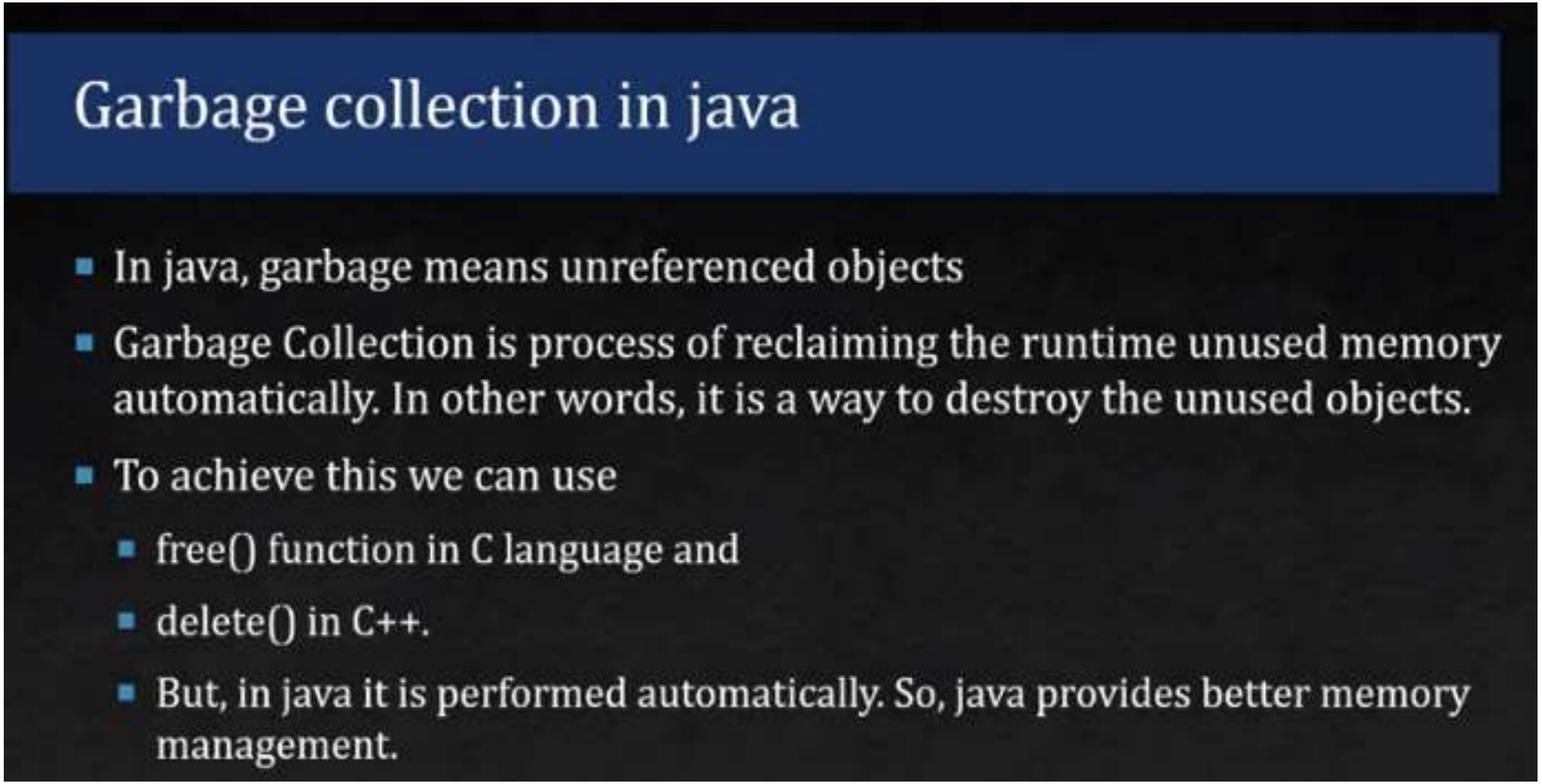
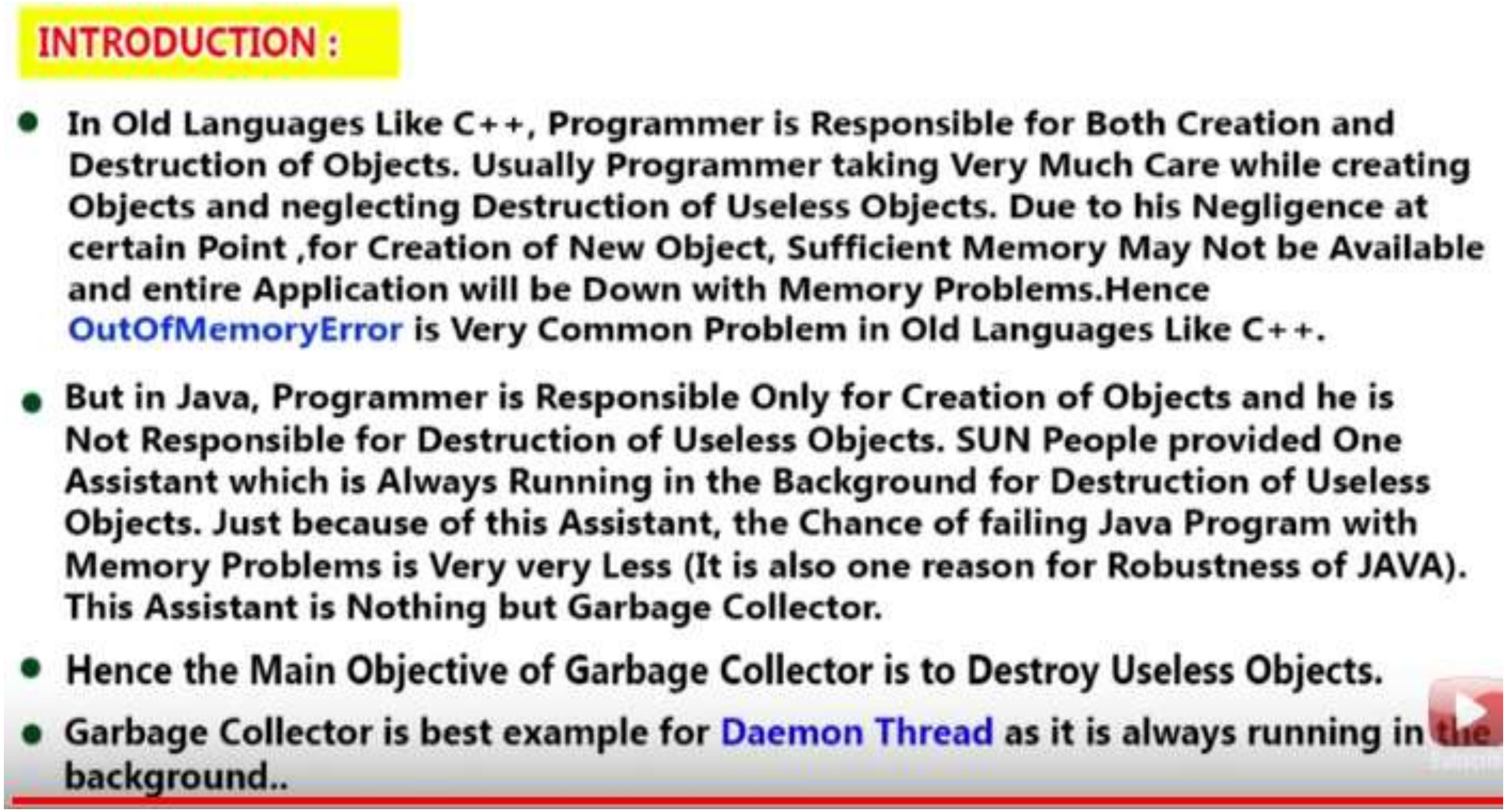
Garbage collection in java



