Assertions

1. Introduction
2. "assert" as a keyword & Identifier
3. Types of assert statement
4. various possible Runtime flags
5. Appropriate & Inappropriate use of assertions
6. AssertionError

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 S.O.P statement otherwise this S.O.P will be executed at runtime for every request, which creates performance problems & disturbs server logs.

→ To overcome this problem SUN people introduced "Assertions" concept in ~~1.4~~ 1.4 V.

→ The main advantage of "Assertions" when compared with S.O.P is after fixing the bug, we are not required to remove "assert" statement, bcz they won't be executed by default at run-time. Based on our requirement we can enable & disable Assertions & by default Assertions are disabled.

→ Hence, the main objective of Assertions is to perform Debugging.

→ Usually, we can perform debugging in Development & Test environment but not in production environment, hence Assertion concept applicable for Development & Test environment but not for production.

"assert" as keyword & Identifier

→ "assert" keyword introduced in 1.4 V, hence from 1.4 V onwards we can't use "assert" as identifier otherwise we will get C.E.

Ex.

```
class Test
{
    p s v main(Str[] )
    {
        int assert = 10;
        S.o.p(assert);
    }
}
```

javac Test.java

C.E. as of release 1.4, "assert" is a keyword, and may not be used as identifier

(use -source 1.3 or lower to use assert as an identifier)

javac -source 1.3 Test.java

Code compiles, but with warning

java Test

10

	C.E.
javac -source 1.2 Test.java	NO
javac -source 1.3 Test.java	NO
javac -source 1.4 Test.java	yes
javac -source 1.5 Test.java	yes

Note :- (1) If we are using "assert" as identifier & If we are trying to compile according to old version (1.3 or lower) then code compiles fine but with warning.

(2) We can compile a java program according to a particular version by using -source option.

Types of Assert statements

There are two types of assert statements

1) Simple version
2) Augmented version

Simple version :-

- Syntax :-

assert(b);

b should be boolean Type

If b is true : our assumption satisfies & hence rest of the program will be executed normally.

→ Ex.

```
class Test
{
    p s v main(Str...)
    {
        int assert = 10;
        S.o.p(assert);
    }
}
```

javac Test.java

C.E. as of release 1.4, "assert" is a keyword, and may not be used as identifier

(use -source 1.3 or lower to use assert as an identifier)

javac -source 1.3 Test.java

code compiles, but with warning

java Test

10

	C.E.
→ javac -source 1.2 Test.java	NO
javac -source 1.3 Test.java	NO
javac -source 1.4 Test.java	yes
javac -source 1.5 Test.java	yes

→ Note :- (1) If we are using "assert" as identifier & if we are trying to compile according to old version (1.3 or lower) then code compiles fine but with warning.

(2) we can compile a java program according to a particular version by using -source option.

Types of Assert statements

→ There are two types of assert statements

1) Simple version
2) Augmented version

→ Simple version :-

- Syntax :-

assert(b);

b should be boolean Type

If b is true : our assumption satisfies & hence rest of the program will be executed normally.

it b is false :

our assumption fails i.e somewhere something goes wrong hence the program will be terminated abnormally by rising AssertionError. once we get AssertionError we can analyze the code & we can fix the program.

→ Ex.

```
class Test
{
    p s v main(stm)
    {
        int x=10;
        :
        assert(x>10);
        :
        s.o.p(x);
    }
}
```

javac Test.java ↵

java Test ↵
o/p : 10

java -ea Test ↵
R.E. AssertionError

→ Note : By default "assert" statements won't be executed, bcz Assertions are disabled by default but we can enable Assertions by using `-ea` option

→ Augmented version :-

- we can augment some description with AssertionError by using Augmented version
- syntax :-

`assert (b) : e ;`

b should be boolean type

e can be any type but mostly it is string type

- Ex.

```
class Test
{
    p s v main(stm)
    {
        int x=10;
        :
```

```
        assert(x>10):"Here x value
        should be >10 but it is not";
        :
        S.o.p(x);
    }
}
```

javac Test.java ↵ (1)

java Test ↵
10

java -ea Test ↵
R.E. AssertionError: Here x value should be > 10 but it is not

→ **Conclusion 1 :-**

- `assert(b):e;`
- e will be executed if and only if 1st argument(b) is false, i.e if 1st argument(b) is true then 2nd argument(e) won't be evaluated.

Ex.
```
class Test
{
    p s v main(str)
    {
        int x=10;
        :
        assert(x==10):++x;
        :
        S.o.p(x);
    }
}
```

javac Test.java ↵ (1)

java Test ↵
10

java -ea Test ↵
10

→ **Conclusion 2 :-**

- `assert(b):e;`
- For the 2nd argument(e) we can take method call but void return type method call is not allowed otherwise we will get C.E.

• Ex. class Test
```
{
    p s v main(Sm)
    {
        int x=10;
        :
        assert(x>10):m1();
        :
        s.o.p(x);
    }
    public static int m1()
    {
        return 777;
    }
}
```
javac Test.java ↵ (1)

java Test ↵
10

java -ea Test ↵
R.E AssertionError : 777

• If m1() method return type is void then we will get C.E. "void" type not allowed here.

Note: Among 2 versions of Assertion it is recommended to use 2nd version, bcz its provide more info for debugging

Various possible Runtime flags

(1) -ea | -enableassertions

To enable assertions in every Non-System class (our own classes)

(2) -da | -disableassertions

To disable assertions in every Non-system class.

