Dynamic method dispatch or Runtime Polymorphism

- **Polymorphism in Java**

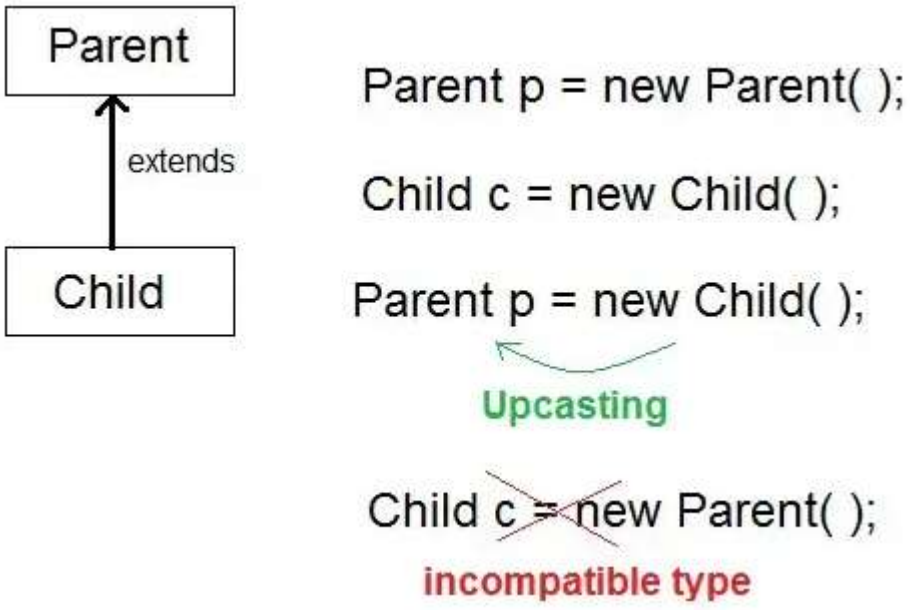
Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So, polymorphism means many forms.

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.

- **Dynamic method dispatch OR Runtime Polymorphism**

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refers to. In simple words the type of object which it referred determines which version of overridden method will be called.



- **Upcasting in Java**

When Parent class reference variable refers to Child class object, it is known as Upcasting. In Java this can be done and is helpful in scenarios where multiple child classes extends one parent class. In those cases, we can create a parent class reference and assign child class objects to it.

Example:

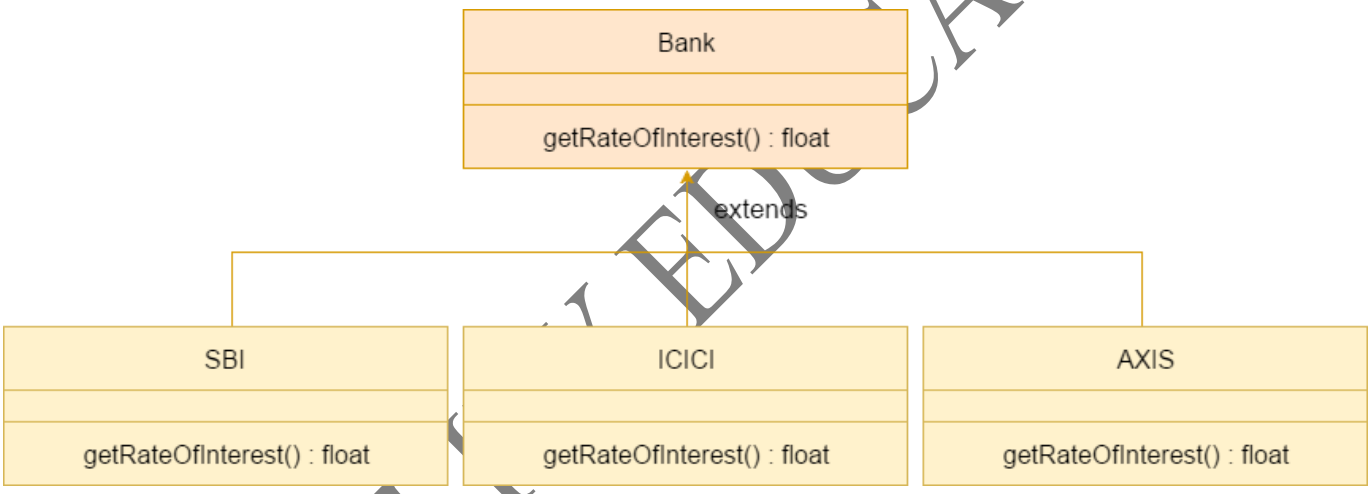
```
class Game
{
	public void type()
	{
		System.out.println("Indoor & outdoor");
	}
}
class Cricket extends Game
{
	public void type()
	{
		System.out.println("outdoor game");
	}
	public static void main(String[] args)
	{
		Game gm = new Game();
		Cricket ck = new Cricket();
		gm.type();
		ck.type();
		gm = ck;	//gm refers to Cricket object
		gm.type();	//calls Cricket's version of type
```

```
    }
}
```

Here, notice the last output. This is because of the statement, gm = ck;. Now gm.type() will call the Cricket class version of type() method. Because here gm refers to the cricket object.

- **Real Life example of Java Runtime Polymorphism: Bank**

Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest.




```
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());
    }
}
```

- **Advantages of Dynamic Method Dispatch**

Dynamic method dispatch allows Java to support overriding of methods which is central for run-time polymorphism.

It allows a class to specify methods that will be common to all of it's derivatives, while allowing subclasses to define the specific implementation of some or all of those methods.

- **Difference between Static binding and Dynamic binding in Java?**

Static binding in Java occurs during compile time while dynamic binding occurs during runtime. Static binding uses type(Class) information for binding while dynamic binding uses instance of class(Object) to resolve calling of method at run-time. Overloaded methods are bonded using static binding while overridden methods are bonded using dynamic binding at runtime.

In simpler terms, Static binding means when the type of object which is invoking the method is determined at compile time by the compiler. While Dynamic binding means when the type of object which is invoking the method is determined at run time by the compiler.