Exceptions (1)

- Exceptions are an appropriate way to handle errors without cluttering code with
error handling procedures.

- In other programming languages, traditionally special return values of functions
are defined, which indicate a certain error condition.

  Problem: different from programmer to programmer.

- Exceptions make error conditions, which can be indicated by a method, and are
an explicit part of the usage of this method.

- An exception is thrown if an unexpected condition occurs. The exception is
encapsulated together with some context information (e.g. a stack trace) in a
special exception object and returned to the calling method for further treatment.
Exception handling is a method for dealing with exceptions and errors.

Exception: the program reaches an abnormal, unexpected state. This can happen due to operations, data or user input.

Error: due to runtime errors the program reaches a state where it cannot continue to run. Consequence: it crashes.

For not being forced to end the program, **Exception** can be used. The program becomes more robust.

Exception can be forwarded through several methods: if an exception occurs in one method, it is forwarded to the caller of that method.

- Example: *CheckYear* -&gt; *Constructor* -&gt; *Main*
• Exceptions are represented by objects being instances of the class \texttt{java.lang.Exception}

Error & 
\textit{RuntimeExceptions} must not be caught: „unchecked“

All other exceptions: „checked“
Syntax for exceptions (1)

- Exceptions can be initiated by creating an instance of `Exception` or of `Throwable` or a subclass of it. The creation is initiated with the instruction `throw`: `throw new Exception();`

  ```java
  throw new BlackoutException("call EnBW");
  ```

- `throw new RuntimeException("This is impossible!");`

- `class BlackoutException extends Exception { … }

  throw new BlackoutException("call EnBW");`
• Exceptions can be solved in the sense that an instance from exception or one of the sub-classes can be created with the directive throw: `throw new Exception();`

• Syntax of the head of a method, in case the method can throw an exception:

```
Normal head of a method with access denoter, type of the return values, name, parameter lists
```

- All „checked“ exceptions, that will be thrown, must be advertized with „throws“
- „unchecked“ exceptions can be advertized with „throws“

But:
- All exceptions, that are advertized with „throws“, **must** be caught with a `try...catch` structure.
Syntax for exceptions (3)

• Calling up a method which can create an exception:
  inside of a `try`-block

```
try {
  Directive and call of the method, to deal with the exception
}
```

• The treatment of the eventual exception inside of a `try`-block follows in the corresponding `catch`-blocks:

```
catch (Object name in the exception class)
  Directive to handle the exception
}
```

• After one `try`-block `catch`-blocks follow
  - In case different exceptions emerge, the treatment of each type of exception can be carried out in its specific `catch`-block
  - For each type of exception only one `catch`-block is possible

• After dealing with the exception and the return value in the `try`-block, it will immediately be left on the corresponding position, and eventual directives will not be performed any more
Generating Exceptions (1)

- An exception can be created by using an instance of the class `Exception` or of one of the numerous subclasses. All that has to be done is creating an instance of exception and throwing this object by using the `throw` statement:

  ```java
  throw new Exception();
  ```

- Stops in the exception-throwing line of the code and control is passed to the calling method

- In every method where an exception occurs, it must be indicated in a `throws` statement in the methods declaration that this exception can occur.
Generating Exceptions (2)

- If a method is called which is able to throw exceptions, it has to be called from within a try-block,
- The Exception-object is created precisely at the position where the error condition (e.g. file not found, access denied…) occurs and afterwards - together with the control flow - passed on to the block in charge, the catch-block
- This is the origin of the terms `throwing` and `catching` exceptions: the Exception-object is first thrown by a location within the code and caught by another block of statements
- Every statement in the try block can rely on the fact that all preceding statements in the block were carried out successfully

```java
// ...
try {
    checkRead("foo");
    ...
} catch (Exception e) {
    // error handling
    System.out.println("Error while reading:" + e);
    ...
} // ...
Example

• Throwing an exception:

```java
public returnType methodName(param1, param2, …) throws Exception {
}
```

• Catching an exception:

```java
try {
    object.methodName(param1, param2, …);
}

catch (Exception exc) {
    // Reaction on the exception
    System.out.println(exc.getText());
}
```