Technische Universität Braunschweig
Institut für Anwendungssicherheit

Prof. Dr. Martin Johns
Marius Musch
Maximilian von Unwerth

Programmieren I

01. Übungsblatt

Melden Sie sich bitte für die Veranstaltung bei Stud.IP (http://studip.tu-bs.de) an und tragen sich in einer der kleinen Übungen ein. (`https://studip.tu-braunschweig.de/dispatch.php/course/details?sem_id=d08ea3cd66c59967be1218a540bdefa7`)

Verspätete, nicht hochgeladene Abgaben, Abgaben, die per Mail getätigt werden, sowie Plagiate führen zur Bewertung der Aufgabe mit 0 Punkten.

*Insgesamt können Sie für das Lösen der Pflichtaufgabe **15 Punkte** erhalten.*
*Berücksichtigt werden ausschließlich Ergebnisse, die bis zum **15.11.2020 um 23:59** in Ihr Git Repository auf `https://programming.ias.cs.tu-bs.de/` gepusht wurden.*

Ihre Lösung muss in den bereits für dieses Übungsblatt angelegten Ordner (blatt1) im Git Repository gespeichert werden, damit diese gewertet wird.

Nur compilierende Programme gelten als Lösung! Ein Programm das Teillösungen beinhaltet aber nicht compiliert oder aus anderen Gründen nicht ausführbar ist erhält 0 Punkte.

Empfehlung: Machen Sie sich mit einer Shell vertraut. Wir empfehlen die Verwendung der *bash*-Shell.

Installieren Sie zur Bearbeitung der Übungsblätter auf Ihrem Rechner das für ihr Betriebssystem passende `Java SE Development Kit`.
Zur Versionierung der Quelltexte setzen wir `git` (`https://git-scm.com/`) voraus.
Im Stud.IP finden Sie kurze Tutorials zur Installation von `git` und `Java SE Development Kit` für die gängigen Betriebssysteme.
Wir raten Ihnen ausschließlich einen normalen Texteditor (e.g. gEdit, Kate, nano, Notepad, emacs, vim) zum Programmieren zu verwenden.

Pflichtaufgabe 1: Zusammenarbeiten mit Git - (3 Punkte) Da zur Abgabe der folgenden Hausaufgaben das Versionsverwaltungssystem *Git* verwendet werden wird, soll in dieser Aufgabe der Umgang mit den verschiedenen Funktionen dieses Systems geübt werden.

- Suchen Sie sich innerhalb Ihrer Gruppe einen Studierenden, mit dem Sie die Hausaufgaben gemeinsam bearbeiten wollen.

- Einigen sich, welches Ihrer Repositories Sie für die Hausaufgaben verwenden wollen und sorgen Sie dafür, dass der Übungsleiter Ihrer Übung Ihnen Beiden Zugang zu diesem gibt.

- Klonen Sie Beide das Repository mit dem Befehl `git clone <url-des-repositories>`. Mit diesem Befehl erstellen Sie an dem Ort, an dem Sie diesen Befehl ausführen Ihr lokales Repository und laden die Dateien, die sich derzeit online befinden herunter.

- Eine Person Ihrer Gruppe verändert nun die Readme wie folgt, die sich jetzt auf ihrem Rechner finden lassen sollte. **Schreiben Sie in die Readme die Angaben, die wir später für die Anrechnung Ihrer Leistung beim Prüfungsamt dringend benötigen: Vorname und Nachname, Matrikelnummer, Studiengang und geplanter Abschluss.**

- Nach dem Speichern müssen die Dateiänderungen mit `git add <datei-mit-aenderungen>` zu ihrem **lokalen** Repository hinzugefügt werden. Erstellen Sie einen Zwischenstand, einen **Commit** des Repositories mittels `git commit -m "<commit-message>"`. Als **Commit-Message** geben Sie an, was Sie in diesem Zwischenstand bearbeitet haben. Haben Sie zum Beispiel an einem bestimmten Feature gearbeitet, erklären Sie in dieser Nachricht kurz ihr Vorgehen, was nun der Stand ist und wie Sie das alles gemacht haben. Die Nachricht ist wichtig für die Personen, die mit Ihnen an einem Projekt arbeiten. Helfen Sie diesen und schreiben Sie *gute* Commit-Messages.

