Einführung in die Programmierung für NF MI

Übung 12

Inhalt

- Exceptions
- Die Spezialklasse „Enum“
- Wiederholung zur Klausur
  - Interfaces
  - UML
  - Oberserver Pattern

Exceptions

- Exceptions dienen der Fehlererkennung und Behandlung
- Während der Laufzeit eines Programms können in bestimmten Fällen Fehler passieren
- Wenn diese Fehler nicht behandelt werden, stürzt das Programm ab
- Java wirft, wenn ein solcher Fehler auftritt, eine Exception

Exceptions

- Eine der bekanntesten Exceptions:

  die NullPointerException

- Die NullPointerException wird immer dann geworfen, wenn etwas verwendet werden soll, dass es nicht gibt, also das null ist

Exceptions

- Beispiel:

```
public class NullPointerTest {

    private static Objekt Objekt;

    public static void main(String[] args) {
        System.out.println(Objekt.toString());
    }
}
```

Das Objekt wurde an der Stelle, an der es ausgegeben werden soll, noch nicht erzeugt, das Programm bricht ab

```
Exception in thread "main" java.lang.NullPointerException
        at NullPointerTest.main(NullPointerTest.java:9)
```

Exceptions

- Wenn wir vermuten, dass hier eine NullPointerException auftreten könnte, können wir diese abfangen:

```
public class NullPointerTest {

    private static Objekt Objekt;

    public static void main(String[] args) {
        try {
            System.out.println(Objekt.toString());
        } catch (NullPointerException e) {
            System.out.println("gibts net!");
        }
    }
}
```

Exceptions

- Es gibt viele Exceptions, besonders bei Netzwerkverbindungen oder wenn Dinge ineinander umgewandelt werden sollen
- Viele solche Dinge müssen mit einem try-catch-Block versehen werden
- Die Exception wird im Catch-Block mit übergeben und kann verwendet werden, z.B. zum Auslesen einer Fehlermeldung

Exceptions

- Außerdem können eigene Exceptions wie andere Klassen auch erstellt werden
- Sie erben dann von der Bibliotheksklasse Exception

```
class MyException extends Exception
{
    public MyException(){
        // Aufruf von Exception mit Fehlertext
        super("Ein Fehler ist aufgetreten");
    }
}
```

Enum

- Ein Enum-Typ (von „Enumeration“) ist ein spezieller Datentyp für eine Variable, die eine vordefinierte Menge an Werten haben kann
- Diese Werte müssen fix definiert werden
- Solche Datentypen werden sehr häufig verwendet, weil in vielen Fällen Variablen nur eine bekannte Menge von Werten einnehmen können

Enum

- Beispiel: Ein Datentyp für Wochentage
- Mit Grunddatentypen ist nur eine Kodierung möglich, z.B. 0 für Montag und 7 für Sonntag
- Dies ist aber sehr unübersichtlich, kaum lesbar und sehr fehleranfällig
- Besser also: ein Datentyp mit den Werten
  MONTAG, DIENSTAG, MITTWOCH,
  DONNERSTAG, FREITAG, SAMSTAG, SONNTAG

Enum in Java

- Beispiel:

```
public enum Wochentag {
    MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG, FREITAG, SAMSTAG, SONNTAG
}
```

… ja, das ist alles ;)

- Nun gibt es einen Datentyp „Wochentag“, der die definierten Werte einnehmen kann
- Diese werden üblicherweise groß geschrieben

Enum in Java

- Verwendung:

```
public class EnumTest {

    Wochentag tag1;
    Wochentag tag2;

    public EnumTest(){
        tag1 = Wochentag.MONTAG;
        tag2 = Wochentag.SONNTAG;
    }

}
```

- Der Datentyp wird also wie üblich verwendet
- Die Werte werden mit Punkt-Notation dargestellt: EnumName.WERT

Enum in Java

- Ausnahme: In switch-case-Blöcken werden die Werte ohne Punktnotation dargestellt
- Das erhöht zusätzlich die Lesbarkeit

```
public EnumTest() {
    Wochentag tag = Wochentag.MONTAG;

    switch (tag) {
    case MONTAG:
        System.out.println("Montage sind doof");
        break;

    case FREITAG:
        System.out.println("Freitage sind besser");
        break;

    case SAMSTAG:
    case SONNTAG:
        System.out.println("Wochenende ist toll");
        break;

    default:
        break;
    }

}
```

Enum in Java

- Aufgabe: Schreiben Sie einen Enum-Typ für die Speicherung von Belegungen von Spielfeldern in einem Schachspiel.
- Folgende Belegungen sind möglich:
  - Leeres Feld
  - Alle Schachfiguren in weiß
  - Alle Schachfiguren in schwarz

Enum in Java

- Enum-Typen können auch mehr enthalten, z.B. weitere Daten pro Wert oder spezielle Funktionen, die mit diesen Werten rechnen
- Im Rahmen dieser Vorlesung genügt jedoch die Kenntnis über den Enum-Typ und seinen praktischen Nutzen

WH: Interfaces

- Nachdem wir bereits Interfaces geschrieben haben und Klassen diese importieren haben lassen, stellt sich die Frage: wozu?
- Ein Grund für Interfaces ist die Austauschbarkeit der dahinter stehenden Klasse in anderen Klassen
- Zum Beispiel: Das Model wird im Controller durch ein anderes Model ersetzt

WH: Interfaces

- Im Code des Controllers ersetzen wir dazu die folgende Zeile

```
public class Controller {

    public Controller(){

        Model model = new Model();
        View view = new View();
        model.addObserver(view);

    }
}
```

durch

```
public class Controller {

    public Controller(){

        IModel model = new Model();
        View view = new View();



    }
}
```

- Nun sind nur noch die Methoden aus dem Interface bekannt

WH: Interfaces

- Im Code des Controllers ersetzen wir dazu die folgende Zeile

```
public class Controller {

    public Controller(){

        Model model = new Model();
        View view = new View();
        model.addObserver(view);

    }
}
```

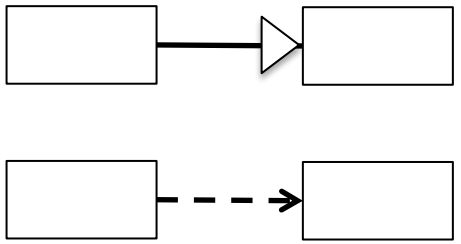
durch

```
public class Controller {

    public Controller(){

        IModel model = new Model();
        View view = new View();



    }
}
```

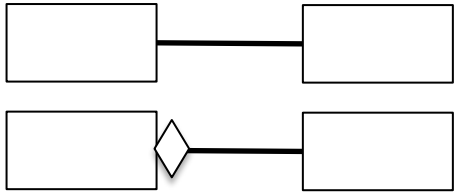
- Nun sind nur noch die Methoden aus dem Interface bekannt

WH: UML

- UML
  - Vererbung von Klassen
  - Implementierung von Interfaces



- Beziehungen
  - Assoziation
  - Aggregation



WH: Observer Pattern

- Observer „überwachen“ ein Observable
- Das Observer-Interface implementiert eine `update()`-Methode im Observer
- Diese Methode wird immer ausgeführt, wenn die Observer benachrichtigt werden:

```
setChanged();
notifyObservers();
```

- Dies geschieht im Observable

Fragen zum Übungsblatt?