HowToDoInJava

Guide to Polymorphism

Last Updated: January 30, 2022 By: Lokesh Gupta Java Object Oriented Programming Java OOP, Polymorphism

In simple words, polymorphism is the ability by which, we **can create functions or reference variables which behaves differently in different programmatic context**.

Polymorphism is one of the major building blocks of object oriented programming along with inheritance, abstraction and encapsulation.

"Subtype polymorphism, often referred to as simply polymorphism in the context of object-oriented programming, is the ability to create a variable, a function, or an object that has more than one form." – Wikipedia

> Recommended Reading : Java Abstraction vs Encapsulation.

Polymorphism in Java

An example of polymorphism is referring the instance of subclass, with reference variable of super-class. e.g.

```
Object o = new Object(); //o can hold the reference of any subtype
Object o = new String();
Object o = new Integer();
```

Here, `String` is subclass of `Object` class. This is basic *example of polymorphism*.

In java language, polymorphism is essentially considered into two versions.

1. Compile time polymorphism (static binding or method overloading)
2. Runtime polymorphism (dynamic binding or method overriding)

Compile Time Polymorphism (static binding or method overloading)

As the meaning is implicit, this is used to write the program in such a way, that **flow of control is decided in compile time** itself. It is *achieved using method overloading*.

In method overloading, an object can have two or more methods with same name, BUT, with their method parameters different. These parameters may be different on two bases:

Parameters type

Type of method parameters can be different. e.g. `java.util.Math.max()` function comes with following versions:

```
public static double Math.max(double a, double b){..}
public static float Math.max(float a, float b){..}
public static int Math.max(int a, int b){..}
public static long Math.max(long a, long b){..}
```

The actual method to be called is decided on compile time based on parameters passed to function in program.

Parameters count

Functions accepting different number of parameters. e.g. in employee management application, a factory can have these methods:

```
EmployeeFactory.create(String firstName, String lastName){...}
EmployeeFactory.create(Integer id, String firstName, String lastName){...}
```

Both methods have same name "create" but actual method invoked will be based on parameters passed in program.

Runtime Polymorphism (dynamic binding or method overriding)

Runtime polymorphism is essentially referred as *method overriding*. Method overriding is a feature which you get when you implement inheritance in your program.

A simple example can be from real world e.g. animals. An application can have `Animal` class, and its specialized sub classes like `Cat` and `Dog`. These subclasses will override the default behavior provided by `Animal` class + some of its own specific behavior.

```
public class Animal {
public void makeNoise()
{
System.out.println("Some sound");
}
}

class Dog extends Animal{
public void makeNoise()
{
System.out.println("Bark");
}
}

class Cat extends Animal{
public void makeNoise()
{
System.out.println("Meawoo");
}
}
```

Now which `makeNoise()` method will be called, depends on type of actual instance created on runtime e.g.

```
public class Demo
{
public static void main(String[] args) {
Animal a1 = new Cat();
a1.makeNoise(); //Prints Meowoo

Animal a2 = new Dog();
a2.makeNoise(); //Prints Bark
}
}
```

> Here, it is important to understand the these divisions are specific to java. In context to software engineering, there are other form of polymorphisms also applicable to different languages, but for java, these are mainly considered.

Important points

1. Polymorphism is the ability to create a variable, a function, or an object that has more than one form.
2. In java, polymorphism is divided into two parts : method overloading and method overriding.
3. Some may argue that method overloading is not polymorphism. Then what does the term compile time "polymorphism" means??
4. Another term operator overloading is also there, e.g. "+" operator can be used to add two integers as well as concat two sub-strings. Well, this is the only available support for operator overloading in java, and you can not have your own custom defined operator overloading in java.

Happy Learning !!

Recommended Reading:

1. Guide to Abstraction
2. Guide to Inheritance
3. Interface vs Abstract Class in Java
4. Encapsulation vs Abstraction in Java
5. Overloading vs Overriding in Java
6. Object Oriented Programming

7. Java Access Modifiers
8. Association, Aggregation and Composition
9. Constructors in Java
0. Java Instance Initializer Blocks

42 thoughts on “Guide to Polymorphism”

OluwaSegun

March 15, 2020 at 7:29 pm

Can we have a program that shows Polymorphism,Encapsulation,Abstraction and Inheritance all together

