What is the difference between dynamic and static polymorphism in Java?

Asked 6 years, 5 months ago Active 1 month ago Viewed 215k times

114

76

Can anyone provide a simple example that explains the difference between **Dynamic** and **Static** polymorphism in Java?

java oop polymorphism overloading overriding

edited Mar 13 '15 at 23:16

ROMANIA_engineer
42.9k ● 23 ● 179 ● 160

asked Dec 26 '13 at 10:30

Prabhakar Manthena
1,863 ● 2 ● 13 ● 30

4 Overrides are sometimes referred to as "static polymorphism". This stretches things a bit, but that's what is going on. – dasblinkenlight Dec 26 '13 at 10:36

@dasblinkenlight thanks for the info. is there any example for that ?? – Prabhakar Manthena Dec 26 '13 at 10:39

Search for "method overloading" and "method overriding". – dasblinkenlight Dec 26 '13 at 10:42

5 I dont understand how overloading is polymorphism. Polymorphism is a concept of object. we should be able to show object B as object A. from below(answer) example you showd Dog as an Animal and hence it is polymorphism. But in overloading you are calling different method but with "same name". How can this be polymorphism. Hence "static binding" is the correct term to use but static polymorphism is not in case of overloading. – Punith Raj Oct 15 '14 at 16:04

@PunithRaj You're probably referring to Subtype polymorphism. There is another kind called Ad hoc which applies to overloading. – Kelvin Dec 28 '18 at 19:58

14 Answers

Active Oldest Votes

191

Polymorphism

1. Static binding/Compile-Time binding/Early binding/Method overloading.(in same class)

2. Dynamic binding/Run-Time binding/Late binding/Method overriding.(in different classes)

overloading example:

```
class Calculation {
  void sum(int a,int 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[]) {
    Calculation obj=new Calculation();
    obj.sum(10,10,10);  // 30
    obj.sum(20,20);     //40
  }
}
```

overriding example:

```
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
   }
}
```

edited Apr 14 '17 at 10:05

answered Dec 26 '13 at 10:40

KhAn SaAb

4,779 ● 5 ● 25 ● 46

5 I'm new to Java, so just curios what is the underlying concept between `Animal reference but Dog object`, why can't we use `Dog reference and dog object`? – pratyay May 18 '17 at 6:48

3 In the above example I tried to show the concept of polymorphism. we can create the reference and object of same class but we cant achieve method overriding. please go through with below post : stackoverflow.com/questions/12159601/… – KhAn SaAb May 19 '17 at 20:11

method overloading is Compile time polymorphism. is same way. is constructor overloading also Compile time polymorphism ? – Gaali Prabhakar May 15 '18 at 11:34

29

- Method overloading would be an example of static polymorphism
- whereas overriding would be an example of dynamic polymorphism.

  Because, in case of overloading, at compile time the compiler knows which method to link to the call. However, it is determined at runtime for dynamic polymorphism

edited Dec 26 '13 at 10:38
keyser
16.8k ● 16 ● 52 ● 89

answered Dec 26 '13 at 10:36
user1374
1,918 ● 2 ● 21 ● 36

17

Dynamic (run time) polymorphism is the polymorphism existed at run-time. Here, Java compiler does not understand which method is called at compilation time. Only JVM decides which method is called at run-time. Method overloading and method overriding using instance methods are the examples for dynamic polymorphism.

For example,

- Consider an application that serializes and de-serializes different types of documents.
- We can have ‘Document’ as the base class and different document type classes deriving from it. E.g. XMLDocument , WordDocument , etc.
- Document class will define ‘ Serialize() ’ and ‘ De-serialize() ’ methods as virtual and each derived class will implement these methods in its own way based on the actual contents of the documents.

- When different types of documents need to be serialized/de-serialized, the document objects will be referred by the ‘ Document’ class reference (or pointer) and when the ‘ Serialize() ’ or ‘ De-serialize() ’ method are called on it, appropriate versions of the virtual methods are called.

Static (compile time) polymorphism is the polymorphism exhibited at compile time. Here, Java compiler knows which method is called. Method overloading and method overriding using static methods; method overriding using private or final methods are examples for static polymorphism

For example,

- An employee object may have two print() methods one taking no arguments and one taking a prefix string to be displayed along with the employee data.
- Given these interfaces, when the print() method is called without any arguments, the compiler, looking at the function arguments knows which function is meant to be called and it generates the object code accordingly.

