# Programmierung 2 - Übungsaufgaben

DHBW Mannheim - WWI17SCB - May 2018

# 1. Exception Handling

# Kontrollfragen

* Welche Unterschiedlichen Fehler kennt ihr im Java-Umfeld?
* Welche Fehler sollten nicht, können oder müssen behandelt werden?
* Welche Arten von Ausnahmen sind Ihnen im Java-Umfeld bekannt?
* Erläutern Sie das Grundprinzip der Ausnahmebehandlung!
* Was verstehen Sie unter eine Ausnahme werfen, fangen und weitergeben)

# Kontrollfragen

* Welches sind die wesentlichen Methoden der Klasse Throwable?
* Worin unterscheiden sich checked und unchecked Exceptions?
* Wozu dient der finally-Block bei einer Try-Catch-Anweisung?
* Welche Möglichkeiten sind Ihnen zum Erzeugen eigener Ausnahmeklassen bekannt?
* Was würde passieren, wenn eine Ausnahme nicht abgefangen, sondern immer weitergegeben wird?

# Übung 1.1

Das folgende Programm soll die Zahlen von 1 bis 100 in eine Textdatei mit dem Namen ausgabe.txt schreiben. Leider kann das Programm in dieser Form nicht ausgeführt werden. Was ist der Grund dafür und wie können Sie es korrigieren?

(Anmerkung: Auch wenn der Input- & Output-Stream noch nicht behandelt wurde, können Sie die Frage schon jetzt beantworten.)

# Übung 1.1

```java
import java.io.FileWriter;
public class Uebung1 {
  public static void main(String[] args) {
    FileWriter datei;
    String text;

    datei = new FileWriter("ausgabe.txt");
    text = "1\n";
    for(int i = 2; i <=100; i++) {
      text += i;
      text += "\n";
    }
    datei.write(text, 0, text.length());
    datei.flush();
  }
}
```

# Übung 1.2

In Ihrem Unternehmen beziehen Sie Autositze. Für das Beziehen stehen Ihnen die Materialien Leder und Stoff zur Verfügung. Den Stoff können Sie in jeder beliebigen Farbe liefern. Leder jedoch ist nur in den Farben Schwarz und Weiß lieferbar.

Implementieren Sie eine Klasse AutoSitze gemäß der Vorgaben des UML- Diagramms. Der Konstruktor soll eine Fehlermeldung der von Ihnen zu implementierenden Fehlerklasse FalscheParameter erzeugen, sobald er ungültige Parameterkombinationen erhält. Der Meldungstext der Fehlerklasse soll die fehlerhafte Parameterkombination ausgeben.

Testen Sie Ihre Klasse AutoSitze mit einem kleinen Testprogramm TestAutoSitzeException. In Ihrem Testprogramm soll nach erfolgreichem Durchlauf des Konstruktors eine Meldung ausgegeben werden, in welcher Farbe und in welchem Material der Sitz bezogen wurde. Sollte während dem Durchlauf des Konstruktors eine Ausnahme ausgelöst werden, geben Sie bitte in Ihrem Testprogramm den Meldungstext der Ausnahme sowie einen kleinen Hinweis, dass das Beziehen fehlgeschlagen ist, aus.

# Übung 1.2

| AutoSitze |
| --- |
| - ledersitze: boolean<br>- farbe: String |
| + AutoSitze(bezug: boolean, color: String)<br>+ getFarbe(): String<br>+ isLedersitze(): boolean |

# 2. Collections

# Kontrollfragen

* Was können Datencontainer des Collection Framework enthalten?
* Welche drei Arten von Containern kennen Sie?
* Worin unterscheiden sich die drei Containerarten?
* Was ist der Unterschied zwischen Arrays und Containern des Collection Framework?
* Wozu werden Iteratoren benötigt?

# Kontrollfragen

* Sie möchten ein Schachfeld oder auch den Spielplan von „Schiffe versenken“ in Java implementieren. Dazu müssen Sie einen Repräsentanten des Spielfeldes im Speicher erzeugen und verwalten. Welche Datenstrukturen würden Sie dabei verwenden? Wie begründen Sie Ihre Entscheidung?
* Was müssen Sie bei Containern beachten, in denen die Objekte sortiert abgelegt werden?

# Kontrollfragen

* Wozu brauchen Sie die Interfaces Comparable und Comparator?
* Können Sie Listen-Container sortieren? Wenn ja, wie gehen Sie bei der Realisierung vor?
* Welche Möglichkeiten haben Sie, um Objekte miteinander zu vergleichen? Worin liegt der Unterschied der Vergleichsmöglichkeiten?