Reply

Ekta

January 14, 2019 at 4:04 pm

```
class Animal {
  int id=10;
    public void makeNoise()
    {
        System.out.println("Some sound");
    }
}

class Dog extends Animal{
  int id=20;
    public void makeNoise()
    {
        System.out.println("Bark");
    }
}

class Cat extends Animal{
  int id=30;
    public void makeNoise()
    {
        System.out.println("Meawoo");
    }
}
public class OverridingClassConcepts {

  public static void main(String[] args) {
    // TODO Auto-generated method stub
    Animal a1 = new Cat();
        a1.makeNoise(); //Prints Meowoo
        System.out.println(a1.id); // prints 10
// reference type is Animal  and object Cat class so its called Cat class method but why its calling An
// why its printing Animal class variable and Catclass method.

        Animal a2 = new Dog();
        a2.makeNoise(); //Prints Bark
        System.out.println(a2.id); // print 10

//reference type is Animal  and object Dog class so its called Dog class method but why its calling Ani
// why its printing Animal class variable and Dog class method.

  }

}
```

I understand its not completely related to polymorphism concept but please solve my problem

Reply

peng.w

May 28, 2019 at 11:01 am

I'm Chinese, polymorphism concept only use method.

Reply

peng.w

May 28, 2019 at 11:04 am

The concept of polymorphism can only be applied to methods

Reply

Swapnesh Gupta

August 15, 2019 at 11:25 am

I think you have not gone through the Inheritance concept properly. If you go through the point 3-Accessing inherited parent class members.

"java fields cannot be overridden". Having same name field will hide the field from parent class – while accessing via child class. Attribute accessed will be decided based on the class of reference type.

But in the case of methods access its opposite to field access. Method access uses the type of actual object created in Run Time.

Reply

arun singh

July 11, 2018 at 4:13 am

Hi Lokesh, May I know the following code has static binding or dynamic binding? Thanks!

class A{
void method1(){}
void method1(int a){}
}

```
public class demo extends A{
public static void main(String[] args) {
A obj=new demo();
obj.method1(3);
}
}
```

Reply

pranit

January 16, 2018 at 11:17 am

Very informative and excellent article you have shared here on the concept of 'polymorphism'.
Polymorphism is a very important concept in object oriented programming which enables to change the behavior of the applications in the run time based on the object on which the invocation happens.
Thanks for sharing!

Reply

Priya

October 17, 2017 at 11:21 am

The most common use of polymorphism in OOP happens when a father or mother category referrals is used to make reference to a child category item. Many Thanks for sharing this article.

Reply

Abhinay

August 9, 2016 at 3:12 pm

Hi,

Thanks for the post, I understood the concept of run time polymorphism, but my doubt is why do we need create the Super class reference for the sub class object as below:
Animal c = new Cat();
or
Animal d = new Dog();

What is the purpose of creating this and in which scenario do we use this.
we dont we directly create below
Cat c = new Cat();
Dog d = new Dog();

Reply

namasivayam G

August 9, 2016 at 8:46 pm

hi abhinay,
I hope this code will clarify your doubt.(read comments)

```
class Animal {
public void move(){
System.out.println("Animals can move");
}
}

class Dog extends Animal {

public void move() {
System.out.println("Dogs can walk and run");
}
}

public class TestDog {

public static void main(String args[]) {
Animal a = new Animal(); // Animal reference and object
Animal b = new Dog(); // Animal reference but Dog object

a.move();//output: Animals can move

b.move();//output:Dogs can walk and run
}
}
```

Reply

Prasad Pasupuleti

September 29, 2016 at 7:42 pm

Parent class reference type can hold any type of subclass return types.

If we don't know the what Subclass will return at run time then we can create reference type like this: Parent p = new Child();

For example,

String s = new String(); // "s" can hold only strings
Object o = new String(); // "o" can hold any return type from String class

Hope you understand.

Reply

Zahed

May 30, 2016 at 5:32 pm

```
package firstProject;

public class Animal {
public void move(){
System.out.println("Animal can move");
}

}
public class Dog extends Animal {
public void move(){
System.out.println("Dogs can walk and run");
}
public void bark(){
System.out.println("Dog barks");
}
}
public class Main1 {

public static void main(String[] args) {
Animal a=new Animal();
Dog b=new Dog();
a.move();
b.move();
b.bark();

}
```

}

Reply

Zahed

May 30, 2016 at 5:14 pm

I could not understand your code.Why output is

Instance Method
Type Method2
Can you explain it.

Reply

Lokesh Gupta

May 30, 2016 at 6:35 pm

Which program??

Reply

Hridya

April 18, 2016 at 5:38 pm

Can you plz explain why “number arg method invoked” is printed

```
class A{
  void sum(Number a){System.out.println("number arg method invoked");}

}

class B extends A{

  void sum(Integer a){System.out.println("integer arg method invoked");}

  public static void main(String args[]){

  A obj3=new B();
  obj3.sum(10);
  }

}
```

Reply

yogesh

August 23, 2016 at 10:08 pm

Hi,Hridya, Actually for method overriding you need to write method exactly same as super class …Here in your case you changed the argument type in method arguments.So this is not method Overriding.

Reply

Pramod

February 18, 2015 at 7:15 am

Very nice explanation… Thanks Lokesh.

