Programmiertechnik 1

Übungen

Sie finden in diesem Dokument Anleitungen zum Bearbeiten der Übungsaufgaben.

Um den unbenoteten Leistungsnachweis für die Übungen zu bestehen, müssen Sie die Aufgaben 1 bis 6 erfolgreich bearbeiten und dabei eine Mindestpunktzahl von 18 Punkten erreichen.

Sie bekommen pro Aufgabe
4 Punkte, wenn Sie eine Woche vor dem Abgabetermin fertig werden
3 Punkte, wenn Sie am Abgabetermin fertig werden
2 Punkte, wenn Sie zu einem vorab vereinbarten späteren Termin fertig werden
1 Punkt, wenn Sie ohne rechtzeitige Vereinbarung verspätet fertig werden

Eine Vereinbarung für einen späteren Abgabetermin ist nur aus wichtigem Grund möglich.

Die Punkte einer Aufgabe werden nur gutgeschrieben, wenn Sie jeweils zeitgleich ein ausgefülltes Teilnahmeprotokoll vorlegen (siehe Einstiegsaufgabe).

Durch die Bearbeitung von freiwilligen Zusatzaufgaben können Sie Bonuspunkte erwerben, mit denen Sie bei Bedarf Ihr Punktekonto ausgleichen können. Trotz Bonuspunkten muss aber bei jeder Aufgabe im Pflichtteil mindestens 1 Punkt erreicht sein.

Die Anleitungen zu den Übungen sind für den Präsenzbetrieb in einem Labor der Hochschule geschrieben. Sinngemäß funktioniert das aber so auch auf eigenen PCs unter Linux, Windows oder macOS. Hinweise zum Einrichten Ihres PCs finden Sie unter dem Link "Programmierwerkzeuge" auf der Webseite der Lehrveranstaltung. Lassen Sie sich gegebenenfalls helfen.

Das jeweils unter "Abgabe" verlangte Vorführen Ihrer Lösung im Labor wird im Falle eines erneuten Onlinebetriebs durch das Hochladen in Moodle ersetzt (siehe jeweils die Hinweise im Moodle-Kurs).

Hinweise zum Download der Übungen

Arbeitsverzeichnis

Legen Sie sich für die Übungen Programmiertechnik 1 auf dem Z-Laufwerk ein eigenes Arbeitsverzeichnis an, z.B. mit Name `prog1-uebungen` (Name ist frei wählbar):

```
Z:\prog1-uebungen\
```

Dieses von Ihnen angelegte Verzeichnis ist immer gemeint, wenn in den einzelnen Übungsaufgaben vom Arbeitsverzeichnis die Rede ist. Dieses Arbeitsverzeichnis geben als Ziel beim Download der Übungsaufgaben an und in dieses Arbeitsverzeichnis hinein entpacken Sie die heruntergeladenen Archive.

Hinweis: Falls Sie statt eines Laborrechners Ihren eigenen Rechner nutzen, können Sie das Arbeitsverzeichnis irgendwo auf ihr C-Laufwerk legen: `C:\irgendwo...\prog1-uebungen\`

Download und Entpacken

Zu jeder Übungsaufgabe gibt es vorgegebene Dateien, die Sie als `.jar`-Archive herunterladen können. Speichern Sie die Archive immer im Arbeitsverzeichnis und entpacken Sie sie mit dem Konsolenbefehl

```
jar xf Name.jar
```

Vermeiden Sie das Entpacken per Kontextmenü (rechte Maustaste). Wenn Sie dabei den falschen Menüpunkt auswählen, werden zusätzliche Verzeichnisebenen angelegt. Die Anleitungen zu den Übungsaufgaben passen dann nicht damit zusammen.

Für jede Übungsaufgabe wird so im Verlauf des Semesters beim Entpacken des zugehörigen Archivs im Arbeitsverzeichnis ein weiteres Unterverzeichnis angelegt:

```
Z:\prog1-uebungen\
                 |
                 |- einstieg\
                 |- aufgabe1\
                 |- aufgabe2\
                 |- aufgabe3\
                 |- aufgabe4\
                 |- aufgabe5\
                 |- aufgabe6\
```

Beispiel: Download und Entpacken der Einstiegsaufgabe

Auf der Webseite der Lehrveranstaltung finden Sie ein Archiv `Einstieg.jar` zum Download. Legen Sie das Archiv beim Download in Ihrem Arbeitsverzeichnis ab:

```
Z:\prog1-uebungen\
                 |- Einstieg.jar
```

Nach dem Entpacken des Archivs `Einstieg.jar` der Einstiegsaufgabe sollte Ihr Arbeitsverzeichnis folgenden Inhalt haben (wenn Sie etwas anderes sehen, haben Sie beim Download oder Entpacken etwas falsch gemacht):

```
Z:\prog1-uebungen\
                 |
                 |- Einstieg.jar
                 |- Teilnahmeprotokoll.html
                 |- Teilnahmeprotokoll.pdf
                 |- Lesezeichen.html
                 |- build.xml
                 |- exclude_filter.xml
                 |- suppressions.xml
                 |- einstieg\
                    |
                    |- package-info.java
```

Beim Bearbeiten der Einstiegsaufgabe kommen dann noch weitere Dateien hinzu:

```
Z:\prog1-uebungen\
    ├─ Einstieg.jar
    ├─ Teilnahmeprotokoll.html
    ├─ Teilnahmeprotokoll.pdf
    ├─ Lesezeichen.html
    ├─ build.xml
    ├─ exclude_filter.xml
    ├─ suppressions.xml
    ├─ doc\
    │   ├─ ...
    │   ├─ index.html
    │   └─ ...
    └─ einstieg\
        ├─ package-info.java
        ├─ Einstieg.java
        └─ Einstieg.class
```

Vorlesungsbeispiele

Zur Vorbereitung der einzelnen Übungsaufgaben sollten Sie die jeweils relevanten Programmierbeispiele aus der Vorlesung nachvollziehen. Legen Sie die Vorlesungsbeispiele und sonstige eigene Beispielprogramme nicht im Arbeitsverzeichnis der Übungen ab, sondern legen Sie dafür ein eigenes Verzeichnis mit Name `prog1-beispiele` an und darin für jedes Kapitel ein Unterverzeichnis:

```
Z:\prog1-beispiele\
    ├─ Programme2\
    ├─ Programme3\
    ├─ Programme4\
    ├─ Programme5\
    └─ Programme6\
```

Einstieg: Kennenlernen der Programmierwerkzeuge im Labor

Die Einstiegsaufgabe soll in der ersten Übungsstunde im Labor bearbeitet werden.

Download und Entpacken des Archivs **`Einstieg.jar`** sind im vorhergehenden Abschnitt Hinweise erklärt. Sie finden danach folgende neue Dateien in Ihrem Arbeitsverzeichnis:

`Teilnahmeprotokoll.html` **`Teilnahmeprotokoll.pdf`**	im Semester laufend auszufüllen und bei den Abgaben zu zeigen
`Lesezeichen.html`	erleichtert das Einrichten der Arbeitsumgebung
`build.xml`	Hilfsdatei für das Werkzeug **`ant`**
`suppressions.xml`	Hilfsdateien für das Werkzeug **`checkstyle`**
`exclude_filter.xml`	Hilfsdatei für das Werkzeug **`spotbugs`**
`einstieg\package-info.java`	Unterverzeichnis für Java-Quellcode, darin eine Datei mit Paketdokumentation

Schritt 1:

- Starten Sie einen Texteditor, etwa `VSCode` oder `notepad++`.
- Geben Sie im Editor das auf der nächsten Seite stehende Programm ein.
- Beenden Sie den Editor und speichern Sie dabei Ihr Programm als **`Z:\prog1-uebungen\einstieg\Einstieg.java`**.

Schritt 2:

- Starten Sie ein Terminal und wechseln Sie darin in das Unterverzeichnis `einstieg\`:
  `cd Z:\prog1-uebungen\einstieg`

Schritt 3:

- Übersetzen Sie Ihr Programm im Terminal mit dem Befehl
  `javac Einstieg.java`
- Wechseln Sie mit dem `cd`-Befehl in das Oberverzeichnis von `einstieg\`, also in das Arbeitsverzeichnis (auf die Leerstelle zwischen `cd` und `..` achten):
  `cd ..`
- Lassen Sie Ihr Programm laufen mit dem Befehl
  `java einstieg.Einstieg`

  Das Programm fordert Sie auf, Ihren Vornamen sowie die Anzahl Ihrer bisher geschriebenen Java-Programme einzugeben, und gibt anschließend einen Text aus.

