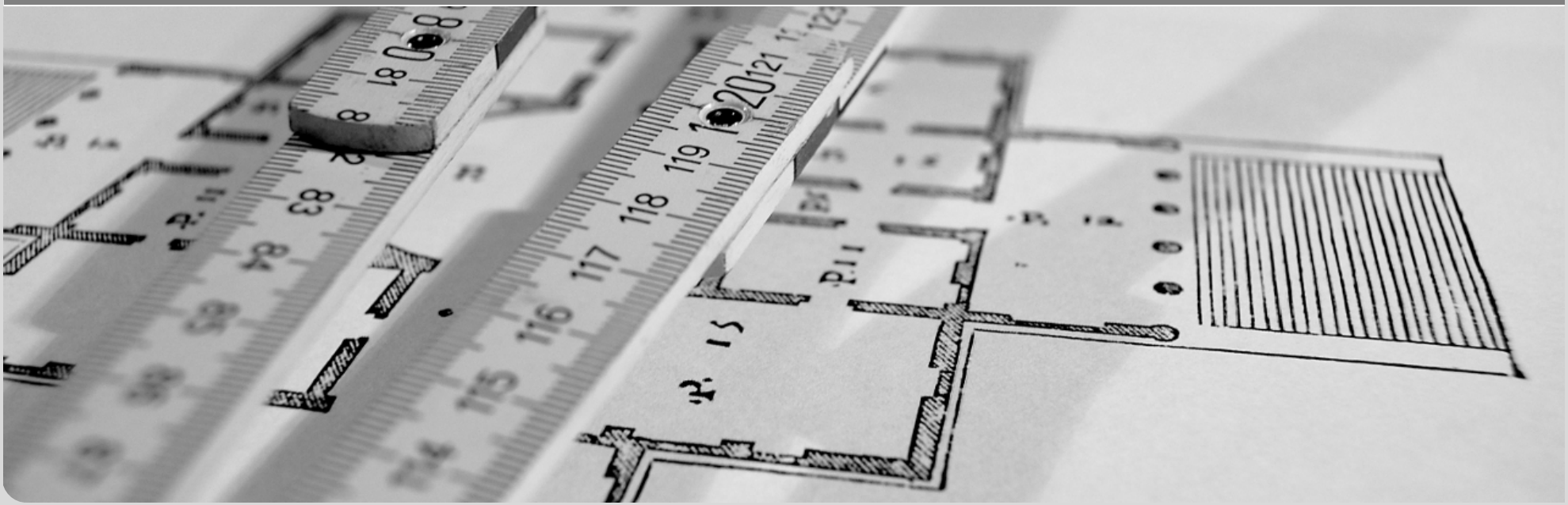
Karlsruher Institut für Technologie

Software Design and Quality

# Programmieren
# 09. Tutorium

Robin Rüde | 19. Januar 2015



# Gliederung

# 1

# Organisatorisches

# Organisatorisches

- Anmeldung zum Übungsschein: 19.01 - 30.01.
- Abschlussaufgaben: 02.02. - 13.02.
- Anmeldung unter `campus.studium.kit.edu`
- NICHT vergessen. Wenn ihr zu spät seid, habt ihr Pech gehabt
- Für Prüfungen allgemein: Abmelden ist leichter als Anmelden

# 2

# Exceptions

# Exceptions

- Ziel: Behandlung von Fehlern
- Unterbrechen den normalen Kontrollfluss
- Werf-und Fangbare Objekte
- Grundklasse `java.lang.Throwable`
- Meist Subklassen von `java.lang.Exception`
- Was ihr sicher schon gesehen habt:
  - IndexOutOfBoundsException
  - NullPointerException
  - StackOverflowError
- All diese sind fangbar

# Exceptions: kleines Beispiel

Alle Exceptions sind fangbar

```
class Main {
    public static void main(String[] args) {
        try {
            runProgram();
        } catch (StackOverflowError e) {
            System.out.println("Program never ends!");
            e.printStackTrace();
            runAlternativeProgram();
        }
    }
}
```

Was aber nicht immer sinnvoll ist.

