Functional Exception Handling

Wie kann man in Scala Exceptions behandeln?

mit `Option[T]`, anstatt null zu verwenden:

- Some mit einem Wert
- None für keinen Wert

```
def safeDivide(a: Int, b: Int): Option[Int] =
  if (b == 0) None else Some(a / b)

val result1 = safeDivide(10, 2) // Some(5)
val result2 = safeDivide(10, 0) // None

// Umgang mit Option
result1 match {
  case Some(value) => println(s"Ergebnis: $value")
  case None        => println("Division durch Null!")
}
```

-> verhindert NullPointerExceptions, erzwingt explizite Behandlung durch `match` oder `getOrElse`

mit `Try[T]`, um eine Funktion auszuführen, die eine Exception werfen könnte:

- Success mit einem Wert
- Failure mit einer Exception

```
import scala.util.{Try, Success, Failure}

def parseNumber(s: String): Try[Int] = Try(s.toInt)

val number1 = parseNumber("42")  // Success(42)
val number2 = parseNumber("abc") // Failure(java.lang.NumberFormatException)

// Umgang mit Try
number2 match {
  case Success(value)  => println(s"Parsen erfolgreich: $value")
  case Failure(error)  => println(s"Fehler aufgetreten: ${error.getMessage}")
}

// Alternative: Standardwert setzen
val safeNumber = number2.getOrElse(0) // 0
```

Was ist der Unterschied zwischen By-Value und By-Name Parametern?

By-Value:

- Wert wird vor der Ausführung der Funktion ausgewertet

By-Name:

- Wert wird erst ausgewertet, wenn er innerhalb der Funktion benötigt wird
- wird mit `=>` definiert

```
def byValue(x: Int): Unit = {
  println("byValue wurde aufgerufen")
  println(x)
}

def byName(x: => Int): Unit = {
  println("byName wurde aufgerufen")
  println(x)
}
```

Was ist Currying oder multiple Parameterlisten?

eine Funktion kann mehrere Parameterlisten haben, mit mehreren Klammern definiert werden

```
def add(a: Int)(b: Int): Int = a + b
```

Ermöglicht partielle Anwendung:

```
val addTwo = add(2) _  // Fixiert das erste Argument
println(addTwo(5))      // Gibt 7 aus
```

Bessere Typableitung:

```
def map1[A, B](list: List[A], f: A => B): List[B] = list.map(f) // ohne
Currying
def map2[A, B](list: List[A])(f: A => B): List[B] = list.map(f) // mit Currying


val numbers = List(1, 2, 3)

val doubled1 = map1(numbers, (x: Int) => x * 2) // Typ nötig
val doubled2 = map2(numbers)(_ * 2)             // Typ automatisch erkannt!
```

Implizite Parameterlisten:

```scala
def greet(name: String)(implicit greeting: String): String = s"$greeting,
$name!"

implicit val defaultGreeting: String = "Hallo"
println(greet("Anna"))  // Gibt "Hallo, Anna!" aus
```