- Wenn Sie nun ein paar Features oder hier eben nur eine kleine Änderung der Readme vorgenommen und commitet haben, können Sie diese in das Online-Repository hochladen. Geben Sie dafür einfach `git push` in die Kommandozeile ein. (1 Punkt)

- Wollen Sie nun die Änderungen Ihres Partners herunterladen, können Sie dies mit `git pull` tun. Wir wollen nun allerdings zuerst einen **Merge-Conflict** erzeugen, der insbesonder beim Arbeiten mit mehreren Personen entstehen kann. Gehen Sie dafür wie folgt vor.

  a) **Die Person, die nicht die Änderungen an der Readme vorgenommen hat, trägt nun ebenfalls seine Daten in die Datei ein**. Machen Sie vorher NICHT `git pull`. Nun gibt es online eine Version mit den Änderungen Ihres Partners.

  b) Wenn Sie Ihre Änderungen nun adden und commiten, gibt es neben der Online-Version der Datei Ihre eigene Version. Diese können durchaus völlig unterschiedlich sein.

  c) Wenn Sie nun versuchen die Datei zu pushen, weißt Sie git darauf hin, dass es online eine Version gibt, die Sie noch nicht kennen. Laden Sie diese mit `git pull` herunter, entsteht ein sog. **Merge-Conflict**.

  d) Haben Sie nun auch noch eine Zeile bearbeitet, die auch Ihr Partner verändert hat, kann Git dieses Problem auch nicht mehr automatisiert lösen und Sie müssen ran.

e) Lösen Sie den Merge-Conflict und pushen Sie die Datei erneut. (1 Punkt)

f) Zur Überprüfungung können Sie in Ihrem Repository einen Graphen finden, der den Verlauf der unterschiedlichen Versionen anzeigt. Machen Sie einen Screenshot dieses Graphen und pushen Sie ihn in ihr Repository. (1 Punkt)

Merke: Um Merge-Konflikte zu vermeiden, hilft es vor dem Beginn der täglichen Arbeit an dem Projekt `git pull` durchzuführen. Vielleicht hat der Partner ja über Nacht noch etwas geschafft?!

In den folgenden Aufgaben werden wir noch weitere Funktionen von Git nutzen. Sorgen Sie dafür, dass Sie diese grundlegenden Funktionen beherrschen.

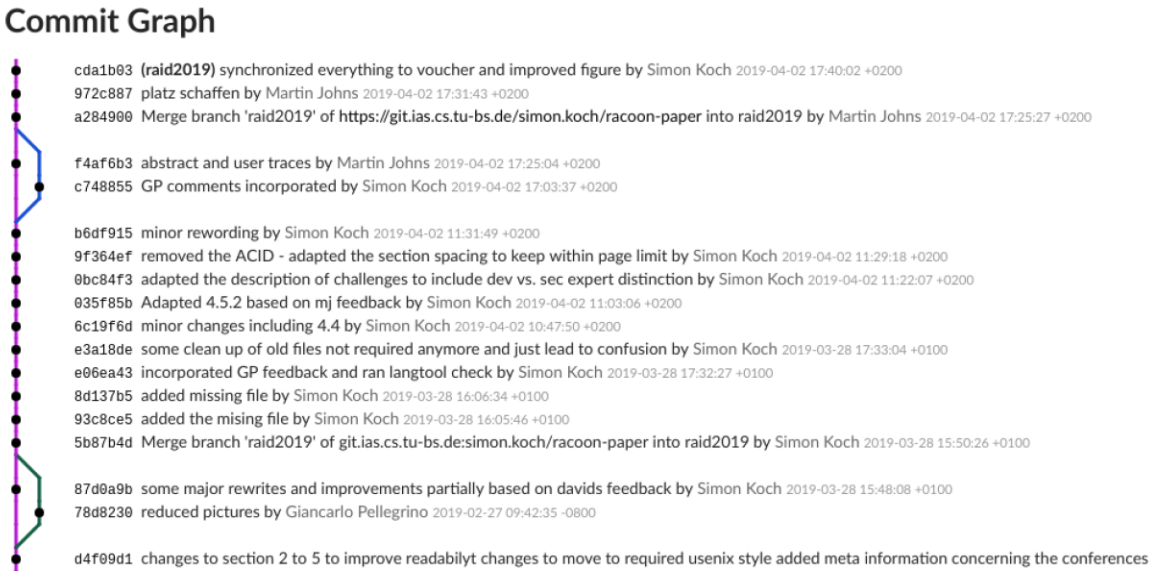

Abbildung 1: Beispielhafter Commit-Graph, bei dem zwischenzeitlich mehrere Versionen existierten, die wieder zu einer verschmolzen (gemergt) wurden.