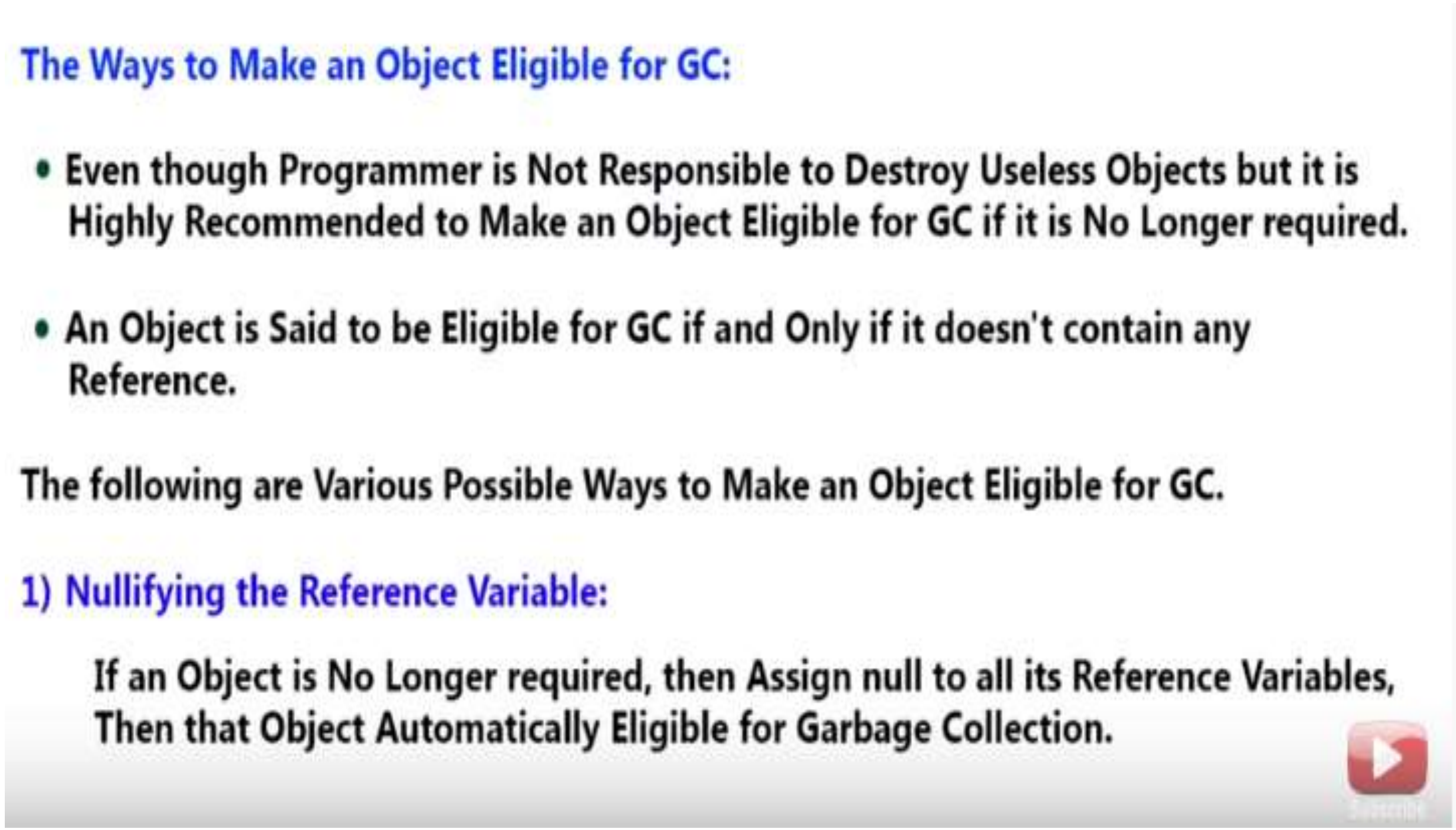
Garbage collection in java

- In java, garbage means unreferenced objects
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To achieve this we can use
  - free() function in C language and
  - delete() in C++.
  - But, in java it is performed automatically. So, java provides better memory management.



INTRODUCTION :

- In Old Languages Like C++, Programmer is Responsible for Both Creation and Destruction of Objects. Usually Programmer taking Very Much Care while creating Objects and neglecting Destruction of Useless Objects. Due to his Negligence at certain Point ,for Creation of New Object, Sufficient Memory May Not be Available and entire Application will be Down with Memory Problems.Hence OutOfMemoryError is Very Common Problem in Old Languages Like C++.
- But in Java, Programmer is Responsible Only for Creation of Objects and he is Not Responsible for Destruction of Useless Objects. SUN People provided One Assistant which is Always Running in the Background for Destruction of Useless Objects. Just because of this Assistant, the Chance of failing Java Program with Memory Problems is Very very Less (It is also one reason for Robustness of JAVA). This Assistant is Nothing but Garbage Collector.
- Hence the Main Objective of Garbage Collector is to Destroy Useless Objects.
- Garbage Collector is best example for Daemon Thread as it is always running in the background..



The Ways to Make an Object Eligible for GC:

- Even though Programmer is Not Responsible to Destroy Useless Objects but it is Highly Recommended to Make an Object Eligible for GC if it is No Longer required.

- An Object is Said to be Eligible for GC if and Only if it doesn't contain any Reference.

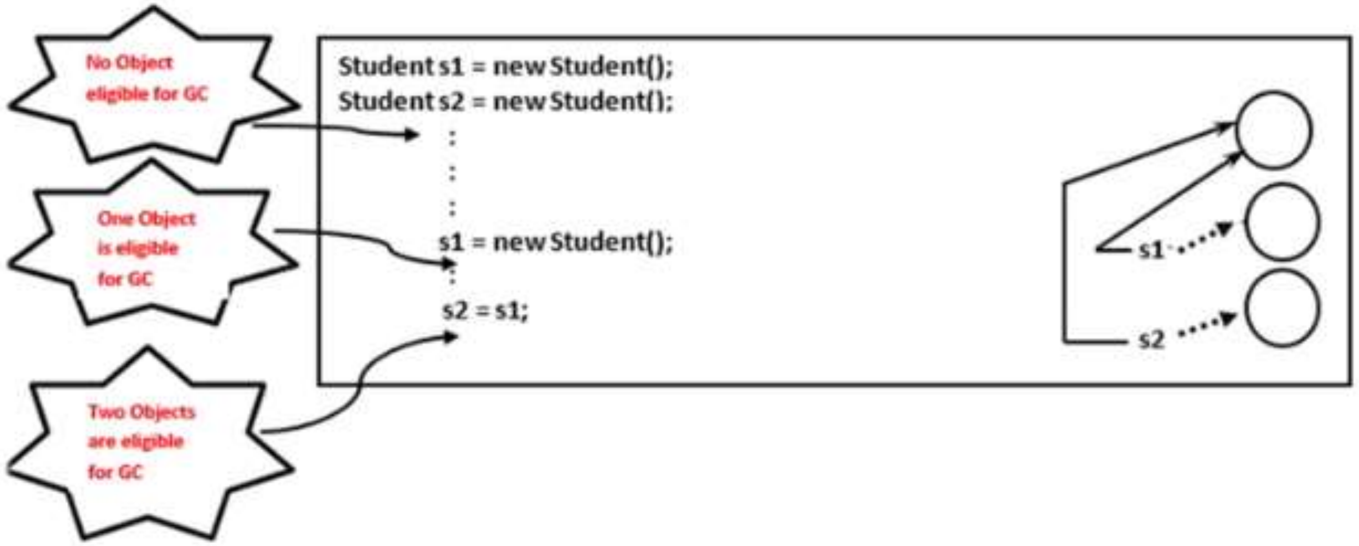
The following are Various Possible Ways to Make an Object Eligible for GC.

1) Nullifying the Reference Variable:

If an Object is No Longer required, then Assign null to all its Reference Variables, Then that Object Automatically Eligible for Garbage Collection.

