

# 1. Praktikum

## Java und Eclipse



Bearbeitungszeitraum: **31.10. - 14.11.22, 9.15 Uhr**

Gesamtpunktzahl: **5 Abgabepunkte + 6 Leistungspunkte**

Praktikumsziel: Im ersten Praktikum sollen Sie sich mit den grundlegenden Werkzeugen vertraut machen, die für alle weiteren Aufgaben zum Einsatz kommen:

- die Entwicklungsumgebung *eclipse*
- der Java-Compiler und der Java-Interpreter

Dabei implementieren Sie in Java einen allerersten (klassischen) Algorithmus.

Voraussetzungen: **Account am Institut - damit Sie uneingeschränkt arbeiten können**

Account bei Moodle und Registrierung für dieses Praktikum - damit Sie Lösungen dieser und aller weiteren Aufgaben abgeben können

Zugang zu den gruenau-Rechnern am Institut oder eigene (lokale) Installation von eclipse .

Aufgabe: Experimentieren Sie mit *eclipse* und dem Java-Compiler/-Interpreter (wie unten erläutert) und implementieren Sie dabei den Euklidischen Algorithmus [http://de.wikipedia.org/wiki/Euklidischer\\_Algorithmus](http://de.wikipedia.org/wiki/Euklidischer_Algorithmus) zur Bestimmung des größten gemeinsamen Teilers zweier positiver ganzer Zahlen in Java.

FAQ: Am Ende dieser Seite finden Sie die am häufigsten gestellten Fragen sowie (spätere) Konkretisierungen der Aufgabenstellung. Sie sind Teil der Aufgabenstellung und könnten sich während des Bearbeitungszeitraumes ändern, wir werden Sie ggf. darüber im Forum zu dieser Veranstaltung informieren.

Sollten Sie sich die Aufgabenstellung ausdrucken, so achten Sie bitte darauf, dass sich die FAQ weiterhin ändern können.

Abgabe: Senden Sie die Datei **GGT.java** über **Moodle** ein.  
Diese Aufgabe ist noch eine Einzelabgabe, d.h. **jeder muss seine Lösung bei Moodle separat abgeben!**

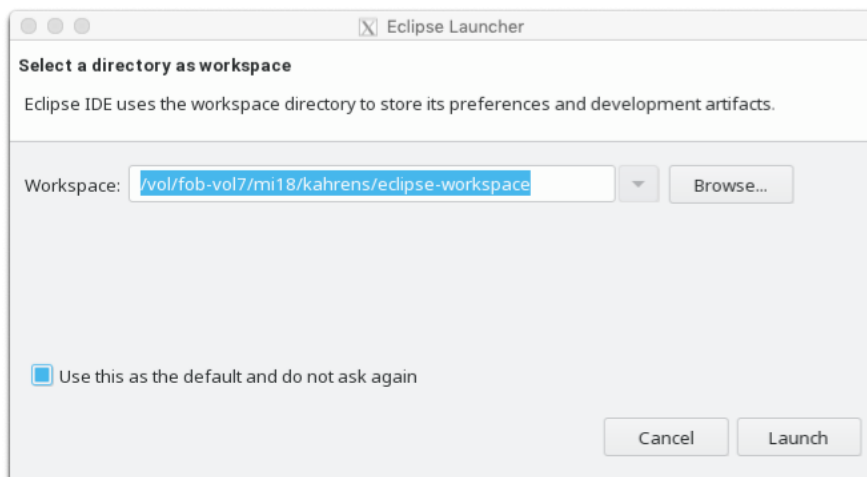
## Hinweise

Zunächst wollen wir Ihnen zeigen, wie Sie mit *eclipse* arbeiten. Dabei handelt es sich um eine Entwicklungsumgebung für die komfortable Programmierung in Java (und vielen anderen Sprachen), die selbst in Java implementiert ist und in verschiedenen Versionen auf allen

