

Objektorientierte Programmierung

Blatt 8

Institut für Softwaretechnik und Programmiersprachen | Sommersemester 2024
Matthias Tichy, Raphael Straub und Florian Sihler

Abgabe (git) bis
23. Juni 2024

Exceptions and IO

6 ★ 12 ■ 8 ●

- Exception handling in Java
- Java-IO
- Reguläre-Ausdrücke

Aufgabe 1: Exceptional Ice Cream

In dieser Aufgabe helfen wir unserem Lieblings-Eisverkäufer, den wir bereits von einem früheren Blatt kennen, beim Erstellen eines Eisautomaten, welcher ihm die Arbeit abnimmt. Der Eisverkäufer hat bereits eine Enumeration `IceCream` erstellt, die die Sorte des Eisbechers repräsentiert. Da alle Eisbecher die gleiche Größe haben, reicht uns die Enumeration. Außerdem hat er die Klasse `IceCreamMachine` erstellt, die die verfügbare Menge an Eisbechern speichert. Diese Klassen finden Sie im vorbereiteten Repository. Die Funktionalität des Automaten soll nun von uns implementiert werden.

a) Buying Ice Cream



Implementieren Sie die Methode `IceCream buyIceCream(String sort)` in der Klasse `IceCreamMachine`. Diese Methode soll die private Methode `IceCream produceIceCream(String sort)` verwenden, um einen Eisbecher der Sorte `sort` zu erhalten. Wenn die Sorte nicht verfügbar ist, soll eine `IllegalArgumentException` geworfen werden.

Wenn der Kauf erfolgreich war, soll die Methode die gelagerte Menge der entsprechenden Sorte um eins reduzieren.

b) Prepare for the Worst



Wir bereiten uns direkt auf das Schlimmste vor und implementieren eine Exception, die im Laufe der Aufgabe benötigt wird. Erstellen Sie die Klasse `IceCreamNotAvailableException`. Hierbei soll `IceCreamNotAvailableException` geworfen werden, wenn der Automat keine Eisbecher mehr hat. `IceCreamNotAvailableException` soll von `Exception` erben. Sorgen Sie dafür, dass die Exception eine sinnvolle Fehlermeldung erzeugt.

Verwenden Sie `IceCreamNotAvailableException` in der Methode `buyIceCream`, falls keine Eisbecher mehr verfügbar sind.

c) Difference



Wie unterscheidet sich die Verwendung von `IceCreamNotAvailableException` und `IllegalArgumentException`?

d) Bulk Orders



Implementieren Sie die Methode `List<IceCream> buyIceCream(List<String> orders)` in der Klasse `IceCreamMachine`. Diese Methode soll eine Liste von Eisbechern zurückgeben, die die Bestellungen der Kunden erfüllen. Falls beim Kauf eines Eisbechers ein Fehler auftritt, soll dieser ignoriert werden und die restlichen Bestellungen weiter bearbeitet werden. Die resultierende Liste soll alle Eisbecher enthalten, die erfolgreich gekauft wurden.

e) Maintenance Error



Leider haben wir festgestellt, dass die Maschine nicht immer einwandfrei funktioniert. Wenn wir trotz eines technischen Defekts versuchen, ein Eis zu kaufen, besteht die Gefahr, dass die Maschine irreparabel beschädigt wird. Deswegen wollen wir vor jedem Kauf prüfen, ob die Maschine in einem guten Zustand ist.

Zum Glück verfügen wir über eine High-Tech Eismaschine, die sich zu einem gewissen Grad selbst reparieren kann. Passen Sie die Methode `buyIceCream` so an, dass sie vor jedem Kauf die existierende Methode `performMaintenance` aufruft.

Beachten Sie hierbei, dass die Methode `performMaintenance` eine `MaintenanceException` werfen kann, die signalisiert, dass sich die Maschine in einem kritischen Zustand befindet, und eine selbstständige Reparatur nicht mehr möglich ist.

Falls dies auftritt, soll die Methode `buyIceCream` die Exception abfangen und eine `RuntimeException` werfen, die die `MaintenanceException` als Ursache enthält.

