Garbage Collection: " Garbage collection is a process of reacquiring the heap space by destroying all the unused and unreferenced objects."

1) Garbage collection is done by the garbage collector.
2) Garbage Collector is a demon thread (It is a low priority thread)
3) Garbage Collector is going to use **M**ark and **S**weep algorithm, to destroy all the unused or unreferenced Objects
4) Garbage Collector is going to **mark** all the **live objects** and destroys all unmarked objects.
5) Garbage Collector will be working in the background of every Java program, no need to call it manually but if we want to call it manually also we can do that by using pre-defined classes. [**System Class** & **Runtime Class**]

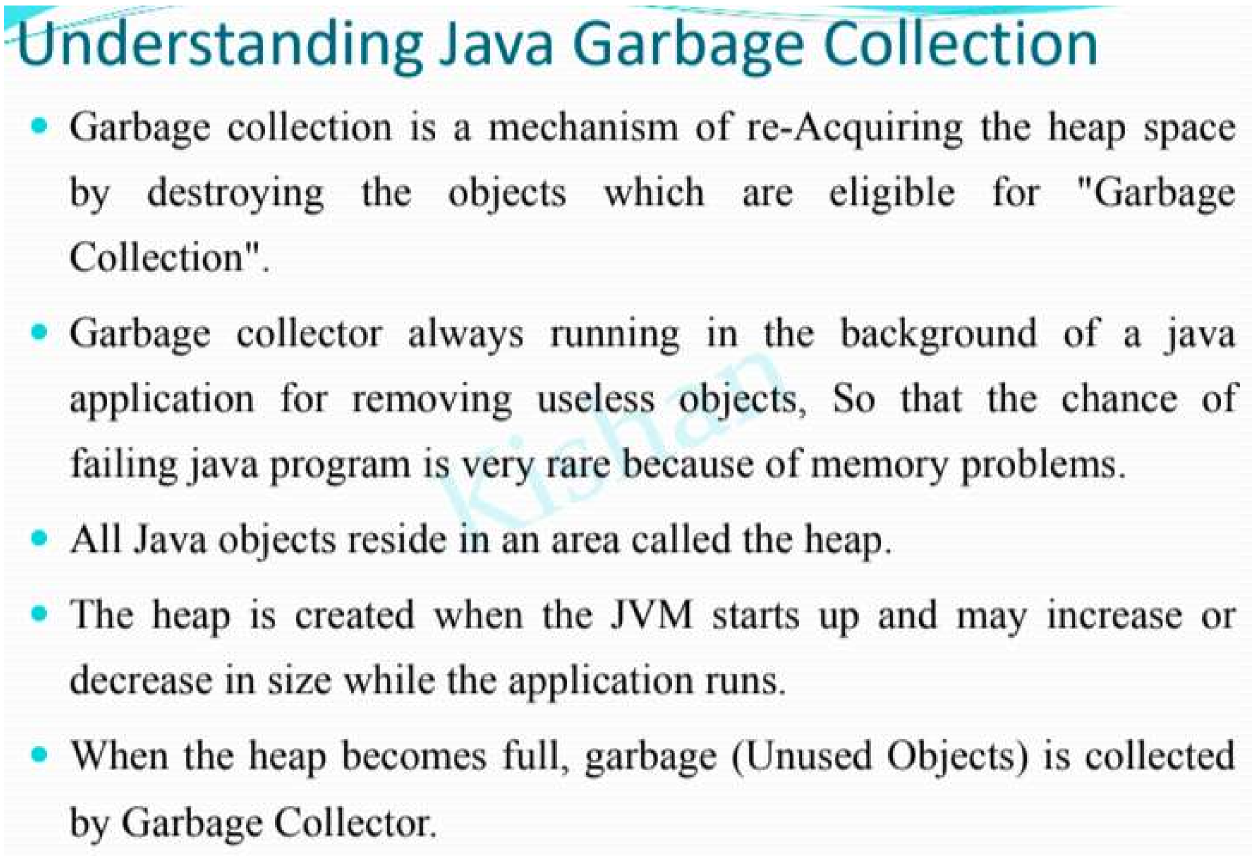
Q) When an Object will be eligible for destruction?

A) An object will be eligible for destruction in any of the below mentioned scenarios

- By Re-assigning the reference variable
- By Nullifying the reference variable
- All Objects created inside method