Schritt 4:

- Prüfen Sie mit dem Werkzeug checkstyle, ob Sie in Ihrem Programm Stilregeln verletzt haben. Sie können checkstyle über das Automatisierungswerkzeug ant aufrufen:
  `ant -Dpackage=einstieg style`

  Auf Ihrem eigenen Rechner müssen Sie die Datei build.xml angepasst haben, damit das funktioniert (siehe die Installationsanleitungen auf der Webseite der Lehrveranstaltung).
- Bessern Sie nach, falls checkstyle Stilverletzungen meldet, und gehen Sie zurück zu Schritt 3.

Schritt 5:

- Erzeugen Sie HTML-Dokumentation zum Programm mit dem Befehl:
  `javadoc -html5 -d doc -linksource -author -version einstieg`
- Betrachten Sie die Dokumentation:

  *Im Dateimanager im neuen Unterverzeichnis **`doc\`** die Datei **`index.html`** anklicken und dann die Links auf der angezeigten Seite weiterverfolgen,*
  *oder die Datei **`Lesezeichen.html`** im Arbeitsverzeichnis als Einstieg verwenden.*

  Untersuchen Sie, welche Texte aus den `.java`-Dateien wo in den HTML-Seiten stehen.

Das einzugebende Java-Programm:

```
// Einstieg.java
package einstieg;

import java.util.Scanner;

/**
 * Einstieg ist ein kleines Java-Programm.
 * Es verwendet die Klassen Scanner und System aus der Java-Bibliothek.
 * @author TODO: Name eintragen und TODO inklusive Doppelpunkt löschen
 * @version TODO: Datum eintragen und TODO inklusive Doppelpunkt löschen
 */
public final class Einstieg {
    private Einstieg() { }

    private static final Scanner EINGABE = new Scanner(System.in);

    /**
     * Die Klassenmethode main ist der Startpunkt des Programms.
     * main verwendet die Methoden print und printf zum Ausgeben von Text
     * sowie die Methoden next und nextInt zum Einlesen von Text
     * und einer ganzen Zahl.
     * @param args wird nicht verwendet
     */
    public static void main(String[] args) {
        System.out.print("Vorname: ");
        String vorname = EINGABE.next();

        System.out.print("Anzahl bisher geschriebener Java-Programme: ");
        int anzahl = EINGABE.nextInt();

        System.out.printf("%ss %d. Java-Programm funktioniert!%n",
                          vorname, anzahl + 1);
    }
}
```

Übungsaufgabe 1: Ganze Zahlen

Als Voraussetzung für diese Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 2-17 nachbereitet und die zugehörigen Programmbeispiele nachvollzogen haben.
Ihr Unterverzeichnis `prog1-beispiele\Programme2\` sollte danach mindestens folgende Programme enthalten: `IntLiteral.java`, `DoubleLiteral.java`, `CharLiteral.java`, `BooleanVar.java`.

Nach dem Download und dem Entpacken von `Aufgabe1.jar` sollten in Ihrem Arbeitsverzeichnis `prog1-uebungen\` folgende Dateien hinzugekommen sein:

`aufgabe1\package-info.java`	Unterverzeichnis für Java-Quellcode,
`aufgabe1\IntVar.java`	darin die Paketdokumentation und ein noch unvollständiges Java-Programm

Schritt 1: Vorbereitung

Aktualisieren Sie Ihre Javadoc-Seiten:

```
ant doc
```

Öffnen Sie im Browser alle Lesezeichen aus `Lesezeichen.html`.

Lesen Sie die Spezifikation des zu erstellenden Programms auf der Javadoc-Seite des Pakets `aufgabe1`. Die Javadoc-Seite finden Sie im Browser-Tab "*Overview (Programmiertechnik 1)*" oder indem Sie die Datei `doc\index.html` mit dem Browser öffnen.

Schritt 2: Programmierung

Das Programm `aufgabe1\IntVar.java` ist noch unvollständig. Sie sollen den fehlenden Code an den mit `TODO:` markierten Stellen ergänzen:

- Die Konstante `max` soll mit der größten als `int` darstellbaren Zahl initialisiert werden.
  Das geht am einfachsten, wenn Sie das entsprechende Literal hexadezimal schreiben.
  Das gleiche gilt für die kleinste darstellbare Zahl bei der Konstanten `min`.
- Wie die Eingabeaufforderung aussehen soll, können Sie dem Beispiel auf der Javadoc-Seite des Pakets `aufgabe1` entnehmen. Halten Sie sich genau an diese Vorgabe.
- Wie Sie ganze Zahlen einlesen, können Sie dem Programm `Einstieg` aus der ersten Übungsstunde entnehmen.
- Was die übrigen Ausgaben des Programms sind, können Sie wieder dem Beispiel auf der Javadoc-Seite des Pakets `aufgabe1` entnehmen. Halten Sie sich genau an diese Vorgabe.
  Verwenden Sie am besten die formatierte Ausgabe mit `System.out.printf`.
  Beispiele dazu finden Sie in `IntLiteral.java` aus der Vorlesung und in `Einstieg.java` aus der ersten Übung.

Löschen Sie nach getaner Arbeit die TODO: Markierungen aus dem Code.

Schritt 3: Test und Qualitätssicherung

- Testen Sie Ihr Programm systematisch mit verschiedenen Eingaben und prüfen Sie dabei, ob die arithmetischen Operationen und Vergleiche immer die erwarteten Ergebnisse liefern.
  Was passiert bei sehr großen bzw. kleinen Zahlen oder Zahlen mit verschiedenen Vorzeichen?
  Was passiert, wenn Sie in `int` nicht darstellbare Zahlen eingeben?
  Was passiert, wenn Sie Text statt Zahlen eingeben?
  Was passiert, wenn Sie ein Eingabeende `STRG-Z` statt Zahlen eingeben? (Linux: `Strg-D`)

Prüfen Sie mit checkstyle, ob Sie Stilregeln verletzt haben. Bessern Sie nach, solange das Werkzeug Fehler meldet. Rufen Sie checkstyle im Arbeitsverzeichnis über ant auf:

```
ant -Dpackage=aufgabe1 style
```

- Prüfen Sie mit spotbugs, ob Ihr Programm problematischen Code enthält. Bessern Sie nach, solange das Werkzeug Probleme meldet. Rufen Sie spotbugs wie folgt über ant auf:

  ```
  ant -Dpackage=aufgabe1 clean bugs
  ```

  spotbugs schreibt seine Meldungen nicht auf den Bildschirm, sondern in eine Datei `bugs.html`, die Sie mit einem Webbrowser anschauen müssen.

Schritt 4: Abgabe

Den spätesten Abgabetermin finden Sie auf der Webseite der Lehrveranstaltung.

Geben Sie bitte erst ab, nachdem Sie Schritt 3 erledigt haben.
Die Werkzeuge checkstyle und spotbugs dürfen keine Fehler mehr melden!

- Führen Sie einige besonders interessante Testfälle vor.
- Sie müssen die Fragen aus Schritt 3 beantworten können.
- Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Ergänzende Übungen für Schnellprogrammierer *(freiwillig, pro Spiegelpunkt 1 Bonuspunkt)*

Wenn Sie den Pflichtteil von Übungsaufgabe 1 zügig lösen konnten, hier einige Anregungen für Erweiterungen Ihrer Lösung. Legen Sie für die Bonusaufgaben eine Unterverzeichnis `bonus\` in `aufgabe1\` an und ordnen Sie die Programme dem package `aufgabe1.bonus` zu.
Beim ant-Aufruf geben Sie dann **`-Dpackage=aufgabe1\bonus`** an.

