Assertions

Agenda

1. **Introduction**
2. **Assert as keyword and identifier**
3. **Types of assert statements**
   i. **Simple version**
   ii. **Argumented version**
4. **Various runtime flags**
5. **Appropriate and Inappropriate use of assertions**
6. **AssertionError**

Introduction:

1. **The most common way of debugging is uses of sops. But the main disadvantage of sops is after fixing the bug compulsory we should delete extra added sops otherwise these sops also will be executed at runtime which impacts performance of the system and disturbs logging mechanism.**
2. **To overcome these problems sun people introduced assertions concept in 1.4 versions.**
3. **The main advantage of assertions over sops is based on our requirement we can enable or disable assertions and by default assertions are disable hence after fixing the bug it is not required to delete assert statements explicitly.**
4. **Hence the main objective of assertions is to perform debugging.**
5. **Usually we can perform debugging either in development environment or Test environment but not in production environment hence assertions concept is applicable for the development and test environments but not for the production.**

Assert as keyword and identifier:

assert keyword is introduced in 1.4 version hence from 1.4 version onwards we can't use assert as identifier but until 1.3 we can use assert as an identifier.

Example:

```
class Test
{
        public static void main(String[] args)
        {
                int assert=10;
                System.out.println(assert);
        }
}
```

Output:

javac Test.java(invalid)

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(valid)

The code compiles fine but warnings will be generated.

java
Test 10

Note: It is always possible to compile a java program according to a particular version by using -source option.

Types of assert statements:

There are 2 types of asset statements.

1. **Simple version**
2. **Argumented version**

Simple version:

Syntax: assert(b);//b should be boolean type.

If 'b' is true then our assumption is correct and continue rest of the program normally.
If 'b' is false our assumption is fails and hence stop the program execution by raising assertion error.

Example:

```
class Test
{
       public static void main(String[] args)
       {
              int x=10;
              ;;;;;;;;;
              assert(x>10);
              ;;;;;;;;;
              System.out.println(x);
       }
}
```

```
Output:
javac Test.java
java Test
10
java -ea Test(invalid)
R.E: AssertionError
```

Note: By default assertions are disable and hence they won't be executed by default we have to enable assertions explicitly by using -ea option.

Argumented version:

By using argumented version we can argument some extra information with the assertion error.
Syntax: assert(b):e;
'b' should be boolean type. 'e' can be any type.

```
Example:

class Test
{
	public static void main(String[] args)
	{
		int x=10;
		;;;;;;;;;
		assert(x>10):"here x value should be >10 but it
is not";
		;;;;;;;;;
		System.out.println(x);
	}
}
Output:
javac Test.java
java Test
10
java -ea Test(invalid)
R.E: AssertionError: here x value should be >10 but it is not
```

Conclusion 1:

assert(b):e;

'e' will be evaluated if and only if 'b' is false that is if 'b' is true then 'e' won't be evaluated.

```
Example:
class Test
{
	public static void main(String[] args)
	{
		int x=10;
		;;;;;;;;;
		assert(x==10):++x;
		;;;;;;;;;
		System.out.println(x);
	}
}
Output:
javac Test.java
java Test
10
java -ea
Test 10
```

Conclusion 2:

assert(b):e;

For the 2nd argument we can take method call also but void type method call not allowed.

Example:

```
class Test
{
	public static void main(String[] args)
	{
		int x=10;
		;;;;;;;;;
		assert(x>10):methodOne();
		;;;;;;;;;
		System.out.println(x);
	}
	public static int methodOne()
	{
		return 999;
	}
}
```

Output:

```
javac Test.java
java Test
10
java -ea Test
R.E: AssertionError: 999
```

If methodOne() method return type is void then we will get compile time error saying void type not allowed here.

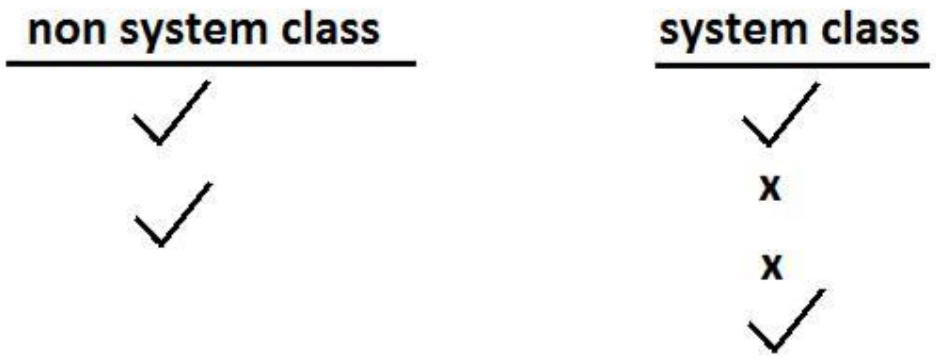
Various runtime flags:

1. **-ea: To enable assertions in every non system class(user defined classes). -enableassertions: It is exactly same as -ea.**
2. **-da: To disable assertions in every non system class. -disableassertions: It is exactly same as -da.**
3. **-esa: To enable assertions in every system class(predefined classes or application classes). -enablesystemassertions: It is exactly same as -esa.**
4. **-dsa: To disable assertions in every system class. -disablesystemassertions: It is exactly same as -dsa.**

