SYNTAX
TECHNOLOGIES

JAVA

Class 24

Agenda

Method Overriding
Method Overriding vs Method Overloading
Polymorphism in Java:
- Compile Time
- Run Time

Method Overriding in Java

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**.

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

Note: Without Inheritance method overriding is not possible.

Method Overriding in Java

Advantage of Java Method Overriding

Method Overriding is used to provide specific implementation of a method that is already provided by its super class.

Method Overriding is used for Runtime Polymorphism

Note: Without Inheritance method overriding is not possible.

Method Overriding in Java

Rules for Method Overriding

- must be IS-A relationship (inheritance).
- method must have same name as in the parent class and child class.
- method must have same parameter as in the parent class.

Note: Without Inheritance method overriding is not possible.

Example of method overriding in Java

We have defined the walk method in the subclass as defined in the parent class but it has some specific implementation.
The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Walking {
    void walk() {
    SOP("Man walking fastly");
    }
}
class Man extends walking {
     void walk() {
     SOP("Man walking slowly");
     }
}
```

```
class OverridingDemo {
     public static void main(String args[]) {
     Man obj = new Man();
     obj.walk();
     }
}
```

Note: Whenever we are calling overridden method using derived class object reference the highest priority is given to current class (derived class). We can see in the above example high priority is derived class.

Rules for Java Method Overriding

- The method signature i.e. method name, parameter list and return type have to match exactly in super and subclass.

- The access level **cannot be more restrictive** than the overridden method access level. For example: if the super class method is declared public then the overriding method in the sub class cannot be either private, default or protected.

What is not possible in Overriding

- If a method cannot be inherited then it cannot be overridden, therefore, private methods cannot be overridden.
- Constructors do not participate in inheritance, therefore, they cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- Main method cannot be overridden.
- A method declared final cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.

Important Note

Note: In overloading we have to check only methods names (must be same) and arguments types (must be different) except these the remaining like return type access modifiers etc. are not required to check
But in overriding every things check like method names arguments types return types access modifiers etc.

Modifiers
The access specifier for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the superclass can be made public, but not private, in the subclass.
You will get a compile-time error if you attempt to change an instance method in the superclass to a class method in the subclass, and vice versa.

Overloading	Overriding
Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading.	Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding.
Arguments of method must be different.	Argument of method must be same including order.
Method signature must be different.	Method signature must be same.
Private, static and final methods can be overloaded.	Private, static and final methods can not be override.
Access modifiers point of view no restriction.	Access modifiers point of view not reduced scope of Access modifiers but increased.
Also known as compile time polymorphism or static polymorphism or early binding.	Also known as run time polymorphism or dynamic polymorphism or late binding.
Overloading can be exhibited both are method and constructor level.	Overriding can be exhibited only at method label.
The scope of overloading is within the class.	The scope of overriding is base class and derived class.
Overloading can be done at both static and non-static methods.	Overriding can be done only at non-static method.
For overloading methods return type may or may not be same.	For overriding method return type should be same.

Polymorphism

Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms.

Polymorphism is the ability of an object to take on many forms.

Real life example of polymorphism

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter. Here one person present in different-different behaviors.

Polymorphism

Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways.

Polymorphism is the capability of a method to do different things based on the object that it is acting upon.

An important usage of polymorphism occurs in oops is how a parent class refers to a child class object.

Polymorphism can be achieved in two of the following ways:

1. **Compile time Polymorphism/Static Binding through Method Overloading**

2. **Run time Polymorphism/ Dynamic Binding through Method Overriding**

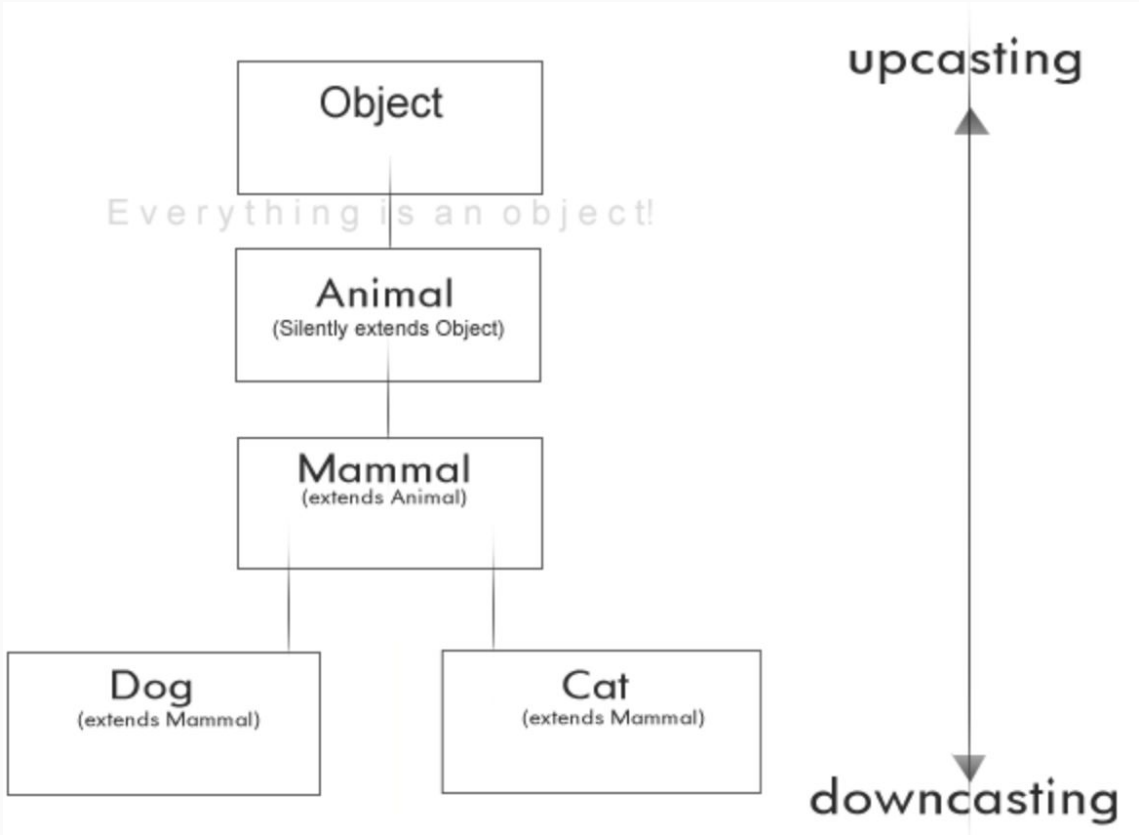
Type Casting in Java

Assigning a value of one type to a variable of another type is known as Type Casting.

2 two types of casting in Java:

- Widening Casting(Implicit/ Automatic)/UpCasting - converting lower data type into higher.
- Narrowing Casting(Explicit/ Manual)/DownCasting - converting higher data type into lower.

Type Casting in Java



Type Casting in Java

Upcasting: Casting from a subclass to a superclass is called upcasting

Downcasting: It's the casting from a superclass to a subclass.