Reply

vixir

January 17, 2015 at 7:34 pm

Hi Lokesh,
I was going through tutorials point and I found this statement –
"The type of the reference variable would determine the methods that it can invoke on the object."
I have a feel that this statement is wrong. Can you tell me.
My logic :
As per your example –
Animal a1 = new Cat();
as per above line of code is this sentence correct: a1 is the variable that holds reference of type Animal.
or a1 is a variable that refers to instance of type Cat.
How would you say that statement?
Its actually instance that determines which method to invoke. right?

Reply

Lokesh Gupta

January 18, 2015 at 9:57 am

Actually both statement are correct. Let's take an example:

```
package com.howtodoinjava.nio2.examples;

public class Type
{
  public void method()
  {
    System.out.println(&amp;quot;Type Method&amp;quot;);
  }

  public void method2()
  {
    System.out.println(&amp;quot;Type Method2&amp;quot;);
  }

  public static void main(String[] args)
  {
    Type t = new Instance();
    t.method();   //Method called from &amp;quot;Instance&amp;quot; becoz it's available in Instance and Type
    //t.method1();  //Method 1 is not visible here
    t.method2();  //Method 2 called from &amp;quot;Type&amp;quot; becoz not available in Instance; BUT availab
  }
}

class Instance extends Type
{
  public void method()
  {
    System.out.println(&amp;quot;Instance Method&amp;quot;);
  }

  public void method1()
  {
    System.out.println(&amp;quot;Instance Method1&amp;quot;);
  }
}

Output:

Instance Method
Type Method2
```

The type of the reference variable would determine the methods that it "CAN" invoke on the object. [TRUE] Its actually instance that determines which method to invoke. right? [TRUE if method is present in type and instance both].

Infact it may be say like this : "a1 is reference variable of type Animal which points to instance of type Cat".

Which method can be invoked depends on "reference type"; but method will be invoked from "actual type" if available there otherwise will be called from "reference type" itself.

And it is quite obvious as well because compiler can not determine the actual type in compile time (e.g. using dependency injection). Actual instance is known only at runtime. To generate bytecode, Compiler must ensure at compile time that method (it's argument and return types) are valid; and that can be only correctly verified only if information is taken from reference type only.

Reply

vixir

February 27, 2015 at 7:19 pm

Thanks sir

Reply

Parth

July 2, 2015 at 7:34 am

Hi,
I don't understand why t.method1() is not visible in the main method. Also, to ensure that we are able to run t.method1() what must we do?

Reply

Lokesh Gupta

July 2, 2015 at 8:06 am

1) Because superclass can not see the data/method defined by subclass.
2) We can not do anything.

Reply

HIMANSU NAYAK

May 4, 2014 at 1:07 am

Hi Pankaj,
To add few things
change the return type or access specifier/modifier in method overloading is not allowed and leads to compiler time error i hope but the same works in method overriding some time called as covariant return type.

overriding with different return type and access specifer

Object Class
protected native Object clone() throws CloneNotSupportedException{
}

User Class
public User clone(){
}

Reply

Manish
February 25, 2014 at 8:12 am

Lokesh,
Wondering why someone will create Animal class as per your example. if Amimal class has no use at all. better it should be interface then a Class to force concreat class to implement

or the same method of base class gives different output would have been better example.

Reply

Lokesh Gupta
February 25, 2014 at 10:02 am

I respectfully disagree. As a general rule, you should create classes for nouns and interfaces for verbs i.e. actions. So, actually Animal should be a class with default behavior which all animals will have e.g. that all can move, make noise, eat and many things like that. There is no need to override every behavior for all animals. It will be code duplication. Each animal can override it's specific behavior only. With interface it is not possible.

Interface which can make sense are Walkable, Swimmable or Fyable which a group of animals can implement based on needs. But still they waill have all default behavior from Animal.class.

Reply

Praveenkumar
February 19, 2014 at 9:06 am

Hi Lokesh,
Why JVM call always Overloaded method if i pass null ?
class MyClass {

public void test(Object o) {
System.out.println("From() method ");
}
public void test(String o) {
System.out.println("From Overloaded test() method ");
}
}

public class Demo {
public static void main(String[] args) {
MyClass m = new MyClass();
m.test(null);
}
}

O/P= From Overloaded test() method

Reply

Lokesh Gupta
February 19, 2014 at 9:50 am

While resolving the overloaded method to call, JVM tries to find method with most specialized type. NULL can be referred from spring as well as object type variables. But, String is more specialized. So, JVM chooses to call "public void test(String o)".

I will recommend reading following thread: https://stackoverflow.com/questions/13041042/using-null-in-overloaded-methods-in-java

Reply

HIMANSU NAYAK
May 4, 2014 at 12:47 am

Hi Praveen,
More fun happens when you also add this

public void test(Integer o) {
System.out.println("From() method ");
}

Reply

Abhijit
January 28, 2014 at 6:17 am

nice explanation……..

