Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 2 SoSe 2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof R. Lorenz, L. Petrak, J. Linne, V. Le Claire, P. Schalk 19.04.2024

Übungsblatt 1

Abgabe: 29.04.2024, 10:00 Uhr (im Digicampus via VIPS: .java-Dateien für Code, .uxf für UML, .pdf für alles andere)

- Dieses Übungsblatt muss im Team abgegeben werden (Einzelabgaben sind nicht erlaubt!).
- Die **Zeitangaben** geben zur Orientierung an, wie viel Zeit für eine Aufgabe später in der Klausur vorgesehen wäre; gehen Sie davon aus, dass Sie zum jetzigen Zeitpunkt wesentlich länger brauchen und die angegebene Zeit erst nach ausreichender Übung erreichen.

* leichte Aufgabe / ** mittelschwere Aufgabe / *** schwere Aufgabe

Allgemeine Hinweise zum Übungsbetrieb

1. Die Übungsblätter sind so gedacht, dass Sie die ersten Teilaufgaben in den Tutorien lösen. Ihr/e Tutor/in wird sie dabei unterstützen, haben Sie also keine Angst davor, ihm/ihr all Ihre Fragen zu stellen. Natürlich dürfen Sie auch Fragen zu anderen Aufgabenteilen des Übungsblatts oder zu Teilen der Vorlesung stellen.

   In den Tutorien werden Sie in Teams von jeweils 3 Personen eingeteilt, wenn Sie nicht schon ein Team haben. Dies dient dem Zweck, dass Sie schon jetzt Erfahrung damit sammeln, mit anderen Personen gemeinsam produktiv zu arbeiten. Ihr gebildetes Team hinterlegen Sie bitte im Digicampus, indem Sie in der Veranstaltung *„Informatik 2“* auf den Tab `Vips` und anschließend links auf `Übungsgruppen` klicken. Suchen Sie sich danach in der angezeigten Liste eine Gruppe aus, in der noch genügend Platz für alle Team-Mitglieder ist und klicken Sie rechts von dem Gruppennamen auf das Symbol, um sich zur Gruppe hinzuzufügen. Dies müssen **alle** Teammitglieder machen.

   Im Idealfall teilen Sie sich die zu erledigenden Aufgaben eines Übungsblattes **nicht** im Team auf, sondern bearbeiten unabhängig voneinander alle Aufgaben vom Übungsblatt. Anschließend diskutieren Sie im Team über Ihre Lösungen und einigen sich auf **eine** gemeinsame Lösung, die Sie abgeben möchten.

   Ihre Lösung zu einem Übungsblatt muss nur von einer Person aus Ihrem Team abgegeben werden. Eine Anleitung dazu, wie Sie dies machen, finden Sie im Digicampus oder unter https://mediastore.rz.uni-augsburg.de/get/aphEoyIlhN/.

2. Neben den Tutorien bieten wir jede Woche den *Offenen Inforaum* an: Dieser findet jeden **Donnerstag** von **15:45 bis 17:15 Uhr** im Raum **2013N** statt. Es ist eine zusätzliche Möglichkeit, um sich mit Ihrem Team zu treffen und weiter am Übungsblatt zu arbeiten. Für Fragen steht Ihnen die Mitarbeiterin *Patrizia Schalk* zur Verfügung. Sie können im Offenen Inforaum auch Fragen stellen, die sich nicht auf das aktuelle Übungsblatt beziehen. Der erste Offene Inforaum findet am **Donnerstag, den 25.04.24** statt.

3. Zusätzlich zu den Übungsblättern können Sie Ihre Programmierfähigkeiten im *Betreuten Programmieren* ausbauen. Hierzu erscheint jede Woche ein eigenes Übungsblatt mit einer größeren Programmieraufgabe. Jeden **Mittwoch** von **14:00 bis 17:15 Uhr** erhalten Sie

hierzu Betreuung in den CIP-Pools 1001N, 1002N und 1005N (bei geringer Nachfrage wird die Betreuung evtl. nur noch in einem oder zwei der CIP-Pools stattfinden). Die Aufgaben können Sie dort alleine oder im Team bearbeiten. Das erste Betreute Programmieren findet am **Mittwoch, den 24.04.24** statt.

