Static vs Dynamic Binding in Java

[**2.1**](https://www.geeksforgeeks.org/easy/)

**Static Binding:**The binding which can be resolved at compile time by compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time .

**Why binding of static, final and private methods is always a static binding?**  
Static binding is better performance wise (no extra overhead is required). Compiler knows that all such methods **cannot be overridden** and will always be accessed by object of local class. Hence compiler doesn’t have any difficulty to determine object of class (local class for sure). That’s the reason binding for such methods is static.  
Let’s see by an example

|  |
| --- |
| public class NewClass  {      public static class superclass      {          static void print()          {              System.out.println("print in superclass.");          }      }      public static class subclass extends superclass      {          static void print()          {              System.out.println("print in subclass.");          }      }        public static void main(String[] args)      {          superclass A = new superclass();          superclass B = new subclass();          A.print();          B.print();      }  } |

Run on IDE

Before scrolling further down, Guess the output of the above program?

**Output**:

print in superclass.

print in superclass.

As you can see, in both cases print method of superclass is called. Lets see how this happens

* We have created one object of subclass and one object of superclass with the reference of the superclass.
* Since the print method of superclass is static, compiler knows that it will not be overridden in subclasses and hence compiler knows during compile time which print method to call and hence no ambiguity.

*As an exercise, reader can change the reference of object B to subclass and then check the output.*

**Dynamic Binding:**In Dynamic binding compiler doesn’t decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have same method . Let’s see by an example

|  |
| --- |
| public class NewClass  {      public static class superclass      {          void print()          {              System.out.println("print in superclass.");          }      }        public static class subclass extends superclass      {          @Override          void print()          {              System.out.println("print in subclass.");          }      }        public static void main(String[] args)      {          superclass A = new superclass();          superclass B = new subclass();          A.print();          B.print();      }  } |

Run on IDE

**Output:**

print in superclass.

print in subclass.

Here the output differs. But why? Let’s break down the code and understand it thoroughly.

* Methods are not static in this code.
* During compilation, the compiler has no idea as to which print has to be called since compiler goes only by referencing variable not by type of object and therefore the binding would be delayed to runtime and therefore the corresponding version of print will be called based on type on object.

**Important Points**

* private, final and static members (methods and variables) use static binding while for virtual methods (In Java methods are virtual by default) binding is done during run time based upon run time object.
* Static binding uses Type information for binding while Dynamic binding uses Objects to resolve binding.
* Overloaded methods are resolved (deciding which method to be called when there are multiple methods with same name) using static binding while overridden methods using dynamic binding, i.e, at run time.

This article is contributed by **Rishabh Mahrsee.** If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org.

See your article appearing on the GeeksforGeeks main page and help other Geeks.