Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Dynamic Method Dispatch/5b1e574b7becc0459deae14c

TUTORIAL

Java : Dynamic Method Dispatch

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism. As per the statement “a superclass reference variable can refer to a subclass object”. Java uses this fact to resolve calls to overridden methods at run time. Here is how. When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. When different types of objects are referred to, different versions of an overridden method will be called. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed. Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed. Here is an example that illustrates dynamic method dispatch:

Java

```
1  // Dynamic Method Dispatch
2  class A {
3    void callme() {
4      System.out.println("CodeQuotient - A");
5    }
6  }
7
8  class B extends A {
9    // override callme()
10   void callme() {
```

```
    System.out.println("CodeQuotient - B");
  }
}

class C extends A {
  // override callme()
  void callme() {
    System.out.println("CodeQuotient - C");
  }
}

class Main {
  public static void main(String args[]) {
    A a = new A(); // object of type A
    B b = new B(); // object of type B
    C c = new C(); // object of type C
    A r; // obtain a reference of type A
    r = a; // r refers to an A object
    r.callme(); // calls A's version of callme
    r = b; // r refers to a B object
    r.callme(); // calls B's version of callme
    r = c; // r refers to a C object
    r.callme(); // calls C's version of callme
  }
}

```

The output from the program is shown here:

```
CodeQuotient – A
CodeQuotient - B
CodeQuotient - C
```

This program creates one superclass called A and two subclasses of it, called B and C. Subclasses B and C override callme( ) declared in A. Inside the main( ) method, objects of type A, B, and C are declared. Also, a reference of type A, called r, is declared. The program then assigns a reference to each type of object to r and uses that reference to invoke callme( ). As the output shows, the version of callme( ) executed is determined by the type of object being referred

to at the time of the call. Had it been determined by the type of the reference variable, r, you would see three calls to A's callme( ) method.