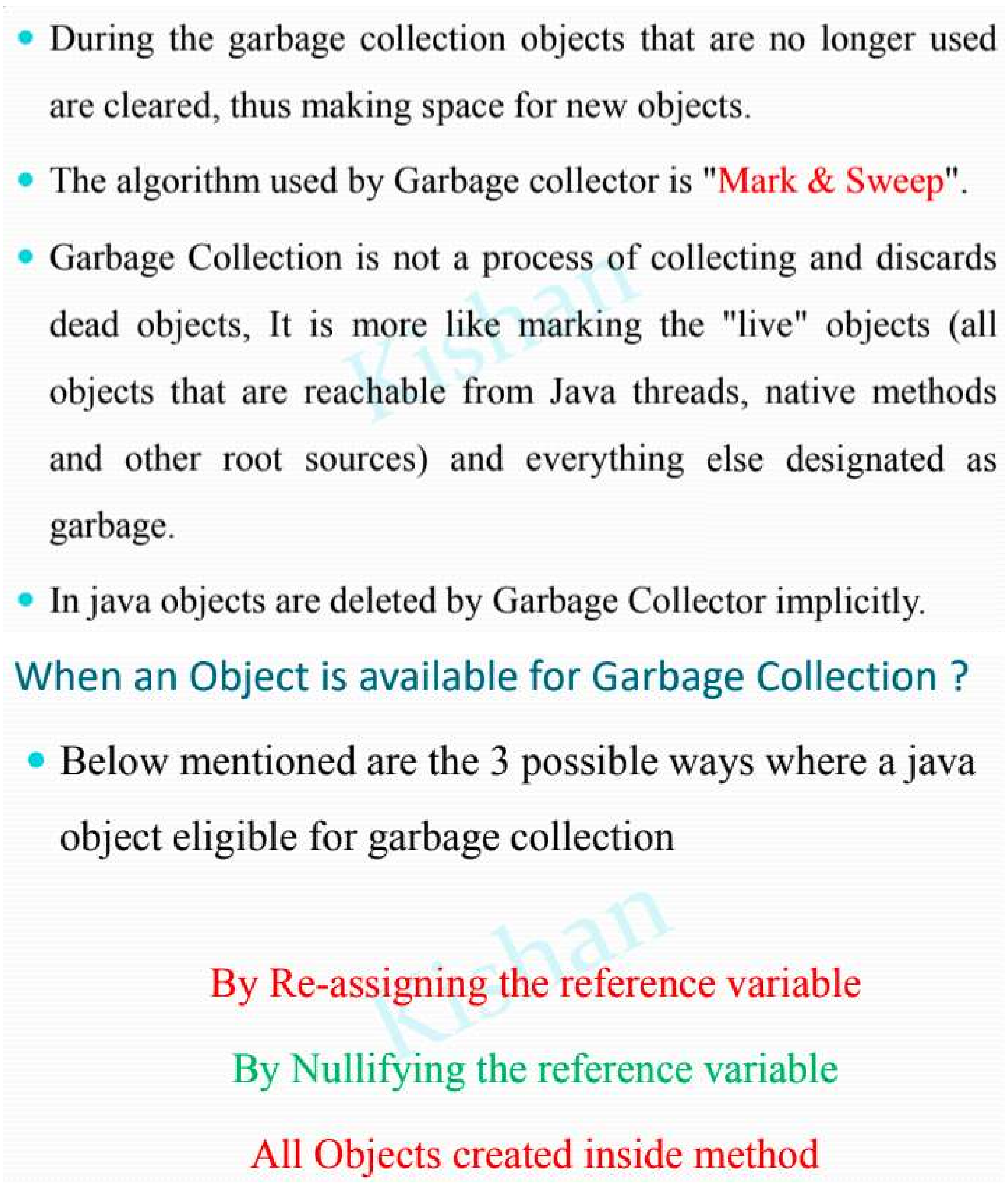
finalize(): present in Object Class

1) finalize() will be internally called by the garbage collector.
2) finalize() is going to terminate all the existing connections which were associated with the object that is going to be destroyed.



Understanding Java Garbage Collection

- Garbage collection is a mechanism of re-Acquiring the heap space by destroying the objects which are eligible for "Garbage Collection".
- Garbage collector always running in the background of a java application for removing useless objects, So that the chance of failing java program is very rare because of memory problems.
- All Java objects reside in an area called the heap.
- The heap is created when the JVM starts up and may increase or decrease in size while the application runs.
- When the heap becomes full, garbage (Unused Objects) is collected by Garbage Collector.



- During the garbage collection objects that are no longer used are cleared, thus making space for new objects.
- The algorithm used by Garbage collector is "Mark & Sweep".
- Garbage Collection is not a process of collecting and discards dead objects, It is more like marking the "live" objects (all objects that are reachable from Java threads, native methods and other root sources) and everything else designated as garbage.
- In java objects are deleted by Garbage Collector implicitly.

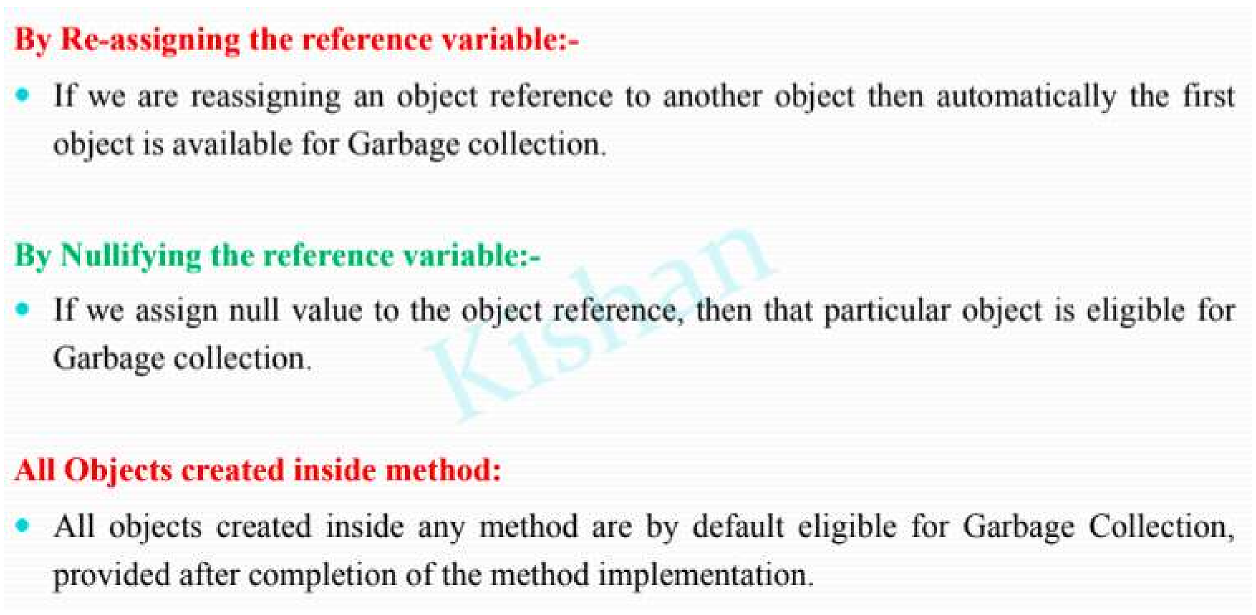
When an Object is available for Garbage Collection ?

- Below mentioned are the 3 possible ways where a java object eligible for garbage collection

By Re-assigning the reference variable

By Nullifying the reference variable

All Objects created inside method



By Re-assigning the reference variable:-

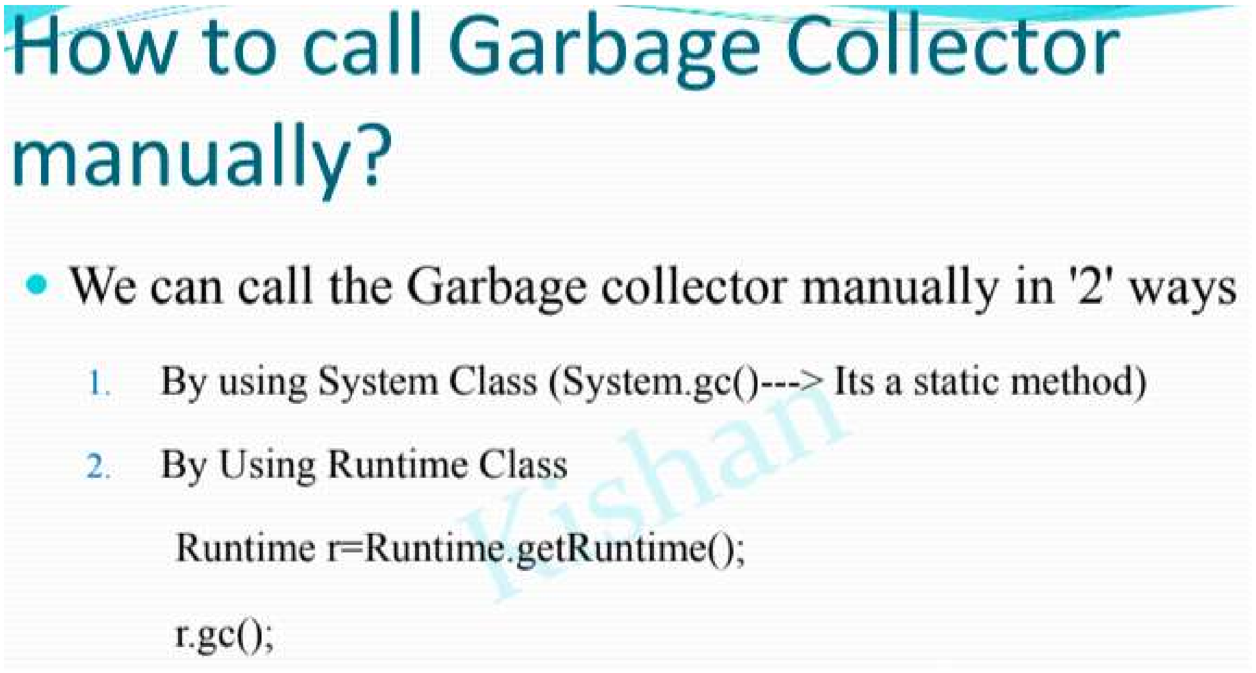
- If we are reassigning an object reference to another object then automatically the first object is available for Garbage collection.