Allgemeine Hinweise zur Abgabe der Übungsblätter
Wir geben Ihnen die Möglichkeit, Ihre Lösung der wöchentlich herausgegebenen Übungsblätter abzugeben und von einem Tutor korrigieren zu lassen. So erhalten Sie zeitnah Feedback, wo Sie stehen und erhalten im Idealfall noch einen Notenbonus auf eine bestandene Klausur. Folgende Punkte sind dabei zu beachten:

- **Abgabeort:** Die Abgabe erfolgt digital als Upload im Vips-Tool von Digicampus. Eine Anleitung dafür, wie man seine Lösung für ein Übungsblatt in Vips hochlädt, finden Sie im Digicampus oder unter https://mediastore.rz.uni-augsburg.de/get/aphEoyIlhN/.

- **Abgabeformat:** Für Programmieraufgaben sind alle Quellcode-Dateien als `*.java`-Dateien abzugeben. Diese Dateien legen Sie in einen Ordner mit dem Namen `src`, sodass der Tutor die Dateien leicht finden und ausführen kann. Neben diesem `src`-Ordner gehört **eine einzelne** `*.pdf`-Datei zu Ihrer Abgabe, in der Sie den Code für jede Abgabe lesbar einbinden, damit der Tutor seine Kommentare zum Code direkt in die PDF-Datei eintragen kann. In dieser `*.pdf`-Datei geben Sie auch alle weiteren Aufgaben ab (z.B. als gut lesbaren Scan Ihrer handschriftlichen Bearbeitung). Den `src`-Ordner und die `*.pdf`-Datei verpacken Sie zusammen in ein `*.zip`-Archiv, das Sie in Vips hochladen. Aus dem Namen des `*.zip`-Archivs sollte hervorgehen, zu welchem Team die Abgabe gehört (zum Beispiel durch Verwendung der in Vips gewählten Gruppennummer).

  Eine einfache Möglichkeit, Quellcode-Dateien in einer `*.pdf`-Datei darzustellen, bietet LaTeX: Sie finden im Digicampus sowohl eine Anleitung als auch eine passende Vorlage, die Sie verwenden können. Die Verwendung von LaTeX ist *nicht* verpflichtend, Sie können die notwendige `*.pdf`-Datei auch auf anderen Wegen erstellen.

- **Korrektur:** Wenn Sie sich bei Ihrer Abgabe an die obigen Punkte gehalten haben, erhalten Sie nach ca. einer Woche die Korrektur Ihrer Abgabe via Vips. Darin finden Sie neben der Bewertung Ihrer Lösung auch hilfreiche Hinweise und Tipps, was Sie in Zukunft beachten sollten. Lesen Sie sich Ihre Korrektur sorgfältig durch und beachten Sie die darin enthaltenen Hinweise für die Zukunft – und besonders für die Klausur.

Hinweise zum Notenbonus
Auf jedem Übungsblatt können Sie bis zu vier Übungspunkte sammeln, die am Ende des Semesters automatisch in Bonuspunkte auf die Klausur umgewandelt werden.

- Insgesamt ist es möglich, bis zu 51 Übungspunkte zu sammeln:
  - $10 \cdot 4 = 40$ Übungspunkte durch die Bearbeitung der Übungsblätter,
  - $11 \cdot 1 = 11$ Übungspunkte durch die Teilnahme am Betreuten Programmieren,

- Für jeweils 2.5 gesammelte Übungspunkte erhalten Sie 0.5 Bonuspunkte auf die Klausur, **sofern** Sie die Klausur ohne die Bonuspunkte bestanden haben. Sie können allerdings nicht mehr als 6 Bonuspunkte für die Klausur sammeln.

- 6 Bonuspunkte ($\hat{=}$ 30 Übungspunkte) entsprechen einer Notenstufe. Mit diesem ("vollen") Notenbonus können Sie Ihre Note also zum Beispiel von 2.3 auf 2.0 oder von 1.7 auf 1.3 verbessern. Eine Verbesserung von 4.3 auf 4.0 ist aber **nicht** möglich.