- Schreiben Sie zu `IntVar` analoge Programme `ByteVar` und `ShortVar`, die mit Variablen und Eingaben vom Typ `byte` bzw. `short` statt `int` arbeiten. Testen Sie die Programme wie im Pflichtteil beschrieben. Beantworten Sie insbesondere wieder die Fragen aus Schritt 3. Kommt es hier auch beim Rechen mit großen Zahlen zu Überläufen im Ergebenis?
- Verbessern Sie das Verhalten Ihres Programms `IntVar` für den Fall, dass der Anwender keine gültigen Zahlen eingibt, sondern Text oder Eingabeende. Schlagen Sie dazu auf der Javadoc-Seite der Klasse `java.util.Scanner` nach, wie Sie prüfen können, ob der Anwender eine gültige Zahl eingegeben hat.
- Probieren Sie nicht immer alle arithmetischen und Vergleichsoperatoren aus, sondern lesen Sie den gewünschten Operator zusätzlich zu den beiden ganzen Zahlen als String ein. Schreiben Sie dann eine Fallunterscheidung für den eingelesenen String und geben Sie nur das Ergebnis der verlangten Operation aus.
- Legen Sie in Ihrem Programm `IntVar` die eingegebenen Zahlen (weiterhin aus dem Zahlbereich von `int`) in Variablen vom Typ `long` ab, damit die arithmetischen Operationen mit long gerechnet werden. Speichern Sie das Ergebnis der arithmetischen Operationen in einer weiteren Variablen vom Typ `long` . Prüfen Sie mit `if`-Anweisungen ob, das Rechenergebnis in `int` darstellbar ist, und geben Sie gegebenenfalls eine Fehlermeldung statt des Rechenergebnisses aus.

Übungsaufgabe 2: Histogramm

Als Voraussetzung für diese Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 2-26 nachbereitet und die zugehörigen Programmbeispiele nachvollzogen haben.
Ihr Unterverzeichnis `prog1-beispiele\Programme2\` sollte danach mindestens folgende Programme enthalten: `StringVar.java, Klausur.java, KlausurArray.java, Bubblesort.java, ArrayVar.java.`

Nach dem Download und dem Entpacken von `Aufgabe2.jar` sollten in Ihrem Arbeitsverzeichnis folgende Dateien hinzugekommen sein:

`aufgabe2\package-info.java`	Unterverzeichnis für Java-Quellcode,
`aufgabe2\Histogramm.java`	darin die Paketdokumentation und
`aufgabe2\Wuerfel.java`	das noch unvollständige Java-Programm

Schritt 1: Vorbereitung

- Aktualisieren Sie Ihre Javadoc-Seiten:
  `ant doc`

  Öffnen Sie im Browser alle Lesezeichen aus `Lesezeichen.html`.

  Lesen Sie die Spezifikation des zu erstellenden Programms auf der Javadoc-Seite des Pakets `aufgabe2`. Die Javadoc-Seite finden Sie im Browser-Tab *"Overview (Programmiertechnik 1)"* oder indem Sie die Datei `doc\index.html` mit dem Browser öffnen.
- Übersetzen Sie die Datei Wuerfel.java mit
  `javac aufgabe2\Wuerfel.java`

  Wuerfel ist ein Testprogramm für Schritt 3. Wenn Sie Interesse haben, können Sie sich den Quellcode Wuerfel.java ansehen, aber zum erfolgreichen Bearbeiten der Übungsaufgabe ist das nicht erforderlich.

Schritt 2: Programmierung

Das Programm `aufgabe2\Histogramm.java` ist noch unvollständig.
Sie sollen den fehlenden Code an den mit `TODO` markierten Stellen ergänzen:

- Direkt nach dem `TODO`-Kommentar mit der Nummer (1):

  Sie sollen für jede der Zahlen 1 bis 6 zählen, wie oft sie eingegeben wurden. Dafür brauchen Sie sechs Zähler. Realisieren Sie diese sechs Zähler als Elemente eines `int`-Arrays der Länge 6. Definieren und initialisieren Sie die Array-Variable hier vor der `while`-Schleife.
- Direkt nach dem `TODO`-Kommentar mit der Nummer (2):

  Verwenden Sie hier innerhalb der `while`-Schleife eine `if-else`-Anweisung für die Unterscheidung, ob die eingegebene Zahl innerhalb oder außerhalb des Bereichs von 1 bis 6 liegt. Greifen Sie bei Zahlen zwischen 1 und 6 mit dem Index-Operator auf den richtigen Zähler zu.
- Direkt nach dem `TODO`-Kommentar mit der Nummer (3):

  Geben Sie hier hinter der `while`-Schleife die Zeilen des Histogramms in einer `for`-Schleife über das Zählerfeld (das `int`-Array) aus. Innerhalb dieser `for`-Schleife geben Sie den Histogrammbalken in einer weiteren `for`-Schleife über den jeweiligen Zählerstand aus.

  Beispiele für `for`-Schleifen finden Sie auf den Folien 2-24 und 2-26 der Vorlesung.

Schritt 3: Test und Qualitätssicherung

- Testen Sie Ihr Programm mit verschiedenen manuellen Eingaben
  ```
  javac aufgabe2\Histogramm.java
  java aufgabe2.Histogramm
  ```
  Liefert Ihr Programm exakt die verlangten Ausgaben?
- Testen Sie Ihr Programm mit automatisch erzeugten Eingaben.
  Verwenden Sie dazu das in Schritt 1 erzeugte Programm Wuerfel.
  Starten Sie die Programme Wuerfel und Histogramm gemeinsam mit dem Kommando
  ```
  java aufgabe2.Wuerfel 100 | java aufgabe2.Histogramm
  ```
  Die Ausgabe von `aufgabe2.Wuerfel` (100 Zufallszahlen) wird dann über eine so genannte "Pipe" (geschrieben als senkrechter Strich) als Eingabe an `aufgabe2.Histogramm` weitergegeben.
- Prüfen Sie mit checkstyle, ob Sie Stilregeln verletzt haben. Bessern Sie nach, solange das Werkzeug Fehler meldet. Rufen Sie checkstyle im Arbeitsverzeichnis über ant auf:
  ```
  ant -Dpackage=aufgabe2 style
  ```
- Prüfen Sie mit spotbugs, ob Ihr Programm problematischen Code enthält. Bessern Sie nach, solange das Werkzeug Probleme meldet. Rufen Sie spotbugs wie folgt über ant auf:
  ```
  ant -Dpackage=aufgabe2 clean bugs
  ```
  spotbugs schreibt seine Meldungen nicht auf den Bildschirm, sondern in eine Datei `bugs.html`, die Sie mit einem Webbrowser anschauen müssen.

Schritt 4: Abgabe

Den spätesten Abgabetermin finden Sie auf der Webseite der Lehrveranstaltung.

Geben Sie bitte erst ab, nachdem Sie Schritt 3 erledigt haben.
Die Werkzeuge checkstyle und spotbugs dürfen keine Fehler mehr melden!

- Führen Sie einen Test mit manueller Eingabe vor.
- Führen Sie einen Test mit automatischer Eingabe vor.
- Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Ergänzende Übungen für Schnellprogrammierer

finden Sie auf der nächsten Seite ...