By Nullifying the reference variable:-

- If we assign null value to the object reference, then that particular object is eligible for Garbage collection.

All Objects created inside method:

- All objects created inside any method are by default eligible for Garbage Collection, provided after completion of the method implementation.



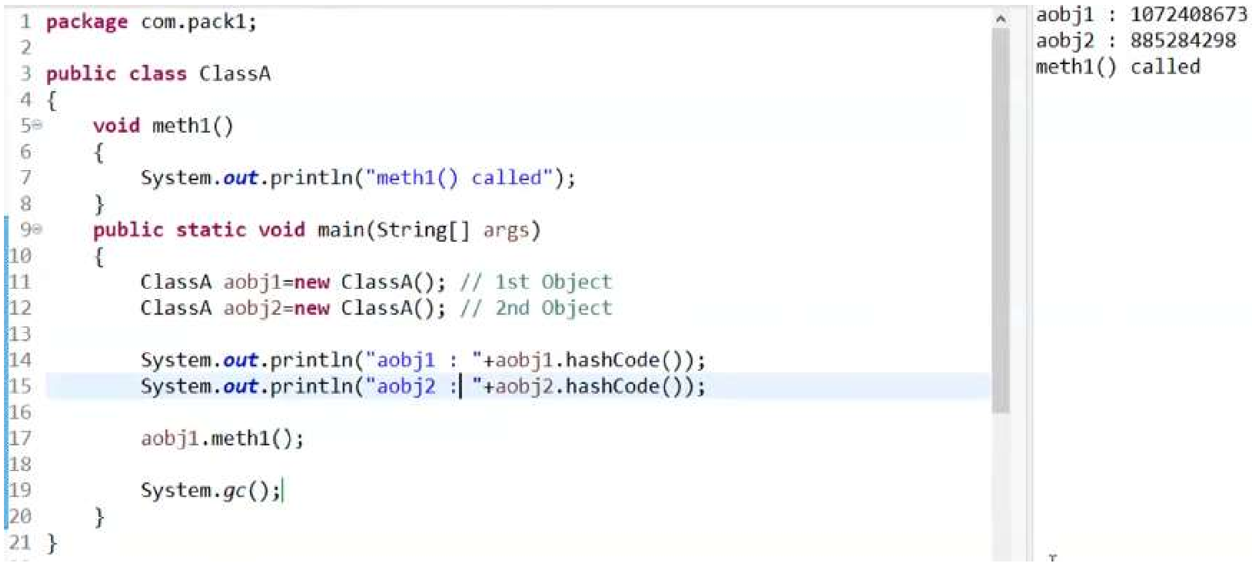
How to call Garbage Collector manually?

- We can call the Garbage collector manually in '2' ways
  1. By using System Class (System.gc()---> Its a static method)
  2. By Using Runtime Class

     Runtime r=Runtime.getRuntime();

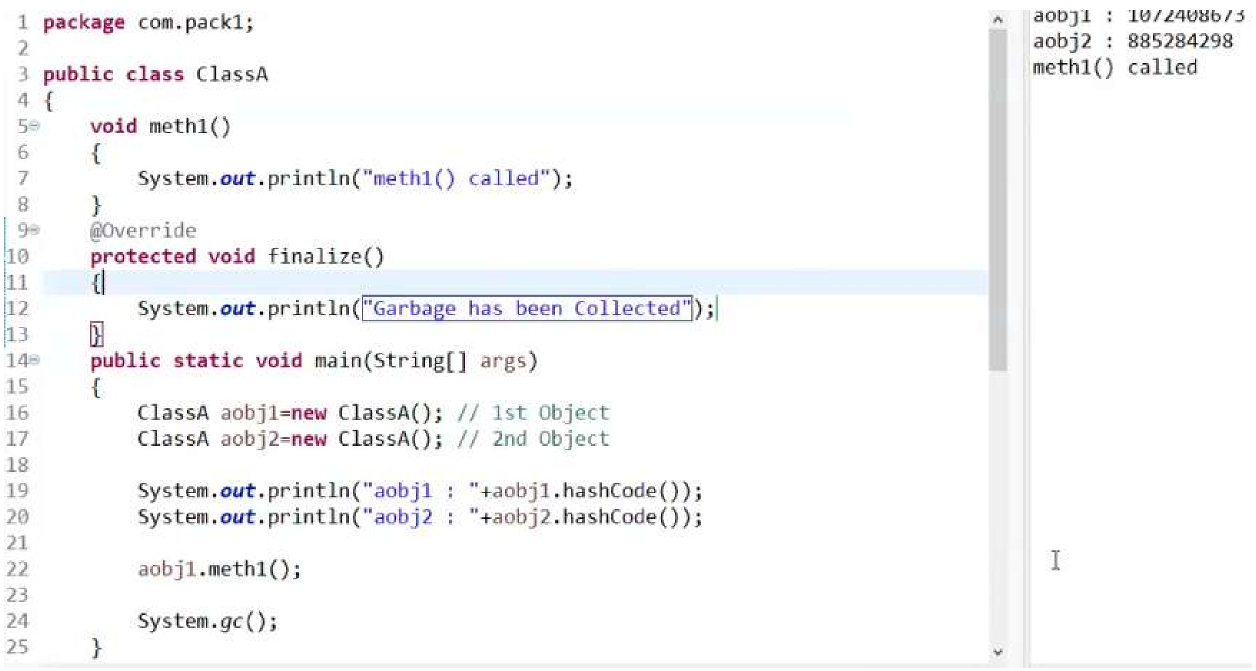
     r.gc();

There are 3 situations where objects are eligible for destruction



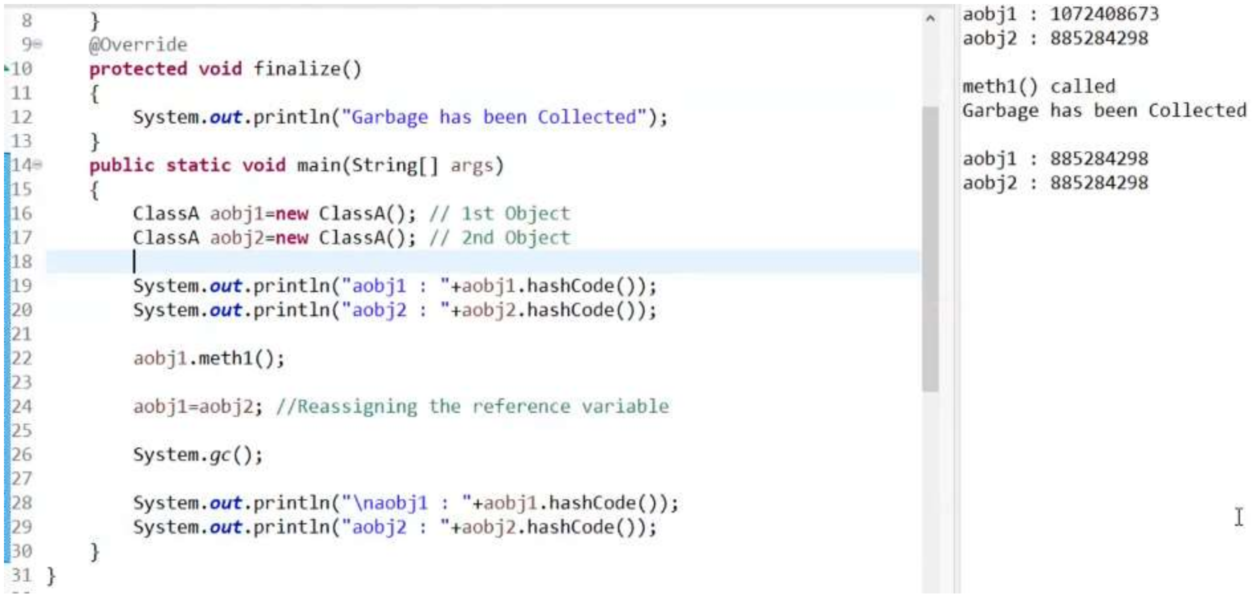
```java
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("meth1() called");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 : "+aobj1.hashCode());
        System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        System.gc();
    }
}
```

```
aobj1 : 1072408673
aobj2 : 885284298
meth1() called
```



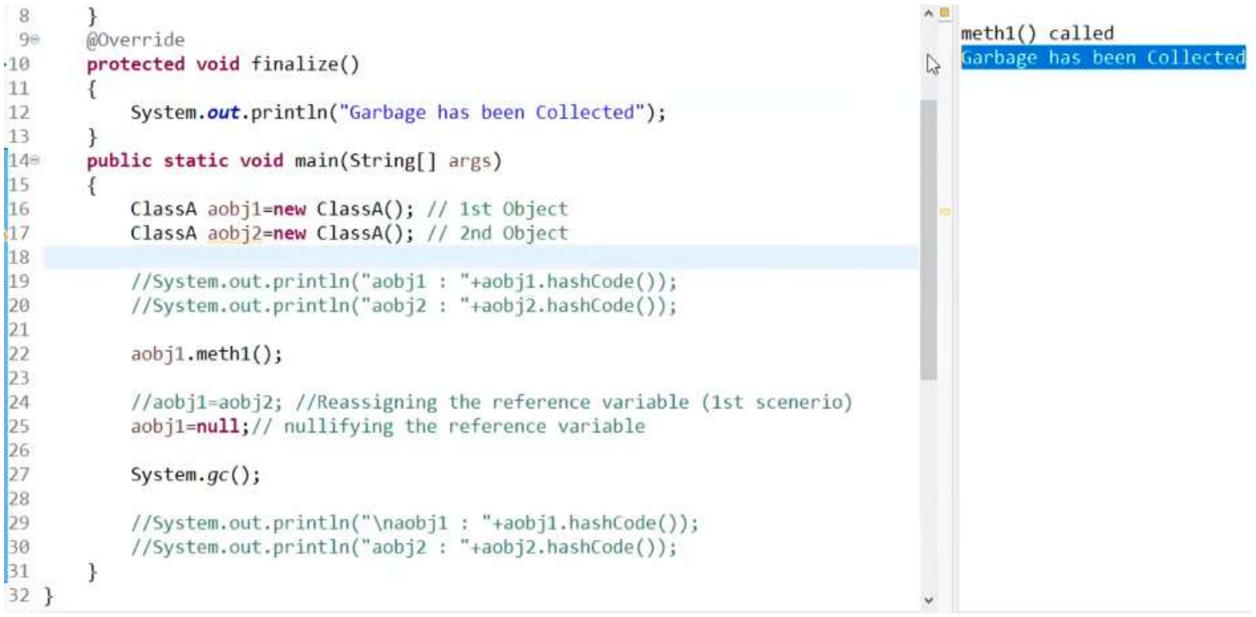
```java
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("meth1() called");
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 : "+aobj1.hashCode());
        System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        System.gc();
    }
```

```
aobj1 : 1072408673
aobj2 : 885284298
meth1() called
```



