Dynamic Method Dispatch

It is the process of calling a overridden method at run time by the super class reference variable.

- A superclass reference variable can refer to a subclass object
- When a superclass reference is used to call an overridden method, JVM determines which version of the method to execute based on the type of the object being referred to at the time call.

Advantages of dynamic method dispatch

1. It allows Java to support `overriding of methods`, which are important for run-time polymorphism.
2. It allows a class to define methods that will be shared by all its derived classes, while also allowing these sub-classes to define their `specific implementation` of a few or all of those methods.
3. It allows subclasses to incorporate their own methods and define their implementation.

```
class Bank

{

int getRateOfInterest()

{

System.out.println("Interest of bank");

return(0);

}//method has no body only declaration

}


class SBI extends Bank

{

int getRateOfInterest() //method is overriden

{
```

```
return 8;
}
void show()
{
 System.out.println("SBI show()");
}
}
class ICICI extends Bank
{
int getRateOfInterest() //overridden method
{
return 7;
}
}
class AXIS extends Bank
{
int getRateOfInterest() //overridden method
{
return 9;
}
}
class Test2{
public static void main(String args[])
```

```
{ Bank b=new Bank();
 System.out.println("Bank Rate of Interest: "+b.getRateOfInterest());

b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
//System.out.println("SBI Rate of Interest: "+b.show());// this is an error ie only overriden method of parent //class can be called by b
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("Axis Rate of Interest: "+b.getRateOfInterest());
/*
SBI s=(SBI)new Bank();
System.out.println("Bank Rate of Interest: "+s.getRateOfInterest());*/

}
}

SBI s1=new SBI();
System.out.println("Bank Rate of Interest: "+s1.getRateOfInterest());
```

Note:

Bank b=new Bank();
But its reference can be created which can point to a sub class object
Bank b=new SBI();
But reverse can not be possible
SBI s=new Bank();compilation error

Upcasting

When superclass reference variable holds the address of sub class object. It is known as upcasting.

Bank b=new SBI();

Bank b=new Bank();

SBI S=new SBI();

b=new SBI();

S=(SBI)new Bank(); // not possible

SBI object IS-A type of Bank type

By b only the overriden methods can be called.

Downcasting

When sub class reference variable holds the object of super class object, which is not possible automatically. For which explicit type casting needed.

SBI s=(SBI)new Bank();

This process is known as downcasting.

```
class employee
{
        int get_cl();
        {
    return 0;
        }
        double get_salary()
        {
```

```
        return 0;
        }
}
class security  extends employee
{

        int get_cl()
        {
        return 12;
        }
double get_salary()
        {
        return 9.6;}
}
class faculty extends employee
{
        int get_cl()
        {
        return 15;
        }
        double get_salary()
        {
        return 57000;
        }
```

```
}
class test
{
 public static void main(String[] s)
{
employee e;
 e=new security();
System.out.println(e.get_cl());
System.out.println(e.get_salary());
 e=new faculty();
System.out.println(e.get_cl());
System.out.println(e.get_salary());


}
}
```

Final Keyword

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context.

Final can be:

1. variable (**final int i=7**; //i value can not be changed)

2. method (final void n(){} // **method cannot be overridden by sub class)**

3. class (final class A{}// **class A can not be inherited)**

Abstract class

* A class that is declared with abstract keyword is known as abstract class
* It can have abstract(**without having body and only declaration is present**) and non-abstract methods (**method with body**).
* It needs to be extended and its methodmust be implemented.
* It cannot be instantiated.

```
abstract class Bank
{
abstract int getRateOfInterest();//method has no body only declaration
}
class SBI extends Bank{
int getRateOfInterest() //method is overriden
{return8;}
}
class ICICI extends Bank{
int getRateOfInterest() //overridden method
{return7;}
}
class AXIS extends Bank{
int getRateOfInterest() //overridden method
{return9;}
}
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.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
} }
```

Interface in Java

* An **interface in java** is a blueprint of a class. It has static constants and abstract methods.
* The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
* Java Interface also **represents IS-A relationship**.
* It cannot be instantiated just like abstract class.

Internal addition by compiler

```
//Interface declaration: by first user
interface shape{
void draw();
}
//Implementation: by second user
class Rectangle implements  shape{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements shape{
publicvoid draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
shape d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw(); }}
```

Output:drawing circle

Multiple inheritance in Java by interface

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print()
{System.out.println("Hello");}
public void show()
{System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
}}
```

Output
Hello
Welcome