# Kontrollfragen

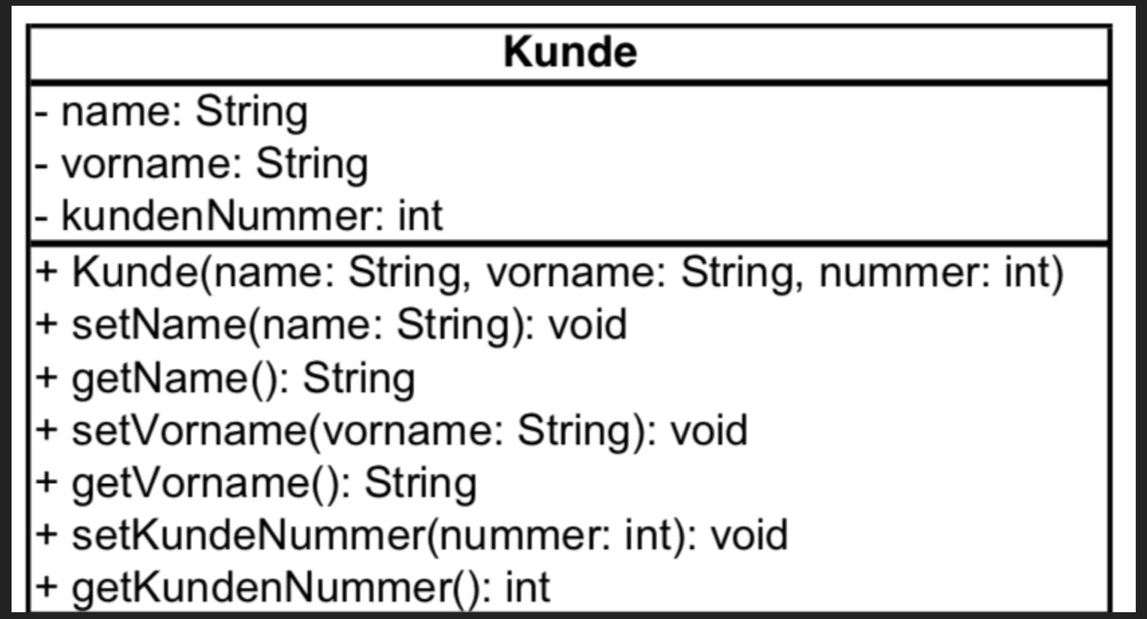
* Was müssen Sie beim Überschreiben der equals()-Methode für direkte und indirekte Sub-Klassen der Klasse Object beachten?
* Worin besteht der Zusammenhang zwischen der hashCode()- und der equals()- Methode?
* Beschreiben Sie zwei einfache Möglichkeiten, um eine hashCode()-Methode zu überschreiben!

# Kontrollfragen

* Was sind Wrapper-Klassen und wozu werden sie benötigt?
* Sie legen Objekte der Wrapper-Klasse Double in einem Datencontainer der Klasse TreeSet ab. Warum ist dies problemlos möglich?
* Worin liegen die wesentlichen Unterschiede zwischen Maps und Lists?

# Übung 2.1

Implementieren Sie zunächst eine Klasse Kunde gemäß der Vorgaben des UML- Diagramms.




Schreiben Sie sich ein Testprogramm TestKunde. In diesem Testprogramm erzeugen Sie sich zunächst fünf Objekte der Klasse Kunde. Namen, Vornamen und Kundennummern finden Sie in der folgenden Tabelle.

# Übung 2.1

| Name | Vorname | Kundennummer |
|---|---|---|
| Mustermann | Klaus | 4711 |
| Beispiel | Hans | 5180 |
| Mustermann | Hilde | 4712 |
| Vorbild | Theodor | 8278 |
| Dummy | Jimmy | 1111 |

Nachdem Sie die Objekte erzeugt haben, möchten Sie die Objekte in einem Datencontainer der Klasse TreeSet ablegen. Erweitern Sie entsprechend Ihre Klasse Kunde, so dass die Objekte der Klasse Kunde aufsteigend nach der Kundennummer sortiert werden.

Geben Sie abschließend die Elemente aus dem TreeSet über einen Iterator auf der Konsole aus, um den Erfolg der Sortierung zu überprüfen.

