Assertions

Introduction

- The assertion mechanism supported by Scala is very similar to the one we all know in Java.
- The `Predef` object includes the definition for `assert`, `assume` and `require`. The three methods are very similar. The three methods exist so we could use each and every one of them in the right context. Technically they work the same.

Introduction

- We will use the `require` method for testing a precondition. We will use this method to check a pre-condition the caller should have verified before calling our method.
- We will use the `assume` method for testing a static condition we expect to fulfill during the execution of our code.
- We will use the `assert` method for testing a condition we expect to be true following the execution of code we wrote.

The `assert` Method

- The assertions in Scala written as calls to the predefined method `assert`.

```
...
assert(condition)
...
```

- If the condition is false an `AssertionError` is thrown.

The `assert` Method

- Calling the assert method we can also pass over a textual explanation. That textual explanation will be passed over to the new `AssertionError` created object.

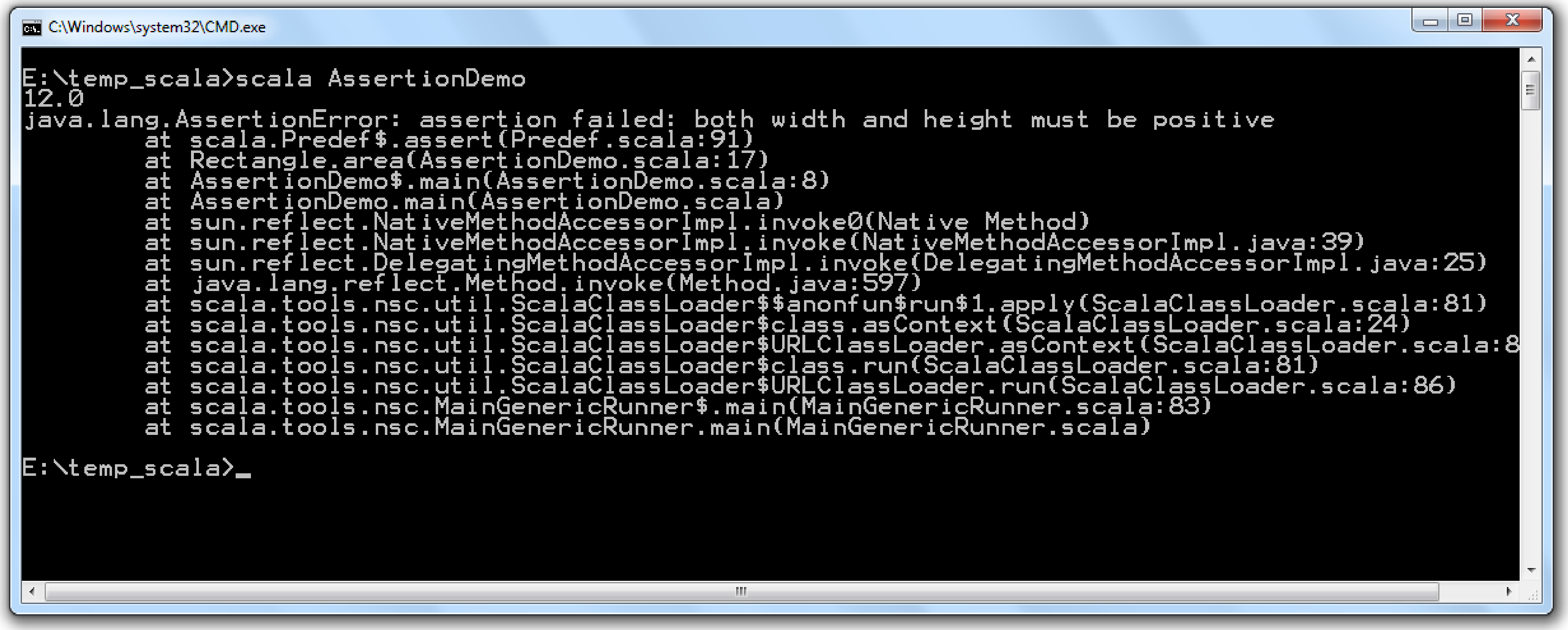
```
...
assert(condition,”explanation...”)
...
```

The `assert` Method

```
object AssertionDemo
{
    def main(args: Array[String])
    {
        var obA = new Rectangle(4,3)
        println(obA.area)
        var obB = new Rectangle
        println(obB.area)
    }
}

class Rectangle(var width:Double,var height:Double)
{
  require(width>0 && height>0)
  def this() = this(0,0)
  def area(): Double =
  {
    assume(width>0 && height>0,
        "width and height must be positive")
    val result:Double = width * height
    assert(result>0)
    result
  }
}
```

The `assert` Method


C:\Windows\system32\CMD.exe

```
E:\temp_scala>scala AssertionDemo
12.0
java.lang.AssertionError: assertion failed: both width and height must be positive
        at scala.Predef$.assert(Predef.scala:91)
        at Rectangle.area(AssertionDemo.scala:17)
        at AssertionDemo$.main(AssertionDemo.scala:8)
        at AssertionDemo.main(AssertionDemo.scala)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at scala.tools.nsc.util.ScalaClassLoader$$anonfun$run$1.apply(ScalaClassLoader.scala:81)
        at scala.tools.nsc.util.ScalaClassLoader$class.asContext(ScalaClassLoader.scala:24)
        at scala.tools.nsc.util.ScalaClassLoader$URLClassLoader.asContext(ScalaClassLoader.scala:8
        at scala.tools.nsc.util.ScalaClassLoader$class.run(ScalaClassLoader.scala:81)
        at scala.tools.nsc.util.ScalaClassLoader$URLClassLoader.run(ScalaClassLoader.scala:86)
        at scala.tools.nsc.MainGenericRunner$.main(MainGenericRunner.scala:83)
        at scala.tools.nsc.MainGenericRunner.main(MainGenericRunner.scala)

E:\temp_scala>_
```

The `@elidable` Annotation

- As of Scala 2.8 we can use the `@elidable` annotation in order to mark methods we want to be able to remove their execution in compile time.
- The assert, assume and require methods were marked with this annotation.
- When marking a method with the `@elidable` annotation we should specify a number. That number would be the priority we assign the marked method.

The `@elidable` Annotation

- The assert, require and assume methods were marked with the `@elidable` together with the `scala.annotation.elidable.ASSERTION` number. Checking the source code of `scala.annotation.elidable` we will find that the value of this constant is 2000.

The `@elidable` Annotation

- In order to exclude the use of the `require`, `assume` and `assert` methods marked with the `@elidable` annotation from the compilation we should pass over the `-Xelide-below` argument to the scalac compiler. We should do so together with numeric value bigger than 2000. This way our use of `require`, `assume` and `assert` will be excluded from the compilation.