Exceptions

Adam Krechowicz

Exception

Exception is an event that occurs during program execution, it breaks the typical program flow.

Exception is an object

- Each exception is derived from Exception class
- Exception class implements Throwable interface
- Error class implements Throwable class
- Exception class hierarchy is very important

Throwable

- getMessage()
- getStackTrace()
- getCause()

Types of exceptions

- Error – severe situations. Indicates the problem outside of the application (e. g. Class can not be loaded)
- Exception – situation that occurs in application. Should be properly handled (e. g. No such file to open)
- Runtime exception – subset of Exceptions. Situations in application which should not occur (e. g. Division by zero)

Categories of exceptions

- Unchecked exceptions – we do not know when they might occurs (RuntimeException, Error)
- Checked exceptions – we know when they can occurs during the compilations

Exceptional situation

```
1  public class Event {
2
3      public static void main(String[] args){
4          int a = 5;
5          int b = 0;
6          System.out.println(a/b);
7      }
8
9  }
```

Result

```
1 Exception in thread "main" java.lang.ArithmeticException: / by zero
2   at com.wyklad.exceptions.Event.main(Event.java:17)
3 Java Result: 1
```

Stos wywołań metod

```
1  public class Stack {
2
3      private void first(){
4          second();
5      }
6
7      private void second(){
8          third();
9      }
10
11     private void third(){
12         int i = 0;
13         System.out.println(5/i);
14     }
15
16     public static void main(String[] args){
17         Stack s = new Stack();
18         s.first();
19     }
20
21 }
```

Result

```
1  Exception in thread "main" java.lang.ArithmeticException: / by zero
2    at com.wyklad.exceptions.Stack.third(Stack.java:24)
3    at com.wyklad.exceptions.Stack.second(Stack.java:19)
4    at com.wyklad.exceptions.Stack.first(Stack.java:15)
5    at com.wyklad.exceptions.Stack.main(Stack.java:29)
```

Throwing exception

```
1  public class Throw {
2
3  public static void main(String[] args) throws Exception{
4    int i = 0;
5    int j = 7/i;
6  }
```

Method that throw exception

```
1  public class Throw {
2
3    private void method(int i){
4      int j = 5/i;
5    }
6
7  public static void main(String[] args) throws Exception{
8    Throw t = new Throw();
9    t.method(0);
10 }
```

Throwing exception manually

```
1  public class Throw {
2
3  public static void main(String[] args) throws Exception{
4    throw new Exception("Exception");
5  }
```

Handling exceptions

- If exception can occur it must be properly handled
- Exception that was thrown should be caught
- If method in which exception was thrown do not catch that exception it will be interrupted
- Exception is propagated to the method that called interrupted method
- If exception is not caught in main method the program will be interrupted

try...catch

```
1  public class Catching {
2
3      public static void main(String[] args){
4          Catching c = null;
5          try{
6              System.out.println(c.toString());
7          } catch (Exception e){
8              e.printStackTrace();
9          }
10     }
11
12 }
```

Many catch blocks

```
1  public class Multiple {
2
3      public static void main(String[] args){
4          try {
5              Files.readAllBytes(Paths.get("C:\\", "test.txt"));
6          } catch (NoSuchFileException ex){
7              System.out.println("No such file");
8          } catch (FileSystemException ex){
9              System.out.println("File system problem");
10         } catch (IOException ex) {
11             System.out.println("Input/output problem");
12         } catch (Exception ex){
13             System.out.println("Some exception");
14         } catch (Throwable t){
15             System.out.println("Anything that can be thrown");
16         }
17     }
18
19 }
```

Many exceptions

```
1  public class Multiple {
2
3      public static void main(String[] args){
4          try {
5              Files.readAllBytes(Paths.get("C:\\", "test.txt"));
6              Socket s = new Socket("192.168.0.15", 10);
7              s.getOutputStream();
8          } catch (NoSuchFileException ex) {
9              Logger.getLogger(Multiple.class.getName()).log(Level.SEVERE, null, ex);
10         } catch (UnknownHostException ex) {
11             Logger.getLogger(Multiple.class.getName()).log(Level.SEVERE, null, ex);
12         } catch (IOException ex){
13             Logger.getLogger(Multiple.class.getName()).log(Level.SEVERE, null, ex);
14         }
15     }
16
17 }
```

Multi catch

```
1  public class Multiple {
2
3      public static void main(String[] args){
4          try {
5              Files.readAllBytes(Paths.get("C:\\", "test.txt"));
6              Socket s = new Socket("192.168.0.15", 10);
7              s.getOutputStream();
8          } catch (NoSuchFileException | UnknownHostException ex) {
9              Logger.getLogger(Multiple.class.getName()).log(Level.SEVERE, null, ex);
10         } catch (IOException ex){
11             Logger.getLogger(Multiple.class.getName()).log(Level.SEVERE, null, ex);
12         }
13     }
14
15 }
```

