Exceptions and Assertions

Prof. Michele Loreti

Programmazione Avanzata
Corso di Laurea in Informatica (L31)
Scuola di Scienze e Tecnologie

What should a method do when it encounters a situation in which it cannot fulfil its contract?

Exception Handling. . .

What should a method do when it encounters a situation in which it cannot fulfil its contract?

One solution is to return an error code.

Exception Handling. . .

What should a method do when it encounters a situation in which it cannot fulfil its contract?

One solution is to return an error code.

This solution is cumbersome for the programmer calling the method!

Exception Handling...

What should a method do when it encounters a situation in which it cannot fulfil its contract?

One solution is to return an error code.

This solution is cumbersome for the programmer calling the method!

- The caller must handle the error (and return another error code if needed);

Exception Handling...

What should a method do when it encounters a situation in which it cannot fulfil its contract?

One solution is to return an error code.

This solution is cumbersome for the programmer calling the method!

- The caller must handle the error (and return another error code if needed);
- It is hard to check if errors have been properly handled.

Exception Handling...

What should a method do when it encounters a situation in which it cannot fulfil its contract?

One solution is to return an error code.

This solution is cumbersome for the programmer calling the method!

- The caller must handle the error (and return another error code if needed);
- It is hard to check if errors have been properly handled.

Exception Handling...

What should a method do when it encounters a situation in which it cannot fulfil its contract?

One solution is to return an error code.

This solution is cumbersome for the programmer calling the method!

- The caller must handle the error (and return another error code if needed);

- It is hard to check if errors have been properly handled.

Instead of having error codes Java support exception handling:

... a method can signal serious problems by throwing an exception;

... one of the method in the call chain can handle the exception.

Throwing exceptions

A method may be in a situation where it cannot carry out the task at hand.

Throwing exceptions

A method may be in a situation where it cannot carry out the task at hand.

Example:

```java
public static int randInt( int low , int high ) {
  return low + (int) (Math.random()*(high-low+1));
}
```

Throwing exceptions

A method may be in a situation where it cannot carry out the task at hand.

Example:

```java
public static int randInt( int low , int high ) {
  return low + (int) (Math.random()*(high-low+1));
}
```

Question: what should happen if someone calls randInt(10,5)?

Throwing exceptions

A method may be in a situation where it cannot carry out the task at hand.

Example:

```java
public static int randInt( int low , int high ) {
    return low + (int) (Math.random()*(high-low+1));
}
```

Question: what should happen if someone calls randInt(10,5)?

Solution: throw appropriate exceptions!

Throwing exceptions

A method may be in a situation where it cannot carry out the task at hand.
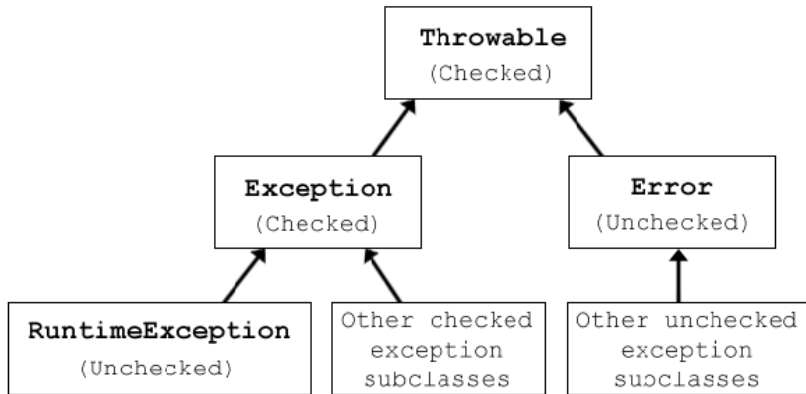
Example:

```java
public static int randInt( int low , int high ) {
  return low + (int) (Math.random()*(high-low+1));
}
```

Question: what should happen if someone calls randInt(10,5)?