Ergänzende Übungen für Schnellprogrammierer (freiwillig)

Wenn Sie **`aufgabe2.Wuerfel`** sehr viele Zahlen ausgeben lassen (z.B. 1000 statt 100), wird das Histogramm sehr unansehnlich, weil die Balken über mehrere Zeilen gehen. Dies lässt sich durch Skalieren und/oder Zuschneiden verhindern oder durch eine andere Ausrichtung. Legen Sie für die Bonusaufgaben einer Unterpaket `aufgabe2.bonus` an (vergleiche Aufgabe 1)

- Lineare Skalierung *(1 Bonuspunkt)*:
  Geben Sie bei den Histogrammbalken nur jedes n-te (zweite, dritte, ...) Zeichen aus, so dass maximal 50 Zeichen nebeneinander stehen. Berechnen Sie die erforderliche Skalierung `n` aus dem größten vorkommenden Zählerstand `max` mit der Formel `n = max / 50 + 1` und verwenden Sie `n` als Schrittweite in der inneren `for`-Schleife, die die Histogrammbalken ausgibt.
- Zuschneiden *(1 Bonuspunkt)*:
  Lassen Sie bei den Histogrammbalken alle Zeichen vor dem kleinsten vorkommenden Zählerstand weg. Die Häufigkeitsunterschiede sehen dann besonders spektakulär aus, weshalb diese Methode bei Diagrammen in der Presse sehr beliebt ist. Beispiel:

```
1111111111 (10)                          11 (10)
2222222222222 (13)                       22222 (13)
333333333333 (12)      wird dargestellt als:   3333 (12)
444444444 (9)                            4 (9)
5555555555555555555 (18)                 5555555555 (18)
66666666666 (11)                         666 (11)
```

  Verwenden Sie den kleinsten vorkommenden Zählerstand als Startwert der inneren `for`-Schleife, die die Histogrammbalken ausgibt. Sie müssen das Zuschneiden mit der linearen Skalierung kombinieren, wenn die Zählerstände sehr unterschiedlich groß werden.
- Ausrichtung *(1 Bonuspunkt)*:
  Geben Sie das Histogramm von oben nach unten statt von links nach rechts aus. Das Beispiel aus der Paketdokumentation könnte dann wie folgt aussehen:

```
Ganze Zahlen zwischen 1 und 6 eingeben (Ende mit Strg-D/Strg-Z):
1 1 2 2 2 3 3 3 3 4 4 5 5 5 6 7
Falsche Eingabe wird ignoriert: 7
3 5 5 5 5 5 5 5 5 6
Histogramm:
1 2 3 4 5 6  (1)
1 2 3 4 5 6  (2)
  2 3   5    (3)
    3   5    (4)
    3   5    (5)
        5    (6)
        5    (7)
        5    (8)
        5    (9)
        5    (10)
        5    (11)
```

Übungsaufgabe 3: Notenstatistik

Als Voraussetzung für diese Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 3-20 nachbereitet und die zugehörigen Programmbeispiele nachvollzogen haben.
Ihr Unterverzeichnis `prog1-beispiele\Programme3\` sollte danach mindestens folgende Programme enthalten: `Verzweigung.java, Fallunterscheidung.java.`

Nach dem Download und dem Entpacken von `Aufgabe3.jar` sollten in Ihrem Arbeitsverzeichnis folgende Dateien hinzugekommen sein:

`aufgabe3-in1.txt,aufgabe3-out1.txt` **`aufgabe3-in2.txt,aufgabe3-out2.txt`** **`aufgabe3-in3.txt,aufgabe3-out3.txt`**	Dateien mit Eingaben und Sollausgaben für das automatisierte Testen in Schritt 3
`aufgabe3\package-info.java` **`aufgabe3\Notenstatistik.java`**	Unterverzeichnis für Java-Quellcode, darin die Paketdokumentation und das noch unvollständige Java-Programm

Schritt 1: Vorbereitung

Aktualisieren Sie Ihre Javadoc-Seiten:

```
ant doc
```

Öffnen Sie im Browser alle Lesezeichen aus `Lesezeichen.html`.

Lesen Sie die Spezifikation des zu erstellenden Programms auf der Javadoc-Seite des Pakets `aufgabe3`. Die Javadoc-Seite finden Sie im Browser-Tab "*Overview (Programmiertechnik 1)*" oder indem Sie die Datei `doc\index.html` mit dem Browser öffnen.

Schritt 2: Programmierung

Das Programm `aufgabe3\Notenstatistik.java` ist noch unvollständig. Sie sollen den fehlenden Code gemäß der vorgegebenen Gliederung an den mit TODO markierten Stellen ergänzen:

- Überlegen Sie bei jedem der folgenden TODOs, welche zusätzlichen Variablen Sie brauchen, mit welchem Gültigkeitsbereich (Scope) Sie diese Variablen definieren müssen, welchen Typ sie haben müssen und mit welchen Werten sie am besten initialisiert werden.

  Definieren Sie Variablen immer mit dem kleinst möglichen Scope, d.h. so spät wie möglich!
- TODO-Kommentar mit der Nummer (1): Sie müssen das Format des eingegebenen Strings prüfen (Länge 3, erst Ziffer, dann Punkt oder Komma, am Ende wieder Ziffer) und bei richtigem Format zusätzlich den Zahlbereich von Vor- und Nachkommateil (siehe die drei Arten von Fehlermeldungen in den Javadoc-Seiten aus Schritt 1).

  Verwenden Sie sowohl `if-else`- als auch `switch-case`-Anweisungen. Letzteres bieten sich vor allem für die Vorkommaziffern an. Bauen Sie die Fallunterscheidungen so auf, dass Sie möglichst wenige Wiederholungen immer gleicher Anweisungen haben.

  Auf die Zeichen können Sie mit `note.charAt(index)` zugreifen. Ob ein Zeichen eine Ziffer ist, können Sie mit `Character.isDigit(zeichen)` prüfen.
- TODO-Kommentar mit der Nummer (2): Überlegen Sie, wie Sie aus den Ziffern im String eine Note als Gleitkommazahl erzeugen können.
- TODO-Kommentar mit der Nummer (3): Achten Sie darauf, dass Sie zeichengenau die verlangten Ausgaben liefern.

  Verwenden Sie bei `printf` für Gleitkommazahlen die Formatangabe `%.1f`, um die Darstellung mit einer Nachkommastelle zu erreichen.

Schritt 3: Test und Qualitätssicherung

- Testen Sie Ihr Programm mit verschiedenen manuellen Eingaben
  ```
  javac aufgabe3\Notenstatistik.java
  java aufgabe3.Notenstatistik
  ```
  Liefert Ihr Programm exakt die verlangten Ausgaben?
- Testen Sie das Programm mit den drei vorgegebenen Eingaben aus dem entpackten Archiv. Verwenden Sie die folgenden Kommandos:
  ```
  java aufgabe3.Notenstatistik < aufgabe3-in1.txt > out1.txt
  java aufgabe3.Notenstatistik < aufgabe3-in2.txt > out2.txt
  java aufgabe3.Notenstatistik < aufgabe3-in3.txt > out3.txt
  ```
  Ihr Programm liest dank des Operators < aus der angegebenen Datei statt von Tastatur und schreibt dank des Operators > in die angegebene Datei statt auf das Konsolenfenster.

  Achtung: In der Windows powershell gibt es die Operatoren nicht. Verwenden Sie unter Windows stattdessen cmd (Eingabeaaufforderung) oder die Git Bash.
- Vergleichen Sie die Ausgaben Ihres Programms mit den vorgegebenen Soll-Ausgaben, die ebenfalls im zu Beginn heruntergeladenen Archiv enthalten waren:
  ```
  diff -w aufgabe3-out1.txt out1.txt
  diff -w aufgabe3-out2.txt out2.txt
  diff -w aufgabe3-out3.txt out3.txt
  ```
  Das Linux-Kommando `diff` vergleicht den Inhalt zweier Dateien und gibt die Unterschiede aus. Ziel ist, dass die Kommandos nichts ausgeben, also keine Unterschiede zwischen Soll-Ausgabe und Ihrer Ausgabe finden. Die Ausgabe von Unterschieden ist etwas kryptisch. Verwenden Sie deshalb das compare-Plugin in `notepad++` oder die compare-Funktionalität in VS Code für die genaue Analyse der Unterschiede.
- Prüfen Sie mit checkstyle, ob Sie Stilregeln verletzt haben. Bessern Sie nach, solange das Werkzeug Fehler meldet. Rufen Sie checkstyle im Arbeitsverzeichnis über ant auf:
  ```
  ant -Dpackage=aufgabe3 style
  ```
- Prüfen Sie mit spotbugs, ob Ihr Programm problematischen Code enthält. Bessern Sie nach, solange das Werkzeug Probleme meldet. Rufen Sie spotbugs wie folgt über ant auf:
  ```
  ant -Dpackage=aufgabe3 clean bugs
  ```
  spotbugs schreibt seine Meldungen nicht auf den Bildschirm, sondern in eine Datei `bugs.html`, die Sie mit einem Webbrowser anschauen müssen.

Schritt 4: Abgabe

Den spätesten Abgabetermin finden Sie auf der Webseite der Lehrveranstaltung. Geben Sie bitte erst ab, nachdem Sie Schritt 3 vollständig erledigt haben. Die Werkzeuge checkstyle und spotbugs dürfen keine Fehler mehr melden!