For more details please read "What is Polymorphism" (Google it).

edited Dec 26 '15 at 13:59

Rolf ツ

7,859 ●5 ●43 ●71

answered Dec 26 '13 at 10:41

rachana

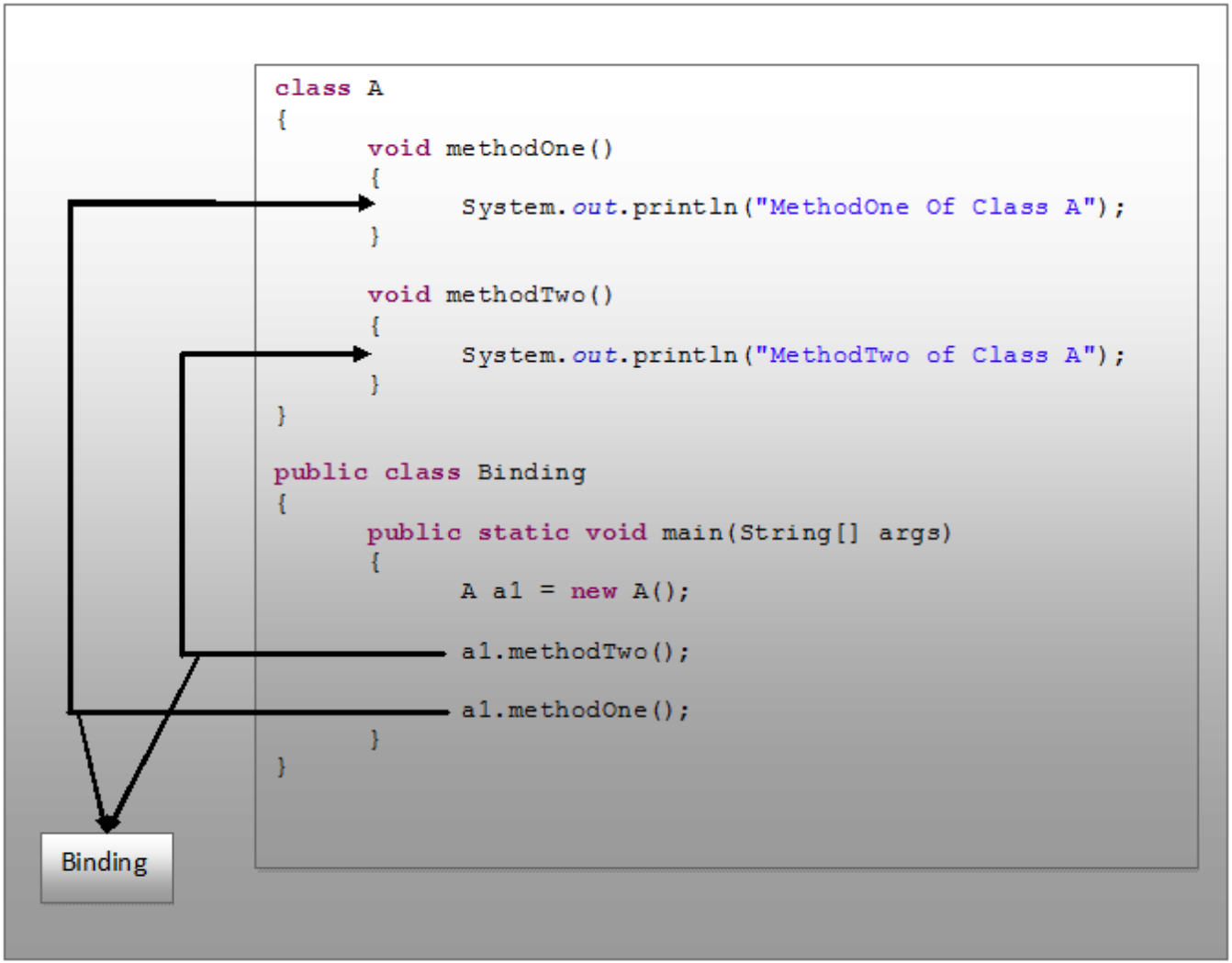
2,918 ●5 ●25 ●42

1 This answer full of errors: (1) Method overloading is not dynamic polymorphism. It is static polymorphism. (2) Static methods are never overridden, they are hidden/shadowed. (3) Private methods are not "overridden". They are never inherited in the first place. – John Red Mar 8 '18 at 3:59

12

Binding refers to the link between method call and method definition.