Order of catch blocks

```
1  public class Order {
2
3      public static void main(String[] args){
4          try {
5              Files.readAllBytes(Paths.get("C:\\", "test.txt"));
6          } catch (Throwable t){
7              System.out.println("Anything");
8          } /*catch (NoSuchFileException ex){
9              System.out. println ("No such  file ");
10         } catch (FileSystemException ex){
11             System.out. println ("Filesystem  problem");
12         } catch (IOException ex) {
13             System.out. println ("Input/output problem");
14         } catch (Exception ex){
15             System.out. println ("Exception");
16         }  */
17     }
18
19 }
```

finally block

```
1  public class Finally {
2
3      public static void main(String[] args){
4          try{
5              int i = 0;
6              System.out.println(7/i);
7          } catch (Exception e){
8              e.printStackTrace();
9          } finally {
10             System.out.println("Always executed");
11         }
12     }
13
14 }
```

throws

```
1  public class Throw {
2
3      private void method() throws Exception{
4          throw new Exception("I don't like it");
5      }
6
7      public static void main(String[] args) throws Exception{
8          Throw t = new Throw();
9
10         try{
11             t.method();
12         } catch (Exception e){
13             e.printStackTrace();
14         }
15     }
```

throws

```
1  public class Throw {
2
3      private void method() throws Exception{
4          throw new Exception("I don't like it");
5      }
6
7      private void method1() throws FileNotFoundException{
8          throw new FileNotFoundException("Not working");
9      }
10
11     private void method2() throws Exception{
12         throw new FileNotFoundException("No!");
13     }
14
15 }
```

throws

```
1  public class Throw {
2
3      /*private void method3() throws FileNotFoundException{
4          throw new Exception("Reject");
5      }*/
6
7      private void method4() throws FileNotFoundException, SocketException{
8          if (Math.random() > 0.5)
9              throw new FileNotFoundException();
10         else
11             throw new SocketException();
12     }
13
14     private void method5() throws FileNotFoundException{
15
16     }
17
18 }
```

Closing resources

```
1        Socket s = null;
2        ServerSocket server = null;
3        try {
4            server = new ServerSocket(65500);
5            s = server.accept();
6            System.out.println("accepted");
7            InputStream is = s.getInputStream();
8            ObjectInputStream ois = new ObjectInputStream(is);
9            byte[] buffer = new byte[128];
10           int i = is.read(buffer);
11           System.out.println(new String(buffer));
12       } catch (IOException ex) {
13           ex.printStackTrace();
14       } finally {
15           try {
16               if (s != null) s.close();
17               if (server != null) server.close();
18           } catch (IOException ex) {
19               ex.printStackTrace();
20           }
21       }
```

try-with-resource

```
1  public class Resource {
2
3      public static void main(String[] args) throws IOException{
4          try (BufferedReader br =
5                  new BufferedReader(new FileReader("test.txt"))) {
6              System.out.println(br.readLine());
7          }
8      }
9
10 }
```

AutoCloseable

- Object need to implement AutoClosable interface
- After try block the resource will be automatically closed
- BufferedReader, BufferedWriter
- FileInputStream, FileOutputStream
- FileWriter, FileReader
- Socket, ServerSocket, DatagramSocket

Rethrowing exception

```
1  public class Rethrow {
2
3      private void inner() throws Exception{
4          throw new Exception("Internal");
5      }
6
7      private void method() throws Exception{
8          try{
9              inner();
10         } catch (Exception e){
11             throw new Exception("External", e);
12         }
13     }
14
15   public static void main(String[] args) throws Exception{
16         Rethrow r = new Rethrow();
17         r.method();
18     }
19
20 }
```

Result

```
1  Exception in thread "main" java.lang.Exception: Zewnetrzny
2    at com.wyklad.exceptions.Rethrow.method(Rethrow.java:22)
3    at com.wyklad.exceptions.Rethrow.main(Rethrow.java:29)
4  Caused by: java.lang.Exception: Wewnetrzny
5    at com.wyklad.exceptions.Rethrow.inner(Rethrow.java:15)
6    at com.wyklad.exceptions.Rethrow.method(Rethrow.java:20)
7     ... 1 more
8  Java Result: 1
```