# Fangen und Werfen von Exceptions

- Werfen: `throw new FooException("Fehlermeldung")`
- eigene Exception erstellen (Klasse, die Exception erweitert)
- Fangen

```
try {
    /* Code */
} catch (FooException e) {
    /* Code im Fehlerfall */
}
```

## Wichtig

NIE allgemein Exception werfen und fangen. Immer entweder eine Exception aus der Java-API (NoSuchElement, ...) oder eine eigene (Erstellen als neue Klasse mit:
`class MyException extends Exception`) werfen.
NIEMALS allgemein Exception fangen. Dies ist nie nötig, wenn euer Programm korrekt programmmiert ist.

# RuntimeException

- Exception
  - **müssen** gefangen werden
  - sonst beschwert sich der Kompiler
  - Beispiel: IOException
  - `extends Exception`
- RuntimeException (`.. extends RuntimeException`)
  - **können** gefangen werden
  - Programm stürzt ab wenn nicht gefangen
  - Beispiel: NullPointerException

# Verwendung

Runtime Exception

- Programmierfehler, die erst zur Laufzeit festgestellet werden können
- kein sinnvolles fortsetzen des Programms möglich
- z. Bsp. NullPointerException

Exception

- behandelbare Programmfehler
- z. Bsp. FileNotFoundException

# Exceptions: Beispiel

```
class TreeElement {
List<TreeElement> children = new ArrayList<>();
    TreeElement getFirstChild() {
        return children.get(0);
    }
}
new TreeElement().getFirstChild();
// Exception in thread main:
// java.lang.IndexOutOfBoundsException
// [stack trace]
```

- getFirstChild ist nicht immer möglich
- was tun?
  - Fehler propagieren / ignorieren (siehe oben)
  - Fehler umgehen
  - Fehler fangen und aufessen

# Beispiel: Fehler fangen und essen

```
class TreeElement {
    List<TreeElement> children = new ArrayList<>();
    TreeElement getFirstChild() {
        try {
            return children.get(0);
        } catch (IndexOutOfBoundsException e) {
            sysout("Doesn't have first child!");
            return null;
        }
    }
}
new TreeElement().getFirstChild();
// => null
```

- **Selten** sinnvoll
- Normalerweise zumindest mit Logausgabe. Sollte bei korrekter Programmausführung nie vorkommen.

# Exceptions: Beispiel: Fehler umgehen

```
class TreeElement {
    List<TreeElement> children = new ArrayList<>();
    TreeElement getFirstChild() {
        if(children.size() == 0) return null;
        else return children.get(0);
    }
}
new TreeElement().getFirstChild();
// => null
```

- Beste Lösung für den Idealfall

# Exceptions: Beispiel

- In der JavaDoc beschreiben was in Sonderfällen passiert!

```
class TreeElement {
    List<TreeElement> children = new ArrayList<>();
    /**
     * get the first child.
     * if there is none, returns null
     */
    TreeElement getFirstChild() {
        if(children.size() == 0) return null;
        else return children.get(0);
    }
}
```

# Exceptions propagieren

Exceptions können auch nach beliebig nach oben weitergegeben werden

```
class Util {
    static String readFirstLine(String fname)
            throws IOException {
        new BufferedReader(new FileReader(fname)).readLine();
    }
}
```

Dann müssen sie vom Aufrufer gefangen werden

# 3

# Debugging

# Debugging in Eclipse

- entweder mit System.out.println Variablen o. ä. ausgeben
- oder mit den Tools die einem Eclipse zur Verfügung stellt
- (Beispiel)

# 4

# Generics (Wdh)

# Generics (Wdh)

Solchen Code:

```
class Something<T extends Comparable<T>> {
    T doSomething() {}
}
```

Vorstellen als:

```
class T extends Comparable<T> {
    // no properties except those of Comparable
}
class Something {
    T doSomething() {}
}
```

Das „T“ hat keine spezielle Bedeutung! Kann beliebiger anderer Identifier sein.

