## 16.2 Inheritance

- Inheritance is an oops concept which provides functionality to use reuse methods by creating parent and child class.
- it is also called as “**is-A**“ relationship.
- In inheritance a child/sub class is created by using extend keyword in signature of a parent/super class.

```
public class HR extends Employee{

// here parent is Employee and child is HR

}
```

- Using inheritance we can override methods from the parent class into child class and include logic as per requirements.
- Simply in inheritance property of parent class are consumed by child class.

**Inheritance from variable perspective →**

***Parent class [ Employee ]*** →

```
public class Employee{
public int a = 12;
public int b = 24;
}
```

***Child class [HR]*** →

```
public class HR extends Employee{
public int a = 99;
public int c = 100;
}
```

***Demo.java***

```
public class Demo{

public static void main(String[] args){

 //case 1 -> Obj with Employee ref and constructor
 Employee e = new Employee();
System.out.println(e.a); // op -12
System.out.println(e.b); // op - 24
System.out.println(e.c); // compilation error as variable c is not visible to parent

//case 2 -> Obj with HR ref and constructor
```

```
HR h = new HR();
System.out.println(h.a); // op -99 - var a is found in HR
System.out.println(h.b); // op - 24 - var b is not in HR but due to parent child
                         //           relation, if it is not in HR,
                         //           flow will try to find it in parent class
System.out.println(h.c); // op - 100 - var c is found in HR

//case 3 ->
Employee emp = new HR();
System.out.println(hr.a);
System.out.println(hr.b);
System.out.println(hr.c);

}
}
```

### Inheritance from method perspective →

***Parent class → Employee.java***

```
public class Employee{

public void getAge(){
  System.out.println("In getAge - parent class");

}

public void getArea(){
System.out.println("In getArea - parent class");

}

}
```

### Child class → HR.java

***N.B.*** → Right click in class body → source → Implement/override methods(click on it, get pop up to select methods to override better select all) → and all methods will be overridden in child class.

```
public class HR extends Employee{

@Override
public void getAge(){

System.out.println("In getAge - child class");
}

@Override
public void getArea(){
System.out.println("In getArea - child class");
}

//non-overriden method
public void getSalary(){
System.out.println("In getSalary - child class");
```

```
17
18 }
19 }
```

***Demo.java***

```java
public class Demo{

//complete this piece of code in notebook.

public static void main(String[] args){
Employee e = new Employee();

HR h = new HR();

Employee emp = new HR();

}


}
```

### Inheritance from Constructor perspective →

- Constructors have same name as a class name so they can be inherited to child class as they will child class will have a different name.

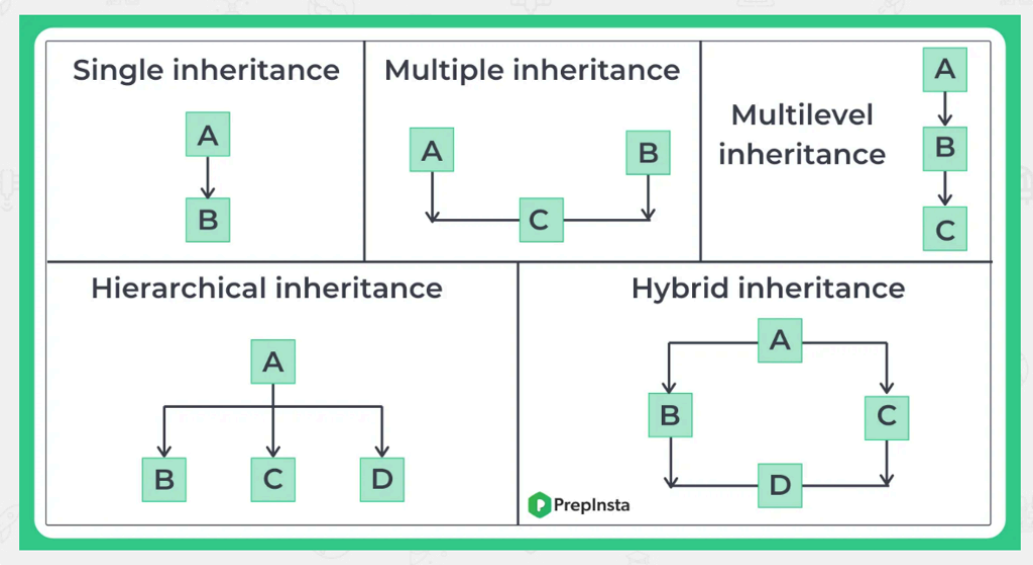
### Advantages of Inheritance in Java →

- **Code Reusability:** Inheritance allows for code reuse and reduces the amount of code that needs to be written. The subclass can reuse the properties and methods of the superclass, reducing duplication of code.
- **Abstraction:** Inheritance allows for the creation of abstract classes that define a common interface for a group of related classes. This promotes abstraction and encapsulation, making the code easier to maintain and extend.
- **Class Hierarchy:** Inheritance allows for the creation of a class hierarchy, which can be used to model real-world objects and their relationships.
- **Polymorphism:** Inheritance allows for polymorphism, which is the ability of an object to take on multiple forms. Subclasses can override the methods of the superclass, which allows them to change their behavior in different ways.

### Types of inheritance :

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance

- Hybrid Inheritance




💡 ***Java does not support multiple inheritance for classes*** *directly due to the complexity and potential for ambiguity it introduces (such as the "Diamond Problem", where a class inherits from two classes that have a common ancestor, leading to ambiguity in which ancestor's method or property to use).*

**Conclusion →**

| | Can inherit? | Important point |
|---|---|---|
| Class | ✅ | using extends keyword |
| method | ✅ | |
| private method | ❌ | |
| Variable | ✅ | |
| private variable | ❌ | |

Reference document for '**super**' and '**this**' keyword → refer following document for super and this keyword.

1. super and this keywords in Java - GeeksforGeeks

**Interview question links for inheritance →**

1. 50 Java Inheritance Interview Questions and Answers - Scientech Easy

Document by ***Suyash*** 😇