- Der Notenbonus zählt auch für die Wiederholungsklausur, nicht aber für Klausuren in späteren Semestern.

UML-Standard

Für die Modellierungssprache UML existieren verschiedene **Standards** (Spezifikationen), die im Laufe der Zeit zur Normierung der Sprache von speziellen Komitees entwickelt wurden bzw. immer noch werden.
Weil die Vorgaben dieser Standards im Bezug auf grafische Elemente manchmal vage sind, verwenden wir in der Veranstaltung *Informatik 2* einen eigenen Stil, den Sie im Laufe der Vorlesung kennen lernen – halten Sie sich also an den Stil der Vorlesung!

UML-Modellierungsumgebungen

Empfohlene Modellierungsumgebungen für die Erstellung von Klassenkarten und UML-Diagrammen:

- *UMLet*
  https://www.umlet.com/
- *UMLetino*
  https://www.umletino.com/

Java-Standard

Für die Programmiersprache Java existieren verschiedene **Standards** (Spezifikationen), die im Laufe der Zeit zur Normierung der Sprache von speziellen Komitees entwickelt wurden bzw. immer noch werden.
In der Veranstaltung *Informatik 2* werden wir uns an den Standard **Java SE 21** (häufig kurz: *Java 21*) halten, der die aktuelle LTS (Long Term Support) - Version ist.

- *OpenJDK 21* Bibliotheken, Compiler, Interpreter
  https://adoptium.net/ (Temurin)
- *Java SE 21 - API* (vgl. Standard-Bibliothek in C)
  https://docs.oracle.com/en/java/javase/21/docs/api/index.html

Achtung:

- Achten Sie darauf, dass auf Ihrem System bzw. in Ihrer Entwicklungsumgebung die geforderte Variante des *JDK* (Java Development Kit) installiert ist.
- Achten Sie bei Suchanfragen in Suchmaschinen darauf, dass nicht auf eine veraltete API verwiesen wird (leicht zu erkennen daran, dass z. B. eine andere Zahl als die 21 im Link zur API steht).

Entwicklungsumgebung (IDE)

Die wichtigsten Entwicklungsumgebungen für die Entwicklung von Java-Programmen:

- *Eclipse* (Eclipse IDE for Java Developers)
  https://www.eclipse.org/
- *IntelliJ IDEA*
  https://www.jetbrains.com/idea/

Alle genannten IDEs sind empfehlenswert und besitzen unterschiedliche Vorteile. Es ist ausreichend, eine der genannten Umgebungen zu verwenden.

Coding Conventions

Ein *guter Programmierstil* zeichnet sich dadurch aus, dass sich der Programmierer an vorgegebene Konventionen hält, die sich im Laufe der Zeit als vorteilhaft erwiesen haben. Wesentliche Gründe für die Einhaltung eines guten Stils sind:

- Bessere Lesbarkeit
- Bessere Wartbarkeit
- Bessere Team-/Projektarbeit
- Geringere Fehleranfälligkeit

Es gibt allerdings nicht *den richtigen Stil*, sondern lediglich eine Menge an verschiedenen Vorschlägen, die viele Gemeinsamkeiten aufweisen. Wichtig ist es deshalb, sich an firmen-/projektspezifische Vorgaben halten zu können, indem man die Grundlagen lernt.

Beispielhafte Style Guides für Java:

- *Java Coding Style Guide*
  `https://user.eng.umd.edu/~austin/ence489c.d/java-style.html`
- *Google Java Style*
  `https://google.github.io/styleguide/javaguide.html`

In dieser Veranstaltung werden wir uns an die Vorgaben des *Java Coding Style Guide* halten. Betrachten Sie den *Google Java Style* als optionale Ergänzung.

Achten Sie bei allen zukünftigen Übungsaufgaben auf **Kompilierbarkeit** und die **Einhaltung der Coding Conventions**. Kommentieren Sie gegebenenfalls nicht funktionierende Programmteile aus, um ein minimales kompilierbares Programm abgeben zu können. Schreiben Sie in den auskommentierten Teil auch, was dieser bezwecken sollte.

