BCA – 401: Java Programming

Rahul Kumar Singh

In today's Class we have discussed on visibility control and finalizer methods in Java.

Visibility Control:-

In java programming, we use Modifiers (Specifiers) to control the visibility of program.

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

Access Modifiers:-

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. It is used to controls the access level. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers in Java programming:

- Private
- Default
- Protected
- Public

Understanding Java Access Modifiers:-

Understand following access modifiers by a simple table.

Access Modifier	Within class	Within package	Outside package (by subclass only)	Outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Private:-

The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Example:-

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
class A

{

private int data=40;

private void msg()

{

System.out.println("Hello java");
}
}
public class Simple
{
 public static void main(String args[])
{
   A obj=new A();
   System.out.println(obj.data);       //Compile Time Error
   obj.msg();          //Compile Time Error
  }
}
```

Output:-

```
Compile time error
```

Default:-

The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If we do not specify any access level, it will be the default.

It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example:-

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A
{
 void msg()
 {
     System.out.println("Hello");
  }
}
//save by B.java
package mypack;
import pack.*;
class B
{
 public static void main(String args[])
 {
    A obj = new A();    //Compile Time Error
    obj.msg();             //Compile Time Error
  }
}
```

Output:-

```
Compile Time Error
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

Protected:-

The access level of a protected modifier is within the package and outside the package through child class only. If you do not make the child class, it cannot be accessed from outside the package.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example:-

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

//save by A.java

```
package pack;

public class A

{

protected void msg()

{

System.out.println("Hello");

}

}
```

//save by B.java

```
package mypack;

import pack.*;

class B extends A

{

  public static void main(String args[])

  {

   B obj = new B();

   obj.msg();
  }
}
```

Output:-

```
Hello
```

Public:-

The access level of a public modifier is everywhere. It is accessible everywhere. It has the widest scope among all other modifiers. It can be accessed from within the class, outside the class, within the package and outside the package.

Example:-

//save by A.java

```
package pack;
public class A
{
public void msg()
{
System.out.println("Hello");
}
}
```

//save by B.java

```
package mypack;
import pack.*;
class B
{
  public static void main(String args[])
{
   A obj = new A();
   obj.msg();
  }
}
```

Output:- Hello

Non-access modifiers:-

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc.

Java provides a number of non-access modifiers to achieve many other functionalities.

- The static modifier for creating class methods and variables.

- The final modifier for finalizing the implementations of classes, methods, and variables.
- The abstract modifier for creating abstract classes and methods.
- The synchronized and volatile modifiers, which are used for threads.

Java Garbage Collection:-

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.

Java objects are created on the heap, which is a section of memory dedicated to a program. When objects are no longer needed, the garbage collector finds and tracks these unused objects and deletes them to free up space. Without garbage collection, the heap would eventually run out of memory, leading to a runtime **OutOfMemoryError**.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection:-

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

- It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.
- By finalize () method and gc() method

1) By nulling a reference:

Example:-

```
Employee e=new Employee();

e=null;
```

2) By assigning a reference to another:

Example:-

```
Employee e1=new Employee();

Employee e2=new Employee();

e1=e2;   //now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

Example:-

```
new Employee();
```

finalize() method:-

The finalize() method is called the finalizer.

Finalizers get invoked when JVM figures out that this particular instance should be garbage collected. Such a finalizer may perform any operations, including bringing the object back to life.

The main purpose of a finalizer is, however, to release resources used by objects before they're removed from the memory. A finalizer can work as the primary mechanism for clean-up operations, or as a safety net when other methods fail.

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.

The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

This method is defined in Object class as:

```
protected void finalize()

{

}
```

gc() method:-

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Example:-

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

Simple Example of garbage collection in java:-

```
public class TestGarbage
{
 public void finalize()
{
System.out.println("object is garbage collected");
}
 public static void main(String args[])
{
  TestGarbage tg=new TestGarbage();
  TestGarbage tg=new TestGarbage();
  tg=null;
  tg=null;
  System.gc();
 }
}
```

Output:-

```
object is garbage collected
object is garbage collected
```