Nesting the cause

```
1  public class Rethrow {
2
3      private void inner() throws Exception{
4          throw new Exception("Internal");
5      }
6
7      private void method() throws Exception{
8          try{
9              inner();
10         } catch (Exception e){
11             throw new Exception("External", e);
12         }
13     }
14
15     public static void main(String[] args) throws Exception{
16         Rethrow r = new Rethrow();
17         try{
18             r.method();
19         } catch (Exception e){
20             throw new Exception("Very external", e);
21         }
22     }
```

Result

```
1  Exception in thread "main" java.lang.Exception: Bardzo zewnetrzny
2    at com.wyklad.exceptions.Rethrow.main(Rethrow.java:31)
3  Caused by: java.lang.Exception: Zewnetrzny
4    at com.wyklad.exceptions.Rethrow.method(Rethrow.java:22)
5    at com.wyklad.exceptions.Rethrow.main(Rethrow.java:29)
6  Caused by: java.lang.Exception: Wewnetrzny
7    at com.wyklad.exceptions.Rethrow.inner(Rethrow.java:15)
8    at com.wyklad.exceptions.Rethrow.method(Rethrow.java:20)
9     ... 1 more
10 Java Result: 1
```

Custom exception

```
1  public class CustomException extends Exception{
2
3      public CustomException(String message){
4          super(message);
5      }
6
7  }
```

Using custom exception

```
1  public class Own {
2
3      public static void main(String[] args) throws CustomException{
4          throw new CustomException("Exceptional exception");
5      }
6
7  }
```

Custom exception

- If the exception can be treated as already existing exception?
- Exception may represent logical situation not only technical (e. g. NoFeeTicketsAvailableException)
- Proper exception base class should be chosen

Runtime exceptions

- RuntimeExceptions do not need to be caught
- Method do not need to mark with trows keyword
- Can occur in many places
- Can be eliminated by proper programming constructions (e. g. if construction to check if we try to divide by zero)
- Often are overused by programmers due to simplicity

RuntimeExceptions – problems

- Ease to use:
  - Do not need to be declared
  - Do not need to be caught
- In proper wrote program we are avoiding RuntimeException
- If user can react on exception we use Exception class
- If user can not react on exception we use RuntimeException

Typical exceptions – RuntimeException

- ArithmeticException – exception during arithmetician (np. division by zero)
- ClassCastException – exception during class casting
- NullPointerException – exception during referencing null reference
- ArrayIndexOutOfBoundsException – exception during referencing bad array index
- StringIndexOutOfBoundsException – exception during referencing non existing string character

Typical exceptions – Exception

- IOException – base exception for input/output exceptions
- InterruptedException – exception during interruption of thread
- CloneNotSupportedException – exception during cloning object that do not implements Clonable interface
- SQLException – exception during execution of SQL query

Typical exceptions – IOException

- EOFException – end of tile exception
- FileNotFoundException – file does not exists
- MalformedURLException – error in URL format
- SocketException – exception during network communication

Assertions

```
1  public class Assertions {
2
3      public static void main(String[] args) {
4          Assertions a = null;
5          assert a != null;
6      }
7
8  }
```

Using assertions

java -ea com.adeik.wyklad.exceptions.testing.Assertions

Result

Exception in thread "main"java.lang.AssertionError at com.wyklad.exceptions.testing.Assertions.main(Assertions.java:12)

Assertions

```
1  public class Assertions {
2
3      public static void main(String[] args) {
4          Assertions a = null;
5          assert a != null: "A is null";
6      }
7
8  }
```

Result

Exception in thread "main"java.lang.AssertionError: A is null at com.wyklad.exceptions.testing.Assertions.main(Assertions.java:12)

What should be checked:

- Test if reference is null
- Test if Optional object contains something
- Test if code should be executed (e. g. default block in switch)
- Testing conditions that are connected with program logic should not use assertions

Optional

```
1  public class OptionalTest {
2
3      public Optional<String> getSth(){
4          if (Math.random() > 0.5)
5              return Optional.of("Testing");
6          else return Optional.empty();
7      }
8
9      public static void main(String[] args) {
10         OptionalTest ot = new OptionalTest();
11         Optional<String> result = ot.getSth();
12         if (result.isPresent())
13             System.out.println(result.get());
14     }
15
16 }
```

