

Programmieren I

Fehlerbehandlung – Exceptions



Heusch 2. Bd, 3
Ratz 10

Institut für Automation und angewandte Informatik

```
final List<String> allResults = new ArrayList<String>();  
final Map<String, Integer> typeWordResultCount = new HashMap<String, Integer>();  
final Map<String, Integer> typePoints = new HashMap<String, Integer>();  
evaluation.put(type, typePoints);  
  
for (final Sheet sheet : this.sheets) {  
    final String sheetResult = sheet.getPlayerInput(type);  
    if (sheetResult.startsWith(start) && this.isValidWord(sheetResult, type)) {  
        validWordCountForType++;  
        allResults.add(sheetResult);  
    }  
}
```

Exceptions und ihre Behandlung

- Exception - Ausnahmebedingung (wie z.B. Fehler)
- Exception erzeugen (engl. „to throw“) bedeutet Ausnahmebedingung zu signalisieren.
- Exception abfangen (engl. „to catch“) bedeutet eine Ausnahmebedingung zu behandeln, d.h. alle Aktionen durchzuführen, die notwendig sind um den „normalen“ Zustand wiederherzustellen.

Exception-Objekte

- Eine Exception ist in Java ein Objekt, nämlich eine Instanz irgendeiner Unterklasse von `java.lang.Throwable`.
- `Throwable` besitzt 2 Standardunterklassen:
 - `java.lang.Error`
für Probleme beim dynamischen Laden oder bei Probleme der JVM. Sie werden i.d.R. nicht abgefangen.
 - `java.lang.Exception`
Exceptions dieser Unterklasse weisen auf Bedingungen hin, die abgefangen und behoben werden können, z.B.
 - `java.io.EOFException` oder
 - `java.lang.ArrayIndexOutOfBoundsException`

Exception-Handling

■ **try / catch / finally**-Anweisung

■ **try**

stellt einen Codeblock zur Verfügung, mit dem Exceptions und abnormale Abbrüche behandelt werden.

■ **catch**

dem `try`-Block folgen null oder mehr `catch`-Klauseln, die bestimmte `Exception`-Typen abfangen und behandeln.

■ **finally**

den `catch`-Klauseln folgt optional ein `finally`-Block, der hinterher „aufräumt“. Die Anweisungen eines `finally`-Blocks werden garantiert ausgeführt, gleichgültig mit welchem Status der `try`-Block beendet wird.

Exceptions: So sieht es aus

```
try {  
    /* Programmcode der Exceptions erzeugt */  
}  
catch (ExceptionType1 e1) {  
    /* Handle Exception e1 von Typ ExceptionType1 */  
}  
catch (ExceptionType2 e2) {  
    /* Handle Exception e2 von Typ ExceptionType2 */  
}  
finally {  
    /* Dieser Code wird immer ausgeführt */  
}
```

Exceptions: Beispiel

```
import java.util.Scanner;

public class ExceptionExample {

    public static void main(String[] args) {
        try {
            System.out.print("Please enter an integer: ");
            Scanner scan = new Scanner(System.in);
            String input = scan.next(); // String einlesen
            int intNumber = Integer.parseInt(input);
            System.out.println("Tripled: " + 3 * intNumber);
        } catch (NumberFormatException e) {
            System.err.println("Error during conversion: "
                               + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Einige Methoden von Throwable

■ **String getMessage()**

- Gibt die Beschreibung der Exception zurück.

■ **String toString()**

- Gibt eine kurze Beschreibung von Throwable einschließlich der detaillierten Beschreibung.

■ **void printStackTrace()**

- Gibt den Call-Stack-Trace aus.

```
e.printStackTrace();
```

Ausgabe

```
Error during conversion: For input string: "a"  
java.lang.NumberFormatException: For input string: "a"  
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.lang.Integer.parseInt(Integer.java:492)  
    at java.lang.Integer.parseInt(Integer.java:527)  
    at slides.exceptions.ExceptionExample.main(ExceptionExample.java:17)
```

Call-Stack-Trace: Beispiel aus der Praxis

Exception StackTrace

```
EJava.Exception.InvocationException: /uinbw/Entry/uinbw2GSA.html (Line: 36)
failed to invoke method 'nextPageURL' on object Application.gsa.Search@132021a with parameters:
[Ljava.lang.Object;@2803d5
    at EJava.Reflection.JavaMethodInvocation.value(JavaMethodInvocation.java:141)
    at EJava.Scripting.AccessStatement.doProcess(AccessStatement.java:52)
    at EJava.Scripting.ScriptingElement.processWith(ScriptingElement.java:133)
    at EJava.Interpreter.processAll(Interpreter.java:286)
    at EJava.Interpreter.processFileHelper(Interpreter.java:266)
    at EJava.Scripting.EmbedStatement.doProcess(EmbedStatement.java:75)
    at EJava.Scripting.ScriptingElement.processWith(ScriptingElement.java:133)
    at EJava.Interpreter.processAll(Interpreter.java:286)
    at EJava.Scripting.ConditionalStatement.doProcess(ConditionalStatement.java:65)
    at EJava.Scripting.ScriptingElement.processWith(ScriptingElement.java:133)
    at EJava.Interpreter.processAll(Interpreter.java:286)
    at EJava.Interpreter.processFileHelper(Interpreter.java:266)
    at EJava.Scripting.EmbedStatement.doProcess(EmbedStatement.java:75)
    at EJava.Scripting.ScriptingElement.processWith(ScriptingElement.java:133)
    at EJava.Interpreter.processAll(Interpreter.java:286)
    ...
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:468)
    at java.lang.Integer.parseInt(Integer.java:497)
    at Application.gsa.Search.nextPageURL(Search.java:779)
    at sun.reflect.GeneratedMethodAccessor50.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:585)
    at EJava.Reflection.JavaMethodInvocation.value(JavaMethodInvocation.java:131)
    ...
```


Exceptions deklarieren

- Java erfordert, dass jede Methode, die eine `Exception` verursachen kann, diese entweder abfangen oder in der Methodendeklaration den Typ der `Exception` mit einer `throws`-Klausel angeben muss.
- Beispiele:

```
public void openFile(/*...*/) throws IOException {  
    /* Hier stehen Anweisungen die eine nicht  
    * abgefangene java.io.IOException  
    * verursachen können. */  
}  
  
public void myFunc(/*...*/) throws TooBigEx, TooSmallEx, DivByZeroEx {  
    /* Hier stehen Anweisungen, die die nicht abgefangenen  
    * Exceptions TooBigEx, TooSmallEx und DivByZeroEx  
    * verursachen können. */  
}
```

Eigene Exceptions

- Neben den Standard Java-Exceptions (Klasse `Throwable` mit Unterklassen `Error` und `Exception`, siehe java.oracle.com) können auch eigene Exceptions definiert werden:
- Beispiel:

```
class MyException extends Exception {  
  
    public MyException(){  
    }  
  
    public MyException(String message){  
        super(message);  
    }  
  
}
```

Eigene Exceptions: Beispiel (1)

```
public class DivZeroException extends Exception {  
  
    public DivZeroException(){  
    }  
  
    public DivZeroException(String message){  
        super(message);  
    }  
  
}
```

Eigene Exceptions: Beispiel (2)

```
public class MyMath {  
  
    public static int divide(int a, int b)  
        throws DivZeroException {  
        if ( b == 0 ){  
            throw new DivZeroException("Division by zero!");  
        } else {  
            return a / b; // Ganzzahldivision!  
        }  
    }  
}
```

Eigene Exceptions (4)

```
public class Test {  
  
    public static void main(String[] args) {  
        int res;  
        try {  
            res = MyMath.divide(8, 0);  
            System.out.println("Result: " + res);  
        } catch (DivZeroException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

Exception Matching (1)

- Der Exception-Handler sucht in den catch-Klauseln, die erste die „passt“ (erfordert kein „genaues passen“).
- Ein Objekt einer erweiterten Exception-Klasse wird auch von einer catch-Klausel der Basis-Klasse gefangen.

```
class FatherException extends Exception { /*...*/ }
class SonException extends FatherException { /*...*/ }

public class FatherAndSon {
    public static void main(String[] args) {
        try { throw new SonException(); }
        catch( SonException s ){
            System.err.println("Caught SonException");
        }
        catch (FatherException f) {
            System.err.println("Caught FatherException");
        }
    }
}
```

Exception Matching (2)

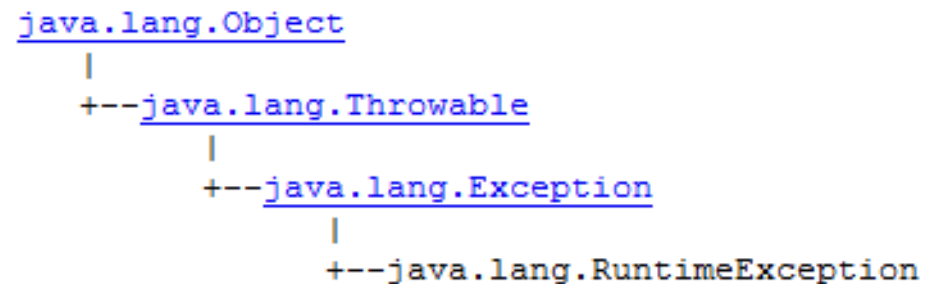
- Die `SonException` wird natürlich von der ersten `catch`-Klausel gefangen.
- Wenn die erste `catch`-Klausel gelöscht wird, so ist der Code immer noch lauffähig, da `catch (FatherException f)` alle `FatherExceptions` oder von ihr abgeleiteten `Exceptions` fängt.
- Das Vertauschen der `catch`-Klauseln zu

```
try {  
    throw new SonException();  
}  
catch (FatherException f) {  
    System.err.println("Caught FatherException");  
}  
catch( SonException s ){  
    System.err.println("Caught SonException");  
}
```

führt zu einem Fehler beim Compilieren, da der Compiler erkennt, dass die `SonException`-catch-Klausel nie erreicht wird.

Runtime-Exceptions

- Bei `RuntimeException` handelt es sich um Laufzeitfehler, die während des Programmverlaufs auftreten können („Programmierfehler“), z.B.
 - fehlerhafte Typkonvertierung (`ClassCastException`)
 - Zugriff über die Arraygrenze hinaus (`ArrayIndexOutOfBoundsException`)
 - `NumberFormatException`
 - Zugriff auf einen leeren Zeiger (`NullPointerException`)
- `RuntimeException` müssen nicht abgefangen werden („unchecked exceptions“)



Erweiterung ab Java 1.7

- Mehrere Exceptions in einem Catch-Block (" | ")

```
try {  
    throwAorB();  
}  
catch( ExceptionA | ExceptionB ex ){  
    throw ex;  
}
```

„oder“