Prof. Dr. Christian Kohls
Alexander Dobrynin, M.Sc.

April 11, 2022
SoSe 2022

Algorithmen und Programmierung 2
Praktikum 0 - Warm Up

Dieses Übungsblatt ist für den Übergang von AP1 zu AP2 gedacht. Es kommen zwar keine neuen Themen hinzu, allerdings müssen Sie alle Aufgaben mit der Programmiersprache Kotlin in IntelliJ lösen.

Für die Aufgaben sind die Videos "2. Entwicklungsumgebung einrichten" bis "11. ArrayList verwenden" und "17. Listen in Kotlin" aus der Playlist auf YouTube bzw. die Kapitel 8 bis 12.4 aus dem Buch Programmieren lernen mit Kotlin relevant.

Hier noch ein paar nützliche Links:

- Screencasts für die Veranstaltung: OOP mit Kotlin von Christian Kohls.
- Kotlin anhand von Beispielen kennenlernen: Learn Kotlin by Example.
- Offizielle Dokumentation von Kotlin.
- Kotlin online ausprobieren: Try Kotlin online.

Contents

1 IntelliJ einrichten und das erste Programm kompilieren

Richten Sie zunächst IntelliJ und Kotlin ein. Orientieren Sie sich dabei an dem offiziellen Getting Started von JetBrains (den Machern von IntelliJ und Kotlin). Sorgen Sie dafür, dass Sie die im Tutorial gezeigte `main` Funktion ohne Compilerfehler o.Ä. ausführen können.

2 Die Klasse "Laufstrecke" definieren

In dieser Aufgabe soll eine Laufstrecke als Klasse festgelegt werden.

2.1 Basisstruktur

a) Definieren Sie eine Klasse namens `RunningTrack` mit den folgenden Eigenschaften:

- Name der Strecke. z.B. `"Im Wald"` oder `"Am See"`.
- Länge der Strecke. z.B. `2000` oder `1500`.
- Sehenswürdigkeiten entlang der Strecke. z.B. `"Baumhaus"`, `"Tiergehege"` oder `"Brücke"`, `"Tunnel"`, `"Fluss"`. Verwenden Sie hierfür ein Array oder eine Liste.

2.2 toString überschreiben

Überschreiben Sie für diese Klasse die `toString` Methode, sodass alle Eigenschaften eingebaut werden. Beispiel:

```
fun main() {
  val track = RunningTrack(
    "Im Wald",
    2000,
    listOf("Baumhaus", "Tiergehege")
  )
  println(track.toString())
}
```

Die Ausgabe würde dann so aussehen:

```
Die Strecke Im Wald (2000 m) verläuft entlang Baumhaus, Tiergehege
```

2.3 Mehrere Laufstrecken erzeugen

Ändern Sie Ihre `main` Funktion aus Aufgabe 1 so ab, dass Sie eine Liste mit 5 Laufstrecken erzeugen und diese der Variable `tracks` zuweisen (siehe Code). In Aufgabe 3 werden Sie gleich verschiedene Funktionen implementieren. Nutzen Sie die `main` Funktion, um Ihre Implementierungen zu testen.

```
fun main() {
  val tracks = TODO()

  // Nutzen Sie hier alle weiteren Funktionen und geben Sie das
     Ergebnis auf der Konsole aus:
}
```

3 Funktionen implementieren

In dieser Aufgabe implementieren Sie diverse Funktionen, die mit einer Liste von Laufstrecken arbeiten. Testen Sie die Funktionen in Ihrer `main` Funktion, nachdem Sie diese implementiert haben.
Ersetzen Sie jeweils den Aufruf von `TODO()` mit Ihrer eigenen Implementierung.

3.1 Funktionen, die etwas auf der Konsole ausgeben

```
fun printAll(tracks: List<RunningTrack>) {
  // Gibt alle Strecken auf der Konsole aus.
  TODO()
}

fun printMinMax(tracks: List<RunningTrack>, min: Int, max: Int) {
  // Gibt alle Strecken aus, die eine Länge zwischen min und max haben.
  TODO()
}

fun printUntilDestinationLength(tracks: List<RunningTrack>,
   destinationLength: Int) {
  // Gibt solange jede Strecke aus, bis die Ziellänge
     (destinationLength) erreicht ist.
  // Nachdem alle Strecken ausgegeben worden sind, wird die Anzahl der
     gelaufenen Meter ebenfalls ausgegeben.
  TODO()
}

fun printUntilTrack(tracks: List<RunningTrack>, track: RunningTrack) {
  // Gibt alle Strecken bis zur übergebenen Strecke (track) auf der
     Konsole aus.
  TODO()
}

fun printShortestTrack(tracks: List<RunningTrack>) {
  // Gibt nur die Strecke mit der kürzesten Länge auf der Konsole aus
  TODO()
}

fun printLongestTrack(tracks: List<RunningTrack>) {
  // Gibt nur die Strecke mit der längsten Länge auf der Konsole aus
  TODO()
}
```

3.2 Funktionen, die einen Wert berechnen und zurückgeben

```
fun sumLength(tracks: List<RunningTrack>): Int {
  // Gibt die Summe der Längen aller Strecken zurück
  TODO()
}

fun avgLength(tracks: List<RunningTrack>): Int {
  // Gibt die durchschnittliche Länge aller Strecken zurück
  TODO()
}

fun hasPlaceOfInterest(tracks: List<RunningTrack>, placeOfInterest:
   String): Boolean {
  // Gibt zurückt, ob bei einer der Strecken die gesuchte
     Sehenswürdigkeit (placeOfInterest) dabei ist
  TODO()
}
```

3.3 Funktionen, die eine Laufstrecke zurückgeben

```
fun longestTrack(tracks: List<RunningTrack>): RunningTrack {
  // Gibt die Strecke mit der längsten Länge zurück. Überlegen Sie
     sich, was zurückgegeben werden soll, wenn die Liste leer ist.
     Passen Sie ggf. den Rückgabetyp an.
  TODO()
}

fun shortestTrack(tracks: List<RunningTrack>): RunningTrack {
  // Gibt die Strecke mit der kürzesten Länge zurück. Überlegen Sie
     sich, was zurückgegeben werden soll, wenn die Liste leer ist.
     Passen Sie ggf. den Rückgabetyp an.
  TODO()
}

fun tracksWithPlaceOfInterest(tracks: List<RunningTrack>,
   placeOfInterest: String): List<RunningTrack> {
  // Gibt alle Strecken zurück, die an der Sehenswürdigkeit
     (placeOfInterest) vorbei führen.
  TODO()
}

fun longestTrackOf(track1: RunningTrack, track2: RunningTrack):
   RunningTrack {
```

```
  // Gibt die längere Strecke zurück.
  TODO()
}
```