

# FOP Recap #8

## Exceptions



# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Switch

Statement

Expression

Primitiven Wert in String umwandeln

Exceptions

assert

JUnit-Tests

Boxing



- Bessere Alternative zu gigantischen `if-else`-Konstrukten
- Funktionieren mit
  - ▣ Ganzzahligen Zahlen: `byte`, `short`, `char`, `int`
  - ▣ Strings
  - ▣ Enums

# Switch

Statement — Simuliert mit if-else



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  int j = 5;
```

```
1  if(j == 0) {  
2      // ....  
3  }  
4  else if(j == 1) {  
5      // ....  
6  }  
7  else if(j == 2) {  
8      // ....  
9  }  
10 else ....
```

# Switch

## Statement — Mit break



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  switch(j) {  
2      case 0:  
3          System.out.println("ZERO!");  
4          break;  
5      case 1:  
6          System.out.println("ONE!");  
7          break;  
8  
9      ....  
10  
11     default:  
12         System.out.println("DEFAULT!");  
13 }
```

- $j = 0 \rightarrow \text{ZERO!}$
- $j = 1 \rightarrow \text{ONE!}$
- $j = -1 \rightarrow \text{DEFAULT!}$

# Switch

## Statement — Ohne break



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  switch(j) {  
2      case 0:  
3          System.out.println("ZERO!");  
4          // break;  
5      case 1:  
6          System.out.println("ONE!");  
7          // break;  
8  
9      ....  
10  
11     default:  
12         System.out.println("DEFAULT!");  
13 }
```

- $j = 0 \rightarrow$ 
  - ▣ ZERO!
  - ▣ ONE!
  - ▣ ....
  - ▣ DEFAULT!
- $j = 1 \rightarrow$ 
  - ▣ ONE!
  - ▣ ....
  - ▣ DEFAULT!
- $j = -1 \rightarrow$ 
  - ▣ DEFAULT!

# Switch

## Statement — Enhanced Switch



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  switch(j) {  
2      case 0 -> {  
3          System.out.println("ZERO!");  
4          break; // nicht notwendig  
5      }  
6      case 1 -> {  
7          System.out.println("ONE!");  
8      }  
9      ....  
10     default ->  
11         ↪ System.out.println("DEFAULT!");  
12         // keine geschweiften Klammern  
13 }
```

- $j = 0 \rightarrow \text{ZERO!}$
- $j = 1 \rightarrow \text{ONE!}$
- $j = -1 \rightarrow \text{DEFAULT!}$

# Switch

## Expression – Warum?



```
1 String result;  
2  
3 switch(j) {  
4     case 0:  
5         result = "YES";  
6         break;  
7     case 1:  
8         result = "OK";  
9         break;  
10    default:  
11        result = "MAYBE";  
12 }
```



# Switch

Expression – Mit ->



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String result;  
2  
3 result = switch(j) {  
4     case 0 -> "YES";  
5     case 1 -> "OK";  
6     default -> "MAYBE";  
7 };
```

# Switch

## Expression – Mit yield



```
1 String result;  
2  
3 result = switch(j) {  
4     case 0 -> {  
5         yield "YES";  
6     }  
7     case 1 -> {  
8         // ....  
9         yield "OK";  
10    }  
11    default -> {  
12        yield "MAYBE";  
13    }  
14 };
```

# Switch

## Expression – Mit `yield`



```
1 String result;  
2  
3 result = switch(j) {  
4     case 0:  
5         // ....  
6         yield "YES";  
7     case 1:  
8         // ....  
9         yield "OK";  
10    default:  
11        // ....  
12        yield "MAYBE";  
13    };
```

# Switch

Expression — Achtung: Nicht -> und : vermischen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String result;  
2  
3 result = switch(j) {  
4     case 0 -> {  
5         yield "YES";  
6     }  
7     default: // Error: Different case kinds used in the switch  
8         yield "MAYBE";  
9 };
```