This picture clearly shows what is binding.



In this picture, “a1.methodOne()” call is binding to corresponding methodOne() definition and “a1.methodTwo()” call is binding to corresponding methodTwo() definition.

For every method call there should be proper method definition. This is a rule in java. If compiler does not see the proper method definition for every method call, it throws error.

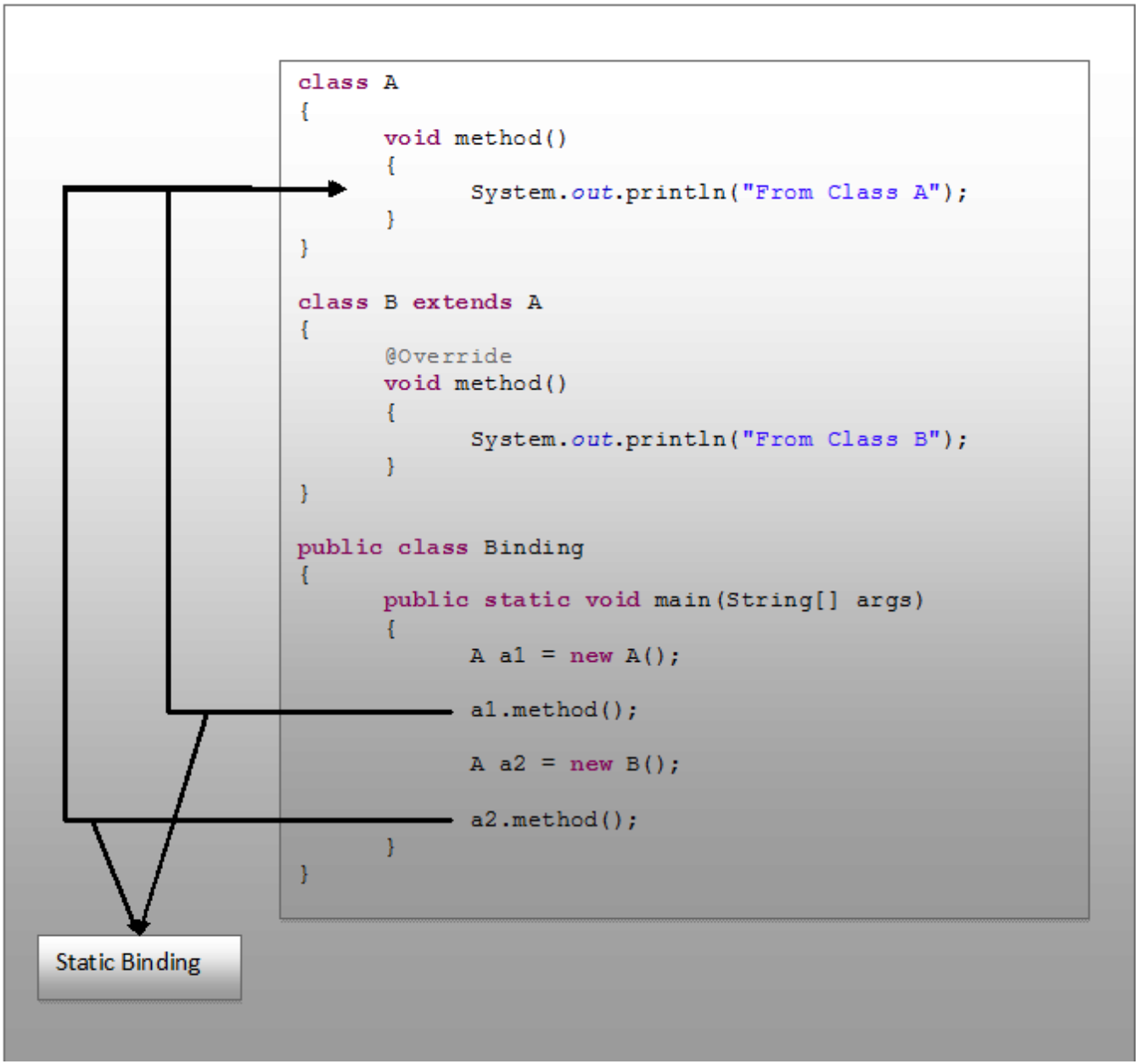
Now, come to static binding and dynamic binding in java.

