

Objektinteraktion

Lernziele

- Sie können ein bestehendes Programm erweitern.
- Sie erkennen die Vorzüge der Modularisierung.
- Sie können eine einfache Problemstellung analysieren, geeignet auf verschiedene Klassen aufteilen und diese Klassen implementieren – unter Berücksichtigung der Clean Code Regeln und eines sauberen Klassendesigns.

Aufgabe 1

Forken Sie für diese Aufgabe die Projekte https://github.engineering.zhaw.ch/prog1-kurs/03_Praktikum_ZeitanzeigeEineKlasse und https://github.engineering.zhaw.ch/prog1-kurs/03_Praktikum_ZeitanzeigeZweiKlassen. Nutzen Sie BlueJ um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Im Lehrbuch wurde das Uhrenbeispiel diskutiert, den Code finden Sie im Projekt 03_Praktikum_ZeitanzeigeZweiKlassen. Das Projekt 03_Praktikum_ZeitanzeigeEineKlasse löst die genau gleiche Aufgabe, es wird aber nur eine Klasse verwendet.

Studieren Sie den Code in 03_Praktikum_ZeitanzeigeEineKlasse und vergleichen Sie ihn mit dem Code in 03_Praktikum_ZeitanzeigeZweiKlassen. Welche Variante gefällt ihnen besser? Wieso? Was sind die Vorteile der einen Variante im Vergleich zur anderen?

Meines Erachtens ist die Variante mit zwei Klassen sinnvoller. Denn wie wir in Kapitel 3 gelernt haben, sollte man grosse "Probleme" in kleinere "Teilprobleme" unterteilen. In diesem Fall mit zwei Klassen ist zum Beispiel die Limite ein solches Beispiel. Hier ist es möglich, eine Methode für zwei unterschiedliche Limite zu schreiben. Wenn etwas an dieser Limite geändert werden soll, muss man dies nur einmal tätigen.

Aufgabe 2

Das Uhrenbeispiel soll erweitert werden, dass nebst Stunden und Minuten auch noch Sekunden verwaltet werden. Entsprechend soll die Methode `taktsignalGeben` der Klasse `Uhrenanzeige` nun dazu führen, dass die Zeit um eine Sekunde (und nicht wie bisher Minute) voranschreitet.

- a) Erweitern Sie das Projekt `03_Praktikum_ZeitanzeigeZweiKlassen`, damit diese Anforderungen erfüllt werden. Testen Sie dann in BlueJ, ob das erweiterte Programm funktioniert.

Welche Teile (Konstruktoren und Methoden) des bestehenden Codes mussten Sie ändern? Gibt es Teile, die Sie überhaupt nicht ändern mussten?

Geändert werden muss: Datenfeld, Konstruktoren, `setzeUhrzeit`, `anzeigeAktualisieren`, JavaDoc

`gibUhrzeit` und die Klasse `Nummernanzeige` muss nicht angepasst werden

- b) Betrachten Sie nun wiederum das Projekt `03_Praktikum_ZeitanzeigeEineKlasse`, setzen Sie die Erweiterungen auch hier um und testen Sie das Programm. Welche Teile (Konstruktoren und Methoden) mussten Sie hier ändern? Mussten Sie mehr oder weniger Codezeilen als oben hinzufügen bzw. ändern?

Geändert werden muss: Datenfeld, Konstruktoren, `taktsignalGeben`, `setzeUhrzeit`, `anzeigeAktualisieren`, JavaDoc

`gibUhrzeit` musste nicht angepasst werden.

Die Anpassungen waren komplexer vorzunehmen.

- c) Schliesslich möchte ein Informatiker die Anzeige noch so ändern, dass die einzelnen Zahlen hexadezimal dargestellt werden. Statt `19:47:58` soll also z.B. `13:2F:3A` angezeigt werden. Sie müssen dies nicht programmieren, aber überlegen Sie sich auch hier, was Sie in beiden Fällen jeweils anpassen müssten. In welchem Fall müssen Sie weniger Code ändern?

Mittels Modulo-Operator und der Zahl 16, könnte eine entsprechende Anpassung vorgenommen werden.

Des Weiteren wäre auch hier die Anpassung mit zwei Klassen einfacher, da die Anpassung nur auf Methoden vorgenommen werden kann bzw. Allgemein gehalten werden kann

- d) Basierend auf Ihren Erfahrungen in dieser Aufgabe, was sind Ihre Schlussfolgerungen bezüglich Modularisierung und Erweiterungen bzw. Änderungen des Programms?

Mit diesen Methoden ist einfacher im Nachhinein Codeanpassungen vorzunehmen, da diese dann für alle andere übernommen wird und nicht für jede einzelne Möglichkeit einzel angepasst werden muss.