Reply

Bharat Shivram
November 25, 2013 at 9:15 pm

An explanation of the following code would be highly appreciated:

public Car {
public static void m1()[
System.out.println("a");
}
public void m2()[
System.out.println("b");
}
}

class Mini extends Car {
public static void m1() {
System.out.println("c");
}
public void m2()[
System.out.println("d");
}
public static void main(String args[]) {
Car c = new Mini();
c.m1();
c.m2();
}
}

Why does this print “a d”? Also what would happen if the method m1() is not defined in Car class?

Reply

Lokesh Gupta

November 25, 2013 at 10:49 pm

Code was not compiling. It changed it to below:

```
public class Car {
public static void m1() {
System.out.println("a");
}

public void m2() {
System.out.println("b");
}

public static void main(String args[]) {
Mini c = new Mini();
c.m1();
c.m2();
}
}

class Mini extends Car {
public static void m1() {
System.out.println("c");
}

public void m2() {
System.out.println("d");
}
}
```

Output is: c d for obvious reasons.

Reply

Saurabh Moghe

December 6, 2013 at 12:27 pm

@Lokesh code of bharat works if you write main method in public class Car from there call main method of mini which is non public.
@Bharat static method are not available for method overriding.It is part of class rather than object. But you can always overload Static method.

Reply

Shwetha
February 26, 2014 at 2:50 pm

can u please explain more about the above points ?

if you write main method in public class Car from there call main method of mini which is non public.????

always it should call the method which object is created right ?? but why here the parent method object is called..???

Reply

Shwetha
February 26, 2014 at 2:56 pm

Consider the below example

public class Parent {

public static void method(){
System.out.println("parent method is called");
}
/**
* @param args
*/
public static void main(String[] args) {

Parent p=new Child();
p.method();

}

}

public class Child extends Parent {

public static void method(){
System.out.println(“parent method is called”);
}

}

Output is : parent method is called.

According to above explanations, it should call Child method right..?? why is it calling parent method..??

Shwetha
February 26, 2014 at 3:01 pm

Sorry , i donno how to edit the comment in the above code…

a small correction…in the above code..

public class Child extends Parent {

public static void method(){
System.out.println(“child method is called”);
}

}

Lokesh Gupta
February 26, 2014 at 3:48 pm

The reason is that static methods are always called for reference type, and not from actual instance. If you look at eclipse warning (“a yellow triangle”), it also hints “The static method method() from the type Parent should be accessed in a static way”.

Also Saurabh pointed out that “static method are not available for method overriding.It is part of class rather than object. But you can always overload Static method.”

Actually there is no difference between below both statements:

1) p.method();
2) Parent.method(); //Both are same

```
public class Parent
{
public static void method()
{
System.out.println("parent method is called");
}

public void method1()
{
System.out.println("parent method1 is called");
}

public static void main(String[] args)
{
Parent p = new Child();

p.method();
Parent.method();

p.method1(); //A call to non-static method will call method from actual instance
}
}

public class Child extends Parent
{
public static void method()
{
System.out.println("child method is called");
}

public void method1()
{
System.out.println("child method1 is called");
}
}
```

Let me know if there are more confusions.

amar

May 31, 2014 at 9:50 am

dont know but i like your question……

HIMANSU NAYAK

May 4, 2014 at 12:42 am

Hi,
Overriding a static method is some time also refer as masking or hiding

Reply

Nitya

August 20, 2013 at 11:45 pm

Why method overloading happens at compile time only not at run time?please reply

Reply

Lokesh Gupta

August 21, 2013 at 12:00 am

At compile time, JVM generates the bytecode for a program i.e. your class. This bytecode is essentially a sequence of operations. These operations are in low level languages and are executed in sequence.
In case overloading, JVM is able to and it determines which method call it needs to make for a certain operation, so it generates the bytecode equivalent to method call in compile time itself.
In case of overriding, it is unable to determine this so it generates code with certain conditional keywords which determine the actual method call when bytecode runs on machine.

Reply

Praveenkumar

February 17, 2014 at 11:17 am

I Guess Nitya's question is "Why method overloading happens at run time only why not at compile time "? Compiler dose not know which method is called at compilation time, because Objects are created at the time of Running program. So JVM knows based on the object creation, which method has to call at the time of running the program..

Reply

Lokesh Gupta

February 17, 2014 at 12:13 pm

NO. Nitya is right. Overloading is resolved at compile time itself. Overriding is resolved on runtime because actual instance is resolved in runtime time only. Remember, in java super class reference can point to instances of child classes also. Actual instance is resolved at runtime only.

Reply

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...