Programmieren trainieren

Mit über 150 Workouts in Java und Python

2 Einführung in die Programmierung

2.1 Warm-up

Dein Training beginnt in diesem Kapitel mit ersten einfachen Programmen. Dazu musst du wissen, wie der grundlegende Aufbau eines Programms sowie der Aufbau der Anweisungen in einer bestimmten Programmiersprache sind. Letzteres gehört zur sogenannten **Syntax** einer Programmiersprache. So wie z. B. die Syntax einer natürlichen Sprache Prinzipien und Regeln des Wort- und Satzbaus festlegt, so legt die Syntax einer Programmiersprache das Vokabular und den Aufbau von Anweisungen fest.

Für die allerersten Programme, die du entwickeln sollst, genügt zunächst die allereinfachste Struktur überhaupt. Hierbei werden Programme als eine lineare Abfolge von Anweisungen angegeben. Anweisungen verfügen immer über einen Namen und eine Liste von Parametern, die die Anweisung verarbeiten soll. Um den Anweisungsnamen von der Parameterliste unterscheiden zu können, werden die Parameter häufig eingeklammert und dem Anweisungsnamen nachgestellt.

```
nameAnweisung(parameter);
```

Verfügt die Parameterliste über mehrere Einträge, so werden diese mit Komma (`,`) voneinander getrennt.

```
nameAnweisung(parameter1, parameter2);
```

Parameterlose Anweisungen sind durch ein leeres Klammernpaar gekennzeichnet.

```
nameAnweisung();
```

Um mehrere Anweisungen voneinander unterscheiden zu können, wird dafür ein Trennzeichen in der Syntax einer Programmiersprache festgelegt. In Java ist das das Semikolon (`;`). Das folgende Beispiel zeigt ein abstraktes Programm, das sich aus sieben Anweisungen zusammensetzt, die in der angegebenen Reihenfolge ausgeführt werden. Die lineare Programmabfolge führt die programmierten Anweisungen zeilenweise von links nach rechts beginnend mit der obersten Zeile aus.

```
Anweisung1(); Anweisung2(); Anweisung3(); Anweisung4(); Anweisung5();
Anweisung6(); Anweisung7();
```

Durch diese Syntaxregel können die einzelnen Anweisungen separiert werden, unabhängig davon, wie du diese in die Quelltextdatei schreibst. Zur besseren Lesbarkeit empfehlen wir dir aber, dich auf eine Anweisung pro Zeile zu beschränken und die Anweisungen untereinander zu schreiben.

```
Anweisung1();
Anweisung2();
Anweisung3();
Anweisung4();
Anweisung5();
Anweisung6();
Anweisung7();
```

Die Programmiersprache Python legt in ihrer Syntax als Trennzeichen von Anweisungen den Zeilenumbruch fest. Ein Zeilenumbruch kann je nach Betriebssystem aus einem oder zwei Zeichen bestehen (`'\n'`, `'\r'` oder `'\r\n'`).

In der Programmierliteratur hat sich das `"Hello World!"`-Programm als einführendes Beispiel zur Darstellung der grundlegenden Syntax eines einfachen Programms in einer bestimmten Programmiersprache etabliert. Das Hello-World-Programm gibt in der Konsole einen einfachen Text aus, nämlich `Hello World!`. Wir wollen es zur Konkretisierung der einführenden Erläuterungen verwenden.

Java:

```
print("Hallo Welt!");
```

Python:

```
print("Hallo Welt!")
```

Die `print()`-Anweisung bekommt einen Parameter übergeben. Dieser enthält den Text, den die Anweisung in der Konsole ausgeben soll. Um den Text eingrenzen zu können, wird dieser von doppelten Anführungszeichen (`"`) eingerahmt.