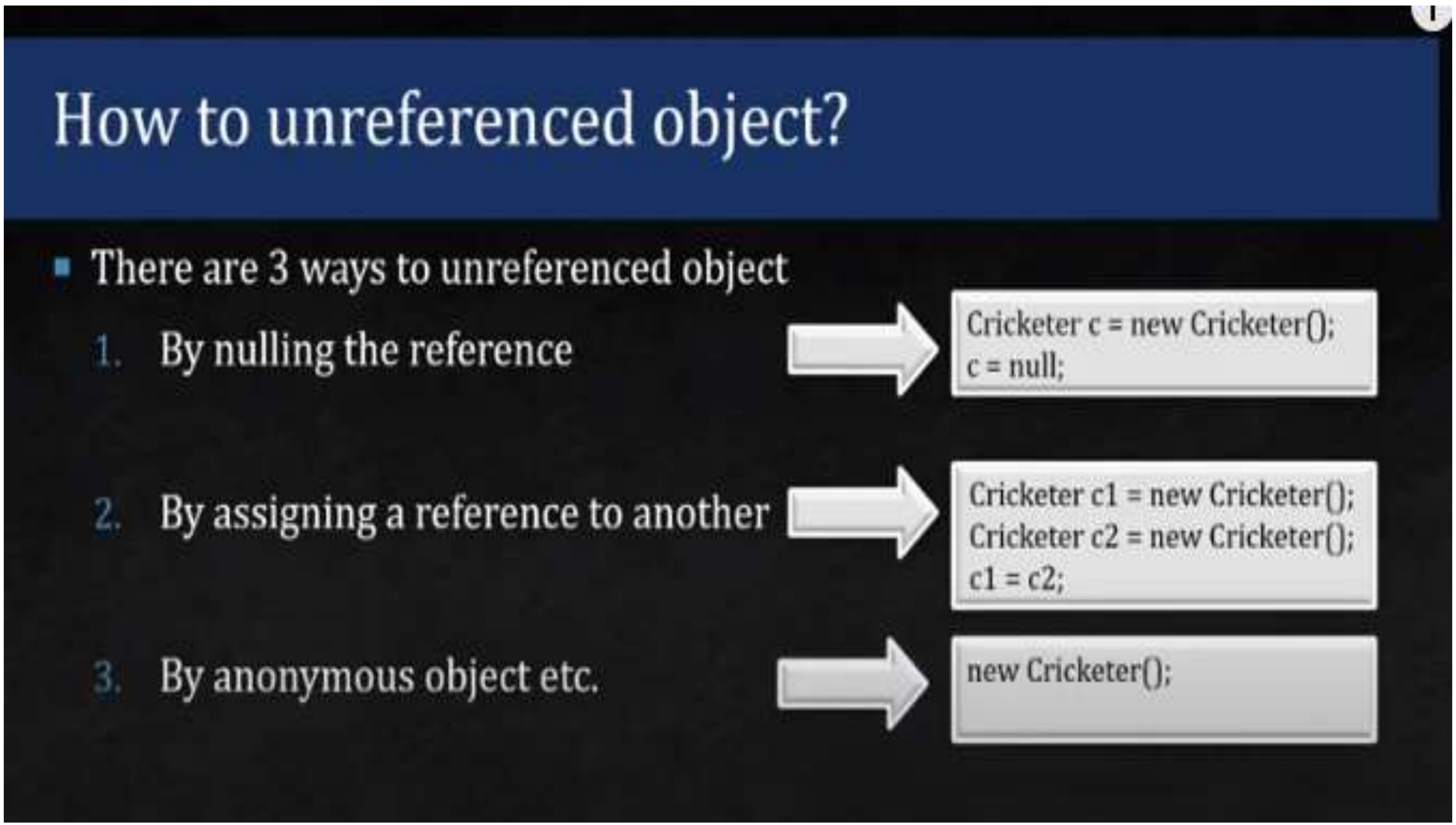
2) Re Assigning the Reference Variable:

If an Object is No Longer required then Re Assign its Reference Variable to any other object then Old Object is Automatically Eligible for GC.



```
Student s1 = new Student();
Student s2 = new Student();
      :
      :
      :
      s1 = new Student();
      :
      s2 = s1;
```

No Object eligible for GC
One Object is eligible for GC
Two Objects are eligible for GC
s1
s2



How to unreferenced object?

- There are 3 ways to unreferenced object
  1. By nulling the reference

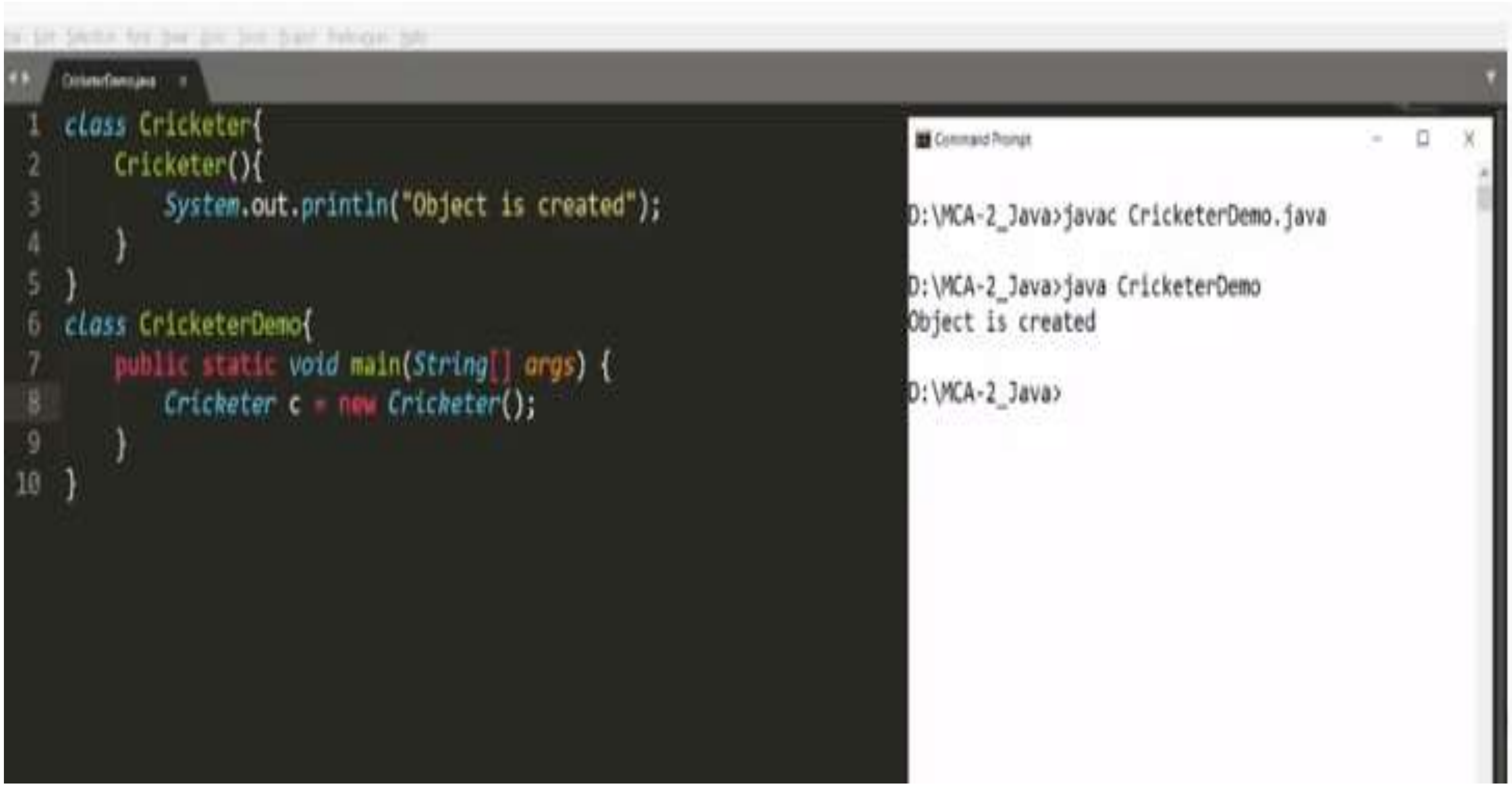
```
Cricketer c = new Cricketer();
c = null;
```

