6 Exceptions

- Introduction
- Handling Exceptions
- Checked & Unchecked Exceptions

- For some problems at runtime, the Java Virtual Machine generates (‘*throws*’) special objects describing the observed exceptional situation.
- These objects belong to subclasses of **`Error`** and **`Exception`**.
- **`Error`** and **`Exception`** are part of the package **`java.lang`** and are subclasses of **`Throwable`**.
- **`Exception`** objects can be *caught* and handled in the program.
- **`Error`** objects represent situations that the program cannot repair itself.

- **`Error`** has the subclasses
  - **`LinkageError`**
  - **`ThreadDeath`**
  - **`VirtualMachineError`**
  - **`AWTError`**
  - ...
- **`Exception`** has the subclasses:
  - **`RuntimeException`** mit den Unterklassen
    - **`ArithmeticException`**
    - **`IndexOutOfBoundsException`**
    - **`NullPointerException`**
    - **`NoSuchElementException`**
    - ...
  - **`NoSuchMethodException`**
  - **`ClassNotFoundException`**
  - **`IOException`**
  - ...

6 Exceptions
- Introduction
- Handling Exceptions
- Checked & Unchecked Exceptions

Exceptions can be handled in three ways:

- The program does not handle the exception.
- Exceptions are caught where they are generated.
- Exceptions are caught somewhere else in the program.

If an exception is not caught, the program terminates with an output on the standard error output (**`System.err`**) with details about:

- the type of exception
- the location in the program where the exception was thrown

No handling of exceptions:

```
1 public class K6B01E_UncaughtException {
2
3   public static void main (String[] args) {
4     int result = 1 / 0;
5     System.out.println("This output is not shown" );
6   }
7
8 }
```

Division by 0 generates **`ArithmeticException`**:

```
1 Exception in thread "main" java.lang.ArithmeticException: / by zero
2         at K6B01E_UncaughtException.main(K6B01E_UncaughtException.java:4)
```

Catching exceptions where they are generated:

```
1  public class K6B02E_CaughtException {
2   public static void main (String[] args) {
3     try {
4       String str = System.console().readLine("denominator = ");
5       int denominator = Integer.parseInt(str);
6       int int result = 1 / denominator;
7       System.out.println("without error: output 1");
8     }
9     catch (ArithmeticException newAE) {
10      System.out.println("Arithmetic␣Exception!");
11    }
12  ...
```

input	reaction
1	**without error: output 1**
0	**Arithmetic Exception!**
to	program termination

```
1  ...
2    try {
3       String str = System.console().readLine("denominator = ");
4       int denominator = Integer.parseInt(str);
5       int result = 1 / denominator;
6       System.out.println("without error: output 2");
7    }
8    catch (ArithmeticException newAE) {
9       System.out.println(newAE.toString());
10   }
11   catch (NumberFormatException newNFE) {
12      System.out.println(newNFE.toString());
13   }
14   finally {
15      System.out.println("always: output 3");
16   }
17  }
18 }
```

- **`try`** block without error: continue with (optional) **`finally`** block.
- exception in **`try`** block:
  - if matching **`catch`** block exists:
    continue with this **`catch`** block, then continue with **`finally`** block.
  - if no matching **`catch`** block exists:
    continue with **`finally`** block **(then program termination!)**

Handling exceptions somewhere else:

- If an exception is not caught within a method, then:
  - the method execution is aborted immediately;
  - the exception is propagated to the calling method
- The calling method can catch the propagated exception only if the propapating method was executed within a **`try`** block.
- The propagation continues until the exception is either caught or propagated outside the method **`main`**.
  The latter causes a program termination.