Hinweis: Laden Sie ab hier all Ihre Dateien in der Ordner des entsprechenden Übungsblattes, also hier *blatt1* hoch. Andernfalls kann Ihre Abgabe nicht gewertet werden!

Pflichtaufgabe 2: Hello World in Java - (2 Punkte) Den Einstieg in eine neue Programmiersprache beginnt man traditionell mit einem einfachen kleinen Programm, dass nicht mehr macht als einen Text, meist *"Hello World!"* oder diverse Abwandlungen hiervon, darzustellen. Dadurch erlangt man einen ersten Einblick in die Syntax der Sprache und den Aufbau eines Programms. Genau dies soll auch in dieser Aufgabe zu erledigen sein.

- Legen Sie die Datei `HelloWorld.java` an und fügen Sie in diese den folgenden Code ein.

```
1    public class HelloWorld {
2    public static void main(String[] args) {
```

```
3     System.out.println("Hello World!");
4     }
5     }
6
```

- Übersetzen Sie das Programm mit dem Java-Compiler (**javac**) und führen Sie es anschließend mit dem Java-Interpreter **java** in der Kommandozeile aus.

  ```
  javac HelloWorld.java
  ```

  ```
  java HelloWorld
  ```

- In der Kommandozeile können Sie nun die Ausgabe *Hello World!* sehen.
- Ändern Sie den auszugebenden Text in Ihrem Programm und führen Sie dieses erneut aus. (1 Punkt)

Auch, wenn Sie in diesem kurzen Programm viel sehen, was Sie jetzt noch nicht zuordnen können, kann man hier schon viel erkennen:

Ein *Java*-Programm startet immer mit der *Main*-Methode. Alles, was sich also zwischen diesen geschweiften Klammern befindet wird von oben nach unten ausgeführt. Immer, wenn Sie Ihr Programm starten. Diese Methode steht außerdem in einer Klasse. Wir werden das Konzept von Klassen später kennenlernen. Für die Ausgabe sorgt die Anweisung `System.out.println("text");`. Wichtig ist, dass wir hinter einer Anweisung immer ein Semikolon am Ende der Zeile finden. Das `String[] args` beschreibt das Array von Kommandozeilenparametern. In einer der folgenden Aufgaben werden wir damit arbeiten.

- Beobachten Sie welche Dateien während des Übersetzungsvorgangs angelegt werden. Überlegen Sie, welche Dateien Sie anderen Personen über Ihr Git zur Verfügung stellen müssten, damit diese selbstständig das Programm übersetzen und ausführen können. Speichern Sie dafür Ihre *HelloWorld*-Dateien in Ihrem lokalen Git Repository ab.
- Git erlaubt es uns bestimmte Dateien vom Hochladen auszuschließen. Recherchieren Sie dafür das Konzept der *.gitignore*-Datei. Eine solche Datei können Sie bereits in Ihrem Repository finden. Erstellen Sie eine Datei, in der Sie reinschreiben, in welcher Zeile die .gitignore wie festlegt, dass keine kompilierten Java Dateien in Ihr Repository hochgeladen werden. (1 Punkt)

Pflichtaufgabe 3: Das Leben erleichtern durch Programmieren - (5 Punkte) In dieser Aufgabe wollen wir unser erstes Programm schreiben, mit dem wir uns die Macht des Programmierens zu Nutze machen wollen. Wir programmieren einen kleinen aber einfachen Taschenrechner. Zwar gibt es Taschenrechner wie Sand am Meer, jedoch wollen wir durch dieses Programm wichtige Grundlagen lernen, die wir für komplexere Programme dringend verinnerlichen müssen.

