Runtime Polymorphism in Java

1. [Runtime Polymorphism](http://www.javatpoint.com/runtime-polymorphism-in-java)
2. [Upcasting](http://www.javatpoint.com/runtime-polymorphism-in-java#upcasting)
3. [Example of Runtime Polymorphism](http://www.javatpoint.com/runtime-polymorphism-in-java#polymorphismex)
4. [Runtime Polymorphism with data members](http://www.javatpoint.com/runtime-polymorphism-in-java#polymorphismdatamember)

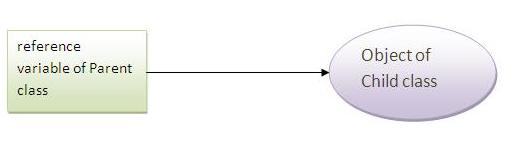
**Runtime polymorphism** or **Dynamic Method Dispatch**is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

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**

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



1. **class** A{}
2. **class** B **extends** A{}
3. A a=**new** B();//upcasting

Example of Runtime Polymorphism

In this example, we are creating two classes Bike and Splendar. Splendar class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime.

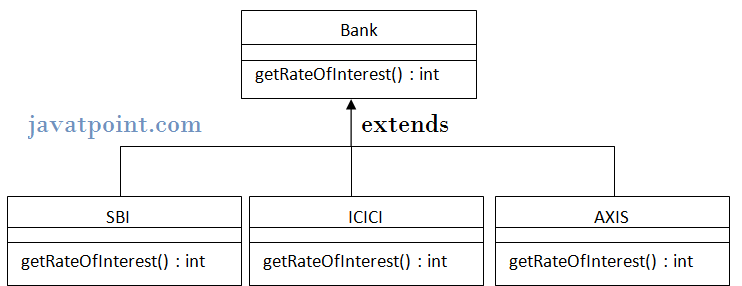
Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

1. **class** Bike{
2. **void** run(){System.out.println("running");}
3. }
4. **class** Splender **extends** Bike{
5. **void** run(){System.out.println("running safely with 60km");}
7. **public** **static** **void** main(String args[]){
8. Bike b = **new** Splender();//upcasting
9. b.run();
10. }
11. }

Output:running safely with 60km.

Real example of Java Runtime Polymorphism

Consider a scenario, Bank is a class that provides method to get the rate of interest. But, rate of interest may differ according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.



Note: It is also given in method overriding but there was no upcasting.

1. **class** Bank{
2. **int** getRateOfInterest(){**return** 0;}
3. }
5. **class** SBI **extends** Bank{
6. **int** getRateOfInterest(){**return** 8;}
7. }
9. **class** ICICI **extends** Bank{
10. **int** getRateOfInterest(){**return** 7;}
11. }
12. **class** AXIS **extends** Bank{
13. **int** getRateOfInterest(){**return** 9;}
14. }
16. **class** Test{
17. **public** **static** **void** main(String args[]){
18. Bank b1=**new** SBI();
19. Bank b2=**new** ICICI();
20. Bank b3=**new** AXIS();
21. System.out.println("SBI Rate of Interest: "+b1.getRateOfInterest());
22. System.out.println("ICICI Rate of Interest: "+b2.getRateOfInterest());
23. System.out.println("AXIS Rate of Interest: "+b3.getRateOfInterest());
24. }
25. }

Output:

SBI Rate of Interest: 8

ICICI Rate of Interest: 7

AXIS Rate of Interest: 9

Runtime Polymorphism with data member

|  |
| --- |
| Method is overridden not the datamembers, so runtime polymorphism can't be achieved by data members. |
| In the example given below, both the classes have a datamember speedlimit, we are accessing the datamember by the reference variable of Parent class which refers to the subclass object. Since we are accessing the datamember which is not overridden, hence it will access the datamember of Parent class always. |

***Rule: Runtime polymorphism can't be achieved by data members.***

1. **class** Bike{
2. **int** speedlimit=90;
3. }
4. **class** Honda **extends** Bike{
5. **int** speedlimit=150;
7. **public** **static** **void** main(String args[]){
8. Bike obj=**new** Honda();
9. System.out.println(obj.speedlimit);//90
10. }

Output:90

Runtime Polymorphism with Multilevel Inheritance

Let's see the simple example of Runtime Polymorphism with multilevel inheritance.

1. **class** Animal{
2. **void** eat(){System.out.println("eating");}
3. }
5. **class** Dog **extends** Animal{
6. **void** eat(){System.out.println("eating fruits");}
7. }
9. **class** BabyDog **extends** Dog{
10. **void** eat(){System.out.println("drinking milk");}
12. **public** **static** **void** main(String args[]){
13. Animal a1,a2,a3;
14. a1=**new** Animal();
15. a2=**new** Dog();
16. a3=**new** BabyDog();
18. a1.eat();
19. a2.eat();
20. a3.eat();
21. }
22. }

Output: eating

eating fruits

drinking Milk

**Try for Output**

1. **class** Animal{
2. **void** eat(){System.out.println("animal is eating...");}
3. }
5. **class** Dog **extends** Animal{
6. **void** eat(){System.out.println("dog is eating...");}
7. }
9. **class** BabyDog **extends** Dog{
10. **public** **static** **void** main(String args[]){
11. Animal a=**new** BabyDog();
12. a.eat();
13. }}

Output: Dog is eating

Since, BabyDog is not overriding the eat() method, so eat() method of Dog class is invoked.