üblichen Plattformen (Windows, Mac, Linux, ...) verfügbar ist. Auf unserem Referenzsystem für dieses Praktikum (openSUSE Leap 15.3) ist die Version 2020-09 (4.17.0) installiert. Sie können auf ihrem heimischen Rechner irgendeine Version installieren, ohne dass dies für das Praktikum einen wesentlichen Unterschied macht. Möglicherweise sehen dann die Fenster und Menüs etwas anders aus, ohne dass das für unseren Anwendungsfall von großer Bedeutung ist. Falls Sie dabei Hilfe brauchen, wenden Sie sich bitte an Ihre Kommilitonen ggf. aus höheren Semestern (z.B. über die Fachschaft). Starten Sie *eclipse* zum, indem Sie in einem Terminalfenster

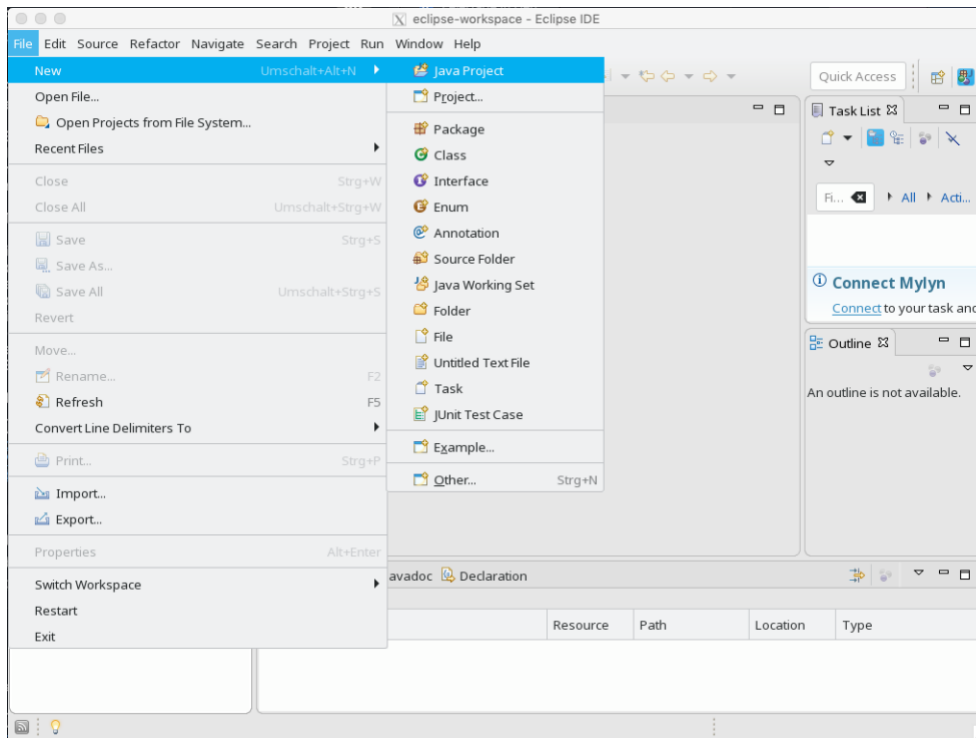
### ***eclipse***

eingeben. (Die Warnungsmeldungen im Terminalfenster, die ggf. erscheinen, können Sie einfach ignorieren.) *eclipse* empfängt Sie nach einem Startfenster mit einer Anfrage wie dieser:

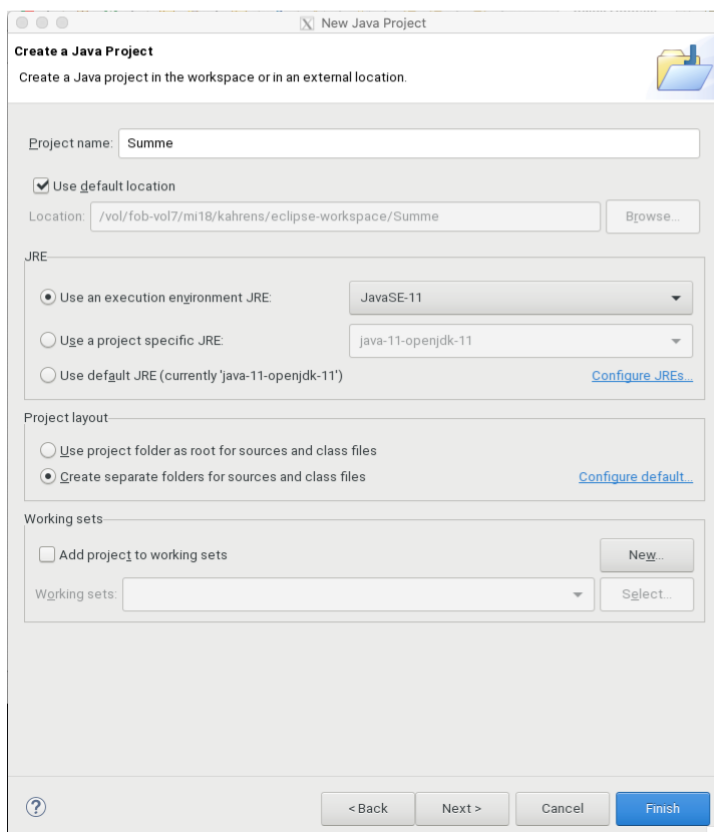


*eclipse* schlägt vor, in Ihrem Home-Verzeichnis ein Unterverzeichnis *eclipse-workspace* anzulegen. In diesem wird es fortan all Ihre *eclipse*-Projekte (jeweils in einem Verzeichnis unterhalb von *eclipse-workspace*) anlegen. Wenn Sie, wie gezeigt, den Haken bei *Use this as the default ....* setzen, wird später nicht erneut nachgefragt. Nach Klick auf *Launch* erscheint ein Welcome-Fenster, welches Sie im Moment gleich wieder schließen sollten, indem Sie auf das X im Tab *Welcome* klicken. Später können Sie sich in Ruhe die hier hinterlegten Tutorials und Beispiele ansehen, indem Sie im Menü *Help -> Welcome* auswählen.

Danach erscheint die noch leere Arbeitsoberfläche von *eclipse*. Ein neues (Java-) Projekt legen Sie an durch die nachfolgend angezeigte Menüauswahl:

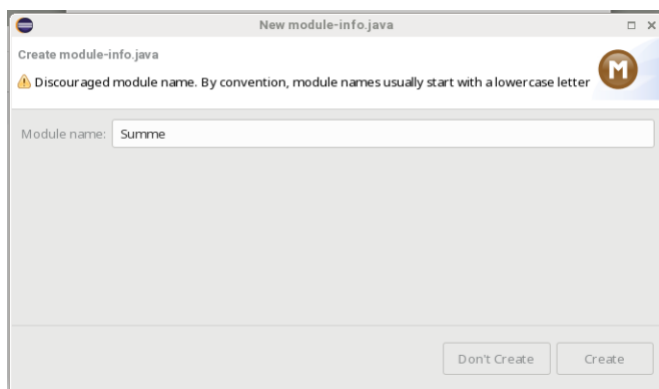


*eclipse* fragt nun nach dem Namen des Projektes und seinen weiteren Eigenschaften. Nennen Sie dieses Projekt *Summe*, weil es zunächst darum geht, die Summe zweier ganzen Zahlen mit einem vorgegebenen Quelltext zu berechnen. Anhand dieses Projektes soll nun erläutert werden, wie man ein Java-Programm in *eclipse* zur Ausführung bringt und insbesondere, wie man dieses dabei mit Programmargumenten versorgt. Später sollen Sie dann für die eigentliche Aufgabe natürlich ein eigenes neues Projekt anlegen.

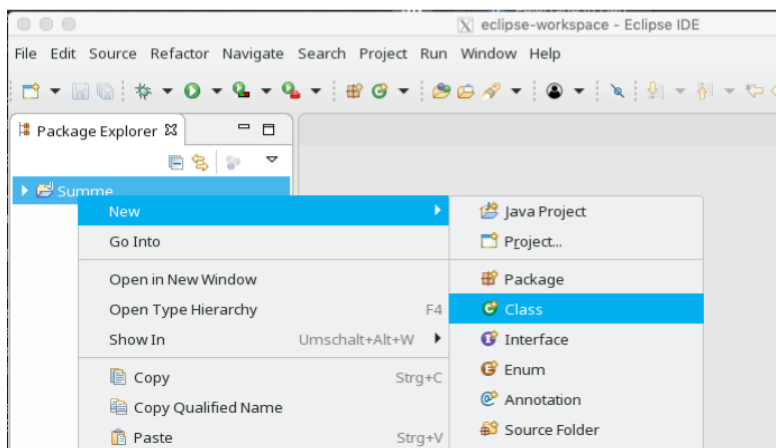


Wählen Sie (auch später für neue Projekte) immer einen Namen, der Ihnen die Zuordnung leicht ermöglicht. Auch Leerzeichen sind im Prinzip erlaubt - das einzige Problem mit diesen ist die Tatsache, dass (wie zu sehen) ein gleichbenanntes Unterverzeichnis in *eclipse-workspace* angelegt wird und Dateien mit Leerzeichen im Namen unter Unix nicht besonders handlich sind. Belassen Sie ansonsten die gezeigten Voreinstellungen. Die Vorauswahl (s.o.) bei *Create separate folders* bewirkt, dass unterhalb des neuen Projektverzeichnisses noch einmal zwei separate Verzeichnisse erzeugt werden: *src* (wie *source* - für ihre lesbaren Java-Quelltexte) und *bin* (wie *binary* - für die daraus durch Übersetzung erzeugten, binären Bytecodes). Nach Klick auf *Finish* erscheint ein neues Fenster wie dieses:

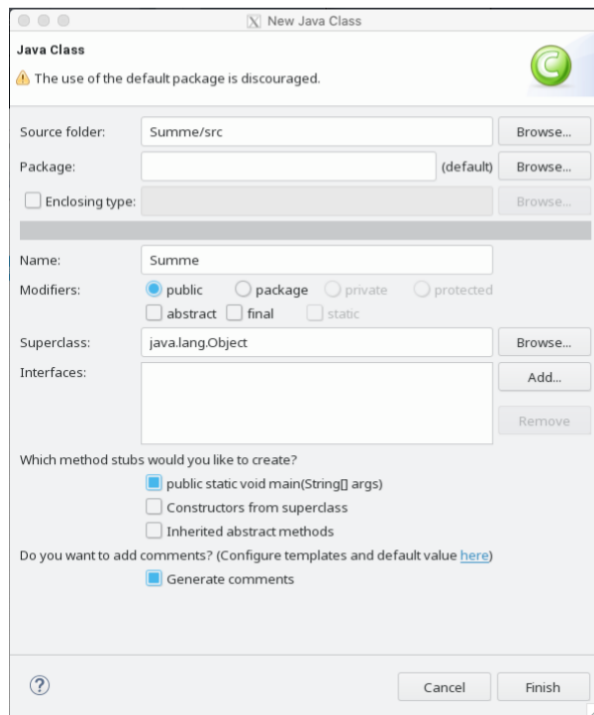
Unser *eclipse* benutzt eine Java-Version, die ein Feature namens „Module“ unterstützt, welches wir aber weder im Praktikum noch in der Vorlesung benötigen. Klicken Sie daher in dem Fenster auf *Don't create*.



Daraufhin sehen Sie im sog. *Project Explorer* Ihr neues (noch leeres) Projekt.



