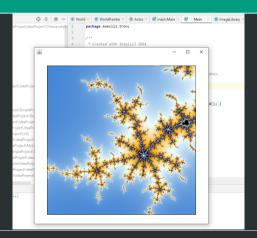
## FOP Recap #9



#### **Exceptions**



# Willkommen zurück!

#### Das steht heute auf dem Plan



Primitiven Wert in String umwandeln

Mehrdimensionale Arrays

Exceptions

assert

JUnit-Tests



```
int value = 0;
String stringValue = value; // ERROR!
```



```
int value = 0;
String stringValue = "" + value; // OK
```



```
int value = 0;
String stringValue = String.valueOf(value); // Super
```



```
int value = 0;
String stringValue = Integer.toString(value); // Exotisch
```

#### Das steht heute auf dem Plan



Primitiven Wert in String umwandeln

Mehrdimensionale Arrays
2D Initialisierung
2D Zugriff auf Elemente
2D Beispiel

Exceptions

assert

JUnit-Tests



- Haben als Komponentypen wieder Arrays
- Typischerweise 2D oder 3D Arrays
- Zugriff über mehrere Indices
- Wichtig: Subarrays müssen nicht die gleiche Länge haben
- Zusätzliche Kniffe fürs Erstellen

2D Initialisierung



```
int[][] bigArray = new int[2][5]:
int[][] bigArray = new int[2][]:
bigArray[0] = new int[5];
bigArray[1] = new int[5];
int[][] bigArray = new int[2][];
bigArray[0] = new int[] {0, 0, 0, 0, 0};
bigArray[1] = new int[] \{0, 0, 0, 0, 0\};
int[][] bigArray = new int[][] {{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}};
```

2D Zugriff auf Elemente



```
int[][] bigArray = new int[2][5];
```

```
int[] smallArray = bigArray[0];
```

- smallArray[2] = 3;
- bigArray[0][2] = 3;

2D Zugriff auf Elemente



```
int[][] bigArray = new int[2][5];
```

```
int[] smallArray = bigArray[0];
```

- int value = smallArray[2];
- int value = bigArray[0][2];

# Mehrdimensionale Arrays 2D Beispiel



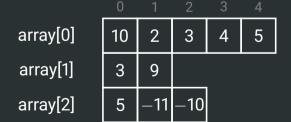
```
int[][] array = new int[3][];
array[0] = new int[] {1, 2, 3, 4, 5};
array[1] = new int[] {3, 9};
array[2] = new int[] {5, -11, 8};
```



# Mehrdimensionale Arrays 2D Beispiel



```
int[][] array = ...;
array[0][0] = 10;
array[2][2] = -10;
```



# Mehrdimensionale Arrays 2D Beispiel



#### Das steht heute auf dem Plan



Primitiven Wert in String umwandeln

Mehrdimensionale Arrays

# Exceptions Einfaches Beispiel Besseres Beispiel Mehrere Exceptions Falsche Reihenfolge der Catch-Blöcke Korrekte Reihenfolge der Catch-Blöcke Mehrere Exceptions Weiterreichen Ausnahme RuntimeException



- Grobe Unterscheidung zwischen
  - Fehlern
  - Laufzeitfehlern
- In Java: Viele Fehler bereits beim Kompilieren erkennbar
- Laufzeitfehler sind:
  - Fehler die erst während der Laufzeit auftreten
  - Fehler, die situationsbedingt auftreten können
- Bereits bekannte Laufzeitfehler:
  - NullPointerException
  - arrayIndexOutOfBoundsException
  - ClassCastException
  - IOException
  - о ...
- Erben alle direkt oder indirekt von Exception

# Exceptions Einfaches Beispiel



```
public int buyCoolHat(int amountOfMoney) throws Exception {
   if(amountOfMoney < 50) {
      throw new Exception("Insufficient funds!");
}

System.out.println("Good choice.");
return amountOfMoney - 50;
}</pre>
```

**Einfaches Beispiel** 



```
public int buyCoolHat(int amountOfMoney) throws Exception {
  int myMoney = 70:
  trv {
      myMoney = buyCoolHat(myMoney);
  catch(Exception e) {
      e.printStackTrace():
8
```

**Einfaches Beispiel** 



```
public int buyCoolHat(int amountOfMoney) throws
      NullPointerException {
       throw new NullPointerException("Not available!");
  int myMoney = 70;
  trv {
      mvMonev = buvCoolHat(mvMonev):
  catch(NullPointerException e) {
      e.printStackTrace():
8
```

# Exceptions Besseres Beispiel



```
public int buyCoolHat(int amountOfMoney) throws

→ InsufficientFundsException {
  int myMoney = 70;
  trv {
      myMoney = buyCoolHat(myMoney);
  catch(InsufficientFundsException e) {
      e.printStackTrace();
8
```

**Besseres Beispiel** 



```
public int buyCoolHat(int amountOfMoney) throws

    □ InsufficientFundsException {
    if(amountOfMoney < 50) {</pre>
        throw new InsufficientFundsException(50 - amountOfMoney);
public class InsufficientFundsException extends Exception {
    public InsufficientFundsException(int missingMoney) {
        super("Insufficient funds! Needed " + missingMoney +
         → more money."):
```



```
public void test() throws NullPointerException.
   try {
      test();
  catch(NullPointerException e) {
      e.printStackTrace():
  catch(ClassCastException e) {
      e.printStackTrace();
8
9
```



```
public void test() throws NullPointerException,
   trv {
     test();
  catch(NullPointerException | ClassCastException e) {
      e.printStackTrace():
5
```



```
public void test() throws NullPointerException,
   trv {
     test();
  catch(Exception e) {
     e.printStackTrace();
5
```

