Java

Inheritance

Inheritance

- Same inheritance concept of C++ in Java with some modifications

- In Java,

  - 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**

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();
    }
}
```

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   }
```

Superclass variable reference to Subclass object

```java
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
```

Using super

```
3   class BoxWeightNew extends Box {
4       double weight;
5
6       BoxWeightNew(BoxWeightNew ob) {
7           super(ob);
8           weight = ob.weight;
9       }
10
11      BoxWeightNew(double w, double h, double d, double m) {
12          super(w, h, d);
13          weight = m;
14      }
15
16      BoxWeightNew() {
17          super(); // must be the 1st statement in constructor
18          weight = 1;
19      }
20
21      BoxWeightNew(double len, double m) {
22          super(len);
23          weight = m;
24      }
25
26      void print() {
27          System.out.println("Box(" + width + ", " + height +
28                             ", " + depth + ", " + weight + ")");
29      }
30  }
```

Using super

```
31
32    public class SuperTest {
33        public static void main(String[] args) {
34            BoxWeightNew box1 = new BoxWeightNew(10, 20, 15, 34.3);
35            BoxWeightNew box2 = new BoxWeightNew(2, 3, 4, 0.076);
36            BoxWeightNew box3 = new BoxWeightNew();
37            BoxWeightNew cube = new BoxWeightNew(3, 2);
38            BoxWeightNew clone = new BoxWeightNew(box1);
39            box1.print();
40            box2.print();
41            box3.print();
42            cube.print();
43            clone.print();
44        }
45
46    }
47
```

Using super

```java
class C {
    int i;
    void show() {
    }
}

class D extends C {
    int i; // this i hides the i in C

    D(int a, int b) {
        super.i = a; // i in C
        i = b; // i in D
    }

    void show() {
        System.out.println("i in superclass: " + super.i);
        System.out.println("i in subclass: " + i);
        super.show();
    }
}

public class UseSuper {
    public static void main(String[] args) {
        D subOb = new D( a: 1,  b: 2);
        subOb.show();
    }
}
```

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

Method Overriding

```java
3     class Base {
4         int a;
5         Base(int a) {
6             this.a = a;
7         }
8         void show() {
9             System.out.println(a);
10        }
11    }
12
13    class Child extends Base {
14        int b;
15
16        Child(int a, int b) {
17            super(a);
18            this.b = b;
19        }
20
21        // the following method overrides Base class's show()
22        @Override // this is an annotation (optional but recommended)
23        void show() {
24            System.out.println(a + ", " + b);
25        }
26    }
```

```java
28    public class MethodOverride {
29        public static void main(String[] args) {
30            Child o = new Child( a: 10, b: 20);
31            o.show();
32            Base b = o;
33            b.show(); // will call show of Override
34        }
35    }
```

Dynamic Method Dispatch

```java
3    class P {
4        void call() {
5            System.out.println("Inside P's call method");
6        }
7    }
8    class Q extends P {
9        void call() {
10            System.out.println("Inside Q's call method");
11        }
12    }
13   class R extends Q {
14       void call() {
15           System.out.println("Inside R's call method");
16       }
17   }
18
19   public class DynamicDispatchTest {
20       public static void main(String[] args) {
21           P p = new P(); // object of type P
22           Q q = new Q(); // object of type Q
23           R r = new R(); // object of type R
24           P  x;           // reference of type P
25           x = p;          // x refers to a P object
26           x.call();       // invoke P's call
27           x = q;          // x refers to a Q object
28           x.call();       // invoke Q's call
29           x = r;          // x refers to a R object
30           x.call();       // invoke R's call
31       }
32   }
```

Abstract Class

- ***abstract class A***

- contains abstract method ***abstract method f()***

- No instance can be created of an abstract class

- The subclass must implement the abstract method

- Otherwise the subclass will be a abstract class too

Abstract Class

```java
3   abstract class S {
4       // abstract method
5       abstract void call();
6       // concrete methods are still allowed in abstract classes
7       void call2() {
8           System.out.println("This is a concrete method");
9       }
10  }

12  class T extends S {
13      void call() {
14          System.out.println("T's implementation of call");
15      }
16  }

18  class AbstractDemo {
19      public static void main(String args[]) {
20          //S s = new S(); // S is abstract; cannot be instantiated
21          T t = new T();
22          t.call();
23          t.call2();
24      }
25  }
```

Anonymous Subclass

```java
3   abstract class S {
4       // abstract method
5       abstract void call();
6       // concrete methods are still allowed in abstract classes
7       void call2() {
8           System.out.println("This is a concrete method");
9       }
10  }
11
12  class AbstractDemo {
13      public static void main(String args[]) {
14          //S s = new S(); // S is abstract; cannot be instantiated
15          S s = new S() {
16              void call() {
17                  System.out.println("Call method of an abstract class");
18              }
19          };
20          s.call();
21      }
22  }
```

Using final with Inheritance

To prevent overriding

```java
class A {
    final void f() {
        System.out.println("This is a final method.");
    }
}

class B extends A {
    void f() { // Error! Can't override.
        System.out.println("Illegal!");
    }
}
```

To prevent inheritance

```java
final class A {
    //...
}

// The following class is illegal.
class B extends A { // Error! Can't subclass A
    //...
}
```

Object Class

- There is one special class, **_Object_**, defined by Java
- All other classes are subclasses of Object
- That is, Object is a superclass of all other classes
- This means that a reference variable of type Object can refer to an object of any other class
- Also, since arrays are implemented as classes, a variable of type Object can also refer to any array

Object's toString()

- The **toString**( ) method returns a string that contains a description of the object on which it is called

- Also, this method is automatically called when an object is output using println()

- Many classes override this method

- Doing so allows them to provide a description specifically for the types of objects that they create

Object's toString()

```java
class Point {
    int x, y;

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}

public class ObjectTest {
    public static void main(String[] args) {
        Point p1 = new Point( x: 10, y: 20);
        // without override toString() method the
        // following will print something like this
        // Point@3cd1a2f1
        System.out.println(p1);
    }
}
```

Object's equals() and hashCode()

- **==** is a reference comparison, whether both variables refer to the same object

- Object's **equals**() method does the same thing

- String class override **equals**() to check contents

- If you want two different objects of a same class to be equal then you need to override **equals**() and **hashCode**() methods
  - **hashCode**() needs to return same value to work properly as keys in Hash data structures

Object's equals() and hashCode()

```java
 3    import java.util.HashMap;
 4    import java.util.Objects;
 5
 6    class Point {
 7        int x, y;
 8        Point(int x, int y) {
 9            this.x = x;
10            this.y = y;
11        }
12
13        @Override
14        public boolean equals(Object o) {
15            if (o == this) return true;
16            if (!(o instanceof Point)) {
17                return false;
18            }
19            Point p = (Point) o;
20            if (p.x == this.x && p.y == this.y) return true;
21            return false;
22        }
23
24        @Override
25        public int hashCode() {
26            return Objects.hash(x, y);
27        }
28    }
```

```java
30    public class ObjectTest {
31        public static void main(String[] args) {
32            Point p1 = new Point( x: 10, y: 20);
33            Point p2 = new Point( x: 10,  y: 20);
34            System.out.println(p1.equals(p2));
35            System.out.println(p1 == p2);
36            HashMap m = new HashMap();
37            m.put(p1, "Hello");
38            System.out.println(m.get(p2));
39        }
40    }
41
```