Method Overriding in Java

Method overriding is a process of overriding base class method by derived class method with more specific definition.

Method overriding performs only if two classes have **is-a** relationship. It means classes must have inheritance. In other words, It is performed between two classes using inheritance relation.

In overriding, methods of both classes must have same name and equal number of parameters.

Method overriding is also referred to as runtime polymorphism because calling method is decided by JVM during runtime.

The key benefit of overriding is the ability to define method that's specific to a particular subclass type.

Rules for Method Overriding

1. Method name must be same for both parent and child classes.

2. Access modifier of child method must not restrictive than parent class method.

3. Private, final and static methods cannot be overridden.

4. There must be an IS-A relationship between classes (inheritance).

Example of Method Overriding

Below we have simple code example with one parent class and one child class wherein the child class will override the method provided by the parent class.

```
class Animal
{
	public void eat()
	{
		System.out.println("Eat all eatables");
	}
}
```

EQUALITY EDUCATION

```
class Dog extends Animal
{
	public void eat()   //eat() method overridden by Dog class.
	{
		System.out.println("Dog like to eat meat");
	}
	public static void main(String[] args)
	{
		Dog d = new Dog();
		d.eat();
	}
}
```

As you can see here Dog class gives its own implementation of eat() method. For method overriding, the method must have same name and same type signature in both parent and child class.

NOTE: Static methods cannot be overridden because, a static method is bounded with class whereas instance method is bounded with object.

Example: Access modifier is more restrictive in child class

Java does not allow method overriding if child class has more restricted access modifier than parent class.

In the below example, to the child class method, we set protected which is restricted than pubic specified in parent class.

```
class Animal
{
	public void eat()
	{
		System.out.println("Eat all eatables");
```

EQUALITY EDUCATION

```
        }
}
class Dog extends Animal
{
        protected void eat()   //error
        {
                System.out.println("Dog like to eat meat");
        }
        public static void main(String[] args)
        {
                Dog d = new Dog();
                d.eat();
        }
}
```

Covariant return type

Since Java 5, it is possible to override a method by changing its return type. If subclass overrides any method by changing the return type of super class method, then the return type of overridden method must be subtype of return type declared in original method inside the super class. This is the only way by which method can be overridden by changing its return type.

Example:

```
class Animal
{
   Animal getObj()
   {
      System.out.println("Animal object");
```

EQUALITY EDUCATION

```
        return new Animal();
    }
}
class Dog extends Animal
{
    Dog getObj()    //Legal override after Java5 onward
    {   System.out.println("Dog object");
        return new Dog();
    }
    public static void main(String[] args) {
        new Dog().getObj();
    }
}
```

Difference between Overloading and Overriding

Method overloading and Method overriding seems to be similar concepts but they are not. Let's see some differences between both of them:

Method Overloading	**Method Overriding**
Parameter must be different and name must be same.	Both name and parameter must be same.
Compile time polymorphism.	Runtime polymorphism.
Increase readability of code.	Increase reusability of code.
Access specifier can be changed.	Access specifier cannot be more restrictive than original method(can be less restrictive).
It is Compiled Time Polymorphism.	It is Run Time Polymorphism.
It is performed within a class	It is performed between two classes using inheritance relation.

EQUALITY EDUCATION

Method Overloading	Method Overriding
It is performed between two classes using inheritance relation.	It requires always inheritance.
It should have methods with the same name but a different signature.	It should have methods with same name and signature.
It can not have the same return type.	It should always have the same return type.
It can be performed using the static method	It can not be performed using the static method
It uses static binding	It uses the dynamic binding.
Access modifiers and Non-access modifiers can be changed.	Access modifiers and Non-access modifiers can not be changed.
It is code refinement technique.	It is a code replacement technique.
No keywords are used while defining the method.	Virtual keyword is used in the base class and overrides keyword is used in the derived class.
Private, static, final methods can be overloaded	Private, static, final methods can not be overloaded
No restriction is Throws Clause.	Restriction in only checked exception.
It is also known as Compile time polymorphism or static polymorphism or early binding.	It is also known as Run time polymorphism or Dynamic polymorphism or Late binding

EQUALITY EDUCATION