- Führen Sie einen Test mit manueller Eingabe vor.
- Führen Sie die Tests mit automatischer Eingabe aus Schritt 3 vor.
- Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Ergänzende Übungen für Schnellprogrammierer *(freiwillig, pro Spiegelpunkt 1 Bonuspunkt)*

- Verwenden Sie zum Prüfen der eingegebenen Noten Ausnahmebehandlung.
  Klammern Sie dazu die Prüfung der Eingabe in einen `try`-Block, werfen Sie darin falsche Eingaben mit `throw new java.util.InputMismatchException(...)` und fangen Sie die Ausnahmen in einem `catch`-Block. Geben Sie innerhalb des `catch`-Blocks die Fehlermeldung aus und springen Sie von dort mit `continue` zum nächsten Schleifendurchlauf.
- Erlauben Sie die Noteneingabe zusätzlich auch im Format `1`, `1-`, `2+`, `2`, `2-`, `3+`, `3`, `3-`, `4+`, `4`, `5`. Dabei soll eine `1` als `1,0` gewertet werden, eine `1-` als `1,3` und eine `2+` als `1,7` usw.

Übungsaufgabe 4: Klassenvariablen und Methoden

Als Voraussetzung für diese Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 4-22 nachbereitet und die zugehörigen Programmbeispiele nachvollzogen haben.
Ihr Unterverzeichnis `prog1-beispiele\Programme4\` sollte danach mindestens folgende Programme enthalten: `Maximum.java, maximum\Maximum.java, maximum\IntegerMethods.java, ClassVar.java.`

Nach dem Download und dem Entpacken von Aufgabe 4 sollten in Ihrem Arbeitsverzeichnis folgende Dateien hinzugekommen sein:

`aufgabe4-diff1.txt`, **`aufgabe4-diff3.txt`**	Dateien für den automatisierten Funktionstest
`aufgabe4\package-info.java` **`aufgabe4\Klausurergebnis.java`**	Unterverzeichnis für Java-Quellcode, darin die Paketdokumentation und das noch unvollständige Java-Programm

Schritt 1: Vorbereitung

Aktualisieren Sie Ihre Javadoc-Seiten:

```
ant doc
```

Beim Paket aufgabe4 kommen einige Fehlermeldungen, weil sie dieses Paket in Schritt 2 erst noch vervollständigen müssen. Die Meldungen können Sie ignorieren.

Öffnen Sie im Browser alle Lesezeichen aus `Lesezeichen.html`.

Lesen Sie die Spezifikation des zu erstellenden Programms auf der Javadoc-Seite des Pakets `aufgabe4`. Die Javadoc-Seite finden Sie im Browser-Tab "*Overview (Programmiertechnik 1)*" oder indem Sie die Datei `doc\index.html` mit dem Browser öffnen.

Schritt 2: Programmierung

Erstellen Sie zunächst im Paket `aufgabe4` eine öffentliche Utility-Klasse `Noten`. Diese Klasse soll das Notensystem der HTWG Konstanz beschreiben und zwar mittels der folgenden Elemente:

- zwei öffentlichen konstanten Klassenvariablen `BESTE` und `SCHLECHTESTE` vom Typ `double`, die mit der besten bzw. schlechtesten zulässigen Note initialisiert sind
- einer öffentlichen Klassenmethode `istZulaessig`, die als Parameter eine Note vom Typ `String` hat und als Wert `true` liefert, wenn die Note zulässig ist, sonst `false`.
  Die Methode soll keine Fehlermeldungen auf die Konsole schreiben.
  Zu den zulässigen Noten siehe Übungsaufgabe 3. Den Code für das Prüfen der Noten können Sie mit geringen Änderungen von dort übernehmen.
- einer öffentlichen Klassenmethode `toDouble`, die als Parameter eine Note vom Typ `String` hat und diese Note als Gleitkommazahl vom Typ `double` zurückliefert. Für unzulässige Noten soll eine `IllegalArgumentException` geworfen werden.
- einer öffentlichen Klassenmethode `toString`, die als Parameter eine Note vom Typ `double` hat und die `String` –Darstellung dieser Note zurückliefert. Für Noten, die nicht zwischen `BESTE` und `SCHLECHTESTE` einschließlich liegen soll eine `IllegalArgumentException` geworfen werden.
  Verwenden Sie `String.format` mit der Formatangabe `%.1f`, um die Darstellung mit einer Nachkommastelle zu erreichen.
- einer öffentlichen Klassenmethode `istBestanden`, die als Parameter eine Note vom Typ `double` hat und als Wert `true` liefert, wenn die Note kleiner oder gleich `4.0` ist, sonst `false`
- zwei öffentlichen Klassenmethoden `bessere` und `schlechtere`, die als Parameter jeweils zwei Noten vom Typ `double` haben und als Wert die bessere bzw. schlechtere dieser beiden Noten liefern

Vervollständigen Sie nun die Main-Klasse `aufgabe4.Klausurergebnis`:

- übernehmen Sie die für die Notenstatistik erforderlichen Variablen aus Ihrer Lösung `aufgabe3.Notenstatistik.java`.

  Realisieren Sie dann das Prüfen und Erfassen der Noten in `aufgabe4.Klausurergebnis` analog zu `aufgabe3.Notenstatistik.java`, nur jetzt mit Hilfe der Klassenvariablen und Klassenmethoden Ihrer neuen Klasse `aufgabe4.Noten`.

  Für unzulässige Noten soll anders als in Übungsaufgabe 3 eine einheitliche Fehlermeldung
  `"Unzulaessige Note ... wird ignoriert!"`
  auf die Standardausgabe geschrieben werden (die Unterschiede zwischen alten und neuen Meldungen finden Sie auch in den Testdateien `aufgabe4-diff*.txt`).

  Sie müssen in `aufgabe4.Klausurergebnis` alle Klassenvariablen und alle Klassenmethoden von `aufgabe4.Noten` sinnvoll verwenden.

Schritt 3: Test und Qualitätssicherung

- Beim Übersetzen von `Klausurergebnis.java` muss javac die Klasse `Noten` finden. Wenn Sie javac im Arbeitsverzeichnis aufrufen, funktioniert das problemlos:
  `javac aufgabe4\Klausurergebnis.java`

  Wenn Sie javac im Paketverzeichnis aufrufen, müssen Sie alle .java-Dateien angeben:
  `javac *.java`

  oder mit der Option -cp das Arbeitsverzeichnis als Class-Path angeben:
  `javac -cp .. KlausurErgebnis.java`
- Testen Sie das Programm mit den drei vorgegebenen Eingaben aus Aufgabe 3:

  ```
  java aufgabe4.Klausurergebnis < aufgabe3-in1.txt > out1.txt
  java aufgabe4.Klausurergebnis < aufgabe3-in2.txt > out2.txt
  java aufgabe4.Klausurergebnis < aufgabe3-in3.txt > out3.txt

  diff -w aufgabe3-out1.txt out1.txt | diff -w - aufgabe4-diff1.txt
  diff -w aufgabe3-out2.txt out2.txt
  diff -w aufgabe3-out3.txt out3.txt | diff -w - aufgabe4-diff3.txt
  ```

  Die Linux-Kommandofolgen mit `diff` dürfen wie gehabt keine Unterschiede zwischen der Soll-Ausgabe und Ihrer Ausgabe finden. Verwenden Sie bei Bedarf zur Fehlersuche das compare-Plugin in `notepad++` oder die compare-Funktionalität in VS Code.
- Prüfen Sie mit checkstyle, ob Sie Stilregeln verletzt haben. Bessern Sie nach, solange das Werkzeug Fehler meldet. Rufen Sie checkstyle im Arbeitsverzeichnis über ant auf:
  `ant -Dpackage=aufgabe4 style`

  checkstyle wird von Ihnen insbesondere ordentliche Javadoc-Kommentare für die öffentlichen Klassenvariablen und Klassenmethoden verlangen. Sehen Sie sich dazu als Vorlage die auf der Webseite der Lehrverranstaltung verlinkten Vorlesungsbeispiele an.
- Erstellen Sie Ihre endgültigen Javadoc-Seiten. Bessern Sie Ihre Javadoc-Kommentare nach, solange javadoc noch Fehler meldet:
  `ant doc`
- Prüfen Sie mit spotbugs, ob Ihr Programm problematischen Code enthält. Bessern Sie nach, solange das Werkzeug Probleme meldet. Rufen Sie spotbugs wie folgt über ant auf:
  `ant -Dpackage=aufgabe4 clean bugs`

  spotbugs schreibt seine Meldungen nicht auf den Bildschirm, sondern in eine Datei *`bugs.html`*, die Sie mit einem Webbrowser anschauen müssen.

