INHERITANCE

CSE215 | DAFFODIL INTERNATIONAL UNIVERSITY

OBJECT ORIENTED PROGRAMMING LAB

SESSION - 04

04

**Inheritance:**

* **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.
* The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
* Inheritance represents the **IS-A relationship**, also known as parent-child relationship.
* extends is the keyword used to inherit the properties of a class. It indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
* Child class share property and behavior from parent.

**Why we use Inheritance in Java?**

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

Syntax:

class Super {

.....

.....

}

class Sub extends Super {

.....

.....

}

\* There can be **three** types of inheritance in java:

* Single.
* Multilevel.
* Hierarchical.
* Multiple (not supported in java)



## Single Inheritance Example

class Animal

{

void eat()

{

System.out.println("eating...");

}

}

class Dog extends Animal

{

void bark()

{

System.out.println("barking...");

}

}

class TestInheritance

{

public static void main(String args[])

{

Dog d=new Dog();

d.bark();

d.eat();

}

}

Output:

barking...

eating...

**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();

}

}

Output:

weeping...

barking...

eating..

**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();

}

}

**Why Multiple inheritance doesn’t supported by Java?**

* To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

What Java supports?

|  |
| --- |
| A |
|  |

|  |
| --- |
| <<B>> |
|  |
|  |

Interface

is a

|  |
| --- |
| C |
|  |
|  |

Inheritance Exercise:

|  |
| --- |
| Shape |
| # location: int |
| + Shape (int)  + display(): void |

is a

|  |
| --- |
| Rectangle |
| - height: int  - width: int |
| + Rectangle (int, int)  + display(): void  + main (String [ ] ): void |

public class Shape {

private int location;

public Shape(int location) {

this.location = location;

}

public void display() {

System.*out*.println("Location: "+location);

}

}

public class Rectangle extends Shape {

private int height;

private int width;

public Rectangle(int height, int width, int location) {

super(location);

this.height = height;

this.width = width;

}

public void display() {

super.display();

System.*out*.println("Height " + height + " width " + width);

}

public static void main(String[] args) {

Rectangle r = new Rectangle(10, 5, 5);

r.display();

}

}

Output is-

Location: 5

Height 10 width 5

**Association:**

* The associationrelationship indicates that a class knows about, and has a reference to, another class.
* Associations can be described as a "has-a" relationship. Because the typical implementation in Java is through the use of an instance field.
* [Aggregation](https://www.thoughtco.com/aggregation-2033995) and [composition](https://www.thoughtco.com/java-composition-definition-and-example-2034053) are types of association relationships.

**Aggregation:**

* If a class have an entity reference, it is known as Aggregation.
* Aggregation represents HAS-A relationship.

-Consider a situation, Employee has an object of Address, address object contains its own information such as city, state, country etc. In such case relationship is Employee HAS-A address.

Look the UML & example below:

|  |
| --- |
| Empolyee |
| #id:int  #name:String |
| + Empolyee(int , String)  +display():void  +main(String[] x):void |

|  |
| --- |
| Address |
| -city:String  -state:String  -country:String |
| +Address( String, String, String) |

has a

public class Address{

private String city, state, country;

public Address(String city, String state, String country){

this.city = city;

this.state = state;

this.country = country;

}

}

public class Employee

{

int id;

String name;

Address address;

public Employee(int id, String name,Address address){

this.id = id;

this.name = name;

this.address=address;

}

Public void display(){

System.out.println(id+" "+name);

System.out.println(address.city+" "+address.state+" "+address.country);

}

public static void main(String[] args){

Address address1=new Address("abc","UP","BD");

Address address2=new Address("def","UP","BD");

Employee e=new Employee(111,"MRI",address1);

Employee e2=new Employee(114,"Logan",address2);

e.display();

e2.display();

}

}

Output:

111 MRI

abc UP BD

112 Logan

def UP BD

**Example: (association implementation)**

|  |
| --- |
| Rectangle |
| - height: int  - width: int |
| + Rectangle (int, int)  + display(): void  + main (String [ ] ): void |

|  |
| --- |
| Circle |
| - radius: double |
| + Circle (double)  + display(): void |

has

has

|  |
| --- |
| Line |
| - length: int |
| + Line (int)  + display(): void |

public class Line {

private int length;

public Line(int length) {

this.length = length;

}

public void display() {

System.*out*.println("Length: " + length);

}

}

public class Circle {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

public void display() {

System.*out*.println("Radius: " + radius);

}

}

public class Rectangle {

private int height;

private int width;

private Line line;

private Circle circle;

public Rectangle(int height, int width, Line line, Circle circle) {

this.height = height;

this.width = width;

this.line = line;

this.circle = circle;

}

public void display() {

line.display();

circle.display();

System.*out*.println("Height: " + height + " Width: " + width);

}

public static void main(String[] args) {

Line l = new Line(10);

Circle c = new Circle(3.5);

Rectangle r = new Rectangle(10, 15, l, c);

r.display();

}

}

Output is-

Length: 10

Radius: 3.5

Height: 10 Width: 15

* Java Static keyword:

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

1) Java static variable

If you declare any variable as static, it is known static variable.

* The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
* The static variable gets memory only once in class area at the time of class loading.

### **Example of static variable**

1. //Program of static variable
3. **class** Student8{
4. **int** rollno;
5. String name;
6. **static** String college ="ITS";
8. Student8(**int** r,String n){
9. rollno = r;
10. name = n;
11. }
12. **void** display (){System.out.println(rollno+" "+name+" "+college);}
14. **public** **static** **void** main(String args[]){
15. Student8 s1 = **new** Student8(111,"Karan");
16. Student8 s2 = **new** Student8(222,"Aryan");
18. s1.display();
19. s2.display();
20. }
21. }

Output:111 Karan ITS

222 Aryan ITS

## 2) Java static method

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* static method can access static data member and can change the value of it.

### **Example of static method**

1. //Program of changing the common property of all objects(static field).
3. **class** Student9{
4. **int** rollno;
5. String name;
6. **static** String college = "ITS";
8. **static** **void** change(){
9. college = "BBDIT";
10. }
12. Student9(**int** r, String n){
13. rollno = r;
14. name = n;
15. }
17. **void** display (){System.out.println(rollno+" "+name+" "+college);}
19. **public** **static** **void** main(String args[]){
20. Student9.change();
22. Student9 s1 = **new** Student9 (111,"Karan");
23. Student9 s2 = **new** Student9 (222,"Aryan");
24. Student9 s3 = **new** Student9 (333,"Sonoo");
26. s1.display();
27. s2.display();
28. s3.display();
29. }
30. }

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

**super keyword in java**

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

## Usage of java super Keyword

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

## super keyword Exercise:

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

1. **class** Person{
2. **int** id;
3. String name;
4. Person(**int** id,String name){
5. **this**.id=id;
6. **this**.name=name;
7. }
8. }
9. **class** Emp **extends** Person{
10. **float** salary;
11. Emp(**int** id,String name,**float** salary){
12. **super**(id,name);//reusing parent constructor
13. **this**.salary=salary;
14. }
15. **void** display(){System.out.println(id+" "+name+" "+salary);}
16. }
17. **class** TestSuper5{
18. **public** **static** **void** main(String[] args){
19. Emp e1=**new** Emp(1,"ankit",45000f);
20. e1.display();
21. }}

Output:

1 ankit 45000