Die Aufgaben dieses Kapitels drehen sich um derartige Programme. Deine Aufgabe wird es sein, die zur Lösung der Aufgabenstellung benötigten Anweisungen zu identifizieren und diese dann in einer geeigneten Abfolge zu platzieren. Welche Anweisungen eine Programmiersprache im Standardumfang bereitstellt, sind in der Referenzdokumentation aufgeführt. Die Referenz der von Processing bereitgestellten Anweisungen kann im Internet eingesehen werden:

- https://processing.org/reference/ (Java)
- http://py.processing.org/reference/ (Python)

Referenzen sind sehr umfangreich. Dies gilt auch für die von Processing. Es kann daher etwas dauern, bist du dich darin zurechtfindest. Für die in diesem Kapitel bereitgestellten Trainingsaufgaben sind insbesondere Funktionen zur Ausgabe von Texten in der Konsole und Funktionen zur Ausgabe elementarer geometrischer Formen im grafischen Ausgabefenster wichtig. Um dir das Auffinden dieser Anweisungen zu erleichtern, führen wir dir in der nachfolgenden Auflistung die relevanten auf.

- Konsolenausgabe
  - https://processing.org/reference/print_.html (Java)
  - http://py.processing.org/reference/print.html (Python)
- Linie
  - https://processing.org/reference/line_.html (Java)
  - http://py.processing.org/reference/line.html (Python)

- Dreieck
  - https://processing.org/reference/triangle_.html (Java)
  - http://py.processing.org/reference/triangle.html (Python)
- Rechteck
  - https://processing.org/reference/rect_.html (Java)
  - http://py.processing.org/reference/rect.html (Python)
- Viereck
  - https://processing.org/reference/quad_.html (Java)
  - http://py.processing.org/reference/quad.html (Python)
- Ellipse
  - https://processing.org/reference/ellipse_.html (Java)
  - http://py.processing.org/reference/ellipse.html (Python)
- Kreisausschnitt
  - https://processing.org/reference/arc_.html (Java)
  - http://py.processing.org/reference/arc.html (Python)

Um sich mit der Funktionsweise der Anweisungen vertraut zu machen, empfehlen wir dir, die Beschreibung in der Referenz aufmerksam zu lesen. Dies ist eine wichtige Grundfertigkeit, die zum Programmieren dazugehört.

Verwenden werden wir in diesem Buch die Entwicklungsumgebung Processing. Hiermit können wir Programme sowohl in Java als auch in Python schreiben. Processing bietet nicht nur den Vorteil der einfachen Installation auf nahezu allen Betriebssystemen. Wir können damit auch sehr einfach (grafische) Programme auf Basis von Anweisungen schreiben. Aber auch höherwertige Konzepte, wie wir sie in den späteren Kapiteln umsetzen werden, sind in Processing möglich; perfekte Voraussetzungen also zum Trainieren deiner Programmiertechniken mit diesem Buch.

Alle Installationsschritte von Processing findest du in Abschnitt A.1. Wie du an die digitalen Quelltexte unserer Lösungsvorschläge zu einzelnen Aufgaben kommst und wie du sie in Processing öffnest, steht im Anhang C.1.1 für Java und im Anhang D.1.1 für Python.

Dateien mit Quelltext können wir in Processing mit Klick auf *Datei → Öffnen …* laden. In Bild 2.1 haben wir zum Beispiel eine solche Datei geöffnet. Dort können wir gut die grafische Bedienoberfläche von Processing erkennen:

- Mit dem Start- und Stopp-Button *(1)* kannst du deinen Java- bzw. Python-Code ausführen.
- Um vom Java- auf den Python-Modus zu wechseln, kannst du den Modus-Auswahlreiter *(2)* verwenden. Wie du den Python-Modus in Processing installierst, steht in Anhang A.5. Links neben diesem Button ist der integrierte Debugger, den du zur Analyse von Java-Code verwenden kannst. Mehr dazu findest du in Anhang C.1.4.
- In der Mitte der Bedienoberfläche *(3)* steht der eigentliche Quelltext. In diesen Bereich kannst du deinen Java- bzw. Python-Code hineinschreiben.
- Entsprechende Ausgaben in der Konsole findest du im darunterliegenden Bereich *(4)*. Hier werden auch auftretende Fehler im Code angezeigt, sofern es welche gibt.