Schritt 4: Abgabe

Den spätesten Abgabetermin finden Sie auf der Webseite der Lehrveranstaltung.

Geben Sie bitte erst ab, nachdem Sie Schritt 3 erledigt haben.
Die Werkzeuge checkstyle, javadoc und spotbugs dürfen keine Fehler mehr melden!

- Führen Sie die drei schon bei Aufgabe 3 verwendeten Testfälle vor.
- Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Ergänzende Übung für Schnellprogrammierer *(freiwillig, 1 Bonuspunkt)*

Erstellen Sie eine Klasse `aufgabe4.schweiz.Noten` . Die Klasse soll genau die gleichen Klassenvariablen und -methoden habe wir die die Klasse `aufgabe4.Noten`, soll aber das Schweizer Notensystem beschreiben:

Erlaubt sind (in dieser Schreibeweise) die Noten `1 1,5 2 2,5 3 3,5 4 4,5 5 5,5 6`.
Beste Note ist dabei die `6`, schlechteste die `1`. Als bestanden gelten Noten größer oder gleich `4`.

Passen Sie die Dateien mit Eingaben und Sollausgaben für die automatisierten Tests an das Schweizer Notensystem an.

Die einzige Änderung in `Klausurergebnis.java` im Verzeichnis `aufgabe4` ist die Anweisung

```
import aufgabe4.schweiz.Noten;
```

am Dateianfang.

Übungsaufgabe 5: Notenspiegel

Als Voraussetzung für diese Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 4-36 nachbereitet und die zugehörigen Programmbeispiele nachvollzogen haben.
Ihr Unterverzeichnis `prog1-beispiele\Programme4\` sollte danach mindestens folgende Programme enthalten: `Datum.java, Termin.java, nested\IntList.java.`

Nach dem Download und dem Entpacken von Aufgabe 5 sollten in Ihrem Arbeitsverzeichnis folgende Dateien hinzugekommen sein:

`aufgabe5\package-info.java`	Unterverzeichnis für Java-Quellcode, darin die Paketdokumentation und das noch unvollständige Java-Programm
`aufgabe5\Notenspiegel.java`	

Schritt 1: Vorbereitung

Aktualisieren Sie Ihre Javadoc-Seiten:

```
ant doc
```

Beim Paket aufgabe5 kommen einige Fehlermeldungen, weil sie dieses Paket in Schritt 2 erst noch vervollständigen müssen. Die Meldungen können Sie ignorieren.

Öffnen Sie im Browser alle Lesezeichen aus `Lesezeichen.html`.

Lesen Sie die Spezifikation des zu erstellenden Programms auf der Javadoc-Seite des Pakets `aufgabe5`. Die Javadoc-Seite finden Sie im Browser-Tab "*Overview (Programmiertechnik 1)*" oder indem Sie die Datei `doc\index.html` mit dem Browser öffnen.

Schritt 2: Programmierung

Erstellen Sie eine instanziierbare Klasse `aufgabe5.Note` für Wertobjekte wie folgt:

- mit einer privaten konstanten Instanzvariable `note` vom Typ `int`
- mit einem privaten `Konstruktor` mit einem Parameter zum Initialisieren der Instanzvariable
- mit zwei öffentlichen konstanten Klassenvariablen `BESTE` und `SCHLECHTESTE`, die jeweils ein `Note`-Objekt mit der besten bzw. schlechtesten Note referenzieren
- mit einer öffentlichen Fabrikmethode `valueOf` mit einem Parameter vom Typ `int`, die als Rückgabewert ein `Note`-Objekt mit der im Parameter angegebenen Note liefert

  Werfen Sie bei unzulässiger Note eine Ausnahme vom Typ `IllegalArgumentException` mit `"unzulaessige Note "` und der falschen Note als Fehlertext
  (zulässig sind die Noten `10`, `13`, `17`, `20`, `23`, `27`, `30`, `33`, `37`, `40`, `50`)
- mit einer weiteren öffentlichen Fabrikmethode `valueOf`, diesmal mit einem Parameter vom Typ `String`. Der übergebene String muss das von `toString` (s.u.) gelieferte Format haben.

  Werfen Sie bei falschem Format oder unzulässiger Note eine `IllegalArgumentException` mit `"unzulaessige Note "` und der falschen Note als Fehlertext.
- mit einer öffentlichen Instanzmethode `intValue`, die die im Objekt gespeichert Note liefert
- mit einer öffentlichen Instanzmethode `istBestanden`, die `true` liefert, wenn der Wert der Instanzvariable kleiner oder gleich `40` ist, sonst `false`
- mit öffentlichen Instanzmethoden `toString`, `equals` und `hashCode`.

  Verwenden Sie das Vorlesungsbeispiel `Datum` aus Teil 4 der Vorlesung als Vorlage.
  Die String-Darstellung der Objekte soll `"1,0"`, `"1,3"` usw. sein.
  Als Hashcode können Sie einfach den Wert der Instanzvariable verwenden.

Erstellen Sie eine instanziierbare Klasse `aufgabe5.Fachnote` für Entitäten wie folgt:

- mit zwei privaten konstanten Instanzvariablen `fach` vom Typ `String` und `note` vom Typ `Note` sowie zwei öffentliche Instanzmethoden gleichen Namens, die jeweils den Wert der Variablen liefern.
- mit einem öffentlichen `Konstruktor` mit zwei Parametern zum Initialisieren der beiden Instanzvariablen

  Fachname und Note dürfen nicht `null` sein und der Fachname darf nicht die Länge `0` haben. Werfen Sie eine Ausnahme vom Typ `IllegalArgumentException`, wenn die Werte der Parameter diese Konsistenzregel verletzten

Erstellen Sie eine instanziierbare Klasse `aufgabe5.FachnotenListe` für Entitäten:

- Verwenden Sie das Vorlesungsbeispiel `nested.IntList` aus Teil 4 als Vorlage.
  Passen Sie darin einfach den Paket- und Klassennamen an und stellen Sie den Typ der gespeicherten Werte von `int` auf `Fachnote` um.

Vervollständigen Sie die Main-Klasse `aufgabe5.Notenspiegel` aus dem entpackten Archiv:

- TODO-Kommentar (3):

  Halten Sie sich an das Ausgabeformat, das auf der zu Beginn erzeugten Javadoc-Seite des Pakets `aufgabe5` beschrieben ist.

  Damit eine Tabellenstruktur mit linksbündigen Spalten entsteht, müssen Fachnamen, die kürzer als der längste vorkommende Fachname sind, rechts mit Leerstellen aufgefüllt werden. Bestimmen Sie deshalb vor der Ausgabe zunächst den längsten Fachnamen.

Schritt 3: Test und Qualitätssicherung

- Beim Übersetzen von `Notenspiegel.java` muss javac die Klassen `Fachnote` und `Noten` finden. Wenn Sie javac im Arbeitsverzeichnis aufrufen, funktioniert das problemlos:
  `javac aufgabe5\Notenspiegel.java`

  Wenn Sie javac im Paketverzeichnis aufrufen, müssen Sie mit `-cp` den Classpath angeben:
  `javac -cp .. Notenspiegel.java`
- Testen Sie das Programm mit verschiedenen Eingaben. Erstellen Sie dazu Dateien mit Testeingaben im Stil von Aufgabe 3 und 4 (`aufgabe5-in1.txt` usw.)
- Prüfen Sie mit checkstyle, ob Sie Stilregeln verletzt haben. Bessern Sie nach, solange das Werkzeug Fehler meldet. Rufen Sie checkstyle im Arbeitsverzeichnis über ant auf:
  `ant -Dpackage=aufgabe5 style`
- Erstellen Sie Ihre endgültigen Javadoc-Seiten. Bessern Sie Ihre Javadoc-Kommentare nach, solange javadoc noch Fehler meldet:
  `ant doc`
- Prüfen Sie mit spotbugs, ob Ihr Programm problematischen Code enthält. Bessern Sie nach, solange das Werkzeug Probleme meldet. Rufen Sie spotbugs wie folgt über ant auf:
  `ant -Dpackage=aufgabe5 clean bugs`

  spotbugs schreibt seine Meldungen nicht auf den Bildschirm, sondern in eine Datei *`bugs.html`*, die Sie mit einem Webbrowser anschauen müssen.

