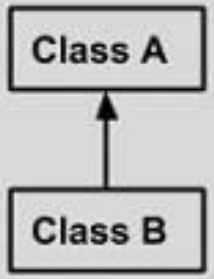
- ***Inheritance***

Inheritance

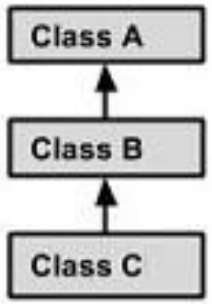
- Same inheritance concept of C++ in Java with some modifications
  - One class inherits the other using ***extends*** keyword
  - The classes involved in inheritance are known as ***superclass*** and ***subclass***
  - ***Multilevel*** inheritance but no ***multiple*** inheritance
  - There is a special way to call the superclass's ***constructor***
  - There is automatic ***dynamic method dispatch***
- Inheritance provides code reusability (code of any class can be used by extending that class)

Single Inheritance



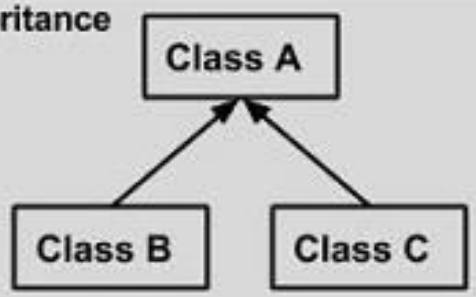

```
public class A {
   .....
}
public class B extends A {
   .....
}
```

Multi Level Inheritance



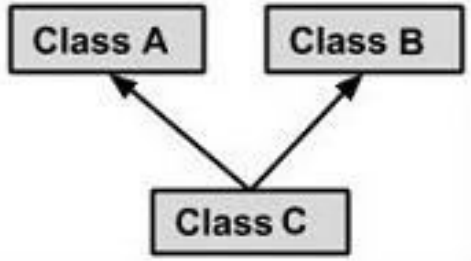

```
public class A { ..................}
public class B extends A {.....................}
public class C extends  B {..................... }
```

Hierarchical Inheritance




```
public class A { ..................}
public class B extends A {.....................}
public class C extends A {..................... }
```

Multiple Inheritance




```
public class A { ..................}
public class B {.....................}
public class C extends  A,B {
   .....................
} // Java does not support mutiple Inheritance
```

Simple Inheritance

```java
class A {
    int i, j;

    void showij() {
        System.out.println(i+" "+j);
    }
}

class B extends A{
    int k;

    void showk() {
        System.out.println(k);
    }

    void sum() {
        System.out.println(i+j+k);
    }
}
```

```java
public class SimpleInheritance {
    public static void main(String[] args) {
        A superOb = new A();
        superOb.i = 10;
        superOb.j = 20;
        superOb.showij();
        B subOb = new B();
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        subOb.showij();
        subOb.showk();
        subOb.sum();
    }
}
```


P : SimpleInheritance.java

Inheritance and Member Access

```java
class M {
    int i;
    private int j;

    void set(int x, int y) {
        i = x;
        j = y;
    }
}

class N extends M {
    int total;

    void sum() {
        total = i + j;
        // Error, j is not accessible here
    }
}
```

```java
public class SimpleInheritance2 {
    public static void main(String[] args) {
        N obj = new N();
        obj.set(10, 20);
        obj.sum();
        System.out.println(obj.total);
    }
}
```

- A class member that has been declared as private will remain private to its class
- It is not accessible by any code outside its class, including subclasses

P : SimpleInheritance2.java

Practical Example

```
3   class Box {
4       double width, height, depth;
5
6       Box(Box ob) {
7           width = ob.width; height = ob.height; depth = ob.depth;
8       }
9
10      Box(double w, double h, double d) {
11          width = w; height = h; depth = d;
12      }
13
14      Box() { width = height = depth = 1; }
17
18      Box(double len) { width = height = depth = len; }
21
22      double volume() { return width * height * depth; }
25  }
26
27  class BoxWeight extends Box {
28      double weight;
29
30      BoxWeight(double w, double h, double d, double m) {
31          width = w; height = h; depth = d; weight = m;
32      }
33  }
```


P : RealInheritance.java

Superclass variable reference to Subclass object

```
34
35  public class RealInheritance {
36      public static void main(String[] args) {
37          BoxWeight weightBox = new BoxWeight( w: 3,  h: 5,  d: 7,  m: 8.37);
38          System.out.println(weightBox.weight);
39          Box plainBox = weightBox; // assign BoxWeight reference to Box reference
40          System.out.println(plainBox.volume()); // OK, volume() defined in Box
41          System.out.println(plainBox.weight); // Error, weight not defined in Box
42          Box box = new Box( w: 1,  h: 2,  d: 3); // OK
43          BoxWeight wbox = box; // Error, can't assign Box reference to BoxWeight
44      }
45  }
46
```


P : RealInheritance.java

Using super to call Superclass Constructors

There are three cases to use super() in Java.