- Erstellen Sie ein neues *Java*-Programm mit dem Namen *Calculator.java*.
- Erstellen Sie eine neue Klasse und die *main*-Methode wie in Aufgabe 2.
- Legen Sie nun drei Variablen an, die für die zwei Zahlen, die verrechnet werden und das Rechensymbol stehen. Unser Taschenrechner soll in seiner ersten Form nur zwei Zahlen + oder - rechnen können. Überlegen Sie sich, welche Variablentypen Sie nutzen wollen. Eine Variable legen Sie wie folgt an: `Typ name = optionaler-startwert;` (1 Punkt)
- Fragen Sie mit Hilfe einer *if*-Abfrage ab, welche Rechenoperation der Taschenrechner verwenden soll und geben Sie daraufhin die Lösung auf der Kommandozeile aus. (1 Punkt)
- Da die Verwendung von Hartkodierten Werten im Quelltext sich nicht wirklich für einen Anwender eignet, der nicht ständig das Programm neu kompilieren soll, wollen wir nun die Kommandozeilen-Parameter verwenden, um die durchzuführende Berechnung anzugeben. Wenn Sie ihr Programm starten `java Calculator <ersteZahl> <zweiteZahl> <Rechenoperation>`, werden `<ersteZahl>` und die weiteren angegebenen Parameter dem Programm übergeben. In Ihrer *main*-Methode finden Sie die Liste von Zeichenketten (das String-Array) *args*. Sie können auf die einzelnen Werte so zugreifen: `Typ variablenname = args[0]`. Die vorherige Zeile nimmt den ersten Wert aus der Liste der Kommandozeilen-Parameter und speichert ihn in einer neuen Variable. Lesen Sie nun alle drei Parameter ein und lassen Sie den Taschenrechner die Lösung Ihrer Anfrage berechnen. Beachten Sie hierbei, dass gegebenenfalls Typenkonvertierungen notwendig sein könnten. (2 Punkte)
- Erstellen Sie für die folgende Aufgabe die *Calculator2.java*-Datei, damit die Bewertung der Aufgabe erleichtert wird. Erweitern Sie Ihr Programm um die Rechenmethoden * und /. Verwenden Sie zur Unterscheidung der Rechenoperation un anstelle einer *if*-Abfrage einen *switch-case*-Block. (1 Punkt)
- Denken Sie daran, dass Ihre Programme nur bewertet werden können, wenn Sie diese in Ihrem Git verwalten.

Pflichtaufgabe 4: Ihr erstes Computerspiel - (5 Punkte) In dieser Aufgabe sollen Sie ein kleines Ratespiel programmieren. Und zwar gleich auf zwei verschiedene Weisen. Da Sie zu zweit arbeiten, sollen Sie nun die Arbeit aufteilen. Eine Person soll den ersten Teil, die andere den zweiten Teil bearbeiten. Wir lernen dafür eine weitere Git Funktion kennen, die hierfür sehr nützlich ist.

Für die Implementierung von Features oder zur Programmierung von bestimmten Teilen des Programms verwendet man für gewöhnlich eine eigene **Branch** (von engl. branch - Ast, Zweig, Abschnitt). Derzeit hat Ihr git nur den *master*-Branch, auf der stets die eine Version eures Codes liegt, bei der Sie gerade sind. Jedoch sollte dieser *branch* nur wirklich

stabilen Code enthalten. Alles andere machen Sie auf Ihrem eigenen **Feature-Branch**, die am Ende, wenn das Feature fertig oder der bestimmte Fehler behoben ist, wieder in den Master *gemergt* wird. Hier soll also nun jeder Teilnehmer Ihrer Gruppe einen neuen *branch* für das Feature, das er implementieren wird erstellen. Dies können Sie mit `git branch <feature-name>` durchführen. Dabei erstellen Sie eine Spiegelung des aktuellen Stands des *master-branches*, auf der Sie nun solange weiterarbeiten, bis Sie Ihr Feature soweit implementiert haben, dass Sie es wieder mit dem *master* zusammenführen können. Beachten Sie beim Hochladen ins Git, dass Sie in Ihrem Branch arbeiten und dass Sie erst nach Fertigstellung des Features einen Merge durchführen. Für das Durchführen des finalen *Merge* gibt es diverse Möglichkeiten. Sie können dies in der Kommandozeile oder in der grafischen Oberfläche online tun.

