



From good to great!

Gutes kann noch besser werden,
Sortiments- und Platzierungsoptimierung in einer neuen Dimension.

<https://github.com/HInformatikAG>



it-ag@hoffrogge.com

Lehreinheit 5

Ziele

- Interfaces
- Collections
- Vererbung
- Exception Handling
- Testen mit JUnit
- Tetromino Basisklasse

Interfaces

Muster für Klassen

- Ein Interface ist eine besondere Art einer Klasse
- Das Interface definiert, welche Aufgaben (Methoden) eine Klasse ausführen kann
 - Aber nicht, wie die Klasse diese Methoden ausführt
- Jede Klasse, die dieses Muster übernimmt, implementiert das Interface
- Vorteil: Viele verschiedene Klassen, die aber alle die gleichen Methoden haben
 - Sie können natürlich noch andere Methoden haben, aber sie haben auf jeden Fall die Methoden des Interfaces

Interfaces

```
package com.hoffrogge.lehreinheit04;

import java.awt.Graphics;

import com.hoffrogge.lehreinheit03.Farbe;

public interface GeometrischeFigur {

    void setMittelpunkt(int x, int y);

    Punkt getMittelpunkt();

    void setDurchmesser(int d);

    void setLinienFarbe(Farbe farbe);

    void zeichnen(Graphics graphics);
}
```

Interfaces

```
import java.awt.Graphics;  
import com.hoffrogge.lehreinheit03.Farbe;
```

```
public class Quadrat implements GeometrischeFigur {
```

```
    private int mittelpunktX;  
    private int mittelpunktY;  
    private int durchmesser;
```

```
    private Farbe farbe;
```

```
    public Quadrat() {  
        farbe = new Farbe(0, 0, 0);  
    }
```

```
    @Override  
    public void zeichnen(Graphics graphics) {  
        /*Quellcode zum Zeichnen*/  
    }
```

```
    @Override  
    public void setMittelpunkt(int x, int y) {  
        this.mittelpunktX = x;  
        this.mittelpunktY = y;  
    }
```

```
    @Override  
    public Punkt getMittelPunkt() {  
        return new Punkt(mittelpunktX, mittelpunktY);  
    }
```

```
    /*weitere Methoden aus dem Interface*/  
}
```

Implementiert das Interface

Interfaces

```
import java.awt.Graphics;
import com.hoffrogge.lehreinheit03.Farbe;

public class Quadrat implements GeometrischeFigur {

    private int mittelpunktX;
    private int mittelpunktY;
    private int durchmesser;

    private Farbe farbe;

    public Quadrat() {
        farbe = new Farbe(0, 0, 0);
    }

    @Override
    public void zeichnen(Graphics graphics) {
        /*Quellcode zum Zeichnen*/
    }

    @Override
    public void setMittelpunkt(int x, int y) {
        this.mittelpunktX = x;
        this.mittelpunktY = y;
    }

    @Override
    public Punkt getMittelPunkt() {
        return new Punkt(mittelpunktX, mittelpunktY);
    }

    /*weitere Methoden aus dem Interface*/
}
```



Methode aus dem Interface

Interfaces

- Klasse Quadrat implementiert das Interface GeometrischeFigur
- Klasse Dreieck implementiert das Interface GeometrischeFigur
- Klasse Kreis implementiert das Interface GeometrischeFigur
- Klasse Rechteck implementiert das Interface GeometrischeFigur
- Klasse Stern implementiert das Interface GeometrischeFigur
- Klasse Oktagon implementiert das Interface GeometrischeFigur

Interfaces

- Alle Klassen haben die Methoden aus dem Interface
- Man kann also alle Klassen nach dem Mittelpunkt fragen (`getMittelpunkt()`)
- Oder sie zeichnen (`zeichnen()`)
- Die Klasse Zeichenbrett kann jetzt geometrische Figuren nehmen und zeichnen
- Dabei ist es egal, ob es ein Quadrat oder ein Kreis oder eine andere geometrische Figur ist

Collections

- Collections in Java sind „Sammlungen“ von Objekten
- Wir wollen List und Set nutzen
- Beispiele:
 - `List listeVonRechtecken = new ArrayList();`
 - `Set setVonRechtecken = new HashSet();`

Collections

- List und Set sind Implementierungen des Interfaces Collection
 - <https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html>
- Einige Methoden von Collection:
 - add(Object einObjekt)
 - remove(Object einObjekt)
 - clear()
 - size()

- Das heißt List und Set können
 - add(Object einObjekt)
 - remove(Object einObjekt)
 - clear()
 - size()

- List
 - Eine Liste von Objekten
 - Reihenfolge bleibt erhalten
 - Es können mehrere gleiche Objekte in der Liste sein

■ Set

- Eine unsortierte Sammlung von Objekten
- Reihenfolge bleibt NICHT erhalten
- Es können KEINE gleichen Objekte im Set sein
 - Zwei Quadrate mit Kantenlänge 5 und 10 sind ok
 - Aber zwei Quadrate, die beide die Kantenlänge 10 haben, passen nicht
 - Es wird nur eins davon übernommen

- In Java kann eine Klasse von genau einer anderen Klasse erben
- Das heißt, dass die Kindklasse alles kann, was auch die Elternklasse kann
 - Falls es sichtbar ist, also nicht private, sondern höher ist

Vererbung

Beispiel

- Ein Quadrat ist ein spezielles Rechteck
- Wir kennen die Klasse Rechteck
- Wir können eine neue Klasse Quadrat erstellen, die von Rechteck ableitet
- `public class Quadrat extends Rechteck`
- Die Klasse Quadrat kann dann alles, was ein Rechteck kann
 - Z. B. eine Diagonale berechnen
- Aber bei einem Quadrat sind immer beide Kanten gleich lang

Vererbung

Beispiel

```
package com.hoffrogge.lehreinheit05;

import com.hoffrogge.lehreinheit03.Rechteck;

public class Quadrat extends Rechteck {

    public Quadrat(int kantenlaenge) {

        setBreite(kantenlaenge);
        setLaenge(kantenlaenge);

        System.out.println(berechneDiagonale());
    }
}
```

Vererbung

Beispiel

```
package com.hoffrogge.lehreinheit05;

import com.hoffrogge.lehreinheit03.Rechteck;

public class Quadrat extends Rechteck {

    public Quadrat(int kantenlaenge) {

        setBreite(kantenlaenge);
        setLaenge(kantenlaenge);

        System.out.println(berechneDiagonale());
    }
}
```



Rechteck aus Lehreinheit 03

Vererbung

Beispiel

```
package com.hoffrogge.lehreinheit05;

import com.hoffrogge.lehreinheit03.Rechteck;

public class Quadrat extends Rechteck {

    public Quadrat(int kantenlaenge) {

        setBreite(kantenlaenge);
        setLaenge(kantenlaenge);

        System.out.println(berechneDiagonale());
    }
}
```



Quadrat erbt von Rechteck

Vererbung

Beispiel

```
package com.hoffrogge.lehreinheit05;

import com.hoffrogge.lehreinheit03.Rechteck;

public class Quadrat extends Rechteck {

    public Quadrat(int kantenlaenge) {

        setBreite(kantenlaenge);
        setLaenge(kantenlaenge);

        System.out.println(berechneDiagonale());
    }
}
```



Quadrat braucht nur eine
Kantenlänge

Vererbung

Beispiel

```
package com.hoffrogge.lehreinheit05;

import com.hoffrogge.lehreinheit03.Rechteck;

public class Quadrat extends Rechteck {

    public Quadrat(int kantenlaenge) {

        setBreite(kantenlaenge);
        setLaenge(kantenlaenge);

        System.out.println(berechneDiagonale());
    }
}
```



kann wie das Rechteck die
Diagonale berechnen

Exception Handling

- Exceptions (mögliche Fehler) haben wir schon kennengelernt
 - `ArrayIndexOutOfBoundsException`
 - `NullPointerException`
 - `ArithmeticException`
 - `ClassCastException`

Exception Handling

- Es gibt checked und unchecked Exceptions
 - Checked Exceptions sind Fehler, mit denen man rechnet
 - Unchecked Exceptions sind Fehler, mit denen man nicht rechnet

Exception Handling

- Die bekannten sind unchecked Exceptions
 - Z. B. `ArithmeticException` (teilen durch 0)
 - Methode `teileADurchB(int a, int b)` ist erstmal korrekt
 - Wirft aber einen Fehler, sobald $b = 0$ ist

Exception Handling

- Eine checked Exception ist z. B. die IOException
 - Tritt auf, wenn bei einer Dateioperation ein Fehler auftritt
 - Z. B. Öffnen einer Datei, die nicht existiert
- Exceptions müssen entweder *geworfen* oder *gefangen* werden

Exception Handling

Exception werfen

```
public void erstelleDatei() throws IOException {  
  
    File file = new File("test.txt");  
    file.createNewFile();  
}
```

Exception Handling

Exception werfen

```
public void erstelleDatei() throws IOException {
```

```
    File file = new File("test.txt");  
    file.createNewFile();
```

```
}
```



Wirft Exception

Exception Handling

- Eine geworfene Exception muss an anderer Stelle behandelt werden

Exception Handling

Exception fangen

- Wenn eine Exception nicht geworfen wird, muss sie gefangen werden
- Exception fangen bedeutet, den Fehler zu behandeln/zu beheben
- Eine Exception kann beliebig weit geworfen werden
 - Aber irgendwo muss sie am Ende gefangen werden

Exception Handling

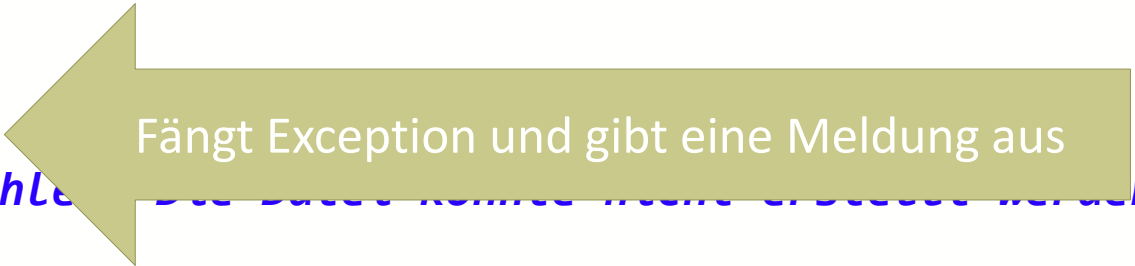
Exception fangen

```
public void erstelleDatei() {  
  
    File file = new File("test.txt");  
  
    try {  
  
        file.createNewFile();  
  
    } catch (IOException e) {  
        System.err.println("Fehler! Die Datei konnte nicht erstellt werden!");  
    }  
}
```

Exception Handling

Exception fangen

```
public void erstelleDatei() {  
    File file = new File("test.txt");  
  
    try {  
        file.createNewFile();  
    } catch (IOException e) {  
        System.err.println("Fehler: Die Datei konnte nicht erstellt werden!");  
    }  
}
```



Fängt Exception und gibt eine Meldung aus

Testen mit JUnit

assertEquals

- JUnit ist ein Framework (Zusatzsoftware) zum Testen in Java
- Es wird ein Test erstellt, der testet, ob eine Methode genau das Ergebnis liefert, das erwartet wird
- Beispiel Rechteck, die Diagonale (d) berechnet sich wie folgt: $d = \sqrt{a^2 + b^2}$
 - Bei einem Rechteck mit den Kantenlängen $a=1$ und $b=1$ ist die Diagonale $d = \sqrt{1^2 + 1^2} = \sqrt{2}$, also ungefähr 1,414
- Ein JUnit Test kann jetzt prüfen, ob das erwartete Ergebnis mit dem tatsächlichen Ergebnis übereinstimmt
- `assertEquals(1.414, berechneteDiagonale)`

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;
import com.hoffrogge.lehreinheit03.Rechteck;
import junit.framework.TestCase;

public class RechteckTest extends TestCase {

    public void testDiagonale() {

        Rechteck rechteck = new Rechteck(1, 1);
        double diagonale = rechteck.berechneDiagonale();
        assertEquals(1.4142135623730951, diagonale);

        rechteck = new Rechteck(17, 4);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(17.46424919657298, diagonale);

        rechteck = new Rechteck(13, 37);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(39.21734310225516, diagonale);

    }
}
```

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;
import com.hoffrogge.lehreinheit03.Rechteck;
import junit.framework.TestCase;

public class RechteckTest extends TestCase {

    public void testDiagonale() {

        Rechteck rechteck = new Rechteck(1, 1);
        double diagonale = rechteck.berechneDiagonale();
        assertEquals(1.4142135623730951, diagonale);

        rechteck = new Rechteck(17, 4);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(17.46424919657298, diagonale);

        rechteck = new Rechteck(13, 37);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(39.21734310225516, diagonale);

    }
}
```



Rechteck aus LE 3

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;  
import com.hoffrogge.lehreinheit03.Rechteck;  
import junit.framework.TestCase;
```

```
public class RechteckTest extends TestCase {
```

```
    public void testDiagonale() {
```

```
        Rechteck rechteck = new Rechteck(1, 1);  
        double diagonale = rechteck.berechneDiagonale();  
        assertEquals(1.4142135623730951, diagonale);
```