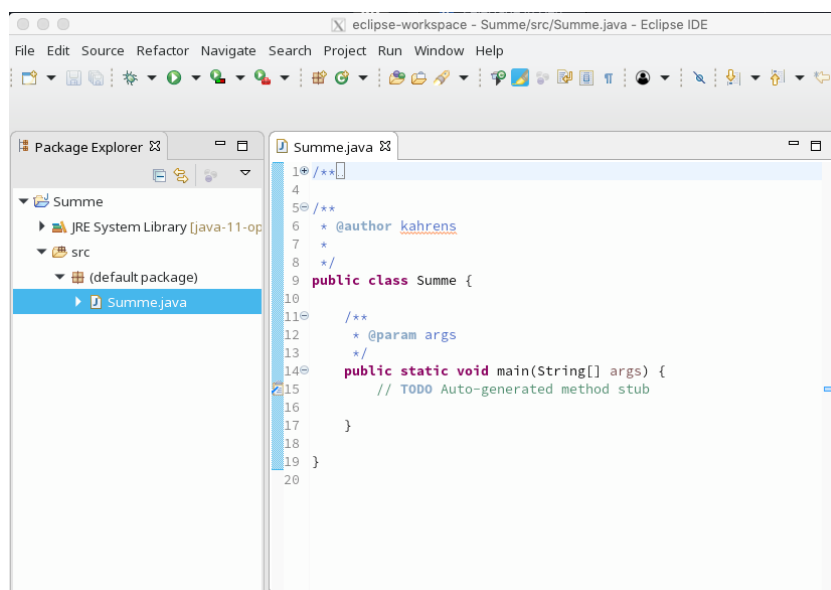
Nun können Sie (wie oben zu sehen) einen neuen Java-Quelltext zu Ihrem Projekt hinzufügen - in Java erscheinen solche immer in Form einer (manchmal auch mehrerer) Klassen. Daher wählen Sie *New* und *Class* aus. Diese Klasse (und der gleichnamige Quelltext) braucht nun noch einen Namen, den Sie im folgenden Fenster festlegen können:



Sie sehen, dass der Quelltext im *src*-Verzeichnis des aktuellen Projektes angelegt wird, der Name der Klasse ist hier ebenfalls *Summe*. Wählen Sie die weiteren Eigenschaften der neuen Klasse, wie angezeigt. Lassen Sie insbesondere das Feld *Package* leer (trotz der möglicherweise angezeigten Warnung).

Da die neue Klasse als Programm ausführbar sein soll, braucht Sie eine *public static void* Methode *main*. Die Auswahl von *Generate comments* erzeugt Kommentar-Fragmente im Quelltext, die später der Ausgangspunkt einer bequem handhabbaren Dokumentation der Klasse sein wird.

Nach erneutem Klick auf *Finish* haben Sie: die neue Klasse im Projekt-Explorer und deren (rudimentären) Quelltext im Editor-Fenster:



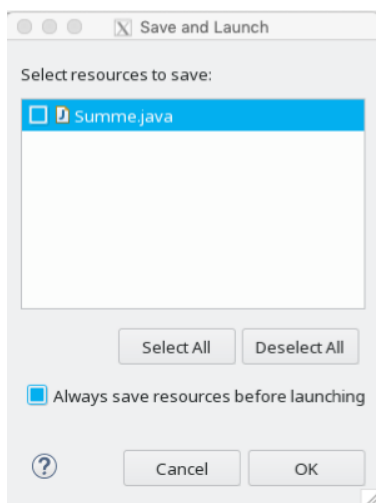
Dieser (automatisch erzeugte) Quelltext ist schon ein fehlerfreies und ausführbares Java-Programm, welches allerdings noch nichts wirklich tut, sondern nach seinem Start die (noch) leere *main*-Methode ruft und sich danach beendet. Probieren Sie es aus, indem Sie auf den Run-Knopf klicken:



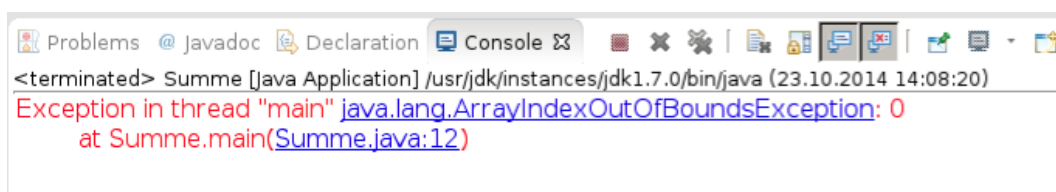
Um daraus ein Programm zu machen, welches wie versprochen zwei Zahlen addiert, sollten Sie nun den Quelltext so ändern, dass er dieser Vorgabe entspricht:

```
class Summe {  
    public static void main(String args[]) {  
        int x = Integer.parseInt(args[0]);  
        int y = Integer.parseInt(args[1]);  
        int summe = x + y;  
        System.out.println(x + " + " + y + " = " + summe);  
    }  
}
```

Wenn Sie nun erneut auf *Run* drücken, fragt *eclipse* (falls Sie diese letzten Quelltextänderungen noch nicht mit *<ctrl>S* gesichert haben) nach, ob diese jetzt gesichert werden sollen und ob das (sinnvollerweise) immer passieren soll.



Diesmal werden im sog. Console-Fenster (unterhalb des Editor-Fensters) tatsächlich Ausgaben erscheinen. Allerdings weisen diese auf ein abnormales Programmende (einen sog. Laufzeitfehler) hin, der darin besteht, dass das Programm auf Argumente zugreift, die gar nicht übergeben wurden.



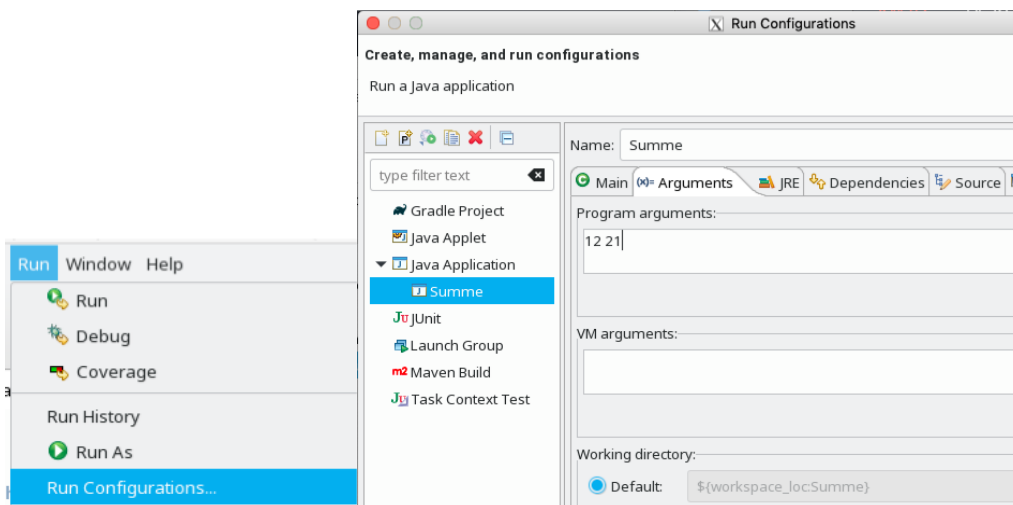
Sie sollten nun einmal (ohne *eclipse* zu verlassen) versuchen, genau dieses Programm von der Kommandozeile zu starten. Wechseln Sie dazu in einem (ggf. neuen) Terminalfenster in das entsprechende Verzeichnis, überzeugen Sie sich davon, dass sich dort der Bytecode der

Übersetzung aus *eclipse* befindet und starten Sie das Programm (ohne und mit Parameter). Dabei sind Ihre Eingaben fett hervorgehoben:

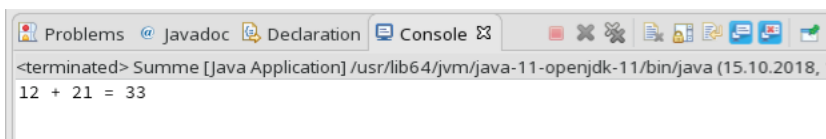
```
$ cd ~/eclipse-workspace/Summe/bin
$ ls
Summe.class
$ java Summe
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0 at
Summe.main(Summe.java:12)
$ java Summe 21
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1 at
Summe.main(Summe.java:13)
$ java Summe 21 12
21 + 12 = 33
```

Sie sehen: natürlich braucht das Programm auch auf der Kommandozeile die geforderten Argumente, man gibt diese einfach nach dem Namen der Klasse an, aber ein Argument reicht hier nicht. Erst wenn das Programm mit (mindestens) zwei Argumenten (die zudem als Zahlen lesbar sein müssen) gerufen wird, tut es das Verlangte.