Nach der Einrichtung von Processing und dem Lesen der Einführung solltest du für dieses Kapitel ausgerüstet sein. In dem Sinne: Viel Spaß bei den ersten Aufgaben, und ran an die Workouts!

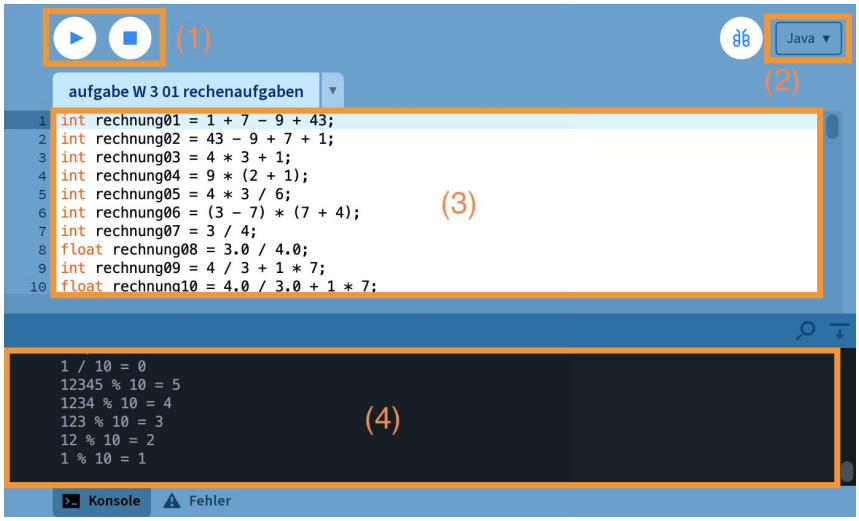
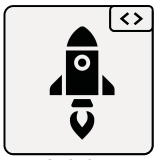


Bild 2.1 So sieht die grafische Bedienoberfläche von Processing aus.

2.2 Workout

W.2.1 Three-Two-One – Mein erstes Programm



Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau von Programmanweisungen
- Ausgabe in der Konsole

Beschreibung

Wir wollen ein erstes Programm schreiben. Der Klassiker hierfür ist die Ausgabe eines Texts – meist der Text `Hello World` – in der Konsole. Dazu braucht es in der Regel nur eine einzige Anweisung. An dieser kannst du aber bereits den Aufbau von Anweisungen und einfachen Programmen nachvollziehen und trainieren. Los geht's!

Aufgabenstellung

Schreibe ein Programm, das den Text `Three-Two-One - Takeoff!` in der Konsole ausgibt. Wenn dein Programm funktioniert, solltest du den angegebenen Text in der Konsole lesen können, so wie nachfolgend exemplarisch zu sehen ist:

```
Three-Two-One - Takeoff!
```

Wenn das geklappt hat, dann mach doch einfach weiter und modifiziere dein erstes Programm nach deinen Wünschen. Ändere z. B. den Text oder füge weitere Anweisungen zur Textausgabe hinzu. Reflektiere dabei, wie dein Programm auf die Änderungen reagiert. Wenn du das Resultat hast kommen sehen, und es ist nichts Unerwartetes bei der Ausführung deines Programms passiert, hast du es im Griff und verstanden, wie Anweisungen und einfache Programme aufgebaut sind.

Testfälle

Zum Testen deines Programms brauchst du in diesem Fall noch keine Testdaten. Starte dein Programm und prüfe, ob die geforderte Ausgabe in der Konsole ausgegeben wird.

(Algorithmische) Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Gib nicht auf. Du solltest es so lange probieren, bis es klappt. Das nennt man *Trial and Error* (Versuch und Irrtum). Versuch es weiter! Vermutlich bist du schon nah dran an der Lösung, denn der Fehler liegt sehr häufig im Detail.
- Wir benötigen eine passende Anweisung, die uns die Programmiersprache zur Ausgabe von Daten in der Konsole bereitstellt. Wie lautet diese?
- Anweisungen folgen einem festgelegten Aufbau. Hier schleichen sich schon mal Tippfehler ein. Was sagen denn die Fehlermeldungen, wenn du versuchst, dein Programm zu starten?

