Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 2 SoSe 2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof R. Lorenz, L. Petrak, J. Linne, V. Le Claire, P. Schalk 17.05.2024

Übungsblatt 4

Abgabe: 03.06.2024, 10:00 Uhr (im Digicampus via VIPS: .java-Dateien für Code, .uxf für UML, .pdf für alles andere)

- Dieses Übungsblatt muss im Team abgegeben werden (Einzelabgaben sind nicht erlaubt!).
- Die **Zeitangaben** geben zur Orientierung an, wie viel Zeit für eine Aufgabe später in der Klausur vorgesehen wäre; gehen Sie davon aus, dass Sie zum jetzigen Zeitpunkt wesentlich länger brauchen und die angegebene Zeit erst nach ausreichender Übung erreichen.

* leichte Aufgabe / ** mittelschwere Aufgabe / *** schwere Aufgabe

Aufgabe 13 ** *(Parametrisierte Klassen und Java-Collections, 20 Minuten)*

In dieser Aufgabe lernen Sie mehr über die Schnittstelle `Collection` und die Klassen, die diese Schnittstelle implementieren.
Mit „ausgeben“ ist immer die Ausgabe auf der Kommandozeile gemeint.

a) *(*, 6 Minuten)*
Implementieren Sie eine Programmklasse, die

- ein `LinkedList`-Objekt für `Integer`-Objekte erzeugt, das von einer `Collection`-Variable `firstCollection` referenziert wird,
- ein `HashSet`-Objekt für `Object`-Objekte erzeugt, das von einer `Collection`-Variable `secondCollection` referenziert wird,
- 5 `Integer`-Objekte mit zufälligem Wert zwischen 0 und 9 (jeweils einschließlich) **ohne** Autoboxing zu `firstCollection` hinzufügt,
- 20 `Integer`-Objekte mit zufälligem Wert zwischen 0 und 9 (jeweils einschließlich) **mit** Autoboxing zu `secondCollection` hinzufügt und
- unter Verwendung einer geeigneten Methode der Schnittstelle ausgibt, ob `firstCollection` vollständig in `secondCollection` enthalten ist oder nicht.

b) *(gemischte Liste, **, 14 Minuten)*
Implementieren Sie eine Programmklasse, die

- ein `ArrayList`-Objekt für `Object`s erzeugt und
  - den String `"Polymorphismus"`,
  - ein `StringBuilder`-Objekt mit dem Inhalt `"spätes Binden"`,
  - ein `LocalDate`-Objekt mit dem aktuellen Datum,
  - ein `Random`-Objekt und
  - ein `Scanner`-Objekt mit dem Standardeingabestrom als Parameter hinzufügt,
- das `Iterator`-Objekt der Liste in einer Variable speichert und mit ihm über die Liste iteriert und dabei
  - mit der `getClass()`-Methode überprüft, ob es sich bei dem aktuellen Objekt um ein `LocalDate`-Objekt handelt,
  - es in diesem Fall in ein `LocalDate`-Objekt umwandelt und das Datum ausgibt, das zwei Wochen zuvor war,
  - mit `instanceof` überprüft, ob es sich bei dem aktuellen Objekt um ein `CharSequence`-Objekt handelt,
  - es in diesem Fall auf den Typ `CharSequence` castet und das erste und das letzte Zeichen davon ausgibt und
  - sonst die Zeichenkettendarstellung des Objekts auf der Kommandozeile ausgibt,
- jedes Objekt mit einer for-each-Schleife auf der Kommandozeile ausgibt.

Wie unterscheiden sich die Zeichenkettendarstellungen des `Random`- und des `Scanner`-Objekts? Beantworten Sie diese Frage in zwei kurzen Sätzen in einem Kommentar in Ihrem Abgabecode.