  2. By assigning a reference to another

```
Cricketer c1 = new Cricketer();
Cricketer c2 = new Cricketer();
c1 = c2;
```

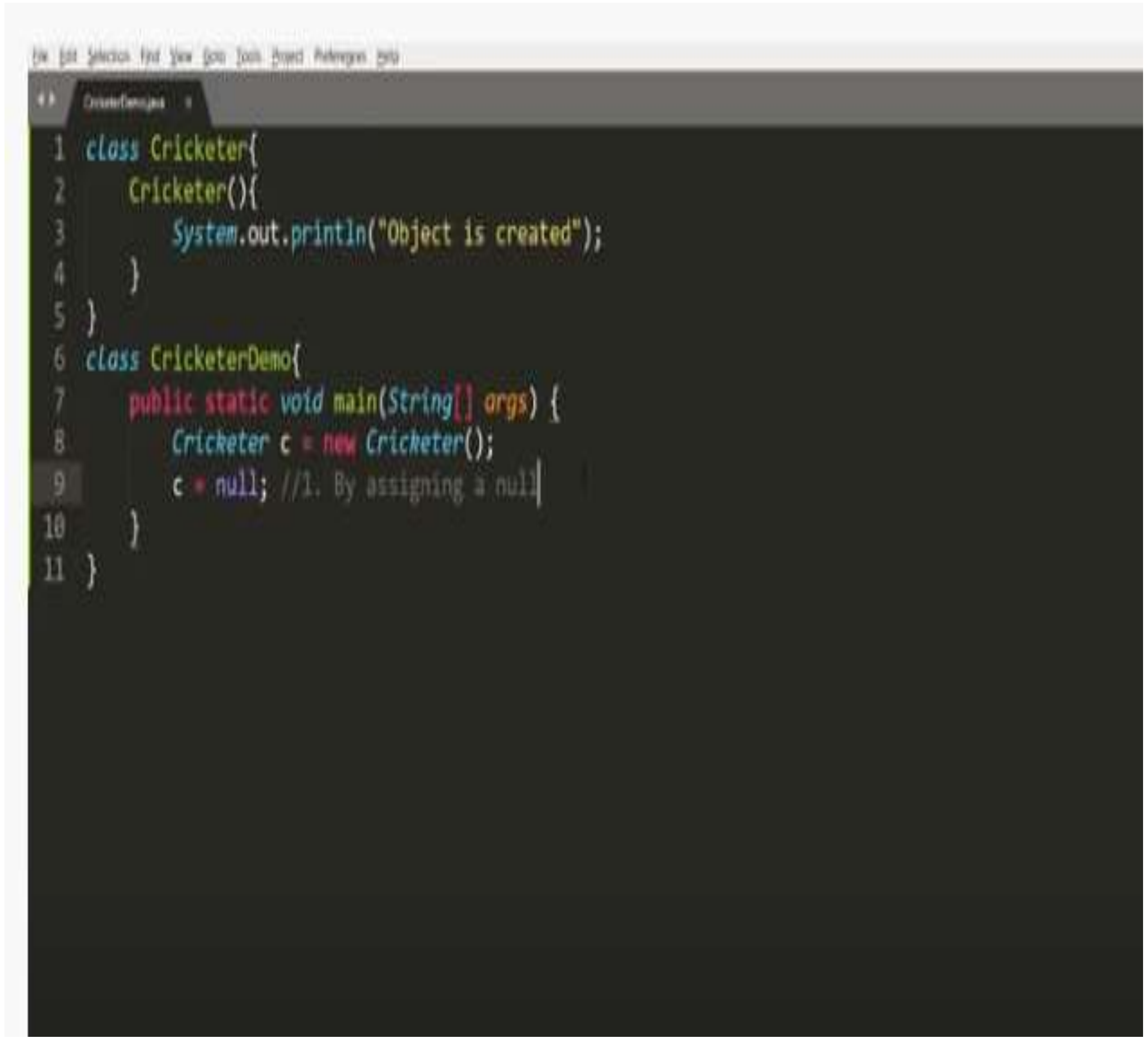
  3. By anonymous object etc.

```
new Cricketer();
```

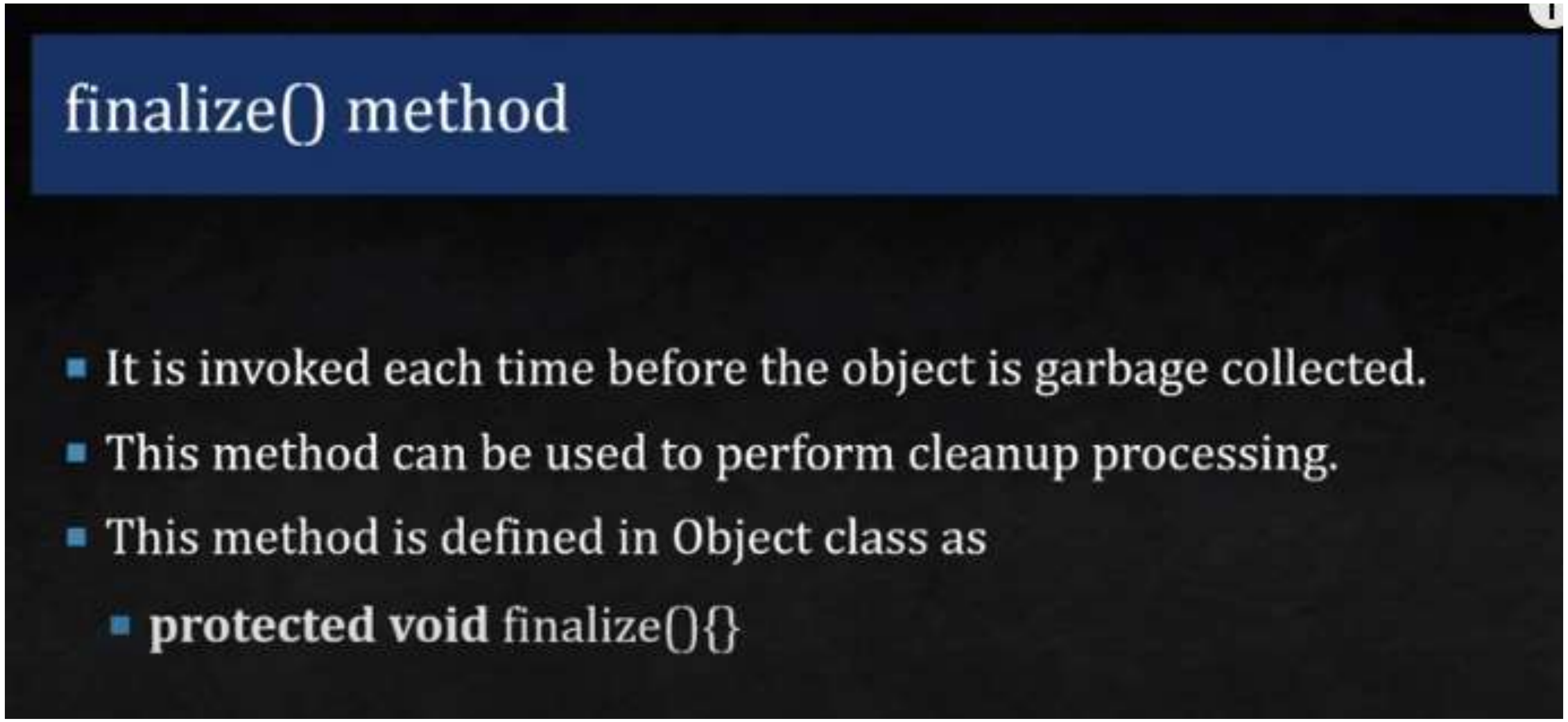


```java
class Cricketer{
    Cricketer(){
        System.out.println("Object is created");
    }
}
class CricketerDemo{
    public static void main(String[] args) {
        Cricketer c = new Cricketer();
    }
}
```

```
D:\MCA-2_Java>javac CricketerDemo.java

D:\MCA-2_Java>java CricketerDemo
Object is created

D:\MCA-2_Java>
```