Hinweis: Bitte arbeiten Sie allein an Ihrem Branch. Jeder Gruppenteilnehmer bekommt nur die Punkte für die Implementierung einer der beiden Branches! Am Ende bekommen Sie beide noch einen Punkt auf das erfolgreiche Mergen.

Branch 1: Der Computer rät und die main: In diesem Branch soll zuerst die *main*-Methode implementiert werden. Denken Sie daran, dass Sie und Ihr Partner in der gleichen Datei in Ihren lokalen Repositories arbeiten, die Sie am Ende verschmelzen. Mit `args.length` bekommen Sie die Anzahl der Kommandozeilenparameter, mit dem das Programm gestartet wurde. Wurde mindestens ein Parameter eingegeben, soll eine Methode `computerRaten(int arg)` aufgerufen werden, die die Zahl, die der Computer raten soll (also den übergebenen Kommandozeilenparameter) übergibt bekommt. (1 Punkt) Als Kommandozeilenparameter geben Sie beim Starten des Programms eine Zahl zwischen 1 und 100 ein.

Dann soll **der Computer** die von Ihnen eingegebene Zahl wie folgend erraten:

- Der Computer soll als erstes eine Zufallszahl generieren und diese mit der übergebenen Zahl vergleichen. (1 Punkt) Ist die Zahl korrekt, soll die Anzahl der vom Computer benötigten Versuche ausgegeben werden. (1 Punkt)
- Ist die Antwort falsch, soll der Computer es mit der nächsten zufälligen Zahl versuchen. Dies wird solange wiederholt, bis der Computer die richtige Zahl errät. Nutzen Sie hierfür das Konzept von Schleifen, sowie die *if*-Abfragen aus der vorherigen Aufgabe. (1 Punkt)

Eine Zufallszahl zwischen 0.0 (inklusive) und 1.0 (exklusive) kann wie folgt generiert werden: `double zahl = Math.random();`.

Branch 2: Selber raten: Beim Start des Programms ohne eine Angabe von Kommandozeilenparametern soll eine andere Methode `selberRaten()` aufgerufen werden. (1 Punkt) Diese Methode soll dann über folgende Funktionalität verfügen.

- Erzeugen Sie eine Zufallszahl zwischen 1 und 100 und speichern Sie diese in einer geeigneten Variable. (1 Punkt)
- Nun soll der Spieler ein Zahl raten. Dies kann mit einem Scanner realisiert werden.

```
1    Scanner sc = new Scanner(System.in);
2    int nextNumber = sc.nextInt();
3
```

  Zeile 1 initialisiert hier einen neuen Scanner, der Eingaben der Tastatur einliest. In Zeile 2 wird dann ein eingegebener Wert in der Variable *nextNumber* gespeichert. Beobachten Sie, was bei einer fehlerhaften Eingabe passiert. Damit werden wir uns in späteren Übungen noch ausführlicher befassen.

- Nutzen Sie nun eine Schleife, die solange den Spieler zu einem erneuten Eingeben einer Zahl auffordert, bis der Spieler die zufällig vom Computer generiert Zahl erraten hat. (1 Punkt)
- Geben Sie nach Eingabe der Zahl auf der Kommandozeile aus, ob die vom Spieler eingegebene Zahl größer oder kleiner als die zu erratende Zahl ist. (1 Punkt)

Gemeinsame Sache - Aus 2 werde 1: Nun haben Sie beide einen Branch mit Ihren Features in Ihrem Git Repository. Verschmelzen Sie nun hintereinander Ihre beiden Branches mit dem *master*-branch, sodass Sie dort Ihr finales gemeinsam programmiertes Spiel haben.

Sie haben nun ihr erstes kleines Programm erstellt und gelernt, wie man gemeinsam an einem Projekt arbeiten kann. Viel Erfolg bei den weiteren Aufgaben. (1 Punkt)

Bonus zur Übung: Überlegen Sie sich, wie Sie das Rateverfahren des Computers verbessern können. Testen und evaluieren Sie Ihren Ansatz anhand Ihrer Simulation und der Anzahl der benötigten Versuche. Vielleicht schaffen Sie es ja den Computer cleverer Raten zu lassen, als Sie selbst?!