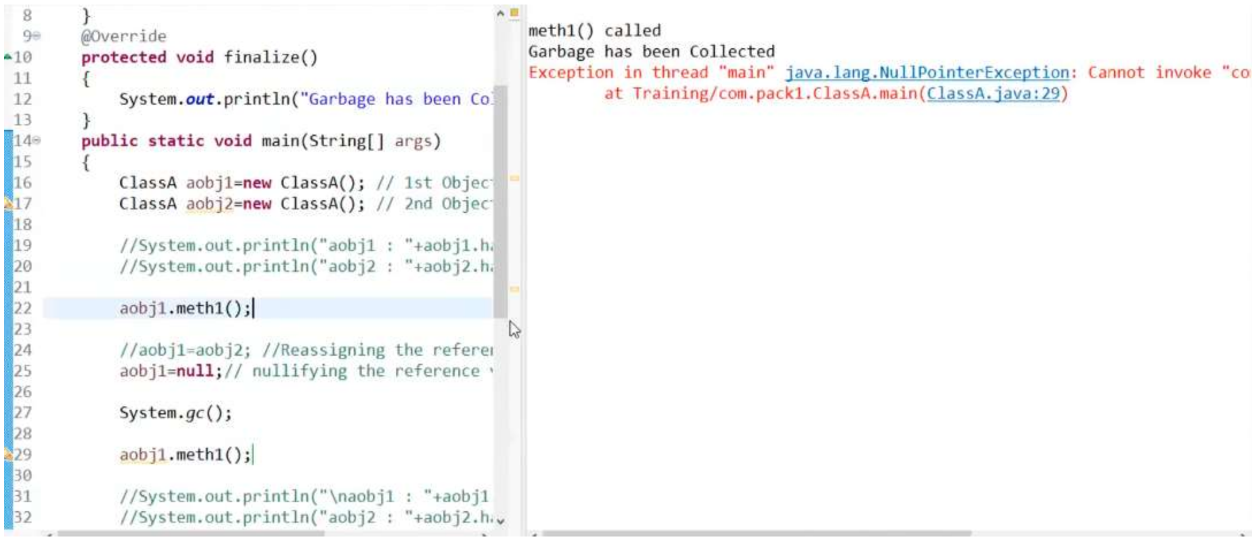
```java
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        System.out.println("aobj1 : "+aobj1.hashCode());
        System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        aobj1=aobj2; //Reassigning the reference variable

        System.gc();

        System.out.println("\naobj1 : "+aobj1.hashCode());
        System.out.println("aobj2 : "+aobj2.hashCode());
    }
}
```

```
aobj1 : 1072408673
aobj2 : 885284298

meth1() called
Garbage has been Collected

aobj1 : 885284298
aobj2 : 885284298
```



```java
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        //System.out.println("aobj1 : "+aobj1.hashCode());
        //System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        //aobj1=aobj2; //Reassigning the reference variable (1st scenerio)
        aobj1=null;// nullifying the reference variable

        System.gc();

        //System.out.println("\naobj1 : "+aobj1.hashCode());
        //System.out.println("aobj2 : "+aobj2.hashCode());
    }
}
```

```
meth1() called
Garbage has been Collected
```



```
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Co
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Objec
        ClassA aobj2=new ClassA(); // 2nd Objec

        //System.out.println("aobj1 : "+aobj1.h
        //System.out.println("aobj2 : "+aobj2.h

        aobj1.meth1();

        //aobj1=aobj2; //Reassigning the refere
        aobj1=null;// nullifying the reference 

        System.gc();

        aobj1.meth1();

        //System.out.println("\naobj1 : "+aobj1
        //System.out.println("aobj2 : "+aobj2.h
```

```
meth1() called
Garbage has been Collected
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "co
        at Training/com.pack1.ClassA.main(ClassA.java:29)
```

Generally, we have 4 types of storage.

1.variable storage/field storage (temporary storage)

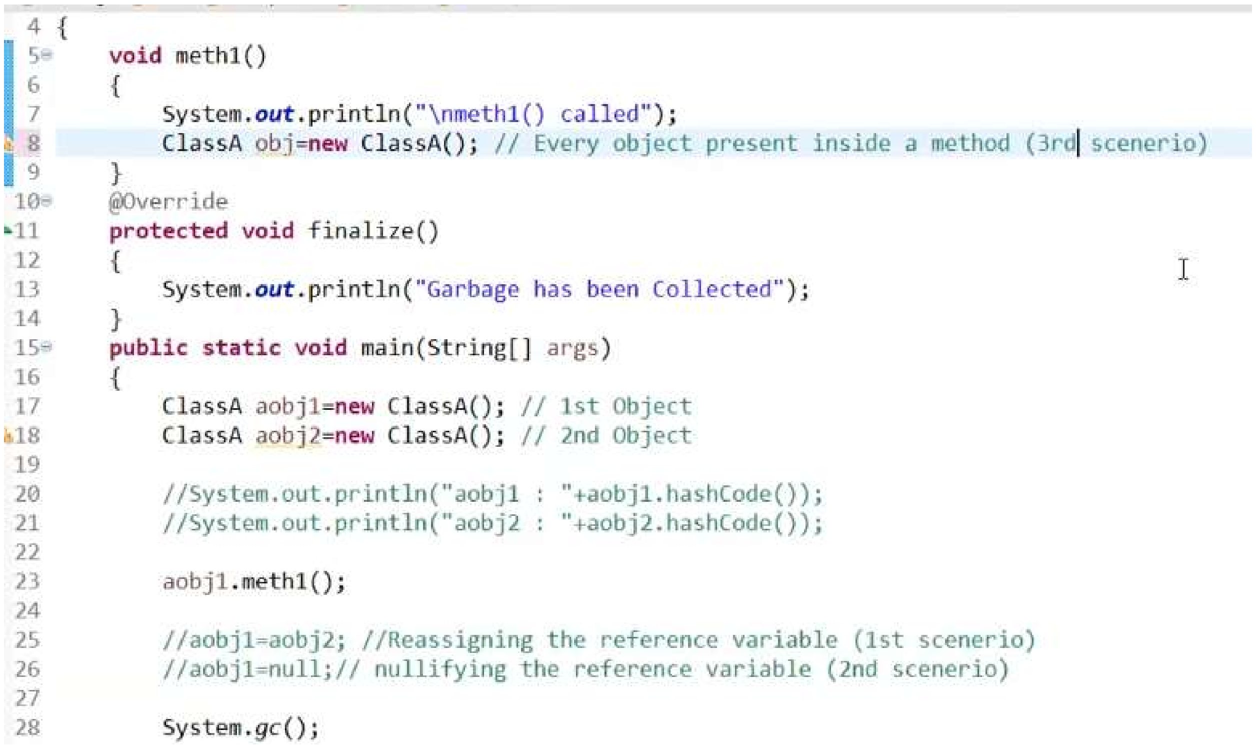
2.Object storage (temporary storage)

3.File storage (IO streams)

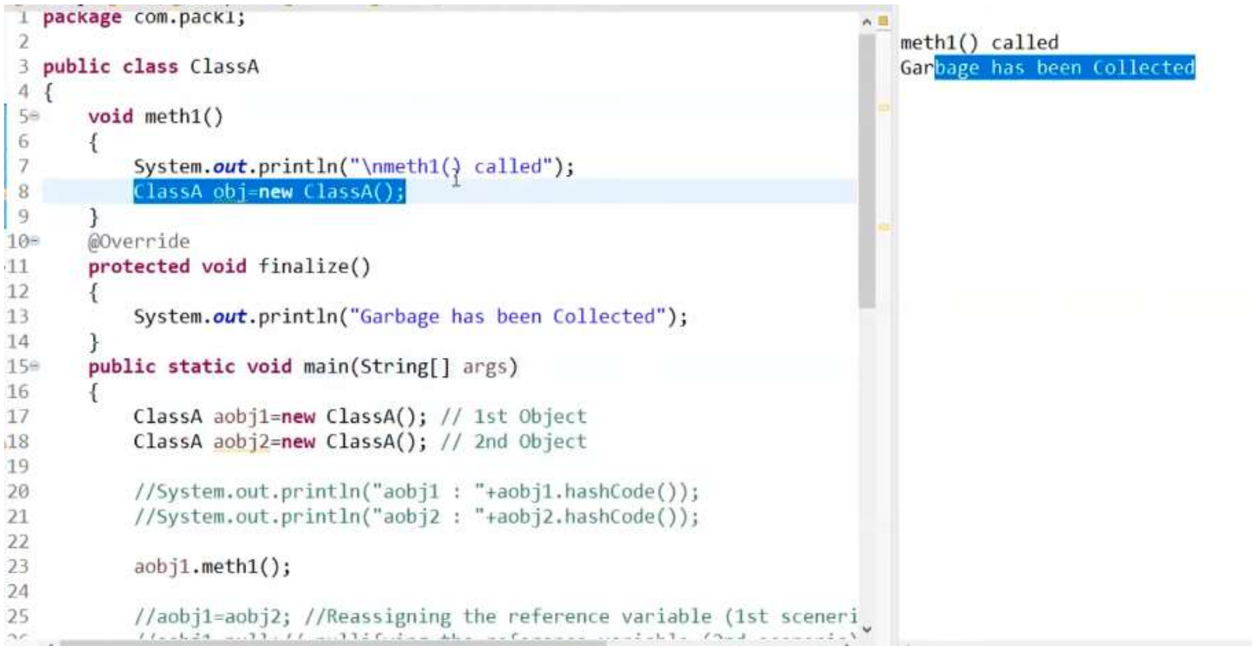
4.Data base storage (JDBC)