Solution: throw appropriate exceptions!

```java
public static int randInt( int low , int high ) {
  if (low > high) {
    throw new IllegalArgumentException (...);
  }
  return low + (int) (Math.random()*(high-low+1));
}
```

```java
public class MyException extends Exception {

  public MyException() {
    super();
  }

  public MyException( String message ) {
    super(message);
  }

  public MyException( String message , Exception cause ) {
    super(message, cause);
  }

  . . .
```

Declaring Checked Exceptions

Any method that might give rise to a checked exception must declare it via the throw clause:

```
public void write( Object o , String filename )
  throws IOException , ReflectiveOperationException
```

Declaring Checked Exceptions

Any method that might give rise to a checked exception must declare it via the throw clause:

```
public void write( Object o , String filename )
    throws IOException , ReflectiveOperationException
```

The throw clause must list all the exceptions that the method might throw (explicitly or due to a recursive call).

Declaring Checked Exceptions

Any method that might give rise to a checked exception must declare it via the throw clause:

```
public void write( Object o , String filename )
    throws IOException , ReflectiveOperationException
```

The throw clause must list all the exceptions that the method might throw (explicitly or due to a recursive call).

Exceptions can be grouped by a super class

... for instance FileNotFoundException, InterruptedIOException could be replaced by IOException

Declaring Checked Exceptions

Any method that might give rise to a checked exception must declare it via the throw clause:

```java
public void write( Object o , String filename )
    throws IOException , ReflectiveOperationException
```

The throw clause must list all the exceptions that the method might throw (explicitly or due to a recursive call).

Exceptions can be grouped by a super class

... for instance FileNotFoundException, InterruptedIOException could be replaced by IOException

Replace multiple exceptions with a single superclass only when these are related!

When do we have to handle an exception?

When do we have to handle an exception?

Someone considers a method that throws an exception harmful!

Handling vs Throwing exceptions. . .

When do we have to handle an exception?

Someone considers a method that throws an exception harmful!

Question: Is this true?

Handling vs Throwing exceptions...

When do we have to handle an exception?

Someone considers a method that throws an exception harmful!

Question: Is this true?
Answer: NO!

Handling vs Throwing exceptions...

When do we have to handle an exception?

Someone considers a method that throws an exception harmful!

Question: Is this true?
Answer: NO!

Throw early, catch late!

Overriding and Exception

When we override a method we **cannot add** more throwing exceptions!

Overriding and Exception

When we override a method we **cannot add** more throwing exceptions!

However, we can **reduce** the list of generated exceptions.

Overriding and Exception

When we override a method we **cannot add** more throwing exceptions!

However, we can **reduce** the list of generated exceptions.

Question: why?

Catching Exceptions

To catch an exception we have to put the code in a `try catch` block:

```java
try {

  //source block

} catch (ExceptionClass1 ex1) {

  //handling block 1

} catch (ExceptionClass2 ex2) {

  //handling block 2

} catch (ExceptionClass3 | ExceptionClass4 ex2) {

  //handling block 3

}
```

Try-with-Resources Statement

Let us consider the following portion of code:

```
String[] lines = ...;
PrintWriter out = new PrintWriter("output.txt");
for (String str: lines) {
  out.println(line.toLowerCase());
}
out.close();
```

Try-with-Resources Statement

Let us consider the following portion of code:

```
String[] lines = ...;
PrintWriter out = new PrintWriter("output.txt");
for (String str: lines) {
  out.println(line.toLowerCase());
}
out.close();
```

This code has a hidden danger!

Try-with-Resources Statement

Let us consider the following portion of code:

```
String[] lines = ...;
PrintWriter out = new PrintWriter("output.txt");
for (String str: lines) {
    out.println(line.toLowerCase());
}
out.close();
```

This code has a hidden danger!

If an exception is thrown, the file is never closed!

