Lernziele Lernkontrolle

Die Handlungsziele kommen aus den Dokumenten der einzelnen Themenblöcken.

Hier eine Zusammenfassung

1 Allgemein

- Ich halte unsere Regeln zum Programmieren ein.

2. Rekursion

- Ich kann erklären, was eine Rekursion ist
- Ich kann erklären, wie eine Rekursion vom Prinzip her funktioniert.
  Merkmale wie: 'Methoden-Signatur mit Parameter', 'Abbruchbedingung', 'Aufruf der Rekursion' und Return-Wert.
- Ich kann erklären, was bei einem rekursiven Aufruf geschieht und wie sich das in Bezug auf Performance und Speicherverbraucht auswirkt.
- Ich kann Beispiele für den Einsatz von Rekursionen nennen.

- Ich kann eine einfache Rekursion auf der Basis eines Ablaufdiagrammes implementieren.
- Ich kann eine Rekursion auf der Basis einer Aufgabenbeschreibung implementieren.
- Ich kann eine vorgegebene Rekursion interpretieren und erläutern was passiert und dabei besonders auf die Merkmale 'Methoden-Signatur mit Parameter', 'Abbruchbedingung', 'Aufruf der Rekursion' und Return-Wert eingehen.
- Ich kann eine Rekursion nach dem Prinzip Treppe auf und Treppe ab erläutern.
- Ich kann eine Implementation mit einer Rekursion testen.
- Ich kann eine Schleife in eine Rekursion transformieren (und umgekehrt).
- Ich kann eine doppelte Schleife in eine Rekursion transformieren (und umgekehrt).

3. Backtracking

- Ich kann erklären, was ein Backtracking ist und wofür es verwendet wird.
- Ich kann ein Backtracking in einem Diagramm oder in einer Implementation erkennen und erläutern, wie es funktioniert.
- Ich kann ein Backtracking implementieren.

4. Bäume

- Ich kann erklären, was die Baum Datenstruktur ist.
- Ich kann erklären, was eine Wurzel, ein Ast/Zweig und ein Blatt sind.
- Ich kann erklären, wie ein Baum vom Prinzip her mit der Klasse Tree und der Klasse Node aufgebaut ist.

- Ich kann Anhand von Beispielen (Morse-Baum, Binärer-Suchbaum) erklären, welche Bedeutung Werte im Knoten und Werte der Äste haben.
- Ich kann erklären, was Traversieren bedeutet.
- Ich kann verschiedene Methoden zum Traversieren erkennen, erläutern und nachvollziehen.
  (Tiefensuche, Breitensuche, TraverseInOrder, TraversePreOrder, TraversePostOrder, TraverseLevelOrder)
- Ich kann die unterschiedlichen Aspekte beim Löschen eines Knoten erläutern.
- Ich kann eine bestehende Implementierung von einem Baum nachvollziehen und erläutern.

- Ich kann die Klassen für einen Baum implementieren.
- Ich kann eine rekursive Methode implementieren, die einen Knoten in einen Baum einfügt.
- Ich kann eine rekursive Methode implementieren, die eine bestimmten Knoten in einem Baum findet.
- Ich kann eine rekursive Methode implementieren, die einen Baum traversiert.
- Ich kann eine rekursive Methode implementieren, die eine Wert in einem Baum sucht.
- Ich kann eine Methode zum Löschen eines Knoten, eines Blattes implementieren.
- Ich kann verschiedene Methoden zum Traversieren eines Baumes implementieren.
- Ich kann eine Min-Max Algorithmus mit Hilfe eines Baumes implementieren.

5. Funktionales Programmieren mit Java angewandt beim Vergleichen und Sortieren

- Ich kann eine Imperative Programmierung in eine funktionale Programmierung überführen.
- Ich kann das Interface *Comparator* erklären und anwenden.
- Ich kann ein Funktional Interface mit Hilfe einer *abgeleiteten Klasse, anonyme Klassen* und/oder mit *Lambdas* implementieren.
  zBsp anhand von *Comparator*
- Ich kann eine *Comparator chain* erklären und anwenden.
- Ich kann die Class *Comparable* erklären und wenden.

- Ich kann eine *Collection von Objekten* nach verschiedenen Aspekten sortieren.
- Ich kann Objekte nach Ihrer *Natural-Order* und *reverse* sortieren.
- Ich kann Klassen-Attribute von Comparatoren für die Sortierung deklarieren, initialisieren und verwenden.
- Ich kann erklären, wo und wie beim Sortieren das funktionale Programmieren zur Ausprägung kommt.
- Ich kann ein Datenbeispiel mit Testdaten und verschiedenen Aspekte des funktionalen Sortierens erklären und umsetzen.
- Ich kann Datenklassen bei Java optimieren, um Boilerplate Code zu vermeiden. (Lombok, Record ..)

Erweiterte Ziele (höhere Kompetenz)

- Ich kann 'tiefer', detaillierter erklären, wie Java *Natural-Order* und *reverse* umsetzt. (Aus der Untersuchung am konkreten Projekt)
- Ich kann 'tiefer' erklären, wie Java das Sortieren für Collections umsetzt. (Aus der Untersuchung am konkreten Projekt)

6. Funktionales Programmieren mit Java, Functional Interfaces

- Ich kann die Functional Interfaces und deren Unterschiede erklären: Anzahl Argumente/Parameter, Returnwert, Bedeutung von <T>, <T,R>, <T, U, R> … Java Interfaces wie zBsp Consumer, Function, Supplier, Predicate, BinaryOperator, Comparator …
- Ich kann die Functional Interfaces spezifisch für deren gedachte Anwendung auswählen.
- Ich kann erklären, wieso Functional Interfaces die Implementierung von verschiedenen Methoden vorsehen. zBsp accept, apply, get, test …
- Ich kann diese Methoden implementieren. (zBsp mit Lambda Expression)
- Ich kann eigene Functional Interfaces entwerfen und anwenden.

ZBsp aus 17.1_JavaFunctionalInterfaces.pdf, 17.2_AufgabenZuJavaFunctionalInterfaces.docx

7. Funktionales Programmieren mit Java Streams

- Ich kann Streams auf Collection von Objekten anwenden.
- Ich kann die Methoden von Streams erklären und anwenden. zBsp: Filter, Map, Collect, Reduce, Optional …
- Ich kann eine bestehende Umsetzung mit Streams interpretieren und erklären. Was passiert bei jedem Schritt, was ist der Input, was wird gemacht, was ist der Output.
- Ich kann eine Menge von Daten auswählen, ein Ziel für eine Auswertung formulieren und dieses Ziel mit Hilfe von Streams umsetzen.
- Ich kann Daten mit Hilfe von Java grafisch darstellen.

Zur Lernkontrolle:

- Es geht dabei auch um ein speditives, zielgerichtetes Anwenden innerhalb eines definierten Zeitrahmens.
- Hilfsmittel:
  Je nach Prüfung: -- keine -- oder -- Übungen, persönliche Notizen, Internetseiten wie Sie im Unterricht verwendet wurden --.
- Sie dürfen keine fremden Lösungen oder auch nur Teile davon verwenden.
  Und Sie dürfen Ihre Lösung oder Teile davon niemandem weitergeben.