Static Binding In Java :

> Static binding is a binding which happens during compilation. It is also called early binding because binding happens before a program actually runs

.

Static binding can be demonstrated like in the below picture.



In this picture, ‘a1’ is a reference variable of type Class A pointing to object of class A. ‘a2’ is also reference variable of type class A but pointing to object of Class B.

During compilation, while binding, compiler does not check the type of object to which a particular reference variable is pointing. It just checks the type of reference variable through which a method is called and checks whether there exist a method definition for it in that

type.

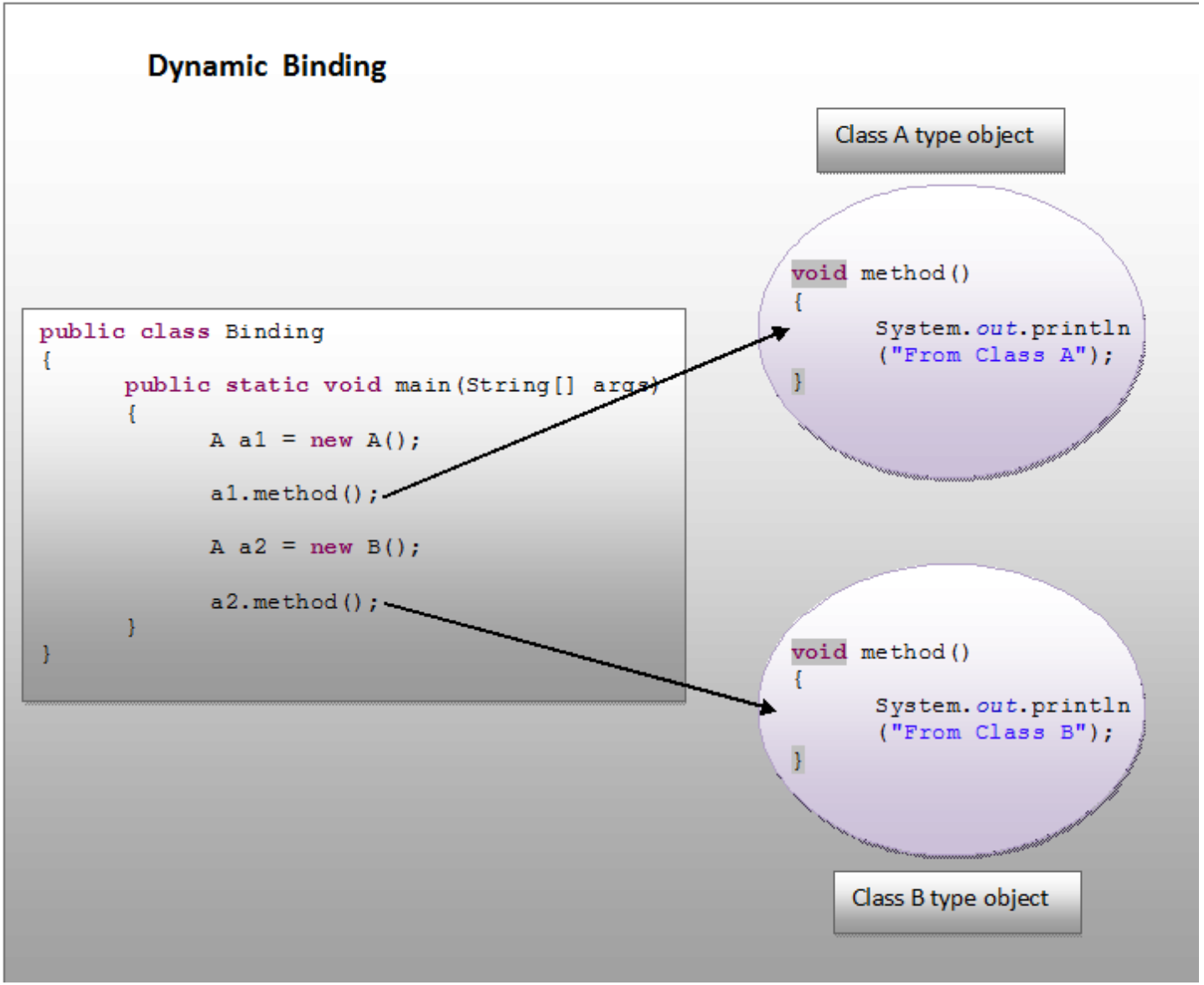
For example, for “a1.method()” method call in the above picture, compiler checks whether there exist method definition for method() in Class A. Because ‘a1′ is Class A type. Similarly, for “a2.method()” method call, it checks whether there exist method definition for method() in Class A. Because ‘a2′ is also Class A type. It does not check to which object, ‘a1’ and ‘a2’ are pointing. This type of binding is called static binding.