# Switch

Expression – Achtung: break vergessen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String result = switch(new Random().nextInt(3)) {  
2     case 0:  
3         result = "YES";  
4     case 1:  
5         result = "OK";  
6     default:  
7         result = "MAYBE";  
8 }  
9 System.out.println(result);
```

```
$ "MAYBE"
```

- ist dann sinnvoll, wenn man den gleichen Code für mehrere Fälle ausführen möchte.
- Bei -> Syntax kann man auch mehrere Fälle angeben, diese werden dann mit , getrennt.

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Switch

Primitiven Wert in String umwandeln

Exceptions

assert

JUnit-Tests

Boxing

# Primitiven Wert in String umwandeln



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

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

# Primitiven Wert in String umwandeln



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

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



# Primitiven Wert in String umwandeln



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

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

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Switch

Primitiven Wert in String umwandeln

Exceptions

- Beispiele

- Catch-Blöcke

- Weiterreichen

- RuntimeException

- Javadoc

- Typische Fehler

- Vererbung

assert

JUnit-Tests

Boxing



- 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`
  - ▣ ....
- Erben alle direkt oder indirekt von `Exception`

# Exceptions

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public int buyCoolHat(int amountOfMoney) throws Exception {  
2     if(amountOfMoney < 50) {  
3         throw new Exception("Insufficient funds!");  
4     }  
5  
6     System.out.println("Good choice.");  
7     return amountOfMoney - 50;  
8 }
```

# Exceptions

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public int buyCoolHat(int amountOfMoney) throws Exception {  
2     // ....  
3 }
```

```
1 int myMoney = 70;  
2  
3 try {  
4     myMoney = buyCoolHat(myMoney);  
5 }  
6 catch(Exception e) {  
7     e.printStackTrace();  
8 }
```

# Exceptions

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public int buyCoolHat(int amountOfMoney) throws  
   ↳ NullPointerException {  
2     throw new NullPointerException("Not available!");  
3 }
```

```
1 int myMoney = 70;  
2  
3 try {  
4     myMoney = buyCoolHat(myMoney);  
5 }  
6 catch(NullPointerException e) {  
7     e.printStackTrace();  
8 }
```

# Exceptions

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public int buyCoolHat(int amountOfMoney) throws
   ↳ InsufficientFundsException {
2     // ....
3 }
```

```
1 int myMoney = 70;
2
3 try {
4     myMoney = buyCoolHat(myMoney);
5 }
6 catch(InsufficientFundsException e) {
7     e.printStackTrace();
8 }
```

# Exceptions

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public int buyCoolHat(int amountOfMoney) throws
   ↳ InsufficientFundsException {
2     if(amountOfMoney < 50) {
3         throw new InsufficientFundsException(50 - amountOfMoney);
4     }
5     // ....
6 }
```

```
1 public class InsufficientFundsException extends Exception {
2     public InsufficientFundsException(int missingMoney) {
3         super("Insufficient funds! Needed " + missingMoney + "
   ↳ more money.");
4     }
5 }
```



# Exceptions

## Catch-Blöcke



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↪ ClassCastException {  
2     // ....  
3 }
```

```
1 try {  
2     test();  
3 }  
4 catch(NullPointerException e) {  
5     e.printStackTrace();  
6 }  
7 catch(ClassCastException e) {  
8     e.printStackTrace();  
9 }
```

# Exceptions

## Catch-Blöcke



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↳ ClassCastException {  
2     // ....  
3 }
```

```
1 try {  
2     test();  
3 }  
4 catch(NullPointerException | ClassCastException e) {  
5     e.printStackTrace();  
6 }
```

# Exceptions

## Catch-Blöcke



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↳ ClassCastException {  
2     // ....  
3 }
```

```
1 try {  
2     test();  
3 }  
4 catch(Exception e) {  
5     e.printStackTrace();  
6 }
```

# Exceptions

## Catch-Blöcke



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↪ ClassCastException {  
2     // ....  
3 }
```

```
1 try {  
2     test();  
3 }  
4 catch(Exception e) {  
5     e.printStackTrace();  
6 }  
7 catch(NullPointerException e) {  
8     System.out.println("Null!"); // Fehler!  
9 }
```

