final class

Article- educba.com

final class in java

- ↳ can extend other classes; can be subclass but not superclass
- If we instantiate any final class, then it becomes created in the pool area, objects created in the pool have a special feature of immutability

Immutable java classes - String, Integer, Double

- final class ⇒ final variables & final methods implicitly

Singleton class

Singleton class

App.java

```
public class App{

  private static App app;   static
                            reference

  private App()
  {

  }

public static App getInstance()
{ if(null == app)
     {
     app= new App();
     }
  return app;
}
}
```

~~Until~~ If getInstance is called,
only then object will be created

Main.java

```
public class Main{
p.s.v.m(-){

  App app= App.getInstance();
  App app1= App.getInstance();
}
```

Static variables - one copy throughout program
- can only touch static variables & methods

private constructor

*private constructor - Object can be created only inside the class.

```
public class Main
{   private Main(){

      }

   p.s.v.m(){
      Main m = new Main();

   }
```

~~return this~~

Main m - creating reference variable

Main m = new Main(); - this will create the object.

this keyword - used as a reference to the current object

return this; // when you return this from a method, current object will be returned

method's return type will be class' name

~~Codechefa playing with strings~~

Static variables are stored in static memory.

Static class

DATE / /

Static class - nested class
- static member of outer class
- may be instantiated without instantiating outer class
- static class can ~~only~~ access only the static members of ~~the~~ outer class.

```
class Outerclass{
    private static ~~msg~~ String msg = "GeeksforGeeks";
    public static class NestedStaticClass
    {
        -- we can access msg from here, since
           it is static member of outer class.
    }
}
class GFG{
    p.s.v.m (String[] args)
    {
        Outerclass.NestedStaticClass printer
                = new Outerclass.NestedStaticClass();
    }
}
```

Method chaining

Article →
* GeeksforGeeks - Method chaining in java

Method chaining

PAGE NO.
DATE / /

Constructor returns object reference

```
class A {
  ~~A() {System.out.print(" ");}~~
  int setint(int a)
  { - __
  }
  void display ()
  {
   - __
  }
}
public class Main
{
   p.s.v.m( - - ){
   new A(). setint(10).display();
}
```

error

because, setint(10) is returning an Integer. So, next method display() cannot be called on the basis of Integer.

So, we'll have to modify setint() method

```
public A setint(int a)
{
     this.a = a;
     return this;
}
```

now we can use,

`new A().setint(10).display();`

Replacement for calling method on object one after another

If you want to use →

`new A(). display().setint(10);`

then, modify display().

```
  A display ()
{
  - __
  return this;
}
```

class Class , Classname.class

class with name Class in java.lang package. Instances of the class Class represent classes and interfaces in a running Java application.
• class after a class name. references the Class Object that represents the given class

```
A a = new A();
Class c = A.class;  // No error
// error -> Class c = a.class
```

nested class

A nested class can be public, private, package private or protected as a member of the outer class. The outer java classes can access inner class private or protected members.

~~nextLine() after nextInt() problem~~

• If you call nextLine() after nextInt(), nextLine() will read new line character instead of the actual data. So, you'll have to add another nextLine() after the nextInt(). nextLine() doesn't leave behind a new line character

```
int x = Integer.parseInt(sc.nextLine());
```

AnyScanner

PAGE No.
DATE / /

```
int a = sc.nextInt(); // 5
int b = sc.nextInt(); // 6

System.out.println(" " + a + b); // 56
System.out.println(a+b); // 13
```

*

Anonymous Inner Classes

(Youtube - Coding with John)

this is of type anonymous subclass of Animal

```
Animal bigfoot = new Animal(){
    public void makeNoise(){
            s.o.ut(_)
    }
};
```

← This is anonymous class

. separate class for particular object.

```
bigfoot.makeNoise();
```

Runnable interface

object of class type that doesn't have name

```
Runnable myAnonymousRunnable = new Runnable(){
        public void run(){
                System.out.println(_ __)
        }
};

myAnonymousRunnable.run();
```