Dynamic Binding In Java :

> Dynamic binding is a binding which happens during run time. It is also called late binding because binding happens when program actually is running.

During run time actual objects are used for binding. For example, for “a1.method()” call in the above picture, method() of actual object to which ‘a1’ is pointing will be called. For “a2.method()” call, method() of actual object to which ‘a2’ is pointing will be called. This type of binding is called dynamic binding.

The dynamic binding of above example can be demonstrated like below.



Reference [static-binding-and-dynamic-binding-in-java]

edited Sep 22 '17 at 21:11

answered Sep 22 '17 at 19:42

Elsayed

1,260 ● 1 ● 15 ● 29

better than before. – AnBisw Sep 22 '17 at 21:17

7

Polymorphism: Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Dynamic Binding/Runtime Polymorphism :

Run time Polymorphism also known as method overriding. In this Mechanism by which a call to an overridden function is resolved at a Run-Time.

```
public class DynamicBindingTest {

    public static void main(String args[]) {
        Vehicle vehicle = new Car(); //here Type is vehicle but object will be Car
        vehicle.start();       //Car's start called because start() is overridden method
    }
}

class Vehicle {

    public void start() {
        System.out.println("Inside start method of Vehicle");
    }
}

class Car extends Vehicle {

    @Override
    public void start() {
        System.out.println("Inside start method of Car");
    }
}
```

Output:

Inside start method of Car

Static Binding /compile-time polymorphism:

Which method is to be called is decided at compile-time only.

```
public class StaticBindingTest {
```

```
    public static void main(String args[])  {
       Collection c = new HashSet();
       StaticBindingTest et = new StaticBindingTest();
       et.sort(c);

    }

    //overloaded method takes Collection argument
    public Collection sort(Collection c){
        System.out.println("Inside Collection sort method");
        return c;
    }


   //another overloaded method which takes HashSet argument which is sub class
    public Collection sort(HashSet hs){
        System.out.println("Inside HashSet sort method");
        return hs;
    }

}
```

Output: Inside Collection sort metho

answered Apr 21 '14 at 6:08

loknath

1,244 ● 14 ● 24

7

In simple terms :

Static polymorphism : Same method name is ***overloaded*** with different type or number of parameters in ***same class*** (different signature). Targeted method call is resolved at compile time.

Dynamic polymorphism: Same method is ***overridden*** with same signature in ***different classes***. Type of object on which method is being invoked is not known at compile time but will be decided at run time.

Generally overloading won't be considered as polymorphism.

From java tutorial page :

> Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class

edited Sep 19 '17 at 10:14

Anatoly Shamov
2,051 ● 1 ● 9 ● 17

answered Dec 27 '15 at 5:55

Ravindra babu
37.7k ● 6 ● 194 ● 171

`Generally overloading won't be considered as polymorphism.` can you please elaborate on this point. – prime Mar 19 '18 at 11:19

1 Dynamic binding and overriding is striking point for polymorphism – Ravindra babu Mar 19 '18 at 12:56

7

method overloading is an example of compile time/static polymorphism because method binding between method call and method definition happens at compile time and it depends on the reference of the class (reference created at compile time and goes to stack).

method overriding is an example of run time/dynamic polymorphism because method binding between method call and method definition happens at run time and it depends on the object of the class (object created at runtime and goes to the heap).

edited Nov 19 '18 at 10:06

Pang
8,051 ● 16 ● 71 ● 110

answered Dec 26 '13 at 11:00

Sujith PS
4,237 ● 3 ● 27 ● 56

*(object create at run time and goes to heap), it should run time – Meet Dec 26 '13 at 11:32

4

Method Overloading is known as **Static Polymorphism** and also Known as **Compile Time Polymorphism** or **Static Binding** because overloaded method calls get resolved at compile time by the compiler on the basis of the argument list and the reference on which we are calling the method.

