Lernzettel

Praktische Übungen und Projektdurchführung: Beispielprojekte, Tests mit ScalaTest, Debugging.

Universität:	Technische Universität Berlin
Kurs/Modul:	Programmieren II für Wirtschaftsinformatiker
Erstellungsdatum:	September 6, 2025

Lernzettel: Praktische Übungen und Projektdurchführung: Beispielprojekte, Tests mit ScalaTest, Debugging

(1) Überblick und Lernzielbezug.
In diesem Abschnitt arbeiten Sie an praktischen Übungen, um Beispielprojekte umzusetzen, Tests mit ScalaTest zu schreiben sowie Debugging-Strategien anzuwenden. Ziel ist es, die Konzepte der funktionalen Programmierung in Scala praktisch zu verwenden und das Vorgehen in Projektdurchführung zu üben.

(2) Projektsetup und -struktur.
Wichtig ist eine klare Projektstruktur und eine reproduzierbare Build-Umgebung.

- Ordnerstruktur:
  - src/main/scala
  - src/test/scala
- Minimaler Build.sbt (Auszug):

```
name := "PraxisScala"
version := "0.1.0"
scalaVersion := "2.13.11"

libraryDependencies ++= Seq(
  "org.scalatest" %% "scalatest" % "3.2.14" % Test
)
```

- Kommandos (im Terminal):

```
sbt test        -- führe alle Tests aus
sbt testOnly *BestellSpec -- gezielten Test ausführen
```

(3) Beispielprojekte (Praxisaufbau).
Zwei Beispielprojekte, die typische Fragestellungen der Wirtschaftsinformatik adressieren.

- Projekt A: Funktionsorientierte Bestellabwicklung
  - Ziel: Unveränderliche Datenstrukturen, klare Funktionen, keine Seiteneffekte.
  - Aufgaben: Modellieren eines einfachen Bestellprozesses (Korb, Artikel, Preisberechnung), Testspezifikation mit ScalaTest.
  - Hinweise: Nutze Map/FlatMap, for-Comprehensions, copy-on-write Muster.
- Projekt B: Daten-Transformation für BI
  - Ziel: Transformation von Rohdaten in aggregierte Kennzahlen mit unveränderlichen Datentypen.
  - Aufgaben: Implementiere Transformationsketten (Map, Filter, Reduce), schreibe Tests zur Korrektheit der Aggregationen.

  - Hinweise: Nutze Seq/Traversable-Operationen, Composed Functions statt Schleifen mit Nebenwirkungen.

(4) Tests mit ScalaTest.
ScalaTest ist das zentrale Werkzeug, um Logik robust zu überprüfen. Wichtige Konzepte:

- Grundaufbau einer Testklasse

```
import org.scalatest.flatspec.AnyFlatSpec
import org.scalatest.matchers.should.Matchers

class BestellungSpec extends AnyFlatSpec with Matchers {
  "addToCart" should "increase item count" in {
    val cart = Cart.empty
    val updated = cart.add("A", 2)
    updated.items("A") should be (2)
  }
}
```

- Testarten
  - Unit-Tests für einzelne Funktionen
  - Property-based Tests (mit ScalaTest-Optionen)
  - Async/Future-Tests (mit whenReady oder Await)
- Typische Hilfsmittel
  - fixture oder BeforeEach/AfterEach für Setup
  - Assertions wie should be, shouldEqual, shouldMatch
  - Grenzen testen: Exceptions, fehlerhafte Eingaben

(5) Debugging-Strategien und -Tools.
Effektives Debugging in Scala-Projekten:

- Strategien
  - Schreibe aussagekräftige Log-Ausgaben statt reiner Console-Prints
  - Nutze den Debugger in der IDE (z.B. IntelliJ) oder Remote-Debugging mit sbt
  - Führe Tests selektiv aus, um Fehler gezielt zu isolieren
  - Verwende Futures-Tools (Future, Promise, Await, whenReady) vorsichtig
- Praktische Tipps
  - Starte mit kleinen Reproduktionsfällen
  - Verwende sbt testOnly, um fehlerhafte Tests zu reduzieren
  - Aktivieren von Warnungen/Fehlermeldungen in der Compile-Phase
- Beispiel für einfaches Logging (ohne zusätzliche Bibliotheken)

```
object Utils {
  private val log = play.api.Logger(this.getClass)
  def info(msg: String): Unit = log.info(msg)
}
```

(6) Projektdurchführung: Ablauf und Best Practices.
Empfohlene Vorgehensweise bei der Umsetzung eines Beispielprojekts:

- Kick-off: Anforderungen klären, Abgrenzungen festlegen
- Umgebung aufsetzen: Build, Abhängigkeiten, Testumgebung
- Prototyping: Kernlogik in Modulen, keine Nebenwirkungen
- Tests schreiben: Unit-Tests zuerst, dann Integrations- bzw. Akzeptanztests
- Debugging-Schleife: Fehler reproduzieren, gezielt debuggen, Fix durchführen
- Review Refactoring: Codequalität sicherstellen, Stilkonventionen beachten
- Lieferung: Dokumentation, Hinweise zu Nutzung der Tests

(7) Kurze Hinweise zum Referenz-Workflow.
- Verwende ScalaTest-Suites als Grundlage für neue Tests.
- Halte Funktionen klein und eindeutig; vermeide Seiteneffekte.
- Schreibe Tests, die reale BI- bzw. Wirtschaftsinformatik-Szenarien abbilden.

(8) Kurzglossar (Auswahl).
- ScalaTest: Framework zum Schreiben von Tests in Scala.
- sbt: Build-Tool für Scala-Projekte.
- Akka: Framework zur Umsetzung von Thead-/Message-Driven-Concurrency.
- Funktionsprinzipien: Unveränderlichkeit, Purie-Funktionen, Composition.

(9) Abschlussbemerkung.
Nutze die gezeigten Strukturen, um praxisnah zu arbeiten: klare Projektdokumentation, nachvollziehbare Tests und gezieltes Debugging.