W.2.2 Weihnachtsbaum

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Struktur eines einfachen Programms
- Aufbau und Abfolge von Programmanweisungen
- Ausgabe in der Konsole

Beschreibung

Wir wollen jetzt ein erstes Muster in die Konsole schreiben. Dafür werden wir bestimmte Zeichen so oft hinter- und untereinander schreiben, bis sich daraus eine Form ergibt. Diese Form des „Malens“ ist bei vielen Konsolenprogrammen üblich und wird auch heute noch verwendet.

Aufgabenstellung

Schreibe ein Programm, das das folgende Muster in der Konsole ausgibt:

```
      *
     ***
    *****
   *******
  *********
 ***********
*************
     ***
```

Testfälle

Wenn die Tanne wie angegeben in der Konsole ausgegeben wird, dann hast du alles richtig gemacht und diese Aufgaben erfolgreich bearbeitet. Gesetzt den Fall, dass du noch weitere Programme dieses Typs erstellen willst, geben wir dir hier noch weitere Anregungen (du kannst dir aber auch gerne selbst was überlegen!):

```
Sanduhr: *****        Pizzastück: **********       Diamant:   **
          ***                     *      *                  *  *
           *                       *    *                  *    *
          ***                       *  *                    *  *
         *****                       **                      **
```

Für diese zusätzlichen Trainingseinheiten bieten wir dir keine Lösungsvorschläge mehr an. Wir sind fest davon überzeugt, dass du das selbst hinbekommst und unsere Hilfe hierfür nicht mehr benötigst.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Schau’ dir die allererste Aufgabe dieses Buches doch noch einmal an und überlege dir, wie die Ausgabe für jede Zeile von oben nach unten aussehen muss.
- In Processing für Java gibt es zwei Befehle, mit denen du Text in die Konsole schreiben kannst. Der eine fügt eine neue Zeile hinzu, der andere hingegen nicht.
- Das Sternchen- und das Leerzeichen führen zum Ziel!

9 Referenzdatentypen

9.1 Warm-up

In allen Übungsaufgaben, mit denen wir bisher in Berührung gekommen sind, haben wir Variablen benötigt, um Werte zu berechnen, zu speichern oder abzurufen. Wir wissen, dass jede Variable einem bestimmten Datentyp zugeordnet wird. Der Datentyp ist eine Menge von Werten und Operationen, die auf diesen Werten definiert sind.

Wenn wir mit Zahlen umgehen, haben wir es mit Ganzzahl- oder Fließkommadatentypen zu tun. Dabei werden die Datentypen in vielen Programmiersprachen noch mal nach ihrem Wertebereich (bei Integer-Werten) bzw. nach ihrer Genauigkeit (bei Fließkommazahlen) in verschiedene Datentypen unterteilt. Dazu kommt dann meistens noch der boolesche Datentyp, der die Werte `true` und `false` verarbeiten kann. Gegebenenfalls gibt es auch noch einen Datentypen für die Repräsentation eines Zeichens, wie zum Beispiel den Datentyp `char` in Java. Bei all diesen Datentypen handelt es sich um **Wertetypen**. Je nach verwendeter Programmiersprache werden sie auch als *primitive Datentypen* oder *elementare Datentypen* bezeichnet. Egal, wie sie nun genannt werden, sie haben eine Sache gemein: Sie operieren auf einfachen Zahlen und Zeichen.

Neben diesen elementaren Datentypen existiert aber noch eine zweite Art von Datentypen, zu denen beispielsweise Strings, Arrays oder Klassen gehören. Sie werden als **Referenzdatentypen** oder auch als **zusammengesetzte Datentypen** bezeichnet. Ein Array ist bekanntlich eine endliche Menge von Elementen eines anderen Datentyps, zum Beispiel ein Array von Integer-Werten. Ein String ist eine endliche Menge von Zeichen und definiert beispielsweise Operationen wie Konkatenation, Länge, Teilstring, Vergleich usw. Zuletzt betrachten wir noch den im vorigen Kapitel besprochenen Datentyp der Klasse. Hier definieren wir selbst, aus welchen Attributen und Methoden sie besteht, und somit auch die verwendeten Daten und Datentypen.

Der entscheidende Unterschied liegt darin, wo die Werte eines Datentyps abgelegt werden: entweder auf dem **Stack**, bei dem es sich um einen kleineren Speicherbereich mit sehr performantem Zugriff handelt, oder dem **Heap**, einem wesentlich größeren Speicherbereich, der aber mit weniger Performance arbeitet. Die elementaren Datentypen werden auf dem Stack abgelegt und enthalten direkt die eigentlichen Daten. Demgegenüber werden die Daten von Referenzdatentypen auf dem Heap gespeichert. Wenn wir beispielsweise die Instanz einer Klasse anlegen, wird zunächst Speicher auf dem Heap allokiert. Wie viel Speicher das System reserviert, hängt von den in der Klasse verwendeten Datentypen ab.

Denn für jeden Datentyp ist in einem Programmiersystem festgehalten, wie viel Speicher er benötigt. Insofern ist die Addition der Werte hier richtungsweisend.

Wenn also der Speicher reserviert wurde, erhalten wir in der Variablen die sogenannte **Objektreferenz**. In Programmiersprachen wie C wird hier auch von einem **Zeiger** (engl. *pointer*) gesprochen. Diese Objektreferenz ist also die Startadresse des Speicherbereichs im Heap, ab der die Daten abgelegt sind. Hierbei ist von der **Objektidentität** die Rede, die dann alle notwendigen Daten zu unserem Objekt hält.

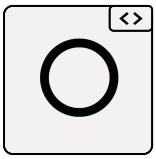
Es kann problematisch sein, mit Referenzdatentypen umzugehen. Dann nämlich, wenn mehrere Variablen auf dieselben Adressen verweisen. Hier führt die Änderung von Werten auf jeder Seite zu Änderungen an denselben Daten! Da dies in den seltensten Fällen tatsächlich gewünscht ist, sollten wir die Daten kapseln. Dies erreichen wir dadurch, dass wir eine Kopie der Daten anlegen, um die Unveränderlichkeit der Daten zu gewährleisten. Unter Umständen müssen wir dies auch in Methoden und für Rückgabewerte berücksichtigen!

Und nun – ran an die Workouts und viel Erfolg beim Training!

9.2 Workout

W.9.1 Kreis-Klasse

Schwierigkeit Zeitaufwand Kreativität



Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Referenzdatentypen

Beschreibung

Zur besseren Charakterisierung von Kreisen wollen wir eine Kreis-Klasse schreiben, die alle Informationen zu einem Kreis in einem Objekt bündelt.

Ein Kreis kann z. B. mit den Parametern der Position (x,y) sowie dem Radius (r) beschrieben werden.

Aufgabenstellung

Schreibe eine Klasse `Coordinate` mit Konstruktor, welche die Informationen zu den Koordinaten beinhaltet. Schreibe dort die Funktion `toString()`, welche die Koordinaten in der Form `(x,y)` als String zurückgibt.

Schreibe eine Klasse `Circle` mit Konstruktor, welche die Informationen zu einem Kreis enthält (Position, Radius). Schreibe dort die Funktion `area()` zur Rückgabe des Flächeninhaltes sowie die Funktion `toConsole()` zur Ausgabe der Kreisinformationen in der Konsole.

Die Ausgabe in der Konsole könnte zum Beispiel so aussehen:

```
Ich stehe bei (50, 100) und bin 1017.87604809 groß.
```

Prüfe anschließend außerhalb der Klassen die Circle-Funktion und erzeuge eine Konsolenausgabe mit Testdaten.