Aufgabe 3

Forken Sie für diese Aufgabe das Projekt https://github.engineering.zhaw.ch/prog1-kurs/03_Praktikum_Eventverwaltung. Nutzen Sie BlueJ um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Sie sollen für einen Eventveranstalter eine Klasse `Event` entwickeln, mit welcher Events wie z.B. Konzerte verwaltet werden können. Die Anforderungen sind nachfolgend gegeben:

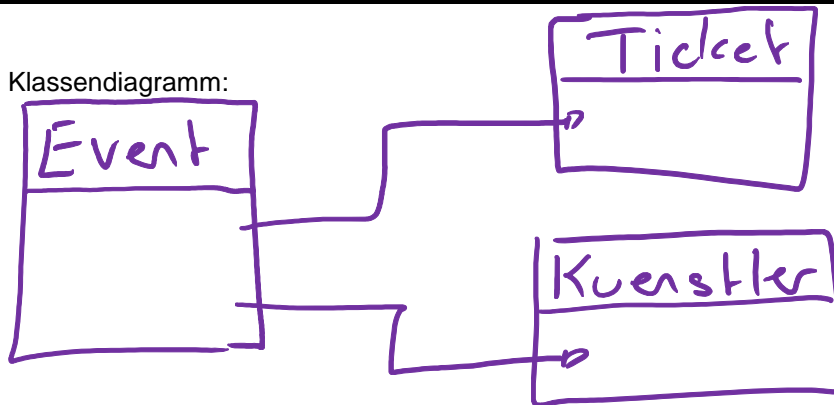
- Ein `Event` hat einen `Künstler`, der eine `Bezeichnung` (z.B. Elvis Presley) und eine `Gage` hat.
- Ein `Event` bietet immer drei `Ticketkategorien`, z.B. `VIP-`, `Tribünen-` und `Innenraumtickets`. Jede Kategorie hat eine `Bezeichnung`, einen `Preis pro Ticket` (immer in ganzen `Franken`) und eine `Anzahl der Tickets` in dieser Kategorie.
- Die Klasse `Event` bietet zwei Konstruktoren:
 - Einen Konstruktor ohne Parameter, der nichts tut.
 - Einen Konstruktor, der für jedes der Klasse `Event` verwendete Datenfeld einen entsprechenden Parameter enthält und mit welchen die Datenfelder gesetzt werden.
- Die Klasse bietet `Methoden`, um die Details des Künstlers und der `Ticketkategorien` gesetzt werden können, z.B. `setzteKuenstler(String bezeichnung, int gage)`. Diese Methoden machen insbesondere nach Verwendung des Konstruktors ohne Parameter Sinn und sollen bei Bedarf Objekte erzeugen. Für die Angabe der `Ticketkategorie` können Sie z.B. eine `Nummer (1-3)` verwenden, welche auf eine entsprechende Kategorie abgebildet wird.
- Es soll eine `Methode` geben, um `Tickets zu kaufen`. Dabei wird die gewünschte Kategorie und die Anzahl spezifiziert. Der Kauf ist nur möglich, wenn noch genügend Tickets in der Kategorie verfügbar sind und es soll in jedem Fall eine sinnvolle Meldung ausgegeben werden.
- Es sollen die wichtigsten Informationen zu einem Event ausgegeben werden können. Dies soll in etwa in der folgenden Art geschehen:

```
Kuenstler: Elvis Presley, Gage: CHF 85000
VIP-Tickets: 25 von 25 verkauft, Einnahmen: CHF 25000
Tribuene-Tickets: 721 von 1000 verkauft, Einnahmen: CHF 68495
Innenraum-Tickets: 327 von 500 verkauft, Einnahmen: CHF 16023
Gesamteinnahmen: CHF 109518
Gewinn: CHF 24518
```

- a) Überlegen Sie sich zuerst, wie Sie das Problem geeignet in mehrere Klassen aufteilen können (Modularisierung). Identifizieren Sie dabei Komponenten, die mehrere Male im Programm vorkommen, denn dies sind typischerweise gute Kandidaten für separate Klassen. Berücksichtigen Sie auch, dass es in der Zukunft neue Anforderungen geben könnte, die mit wenig Aufwand integrierbar sein sollten – z.B. ein Event mit mehreren Künstlern. Notieren Sie sich Ihre Gedanken und zeichnen Sie ebenfalls das Klassendiagramm.

Nach meinem Verständnis ist es sinnvoll mehrere Klassen zu haben. Ich würde drei Klassen erstellen (Event, Ticket, Künstler). Die Klasse Event greift auf die Klassen Ticket und Künstler zu.

Klassendiagramm:



- b) Implementieren und testen Sie die Klasse(n). Berücksichtigen Sie die Clean Code Regeln und prüfen Sie die übergebenen Parameter jeweils auf Korrektheit. Achten Sie auch auf ein sauberes Design, die folgenden Fragen helfen Ihnen, dies zu überprüfen:

- ✓ Haben Sie Methoden mit mehr als 20 Zeilen Code? Wenn ja, dann sollten Sie den Code geeignet auf mehrere Methoden aufteilen – Modularisierung ist auch innerhalb einer Klasse wichtig.
- ✓ Ist Ihr Code auch ohne Kommentare für Dritte gut verständlich?
- ✓ Nehmen Sie an, Sie müssten eine Ausgabe textuell ändern, z.B. die Ausgabe nach einem erfolgreichen Kauf. Müssen Sie dazu den Code nur an einem Ort anpassen? Wenn das nicht möglich ist, dann sollten Sie die Ausgabe in eine separate (private) Methode auslagern.
- Haben Sie keinen Code unnötigerweise dupliziert? Z.B. die Methoden um die Details einer Ticketkategorie zu setzen oder um Tickets zu kaufen. Sollten sich dort Codebereiche wiederholen, so versuchen Sie, die mehrfach vorhandenen Bereiche in eine separate Methode auszulagern.

c) Zeichnen Sie ein Objektdiagramm zur Laufzeit, wenn alle Objekte erzeugt sind.

Objektdiagramm:

