CLASSMATE
Date :
Page :

INHERITANCE IN JAVA

Inheritence in java is a mechanism where a new class (subclass / child class) acquires the properties and behaviours (methods) of an existing class (superclass / parent class). It promotes code reusability and establishes a relationship between classes.

@ Curious--. programmer

1. Types of Inheritence in Java :

a) Single Inheritence :

- A class inherits from one superclass.

Example :

```
class Animal {
    void sound () {
    System.out.println ("some sound");
  }
}

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

b) Multilevel Inheritence

- A class is derived from another derived class (i.e, a class inherits from a class that is already a subclass).

Example:

```
class Animal {
   void sound () {
     System.out.println ("some sound");
   }
 }

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

class Puppy extends Dog {
     void play () {
      System.out.println ("Playing");
  }
}
```

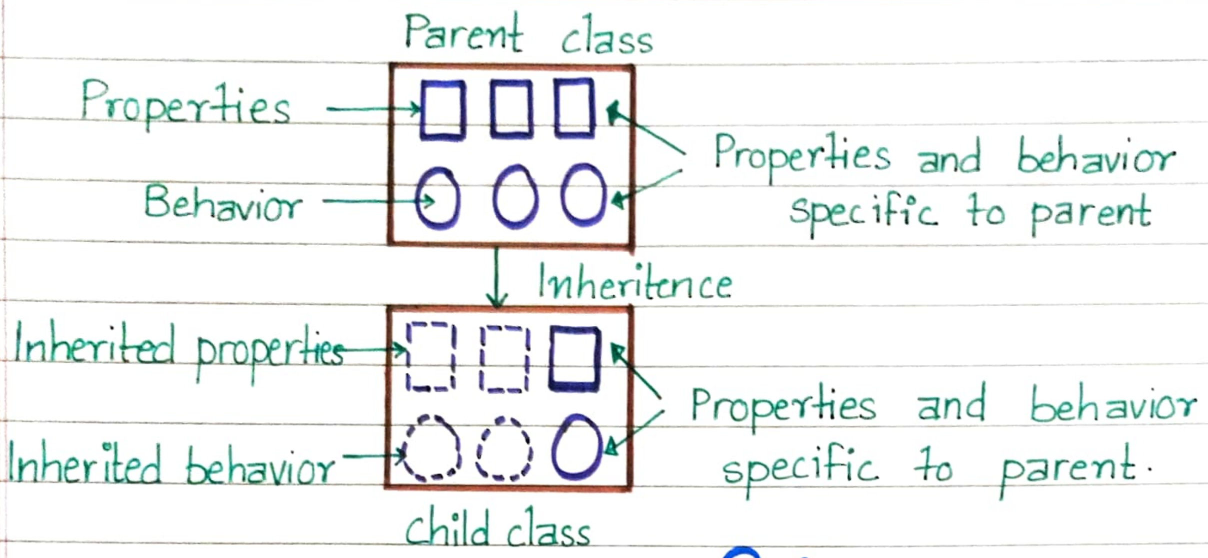
@ Curious_.programmer

c) Hierarchical Inheritence :

- Multiple classes inherit from a single superclass.
- one class can serve as the parent class or base class for several child classes.

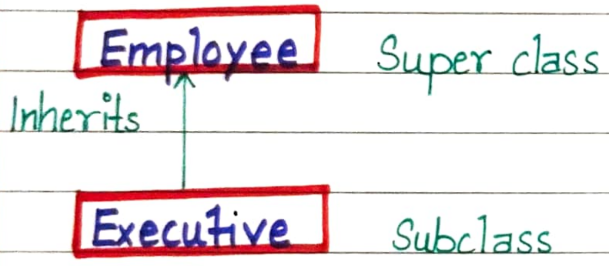
Example :

```
class Animal {
   void sound () {
    System.out.println ("some sound");
  }
}