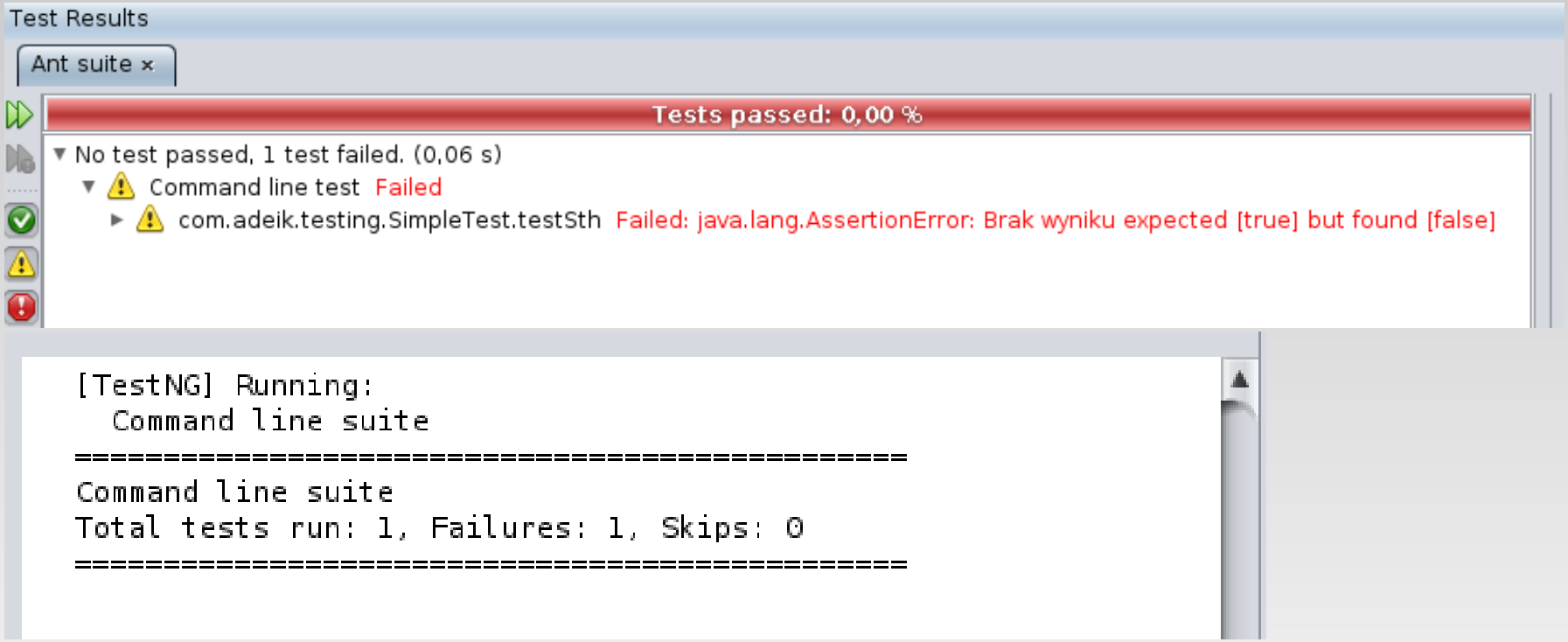
Code testing

- Assertions are executing during normal program execution
- Do not allow to test all possible situations
- For example:
  - User input is correct
  - User input is incorrect

Libraries for testing

- Arquillian
- JTest
- JUnit
- TestNG
- Mockito

```
1  public class SimpleTest {
2
3      @Test
4      public void testSth(){
5          OptionalTest ot = new OptionalTest();
6          Optional<String> result = ot.getSth();
7          Assert.assertTrue(result.isPresent(), "No result");
8      }
9
10 }
```


Test Results
Ant suite ×
Tests passed: 0,00 %
No test passed, 1 test failed. (0,06 s)
Command line test Failed
com.adeik.testing.SimpleTest.testSth Failed: java.lang.AssertionError: Brak wyniku expected [true] but found [false]
[TestNG] Running:
Command line suite
Command line suite
Total tests run: 1, Failures: 1, Skips: 0

```
[TestNG] Running:
  Command line suite

[VerboseTestNG] RUNNING: Suite: "Command line test" containing "1" Tests (config: null)
[VerboseTestNG] INVOKING: "Command line test" - com.adeik.testing.SimpleTest.testSth()
[VerboseTestNG] PASSED: "Command line test" - com.adeik.testing.SimpleTest.testSth() finished in 79 ms
[VerboseTestNG]
[VerboseTestNG] ===============================================
[VerboseTestNG]     Command line test
[VerboseTestNG]     Tests run: 1, Failures: 0, Skips: 0
[VerboseTestNG] ===============================================

===============================================
Command line suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================

Deleting directory /tmp/com.adeik.testing.SimpleTest
test:
BUILD SUCCESSFUL (total time: 2 seconds)
```

THE END!

Additional reading:

- https://docs.oracle.com/javase/tutorial/essential/exceptions/
- https://docs.oracle.com/javase/specs/jls/se7/html/jls-11.html
- https://dzone.com/articles/7-tips-for-writing-better-unit-tests-in-java
- Thinking in Java chapters by Bruce Eckel:
  - Error Handling with Exceptions