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

April 19, 2022
SoSe 2022

Algorithmen und Programmierung 2
Praktikum 1 - Klassen

In diesem Praktikum implementieren Sie einen Parkautomaten, der Parkscheine ausstellt und verwaltet. Zudem können Parkscheine abgestempelt und Einnahmen angezeigt werden.

Für die Aufgaben sind die folgenden Videos aus der Playlist auf YouTube relevant:

- "9. Einfache Klassen" bis "20. Code verbessern mit Expression Body" (mit Ausnahme von Video 19). Hinweis: Beim Video "11. ArrayList verwenden" ersetzen Sie gedanklich die `ArrayList` mit `MutableList`.
- "34. Nullfähige Typen (Nullability)".
- "42. Exceptions" und "43. Exceptions als Ausdruck".

Bzw. folgende Kapitel aus dem Buch Programmieren lernen mit Kotlin:

- Kapitel "14 Klassen" bis inklusive Kapitel "14.5.2 Enum-Klassen".
- Kapitel "19 Nullfähigkeit".
- Kapitel "20 Exceptions".

Contents

1 Die Klasse "Uhrzeit"

In dieser Aufgabe soll eine Uhrzeit als Klasse festgelegt werden.

1.1 Basistruktur

Definieren Sie eine Klasse namens `Time`. Die Klasse soll die Stunden (`hour`) und Minuten (`minute`) als Eigenschaften speichern. Die Eigenschaften sollen vom Typ `Int` und unveränderlich sein.
Beispiel: `Time(12, 0)` für 12:00 Uhr oder `Time(8, 30)` für 08:30 Uhr.

1.2 Konstruktor

Jetzt ist es allerdings möglich, ungültige Uhrzeiten, wie z.B. `Time(26, 120)` zu erzeugen. Um dieses Problem zu beheben, fügen Sie der Klasse `Time` einen Konstruktor hinzu. Der Konstruktor überprüft, ob `hour` und `minute` gültige Werte haben. Wenn nicht, wird eine Exception mit einer aussagekräftigen Fehlermeldung geworfen.

Testen Sie Ihren Code, indem Sie verschiedene Uhrzeiten in einer `main` Funktion anlegen und diese ausführen. Die letzen beiden Uhrzeiten sollten z.B. eine Exception werfen:

```
fun main() {
  val time1 = Time(12, 0) // valide Uhrzeit
  val time2 = Time(12, 30) // valide Uhrzeit
  val time3 = Time(26, 120) // invalide Uhrzeit. Programm stürzt mit
     einer Fehlermeldung (Exception) ab.
  val time4 = Time(-5, -10) // invalide Uhrzeit. Programm stürzt mit
     einer Fehlermeldung (Exception) ab.
}
```

Wenn die Konstruktion der invaliden Uhrzeiten das Programm zuverlässig zum Absturz bringen, können Sie den Code auskommentieren.

2 Die Klasse "Parkschein"

In dieser Aufgabe soll ein Parkschein als Klasse mit entsprechenden Methoden festgelegt werden.

2.1 Basistruktur

Definieren Sie eine Klasse namens `ParkTicket`. Die Klasse soll die Einfahrtszeit (`entryTime`) als unveränderliche Eigenschaft speichern. Die Einfahrtszeit ist vom Typ `Time` aus Aufgabe 1.
Zudem besitzt ein Parkschein auch eine Ausfahrtszeit (`exitTime`). Da die Ausfahrtszeit erst später gesetzt wird, muss diese veränderlich und nullfähig sein. Setzen Sie die Ausfahrtszeit zunächst auf `null`.
Die Klasse `ParkTicket` sollte wie folgt erzeugt werden können:

```
fun main() {
  val entryTime = Time(12, 0)
  val ticket = ParkTicket(entryTime) // exitTime wird zunächst
     standardmäßig auf null gesetzt
}
```

2.2 Parkscheine abstempeln

In dieser Teilaufgabe soll eine Methode implementiert werden, die einen Parkschein abstempelt. Implementieren Sie hierfür die Methode `checkout(exitTime: Time)`. Diese Methode setzt die Eigenschaft `exitTime` auf den übergebenen Parameter `exitTime`, allerdings nur, wenn die übergebene Ausfahrtszeit (`exitTime`) nach der Einfahrtszeit (`entryTime`) liegt[1]. Ansonsten wird eine Exception mit einer entsprechenden Fehlermeldung, wie z.B. `"Ausfahrtszeit $exitTime liegt vor der Einfahrtszeit $entryTime"`, geworfen. Ein Parkschein kann also nur mit einer *gültigen Ausfahrtszeit abgestempelt* werden.
Testen Sie die `checkout` Methode, indem Sie eine gültige und eine ungültige Ausfahrtszeit übergeben:

```
fun main() {
  val entryTime = Time(12, 0)
  val ticket = ParkTicket(entryTime)
  ticket.checkout(Time(12, 30)) // funktioniert. Parkdauer beträgt 30
     min
  ticket.checkout(Time(11, 0)) // funktioniert nicht. Parkdauer kann
     nicht negativ sein. Programm stürzt mit einer Fehlermeldung
     (Exception) ab.
}
```

[1] Es gibt mehrere Möglichkeiten, zwei Uhrzeiten miteinander zu vergleichen, z.B. Umrechnung in Minuten.