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

class Cat extends Animal {
   void meow () {
    System.out.println ("Meowing");
  }
}
```

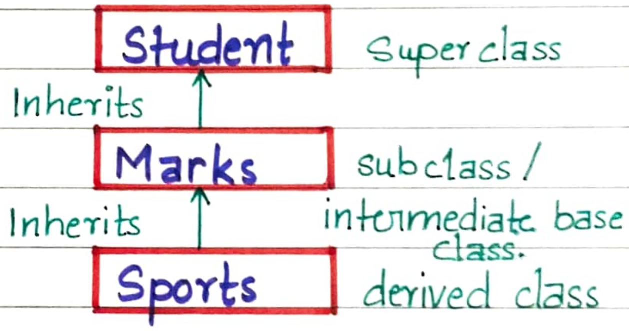


@ Curious_. programmer

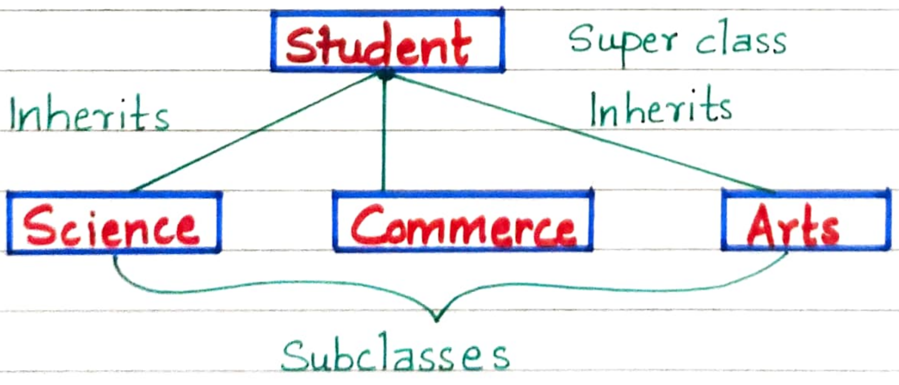
Inheritence



Single Inheritence



Multilevel inheritence



Hierarchical inheritence

2. Super Keyword :

• **Purpose :** It refers to the immediate parent class of the current object.

• **Use cases :**

1. Accessing superclass methods.
2. Accessing superclass constructors.

Example :

```
class Animal {
    void sound () {
    System.out.println ("Animal makes sound ");
   }
 }

class Dog extends Animal {
     void  sound () {
           super.sound () ;  //calling superclass method
    System.out. println ( "Dog barks ");
  }
}
```

@ Curious_.programmer

3. Method Overriding:

• **Definition:** Method overriding allows a subclass to provide its own implementation of a method that is already defined in its superclass.

• **Conditions:**

1. Same method signature (name, parameters, return type).

2. The method in the superclass must not be final, static, or private.

Example: @ Curious_. programmer

```
class Animal {
    void sound () {
      System.out.println ("Animal makes sound");
   }
}

class Dog extends Animal {
    @Override
   void sound () {
      System.out.println ("Dog barks");
   }
}
```

CLASSMATE
Date :
Page :

4. Upcasting and Downcasting:

• Upcasting :

Converting a subclass reference to a superclass reference. Implicit casting happens automatically.

Example:

```
Dog dog = new dog();
Animal animal = dog;      //Upcasting
```

• Downcasting :

Converting a superclass reference to a subclass reference.
It requires explicit casting and can lead to ClassCastException if done incorrectly.

Example:

```
Animal animal = new Dog();
Dog dog = (Dog) animal;   // Downcasting
```

@ Curious — programmer

5. final Keyword in Inheritance :

1. final class :

A class declared with final cannot be subclassed. It cannot be extended by any other class. This means we cannot create a subclass of a final class, ensuring its behavior is not altered through inheritance.

Example :

```
final class Animal {
      // cannot be inherited
      void sound () {
    System. out. println ("Animal makes sound");
   }
}

class Dog extends Animal {
               //compile-time error
// This will throw a compilation error because Animal
     is final and cannot be subclassed.

}
```

@ Curious_. programmer

2. final method:

A method declared as final cannot be overridden by subclasses.

This ensures that the behavior of the method stays the same, regardless of where it is inherited.

Example:

```
class Animal {
    final void sound () {
     System.out.println ("Animal makes sound");
  }
}

class Dog extends Animal {
  // cannot override sound() method
  // This will throw a compile-time error because
     the method sound () is declared final in the
     Animal class.
void sound () {
System.out.println ("Dog barks");
}
}
```

@Curious_.programmer

3. final variable:

A final variable is a constant. Once assigned, its value cannot be changed.

This is useful when you want to define values that should remain constant throughout the execution of the program.

Example:

```
class Dog {
      final int age = 5 ;  // age cannot be reassigned

void changeAge () {
    //age = 6 ; //  compile-time error: cannot assign a
                    value to final variable 'age'.
 }
}
```