Passen Sie die Bulk-Order-Version der `buyIceCream` Methode so an, dass sie mit der neuen Exception gleich umgeht wie mit `IceCreamNotAvailableException`. Welches Problem fällt Ihnen an ihrer Implementation auf?

f) Error Recovery



Unsere Eismaschine hat eine Menge an Ersatzteilen, die wir verwenden können, um die Maschine zu reparieren. Implementieren Sie die Methode `boolean repairMachine(String componentId)` in der Klasse `IceCreamMachine`, die die Maschine repariert. Hierbei soll die Methode prüfen, ob eine Ersatzkomponente für die Komponente mit der ID `componentId` verfügbar ist. Eine Komponente ist eine valide Ersatz für eine andere Komponente, wenn sie den gleichen `componentType` hat. Falls eine passende Komponente gefunden wird, soll die kaputte Komponente ersetzt und `true` zurückgegeben werden. Falls das Ersetzen nicht möglich ist, soll `false` zurückgegeben werden.

Passen Sie nun die Methode `performMaintenance` an, sodass sie die Methode `repairMachine` aufruft, falls ein Fehler auftritt. Falls die Maschine nicht repariert werden kann, soll die Methode eine `MaintenanceException` werfen. Erweitern Sie hierfür gegebenenfalls die `MaintenanceException`, sodass sie die ID der Komponente enthält, die den Fehler verursacht hat.

Aufgabe 2: Sales Data

Nun helfen wir dem Eisverkäufer, die Daten seiner Verkäufe langfristig zu speichern und später wieder zu laden. Diese Daten kann er dann später nutzen, um seine Verkäufe zu analysieren und zu optimieren. Hierbei wollen wir die Daten in einer Datei speichern und aus ihr laden.

Dabei möchten wir uns noch nicht mit komplizierten Dateiformaten beschäftigen, sondern entscheiden uns für ein einfaches Format. Der Eisverkäufer speichert bei jedem Kauf die Sorte des Eisbechers, den Zeitpunkt des Kaufs, und den Ort des Kaufs. Diese Daten werden in der Klasse `Sale`, die Sie im vorbereiteten Repository finden, gespeichert. Jeder `Sale` soll in einer Zeile gespeichert werden, passend zu folgendem Format: `<Sort>, <Timestamp>, <Location>`.

Hier ein Beispiel:

Oasen-Fruchtmix, 01.06.2024, Sietch Tabr
Sanddünen-Schokolade, 02.06.2024, Arrakeen
Melange-Eis, 02.06.2024, Arrakeen
Oasen-Fruchtmix, 03.06.2024, Carthage

a) Save Data



Implementieren Sie die Methode `void saveSales(List<Sale> sales, String filename)`. Diese Methode soll die Daten in der Liste `sales` in der Datei `filename` speichern. Hierfür soll ein `FileWriter` verwendet werden. Verwenden Sie das oben genannte Format.

b) Load Data



Nun wollen wir die Daten laden.

Implementieren Sie dafür `List<Sale> loadSales(String filename)`. Diese Methode soll die Daten aus der Datei `filename` laden und als eine Liste von `Sale`-Objekten zurückgeben. Verwenden Sie zum Laden der Datei einen `Scanner` mit einem `FileInputStream`. Falls es Probleme beim Laden der Daten gibt, soll eine leere Liste zurückgegeben werden und keine Exception geworfen werden.

Hinweis: Sie können `String::split` verwenden, um die Zeilen zu parsen.

c) Measuring Performance



Der Eisverkäufer berichtet uns, dass der Eisverkauf in den letzten Wochen stark angestiegen ist und unsere Lösung effizient sein muss! Implementieren Sie die Klasse `SalesBenchmark` mit einem Konstruktor `SalesBenchmark(String filename)`, der den Dateinamen speichert.

Jetzt wollen wir eine `benchmark`-Methode implementieren, die die Zeit misst, die benötigt wird, um die Daten zu laden. Da es dem Eisverkäufer wichtig ist, dass wir die Performance von verschiedenen Ansätzen vergleichen, soll die Methode `benchmark` einen Parameter vom Typ `Function` übergeben bekommen und einen `long`-Wert zurückgeben. Achten Sie hierbei darauf, keinen `Raw-Type` zu verwenden, sondern den Typ kompatibel zu `List<Sale> loadSales(String filename)` zu wählen.

