26/02/11

Garbage Collection

1) Introduction.
2) Various ways to make an object eligible for G.C.
3) The methods for requesting JVM to run garbage Collector.
4) finalization..

Garbage Collector :->

→ In Old languages like C++, Creation & destruction of object is responsibility of programmer only.

→ Usually programmer taking very much care while Creating objects & his neglecting destruction of useless objects.. due to this negleetance at Second certain point of time for the Creation of new object sufficient memory may not be available & entire program will be collaps due to memory problems.

→ 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 assistent which is always running in the background for destruction of useless objects. due to this assistant the chance of failure java program with memory problem is very rare. this assistant is nothing "Garbage Collector".

→ Hence, the main objective of Garbage Collector is -to "destroy useless objects".

The Various Ways to make an object eligible for G.C :-

→ Eventhough programmer is not responsible to destroy useless objects, it is always a good programming practice to make an object eligible for G.C if it is no longer required.

→ An object is said to be eligible for G.C, if it doesn't contain any references.

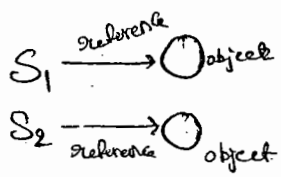
→ The following are various possible ways to make an object eligible for G.C.

(i) nullyfying the reference variable :-

→ If an object is nolonger required then assign null to all its references, then automatically that object eligible for G.C.

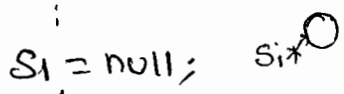
Ex!(1)

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

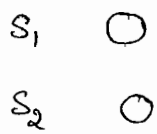


No objects eligible for G.C →

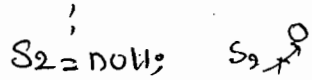
```
s1 = null;
```



one object eligible for G.C →




```
s2 = null;
```



two objects eligible for G.C →

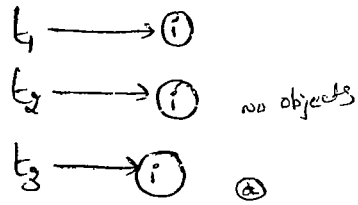
4) Island of isolation :-

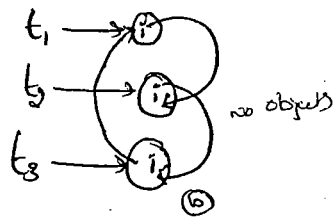
Ex:-

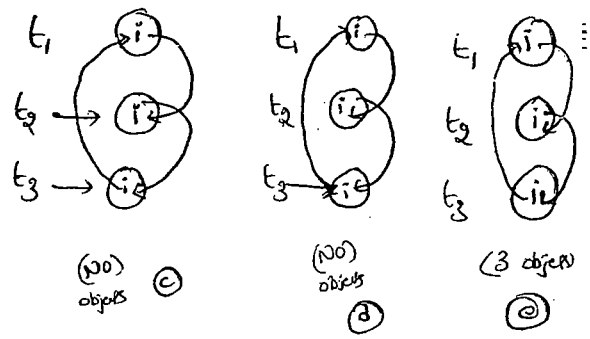
```
Class Test
{
    Test i;
    P.S.V. main(String[] args)
    {
    Test t1 = new Test();
    Test t2 = new Test();
    Test t3 = new Test();
    !
    t1.i = t2;
    t2.i = t3;
    t3.i = t1;
    :
    t1 = null;
    :
    t2 = null;
    :
    t3 = null;
    }
}
```

No objects eligible for G.C

3 objects eligible for G.C







Note:-

→ If an object doesn't have any reference then it is always eligible for Garbage Collector.

→ Eventhough object having the reference still it is eligible for G.C sometimes (Island of isolation)

The methods for requesting JVM to Run Garbage Collector:-

→ When ever we are making an object eligible for G.C it may not be destroyed by G.C immediately when ever JVM runs garbage Collector then only that object will be destroyed.

→ We Can request JVM to run garbage collector, programatically wheather JVM accepts over request are not there is no garantee.

→ The following are various ways for requesting JVM to run G.C.

(1) By System class :-

→ System class Contains a Static method G.C, for this

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

(2) By Runtime class :-

→ By using runtime object a Java application Can Communicate with JVM

→ Runtime class is a Singleton class hence we Can't Create Runtime object by using Constructor.

→ we can Create a Runtime object by using factory method getRuntime()

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

→ Once we got Runtime object we Can apply the following methods on that object.

(a) freeMemory() returns freememory in the heap,

(b) totalMemory() " total a the Heap (Heapsize)

(c) gc() → for requesting JVM to Run garbage Collector,

Ex:-
```
class RuntimeDemo
{
  p.s.v.main(String[] args)
  {
    Runtime r = Runtime.getRuntime();
    S.o.pln(r.totalMemory());
    S.o.pln(r.freeMemory());
    for(int i=1; i<=10000; i++)
    {
      Date d = new Date();
      d=null;
    }
    S.o.pln(r.freeMemory());
    r.gc();
    System.out.println(r.freeMemory());
  }
}
```

d → ○

d ○

Q) Which of the following is the properway of requested JVM to run g.c()?

1) System.gc(); (System is Static method)

X 2) Runtime.gc(); (Runtime is instance method)

X 3) (new Runtime()).gc(); (gc is applicable only Static method)

✓ 4) Runtime.getRuntime().gc();

Note:- gc() present in the System class is a Static method, where as gc() present in the Runtime class is instance method & recommended to use System.gc();

finalization :-

→ Just before destroying any object, garbage collector always calls finalize() method to perform clean-up activities on that object.

→ finalize() method declare in object class with the following declaration.

```
protected void finalize() throws Throwable.
```

Case(1):-

→ Garbage collector always calls finalize() on the object which is eligible for G.c just before destruction, then the corresponding class finalize() will be executed. If String object eligible for G.c then String class finalize() will be executed. but not Test class finalize method.

Ex!.

```
class Test
{
  p.s.v.m(String[] args)
  {
    String s = new String("durga");
    s = null;
    System.gc();
    System.out.println("end of main");
  }
  public void finalize()
  {
    S.o.pln("finalize method called");
  }
}
```

O/P :- end of main

→ In the above Example String object is eligible for g.c. Hence String class finalize() method got executed which has Empty implementation.

→ If we are replacing String object with Test object, then Test class finalize() will be executed.

→ In this case the O/p is ① finalize method called
End of main (or)
② End of main
finalize method called.

Case 2 :-

→ we can call finalize() Explicitly in this case it will be executed Just like a normal method call & Object won't be destroyed.

→ Just Before destruction of an object G.c always call finalize().

Ex:

```
Class Test
{
 p.s.v.m (String [] args)
 {
   Test t = new Test();
     t.finalize();
     t.finalize();
     t=null;
     System.gc();
     S.o.pln("End of main");
 }
 public void finalize()
 {
  S.o.pln("finalize method Called");
}}
```

O/p:

```
finalize method Called
finalize method Called
end of main
finalize method Called
```

→ In the above program finalize() got Executed 3 times, 2 times Explicitly by the programmer & one time by the Garbage Collector.

Note:-

→ Before destruction of Servelet object Web Container always Calls destroy method, to perform clean-up activities. Bcoz

→ It is possible to Called destroy() Explicitly from init() & Service() in this case it will be Executed Just like a normal method Call and Servelet Object won't be destroyed.

Case(3):-

→ If we are Calling finalize() Explicitly & while executing that finalize() if any Exception raised & uncaught, then the program will be terminated abnormally.

→ If G.C calls finalize() & while Executing that finalize(), if any Exception raised is uncaught (no Corresponding catch block) then Jvm Simply ignores that uncaught Exception & rest of the program will be executed normally

Eg:-
```
class Test
{
    p.s.v.m (String[] args)
    {
        Test t = new Test();
        t.finalize();   ---> line①
        t = null;
        System.gc();
        S.o.pln("end of main");
    }
```

```
public void finalize()
{
S.o.println("finalize method called");
S.o.println(10/0);
}
}
```

→ If we are not Comment Line①, then we are Calling the finalize() Explicitly and the program will be terminated abnormally.

→ If we are Commenting Line①, then G.c calls finalize() & the raised A.E is ignored by JVM. Hence in this Case the o/p is

o/p: End of main!
finalize method Called.

Q) Which of the following Statement is True?

×1) while executing finalize() all exceptions are ignored by JVM.

✓2) while " " only uncaught exceptions (no Catch block) ignored by JVM.

Conclusion:

→ on any object G.c calls finalize() only once.

[Note:
→ The Behaviour of G.C is vendor dependent & hence we can't expect Explicitly because of this we can't answer]

Eg:

```
Class finalizeDemo
{
 Static finalizeDemo s;
 p.s.v.m(String[] args) throws Exception
 {
  finalizeDemo f = new finalizeDemo();
   S.o.pln(f.hashCode());
   f=null;
   System.gc();
  Thread.sleep(5000);
  System.out.println(s.hashCode());
  S=null;
  System.gc();
  Thread.sleep(5000);
  S.o.pln("End of main method");
 }
 public void finalize()
 {
   S.o.pln("_finalize method Called");
   S=this;
 }
}
```

O/P:-

```
4072869
finalize method Called
4072869
  End of main method.
```

Note:- The behaviour of the G.c is vendor dependent & hence we can't expect exactly because of this we can't answer the following questions exactly.

① When JVM runs G.c exactly.

② What is the Algorithm following by g.c. ?

③ In which order G.c destroys the Objects.

④ Whether G.c destroys all eligible objects or not. e.t.c

Note:- We can't tell exact algorithm followed by G.c, but most of the cases it is mark & sweep Algorithm.

Memory leaks:-

→ If an object having the Reference then it is not eligible for G.c, eventhough we are not using that object in our program still it is not destroyed by the G.c.. Such type of object is called "Memory leaks". (i.e, Memory Leak is a useless object which is nt eligible for G.c)

→ We can resolve memory Leaks by making useless objects for G.c explicitly & by invoking G.c programmatically.

JProbe
IBM Tivoli
HP J meter

these are monitoring tools for memory Leak.

(20) Assertions (1.4 version)

(1) Introduction

* (2) Assert as Key-word & identifier

(3) Types of assert Statements

(4) Various Runtime flags

(5) Appropriate & Inappropriate use of assertions

(6) Assertion Errors.

Assertions :-

→ Very Common way of debugging is using S.o.p Statements. But the problem with S.o.p's is after fixing the problem Compulsory we should delete these S.o.p's otherwise these S.o.p's will be executed at runtime and effects performance & disturbes logging

→ To resolve this problem Sun people introduced Assertions Concept in 1.4 version. Hence the main objective of assertions is to perform debugging

→ The main Advantage of assertions over Sop is after fixing the Problem it is not required to delete assert Statements because assertions will be disabled automatically at runtime. based on our requirement we can enable & disable assert Statements & Bydefault assertions are disabled.

→ Assertions Concept is applicable for developement & test environment But not for production Environment.

Assert as a Keyword & identifier :-

→ Assert keyword introduced in 1.4 version, Hence from 1.4 version onwards we can't use assert as identifier. But Before 1.4 we can use assert as identifier

Ex:-

```
class Test
{
  p.s.v.m(String[] args)
  {
    int assert=10;
    S.o.pln(assert);
  }
}
```

✗ 1) Javac Test.java

C.E:- as of release 1.4, 'assert' is a keyword, and may not be used as an identifier

Use -Source 1.3 or lower, to use 'assert' as an identifier.

✓ 2) Javac -Source 1.3 Test.java

Java Test ↵

↵

10

Types of Assert Statements :-

→ There are 2 types of Assert Statement

(1) Simple version

(2) Augmented version

(1) Simple Version :-

→ assert(b); b → should be boolean-type

→ If b is true, then our assertion satisfied & rest of the program will be executed normally.

→ If b is false, then our assertion fails the program will be terminated by raising runtime Exception saying assertionError. So, that we can able to fix the problem.

Ex!-

```
Class Test
{
 p.s.v.m(String[] args)
 {
   int x=10;
   ;;;
   assert(x>10);
   ;;;
   S.o.pln(x);
 }
}
```

① Javac Test.java ✓

② Javae Test ✓

10

*③ Java -ea Test ←

R.E: AssertionError

(2) Augmented Version :-

→ we can Augment some discription by using augmented version to the AssertionError.

```
assert(b) : d;
```

b → should be booleantype

d → any description, can be any type. but recommend to use String type.

Ex:-

```
class Test
{
  P.S.V.m(String[] args)
  {
    int x=10;
    .....
    assert(x>10) : "Here x value should be >10 but it is not";
    .....
    S.o.pln(x);
  }
}
```

① Javac Test.Java ↵

② Java Test ↵
10

③ Java -ea Test ↵

R.E: AssertionError: Here x value should be >10 but it is not.

Conclusion(1) :-

```
assert(e1) : e2;
```

→ e2 will be evaluated iff e1 is false. i.e if e1 is True, then e2 won't be evaluated.

Ex:-
```
class Test
{
p.s.v.m(String[] args)
{
int x=10;
=
assert(x==10):++x;
=
S.o.p.ln(x);
}}
```

```
assert(x>10): ++x;
```

✓ javac Test.java ↵

✓ java Test ↵
10

✓ java -ea Test ↵
10

```
javac Test.java

java Test
10

java -ea Test
```
R.E: AssertionError: 11

Conclusion(1):-

```
assert(e1) : e2 ;
```

→ As e2 we can take a method call also but void-type method calls are not allowed.

Ex:-
```
class Test
{
p.s.v.m(String[] args)
{
int x=10;
=
assert(x>10) : m1();
=
S.o.pln(x);
}
public static int m1()
{
return 8888;
}}
```

✓ javac Test.java ↵

✓ java Test ↵
10

```
java -ea Test
```
R.E: AssertionError: 8888

→ If m1() return type is void, then we will get CompileTimeError saying "void type not allowed here."

4) Various Runtime flags :-

① -ea :- To enable assertions in Every non-System class

② -enableassertions :- It is Exactly Same as -ea

③ -da :- To disable assertions in Every non-System class

④ -disableassertions :- Same as -da

⑤ -esa :- To enable assertions in every System class.

⑥ -enableSystemassertions :- It is exactly Same as -esa.

⑦ -dsa :- To disable assertions in Every System class.

⑧ -disableSystemassertions :- It is Same as -dsa.

Ex(1) :-

Java -ea -esa -da ~~-dsa~~ -esa -ea -dsa

NON System class	System class
✓	✓
X	✓
✓	X

→ We can use these flags in together & all these flags executed from Left to right.

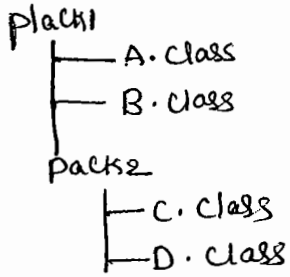
Ex(2) :-

① Java -ea:pack1.A

② Java -ea:pack1.B -ea:pack1.pack2.D

③ Java -ea -da:pack1.B

Ex 2:-



→ To enable assertions in only A class

① java -ea:pack1.A

→ To enable assertions in Both B & D classes

java -ea:pack1.B -ea:pack1.pack2.D

→ To enable assertions in every non-System class Except B

java -ea -da:pack1.B

→ To enable assertions in every class of pack1 & it's Sub packages

java -ea:pack1...

→ To enable assertions in every where with in pack1 Except pack2.

java -ea:pack1... -da:pack1, pack2...

5) Appropriate & Inappropriate use of assertions :-

1) It is always Inappropriate to mix programming logic with assert Statement because there is no garntee of execution of assert Statement at runtime.

Ex:-

```
withdraw(int x)
{
 if(x<100)
 {
  throw new IAE();
 }
}
```

proper way

```
withdraw(int x)
{
 assert(x>=100);
}
```

Inproper way

2) In our program if there is any place where the control not allowed to reach then it is the best place to use assert statement.

Ex:-
```
switch(x)
{
  case1: S.o.pln("JAN");
         break;
  Case2: S.o.pln("feb");
         break;
    :
  Case12: S.o.pln("Dec");
          break;
  default:
     assert(false);
  }
```
→ R.E: A.E can be displayed.

3) It is always Inappropriate to use assertions for validating public method arguments.

4) It is always Appropriate to use assertions for validating private method arguments

5) It is always Inappropriate to use assertions for validating Command-Line arguments because these are arguments to public main().

6) AssertionError :-

→ It is the child class of Error & Hence it is unchecked.

→ It is legal to catch AssertionError by using try-catch but it is stupid kind of activity

Ex:-
```
class Test
{
p.s.v.m(String[]args)
```

Ex!-

```
class Test
{
 p.s.v.m(String[] args)
 {
  int x=10;
   ::
  try
  {
   assert (x>10);
  }
  Catch(AssertionError e)
   {
    S.o.pln("I am Stupid ... b'z I am Catching
   }                                    AssertionError");
   S.o.pln(x);
  }
}
```

Note!.

→ It is possible to enable assertions either class wise or packagewise