Die von Ihnen geschriebenen Programme laden Sie zur Abgabe ausschließlich im entsprechenden Abgabe-Ordner im Digicampus hoch. Neben einer PDF-Datei, die Ihre Lösungen enthält, umfasst Ihre Abgabe auch Ihre Quellcode-Dateien (Dateiendung `.java`). Laden Sie bitte keine `*.class`-Dateien hoch!

Als Zeichenkodierung verwenden Sie **UTF-8**.

Java Coding Style Guide – Ein kleiner Auszug

Die in diesem Auszug gegebenen Hinweise zum Stil sind ausschließlich Hinweise für Code-Elemente, die Sie in den ersten Wochen kennenlernen werden. Erweitern Sie Ihr Wissen durch Nachlesen in den oben genannten Dokumenten **selbstständig**, sobald neue Elemente in der Vorlesung vorgestellt werden!

- In Eclipse:
  - Window → Preferences → Java → Code Style → Formatter
    Hier als Profil „Java Conventions“ auswählen.
  - Automatisches Formatieren: Source → Format (Shift + Ctrl + F)
- Automatisches Formatieren In IntelliJ: Code → Reformat File (Ctrl + Alt + L)

1. Sprechende Namen für Variablen/Methoden im *lowerCamelCase*:

```
FALSCH:
int avh;
int AlterHund;
int alter_hund;
```

```
RICHTIG:
int alterHund;
```

2. Ein Tab einrücken pro Verschachtelungstiefe (== 1x Tab-Taste pro {}-Block):

```
FALSCH:
if (a == b) {
 for (...) {
  x = 4;
  y = 3;
 }
 z = 6;
}
else { a = 8; }
```

```
RICHTIG:
if (a == b) {
    for (...) {
        x = 4;
        y = 3;
    }
    z = 6;
} else {
    a = 8;
}
```

3. Geschweifte Klammer immer in der Zeile des Methodenkopfes:

```
FALSCH:
int meineMethode()
{
    ...
}
```

```
RICHTIG:
int meineMethode() {
    ...
}
```

4. Leerzeichen richtig setzen:

```
FALSCH:
(a,b,c)
( a == b )
(a+b)
while(x != 0){
```

```
RICHTIG:
(a, b, c)
(a == b)
(a + b)
while (x != 0) {
```

Hinweise zu den Programmieraufgaben:
Achten Sie bei allen zukünftigen Programmieraufgaben auf *Kompilierbarkeit* und *Einhaltung der Coding Conventions*; auch dann, wenn es nicht explizit im Aufgabentext gefordert ist.

Gehen Sie davon aus, dass alle Programme in der Programmiersprache Java zu erstellen sind.

Aufgabe 1 *(Einfache Programme ohne Kommandozeilenparameter, 25 Minuten)*

In dieser Aufgabe sollen Sie einfache Programmklassen schreiben und dabei verschiedene API-Klassen kennen lernen. Denken Sie daran, dass Sie Klassen, die nicht im Paket `java.lang` enthalten sind, erst importieren müssen, bevor Sie diese verwenden können. Klassenmethoden von Klassen im Paket `java.lang` können mit `Klassenname.methodenname()` verwendet werden.

a) *(*, Klassenmethoden, 10 Minuten)*
Implementieren Sie zu jeder Unteraufgabe eine eigene Programmklasse, welche die gestellte Aufgabe erfüllt. Implementieren Sie eine Programmklasse, die...

1. das Ergebnis von $\sin(2.5) \cdot \sqrt{5}$ mithilfe der Methoden `sin` und `sqrt` der Klasse `Math` berechnet und ausgibt.
2. das Minimum von $\log_{10}(2534)$ und e^2 bestimmt und ausgibt. Verwenden Sie dafür geeignete Klassenmethoden der Klasse `Math`.
3. das Vorzeichen von $\tan(42)$ mithilfe einer geeigneten Klassenmethode der Klasse `Math` bestimmt und abhängig davon ausgibt, ob der Wert positiv, negativ oder gleich 0 ist.
4. mit einer geeigneten Klassenmethode der Klasse `Integer` die Zeichenkette `"2436"` in einen `int`-Wert verwandelt und das Zweifache dieses Werts ausgibt.

b) *(Objektmethoden, 15 Minuten)*
Implementieren Sie zu jeder Unteraufgabe eine eigene Programmklasse, welche die gestellte Aufgabe erfüllt. Implementieren Sie eine Programmklasse, die...