Exceptions over multiple call levels and classes:

```
1  public class K6B03E_Propagation {
2    public static class Scope { // nested static class ...
3
4      public void level1 () {
5        System.out.println("Level␣1:␣␣Start");
6        int result = 1 / 0;
7        System.out.println("Level␣1:␣␣End");
8      }
9
10     public void level2() {
11       System.out.println("Level␣2:␣␣Start");
12       level1 ();
13       System.out.println("Level␣2:␣␣End");
14     }
15
16 ...
```

```
...
    public void level3() {
      System.out.println("Level␣3:␣␣Start");
      try {  level2(); }
      catch (ArithmeticException newAE) {
        System.out.print  ("Exception␣message:␣␣␣␣");
        System.out.println(newAE.getMessage());
        System.out.print("Exception␣(toString):␣");
        System.out.println(newAE.toString());
        System.out.println("Call␣stack␣trace:-------");
        StackTraceElement[] stackTrace = newAE.getStackTrace();
        for (int i = 0; i < stackTrace.length; i++)
          System.out.println("␣␣" + stackTrace[i].toString());
        System.out.println("-----------------------");
      }
      System.out.println("Level␣3:␣End");
    }
  }
```

Test of the propagation over multiple call levels:

```
1     public static void main (String[] args) {
2         Scope demo = new Scope();
3         System.out.println("Program: Start");
4         demo.level3();
5         System.out.println("Program: End");
6  }  }
```

```
1  Program: Start
2  Level 3:  Start
3  Level 2:  Start
4  Level 1:  Start
5  Exception message:    / by zero
6  Exception (toString): java.lang.ArithmeticException: / by zero
7  Call stack trace:-------
8    K6B03E_Propagation$Scope.level1(K6B03E_Propagation.java:13)
9    K6B03E_Propagation$Scope.level2(K6B03E_Propagation.java:19)
10   K6B03E_Propagation$Scope.level3(K6B03E_Propagation.java:25)
11   K6B03E_Propagation.main(K6B03E_Propagation.java:6)
12 -----------------------
13 Level 3: End
14 Program: End
```

6 Exceptions
- Introduction
- Handling Exceptions
- Checked & Unchecked Exceptions

- Some exceptions can be expected and thus handled in a natural way, others cannot be ‘repaired’...
- Java thus offers two types of exceptions:
  - **Checked Exceptions** for exceptions that can be expected and can be handled by the user (and must be handled)
  - **Unchecked Exceptions** for the hopeless cases
- **The JAVA Tutorials**(Oracle):
  If a client can reasonably be expected to recover from an exception, make it a checked exception.
  If a client cannot do anything to recover from the exception, make it an unchecked exception.
- but for example Bruce Eckel (in **“Thinking in Java”**) states:
  Checked Exceptions are not needed at all...

- **Checked Exceptions** must be either caught within a method or declared in the **`throws`** clause of the method header.
- **Unchecked Exceptions** do not need to be declared in the **`throws`** clause; only objects of the class **`RuntimeException`** are unchecked, e.g.:

```
ArithmeticException
BufferOverflowException
BufferUnderflowException
ClassCastException
EmptyStackException
IllegalArgumentException
IndexOutOfBoundsException
MissingResourceException
NegativeArraySizeException
NoSuchElementException
NullPointerException
NumberFormatException
...
```

Example: `java.io.File` includes methods for accessing files, that can be used with the `Scanner` class similar to `System.in`.

Difference: `System.in` always exists, but files could be missing or unreadable. Thus, when instantiating `Scanner` objects, very critical errors may happen that *must* be handled, see constructor for `Scanner` objects from `File`:

```
1 public Scanner(File source) throws FileNotFoundException
```

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4  public class K6B05E_File{
5    public static void main(String[] args)
6                                throws FileNotFoundException {
7        Scanner sc = new Scanner(System.in);
8        System.out.print ("Input file name: ");
9        String filename = sc.nextLine();
10       File file = new File(filename);
11       Scanner input = new Scanner(file);
12       while ( input.hasNextLine() ){
13         System.out.println(input.nextLine());
14 }    }    }
```

Alternative solution:

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4
5  public class K6B06E_FileCheck{
6    public static void main(String[] args)  {
7
8      Scanner sc = new Scanner(System.in);
9      Scanner input;
10
11      while ( true ) {
12       System.out.print ("Input file name: ");
13       String filename = sc.nextLine();
14       File file = new File(filename);
15       try{
16         input = new Scanner(file);
17         break;
18       } catch ( FileNotFoundException fnfe ) {
19         System.out.println ("\nError: file not found!");
20       }
21     }
22
23     while ( input.hasNextLine() ){
24       System.out.println(input.nextLine());
25     }
26 } }
```