The objects which are presented in the main method also destroyed once we exit our JVM (means Shutdown the JVM means close the Eclipse)

The garbage collector is going to finalize() which is used to terminate all the connections with the object which is going to be destroyed.

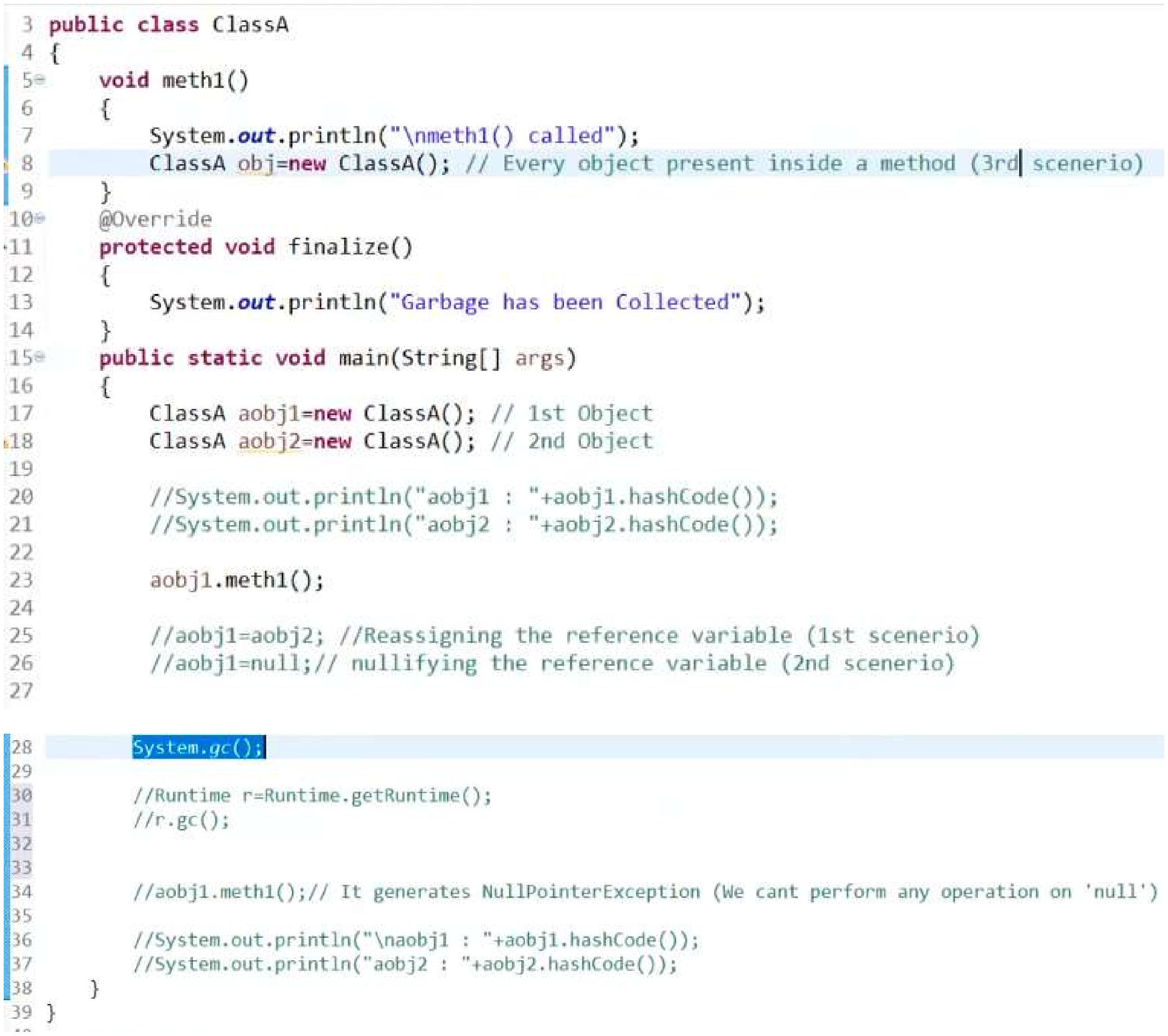


```
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
        ClassA obj=new ClassA(); // Every object present inside a method (3rd scenerio)
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        //System.out.println("aobj1 : "+aobj1.hashCode());
        //System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        //aobj1=aobj2; //Reassigning the reference variable (1st scenerio)
        //aobj1=null;// nullifying the reference variable (2nd scenerio)

        System.gc();
```