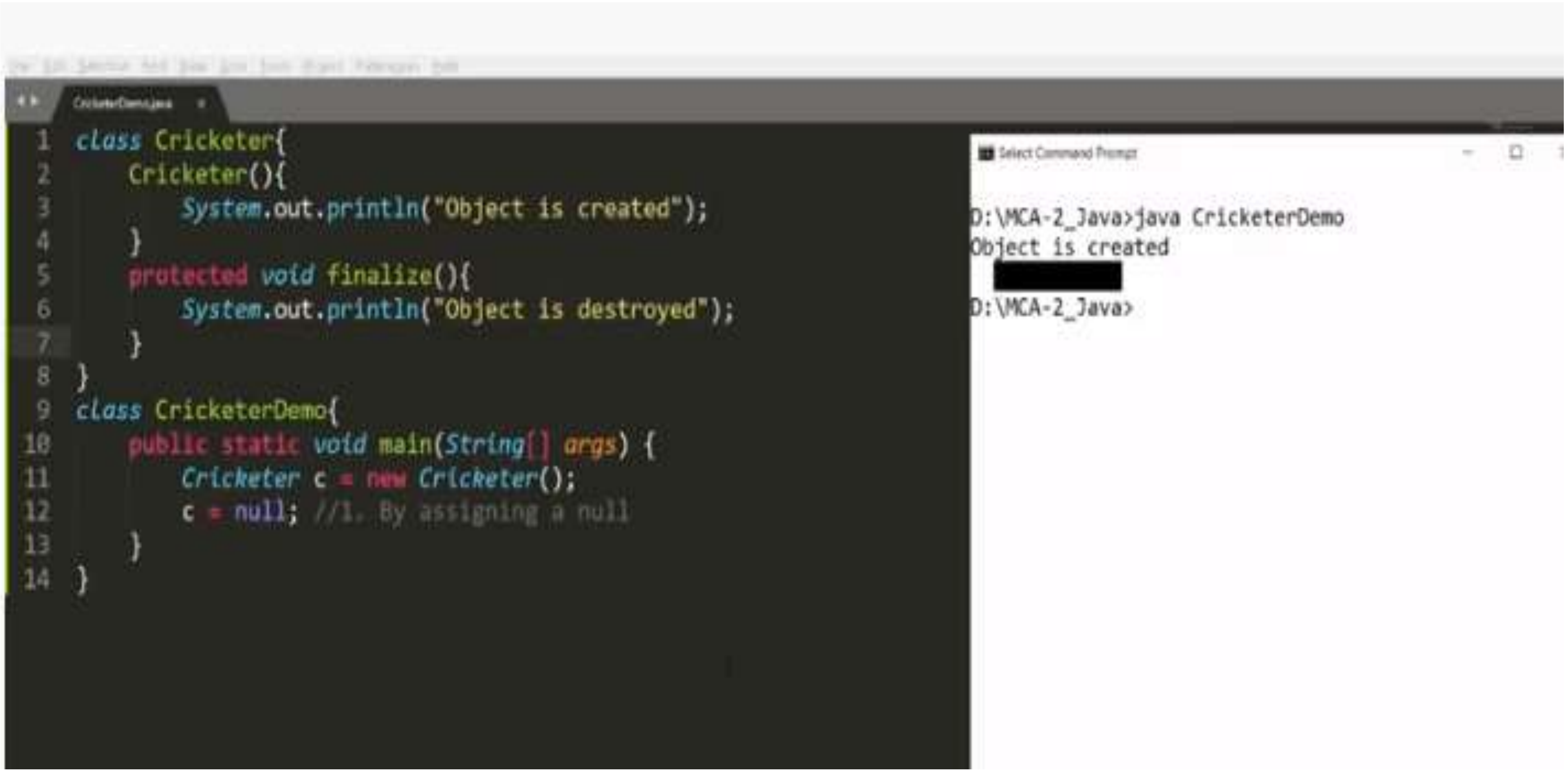


```
class Cricketer{
    Cricketer(){
        System.out.println("Object is created");
    }
}
class CricketerDemo{
    public static void main(String[] args) {
        Cricketer c = new Cricketer();
        c = null; //1. By assigning a null
    }
}
```



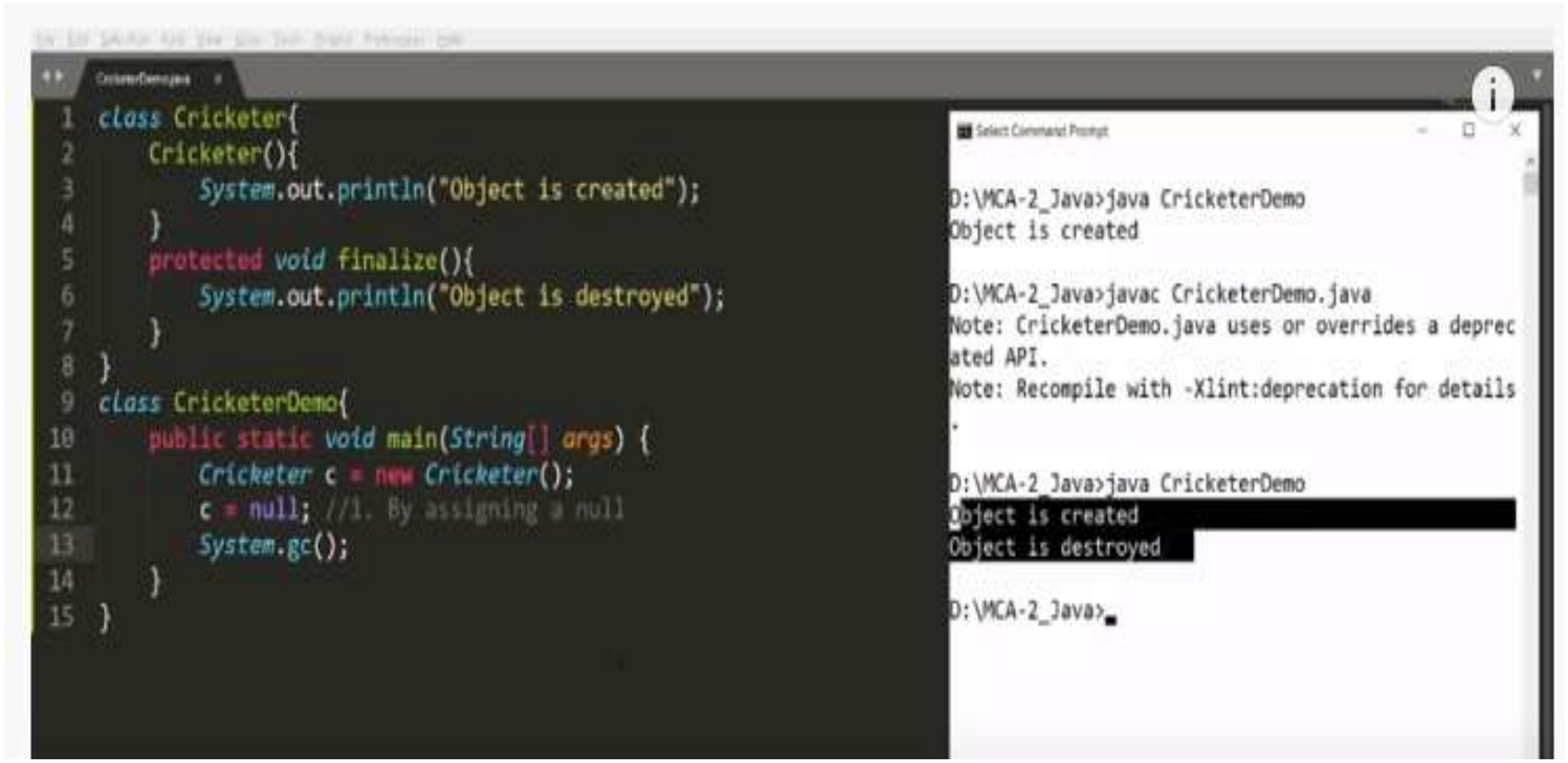
finalize() method

- It is invoked each time before the object is garbage collected.
- This method can be used to perform cleanup processing.
- This method is defined in Object class as
  - **protected void** finalize(){}