```
        rechteck = new Rechteck(17, 4);  
        diagonale = rechteck.berechneDiagonale();  
        assertEquals(17.46424919657298, diagonale);
```

```
        rechteck = new Rechteck(13, 37);  
        diagonale = rechteck.berechneDiagonale();  
        assertEquals(39.21734310225516, diagonale);
```

```
    }
```

```
}
```



Test erbt von TestCase

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;
import com.hoffrogge.lehreinheit03.Rechteck;
import junit.framework.TestCase;

public class RechteckTest extends TestCase {

    public void testDiagonale() {

        Rechteck rechteck = new Rechteck(1, 1);
        double diagonale = rechteck.berechneDiagonale();
        assertEquals(1.4142135623730951, diagonale);

        rechteck = new Rechteck(17, 4);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(17.46424919657298, diagonale);

        rechteck = new Rechteck(13, 37);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(39.21734310225516, diagonale);

    }
}
```



Ein JUnit Testfall

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;  
import com.hoffrogge.lehreinheit03.Rechteck;  
import junit.framework.TestCase;
```

```
public class RechteckTest extends TestCase {
```

```
    public void testDiagonale() {
```

```
        Rechteck rechteck = new Rechteck(1, 1);  
        double diagonale = rechteck.berechneDiagonale();  
        assertEquals(1.4142135623730951, diagonale);
```

```
        rechteck = new Rechteck(17, 4);  
        diagonale = rechteck.berechneDiagonale();  
        assertEquals(17.46424919657298, diagonale);
```

```
        rechteck = new Rechteck(13, 37);  
        diagonale = rechteck.berechneDiagonale();  
        assertEquals(39.21734310225516, diagonale);
```

```
    }
```

```
}
```



Rechteck wird erstellt

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;
import com.hoffrogge.lehreinheit03.Rechteck;
import junit.framework.TestCase;

public class RechteckTest extends TestCase {

    public void testDiagonale() {

        Rechteck rechteck = new Rechteck(1, 1);
        double diagonale = rechteck.berechneDiagonale();
        assertEquals(1.4142135623730951, diagonale);

        rechteck = new Rechteck(17, 4);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(17.46424919657298, diagonale);

        rechteck = new Rechteck(13, 37);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(39.21734310225516, diagonale);

    }
}
```



Diagonale wird berechnet

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;  
import com.hoffrogge.lehreinheit03.Rechteck;  
import junit.framework.TestCase;
```

```
public class RechteckTest extends TestCase {
```

```
    public void testDiagonale() {
```


```
        Rechteck rechteck = new Rechteck(1, 1);  
        double diagonale = rechteck.berechneDiagonale();  
        assertEquals(1.4142135623730951, diagonale);
```

```
        rechteck = new Rechteck(17, 4);  
        diagonale = rechteck.berechneDiagonale();  
        assertEquals(17.46424919657298, diagonale);
```

```
        rechteck = new Rechteck(13, 37);  
        diagonale = rechteck.berechneDiagonale();  
        assertEquals(39.21734310225516, diagonale);
```

```
    }
```

```
}
```



Vergleichen mit
erwartetem Wert

Testen mit JUnit

Ein Beispiel

```
package com.hoffrogge.lehreinheit05;
import com.hoffrogge.lehreinheit03.Rechteck;
import junit.framework.TestCase;

public class RechteckTest extends TestCase {

    public void testDiagonale() {

        Rechteck rechteck = new Rechteck(1, 1);
        double diagonale = rechteck.berechneDiagonale();
        assertEquals(1.4142135623730951, diagonale);

        rechteck = new Rechteck(17, 4);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(17.46424919657298, diagonale);

        rechteck = new Rechteck(13, 37);
        diagonale = rechteck.berechneDiagonale();
        assertEquals(39.21734310225516, diagonale);

    }
}
```



Weitere Beispiele testen

Testen mit JUnit

- RechteckTest in Eclipse ausführen

Testen mit JUnit

fail

- Wenn ein Test bestimmte Voraussetzungen nicht erfüllt, dann soll er fehlschlagen
- *fail(„Erklärung, warum der Test fehlschlägt.“)*

Tetromino Basisklasse

- Aufgabe: Implementiere eine Tetromino Basisklasse
- Ein Tetromino ist ein Tetris-Spielstein
- Die Klasse muss also alle Methoden haben, die jeder Spielstein braucht
- Tipp: fange mit der Klasse GeometrischeFigur an

Vielen Dank!