# Übung 2.2

Erweitern Sie die Klassen Kunde und TestKunde aus der vorangegangenen Übung so, dass Sie die Objekte der Klasse Kunde in einem Datencontainer der Klasse ArrayList nach dem Namen und bei Namensgleichheit nach dem Vornamen aufsteigend sortiert ablegen können.

Geben Sie abschließend auch die Elemente aus der ArrayList über einen Iterator ebenfalls auf der Konsole aus, um auch den Erfolg der zweiten Sortierung zu überprüfen.

# Übung 2.3

Ermöglichen Sie es, dass Objekte der Klasse Kunde aus den vorangegangenen Übungen miteinander verglichen werden können. Implementieren Sie dazu alle erforderlichen Methoden in der Klasse Kunde.

Testen Sie die Vergleichsmethode(n), indem Sie in einem Testprogramm TestVergleichKunde zwei gleiche Objekte der Klasse Kunde erzeugen und diese miteinander vergleichen. Geben Sie das Ergebnis des Vergleichs auf der Konsole aus.

# 5. Testing

# Kontrollfragen

* Welche drei Hauptziele werden mit Testing verfolgt?
* Welche Arten von automatisierten Tests kennt ihr?
* Was ist ein Unit Test und was testet er?
* Was ist ein Integration Test und was testet er?
* Sollte ich Getter-Methoden testen?
* Welchen Vorteil hat die Verwendung von TDD?
* Nach welchem Modell gehe ich als Entwickler bei der Verwendung von TDD vor?

# 6. Threads

# Kontrollfragen

* Was sind die Unterschiede zwischen Prozessen und Threads?
* In welchen Bereichen finden Threads Anwendung? Wodurch ergeben sich daraus Vorteile?
* Welche Möglichkeiten haben Sie, um Threads zu erzeugen? Wie können Threads gestartet werden?
* Nennen und beschreiben Sie kurz fünf unterschiedliche Zustände von Threads!

# Kontrollfragen

* Welche Möglichkeiten haben Sie, um Threads zu beenden? Beschreiben Sie eine Möglichkeit in wenigen Sätzen!
* Was würde passieren, wenn Sie einen Thread über die run()- statt über start()- Methode starten?
* Warum ist eine zeitliche Synchronisation von Threads erforderlich, wenn diese gemeinsamen Daten nutzen? Welche Möglichkeiten haben Sie, um Threads zu synchronisieren?

# Kontrollfragen

* Beschreiben Sie das Consumer-Producer-Problem und skizzieren Sie einen möglichen Lösungsansatz! Wo findet sich dieses Problem in der Praxis wieder?