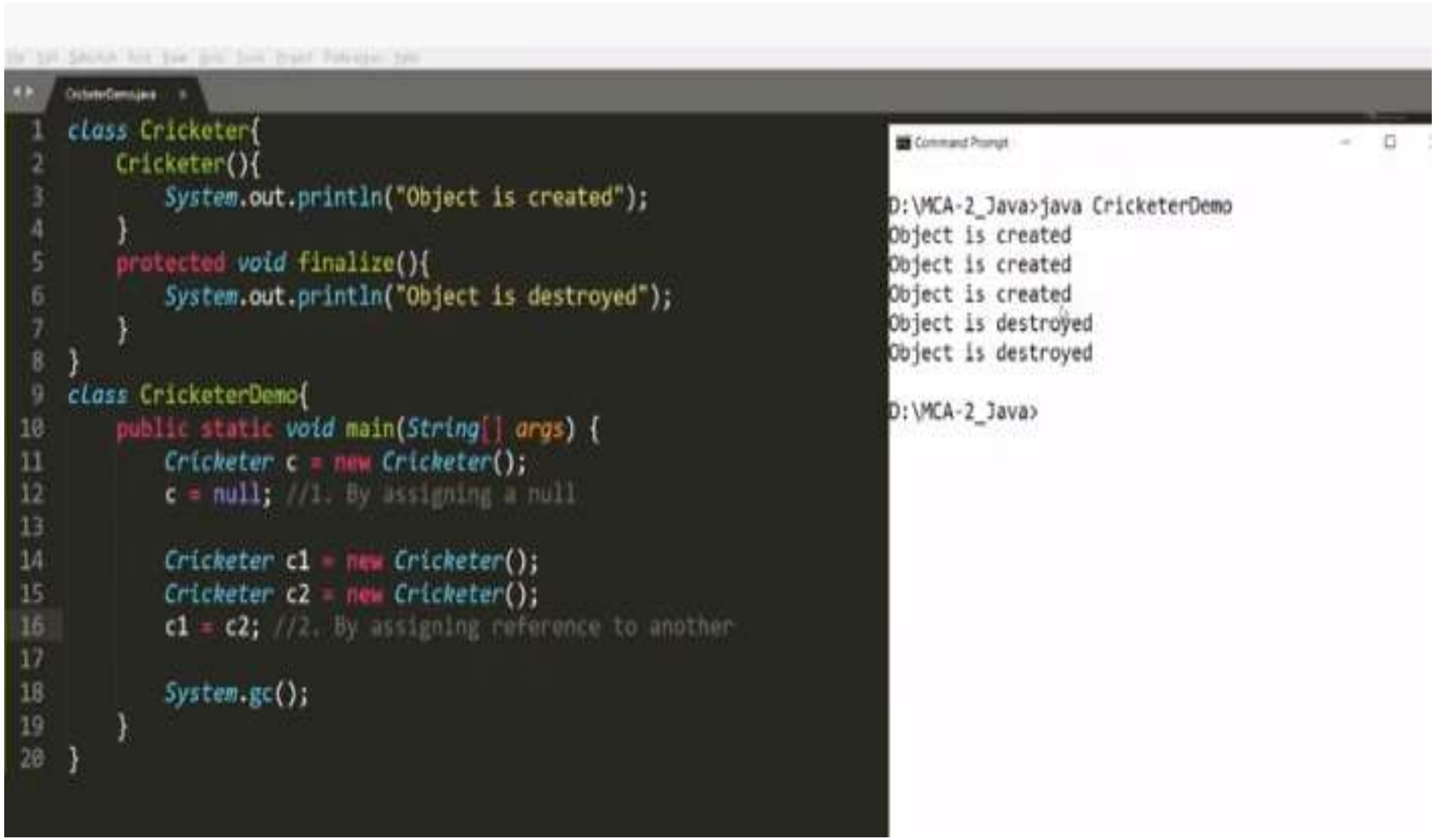
```java
class Cricketer{
    Cricketer(){
        System.out.println("Object is created");
    }
    protected void finalize(){
        System.out.println("Object is destroyed");
    }
}
class CricketerDemo{
    public static void main(String[] args) {
        Cricketer c = new Cricketer();
        c = null; //1. By assigning a null
    }
}
```

```
D:\MCA-2_Java>java CricketerDemo
Object is created
 [redacted]
D:\MCA-2_Java>
```



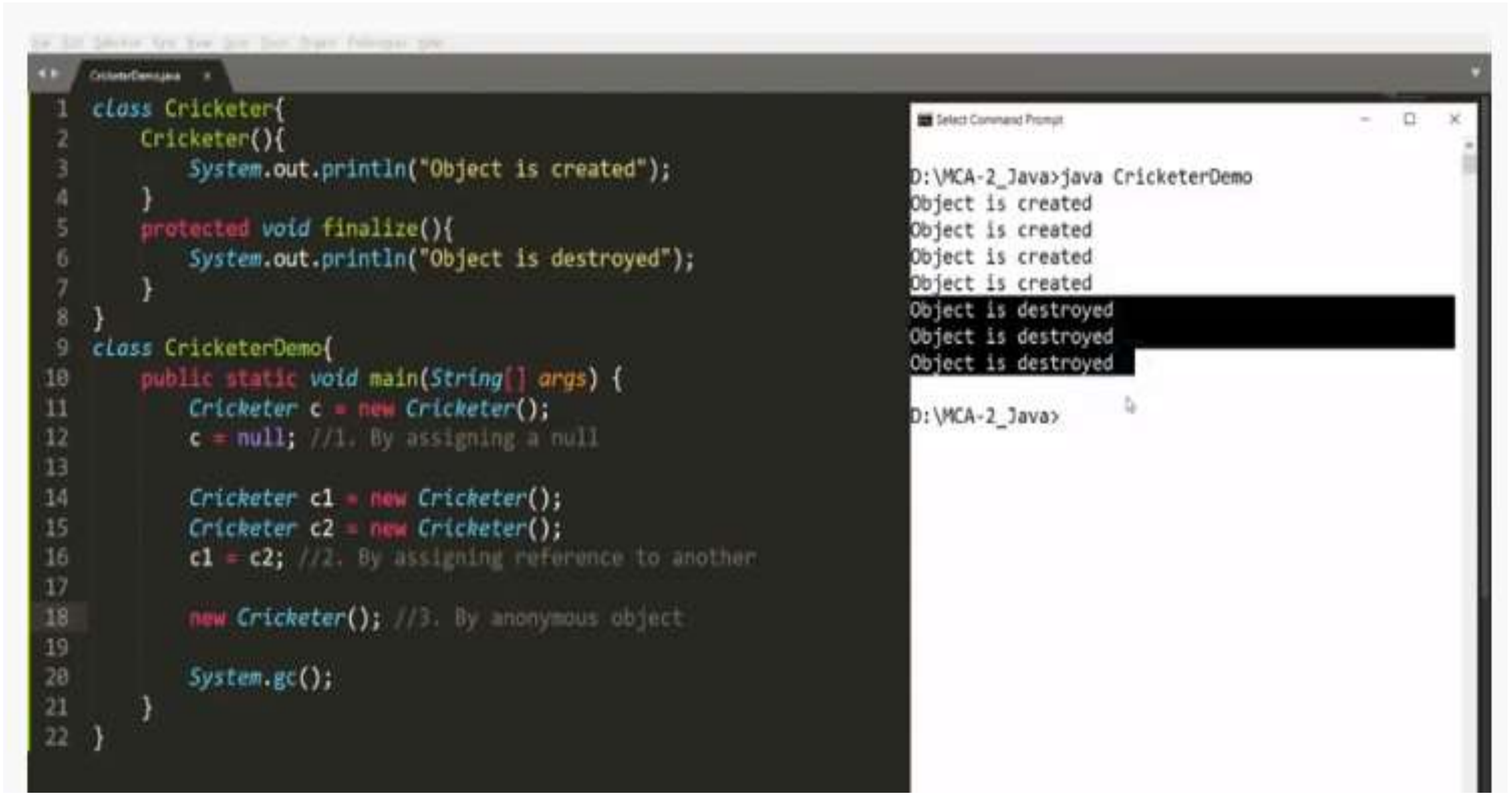
```java
class Cricketer{
    Cricketer(){
        System.out.println("Object is created");
    }
    protected void finalize(){
        System.out.println("Object is destroyed");
    }
}
class CricketerDemo{
    public static void main(String[] args) {
        Cricketer c = new Cricketer();
        c = null; //1. By assigning a null
        System.gc();
    }
}
```

```
D:\MCA-2_Java>java CricketerDemo
Object is created

D:\MCA-2_Java>javac CricketerDemo.java
Note: CricketerDemo.java uses or overrides a deprec
ated API.
Note: Recompile with -Xlint:deprecation for details
.

D:\MCA-2_Java>java CricketerDemo
Object is created
Object is destroyed

D:\MCA-2_Java>
```