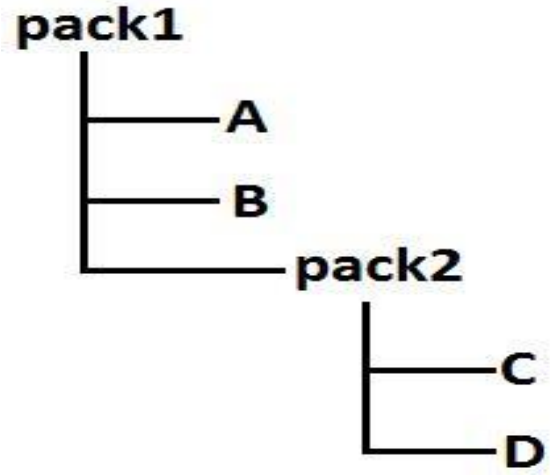
Example: We can use these flags in combination also but all these flags will be executed from left to right.

Example:

```
java -ea -esa -dsa -ea -dsa -esa Test
```



At the end in both system and non system class assertions are enabled. Example:



1. **To enable assertions only in B class. java -ea: pack1.B**
2. **To enable assertions only in B and C classes. java -ea: pack1.B -ea:pack1.pack2.C**
3. **To enable assertions in every class of pack1. java -ea:pack1**
4. **To enable assertions everywhere inside 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...**

It is possible to enable (or) disable assertions either class wise (or) package wise also.

<u>Appropriate and inappropriate use of assertions:</u>

It is always inappropriate to mix programming logic with assert statements, because there is no guaranty for the execution of assert statement always at runtime.

Example:

```
withdraw(double d)
{
    assert(d>=100);
        .
        .
}

inappropriate use of
assert statement.
```

```
withdraw(double d)
{
    if(d<100)
    {
        throw new IllegalArgumentException
    }
    else
    {
        withdraw code
    }
}
```

appropriate use of assert statement.

For validating public method arguments usage of assertions is always inappropriate, because outside person is not aware whether assertions are enabled or disabled in our local system.
While perform debugging if any place where the control is not allow to reach then that is the best place to use assertions.

Example:

```
switch(x)
      {
            case 1:
            System.out.println("Jan");
            break;
            case 2:
            System.out.println("Feb");
            break;
            case 3:
            System.out.println("Mar");
            break;
            case 12:
            System.out.println("Dec");
            break;
            default:assert(false);
      }
```

It is always inappropriate to use assertions for validating public method arguments.

It is always appropriate to use assertions for validating private method arguments.
It is always inappropriate to use assertions for validating command line arguments because these are arguments to public method main.

AssertionError:

1. **It is the child class of Error and it is unchecked.**
2. **Raised explicitly whenever assert statement fails.**
3. **Even though it is legal but it is not recommended to catch AssertionError.**

Example:

```
class Test {
public static void main(String[]
  args){ int x=10;
 try {
    assert(x>10);
       }
       catch (AssertionError e)  {
       System.out.println("not a good programming practice
to catch AssertionError");
       }
              System.out.println(x);
 }
}
Output:
javac Test.java
java Test
10
Not a good programming practice to catch
AssertionError 10
Example 1:
class One
{
       public static void main(String[] args)
       {
              int assert=0;
       }
}
class Two
{
       public static void main(String[] args)
       {
              assert(false);
       }
}
Output:
Javac -source 1.3 one.java//compiles with warnings.
Javac -source 1.4 one.java//compile time error.
Javac -source 1.3 Two.java//compile time error.
```

```
Javac -source 1.4 Two.java//compiles without
warnings. Example 2:
class Test
{
	public static void main(String[] args)
	{
		assert(args.length==1);
	}
}
```

Which two will produce AssertionError?

1) Java Test
2) Java -ea Test//R.E: AssertionError
3) Java Test file1
4) Java -ea Test file1
5) java -ea Test file1 file2//R.E: AssertionError
6) java -ea:Test Test file1

To enable the assertions in a particular class.

```
Example 3:
class Test
{
	public static void main(String[] args)
	{
		boolean assertOn=true;
		assert(assertOn):assertOn=true;
		if(assertOn)
		{
			System.out.println("assert is on");
		}
	}
}
Output: Java
Test Assert
is on Java -
ea Test
Assert is on
```

In the above example boolean assertOn=false then answer following questions.

```
Javac Test.java
Java Test
java -ea Test
R.E: AssertionError: true
```

Example 4:

```
class Test
{
    int z=5;
    public void stuff1(int x)
    {
        assert(x>0);  ---> Inappropriate
        switch(x)
        {
            case 2:
                x=3;
            default:
                assert(false);  ---> appropriate
        }
    }
}
```

because inside public method we are not using assert statements.

Example 5:

```
private void stuff2(int y)
    {
        assert(y<0);  ---- appropriate
    }
```

Example 6:

```
int z=5;
private void stuff3()
      {
            assert(stuff4()); ——— inappropriate
      }
      private boolean stuff4()
      {
            z=6;
            return false;
      }
```

Note: **Because assert statement changes the value of Z. By using assert statement we can not changes the value that is why it is inappropriate.**