1. ein `String`-Objekt mit dem Inhalt `"Informatik"` angelegt, diese mit der Objektmethode `toUpperCase` in Großbuchstaben umwandelt und auf der Kommandozeile ausgibt.
2. eine `String`-Objekt mit dem Wert `"Programmieren"` anlegt, mit einer geeigneten Objektmethode die erste Position des Zeichens `'m'` bestimmt und auf der Kommandozeile ausgibt.
3. eine Zeichenkette `"Parallele"` anlegt und mit einer geeigneten Objektmethode alle Vorkommen der Teilzeichenkette `"le"` durch `"abc"` ersetzt und speichert sowie das Ergebnis auf der Kommandozeile ausgibt.
4. ein leeres `StringBuilder`-Objekt erstellt, an dieses 100 Mal die Zeichenkette `"ha"` anhängt und das Ergebnis auf der Kommandozeile ausgibt.
5. ein `StringBuilder`-Objekt mit dem Inhalt `"Informatik"` erzeugt, mit geeigneten Methoden an jeder zweiten Stelle das Zeichen `'a'` einfügt und das Ergebnis auf der Kommandozeile ausgibt.
   Hinweis: Weil sich beim Einfügen die Position der darauffolgenden Zeichen verschiebt, sollten Sie die Zeichenkette von hinten nach vorne durchlaufen. Das Ergebnis lautet dann `"aInafoarmaataika"`.
6. ein `Scanner`-Objekt erzeugt, das den Standardeingabestrom `System.in` als Konstruktor-Parameter übergeben bekommt. Dann soll eine Zeichenkette eingelesen und danach ausgegeben werden. Die Klasse `Scanner` ist im Paket `java.util` enthalten.

Aufgabe 2 *(Einfache Programme mit Kommandozeilenparametern, 21 Minuten)*

In dieser Aufgabe sollen Sie einfache Programmklassen schreiben und dabei verschiedene API-Klassen kennen lernen. Zusätzlich lernen Sie die Verwendung von Kommandozeilenparametern kennen. Implementieren Sie eine Programmklasse, die...

a) *(*, 3 Minuten)*
jeden übergebenen Kommandozeilenparameter zusammen mit seiner Länge in einer eigenen Zeile auf der Kommandozeile ausgibt.

b) *(*, 3 Minuten)*
eine `String`-Variable mit leerem Inhalt anlegt, an diese nacheinander alle Kommandozeilenparameter anhängt und das Ergebnis auf der Kommandozeile ausgibt.

c) *(**, 5 Minuten)*
zählt, wie oft das Zeichen `'c'` in allen Kommandozeilenparametern zusammen vorkommt und das Ergebnis auf der Kommandozeile ausgibt.

d) *(*, 5 Minuten)*
einen zufällig gewählten Kommandozeilenparameter auf der Kommandozeile ausgibt. Erzeugen Sie dafür ein Objekt der Klasse `Random` und verwenden Sie eine geeignete Objektmethode. Denken Sie daran, dass sie die Klasse importieren müssen, um sie verwenden zu können! Werden keine Kommandozeilenparameter übergeben, soll stattdessen auf der Standardfehlerausgabe `"Error!"` ausgegeben werden.

e) *(**, 5 Minuten)*
mit einem Objekt der Klasse Random faire Münzwurfe[1] simuliert. Solange der Münzwurf Kopf zeigt und noch Kommandozeilenparameter übrig sind, soll der nächste Kommandozeilenparameter ausgegeben werden.