(3) -esa | -enablesystemassertions

To enable assertions in every system class (pre-defined classes)

(4) -dsa | -disablesystemassertions

To disable assertions in every system class.

Note: we can use above flags simultaneously in single command, then JVM will consider this flag from left to right

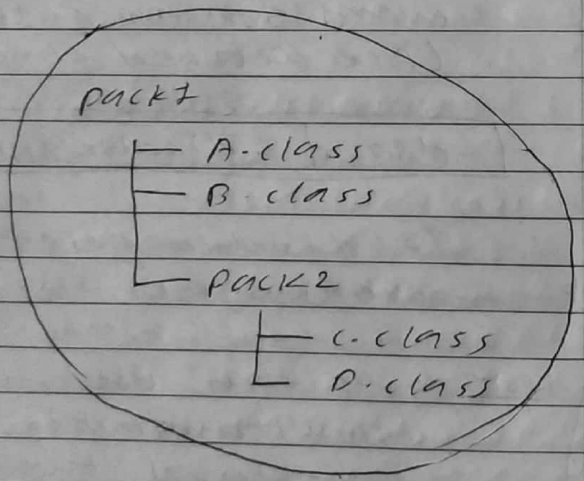
Ex. java -ea -esa -ea -dsa
-da -esa -ea -dsa
Test

- Non-system System

✓ ✓
✓ ✗
✗ ✓
✓ ✗

- At the end, assertions will be enabled in every Non-system class & disable in every System class

~) Case-study :



(1) To enable assertions only in B class

java -ea:pack1.B

(2) To enable assertions in both B and D classes

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

(3) To enable assertions in every class of pack1

java -ea:pack1...

(4) To enable assertions in every class of pack1 except B class

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

(5) To enable assertions in every class of pack1 except pack2 classes

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

Note : we can enable & disable assertions either class-wise (or) package wise also

✱ Appropriate & Inappropriate use of assertions

(1) It is always inappropriate to mix programming logic with assert statement, bcz there is no guarantee for the execution of assert statement alway at runtime (i.e client may haven't enabled assertions)

Ex.
```
public void withdraw (double amount)
{
    if (amount <= 100)
        throw new
            IllegalRequestException();
    else
        process request
}
```
Appropriate way ✓

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

(2) while performing Debugging, if there is any place where the control is not allowed to reach, there is the best place to use assertions

```
switch (x)        → should be a valid month number
{
    case 1:
        s.o.p ("JAN");
        break;
    case 2:
        s.o.p ("FEB");
        break;
    :
    case 12:
        s.o.p ("DEC");
        break;
    default:
        assert (false);
}
```
Appropriate way ✓

RE: AssertionError ←

(3) It is alway inappropriate for validating public method arguments by using assertions bcz outside person doesn't aware whether assertions are enabled / disabled in our system

- It is always appropriate for validating private method arguments by using assertions, bcz local person can aware whether assertions are enabled/disabled in our system.

- It is always inappropriate for validating command line arguments by using assertions, bcz these are arguments of main method which is public.

- Ex-

```
class Test
{
    int z=5;
    public void m1(int x)
    {
        assert(x>10);          Inappropriate (case-3)
        switch(x)
        {
            case 10:
                    S.O.P(10);
                    break;
            case 11:
                    S.O.P(11);
                    break;
            default:
        } Appropriate  assert(false);
    }
    private void m2(int x)
    {
        assert(x<10);          Appropriate (case-3)
    }
    private void m3()
    {
        assert(m4());          Inappropriate (case-1)
    }
    private boolean m4()
    {
            z=6;
            return true;
    }
}
```

Ex.

```
class One
{
    P S V main(Str[] )
    {
        int assert = 10;
        S.O.P(assert);
    }
}
class Two
{
    P S V main(Str[] )
    {
```

```
        int x=10;
        assert(x>10);
    }
}
```

O/P's

✓ (a) javac -source 1.3 One.java
(compiles with warning)

✗ (b) javac -source 1.4 One.java
(C.E)

✗ (c) javac -source 1.3 Two.java
(C.E.)

✓ (d) javac -source 1.4 Two.java
(compiles without warning)

⇒ Ex.

```
class Test
{
    p s v main(Str)
    {
        boolean assertOn=false;
        assert(assertOn):
            assertOn=true;
        if(assertOn)
        {
            S.O.P("assertOn");
        }
    }
}
```


O/P:

If assertions are not enabled

NO O/P

If assertions are enabled

R.E. AssertionError: true

⇒ Ex. class Test

```
{
    p s v main(Str)
    {
        boolean assertOn=true;
        assert(assertOn):
            assertOn=false;
        if(assertOn)
        {
            S.O.P("assertOn");
        }
    }
}
```

O/P:

If assertions are enabled
assertOn

If assertions are not enabled
assertOn

AssertionError

- It is a child class of Error & hence it is unchecked.
- If assert statement fails (i.e argument is false) then we will get AssertionError
- Even though it is legal to catch AssertionError, but it is not a good programming practice
- Ex:

```
class Test
{
    p s v main(str)
    {
        int x = 10;
        ;
        {
        try
        {
R.E: AssertionError <--- assert (x > 10);
        }
        catch (AssertionError e)
        {
            S.O.P ("I am stupid,
                    bcz i am catching
                    AssertionError");
        }
        super(x);
    }
}
```

Note: In the case of web application if we run Java program in debug mode automatically assert statement will be executed.

End of Assertions

Start of Development