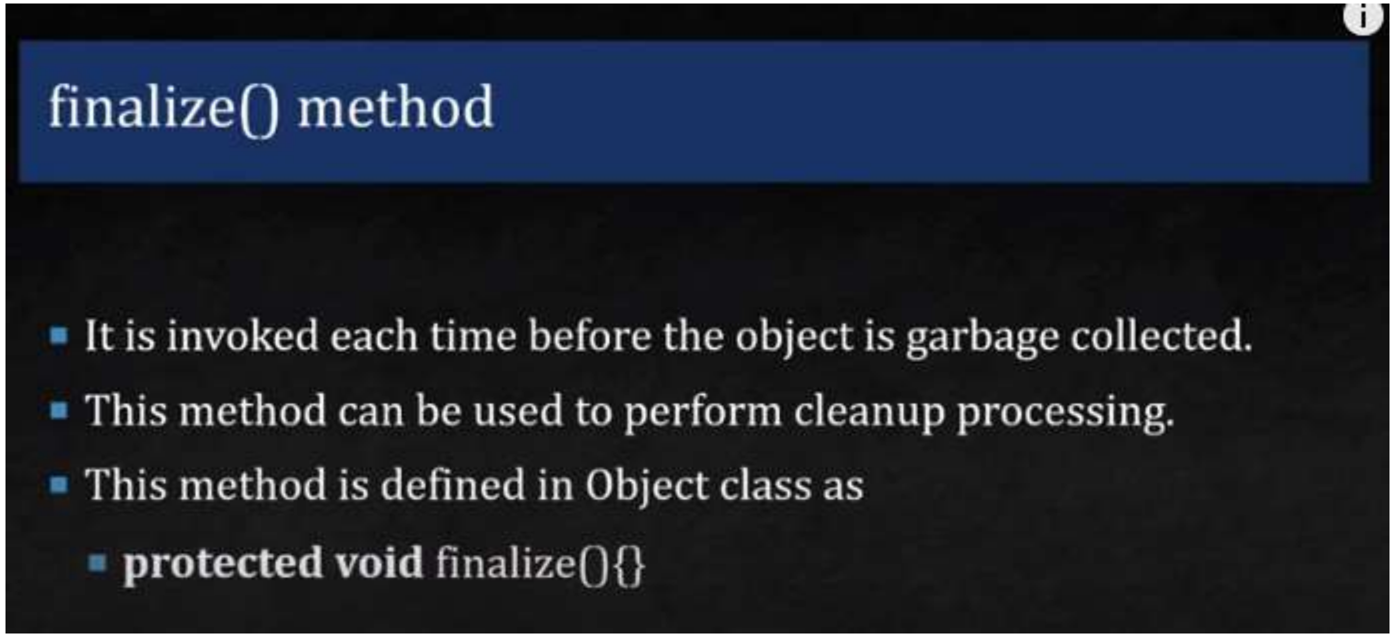
```java
class Cricketer{
    Cricketer(){
        System.out.println("Object is created");
    }
    protected void finalize(){
        System.out.println("Object is destroyed");
    }
}
class CricketerDemo{
    public static void main(String[] args) {
        Cricketer c = new Cricketer();
        c = null; //1. By assigning a null

        Cricketer c1 = new Cricketer();
        Cricketer c2 = new Cricketer();
        c1 = c2; //2. By assigning reference to another

        System.gc();
    }
}
```

```
D:\MCA-2_Java>java CricketerDemo
Object is created
Object is created
Object is created
Object is destroyed
Object is destroyed

D:\MCA-2_Java>
```

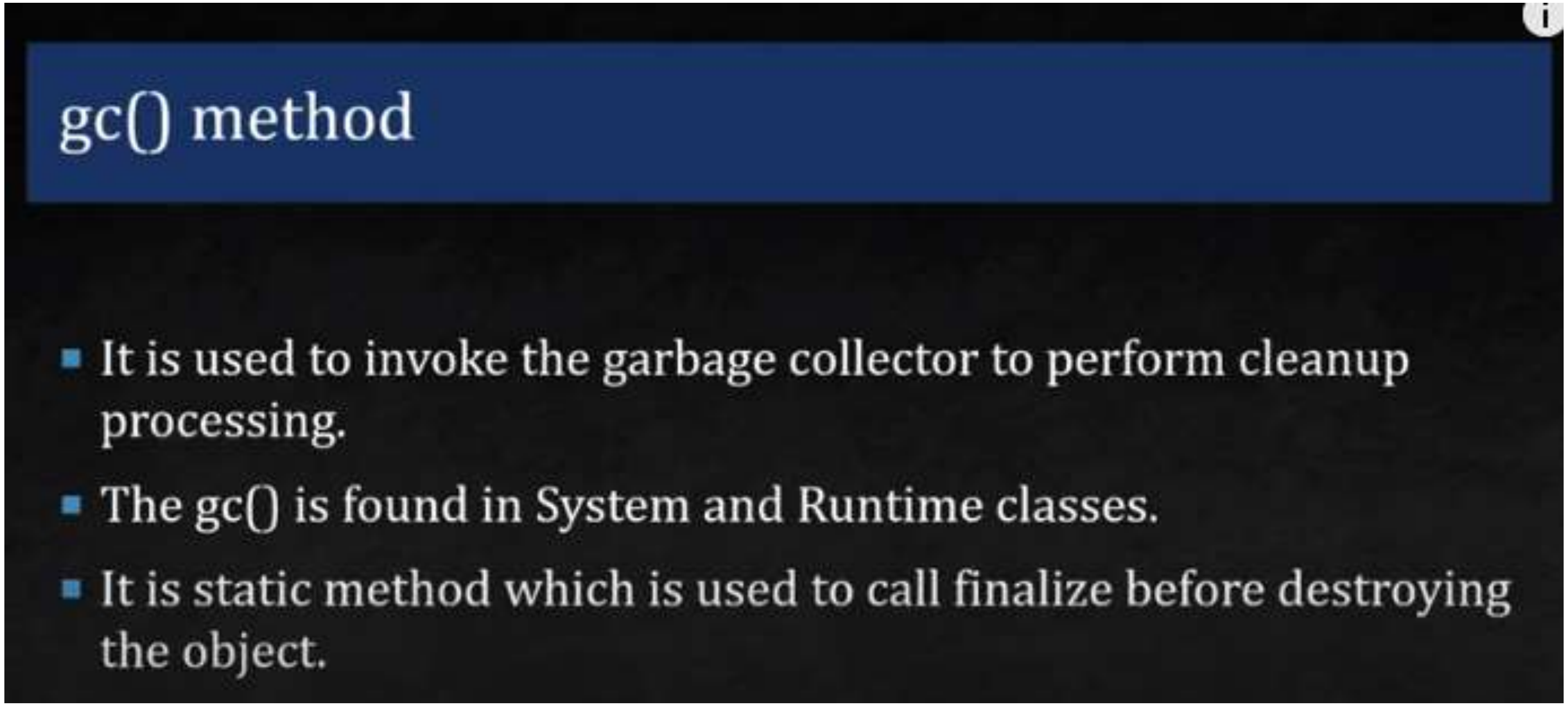


```java
class Cricketer{
    Cricketer(){
        System.out.println("Object is created");
    }
    protected void finalize(){
        System.out.println("Object is destroyed");
    }
}
class CricketerDemo{
    public static void main(String[] args) {
        Cricketer c = new Cricketer();
        c = null; //1. By assigning a null

        Cricketer c1 = new Cricketer();
        Cricketer c2 = new Cricketer();
        c1 = c2; //2. By assigning reference to another

        new Cricketer(); //3. By anonymous object

        System.gc();
    }
}
```

```
D:\MCA-2_Java>java CricketerDemo
Object is created
Object is created
Object is created
Object is created
Object is destroyed
Object is destroyed
Object is destroyed

D:\MCA-2_Java>
```


finalize() method

- It is invoked each time before the object is garbage collected.
- This method can be used to perform cleanup processing.
- This method is defined in Object class as
  - **protected void** finalize(){}



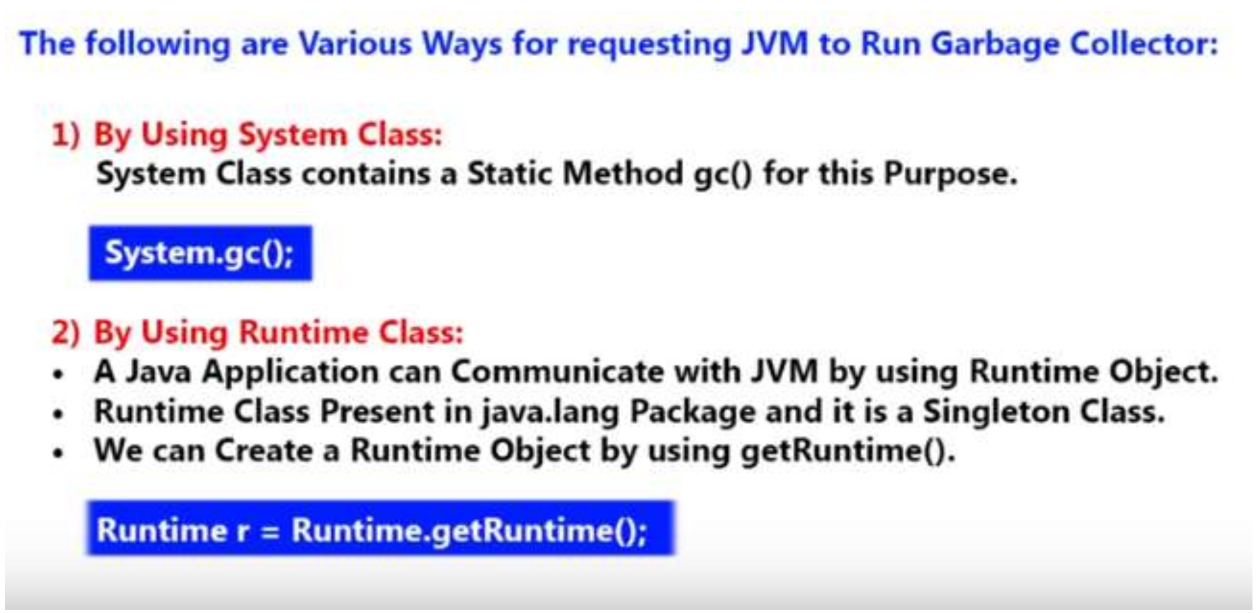
gc() method

- It is used to invoke the garbage collector to perform cleanup processing.
- The gc() is found in System and Runtime classes.
- It is static method which is used to call finalize before destroying the object.