[1] d.h. Kopf und Zahl treten beide mit Wahrscheinlichkeit 0.5 auf

Aufgabe 3 *(Einfache Methoden implementieren, 20 Minuten)*

In dieser Aufgabe sollen Sie einfache statische Klassenmethoden implementieren und jeweils in einer eigenen `main`-Methode testen.

a) *(*, 4 Minuten)*
Implementieren Sie eine statische Klassenmethode mit dem Methodenkopf
`public static String myToLowerCase(String s)`,
die ein `String`-Objekt als Parameter erwartet, mit einer geeigneten Objektmethode der Klasse `String` alle Großbuchstaben darin in Kleinbuchstaben umwandelt und das Ergebnis zurückgibt. Verwenden Sie in der `main`-Methode ein `Scanner`-Objekt, um eine Zeile vom Benutzer über die Kommandozeile einzulesen und geben Sie das Ergebnis der Konvertierung mit `myToLowerCase` zurück.

b) *(*, 5 Minuten)*
Implementieren Sie eine statische Klassenmethode
`public static int tripleMax(int a, int b, int c)`,
die das Maximum der Werte `a, b, c` zurückgibt. Lesen Sie in der `main`-Methode mit einem `Scanner`-Objekt nacheinander drei ganze Zahlen vom Benutzer über die Kommandozeile ein und geben Sie den größten der drei Werte aus. Verwenden Sie eine geeignete Methode der Klasse `Math`!

c) *(*, 5 Minuten)*
Implementieren Sie eine statische Klassenmethode, die als Parameter ein `int`-Array erwartet und das Produkt aller Einträge des Arrays zurückgibt. Enthält das Array keinen Eintrag, soll der Wert 1 zurückgegeben werden. Den Kopf der Methode entwerfen Sie selbst. Legen Sie in der javamain-Methode ein `int`-Array an und befüllen es mit Werten Ihrer Wahl. Geben Sie das Array mithilfe einer geeigneten Methode der Klasse `Arrays` im Paket `java.util` zusammen mit dem Produkt der enthaltenen Werte auf der Kommandozeile aus.

d) *(**, 6 Minuten)*
Implementieren Sie eine Methode, die mithilfe eines `Scanner`-Objekts so viele ganzzahlige Werte wie möglich am Anfang eines übergebenen Strings einliest und die Summe dieser Werte zurückgibt. Verwenden Sie eine geeignete Methode der Klasse Scanner, um zu überprüfen, ob noch eine weitere ganze Zahl ausgelesen werden kann. Wenn das der Fall ist, lesen Sie den Wert ein und addieren ihn zum bisherigen Ergebnis dazu. Wenn keine ganze Zahl eingelesen werden kann, soll der Wert 0 zurückgegeben werden. Den Kopf der Methode entwerfen Sie selbst. Testen Sie Ihre Methode, indem Sie in der `main`-Methode eine Zeile vom Benutzer einlesen und das Ergebnis Ihrer Methode für diesen String ausgeben.

Beispiel: Wird die Zeichenkette `"2 -5 3 abc 1"` eingegeben, soll das Ergebnis `0` ausgegeben werden, weil nach dem Einlesen von `"3"` keine weitere Zahl mehr eingelesen werden kann.

Aufgabe 4 *(Eigene Klasse implementieren, 20 Minuten)*

In dieser Aufgabe sollen Sie eine einfache Klasse zum Verwalten von Rechtecken implementieren. Ein Rechteck hat eine Länge und eine Breite. Außerdem stellt es Methoden zum Bearbeiten und Lesen der beiden Werte zur Verfügung sowie eine Methode, um seinen Umfang zu berechnen.

a) *(*, Klassenkarte entwerfen, 5 Minuten)*
Entwerfen Sie eine Klassenkarte für Rechtecke.

b) *(**, Klasse implementieren, 9 Minuten)*
Implementieren Sie eine Klasse `Rectangle`. Vergessen Sie nicht einen Konstruktor, um neue Objekte vom Typ `Rectangle` erstellen zu können.

c) *(**, Programmklasse implementieren, 6 Minuten)*
Implementieren Sie eine Programmklasse, die

- ein Rechteck `r1` mit Breite 10 und Höhe 2 erzeugt,
- ein Rechteck `r2` mit Breite 5 und Höhe 5 erzeugt,
- die Höhe von `r1` um 1 erhöht,
- die Breite von `r2` verdoppelt und
- ausgibt, welches der beiden Rechtecke den größeren Umfang hat.