**Day 17**

**Access Modifiers in Java**

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

**Aggregation**

* If a class have an entity reference, it is known as Aggregation.
* Aggregation represents HAS-A relationship.
* Aggregation in Java is a relationship between two classes that is best described as a "has-a" and "whole/part" relationship. It is a more specialized version of the association relationship. The aggregate class contains a reference to another class and is said to have ownership of that class.
* Its usage is code reusability.

Example

Class Address

{

String city, state, country;

Address(String city, String state, String country)

{

this.city = city;

this.state = state;

this.country = country;

}

}

Class Employee

{

int id;

String name;

Address address; // aggregation

Employee(int id, String name, Address address )

{

this.id = id;

this.name = name;

this.address = address;

}

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(“TVM”,”Kerala”,”India”);

Address address2 = new Address(“Chennai”,”Tamil Nadu”,”India”);

Employee emp1 = new Employee(11,”Anooja”,address1);

Employee emp2 = new Employee(12,”Rohith”,address2);

emp1.display();

emp2.display();

}

}

**Method Overriding (Run Time Polymorphism)**

In Java, method overriding occurs when a subclass (child class) has the same method as the parent class. In other words, method overriding occurs when a subclass provides a particular implementation of a method declared by one of its parent classes.

Rules

1. Method name must be same as that in parent class
2. Method must have same parameters as the parent class.
3. There must be a IS A relationship.

Example

class A

{

void display()

{

System.out.println(“Display”);

}

}

class B extends A

{

void display()

{

System.out.println(“Printing”);

}

public static void main(String args[])

{

B ob = new B();

ob.display();

}

}

**Difference between Method Overloading and Method Overriding**

**Method Overloading Method Overriding**

|  |  |
| --- | --- |
| 1. Method overloading is a compile-time polymorphism. | Method overriding is a run-time polymorphism. |
| 1. It helps to increase the readability of the program. | It is used to grant the specific implementation of the method which is already provided by its parent class or superclass. |
| 1. It occurs within the class. | It is performed in two classes with inheritance relationships. |
| 1. Method overloading may or may not require inheritance. | Method overriding always needs inheritance. |
| 1. In method overloading, methods must have the same name and different signatures. | In method overriding, methods must have the same name and same signature. |
| 1. In method overloading, the return type can or can not be the same, but we just have to change the parameter. | In method overriding, the return type must be the same or co-variant. |
| 1. Static binding is being used for overloaded methods. | Dynamic binding is being used for overriding methods. |
| 1. Poor Performance due to compile time polymorphism. | It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime. |
| 1. Private and final methods can be overloaded. | Private and final methods can’t be overridden. |
| 1. Argument list should be different while doing method overloading. | Argument list should be same in method overriding. |