SWEN221 Software Design and Engineering
10: Java Assert Keyword

Marco Servetto, David J. Pearce

VUW

Assert Keyword

- Added in Java 1.4
  Basic use is extremely simple:
  - **assert** ⟨*condition*⟩;
    Or
  - **assert** ⟨*condition*⟩ : ⟨*message*⟩;
- Disabled by default. Use `java -ea MyProgram`

```
void foo(A a){
  assert a!=null;
  ...
}
```

```
assert x!=null;
assert x>=0;

assert x!=null: "x can not be not null";
assert x>=0: "x can not be negative";
```

but also

```
assert isConsistent();
assert x!=null : ErrorHelpers.notNull("x","Reason");
```

Semantic

```
assert x!=null;
```

if `x` is **`null`** and assertions are enabled, then the semantic of the assertion is equivalent to simply

```
throw new AssertionError();//Unchecked Exception
```

```
assert x!=null : "msg";
```

if `x` is **`null`** and assertions are enabled, then the semantic of the assertion is equivalent to simply

```
throw new AssertionError("msg");//Unchecked Exception
```

Enable Assertions

- Command line:
  - `java -ea MyMainClass`
  - `java -ea -jar MyExecutablejarFile`
- Eclipse:
  - Run → Run configurations → select your configuration → arguments → vm arguments → write "-ea"

To enable assertions only in a package (and sub-packages)
`-ea:namePackage...` No space before the three dots!!!

Even if Eclipse is in debug mode, even if you are using Junit or similar tools, assertions are not automatically enabled.

DEMO

Quiz time

```
class A{
  public static boolean b;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac A.java

>> java A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Quiz time

```
class A{
  public static boolean b;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac -ea A.java

>> java A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Quiz time

```
class A{
  public static boolean b;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac A.java

>> java -ea A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Quiz time

```
class A{
  public static Boolean b;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac A.java

>> java A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Quiz time

```
class A{
  public static Boolean b;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac A.java

>> java -ea A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Quiz time

```
class A{
  public static Boolean b=true;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac A.java

>> java -ea A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Quiz time

```
class A{
  public static Boolean b=true;
  public static void main(String[]args){
    assert b;
    System.out.println("b is "+b);
    }
  }
```

```
>> javac A.java

>> java A
```

A `AssertionError`

B `"b is true"`

C `"b is false"`

D `"b is null"`

E `NullPointerException`

F *something else*

Contracts

Every method have, implicitly or explicitly, a contract.

If method is called over acceptable parameters, then a certain result is produced.

```
int factorial(int n) {
  if(n<1)return 1;
  return n*factorial(n-1);
}
```

Elements of a software system collaborate with each other on the basis of mutual obligations and benefits. The metaphor comes from business: client and supplier agree on a **contract**,
Similarly, if a method provides a certain functionality, it:
Expects a certain condition to be guaranteed on entry by any client that calls it: the method's precondition is an obligation for the client and a benefit for the supplier (the method itself), as it frees it from having to handle cases outside of the precondition.
Guarantee a certain property on exit: the methods's postcondition is an obligation for the supplier, and obviously a benefit for the client

Contracts

Modify the contract of the method, Make it simpler!

```
int factorial(int n) {
  assert n>=0;
  if(n<1)return 1;
  return n*factorial(n-1);
}
```

When using contracts, a supplier does not need to verify that the contract conditions are satisfied: the code should ”fail hard”, with contract verification being the safety net. The supplier throws an exception to inform the client that the precondition has been broken. Since the intended behaviour of each routine is clearly specified, the debugging of the contract behaviour is simplified.

Pre/Post conditions

Checks at the end of method execution are called **postconditions**
Is a behavioural constraint: ensures your code behave as expected

Checks at the start of the method execution are called **preconditions**
Is a usage limitation: ensures your code is called as expected

```
void methodName(...){
  assert precondition();try{
  ...DoSomething...
  }finally{assert postcondition();}
}
```

(p.s. doing useful checks in the middle of the method execution is a good idea, just there is no a special name)

How much code in assertions?

- Verification code is very important
- As for regular code, you can factorize and modularize it with methods, classes, libraries..
- An equilibrate program with assertions can contain up to 50% of verification code
- In addition to testing

Assertions - Libraries

- great for debug (much better than `System.err.println(..)`)
- cooperate with private state! (preserve valid program state)
- cooperate with documentation (active documentation)
  - make library usage safer
    library used in unacceptable way
    $\rightarrow$ AssertionError with an informative error message