Die Methode `benchmark` soll die übergebene `Function` aufrufen und die Zeit messen und zurückgeben, die diese benötigt. Hierfür können Sie die Methode `System.currentTimeMillis()` verwenden.

Messen Sie nun die Zeit, die für das Laden der mitgelieferten Datei benötigt wird. Hierbei können Sie die Methode `loadSales` direkt als Referenz übergeben.

d) Improve Performance



Implementieren Sie jetzt eine weitere Methode, die die Funktionalität von `loadSales` implementiert, jedoch die Daten in einem `BufferedReader` liest. Vergleichen Sie die Performance der beiden Methoden und diskutieren Sie die Ergebnisse. Welche Version sollte der Eisverkäufer verwenden?

Falls Ihnen weitere Methoden einfallen, die die Datei zu lesen, können Sie diese ebenfalls implementieren und testen.

Aufgabe 3: The Bene Gesserit Code

Die Bene Gesserit sind für ihre geheimen Botschaften und verschlüsselten Kommunikationstechniken bekannt. In dieser Aufgabe helfen wir einem Mitglied der Bene Gesserit beim Dekodieren und Klassifizieren von Nachrichten, die in verschiedenen Strukturen verfasst sind. Diese Nachrichten enthalten versteckte Informationen, die nur durch das Verwenden von regulären Ausdrücken in Java entschlüsselt werden können.

a) Identify Message Types



Die Nachrichten folgen spezifischen Mustern, die es zu identifizieren gilt. Es gibt drei Typen von Nachrichten, die Sie erkennen müssen, und jede Nachricht beginnt mit einem Identifier, der den Typ bestimmt. Der Identifier kann durch einen der drei regulären Ausdrücke erkannt werden:

1. **Meeting** Nachrichten, die Treffen beinhalten, folgen dem Muster `^\d{3}-\d{2}-\d{4}$`.
Beispiel:
`#987-65-4321: Treffen am 01.06.2024 (Sietch Tabr) mit Group 7`
2. **Warning** Nachrichten, die Warnungen enthalten, folgen dem Muster `^[A-Z]{3}\d{4}[@#$$%^&*]{2}[a-z]{3}$`.
Beispiel:
`#ABC1234@#xyz: Harkonnen-Spy (DANGER)->protocol_17`
3. **Message** Allgemeine Nachrichten folgen dem Muster `^[A-Za-f0-9]{6}$`.
Beispiel:
`#FFA07A: Nachricht: Pläne für (Angriff (Sietch Tabr) bei (Dämmerung))`

Versuchen Sie zunächst, die drei Regex-Muster zu verstehen und zu erklären. Nutzen Sie hierfür ein entsprechendes Tool, wie etwa <https://regex101.com/>.

Erstellen Sie eine Methode `MessageType identifyMessageType(String message)`, die den Typ einer Nachricht bestimmt. Verwenden Sie hierfür die regulären Ausdrücke und die Klassen `Pattern` und `Matcher` aus dem Paket `java.util.regex`.

b) Extract Warning Details



Implementieren Sie die Methode `Map<String, String> extractWarningDetails(String message)`, die die Details einer Warning-Nachricht extrahiert. Die Methode soll ein `Map` zurückgeben, welche die folgenden Schlüssel-Wert-Paare enthält: "Topic", "Level", "Reaction".

Sie können davon ausgehen, dass die Nachrichten immer das folgende Format haben: `#<Type>:<Topic> (<Level>)-><Reaction>`. Beachten Sie, dass die Inhalte von `<Type>`, `<Topic>`, `<Level>` und `<Reaction>` keine Leerfelder enthalten. Darüber hinaus können Sie davon ausgehen, dass `<Type>` weder `:`, noch `#` enthält, dass `<Topic>` weder `:`, noch `(` enthält, dass `<Level>` weder `->`, noch `)` enthält, und dass `<Reaction>` nicht das Zeichen `>`, enthält. Allerdings können im restlichen String beliebig viele Leerfelder vorkommen. Verwenden Sie einen regulären Ausdruck und Capture Groups, um die Informationen zu extrahieren.

Beispiel:

Input: `"#ABC1234@#xyz: Harkonnen-Spy (DANGER)->protocol_17"`

Output: `{"Topic": "Harkonnen-Spy", "Level": "DANGER", "Reaction": "protocol_17"}`