Falsche Reihenfolge der Catch-Blöcke



```
public void test() throws NullPointerException,
      ClassCastException {
   try {
       test();
   catch(Exception e) {
       e.printStackTrace();
   catch(NullPointerException e) {
       System.out.println("Null!"); // !!!
9
```

Korrekte Reihenfolge der Catch-Blöcke



```
trv {
   test();
catch(NullPointerException e) {
   System.out.println("Null!");
catch(Exception e) {
   e.printStackTrace():
```

public void test() throws NullPointerException,



```
public void test() throws Exception {
       throw new NullPointerException("Message"):
   try {
       test();
   catch(NullPointerException | ClassCastException e) {
       e.printStackTrace():
   catch(Exception e) {
       System.out.println("You need this :)"):
8
```

Weiterreichen



- Weiterreichen/Weiterleiten von Exceptions ist (unter anderem) dann sinnvoll, wenn
  - der Fehler nicht an dieser Stelle gelöst werden kann
  - 🗖 es eine höhere zentrale Stelle zum Sammeln von Fehlern gibt

Weiterreichen



```
public void test() throws NullPointerException,
      ClassCastException {
       throw new NullPointerException("Message");
   public void run() throws ClassCastException {
       trv {
          test():
      catch(NullPointerException e) { }
      catch(ClassCastException e) {
          throw e:
9
```

Weiterreichen



```
public void test() throws NullPointerException,
       ClassCastException {
       throw new NullPointerException("Message");
   public void run() throws ClassCastException {
       trv {
          test();
      catch(NullPointerException e) {
          e.printStackTrace();
9
```

Ausnahme RuntimeException



- ACHTUNG! Ausnahme!
- Alle Exceptions die von RuntimeException direkt oder indirekt erben
  - Müssen nicht mit throws deklariert werden
  - Müssen nicht mit einem try-and-catch Block abgeprüft werden
- Hierzu zählen zum Beispiel
  - NullPointerException
  - ArrayIndexOutOfBoundsException
  - ClassCastException
  - ····

**Javadoc** 



```
* @param amountOfMoney The amount of money available
    * @return Amount of money left after transaction
    * @throws InsufficientFundsException If there is not enough
     → money available
   public int buyCoolHat(int amountOfMoney) throws

    □ InsufficientFundsException {
       if(amountOfMonev < 50) {</pre>
           throw new InsufficientFundsException(50 - amountOfMoney);
10
       return amountOfMoney - 50;
```

**Typische Fehler** 



Typische Fehler



```
public static int tryToCall() {
    // ....
}
```

**Typische Fehler** 



```
public static int tryToCall() {
       try {
          int returnValue = X.calculateTheAnswer();
      catch(IllegalStateException e) {
           throw new RuntimeException("Look at this: " +
6
           → e.getMessage());
       return returnValue: // ERROR!
```

**Typische Fehler** 



```
public static int tryToCall() {
       int returnValue;
       trv {
           returnValue = X.calculateTheAnswer();
5
       catch(IllegalStateException e) {
           throw new RuntimeException("Look at this: " +
            → e.getMessage());
       return returnValue:
10
```

**Typische Fehler** 



Vererbung



- ExceptionA erbt direkt oder indirekt von Exception
- ExceptionB erbt direkt oder indirekt von ExceptionA

Vererbung



```
public class B extends A {
   @Override
   public void myMethod() throws ExceptionA {
        // Gleicher Exception-Typ erlaubt
public class C extends A {
   @Override
   public void myMethod() throws ExceptionB {
        // "Präzisierung" erlaubt
```

## Das steht heute auf dem Plan



Primitiven Wert in String umwandeln

Mehrdimensionale Arrays

Exceptions

assert

JUnit-Tests

#### assert



- Einfache und kurze Möglichkeit um Fehler zu werfen
- Wirft immer eine AssertionError, der von Error abgeleitet ist
- Error erbt weder von Exception noch RuntimeException
- Ist im Normalfall deaktiviert!
- Muss mittels -enableassertions bzw -ea aktiviert werden

#### assert



public static void dummyTest(int b) {

## **Kontroverse: Exceptions in Java**





## Das steht heute auf dem Plan



Primitiven Wert in String umwandeln

Mehrdimensionale Arrays

Exceptions

assert

JUnit-Tests
Einfaches Beispiel
assertDoesNotThrow
assertThrowsExactly

#### **JUnit-Tests**



- Framework um einfache aber auch sehr komplexe Tests für Java-Programme zu schreiben
- Library muss erst eingebunden werden
- Werden auch in Java geschrieben

#### **JUnit-Tests**

**Einfaches Beispiel** 



```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class StudentTests {
    ....
}
```

#### **JUnit-Tests**

**Einfaches Beispiel** 



```
@Test
   public void myTest() {
       Tree tree = new LemonTree();
       Apple unexpectedFruit = new Apple(5);
       assertTrue(tree.hasFruit());
       assertTrue(tree.getFruit() instanceof Lemon):
       assertNotEquals(unexpectedFruit, tree.getFruit()):
       assertEquals(2. tree.getFruitAmount()):
10
```

## JUnit-Tests assertDoesNotThrow



```
@Test
public void myTest() {
   HatFactory factory = new HatFactory();
    int moneyLeft = assertDoesNotThrow(
        () -> factory.buyCoolHat(52)
    assertEquals(moneyLeft, 2, "Incorrect money left!");
```

## JUnit-Tests assertThrowsExactly



```
@Test
public void myTest() {
    HatFactory factory = new HatFactory():
    Throwable thrownException = assertThrowsExactly(
        InsufficientFundsException.class,
        () -> factory.buyCoolHat(49)
    String message = thrownException.getMessage();
```

# **Live-Coding!**