Innerhalb von *eclipse* wird das Programm auf folgende Weise mit Argumenten versorgt. Klicken Sie auf *Run -> Run Configurations ...* und wählen Sie im folgenden Fenster den Tab *Arguments* aus und tragen Sie dort die Argumente ein:



Wenn Sie dann noch in diesem Fenster auf *Run* klicken, wird das Programm beim Start mit diesen Argumenten versorgt und tut nun (auch in *eclipse*) das Gewünschte:



Sollte übrigens einmal die Programmausführung keine (oder nicht die gewünschten Ausgaben) erzeugen, so kann das u.a. daran liegen, dass Ihr Programm (durch fehlerhafte Programmierung) in einer Schleifenanweisung steckenbleibt und diese Schleife immer wieder durchläuft. Sie erkennen das in *eclipse* daran, dass im Console-Fenster dieses

Quadrat  dunkelrot bleibt. Mit einem Klick darauf kann man das Programm dann

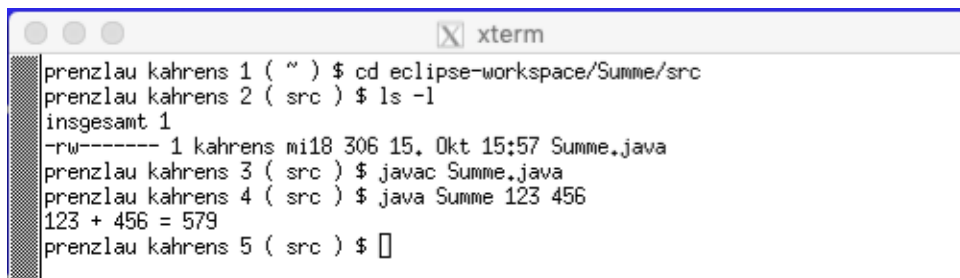
anhalten. Auf der Kommandozeile kann man ein solches Programm mit `^C` (`<ctrl>C`) abbrechen.

Sie beenden *eclipse*, indem Sie auf *File* und Exit klicken. Auch hier fragt *eclipse* nach, ob evtl. letzte Änderungen noch gesichert werden sollen. Wenn Sie *eclipse* das nächste Mal starten, wird der zuletzt hinterlassen Zustand erneut angezeigt. Dies ist möglich, weil

1. *eclipse* alle Projekte und ihren Bearbeitungszustand unterhalb von *eclipse-workspace* abgelegt hat und
2. in einem (normalerweise nicht sichtbaren) Verzeichnis *.eclipse* (ja, mit Punkt am Anfang) all Ihre nutzerspezifischen *eclipse*-Einstellungen abgelegt werden.

Probieren Sie auch das aus (Beenden und Neustart).

Eine Entwicklungsumgebung wie *eclipse* ist bei der Programmentwicklung, bei der Fehlersuche und dem Testen eine enorme Hilfe. Für dieses Praktikum sind allerdings als Abgabe jeweils nur die Java-Quelltexte abzugeben, die dann auch ohne *eclipse* (also auf der Kommandozeile) wie erwartet funktionieren sollen. Dazu ist es für Sie noch wichtig zu erfahren, wo sich die Quelltexte befinden und wie man sie auf der Kommandozeile übersetzen und ausführen kann. Die erste Frage ist durch die obigen Erläuterungen eigentlich schon geklärt, wir zeigen Ihnen hier noch einmal den Weg zu dem Quelltext des gezeigten Beispielprojektes:



```
prenzlau kahrens 1 ( ~ ) $ cd eclipse-workspace/Summe/src
prenzlau kahrens 2 ( src ) $ ls -l
insgesamt 1
-rw----- 1 kahrens mi18 306 15. Okt 15:57 Summe.java
prenzlau kahrens 3 ( src ) $ javac Summe.java
prenzlau kahrens 4 ( src ) $ java Summe 123 456
123 + 456 = 579
prenzlau kahrens 5 ( src ) $
```