Testfälle

- x: 10, y: 43, r: 4
  – `Ich stehe bei (10, 43)und bin 50.2654838562 groß.`

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Der Flächeninhalt eines Kreises kann mit folgender Formel berechnet werden:
  $A = \pi * r^2$
- Vielleicht hilft es, zunächst die Coordinate-Klasse zu testen und anschließend erst die Circle-Klasse zu programmieren.
- Besonders in Python ist es wichtig, Zahlen zunächst in Strings umzuwandeln. Andernfalls wird bei der Ausgabe in der Konsole ein Fehler auftauchen.

W.9.2 Mathematischer Bruch

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Referenzdatentypen

Beschreibung

Zur Darstellung und Berechnung von mathematischen Brüchen wollen wir unseren eigenen Datentyp schreiben. Dieser soll neben der Bruchdarstellung auch Brüche voneinander addieren oder subtrahieren können.

Zur Erinnerung:

- Addition von Brüchen:

$$\frac{a}{b} + \frac{c}{d} = \frac{(a * d) + (c * b)}{b * d}$$

- Multiplikation von Brüchen:

$$\frac{a}{b} * \frac{c}{d} = \frac{a * c}{b * d}$$

Aufgabenstellung

Schreibe die Klasse `Fraction`, die einen Bruch repräsentiert, und gib einen Konstruktor und Implementierungen für die Methoden `add()` und `multiply()` an. Getter- und Setter-Methoden sowie eine `toString()`-Methode sollten ebenfalls eingebaut werden.

Teste die Klasse mit Beispielrechnungen in der Konsole.

Testfälle

- $\frac{1}{2} + \frac{1}{4} = \frac{(1 * 4) + (1 * 2)}{2 * 4} = \frac{6}{8}$
- $\frac{1}{2} * \frac{1}{4} = \frac{1 * 1}{2 * 4} = \frac{1}{8}$

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Natürlich kannst und solltest du deinen Datentyp zur Berechnung mit anderen Brüchen weiterverwenden.
- Als Datentyp bieten sich ganze Zahlen (Integer) an, da sich in Brüchen keine Kommazahlen befinden dürfen.
- Wenn du Bruch 1 mit Bruch 2 addierst oder multiplizierst, sollte sich weder der Wert von Bruch 1 noch von Bruch 2 verändern.

W.9.3 Highscore-Liste

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Referenzdatentypen

Beschreibung

Von klassischen Computerspielen kennen wir vielleicht noch die sogenannte Highscore-Liste. Diese stellen die höchsten in dem entsprechenden Spiel erreichten Punktzahlen in Tabellenform absteigend dar. Eine solche wollen wir mithilfe der folgenden Eigenschaften realisieren:

> *„Eine Highscore-Liste für ein Computerspiel verwaltet mehrere Einträge, die jeweils durch einen Spitznamen der spielenden Person repräsentiert sind. Die Highscore-Liste verfügt zudem über gängige Methoden, die das Hinzufügen von Platzierungen und die Ausgabe der gesamten Liste ermöglichen."*

Aufgabenstellung

Überlege dir ein UML-Klassendiagramm für die beschriebenen Eigenschaften.

Implementiere die Klasse `HighscoreEntry`, mit der wir Einträge für die Highscore-Liste erzeugen können. Diese Klasse soll eine geeignete `toString()`-Methode beinhalten.

Schreibe die Klasse `HighscoreTable`, die die Highscore-Liste realisiert. Sorge dafür, dass die `HighscoreEntry`-Listeneinträge sinnvoll initialisiert werden. Beachte außerdem, dass neue Listeneinträge immer an der entsprechenden Position in der Highscore-Liste hinzugefügt werden müssen. Bereits vorhandene Einträge müssen gegebenenfalls um eine Position verschoben werden.

Testfälle

- Befülle die Highscore-Liste zunächst mit durchnummerierten Namen. Zum Beispiel erst mit `Name0`, dann `Name1` usw. Wenn du danach einen neuen Eintrag in der Mitte der Highscore-Liste hinzufügst, darf kein Eintrag in der neuen Liste fehlen.
- Ebenfalls dürfen keine neuen Einträge doppelt in der Liste vorhanden sein. Wenn dies der Fall sein sollte, hast du vermutlich unabsichtlich die gleiche Variablenreferenz verwendet.