# Exceptions

## Catch-Blöcke



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↪ ClassCastException {  
2     // ....  
3 }
```

```
1 try {  
2     test();  
3 }  
4 catch(NullPointerException e) {  
5     System.out.println("Null!");  
6 }  
7 catch(Exception e) {  
8     e.printStackTrace();  
9 }
```

# Exceptions

## Catch-Blöcke



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws Exception {  
2     throw new NullPointerException("Message");  
3 }
```

```
1 try {  
2     test();  
3 }  
4 catch(NullPointerException | ClassCastException e) {  
5     e.printStackTrace();  
6 }  
7 catch(Exception e) {  
8     System.out.println("You need this :)");  
9 }
```

# Exceptions

## Weiterreichen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- 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

# Exceptions

## Weiterreichen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↳ ClassCastException {  
2     throw new NullPointerException("Message");  
3 }
```

```
1 public void run() throws ClassCastException {  
2     try {  
3         test();  
4     }  
5     catch(NullPointerException e) { }  
6     catch(ClassCastException e) {  
7         throw e;  
8     }  
9 }
```



# Exceptions

## Weiterreichen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public void test() throws NullPointerException,  
   ↳ ClassCastException {  
2     throw new NullPointerException("Message");  
3 }
```

```
1 public void run() throws ClassCastException {  
2     // Kein try und catch für ClassCastException  
3     try {  
4         test();  
5     }  
6     catch(NullPointerException e) {  
7         e.printStackTrace();  
8     }  
9 }
```



- 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
  - ▣ ....

# Exceptions

## Javadoc



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  /**
2   * Buys a cool hat
3   * @param amountOfMoney The amount of money available
4   * @return Amount of money left after transaction
5   * @throws InsufficientFundsException If there is not enough
6   *   ↪ money available
7   */
8  public int buyCoolHat(int amountOfMoney) throws
9   ↪ InsufficientFundsException {
10     if(amountOfMoney < 50) {
11         throw new InsufficientFundsException(50 - amountOfMoney);
12     }
13     return amountOfMoney - 50;
14 }
```

# Exceptions

## Typische Fehler



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public class X {  
2     public static int calculateTheAnswer() throws  
    ↳ IllegalStateException {  
3         if(Y.complexCondition() == false) {  
4             throw new IllegalStateException("Nobody knows why  
                ↳ this failed.");  
5         }  
6         return 42;  
7     }  
8 }
```

# Exceptions

## Typische Fehler



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public static int tryToCall() {  
2     // ....  
3 }
```

# Exceptions

## Typische Fehler



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public static int tryToCall() {
2     try {
3         int returnValue = X.calculateTheAnswer();
4     }
5     catch(IllegalStateException e) {
6         throw new RuntimeException("Look at this: " +
7             ↪ e.getMessage());
8     }
9     return returnValue; // ERROR!
}
```

# Exceptions

## Typische Fehler



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public static int tryToCall() {  
2     int returnValue;  
3     try {  
4         returnValue = X.calculateTheAnswer();  
5     }  
6     catch(IllegalStateException e) {  
7         throw new RuntimeException("Look at this: " +  
8             ↪ e.getMessage());  
9     }  
10    return returnValue;  
}
```

# Exceptions

## Typische Fehler



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public static int tryToCall() {  
2     try {  
3         return X.calculateTheAnswer();  
4     }  
5     catch(IllegalStateException e) {  
6         throw new RuntimeException("Look at this: " +  
           ↪ e.getMessage());  
7     }  
8 }
```





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

```
1 public class A {  
2     public void myMethod() throws ExceptionA {  
3         // ...  
4     }  
5 }
```

# Exceptions

## Vererbung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 public class B extends A {  
2     @Override  
3     public void myMethod() throws ExceptionA {  
4         // Gleicher Exception-Typ erlaubt  
5     }  
6 }
```