# Übung 6.1

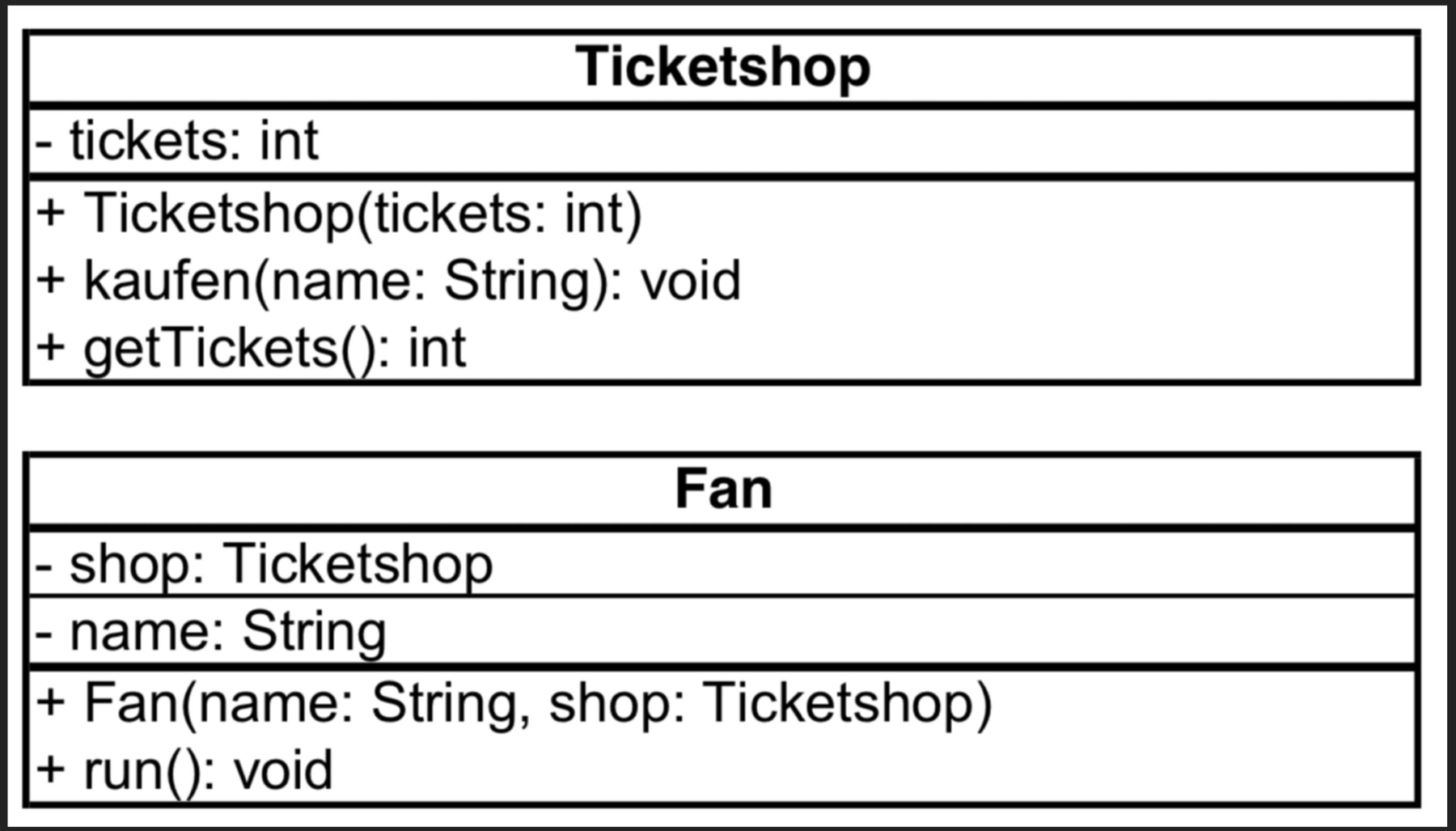
Im Repository findet ihr unter src/threads/ex3_bank eine Reihe von Klassen, die Geldtransfers in einer Bank abbilden.

* Lest und versteht den Code der Klassen.
* Führt die Main-Methode der Klasse MemoryConsistency3 aus. Was geht hier schief?
* An welcher Stelle tritt der Memory Consistence Error hier auf?
* Behebt den Fehler, sodass sich die gesamte Geldsumme nicht mehr verändert!

# Übung 6.2

Ein Ticketshop verkauft eine bestimmte Menge WM-Tickets an die angereisten Fußballfans. Implementieren Sie die Klassen Ticketshop und Fan analog der nachstehenden UML-Diagramme, wobei die Klasse Fan eine Subklasse der Klasse Thread darstellt.

# Übung 6.2


Ticketshop
- tickets: int
+ Ticketshop(tickets: int)
+ kaufen(name: String): void
+ getTickets(): int
Fan
- shop: Ticketshop
- name: String
+ Fan(name: String, shop: Ticketshop)
+ run(): void

# Übung 6.2

Geben Sie dem Konstruktor der Klasse Ticketshop die Anzahl der zu verkaufenden Tickets mit. Erzeugen Sie innerhalb der main()-Methode der Klasse Vorverkauf drei Instanzen der Klasse Fan und starten Sie die entsprechenden Threads. Die Fans kaufen Tickets bis der Vorrat im Ticketshop aufgebraucht ist. Sobald die Tickets ausverkauft sind, sollen die Threads über den Aufruf der interrupt()-Methode beendet werden. Achten Sie darauf, dass ggf. Methoden synchronisiert werden müssen.

# Übung 6.3

**Erweiterung:**

Erweitern Sie Ihr Programm aus Übung 6.2. Die Tickets sind noch nicht gedruckt, sondern sollen während dem Programm von einer Druckerei (Producer) hergestellt werden. Die maximal produzierbare Ticketanzahl beläuft sich auf 100 Stück und es sollen maximal 25 Tickets vorrätig sein. Die Tickets sollen weiterhin über den Ticketshop (Consumer) an die Fans verkauft werden. Setzen Sie das Consumer- Producer-Modell entsprechend zur Lösung der Problematik ein.

Vor der Implementierung sollten Sie zunächst die UML-Klassendiagramme für die Klassen Druckerei und den Ticketshop notieren.