17. Assertions

1) Introduction
2) assert as keyword & identifier
3) Types of assert statements
4) Various possible runtime flags
5) Appropriate & inappropriate use of assertions.
6) AssertionError.

1. Introduction :—

→ Very common way of debugging is usage of S.o.p statements, but the problem with S.o.p is after fixing the bug compulsory we have to delete extra added S.o.p's, o.w. these will be executed at runtime which creates performance problems and disturbs server logging.

→ To overcome these problems SUN people introduced Assertions concept in 1.4 version.

→ The main advantage of assertions over S.o.p's is after fixing the bug we are not required to delete assert statements becoz they won't be executed by default at runtime.

→ Based on our requirement we can enable or disable assertions and by default assertions are disable.

→ Usually we can perform debugging either in Development or Test environment, but not in production.

→ Hence assertions concept applicable only for development & test environments, but not for Production.

→ Hence the main objective of assertions is to perform debugging as alternative to S.o.p's.

2. assert as keyword and identifier:—

→ assert keyword introduced in 1.4 version.

→ Hence from 1.4 version onwards we can't use assert as identifier.

Ex:
```
class Test
{
  p s v m(—)
  {
    int assert=10;
    S.o.p(assert);
  }
}
```

javac Test.java↵ ✗

CE: 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.

javac -source 1.3 Test.java↵ ✓

compiles fine with warnings

java Test↵ ✓

Op: 10

✗ javac -source 1.5 Test.java

✗ javac -source 1.4 Test.java

✓ javac -source 1.3 Test.java

✓ javac -source 1.2 Test.java

Note: ①: We can compile a Java program according to a particular version by using -source option.

② If we are using assert as identifier and if we are trying to compile according to old versions (1.3 or lower) then the code compiles fine but with warnings.

3. Types of assert statements :—

→ There are 2 types of assert statements.

1. Simple version
2. Augmented version.

1. Simple version :—

Syntax :— `assert(b);` → (should be boolean)

if b is true then our assumption satisfy and hence rest of the program will be executed normally.

if b is false then our assumption fails & hence some where something goes wrong due to this the program will be terminated abnormally by raising AssertionError.

Ex:

```
class Test
{
  P s v m(-)
  {
    int x=10;
     :
     :
    assert(x>10);
     :
     :
    S.o.p(x);
  }
}
```

javac Test.java ↵

java Test ↵

O/P : 10

java -ea Test ↵

RE : AssertionError

Note:- By default assertions are disable, but we can enable assertions by using -ea option.

2). Augmented version:-

→ We can augment(append) some description with AssertionError by using Augmented version.

Syntax:- `assert(b):e;`

b should be boolean type

e can be any type

Ex:
```
class Test
{
  p s v m(-)
  {
    int x=10;
     :
    assert(x>10):"Here x value should be x>10 but it is not";
     :
    S.o.p(x);
  }
}
```

✓ javac Test.java↵

✓ java Test↵

o/p(10

java -ea Test↵

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

Conclusions:-

1. assert(b):e;

If 'b' is true then second argument won't be evaluated i.e., if 'b' is false then only second argument will be evaluated.

Ex:
```
class Test
{
  P s v m(-)
  {
    int a=10;
    ;;;;;;;;;
    assert(a==10):++a;
    ;;;;;;;;;
    S.o.p(a);
  }
}
```

javac Test.java↵
java -ea Test↵

O/P = 10

→ If we replace assert line as assert(a>10):++a; then we will get RE saying, AssertionError.

2. assert(b):c;

For the second argument we can take method call, but void return type method call is not allowed.

Ex:
```
class Test
{
  P s v m(-)
  {
    int a=10;
    ;;;;;;;;;
    assert(a>10):m1();
    ;;;;;;;;;
    S.o.p(a);
  }
  P s int m1()
  {
    return 999;
  }
}
```

javac Test.java↵

java Test↵

O/P : 10

java -ea Test↵

RE : AssertionError : 999

→ If m1() method return type is void then we will get CE saying,

CE : void type not allowed here

4. Various possible runtime flags :—

1) -ea/-enableassertions :- To enable assertions in every non-system classes(our own classes).

