Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Final and static in Java class/5b18d9087becc0459de9c5cd

TUTORIAL

Final and static in Java class

Topics

Final in Java

Final keyword can be used in java with three options, It can be used with fields, methods or classes.

- final with a variable makes the variable constant.
- final with a method prevents method overriding.
- final with a class makes the class non-inheritable.

Let us consider the final variables here, rest will be explained later at appropriate places.

Final Fields

You can define an instance field as final. Such a field must be initialized when the object is constructed. That is, it must be guaranteed that the field value has been set after the end of every constructor. Afterwards, the field may not be modified again. For example, the name field of the Employee class may be declared as final because it never changes after the object is constructed—there is no setName method.

```
class Employee
{
        . . .
```

```
        private final String name;
}
```

The final modifier is particularly useful for fields whose type is primitive or an immutable class.

```
// a final static variable PI
static final double PI = 3.141592653589793;
```

The only difference between a normal variable and a final variable is that we can re-assign value to a normal variable but we cannot change the value of a final variable once assigned. Hence final variables must be used only for the values that we want to remain constant throughout the execution of program.

Java

```
import java.util.Scanner;
// Other imports go here
class Main
{
  static final double PI = 3.141592653589793;
  public static void main(String[] args)
  {
    System.out.println(PI);
    //PI=3.14; //will result in error
  }
}
```

static in Java

static keyword can also be used for many purposes in Java.

static fields: If a field is defined as static then it is associated with the class itself not with individual objects. It means all objects will share the same copy of that variable, so all have the common value

for that field. Whereas each object has its own copy for its non-static variables. In the following Java program, static variable *count* is used to count the number of objects created: -

```java
import java.util.Scanner;
// Other imports go here
class Main{
  static int count = 0;
  Main()
  {
    count++;
  }
  public static void main(String[] args)
  {
    Main t1 = new Main();
    System.out.println("Total " + count + " objects created of Main class");
    Main t2 = new Main();
    System.out.println("Total " + count + " objects created of Main class");
    Main t3 = new Main();
    System.out.println("Total " + count + " objects created of Main class");
  }
}
```

static member functions: We can make a function as static. Methods declared as static are class members and have following restrictions:

- They can only call other static methods. For example, the following program fails in compilation. fun() is non-static and it is called in static main()

```java
class Main
{
  int fun()
```

```
  {
    return 20;
  }
  public static void main(String args[])
  {
    System.out.println(fun());
  }
}
```

- They must only access static data.
- They cannot access this or super . For example, the following program fails in compilation.

static data members and static methods can be accessed without creating an object. They can be accessed using class name. For example, in the following program, static data member count and static method fun() are accessed without any object.

Java

```
class Dummy
{
  static int count = 1;
  public static void funDummy()
  {
    System.out.println("Static funDummy() called.");
  }
}

class Main
{
  public static void main(String arr[])
  {
    Dummy d1=new Dummy();
    System.out.println("Test.count = " + Dummy.count);
    Dummy.funDummy();
  }
}
```