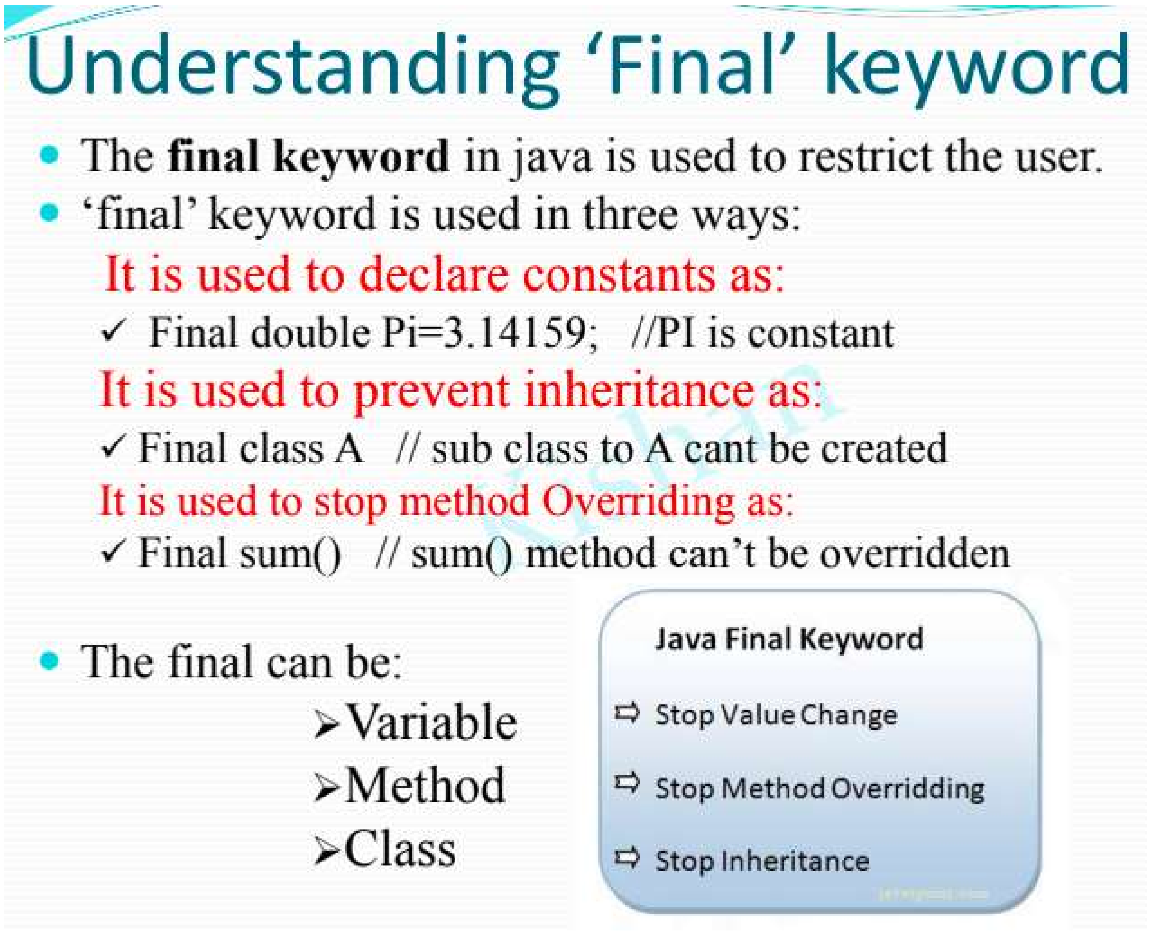


```
package com.pack1;

public class ClassA
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
        ClassA obj=new ClassA();
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        //System.out.println("aobj1 : "+aobj1.hashCode());
        //System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        //aobj1=aobj2; //Reassigning the reference variable (1st sceneri
```

```
meth1() called
Garbage has been Collected
```



```java
public class ClassA
{
    void meth1()
    {
        System.out.println("\nmeth1() called");
        ClassA obj=new ClassA(); // Every object present inside a method (3rd scenerio)
    }
    @Override
    protected void finalize()
    {
        System.out.println("Garbage has been Collected");
    }
    public static void main(String[] args)
    {
        ClassA aobj1=new ClassA(); // 1st Object
        ClassA aobj2=new ClassA(); // 2nd Object

        //System.out.println("aobj1 : "+aobj1.hashCode());
        //System.out.println("aobj2 : "+aobj2.hashCode());

        aobj1.meth1();

        //aobj1=aobj2; //Reassigning the reference variable (1st scenerio)
        //aobj1=null;// nullifying the reference variable (2nd scenerio)


        System.gc();

        //Runtime r=Runtime.getRuntime();
        //r.gc();



        //aobj1.meth1();// It generates NullPointerException (We cant perform any operation on 'null')

        //System.out.println("\naobj1 : "+aobj1.hashCode());
        //System.out.println("aobj2 : "+aobj2.hashCode());
    }
}
```



Understanding 'Final' keyword

- The **final keyword** in java is used to restrict the user.
- 'final' keyword is used in three ways:

  It is used to declare constants as:
  - ✓ Final double Pi=3.14159; //PI is constant

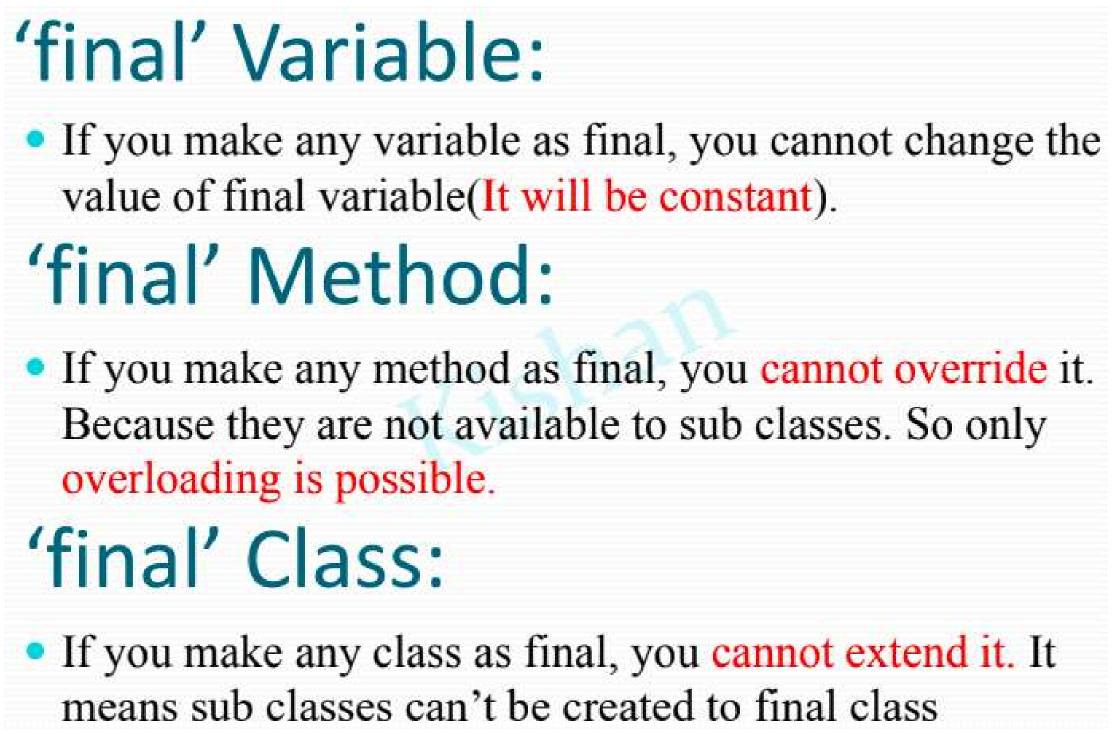
  It is used to prevent inheritance as:
  - ✓ Final class A // sub class to A cant be created

  It is used to stop method Overriding as:
  - ✓ Final sum() // sum() method can't be overridden

- The final can be:
  - ➢Variable
  - ➢Method
  - ➢Class

Java Final Keyword

- ⇨ Stop Value Change
- ⇨ Stop Method Overridding
- ⇨ Stop Inheritance



'final' Variable:

- If you make any variable as final, you cannot change the value of final variable(It will be constant).