And **Method Overriding** is known as **Dynamic Polymorphism** or simple **Polymorphism** or **Runtime Method Dispatch** or **Dynamic Binding** because overridden method call get resolved at runtime.

In order to understand why this is so let's take an example of `Mammal` and `Human` class

```
class Mammal {
    public void speak() { System.out.println("ohlllalalalalalaoaoaoa"); }
}

class Human extends Mammal {

    @Override
    public void speak() { System.out.println("Hello"); }
```

```
    public void speak(String language) {
        if (language.equals("Hindi")) System.out.println("Namaste");
        else System.out.println("Hello");
    }

}
```

I have included output as well as bytecode of in below lines of code

```
Mammal anyMammal = new Mammal();
anyMammal.speak();  // Output - ohlllalalalalaoaoaoa
// 10: invokevirtual #4 // Method
org/programming/mitra/exercises/OverridingInternalExample$Mammal.speak:()V

Mammal humanMammal = new Human();
humanMammal.speak(); // Output - Hello
// 23: invokevirtual #4 // Method
org/programming/mitra/exercises/OverridingInternalExample$Mammal.speak:()V

Human human = new Human();
human.speak(); // Output - Hello
// 36: invokevirtual #7 // Method
org/programming/mitra/exercises/OverridingInternalExample$Human.speak:()V

human.speak("Hindi"); // Output - Namaste
// 42: invokevirtual #9 // Method
org/programming/mitra/exercises/OverridingInternalExample$Human.speak:
(Ljava/lang/String;)V
```

And by looking at above code we can see that the bytecodes of humanMammal.speak() , human.speak() and human.speak("Hindi") are totally different because the compiler is able to differentiate between them based on the argument list and class reference. And this is why **Method Overloading** is known as **Static Polymorphism**.

But bytecode for anyMammal.speak() and humanMammal.speak() is same because according to compiler both methods are called on Mammal reference but the output for both method calls is different because at runtime JVM knows what object a reference is holding and JVM calls the method on the object and this is why Method Overriding is known as Dynamic Polymorphism.

So from above code and bytecode, it is clear that during compilation phase calling method is considered from the reference type. But at execution time method will be called from the object which the reference is holding.

If you want to know more about this you can read more on How Does JVM Handle Method Overloading and Overriding Internally.

edited Jan 18 '19 at 9:21 answered Jun 14 '17 at 17:53

Naresh Joshi
2,705 ● 21 ● 31

3

Static Polymorphism: is where the decision to resolve which method to accomplish, is determined during the compile time. Method Overloading could be an example of this.

Dynamic Polymorphism: is where the decision to choose which method to execute, is set during the run-time. Method Overriding could be an example of this.

edited Nov 16 '17 at 18:04

answered Oct 21 '17 at 17:19

Kulasangar
6,648 ● 2 ● 31 ● 61

2

Polymorphism refers to the ability of an object to behave differently for the same trigger.

Static polymorphism (Compile-time Polymorphism)

- Static Polymorphism decides which method to execute during compile time.
- Method Overloading is an example of static polymorphism, and it is requred to happens static polymorphism.
- Static Polymorphism achieved through static binding.
- Static Polymorphism happens in the same class.
- Object assignment is not required for static polymorphism.
- Inheritance not involved for static polymorphism.

Dynamic Polymorphism (Runtime Polymorphism)

- Dynamic Polymorphism decides which method to execute in runtime.
- Method Overriding is an example of dynamic polymorphism, and it is requred to happens dynamic polymorphism.
- Dynamic Polymorphism achieved through dynamic binding.
- Dynamic Polymorphism happens between different classes.
- It is required where a subclass object is assigned to super class object for dynamic polymorphism.
- Inheritance involved for dynamic polymorphism.

answered Apr 6 '18 at 11:24

Saveendra Ekanayake
2,206 ● 5 ● 24 ● 35

1

Compile time polymorphism(Static Binding/Early Binding): In static polymorphism, if we call a method in our code then which definition of that method is to be called actually is resolved at compile time only.