```
1 public class C extends A {  
2     @Override  
3     public void myMethod() throws ExceptionB {  
4         // "Präzisierung" erlaubt  
5     }  
6 }
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Switch

Primitiven Wert in String umwandeln

Exceptions

**assert**

JUnit-Tests

Boxing



- Einfache und kurze Möglichkeit um Fehler zu werfen
- Wirft immer eine `AssertionError`, der von `Error` abgeleitet ist
- `Error` erbt weder von `Exception` noch `RuntimeException`



- 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` in den VM-Optionen aktiviert werden

```
1 public static void dummyTest(int b) {  
2     assert b >= 0: "Argument must be positive!";  
3 }
```

```
1 public static int sum(int a, int b) {  
2     assert ((long)a + (long)b) >= Integer.MIN_VALUE &&  
3         ((long)a + (long)b) <= Integer.MAX_VALUE: "Overflow  
    ↪ detected!";  
4     return a + b;  
5 }
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Switch

Primitiven Wert in String umwandeln

Exceptions

`assert`

JUnit-Tests

Einfaches Beispiel

`assertDoesNotThrow`

`assertThrowsExactly`

Boxing



- 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



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 import org.junit.jupiter.api.Test;  
2 import static org.junit.jupiter.api.Assertions.*;  
3  
4 public class StudentTests {  
5     ....  
6 }
```

# JUnit-Tests

## Einfaches Beispiel



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1  @Test
2  public void myTest() {
3      Tree tree = new LemonTree();
4      Apple unexpectedFruit = new Apple(5);
5
6      assertTrue(tree.hasFruit());
7      assertTrue(tree.getFruit() instanceof Lemon);
8      assertNotEquals(unexpectedFruit, tree.getFruit());
9
10     assertEquals(2, tree.getFruitAmount());
11 }
```



```
1  @Test
2  public void myTest() {
3      HatFactory factory = new HatFactory();
4
5
6      int moneyLeft = assertDoesNotThrow(
7          () -> factory.buyCoolHat(52)
8      );
9      assertEquals(moneyLeft, 2, "Incorrect money left!");
10 }
```



```
1  @Test
2  public void myTest() {
3      HatFactory factory = new HatFactory();
4
5
6      Throwable thrownException = assertThrowsExactly(
7          InsufficientFundsException.class,
8          () -> factory.buyCoolHat(49)
9      );
10     String message = thrownException.getMessage();
11     // assertEquals....
12 }
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Switch

Primitiven Wert in String umwandeln

Exceptions

assert

JUnit-Tests

Boxing

- Für jeden primitiven Datentypen gibt es eine Wrapper-Klasse
- Später bei Generics sehr wichtig!

Primitiver Datentyp	Wrapper-Klasse
boolean	<code>java.lang.Boolean</code>
byte	<code>java.lang.Byte</code>
short	<code>java.lang.Short</code>
int	<code>java.lang.Integer</code>
double	<code>java.lang.Double</code>
....	....



- Konvertierung in Wrapper-Klasse: Boxing
- Konvertierung zu primitiven Wert: Unboxing
- Automatisch: Auto -> Auto-Boxing und Auto-Unboxing

```
1 Integer a = ....;
2 int aPrimitive = a; // Auto-Unboxing
3
4 double bPrimitive = ....;
5 Double b = bPrimitive; // Auto-Boxing
```



```
1 Integer a = Integer.valueOf(25); // Boxing
2 int aPrimitive = a.intValue(); // Unboxing
3
4 double bPrimitive = 5;
5 Double b = Double.valueOf(bPrimitive); // Boxing
```



- ACHTUNG!
- Keine Widening oder Narrowing Casts bei Wrapper-Klassen
- Weder explizit noch implizit

```
1 Integer myInteger = 5; // Auto-Boxing
2 Long myLong = myInteger; // ERROR!
3 Long myOtherLong = (Long) myInteger; // ERROR!
4 Byte smallByte = myInteger; // ERROR!
5 Byte smallOtherByte = (Byte) myInteger; // ERROR!
```



---

# Live-Coding!