'final' Method:

- If you make any method as final, you cannot override it. Because they are not available to sub classes. So only overloading is possible.

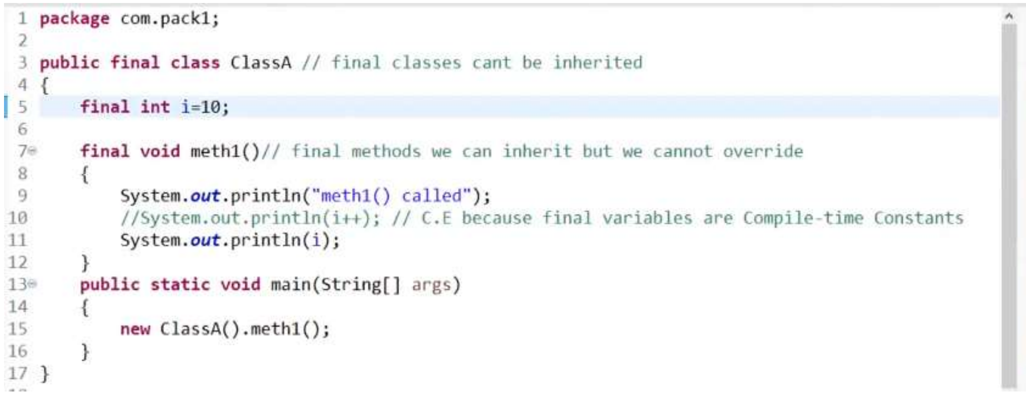
'final' Class:

- If you make any class as final, you cannot extend it. It means sub classes can't be created to final class

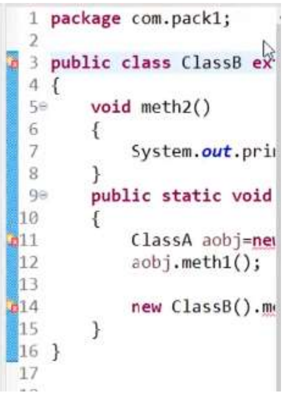
Private methods we cannot inherit so we cannot override.

Final method we can inherit but we cannot override.

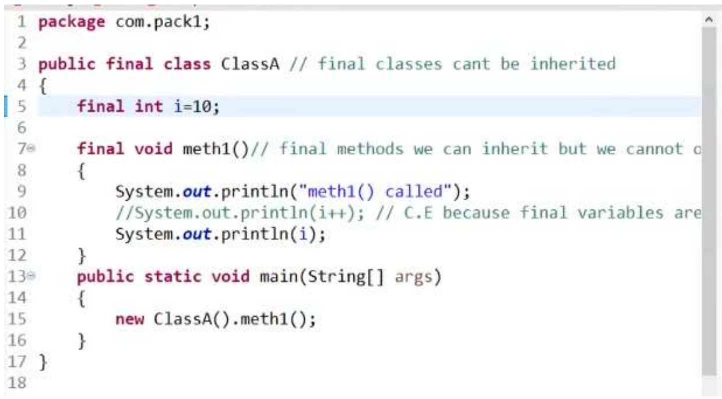
Final class can’t be inherited.



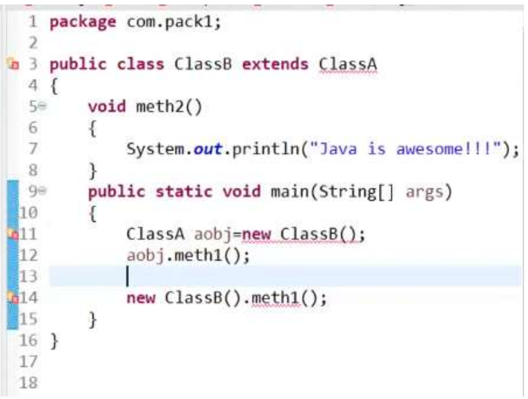
```java
package com.pack1;

public final class ClassA // final classes cant be inherited
{
    final int i=10;

    final void meth1()// final methods we can inherit but we cannot override
    {
        System.out.println("meth1() called");
        //System.out.println(i++); // C.E because final variables are Compile-time Constants
        System.out.println(i);
    }
    public static void main(String[] args)
    {
        new ClassA().meth1();
    }
}
```



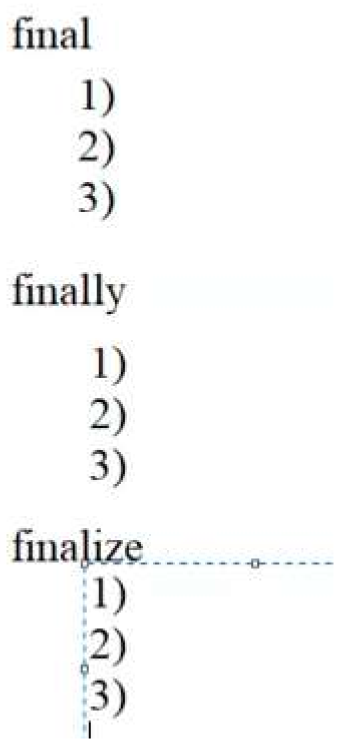
```java
package com.pack1;

public class ClassB ex
{
    void meth2()
    {
        System.out.pri
    }
    public static void
    {
        ClassA aobj=nev
        aobj.meth1();

        new ClassB().m
    }
}
```



```java
package com.pack1;

public final class ClassA // final classes cant be inherited
{
    final int i=10;

    final void meth1()// final methods we can inherit but we cannot o
    {
        System.out.println("meth1() called");
        //System.out.println(i++); // C.E because final variables are
        System.out.println(i);
    }
    public static void main(String[] args)
    {
        new ClassA().meth1();
    }
}
```



```java
package com.pack1;

public class ClassB extends ClassA
{
    void meth2()
    {
        System.out.println("Java is awesome!!!");
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassB();
        aobj.meth1();

        new ClassB().meth1();
    }
}
```



final

1)
2)
3)

finally

1)
2)
3)

finalize

1)
2)
3)

Final key word

Final variable which are compiled time constants

final methods we can inherit but we cannot override

Final Class we cannot inherit and cannot override.

Finally

It is block used in exception handling.

Multiple finally blocks are not allowed.

Finally block will be executed always the code which is present in it is used for closing the connection.

Finalize

It is method from object class

It is called internally by the garbage collector

It is used to terminate the connections of the object which is eligible for destruction