(or)

At compile time, Java knows which method to invoke by checking the method signatures. So, this is called compile-time polymorphism or static binding.

Dynamic Polymorphism(Late Binding/ Runtime Polymorphism): At run time, Java waits until runtime to determine which object is actually being pointed to by the reference. Method resolution was taken at runtime, due to that we call as run time polymorphism.

answered Jan 30 '18 at 9:18

Pavan Reddy
21 ● 1

1

Consider the code below:

```
public class X
{
    public void methodA() // Base class method
    {
        System.out.println ("hello, I'm methodA of class X");
    }
}

public class Y extends X
{
    public void methodA() // Derived Class method
    {
        System.out.println ("hello, I'm methodA of class Y");
    }
}
public class Z
{
public static void main (String args []) {

    //this takes input from the user during runtime
```

```
    System.out.println("Enter x or y");
    Scanner scanner = new Scanner(System.in);
    String value= scanner.nextLine();

    X obj1 = null;
    if(value.equals("x"))
        obj1 = new X(); // Reference and object X
    else if(value.equals("y"))
        obj2 = new Y(); // X reference but Y object
    else
        System.out.println("Invalid param value");

    obj1.methodA();
}
}
```

Now, looking at the code you can never tell which implementation of methodA() will be executed, Because it depends on what value the user gives during runtime. So, it is only decided during the runtime as to which method will be called. Hence, Runtime polymorphism.

edited Mar 25 '19 at 18:44

answered Mar 16 '19 at 8:43

user2048204

616 ● 4 ● 10 ● 24

0

Method overloading is a compile time polymorphism, let's take an example to understand the concept.

```
class Person                                        //person.java file
{
    public static void main ( String[] args )
    {
      Eat e = new Eat();
       e.eat(noodle);                              //line 6
    }

   void eat (Noodles n)      //Noodles is a object    line 8
   {

   }
   void eat ( Pizza p)           //Pizza is a object
  {

  }

}
```

In this example, Person has a eat method which represents that he can either eat Pizza or Noodles. That the method eat is overloaded when we compile this Person.java the compiler resolves the method call " e.eat(noodles) [which is at line 6] with the method definition specified in line 8 that is it method which takes noodles as parameter and the entire process is done by Compiler so it is Compile time Polymorphism. The process of replacement of the method call with method definition is called as binding, in this case, it is done by the compiler so it is called as early binding.

answered Jul 20 '17 at 13:36

Manoj Gururaj
1

0

Following on from Naresh's answer, dynamic polymorphism is only 'dynamic' in Java because of the presence of the virtual machine and its ability to interpret the code at run time rather than the code running natively.

In C++ it must be resolved at compile time if it is being compiled to a native binary using gcc, obviously; however, the runtime jump and thunk in the virtual table is still referred to as a 'lookup' or 'dynamic'. If C inherits B, and you declare `B* b = new C(); b->method1();` , b will be resolved by the compiler to point to a B object inside C (for a simple class inherits a class situation, the B object inside C and C will start at the same memory address so nothing is required to be done; it will be pointing at the vptr that they both use). If C inherits B and A, the virtual function table of the A object inside C entry for method1 will have a thunk which will offset the pointer to the start of the encapsulating C object and then pass it to the real A::method1() in the text segment which C has overridden. For `C* c = new C(); c->method1()` , c will be pointing to the outer C object already and the pointer will be passed to C::method1() in the text segment. Refer to: http://www.programmersought.com/article/2572545946/

In java, for `B b = new C(); b.method1();` , the virtual machine is able to dynamically check the type of the object paired with b and can pass the correct pointer and call the correct method. The extra step of the virtual machine eliminates the need for virtual function tables or the type being resolved at compile time, even when it could be known at compile time. It's just a different way of doing it which makes sense when a virtual machine is involved and code is only compiled to bytecode.

edited Apr 9 at 12:38

answered Feb 10 at 19:11

Lewis Kelsey
1,232 ● 8 ● 17

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.