Schritt 4: Abgabe

Den spätesten Abgabetermin finden Sie auf der Webseite der Lehrveranstaltung.

Geben Sie bitte erst ab, nachdem Sie Schritt 3 erledigt haben! Die Werkzeuge checkstyle, javadoc und spotbugs dürfen keine Fehler mehr melden!

- Führen Sie Ihre in Schritt 3 erstellten Testfälle `aufgabe5-in1.txt` usw. vor.
- Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Ergänzende Übung für Schnellprogrammierer *(freiwillig, pro Spiegelpunkt 1 Bonuspunkt)*

- Bachelorarbeiten werden an der HTWG von zwei Prüfern bewertet. Aus den Bewertungen der beiden Prüfer wird der MIttelwert gebildet. Ergänzen Sie passend dazu in Ihrer Wertklasse Note eine weitere Fabrikmethode `valueOf(Note, Note)`, die ein `Note`-Objekt mit dem Mittelwert der beiden übergebenen Noten liefert.

  Erweitern Sie `Notenspiegel.main` um die Möglichkeit, bei einem Fach wahlweise eine oder zwei Noten eingeben zu können. Damit die Erkennung einer zweiten Note nicht zu kompliziert wird, dürfen Sie verlangen, das beide Noten als ganze Zahlen eigegeben werden. Dann können Sie die zweite Noteneingabe mit der Instanzmethode `hasNextInt()` aus der Klasse `Scanner` erkennen.

- **Als Voraussetzung für diese ergänzende Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 5-26 nachbereitet haben.**

  Sorgen Sie dafür, dass die verkettete Liste der Fachnoten immer nach Noten sortiert ist, mit der besten Note am Listenanfang. Fügen Sie dazu in `FachnotenListe.insert` neue Fachnoten nicht am Anfang der Liste ein, sondern suchen Sie zuerst mit einer Schleife das richtige Vorgängerelement.

  Für diese Zusatzaufgabe muss die Klasse `Note` eine natürliche Ordnung festlegen, indem sie die Schnittstelle `Comparable<Note>` implementiert. Verwenden Sie das Vorlesungsbeispiel `comparator.Datum` aus Teil 5 der Vorlesung als Vorlage.

  Für diese Zusatzaufgabe darf die Instanzvariable `next` in der Klasse `FachnotenListe.Element` keine `final`-Markierung haben.

Übungsaufgabe 6: HTML-Notenspiegel

Als Voraussetzung für diese Aufgabe müssen Sie die Vorlesungsunterlagen bis Seite 5-31 nachbereitet und die zugehörigen Programmbeispiele nachvollzogen haben.
Ihr Unterverzeichnis `prog1-beispiele\Programme5\` *sollte danach mindestens folgende Programme enthalten:* `vererbung\Datum.java, vererbung\Termin.java, vererbung\OrtsTermin.java vererbung\TerminTest.java, local\IntList.java, local\ListVar.java.`

Nach dem Download und dem Entpacken von Aufgabe 6 sollten in Ihrem Arbeitsverzeichnis folgende Dateien hinzugekommen sein:

`BennoBeispiel.html`	Beispiel für einen Notenspiegel
`aufgabe6\package-info.java` **`aufgabe6\HtmlNotenspiegel.java`** **`aufgabe6\Leistung.java`** **`aufgabe6\LeistungsListe.java`**	Unterverzeichnis für Java-Quellcode, darin die Paketdokumentation und das noch unvollständige Java-Programm

Schritt 1: Vorbereitung

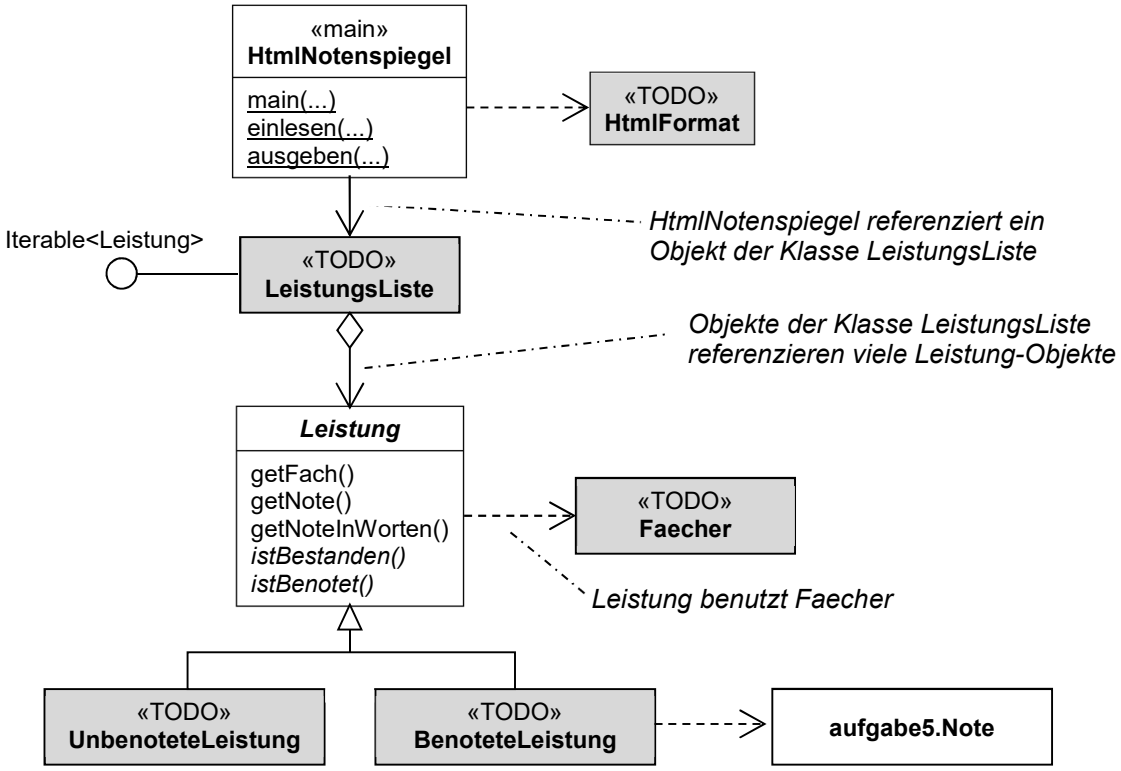
- Aktualisieren Sie Ihre Javadoc-Seiten:
  `ant doc`

  Beim Paket aufgabe6 kommen einige Fehlermeldungen, weil sie dieses Paket in Schritt 2 erst noch vervollständigen müssen. Die Meldungen können Sie ignorieren.

  Öffnen Sie im Browser alle Lesezeichen aus `Lesezeichen.html`.

  Lesen Sie die Spezifikation des zu erstellenden Programms auf der Javadoc-Seite des Pakets `aufgabe6`. Die Javadoc-Seite finden Sie im Browser-Tab "*Overview (Programmiertechnik 1)*" oder indem Sie die Datei `doc\index.html` mit dem Browser öffnen.

