Welcome you all

JAVA PROGRAMMING

DAY : 5

D.Sakthivel
Assistant Professor & Trainer,
KGiSL Micro College
KGiSL Campus, Coimbatore – 641 035.

Day 5

- Polymorphism
- Run time Polymorphism
- Method Overloading
- Method Overriding

Polymorphism in Java

- **Polymorphism in Java** is a concept by which we can perform a *single action in different ways*.
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- There are two types of polymorphism in Java: **compile-time polymorphism and runtime polymorphism**.
- We can perform polymorphism in java by **method overloading and method overriding.**
- If you overload a static method in Java, it is the example of compile time polymorphism.
- Here, we will focus on runtime polymorphism in

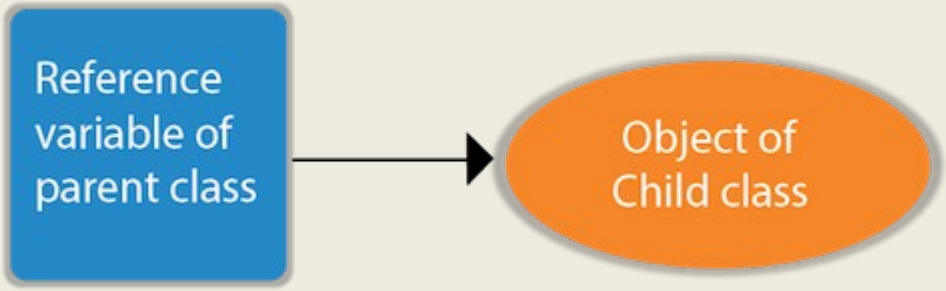
Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

- In this process, an overridden method is called through the reference variable of a superclass.
- The determination of the method to be called is based on the object being referred to by the reference variable.
- Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:




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

```
A a=new B();//upcasting
```

Runtime Polymorphism Examples

```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run()
{System.out.println("running safely with 60km
");}

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Output:

```
SBI Rate of Interest: 8.4
ICICI Rate of Interest: 7.3
AXIS Rate of Interest: 9.7
```

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Java Runtime Polymorphism with Multilevel Inheritance

```
class Animal{
void eat(){System.out.println("eating");}
}
class Dog extends Animal{
void eat(){System.out.println("eating fruits");}
}
class BabyDog extends Dog{
void eat(){System.out.println("drinking milk");}
public static void main(String args[]){
Animal a1,a2,a3;
a1=new Animal();
a2=new Dog();
a3=new BabyDog();
a1.eat();
a2.eat();
a3.eat();
}
```

Output:

```
eating
eating fruits
drinking Milk
```

Method Overloading in Java

- **If a class has multiple methods having same name but different in parameters,** it is known as **Method Overloading**.

Advantage of method overloading

- Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

- ***By changing number of arguments***
- ***By changing the data type***

In Java, Method Overloading is not possible by changing the return type of the method only.

1) Method Overloading: changing no. of arguments

- In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

```
22
33
```

2) Method Overloading: changing data type of arguments

- In this example, we have created two methods that differs in data type.
- The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b)
{return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

```
22
24.9
```

KGiSL MicroCollege

Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
```

guity

Output:

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

KGiSL MicrCollege

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class TestOverloading4{
public static void main(String[] args)
{System.out.println("main with String[]");}
public static void main(String args)
{System.out.println("main with String");}
public static void main()
{System.out.println("main without args");}
}
```

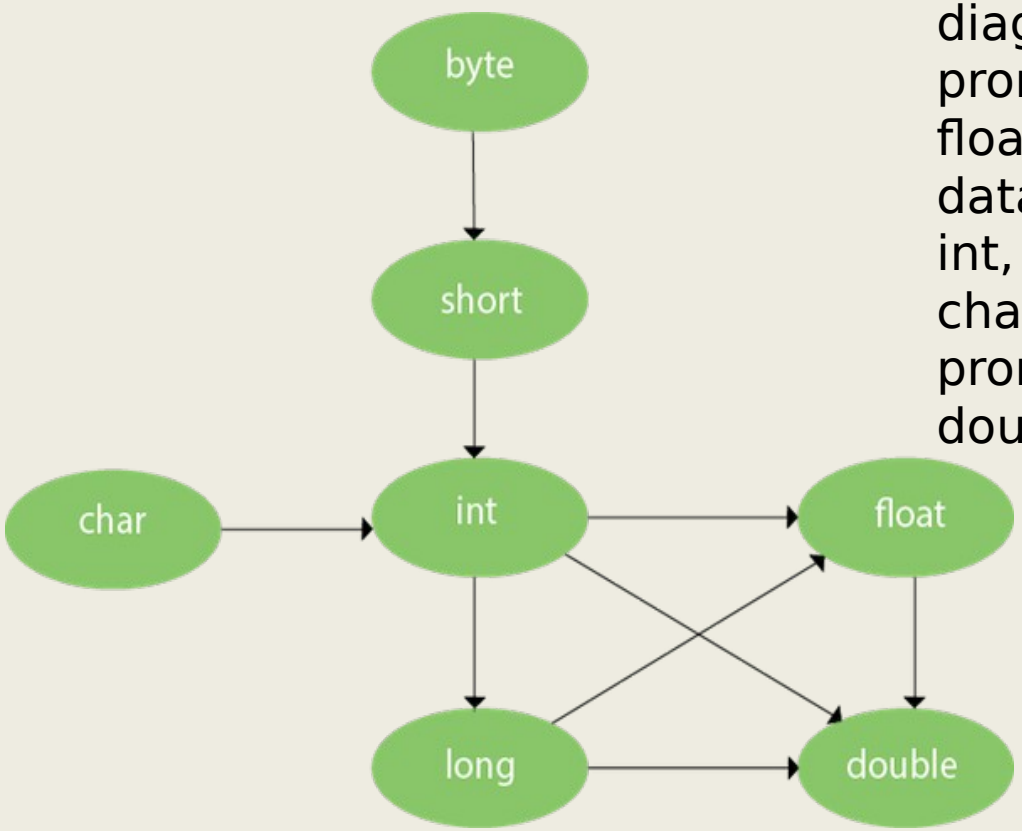
Output:

```
main with String[]
```

KGiSL MicroCollege

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:

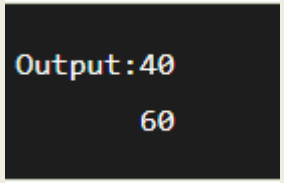


As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int,long,float or double and so on.

KGiSL MicroCollege

Example of Method Overloading with TypePromotion

```
class OverloadingCalculation1{
  void sum(int a,long b){System.out.println(a+b);}
  void sum(int a,int b,int c)
{System.out.println(a+b+c);}


  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculat
ion1();
  obj.sum(20,20);//
now second int literal will be promoted to long
  obj.sum(20,20,20);
```



Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
class OverloadingCalculation2{
  void sum(int a,int b)
{System.out.println("int arg method invoked");}
  void sum(long a,long b)
{System.out.println("long arg method invoked");}


  public static void main(String args[]){
  OverloadingCalculation2 obj=new OverloadingCalculat
ion2();
  obj.sum(20,20);//
now int arg sum() method gets invoked
  }
}
```

```
Output:int arg method invoked
```

KGiSL MicroCollege

Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

```
class OverloadingCalculation3{
  void sum(int a,long b)
{System.out.println("a method invoked");}
  void sum(long a,int b)
{System.out.println("b method invoked");}
 public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCal
culation3();
  obj.sum(20,20);//now ambiguity
  } }
```

```
Output:Compile Time Error
```

Method Overriding in Java

- **If subclass (child class) has the same method as declared in the parent class**, it is known as **method overriding in Java**.
- In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Method Overriding in Java

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Method Overriding in Java

Understanding the problem without method overriding
Let's understand the problem that we may face in the program if we don't use method overriding.

```
//Java Program to demonstrate why we need method overriding
//Here, we are calling the method of parent class with child
//class object.
//Creating a parent class
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

Output:

```
Vehicle is running
```

KGiSL MicroCollege

Method Overriding in Java

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation.

The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
//
Java Program to illustrate the use of Java Method Overridin
g
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
```