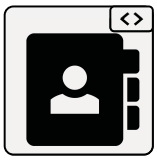
Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Referenzdatentypen sollten am besten in einer Schleife Element für Element kopiert werden. Sonst besteht die Gefahr, dass die gleiche Referenz im neuen Element verwendet wird, also keine richtige Kopie angelegt wurde.
- Lege ein Array für alle Einträge an und speichere darin entsprechend HighscoreEntry-Elemente.
- Wenn deine `HighscoreEntry`-Klasse eine `toString()`-Methode hat, kannst du jeden Eintrag mit einfacher Angabe des Variablennamens ausgeben. Das erleichtert die Programmierarbeit in großen Quelltexten.

W.9.4 Adressbuch

Schwierigkeit Zeitaufwand Kreativität



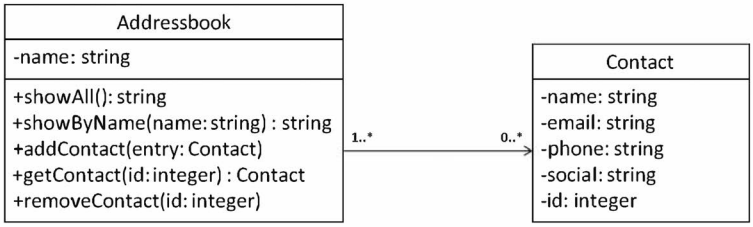
Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Referenzdatentypen

Beschreibung

Wir wollen zur Verwaltung unserer Kontakte ein Adressbuch programmieren. Bereits vorgegeben ist dieses UML-Klassendiagramm, welches zur Implementierung entwickelt wurde:



Die `Contact`-Klasse enthält nur Getter- und Setter-Methoden. Diese sind im Diagramm nicht explizit angegeben.

Aufgabenstellung

Implementiere das dargestellte UML-Klassendiagramm.

Schreibe ein Programm, das ein Adressbuchverwaltungsprogramm realisiert. Instanziiere dazu mithilfe der folgenden Angaben die entsprechenden Objekte:

Adressbuch: Privat

ID	Name	Email	Phone	Social Media
1	Ken Tern	ken.tern@mail.de	+49 221 3982781	@kentern
2	Bill Iger	bill.iger@gmx.de	+49 211 9821348	@billiger
3	Flo Kati	flo.kati@web.de	+49 251 9346441	@flokati
4	Ingeborg Mirwas	inge.mirwas@post.de	+49 228 4663289	@borgmirwas
5	Ann Schweigen	ann.schweigen@gmx.de	+49 231 6740921	@annschweigen
6	Mark Enschuh	mark.enschuh@gmail.com	+49 234 4565657	@markenschuh
7	Lee Köhr	lee.koehr@mail.de	+49 561 8976761	@leekoehr
8	Pit Schnass	pit.schnass@post.de	+49 721 4545754	@pitschnass

Adressbuch: Arbeit

ID	Name	Email	Phone	Social Media
1	Phil Tertüte	phil.tertuete@company.de	+49 177 1786756	@philtertuete
2	Flo Kati	flo.kati@laden.com	+49 161 2336541	@ibm.kati
3	Andreas Kreuz	andreas.kreuz@bazaar.de	+49 163 3442889	@asbazaar
4	Erkan Alles	erkan.alles@solver.de	+49 171 1442553	@easolver
5	Mark Reele	mark.reele@media.de	+49 151 5345612	@mrmedia
6	Roy Bär	roy.baer@media.de	+49 151 5477889	@rbmedia
7	Mario Nette	mario.nette@media.de	+49 151 5113341	@mnmedia
8	Klaus Uhr	klaus.uhr@media.de	+49 151 6743431	@kumedia

Teste anschließend die im UML-Klassendiagramm angegebenen Funktionen und erzeuge entsprechende Testausgaben.

Hinweis: Deine Freundin Flo Kati arbeitet zusammen mit dir. Dies bitte beachten, wenn der Kontakt in die Adressbücher „Privat“ und “Arbeit“ eingefügt wird.

