Day 11

Modifiers In Java

1. PRIVATE
2. PROTECTED
3. PUBLIC
4. STATIC
5. FINAL
6. SYNCHRONIZED
7. VOLATILE
8. TRANSIENT
9. NATIVE
10. INTERFACE
11. ABSTRACT
12. STRICT

Access Modifiers

- If we control visibility of members of the class then we should use access modifiers.
- There are 4 acccess modifiers in Java:
    1. private(-)
    2. package level private / default(~)
    3. protected(#)
    4. public(+)
- Except constructor, all the members of super class inherit into sub class.

Types of Inheritance

```
1. Interface Inheritance.
2. Implementation Inheritance.
```

- Above types are further classified into 4 types:
    1. Single Inheritance
    2. Multiple Inheritance
    3. Hierarchical Inheritance
    4. Multilevel Inheritance

Interface Inheritance

- During inheritance, if super type and sub type is interface then it is called interface inheritance.

Single Inheritance

```
interface A
{    }
interface B extends A   //OK:Single Interface Inheritance
{    }
```

- During inheritance, if there is single super interface and single sub interface then it is called single interface inheritance.

Multiple Inheritance

```
interface A
{    }
interface B
{    }
interface C extends A, B   //OK:Multiple Interface Inheritance
{    }
```

- During inheritance, if there are multiple super interfaces and single sub interface then it is called multiple interface inheritance.

Hierarchical Inheritance

```
interface A
{    }
interface B extends A
{    }
interface C extends A
{    }
```

- During inheritance, if there is single super interface and multiple sub interfaces then it is called hierarchical interface inheritance.

Multi Level Inheritance

```
interface A
{    }
interface B extends A
{    }
interface C extends B
{    }
```

- During inheritance, if single interface inheritance is having multiple levels then it is called multilevel interface inheritance.

Implementation Inheritance

- During inheritance, if super type and sub type is class then it is called implementation inheritance.

Single Inheritance

```
class A
{    }
class B extends A   //OK:Single Implementation Inheritance
{    }
```

- During inheritance, if there is single super class and single sub class then it is called single implementation inheritance.

Multiple Inheritance

```
class A
{    }
class B
{    }
class C extends A, B   //NOT OK:Multiple Implementation Inheritance
{    }
```

- During inheritance, if there are multiple super classes and single sub class then it is called multiple implementation inheritance.
- Java do not support multiple implementation inheritance.

Hierarchical Inheritance

```
class A
{    }
class B extends A   //OK
{    }
class C extends A   //OK
{    }
```

- During inheritance, if there is single super class and multiple sub classes then it is called hierarchical implementation inheritance.

Multi Level Inheritance

```
class A
{    }
class B extends A
```

```
{    }
class C extends B
{    }
```

- During inheritance, if single implementation inheritance is having multiple levels then it is called multilevel implementation inheritance.
- In Java multi class inheritance is not allowed. In other words, class can extends only one super class(abstract/ concrete).
- During inheritance, members of super class inherit into sub class. Hence using sub class instance we can access members of super class as well as sub class.
- During inheritance, members of sub class do not inherit into super class hence using super class instance we can access members of super clas only.
- Members of super class inherit into sub class hence we can consider sub class instance as a super class instance.
- Example : Every employee is a person.
- Since we can consider sub class instance as as super class instance, we can use it in place of super class instance.
- Consider following code:

```
Employee emp1 = null;   //OK

Employee emp2 = new Employee(); //OK
```

```
Person p1 = null;//OK

Person p2 = new Person( );     //OK
```

```
Employee emp = new Employee( )//OK

Person p1 = emp;    //OK

Person p2 = new Employee( );    //OK
```

- Members of sub class do not inherit into super class hence We can not consider super class instance as a sub class instance.
- Example : Every person is not a employee.

- Since, super class instance, can not be considered as sub class instance, we can not use it in place of sub class instance.
- Consider following code:

```
Person p1 = null;   //OK

Person p2 = new Person( );  //OK
```

```
Employee emp1 = null;   //OK

Employee emp2 = new Employee( );    //OK
```

```
Person p = new Person( );

Employee emp1 = p;   //NOT OK

Employee emp2 = new Person( );   //NOT OK
```

- Process of converting reference of sub class into reference of super class is called upcasting.

```
public static void main(String[] args) {
    Derived derived = new Derived();
    derived.displayRecord();
    //Base base = ( Base )derived;  //Upcasting : OK
    Base base = derived;    //Upcasting : OK
    base.showRecord();
}
```

- In other words, super class reference can contain reference of sub class instance. It is called upcasting.

```
public static void main(String[] args) {
    Base base = new Derived();  //Upcasting : OK
}
```

- In case of upcasting, using super class reference variable, we can access fields of super class only.

```
public static void main(String[] args) {
    Base base = new Derived();  //Upcasting : OK
    System.out.println("Num1    :   "+base.num1);   //OK
```

```
    System.out.println("Num2    :   "+base.num2);   //OK
    System.out.println("Num3    :   "+base.num3);   //NOT OK
}
```

- In Case of Upcasting, using super class reference variable, we can not access sub class specific methods(which are not available in super class).

```
public static void main(String[] args) {
    Base base = new Derived();  //Upcasting : OK
    base.showRecord();//OK
    //base.displayRecord( );    //NOT OK
}
```

- In case of upcasting, from sub class instance, super class reference variable can access only member of super class. It is also called object slicing.
- Upcassting help us to reduce maintenance of the application.
- Process of converting reference of super class into reference of sub class is called downcasting.
- In Case of downcasting, explicit typecasting is mandatory.

```
public static void main(String[] args) {
    Base base = new Derived();  //Upcasting : OK
    base.setNum1(10);   //OK
    base.setNum2(20);   //OK

    Derived derived = (Derived) base;   //Downcasting
    derived.setNum3(30);    //OK

    System.out.println("Num1    :   "+base.getNum1());
    System.out.println("Num2    :   "+base.getNum2());
    System.out.println("Num3    :   "+derived.getNum3());
}
```

- process of redefining method of super class inside sub class is called method overriding.
- Process of calling method of sub class using reference of super class is called dynamic method dispatch.

```
public static void main(String[] args) {
    Base base = new Derived();  //Upcasting
    base.print();   //Runtime Polymorphism / Dynamic Method Dispatch
}
```