Method Overriding in Java

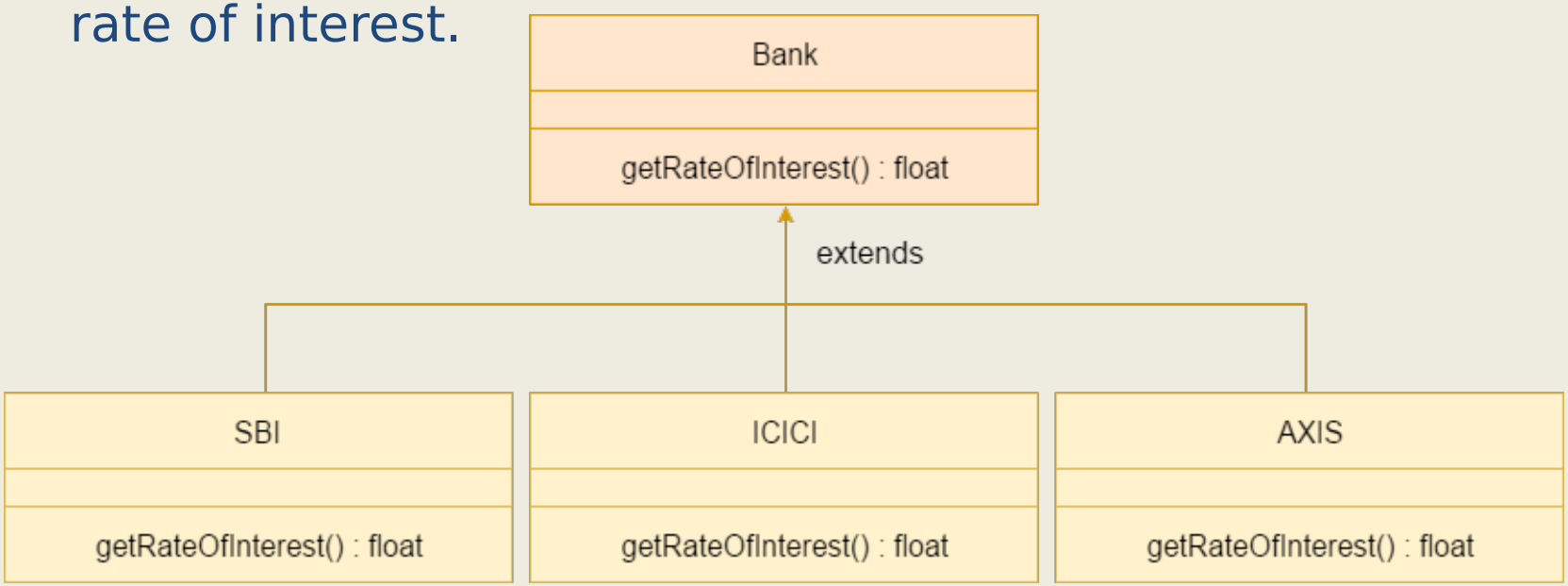
```
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run()
{ System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output:

```
Bike is running safely
```

A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



A real example of Java Method Overriding

```
//
Java Program to demonstrate the real scenario of Java Method
Overriding
//
where three classes are overriding the method of a parent clas
s.
//Creating a parent class.
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
```

A real example of Java Method Overriding

```
class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInt
erest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInt
erest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfIn
terest());
}
```

```
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Can we override static method?

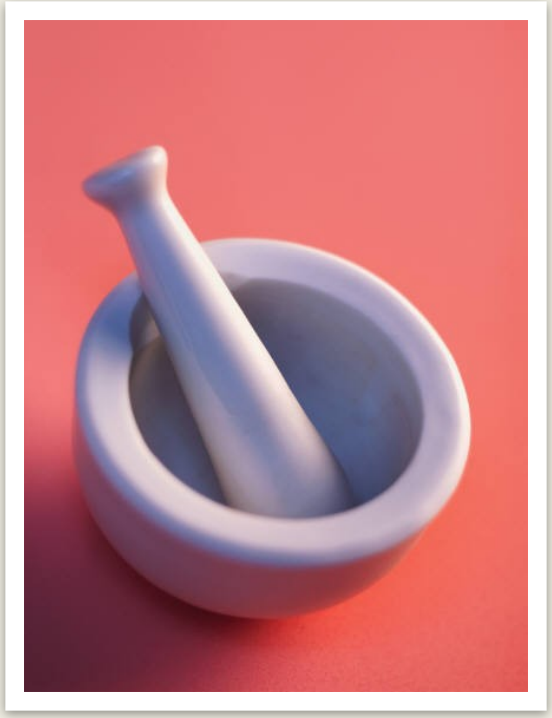
No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

Why can we not override static method?

- It is because the static method is bound with class whereas instance method is bound with an object.
- Static belongs to the class area, and an instance belongs to the heap area.

Can we override java main method?

No, because the main is a static method.



Thank You