Testfälle

- Zeige zunächst das komplette Adressbuch an, lösche einen Eintrag in der Mitte und zeige das Adressbuch erneut an. Sind die restlichen Einträge dann immer noch vorhanden?
- Lasse dir den Kontakt „Roy Bär“ mithilfe der Funktion `showByName()` ausgeben. Kommt der korrekte Inhalt zurück?
- Wird der Kontakt mit der ID 1 mit `getContact(1)` korrekt zurückgegeben?

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Die `Contact`-Klasse könnte eine `toString()`-Methode enthalten. Das würde zumindest den Code übersichtlicher halten.
- Bei den Suchfunktionen könntest du alle Elemente durchgehen und dann prüfen, ob ein Element mit dem gesuchten Wert übereinstimmt. Diesen kannst du dann zurückgeben.
- In ähnlicher Weise könntest du auch bei der Löschmethode herangehen. Unterschied ist hier natürlich, dass der zu löschende Wert einer neuen Liste nicht hinzugefügt wird. Hier gäbe es natürlich auch andere Varianten, das Problem zu lösen.

W.9.5 Digitaler Bilderrahmen

Schwierigkeit Zeitaufwand Kreativität

Themen

Mit dieser Aufgabe wollen wir folgende Dinge trainieren:

- Referenzdatentypen

Beschreibung

Digitale Bilderrahmen

Ein Bilderrahmen hat eine feste Anzahl an Speicherplätzen für Bilder. Diese sind beginnend mit `0` durchgehend nummeriert. Bilder werden mit der Klasse `Picture` repräsentiert, diese soll allerdings nur den Namen des Bildes in dieser Aufgabe speichern. Die umzusetzende `DigitalPictureFrame`-Klasse soll folgende Operationen unterstützen:

- Hinzufügen eines Bildes zum Bilderrahmen (`addPicture`). Wenn alle Speicherplätze bereits belegt sind, hat der Methodenaufruf keine Auswirkungen.
- Entfernen eines Bildes aus dem Bilderrahmen (`removePicture`). Welches Bild gelöscht werden soll, wird über die Position im Speicher angegeben. Die nachfolgenden Bilder rücken im Speicher um eine Position auf. Sind alle Speicherplätze leer, hat der Methodenaufruf keine Auswirkungen.
- Auslesen des nächsten Bildes aus dem Bilderrahmen (`getNext`). Beim ersten Aufruf wird bei der Speicherposition `0` gestartet. Wurde das letzte gespeicherte Bild durch diesen Methodenaufruf ausgelesen, beginnt es wieder von vorne bei der Speicherposition 0.
- Auslesen eines zufälligen Bildes (`getNextRandom`). Hierzu kann die Funktion `random()` verwendet werden.

Aufgabenstellung

Schreibe eine Klasse mit dem Namen `DigitalPictureFrame`, die einen digitalen Bilderrahmen repräsentieren soll, mit Instanzvariablen, geeignetem Konstruktor und den beschriebenen Methoden. Implementiere darüber hinaus die Klasse `Picture`, welche in der anderen Klasse verwendet werden soll.

Testfälle

- Füge drei Bilder hinzu und lösche das vierte Bild.
- Füge drei Bilder hinzu und lösche Bild 2. Bei Ausgabe von `getNextRandom()` sollten nur Bild 1 und Bild 3 angezeigt werden.

Algorithmische Tipps

Wenn du stockst und nicht weiterweißt, dann versuch mal Folgendes:

- Wenn du ein Bild hinzufügst, speichere das Bild in aktueller Array-Position und erhöhe diese um 1.
- Sollte die Array-Position über die erlaubte Menge gehen, muss entsprechend im Code reagiert werden. Echte Profis schaffen den notwendigen Code ohne If-Else-Anweisungen. Kleiner Tipp: Modulo-Operator.
- Beim Löschen eines Bildes könntest du ab der Löschposition loszählen und alle folgenden Bilder um eine Stelle nach vorne kopieren.