- Das folgende UML-Diagramm gibt Ihnen einen Überblick über die in Schritt 2 zu erstellenden Klassen:



Schritt 2: Programmierung

Erstellen Sie eine Utility-Klasse `aufgabe6.Faecher` mit folgenden Komponenten:

- einer privaten konstanten Klassenvariablen `FAECHER` vom Typ Array von Strings, die alle laut AIN-Studienplan des ersten Semesters erlaubten Fachnamen enthält
- einer öffentlichen Klassenmethode `istZulaessig()`, die prüft, ob ein gegebenes Fach im Studienplan vorgesehen ist (zur Signatur siehe den Aufruf der Methode in der Klasse `aufgabe6.Leistung`)

Erstellen Sie eine instanziierbare Klasse `aufgabe6.UnbenoteteLeistung` für Entitäten als Unterklasse von `aufgabe6.Leistung`:

- die Klasse soll eine konstante private Instanzvariable haben, in der gespeichert ist, ob die Leistung bestanden ist oder nicht
- die Klasse soll einen öffentlichen Konstruktor haben (die erforderliche Signatur können Sie dem `new`-Ausdruck in der Klasse `aufgabe6.HtmlNotenspiegel` entnehmen)
- die Klasse muss geerbte Methoden überschreiben. Welche das sind, können Sie der in Schritt 1 erstellten Javadoc-Beschreibung der Oberklasse `aufgabe6.Leistung` entnehmen.

Erstellen Sie eine instanziierbare Klasse `aufgabe6.BenoteteLeistung` für Entitäten als Unterklasse von `aufgabe6.Leistung`:

- die Klasse soll eine konstante private Instanzvariable vom Typ `aufgabe5.Note` zum Speichern der Note haben.
- die Klasse soll einen öffentlichen Konstruktor haben (die erforderliche Signatur können Sie dem `new`-Ausdruck in der Klasse `aufgabe6.HtmlNotenspiegel` entnehmen)
- die Klasse muss geerbte Methoden überschreiben. Welche das sind, können Sie der in Schritt 1 erstellten Javadoc-Beschreibung der Oberklasse `aufgabe6.Leistung` entnehmen.

  Verwenden Sie die entsprechende Instanzmethode der Wert-Klasse `aufgabe5.Note`, um zu entscheiden, ob die gekapselte Note als bestanden gilt.

  Halten Sie sich außerdem an die folgenden Notennamen der HTWG:

`"sehr gut"`	für Noten `1,0` bis `1,5` einschließlich
`"gut"`	für Noten `1,6` bis `2,5` einschließlich
`"befriedigend"`	für Noten `2,6` bis `3,5` einschließlich
`"ausreichend"`	für Noten `3,6` bis `4,0` einschließlich
`"nicht ausreichend"`	für Noten ab `4,1`

Erstellen Sie eine instanziierbare Klasse `aufgabe6.LeistungsListe` für Entitäten:

- Verwenden Sie das Vorlesungsbeispiel `local.IntList` aus Teil 5 als Vorlage. Passen Sie darin den Paket- und Klassennamen an, implementieren Sie die Schnittstelle `java.util.Iterator<Leistung>` statt `java.util.Iterator<Integer>` und stellen Sie den Typ der gespeicherten Werte von `int` auf `Leistung` um. Das Boxing in der Implementierung der `next`-Methode kann entfallen.

Erstellen Sie eine Utility-Klasse `aufgabe6.HtmlFormat` mit einer öffentlichen Klassenmethode `ausgeben`:

- Leiten Sie die Signatur der Klassenmethode aus dem Aufruf in `aufgabe6.HtmlNotenspiegel.ausgeben` ab.
- Leiten Sie die Implementierung der Klassenmethode aus der Beispielausgabe in der Datei `BennoBeispiel.html` ab. Sie können sich den Inhalt von `BennoBeispiel.html` mit einem Texteditor ansehen. Wenn Sie `BennoBeispiel.html` im Browser öffnen, sehen Sie die formatierte Darstellung des Inhalts. Verwenden Sie eine for-each-Schleife zum Ablaufen der übergebenen Leistungsliste und geben Sie den HTML-Text mit `printf` und `println` aus.

  Wenn Sie den Inhalt von `BennoBeispiel.html` etwas tiefer verstehen wollen, sind die folgenden Webseiten eine gute Hilfe:

  ```
  https://wiki.selfhtml.org/
  https://www.w3schools.com/html/default.asp
  ```

Schritt 3: Test und Qualitätssicherung

- Übersetzen Sie das Programm:

  `ant -Dpackage=aufgabe6 compile`

  Andere Möglichkeiten zum Aufruf von **`javac`** sind bei Aufgabe 5 beschrieben.
- Testen Sie das Programm mit verschiedenen Eingaben.

  Erstellen Sie dazu wieder Dateien mit Testeingaben im Stil von Aufgabe 3 und 4 (`aufgabe6-in1.txt` usw.) und betrachten Sie die Ausgabedateien mit dem Browser Firefox:

  `java -ea aufgabe6.HtmlNotenspiegel` ***`Vorname Nachname`***

  Die Option `-ea` sorgt dafür, dass die `assert`-Regeln in `Leistung` ausgewertet werden.

  Verwenden Sie im Firefox auch `rechte Maustaste->Quelltext anzeigen`, um Ihre HTML-Ausgabe zu prüfen.
- Prüfen Sie mit checkstyle, ob Sie Stilregeln verletzt haben. Bessern Sie nach, solange das Werkzeug Fehler meldet. Rufen Sie checkstyle im Arbeitsverzeichnis über ant auf:

  `ant -Dpackage=aufgabe6 style`
- Erstellen Sie Ihre endgültigen Javadoc-Seiten. Bessern Sie Ihre Javadoc-Kommentare nach, solange javadoc noch Fehler meldet:

  `ant doc`
- Prüfen Sie mit spotbugs, ob Ihr Programm problematischen Code enthält. Bessern Sie nach, solange das Werkzeug Probleme meldet. Rufen Sie spotbugs wie folgt über ant auf:

  `ant -Dpackage=aufgabe6 clean bugs`

  spotbugs schreibt seine Meldungen nicht auf den Bildschirm, sondern in eine Datei *`bugs.html`*, die Sie mit einem Webbrowser anschauen müssen.

Schritt 4: Abgabe

Den spätesten Abgabetermin finden Sie auf der Webseite der Lehrveranstaltung.

Geben Sie bitte erst ab, nachdem Sie Schritt 3 erledigt haben!
Die Werkzeuge checkstyle und spotbugs dürfen keine Fehler mehr melden!

- Führen Sie Ihre in Schritt 3 erstellten Testfälle `aufgabe6-in1.txt` usw. vor und zeigen Sie die Notenspiegel im Browser.
- Zeigen Sie ihre Javadoc-Seiten mit dem verlinkten Quellcode der Klassen.
- Zeigen Sie das ausgefüllte Teilnahmeprotokoll.

Ergänzende Übungen für Schnellprogrammierer *(freiwillig, pro Spiegelpunkt 1 Bonuspunkt)*

- Seit Java 15 gibt es zusätzlich zu den String-Literalen `"..."` auch Textblöcke, die in dreifache Anführungszeichen `"""` eingeschlossen werden und über mehrere Zeilen gehen können.
  Lesen Sie dazu `https://docs.oracle.com/en/java/javase/15/text-blocks/`
  und nutzen Sie dann in Ihrer Klasse **`HtmlFormat`** Textblöcke, um die Ausgabe des HTML-Texts einfacher und übersichtlicher zu gestalten.

- Erstellen Sie eine Utility-Klasse **`TextFormat`** analog zu `HtmlFormat`, die einen Notenspiegel im tabellarischen Format nach dem Vorbild von Aufgabe 5 in einer Datei mit Endung `.txt` ausgibt.

  Bauen Sie die Verwendung der Klasse zusätzlich zu `HtmlFormat` in die Methode `aufgabe6.HtmlNotenspiegel.ausgeben` ein.

- Sorgen Sie dafür, dass in der verketteten Liste der Fachnoten keine Leistung doppelt vorkommt.

  Ändern Sie dazu `LeistungsListe.insert` so ab, dass nach dem Einfügen der neuen Leistung am Listenanfang in der restlichen Liste nach einer Leistung gesucht wird, bei der `getFach` und `istBenotet` denselben Wert wie bei der neuen Leistung liefern. Wird eine solche Leistung gefunden, soll Sie aus der Liste entfernt und mit `return` zurückgeliefert werden. Dazu müssen Sie den Rückgabetyp von `LeistungsListe` auf `Leistung` ändern.

  An der Aufrufstelle in `HtmlNotenspiegel.eingeben` soll für jede Leistung, die aus der Liste entfernt wurde, eine Warnung ausgegeben werden.