Aufgabe 14 *(Maps, 25 Minuten)*

In dieser Aufgabe lernen Sie die Datenstruktur `Map` kennen. Es gibt verschiedene Klassen für Maps wie z.B. `HashMap`, welche wir in dieser Aufgabe verwenden, aber auch `IdentityHashMap`, `EnumMap` oder `TreeMap`. Maps sind parametrisierte Klassen und verwalten Key-Value-Paare: Sie ermöglichen es, zu beliebigen Objekt-Typen weitere Informationen in Form von Objekten zu speichern, mit zwei Einschränkungen:

- Es kann kein Key doppelt vorkommen, genauer: Für jedes Paar unterschiedlicher Keys `key1` und `key2` liefert `key1.equals(key2) false`.
- Für jeden Key ist höchstens ein Value hinterlegt.

Die zweite Einschränkung lässt sich durch die allgemein gehaltene Form der Schnittstelle leicht umgehen: Auch eine `ArrayList` ist ein Objekt und kann als Value eines Keys verwendet werden. Mit diesen Vorinformationen können Sie nun die folgenden Teilaufgaben bearbeiten. Die wichtigsten Methoden der Schnittstelle `Map<K, V>` sind:

- `Set<K> keySet()`: Gibt ein `Set`-Objekt zurück, das alle in der Map enthaltenen Keys enthält.
- `V put(K key, V value)`: Setzt den Wert `value` für den Key `key`. Gibt den zuvor für `key` gespeicherten Wert zurück, sofern vorhanden, und sonst `null`.
- `V get(Object key)`: Gibt den Value zurück, der für den Key `key` hinterlegt ist, oder `null`, wenn für `key` nichts hinterlegt ist
- `Collection<V> values()`: Gibt ein `Collection`-Objekt zurück, das alle von der Map verwalteten Werte enthält.

a) *(Maps kennenlernen, **, 8 Minuten)*
Erstellen Sie ein streng typisiertes `HashMap`-Objekt für `Character-Integer`-Paare und eine Variable `map`, welche dieses Objekt referenziert. Iterieren Sie über die Kommandozeilenparameter des Programms und speichern Sie in `map`, wie oft jedes enthaltene Zeichen vorkommt. Geben Sie für alle enthaltenen Zeichen die jeweilige Anzahl der Vorkommen aus. Achten Sie sowohl beim Befüllen als auch beim Ausgeben der Map darauf, dass die `get`-Methode für Keys, die nicht vorhanden sind, `null` zurückgibt.

b) *(Lagerverwaltung, **, 7 Minuten)*
In dieser Aufgabe sollen Sie eine einfache Lagerverwaltung mithilfe einer `HashMap` implementieren. Der Key entspricht dem Namen des gelagerten Gegenstands und der Value der Anzahl der davon gelagerten Exemplare. Implementieren Sie eine Klasse `InventoryManagement` mit einem `HashMap`-Objekt zur Verwaltung von `String-Integer`-Paaren und drei Methoden:

- `boolean containsItem(String item)`: Gibt zurück, ob `item` bereits als Key vorhanden ist.
- `void addItem(String item, int amount)`: Fügt `amount` Stück vom `item` zur bisher gespeicherten Anzahl hinzu. Wenn `item` noch nicht vorhanden ist, ist `amount` die neue Anzahl.
- `int getItemCount(String item)`: Gibt zurück, wie viele Exemplare von `item` auf Lager sind. Wenn `item` nicht vorhanden ist, soll 0 zurückgegeben werden.

Testen Sie Ihre Implementierung mit der Programmklasse `InventoryManagementMain`, die Sie im Digicampus finden.

c) *(***, Notenverwaltung, 10 Minuten)*
In dieser Aufgabe sollen Sie eine einfache Notenverwaltung mithilfe einer `HashMap` implementieren. Der Key entspricht der Matrikelnummer und der Value einer Liste von gespeicherten Noten. Implementieren Sie eine Klasse `GradeTracker` mit einem `HashMap`-Objekt zur Verwaltung von `Integer-ArrayList<Double>`-Paaren und drei Methoden:

- `void addGrade(int enrolmentNumber, double grade)`: Fügt die Note `grade` zur Matrikelnummer `enrolmentNumber` hinzu. Achten Sie darauf, dass auch das Hinzufügen für eine komplett neue Matrikelnummer funktioniert: Wofür müssen Sie dann sorgen?
- `double calculateAverageGrade(int enrolmentNumber)`: Berechnet die Durchschnittsnote des Studierenden mit der Matrikelnummer `enrolmentNumber`. Sind keine Noten für `enrolmentNumber` hinterlegt, soll `-1.0` zurückgegeben werden.
- `double calculateAverageAverageGrade()`: Berechnet den Durchschnitt aller Durchschnittsnoten. Sind noch keine Noten gespeichert, soll `-1.0` zurückgegeben werden.