Variables can be declared (or referenced) in the `try`:

```java
try (PrintWriter out = new PrintWriter(fileName)) {
  for (String str: lines) {
    out.println(str.toLowerCase());
  }
}
```

Try-with-Resources Statement

Variables can be declared (or referenced) in the `try`:

```
try (PrintWriter out = new PrintWriter(fileName)) {
  for (String str: lines) {
    out.println(str.toLowerCase());
  }
}
```

Declared/referenced variable must be an instance of AutoCloseable. This is an interface with the single method:

```
public void close() throws Exception
```

Try-with-Resources Statement

Variables can be declared (or referenced) in the `try`:

```java
try (PrintWriter out = new PrintWriter(fileName)) {
    for (String str: lines) {
        out.println(str.toLowerCase());
    }
}
```

Declared/referenced variable must be an instance of AutoCloseable. This is an interface with the single method:

```java
public void close() throws Exception
```

When the block terminates (normally or due to an exception), the `close()` method is invoked!

The finally clause

The finally clause can be used to execute something at the end of a try block:

```java
try {
  //try block
} catch (Exception1 e1) {
  //handler1 block
} catch (Exception2 e2) {
  //handler2 block
} finally {
  //finally block
}
```

The finally clause

The finally clause can be used to execute something at the end of a try block:

```java
try {
  //try block
} catch (Exception1 e1) {
  //handler1 block
} catch (Exception2 e2) {
  //handler2 block
} finally {
  //finally block
}
```

Finally block must have not a return statement!

Rethrowing an exception

Sometime is useful to handle only partially a given exception:

```java
try {

} catch ( ExceptionClass e ) {
  //Do something ...
  throw e
} catch ( AnotherExceptionClass e ) {
  //Do something ...
  throw new ApplicationSpecificException ( e ) ;
}
```

Rethrowing an exception

Sometime is useful to handle only partially a given exception:

```java
try {

} catch (ExceptionClass e) {
  //Do something...
  throw e
} catch (AnotherExceptionClass e) {
  //Do something...
  throw new ApplicationSpecificException(e);
}
```

This is useful to transform a *checked* exception into an *unchecked* ones.

Uncaught Exceptions

If an exception is not caught everywhere, a stack trace is displayed.

Uncaught Exceptions

If an exception is not caught everywhere, a stack trace is displayed.

If we want to record the exception and save it somewhere else, we can change the default exception handler:

Uncaught Exceptions

If an exception is not caught everywhere, a stack trace is displayed.

If we want to record the exception and save it somewhere else, we can change the default exception handler:

```
Thread.setDefaultUncaughtExceptionHandler((thread,ex) -> {
    //Record exception.
  }
);
```

Uncaught Exceptions

If an exception is not caught everywhere, a stack trace is displayed.

If we want to record the exception and save it somewhere else, we can change the default exception handler:

```
Thread.setDefaultUncaughtExceptionHandler((thread,ex) -> {
    //Record exception.
  }
);
```

When we are not able to handle an exception, the only solution is to report the stack trace:

```
try {
    Class<?> cl = Class.forName(className);
    ...
} catch (ClassNotFoundException e) {
    ex.printStackTrace();
}
```

Class StackWalker can be used to inspect the stack trace.

Class StackWalker can be used to inspect the stack trace.

Class Objects provides utility methods that perform convenient null check:

- Objects.requireNonNull(var)
- Objects.requireNonNullElse(var,e)
- Objects.requireNonNullElseGet(var,f)

Assertions are used to perform defensive programming

Assertions

Assertions are used to perform defensive programming

Example:
```
if (x<0) {
    throw new IllegalStateException(x+" < 0");
}
Math.sqrt(x);
```

Assertions

Assertions are used to perform defensive programming

Example:

```
if (x<0) {
    throw new IllegalStateException(x+" < 0");
}
Math.sqrt(x);
```

Assertions can be used to check if a given condition is satisfied::

```
assert x>=0;
Math.sqrt(x);
```

Assertions

There are two forms of assertions:

```
assert condition ;

assert condition : expression ;
```

Assertions

There are two forms of assertions:

```
assert condition ;

assert condition : expression ;
```

In the second case the expression is used to build the error message!

Assertions

There are two forms of assertions:

```
assert condition;

assert condition : expression;
```

In the second case the expression is used to build the error message!

N.B.: Assertions can be enabled/disabled at execution time via −ea and −da parameters.

To be continued...