Khulna Khan Bahadur Ahsanullah University

Object-oriented programming

CSE 1203

1. What is Inheritance?

- The process of obtaining the data members and methods from one class to another class is known as **inheritance**.
- Inheritance in Java is a mechanism in which one class acquires all the properties and behaviors of a parent class.
- It is one of the fundamental features of object-oriented programming.
- In the inheritance the class **which is give** data members and methods is known as **base or super or parent class**.
- The class **which is taking** the data members and methods is known as **sub or derived or child class**.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

2. Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.
- To implement 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.
- **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.
- **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.
- **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.

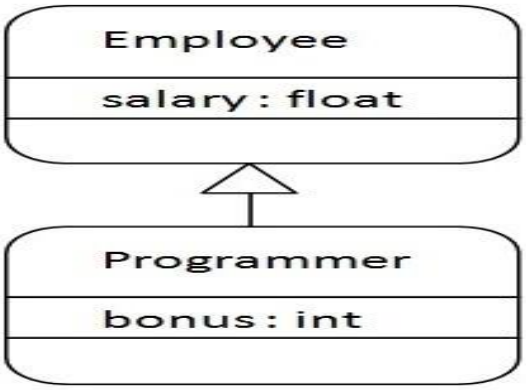
The syntax of Java Inheritance

```
1. class Subclass-name extends Superclass-name
2. {
3.    //methods and fields
4. }
```

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.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Example:



As displayed in the above figure, **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.

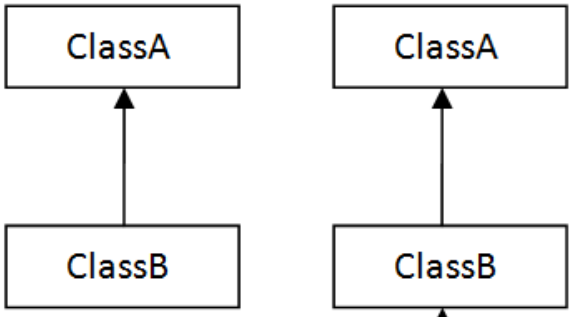
```
1. class Employee {
2.  float salary=40000;
3. }
4. class Programmer extends Employee {
5.  int bonus=10000;
6.  public static void main (String args[]){
7.    Programmer p=new Programmer ();
8.    System.out.println("Programmer salary is:"+p.salary);
9.    System.out.println("Bonus of Programmer is:"+p.bonus);
10.} }
```

Output:

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

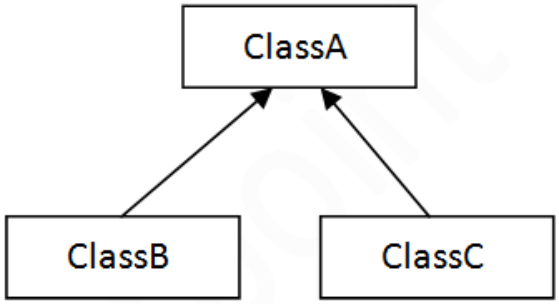
In the above example, Programmer object can access the field of own class as well as of Employee class i.e., code reusability.

Types of inheritance:

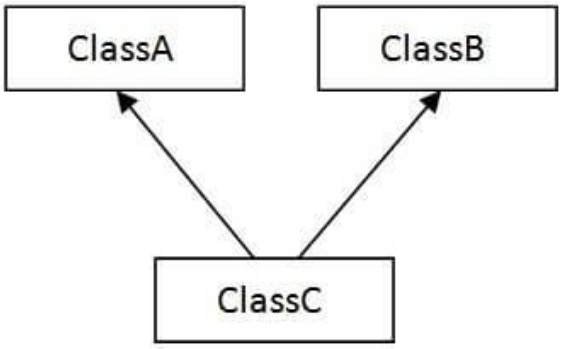


1) Single

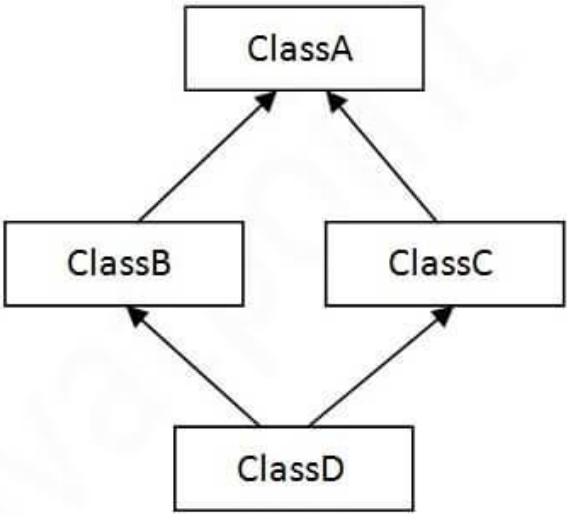
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

1. 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.

File: TestInheritance.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class TestInheritance{
8. public static void main(String args[]){
9. Dog d=new Dog();
10.d.bark();
11.d.eat();
12.}
13.}
```

Output:

```
barking...
eating...
```

2.Multilevel Inheritance

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.

File: TestInheritance2.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class BabyDog extends Dog{
8. void weep(){System.out.println("weeping...");}
9. }
10.class TestInheritance2{
11.public static void main(String args[]){
12.BabyDog d=new BabyDog();
13.d.weep();
14.d.bark();
15.d.eat();
16.}}
```

Output:

```
weeping...
barking...
eating...
```

3. Hierarchical Inheritance

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.

File: TestInheritance3.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class Cat extends Animal{
8. void meow(){System.out.println("meowing...");}
9. }
10.class TestInheritance3{
11.public static void main(String args[]){
12.Cat c=new Cat();
13.c.meow();
14.c.eat();
15.//c.bark();//C.T.Error
16.}}
```

Output:

```
meowing...
eating...
```

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 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.

```
1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4. class B{
5. void msg(){System.out.println("Welcome");}
6. }
7. class C extends A,B{//suppose if it were
8.
9.  public static void main(String args[]){
10.  C obj=new C();
11.  obj.msg();//Now which msg() method would be invoked?
12.}
13.}
```

Compile Time Error