Testen Sie Ihre Implementierung mit der Programmklasse `GradeTrackerMain`, die Sie im Digicampus finden.

Aufgabe 15 *(Funktionale Schnittstellen und Lambda-Ausdrücke, 22 Minuten)*

In dieser Aufgabe lernen Sie funktionale Schnittstellen und Lambda-Ausdrücke näher kennen: Funktionale Schnittstellen dienen als Objekt-Verpackungen von Methoden. Lambda-Ausdrücke bieten die Möglichkeit, funktionale Schnittstellen zu verwenden, ohne diese Schnittstellen explizit in einer Klasse implementieren zu müssen.

a) *(*, 8 Minuten)*
In dieser Teilaufgabe sollen Sie funktionale Schnittstellen explizit implementieren, um das Konzept besser zu verstehen:

- Implementieren Sie eine Klasse `IsEven`, welche die Schnittstelle `Predicate<Integer>` implementiert: `test(Integer value)` soll zurückgeben, ob `value` gerade ist.
- Implementieren Sie eine Klasse `DoubleAndPrint`, welche die Schnittstelle `Consumer<Integer>` implementiert: `accept(Integer value)` soll das Doppelte von `value` auf der Kommandozeile ausgeben.

Schreiben Sie eine Programmklasse, die

- ein `ArrayList`-Objekt für `Integer`-Objekte anlegt,
- diese mit 10 zufälligen Zahlen zwischen 0 und 9 (jeweils einschließlich) befüllt,
- mit einem `IsEven`-Objekt und einer geeigneten Methode der Klasse `ArrayList` alle geraden Elemente aus der Liste entfernt und
- mit einem `DoubleAndPrint`-Objekt und einer geeigneten Methode der Klasse `ArrayList` das Doppelte aller enthaltenen Werte ausgibt.

b) *(**, 2 Minuten)*
Schreiben Sie nun eine Programmklasse, die statt des `IsEven`- und des `DoubleAndPrint`-Objekts Lambda-Ausdrücke verwendet, um dasselbe Verhalten zu erzielen: Schreiben Sie eine Programmklasse, die

- ein `ArrayList`-Objekt für `Integer`-Objekte anlegt,
- diese mit 10 zufälligen Zahlen zwischen 0 und 9 (jeweils einschließlich) befüllt,
- mit einem Lambda-Ausdruck und einer geeigneten Methode der Klasse `ArrayList` alle geraden Elemente aus der Liste entfernt und
- mit einem Lambda-Ausdruck und einer geeigneten Methode der Klasse `ArrayList` das Doppelte aller enthaltenen Werte ausgibt.

c) *(**, 8 Minuten)*
Man kann auch komplizierteres Verhalten in funktionalen Schnittstellen verpacken. Implementieren Sie ein Klasse `ContainsSpecialCharacter`, welche die Schnittstelle `Predicate<String>` implementiert: Der Test soll zurückgeben, ob der übergebene String mindestens eines der Sonderzeichen aus dem String

```
"!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"
```

enthält. Schreiben Sie außerdem eine Programmklasse, die

- ein `ArrayList`-Objekt für `String`-Objekte anlegt,
- 5 Zeilen vom Benutzer aus der Kommandozeile einliest und in der Liste speichert,
- mit einem `ContainsSpecialCharacter`-Objekt und einer geeigneten Methode der Klasse `ArrayList` alle Elemente aus der Liste entfernt, die mindestens eines der Sonderzeichen enthält und
- mithilfe einer Methodenreferenz die verbleibenden Elemente der Liste auf der Kommandozeile ausgibt.

d) *(**, 4 Minuten)*
Implementieren Sie eine Programmklasse, die das Verhalten der vorherigen Teilaufgabe mithilfe eines Lambda-Ausdrucks statt des `ContainsSpecialCharacter`-Objekts implementiert: Die Programmklasse soll

- ein `ArrayList`-Objekt für `String`-Objekte anlegen,
- 5 Zeilen vom Benutzer aus der Kommandozeile einlesen und in der Liste speichern,
- mit einem Lambda-Ausdruck alle Elemente aus der Liste entfernen, die mindestens eines der Sonderzeichen enthalten und
- mithilfe einer Methodenreferenz die verbleibenden Elemente der Liste auf der Kommandozeile ausgeben.