The Ways for requesting JVM to Run Garbage Collector:

- Once we Made an Object Eligible for GC, it May Not Destroy Immediately by the Garbage Collector. Whenever JVM Runs Garbage Collector then Only Object will be Destroyed. But when exactly JVM runs GC, We can't Expect. It Depends on JVM and varied from JVM to JVM.

- Instead of waiting until JVM Runs GC, we can Request JVM to Run Garbage Collector. But there is No Guarantee whether JVM Accept Our Request OR Not. But Most of the Times JVM Accepts Our Request.



The following are Various Ways for requesting JVM to Run Garbage Collector:

1) By Using System Class:

System Class contains a Static Method gc() for this Purpose.

```
System.gc();
```

2) By Using Runtime Class:

- A Java Application can Communicate with JVM by using Runtime Object.
- Runtime Class Present in java.lang Package and it is a Singleton Class.
- We can Create a Runtime Object by using getRuntime().

```
Runtime r = Runtime.getRuntime();
```

CLASSPATH variable

Classpath is a parameter in the Java Virtual Machine or the Java compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable.

The CLASSPATH variable is one way to tell applications, including the JDK tools, where to look for user classes. (Classes that are part of the JRE, JDK platform, and extensions should be defined through other means, such as the bootstrap class path or the extensions directory.)
The preferred way to specify the class path is by using the -cp command line switch. This allows the CLASSPATH to be set individually for each application without affecting other applications. *Setting the CLASSPATH can be tricky and should be performed with care.*
The default value of the class path is ".", meaning that only the current directory is searched. Specifying either the CLASSPATH variable or the -cp command line switch overrides this value.

To check whether CLASSPATH is set on Microsoft Windows NT/2000/XP, execute the following:

```
C:> echo %CLASSPATH%
```

On Solaris or Linux, execute the following:

```
% echo $CLASSPATH
```

If CLASSPATH is not set you will get a **CLASSPATH: Undefined variable** error (Solaris or Linux) or simply **%CLASSPATH%** (Microsoft Windows NT/2000/XP).

To modify the CLASSPATH, use the same procedure you used for the PATH variable.
Class path wildcards allow you to include an entire directory of .jar files in the class path without explicitly naming them individually. For more information, including an explanation of class path wildcards, and a detailed description on how to clean up the CLASSPATH environment variable, see the Setting the Class Path technical note.

Setting the path to execute Java programs

Supplying as application argument

Suppose we have a package called *org.mypackage* containing the classes:

HelloWorld (main class)

SupportClass

UtilClass

and the files defining this package are stored physically under the directory *D:\myprogram* (on Windows) or */home/user/myprogram* (on Linux).

where:

`java` is the Java runtime launcher, a type of SDK Tool (A command-line tool, such as javac, javadoc, or apt)

-classpath D:\myprogram sets the path to the packages used in the program (on Linux, *-cp /home/user/myprogram*) and

org.mypackage.HelloWorld is the name of the main class

Setting the path through an environment variable

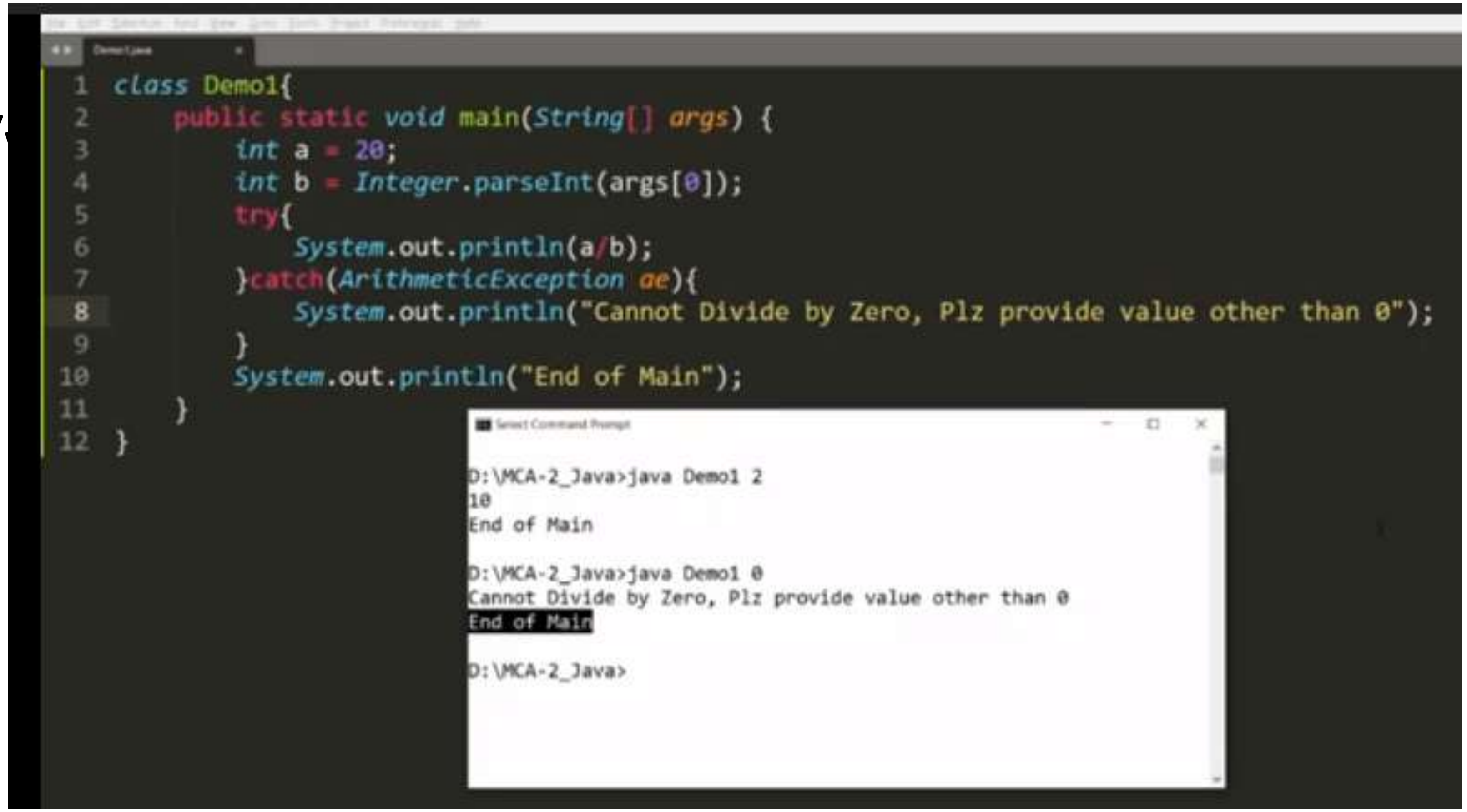
The environment variable named `CLASSPATH` may be alternatively used to set the classpath. For the above example, we could also use on Windows:

```
set CLASSPATH=D:\myprogram java org.mypackage.HelloWorld
```

The rule is that `-classpath` option, when used to start the java application, overrides the `CLASSPATH` environment variable. If none are specified, the current working directory is used as classpath. This means that when our working directory is `D:\myprogram\` (on Linux, `/home/user/myprogram/`), we would not need to specify the classpath explicitly. When overriding however, it is advised to include the current folder `"."` into the classpath in the case when loading classes from current folder is desired.

The same applies not only to java launcher but also to javac, the java compiler.

- https://



```java
class Demo1{
    public static void main(String[] args) {
        int a = 20;
        int b = Integer.parseInt(args[0]);
        try{
            System.out.println(a/b);
        }catch(ArithmeticException ae){
            System.out.println("Cannot Divide by Zero, Plz provide value other than 0");
        }
        System.out.println("End of Main");
    }
}
```

```
D:\MCA-2_Java>java Demo1 2
10
End of Main

D:\MCA-2_Java>java Demo1 0
Cannot Divide by Zero, Plz provide value other than 0
End of Main

D:\MCA-2_Java>
```