# Generics

Im spitzen Klammern wird immer `extends` verwendet:

```
interface Comparable<T> {
    int compareTo(T other);
}
class Something implements Comparable<Something> {
    //
}
// Aber:
class SomethingElse<S extends Comparable<S>> {
    //
}
```

Außerdem ist `Foo` eine valide Füllung für `? extends Foo`

`...<A extends B>` ist also äquivalent zu

`instanceof A` $\Rightarrow$ `instanceof B`

# Generics

Es können mehrere Klassen als Einschränkung geschrieben werden:

```
class SomethingElse<S extends Comparable<S> & Iterable<S>> {
      //
}
// vorstellen als
class S implements Comparable<S>, Iterable<S> {

}
```

Natürlich nur solange maximal eine echte Klasse vorkommt, der Rest Interfaces

# Generics

? steht für „beliebiger Typ“

```
Collection<String> getValues(Map<?, String> map) {
    return map.values();
}
```

Verwendung genauso wie andere Buchstaben

Aber: Jedes Fragezeichen steht für einen anderen Typen.

$\Rightarrow$ Nach Definition nicht mehr verwendbar

# Generics

super heißt „eine beliebige Überklasse“

```
class java.util.Collections {
    static
    <T extends Comparable<? super T>>
    void sort(List<T> list)
}

class Foo implements Comparable<Foo> { }
class Bar extends Foo { }
...
Bar bar = new Bar();
bar instanceof Comparable<Bar> == false
List<Bar> list = new List<Bar>();
Collections.sort(list);
```

# Generics (größeres Beispiel)

Generics können beliebig verschachtelt / komplex werden:

```
public class Attribute {
    private final String name;
...
}
public class DataVector<A extends Attribute> {
    private final String name;
    private final Map<A, Double> values;
    ...
}
```

# Beispiel (Fortsetzung)

```
public class DataStore<V extends DataVector<A>, A extends Attribute> {
    private final List<V> vectors;
    private HashMap<A, List<SplitInformation<V, A>>>
        discretizedAttributes;
        ...
}
public abstract class DecisionTreeMiningLibrary<V extends DataVector<A>,
        A extends Attribute>
        implements IMiningLibrary<V, A> {

    void mineDecisionTree(List<V> vectors, List<A> attributes,
        Tree<DecisionTreeElement<A>, V> parent,
        HashMap<A, List<SplitInformation<V, A>>>
            discretizedAttributes) {
        // ...
    }
}
```

# 5

# Aufgaben

# Aufgabe 1: Eigene Exception

Schreibe ein Programm, welches beim Aufruf der Funktion `runtimeExceptionTest` die RuntimeException `MyRuntimeException` mit der Nachricht `Laufzeitfehler` wirft.
Weiterhin soll eine weitere Methode `exceptionTest` eine Instanz der Klasse `MyException` werfen
Schreibe nun eine main-Methode, die beide Methoden aufruft und die entsprechende Exception fängt und dann die Stacktrace (`exception.printStackTrace()`) ausgibt.

# Verwendung von Exceptions

Welche Art von Exception ist hier sinnvoll und warum?

- IllegalArgumentException - Exception oder RuntimeException?

# Verwendung von Exceptions

Welche Art von Exception ist hier sinnvoll und warum?

- IllegalArgumentException - Exception oder RuntimeException?
- NullPointerException - Exception oder RuntimeException?

# Verwendung von Exceptions

Welche Art von Exception ist hier sinnvoll und warum?

- IllegalArgumentException - Exception oder RuntimeException?
- NullPointerException - Exception oder RuntimeException?
- FileNotFoundException - Exception oder RuntimeException?s

# Verwendung von Exceptions

Welche Art von Exception ist hier sinnvoll und warum?

- IllegalArgumentException - Exception oder RuntimeException?
- NullPointerException - Exception oder RuntimeException?
- FileNotFoundException - Exception oder RuntimeException?s
- IOException - Exception oder RuntimeException?

# Ende

Fragen?

# Ende

Fragen?
Vielen Dank für eure Aufmerksamkeit!