2.3 Parkdauer und angefangene Parkstunden als Methoden

Implementieren Sie jetzt die beiden Methoden `parkingDuration` und `hoursStarted`. Bei beiden Methoden gilt: Wenn die Ausfahrtszeit (`exitTime`) noch nicht gesetzt (`null`) ist, werfen Sie eine Exception mit der Fehlermeldung: `"ParkTicket muss vorher korrekt abgestempelt werden"`. Die Methoden sollen folgendermaßen implementiert werden:

- `parkingDuration(): Int` => Gibt die Parkzeit in Minuten zurück.
- `hoursStarted(): Int` => Gibt die Anzahl der angefangenen Parkstunden zurück. Bei der Einfahrt ins Parkhaus beginnt sofort die erste Stunde. Nach 60 Minuten beginnt die zweite Stunde usw.

Testen Sie die Methoden mit unterschiedlichen Parkscheinen:

```
fun main() {
  val entryTime = Time(12, 0)
  val ticket = ParkTicket(entryTime)
  ticket.checkout(Time(12, 30)) // funktioniert, weil 12:00 < 12:30
  println(ticket.parkingDuration()) // Gibt 30 aus
  println(ticket.hoursStarted()) // Gibt 1 aus

  val ticket2 = ParkTicket(Time(12, 30))
  ticket2.checkout(Time(13, 40)) // funktioniert, weil 12:30 < 13:40
  println(ticket2.parkingDuration()) // Gibt 70 aus
  println(ticket2.hoursStarted()) // Gibt 2 aus
}
```

Kommentieren Sie den Aufruf der `checkout` Methode auf `ticket` oder `ticket1` aus. Das Programm sollte dann abstürzen, weil `parkingDuration` und `hoursStarted` nur mit einer gültigen Ausfahrtszeit funktionieren.

2.4 Berechnete Eigenschaften

Kommentieren Sie die beiden Methoden aus Teilaufgabe 2.3 aus und implementieren Sie diese stattdessen als *berechnete Eigenschaften* (Getter Eigenschaft). Die Logik ist die gleiche.
Sind `parkingDuration` und `hoursStarted` eher berechnete Eigenschaften oder Methoden? Wie würden Sie das begründen?

3 Das Enum "Parktarif"

Der Parktarif legt den Preis pro Stunde fest.

3.1 Basistruktur

Implementieren Sie ein Enum namens `Tariff`, welches die drei Tarife `STANDARD`, `EVENT` und `WEEKEND` kennt.

3.2 Tarifpreis als Methode

Fügen Sie dem Enum eine Methode namens `price` hinzu, welche den jeweiligen Tarifpreis als `Double` zurückgibt:

- Default: 1,99 Euro
- Event: 1,49 Euro
- Wochenende: 2,99 Euro

Enums lassen sich mit dem Schlüsselwort when dekonstruieren. Verwenden Sie das Schlüsselwort, um die Methode `price` zu implementieren.

Hier ein Beispiel zur Verwendung:

```
fun main() {
  val default = Tariff.DEFAULT
  println(default.price()) // Gibt 1.99 aus

  val weekend = Tariff.WEEKEND
  println(weekend.price()) // Gibt 2.99 aus
}
```

4 Die Klasse "Parkautomat"

In dieser Aufgabe soll ein Parkautomat als Klasse festgelegt werden. Ein Parkautomat verwaltet eine Liste von Parkscheinen, teilt diese aus und bietet noch ein paar weitere Funktionalitäten an.

4.1 Basistruktur

Definieren Sie eine Klasse namens `TicketMachine`. Diese Klasse hat die folgenden Eigenschaften:

- Liste mit Parkscheinen. Die Liste muss veränderlich sein, da Parkscheine hinzugefügt werden.
- Aktueller Parktarif. Auch der Tarif kann nachträglich verändert werden.

