared as abstract.
- Any class inheriting the current class must either override the abstract method or declare itself as abstract.
- **Note** – Eventually, a descendant class has to implement the abstract method; otherwise, you would have a hierarchy of abstract classes that cannot be instantiated.

Suppose Salary class inherits the Employee class, then it should implement the **computePay()** method as shown below –

```
/* File name : Salary.java */
public class Salary extends Employee {
    private double salary;    // Annual salary

    public double computePay() {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }
    // Remainder of class definition
}
```

