Praktikum 4

Hinweise:
Um in diesem Praktikumsversuch *Gson* (Aufgabe 1) und *JUnit 5* (Aufgabe 3) in Ihrem Projekt nutzen zu können, müssen diese Libraries als *Dependencies* vermerkt und installiert sein. Hierzu gibt es im Java-Umfeld mehrere Möglichkeiten, u.a. mit Hilfe von *Build-Management-Tools* wie *Maven* und *Gradle* oder alternativ durch eine manuelle Installation der Libraries.
Für diesen Praktikumsversuch werden alle Vorgehensweisen akzeptiert.
Im Iliasordner für das Praktikum 4 befindet sich eine vorgefertigte *pom.xml*-Datei (siehe *Maven*, >= Java 17 erforderlich), die Sie für den Ansatz mit *Maven* nutzen können. Diese enthält bereits Abhängigkeiten zu *Gson* und *JUnit 5*.

Im Ilias befinden sich kurze Lernvideos zu den Themen *Maven* (Aufgabe 1-3) und *Test Coverage* (Aufgabe 3), sowie weiterführende Links zu den Themen *Gson*, *JUnit 5*, *Maven und Gradle*.
Bei Bedarf sollten Sie weitere Quellen selbstständig recherchieren und nutzen.

Aufgabe 1 (Persistierung von Konten):
In dieser Aufgabe sollen alle Konten einer *PrivateBank* (und somit die verknüpften Transaktionen) persistiert werden. Hierfür sollen die unterschiedlichen Typen von Transaktionen zunächst in das *JSON*-Format serialisiert und anschließend im Dateisystem gespeichert werden (analog hierzu das Lesen und Deserialisieren der Daten). Der Dateiname soll dem Namen des Kontos entsprechen (z.B. „Konto Adam.json“, „Konto Eva.json“, …). Der Speicherort (ein spezieller Ordner im Dateisystem, relativer oder absoluter Pfad) der Konten bzw. Transaktionen soll als neues Klassenattribut *directoryName* in der *PrivateBank* angelegt werden und bei der Erzeugung von *PrivateBank*-Objekten konfigurierbar sein.

Für die Serialisierung und Deserialisierung von *JSON*-Dokumenten soll die *Gson*-Library verwendet werden. Damit sowohl *Payment*- als auch *Transfer*-Objekte (genauer: *Incoming*- und *OutgoingTransfer*) verarbeitet werden können, müssen eigene *Custom Serializer* und *Custom Deserializer* implementiert werden. Dafür sollten Sie eine Klasse schreiben, die das Interface *JsonSerializer<Transaction>* und *JsonDeserializer<Transaction>* implementiert. Zusätzlich sollten Sie sich die Klasse *GsonBuilder* genauer anschauen.
In den Ilias-Praktikumsunterlagen zu diesem Praktikum finden Sie ein Beispiel, wie die resultierenden *JSON*-Dokumente aussehen sollen (siehe „PersistenceSample.json“). Achten Sie darauf, dass Ihr Format mit der Vorlage übereinstimmt.
Insbesondere relevant ist das *JSON-Property* „CLASSNAME“, das den konkreten Typ/Klasse der Transaktion repräsentiert, damit Sie beim Vorgang des Deserialisierens wissen, welches Objekt angelegt bzw. erzeugt werden muss.

Damit die Operationen des Lesens und Schreibens von Konten/Transaktionen nicht mehrfach implementiert werden, sollen zwei (private) Methoden in der Klasse *PrivateBank* angelegt und programmiert werden:

1. *void readAccounts() throws IOException*: Diese Methode soll alle vorhandenen Konten vom Dateisystem lesen und im *PrivateBank*-Objekt (genauer: im Klassenattribut *accountsToTransactions*) zur Verfügung stellen.
2. *void writeAccount(String account) throws IOException*: Diese Methode soll das angegebene Konto im Dateisystem persistieren (serialisieren und anschließend speichern).

Überlegen Sie anschließend, an welchen Stellen diese beiden Methoden aufgerufen werden müssen, um zu gewährleisten, dass die Informationen der Konten zur Laufzeit (im *PrivateBank*-Objekt) und die persistierten Konten (in den gespeicherten *JSON*-Dokumenten im Dateisystem) immer identisch sind.
Hierzu wird es notwendig sein, dass Sie an einigen Stellen *IOExceptions* werfen müssen. Sie dürfen hierfür die Methodensignaturen (inkl. des Interfaces *Bank*) diesbezüglich anpassen.

Aufgabe 2 (Dokumentation mit Hilfe von Javadoc):
Erweitern Sie die Dokumentation aus Pratikum 3 mit korrektem *Javadoc* (gilt für Klassendefinitionen, Klassenattribute und Methoden) und verwenden Sie sinnvolle *Javadoc-Tags*, um eine möglichst aussagekräftige Dokumentation des gesamten Programmcodes zu gewährleisten.

Aufgabe 3 (Testen der Funktionalität mit Hilfe von JUnit 5):
In dieser Aufgabe sollen die Klassen *Payment*, *Transfer* und *PrivateBank* (mit der Implementierung aus Aufgabe 1) mit Hilfe von *JUnit 5* getestet werden. Hierfür sollen insgesamt drei *JUnit*-Testklassen implementiert werden:

1. *PaymentTest*: Tests für Konstruktor, Copy-Konstruktor, *calculate()* (*incomingInterest* und *outgoingInterest*), *equals*, *toString*.
2. *TransferTest*: Tests für Konstruktor, Copy-Konstruktor, *calculate()*, *equals*, *toString*. Nur bei dem Test der *calculate*()-Methode sollen zusätzlich *Incoming*- und *OutgoingTransfer* getestet werden.
3. *PrivateBankTest*: Tests für Konstruktor, alle impl. Methoden des Interfaces *Bank*, *equals* und *toString*.

Für alle Testklassen soll eine „init-Methode“ implementiert werden, die mit der *JUnit*-Annotation *@BeforeEach* versehen werden soll (falls erforderlich auch *@AfterEach,* z.B. bei *PrivateBank* für das erzeugte Verzeichnis mit serialisierten *JSON*-Dokumenten), die die Testdaten für jeden *JUnit*-Test zur Verfügung stellen soll. Dies bedeutet, dass in den einzelnen Testmethoden nur zusätzliche Testdaten angelegt werden dürfen, falls diese speziell für den Testfall erforderlich sind.
Verwenden Sie außerdem bei Testmethoden, bei denen es Sinn macht, *parameterized tests* (siehe *@ParameterizedTest* und *@ValueSource*).
Zusätzlich sollten bei Methoden, bei denen Exceptions auftreten können, diese auch getestet werden. Verwenden Sie hierfür *assertThrows* und *assertDoesNotThrows* mit Lambda-Ausdrücken.
Mit Hilfe der sogenannten *Test Coverage* können Sie sich innerhalb Ihrer *IDE* genauer anschauen, welche Stellen genau in Ihrem Programmcode getestet werden. Sie erhalten in der Regel auch eine (prozentuale) Übersicht für alle Klassen in Ihrem Projekt.
Seien Sie in der Lage die *Test Coverage* aufzurufen und schauen Sie sich die Ergebnisse genau an (und verbessern Sie ggf. die Abdeckung durch weitere bzw. angepasste Tests).