- **Case 1: super can be used to refer to the immediate parent class instance variable.**
- **Case 2: super can be used to invoke the immediate parent class method.**
- **Case 3: super() can be used to invoke immediate parent class constructor**

Note:
super( ) must always be the first executable statement inside a subclass' constructor

Using super to call Superclass Constructors

Case 1: super can be used to refer to immediate parent class instance variable.

```
1  class Animal{
2      String color="white";
3  }
4
5  class Dog extends Animal{
6      String color="black";
7      void printColor(){
8          System.out.println(color);//prints color of Dog class
9          System.out.println(super.color);//prints color of Animal class
10     }
11 }
12
13 class TestSuper1{
14     public static void main(String args[]){
15     Dog d=new Dog();
16     d.printColor();
17
18     }
19 }
```

- We can use super keyword to access the data member or field of parent class.
- It is used if parent class and child class have same fields.

P : TestSuper1.java

Using super to call Superclass Constructors

Case 2: super can be used to invoke the immediate parent class method.

```
1  class Animal1{
2      void eat(){
3          System.out.println("eating...");
4          }
5  }
6
7  class Dog1 extends Animal1{
8          void eat(){
9              System.out.println("eating bread...");
10             }
11         void bark(){
12             System.out.println("barking...");
13             }
14         void work(){
15         super.eat();
16         bark();
17         }
18 }
19 class TestSuper2{
20     public static void main(String args[]){
21         Dog1 d=new Dog1();
22         d.work();
23
24     }
25 }
```

- The super keyword can also be used to invoke the parent class method.
- **It should be used if the subclass contains the same method as the parent class.**
- In other words, it is used if the **method is overridden**.

P : TestSuper2.java

Using super to call Superclass Constructors

Case 3: super() can be used to invoke immediate parent class constructor

```
1  class Animal{
2      Animal(){
3          System.out.println("animal is created");
4          }
5  }
6  class Dog extends Animal{
7      Dog(){
8          super();
9          System.out.println("dog is created");
10     }
11 }
12     class TestSuper3{
13     public static void main(String args[]){
14             Dog d=new Dog();
15     }
16 }
```

- The super keyword can also be used to invoke the parent class constructor
- It should be the first executable statement in subclass constructor

P : TestSuper3.java

Using super to access Superclass hidden members

```
3   class C {
4       int i;
5       void show() {
6       }
7   }
8
9   class D extends C {
10      int i; // this i hides the i in C
11
12      D(int a, int b) {
13          super.i = a; // i in C
14          i = b; // i in D
15      }
16
17      void show() {
18          System.out.println("i in superclass: " + super.i);
19          System.out.println("i in subclass: " + i);
20          super.show();
21      }
22  }
23
24  public class UseSuper {
25      public static void main(String[] args) {
26          D subOb = new D( a: 1,  b: 2);
27          subOb.show();
28      }
29  }
```

In general

Multilevel Inheritance

```java
class X {
    int a;
    X() {
        System.out.println("Inside X's constructor");
    }
}

class Y extends X {
    int b;
    Y() {
        System.out.println("Inside Y's constructor");
    }
}

class Z extends Y {
    int c;
    Z() {
        System.out.println("Inside Z's constructor");
    }
}

public class MultilevelInheritance {
    public static void main(String[] args) {
        Z z = new Z();
        z.a = 10;
        z.b = 20;
        z.c = 30;
    }
}
```

Inside X's constructor
Inside Y's constructor
Inside Z's constructor

P : MultilevelInheritance.java

Method Overriding

```java
class Base {
    int a;
    Base(int a) {
        this.a = a;
    }
    void show() {
        System.out.println(a);
    }
}

class Child extends Base {
    int b;

    Child(int a, int b) {
        super(a);
        this.b = b;
    }

    // the following method overrides Base class's show()
    @Override // this is an annotation (optional but recommended)
    void show() {
        System.out.println(a + ", " + b);
    }
}
```

```java
public class MethodOverride {
    public static void main(String[] args) {
        Child o = new Child( a: 10,  b: 20);
        o.show();
        Base b = o;
        b.show(); // will call show of Override
    }
}
```


P : MethodOverride.java

Question-1:

```
3  class X {
4    int a;
5
6    X(int i) { a = i; }
7  }
8
9  class Y {
10   int a;
11
12   Y(int i) { a = i; }
13 }
14
15 class TestClass {
16   public static void main(String[] args) {
17     X x = new X(10);
18     X x2;
19     Y y = new Y(5);
20
21     x2 = x;
22
23     x2 = y;     // Error, not of same type
24   }
25 }
```

Question-2

```
2  class X
3  {
4    int a;
5
6    X(int i) { a = i; }
7  }
8
9  class Y extends X
10 {
11   int b;
12
13   Y(int i, int j)
14   {
15     super(j);
16     b = i;
17   }
18 }
```

```
20 class SupSubRef2 {
21   public static void main(String[] args)
22   {
23     X x = new X(10);
24     X x2;
25     Y y = new Y(5, 6);
26
27     x2 = x; // OK, both of same type
28     System.out.println("x2.a: " + x2.a);
29
30     x2 = y;
31     System.out.println("x2.a: " + x2.a);
32
33     x2.a = 19;
34   }
35 }
```

```
x2.a: 10
x2.a: 6
```