2) -da/-disable assertions :- To disable assertions in every non-system classes.

3) -esa/-enablesystemassertions :— To enable assertions in every system class (predefined classes).

4) -dsa/-disablesystemassertions :— To disable assertions in every system class.

Note :- We can use above flags simultaneously then JVM will consider these flags from left to right.

Ex: java -ea -esa -da -ea -esa -da -dsa -ea Test

Non-system	System
✓	✓
X	✓
✓	X
X	
✓	

Case Study :-

1). To enable assertions only in B class.

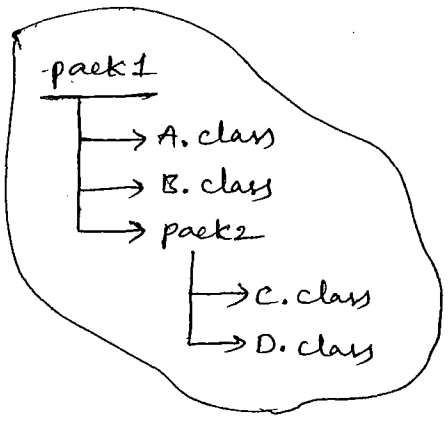
java -ea:paek1.B

2) To enable assertions only in B and D classes.

java -ea:paek1.B -ea:paek1.paek2.D

3) To enable assertions every where inside paek1.

java -ea:paek1...



4) To enable assertions every where inside pack1 except pack2 classes.

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

5) To enable assertions every where inside pack1 except B class.

java -ea:pack1... -da:pack1.B.

Note:- We can enable or disable assertions either class wise or package wise also.

5. Appropriate and Inappropriate use of assertions:—

1). It is always inappropriate to mix programming logic with assert statements becoz there is no guarantee for the execution of assert statement always at runtime.

Ex:
```
withdraw(double amount)
{
  if(amount < 100)
      throw new IllegalArgumentException();
  else
      process request
}
```
Appropriate

```
withdraw(double amount)
{
  assert(amount>=100);
  process request
}
```
Inappropriate

2) While performing debugging in our program if there is any place where control is not allowed to reach that is the best place to use assertions.

Ex:
```
switch(x)
{
  case 1: S.o.p("JAN");
          break;
  case 2: S.o.p("FEB");
          break;
  default: assert(false);
}
```
Should be a valid month number

RE: AssertionError

3) It is always inappropriate to use assertions for validating public method arguments becoz outside person doesn't aware whether assertions are enabled or disabled in our system.

4) It is always appropriate to use assertions for validating private method arguments becoz local person can aware whether assertions are enabled or disabled in our system.

5) It is always inappropriate for validating command line arguments by using assertions becoz these are arguments to public main(-) method.

Ex:
```
class Test
{
   public static void main(String[] args)
   {
      assert(args.length>=3)
   }
}
```
Inappropriate usage of assert statement

6) AssertionError :—

→ It is the child class of Error and hence it is unchecked.

→ It will be raised whenever assert statement fails.

→ Eventhough it is legal to catch AssertionError but it is never recommended. It is a stupid kind of programming practice.

Ex:
```
class Test
{
  P S V m(-)
  {
    int a=10;
    |
    |
    try
    {
      assert(a>10);
    }
```

```
        catch(AssertionError e)
        {
          S.o.p("I m stupid, Becoz I m catching AssertionError");
        }
        :
        :
        S.o.p(a);
    }
}
```

Ex①:

```
class One
{
  p s v m(-)
  {
    int assert=10;
  }
}
class Two
{
  p s v m(-)
  {
    assert(false);
  }
}
```

(identifier) (keyword)

① javac -source 1.3 One.java↵
② javac -source 1.3 Two.java↵
③ javac -source 1.4 One.java↵
④ javac -source 1.4 Two.java↵

Ex②:

```
class Test
{
  p s v m(-)
  {
    boolean assertOn=false;
    assert(assertOn) : "assert on";
    if (assertOn==false)
    {
      S.o.p("assert on");
    }
  }
}
```

① java Test↵

O/P : assert on

② java -ea Test↵

RE : AssertionError : assert on.