4.2 Parkscheine ausstellen

Der Parkautomat muss Parkscheine ausstellen können. Implementieren Sie die Methode `generate`, die eine `entryTime` vom Typ `Time` akzeptiert und ein `ParkTicket` zurückgibt. Die Methode erzeugt einen Parkschein mit der übergebenen Einfahrtszeit und fügt diesen der Liste von Parkscheinen hinzu. Am Ende wird der erstellte Parkschein zurückgegeben.

4.3 Methoden auf mehreren Parkscheinen

Jetzt implementieren Sie ein paar Methoden, die verschiedene Operationen auf den Parkscheinen durchführen. Diese Operationen können Sie aber **nur mit abgestempelten bzw. bezahlten Parkscheinen** ausführen.
Hierfür können Sie sich eine Hilfsfunktion namens `validTickets` schreiben, die eine Liste von abgestempelten Parkscheinen zurückgibt. Ein Parkschein ist abgestempelt, wenn die `exitTime` gesetzt (ungleich `null`) ist. Diese Liste verwenden Sie für die folgenden Methoden:

- `shortestParkingDuration(): Int` => Durchläuft alle abgestempelten Parkscheine (siehe oben) und findet den Parkschein mit der kürzesten Parkdauer. Die Parkdauer dieses Parkscheins wird zurückgegeben.
- `averageParkingDuration(): Int` => Berechnet die durchschnittliche Parkdauer aller abgestempelten Parkscheine und gibt diese zurück.
- `revenues(): Double` => Liefert die Einnahmen, die mit allen gültigen Parkscheinen erzielt wurden. Hierfür werden die angefangenden Stunden (`hoursStarted`) mit dem Tarif des Parkautomats (siehe Aufgabe 3) verrechnet.

4.4 Einen Parkautomaten simulieren

In Ihrer `main` Funktion sollen Sie nun alle Klassen und Methoden verwenden.

Sie können den folgenden Coden zum Ausprobieren verwenden. Beachten Sie, dass `ticket4` nicht abgestempelt wurde (`checkout`) und daher nicht in Methoden `shortestParkingDuration`, `averageParkingDuration` und `revenues` berücksichtigt wird (siehe Aufgabe 4.3):

```
fun main() {
  val machine = TicketMachine(Tariff.DEFAULT)
  val ticket1 = machine.generate(Time(12, 0))
  val ticket2 = machine.generate(Time(12, 30))
  val ticket3 = machine.generate(Time(13, 30))
  val ticket4 = machine.generate(Time(13, 30))

  ticket1.checkout(Time(12, 30)) // 30 min (1h)
  ticket2.checkout(Time(13, 30)) // 60 min (1h)
  ticket3.checkout(Time(14, 50)) // 80 min (2h)
  // ticket4 wird nicht abgestempelt und wird daher für alle folgenden
     Methodenaufrufe nicht berücksichtigt

  println(machine.shortestParkingDuration()) // Gibt 30 aus
  println(machine.averageParkingDuration()) // Gibt 56 aus ((30 + 60 +
     80) / 3)
  println(machine.revenues()) // Gibt 7.96 aus ((1 + 1 + 2) *
     machine.tariff.price())
}
```

Wenn Ihre Ausgaben vom obigen Code abweichen, stimmt etwas nicht mit Ihrer Implementierung[2]. Überprüfen und Korrigieren Sie Ihren Code wenn nötig.

[2]Natürlich kann auch meine Implementierung falsch sein. Hierfür gerne eine Mail an ap2-praktikum@gm.fh-koeln.de.

5 Sichtbarkeitsmodifikatoren

In dieser Aufgabe verbessern Sie das Programm, indem Sie ein paar Sichtbarkeitsmodifikatoren hinzufügen.

Aktuell ist es möglich, die `exitTime` eines Parkscheins an der `checkout` Methode "vorbei" zu setzen:

```
fun main() {
  val machine = TicketMachine(Tariff.DEFAULT)
  val ticket5 = machine.generate(Time(12, 0))
  ticket1.exitTime = Time(10, 0) // exitTime direkt setzen, ohne
     checkout aufzurufen
}
```

Dabei wird nicht überprüft, ob die Ausfahrtszeit nach der Einfahrtszeit liegt. Der Parkschein `ticket5` hat invalide Daten, was im weiteren Verlauf des Programms (z.B. in den Methoden `shortestParkingDuration`, `averageParkingDuration` und `revenues`) falsche Daten erzeugt.
Dieses Problem können Sie mit Sichtbarkeitsmodifikatoren verhindern. Schauen Sie sich das Schlüsselwort `private` an und sichern Sie die Variable `exitTime` damit ab.

Überprüfen Sie in Hinblick auf Sichtbarkeitsmodifikatoren auch die restlichen Variablen und Methoden.