Aufgabe 16 ** *(Lambda-Ausdrücke und Stream-API, 25 Minuten)*

In dieser Aufgabe lernen Sie die Stream-API näher kennen und üben die Anwendung von Lambda-Ausdrücken weiter ein. Recherchieren Sie selbst geeignete Methoden der Klassen `IntStream`, `DoubleStream` und der Schnittstelle `Stream<>`.

a) *(*, 4 Minuten)*
Implementieren Sie eine Programmklasse, die

- mithilfe eines `Random`-Objekts ein `IntStream`-Objekt mit Zahlen zwischen 0 und 9 (jeweils einschließlich) erzeugt,
- alle Zahlen, die nicht durch 3 teilbar sind, aus dem Stream entfernt,
- die Länge des Streams auf 10 beschränkt und
- die Elemente in jeweils einer Zeile mit einer Methodenreferenz auf der Kommandozeile ausgibt.

b) *(*, 4 Minuten)*
Implementieren Sie eine Programmklasse, die

- mithilfe eines `Random`-Objekts ein `DoubleStream`-Objekt mit Zahlen aus dem Intervall $[0, 1[$ erzeugt,
- den Wert aller enthaltenen Zahlen verdoppelt,
- die Länge des Streams auf 10 beschränkt und
- die Summe der enthaltenen Elemente auf der Kommandozeile ausgibt.

c) *(*, 4 Minuten)*
Implementieren Sie eine Programmklasse, die

- ein `IntStream`-Objekt mit den ganzen Zahlen zwischen 1 und 100 erzeugt,
- diesen in ein `DoubleStream`-Objekt konvertiert und
- den Durchschnitt der enthaltenen Zahlen berechnet und auf der Kommandozeile ausgibt.
- Überprüfen Sie, ob das Rückgabeobjekt der passenden Methode tatsächlich einen Wert beinhaltet, bevor Sie diesen ausgeben.

d) *(**, 6 Minuten)*
Implementieren Sie eine Klasse `StringSupplier`, welche die Schnittstelle `Supplier<String>` implementiert:

- Ein `StringSupplier`-Objekt speichert eine Länge sowie ein `Random`-Objekt, die beide beim Erzeugen des Objekts gesetzt und nicht mehr geändert werden können.
- Die `get()`-Methode erzeugt dann einen String mit einer zufälligen Länge zwischen 0 und der maximalen Länge (jeweils einschließlich).
- Die im String enthaltenen Zeichen sollen sichtbare ASCII-Zeichen sein[1], also `char`-Werte mit Werten zwischen 32 und 126 (jeweils einschließlich).
- Mit der `appendCodePoint(int codePoint)` kann ein `char`, gegeben durch seinen Zahlenwert, an ein `StringBuilder`-Objekt angehängt werden. D.h. mit
  `stringBuilder.appendCodePoint(65);`
  wird also das Zeichen `'A'` an das `StringBuilder`-Objekt `stringBuilder` angehängt.

e) *(***, 7 Minuten)*
Implementieren Sie eine Programmklasse, die

- mithilfe eines `StringSupplier`-Objekts einen `String`-Stream erzeugt,
- alle leeren Zeichenketten aus dem Stream entfernt,
- an jedes Element des Streams `", "` anhängt,
- die Länge des Streams auf 10 Elemente begrenzt,
- mit einer geeigneten Methode alle Zeichenketten aneinanderhängt und
- das Ergebnis auf der Kommandozeile ausgibt.

[1] https://www.torsten-horn.de/techdocs/ascii.htm