Wenn Sie ein neues Terminalfenster starten, befinden Sie sich in Ihrem Home-Verzeichnis (`~`). Nach einem Wechsel (`cd` - change directory) nach *eclipse-workspace/...* erkennen Sie auch am sog. Prompt der Shell (dem Programm, welches Ihre Kommandos liest und verarbeitet) das neu eingestellte Verzeichnis. Weiter im Dateibaum abwärts kommt dann das Projektverzeichnis (*Summe*) und dessen Quell-Unterverzeichnis *src*. Dort befindet sich der eigentliche Java-Quelltext unseres Projektes. Mit `ls -l` wird der Inhalt des aktuellen Verzeichnisses im long-Format angezeigt, d.h. mit zusätzlichen Informationen, wie Zugriffsrechte, Eigentümer, Größe in Bytes, Zeitpunkt des letzten schreibenden Zugriffs und Name der Datei.

Der Java-Compiler heißt *javac* und bekommt den zu übersetzenden Quelltext als Argument übergeben. Falls die Übersetzung fehlerfrei ist (wie hier) erfolgt sie ohne weitere Ausgaben. Es entsteht jedoch, wie Sie sehen können, im aktuellen Verzeichnis eine neue Datei mit dem gleichen Namen wie der Quelltext und der Dateiendung *.class* - dies ist der binäre Bytecode. *eclipse* würde ihn im *bin*-Verzeichnis ablegen. Die Ausführung eines Bytecodes erfolgt durch Aufruf des sog. Java-Interpreters *java*, wobei die auszuführende Klasse (hier *Summe*) als Argument zu benennen ist. Der Interpreter sucht automatisch im aktuellen Verzeichnis nach



einem *class*-File passend zur Klasse, lädt dieses in den Speicher und führt die *main*-Methode aus.

Sie sollten sicher IMMER vergewissern, dass der Quelltext, den Sie bei Moodle als Lösung einer Aufgabe einsenden, auf unserer Referenzplattform (gruenau6) von der Kommandozeile übersetzbar ist und dass die Ausführung (ebenfalls von der Kommandozeile) das erwartete Verhalten zeigt. Auf diese Weise werden Ihre Lösungen bei der Bewertung geprüft. Alle Poolmaschinen sind identisch installiert (aktuell mit dem Java-Compiler aus dem genannten *openjdk*: *javac 11.0.16*). Wenn Sie sich explizit auf eine benannte Referenzmaschine bei Ihren abschließenden Tests beziehen wollen, verwenden Sie bitte [gruenau6.informatik.hu-berlin.de](http://gruenau6.informatik.hu-berlin.de), Auf diesem Rechner werden ggf. Streitfälle [„... bei mir ließ es sich aber übersetzen“] entschieden.

## Aufgabe

Erstellen Sie (am besten in einem neuen *eclipse*-Projekt) ein Java-Programm **GGT.java**, welches (in Anlehnung an die demonstrierte Berechnung der Summe zweier Zahlen) den größten gemeinsamen Teiler von zwei positiven ganzen Zahlen nach dem Euklidischen Algorithmus berechnet und ausgibt.

Das Prinzip des Euklidischen Algorithmus wird auch gegenseitige Wechselwegnahme genannt. Eingangsgrößen seien zwei positive ganze Zahlen *a* und *b*. Bei der Berechnung verfährt man nach Euklid wie folgt:

Das Prinzip des euklidischen Algorithmus wird auch <i>gegenseitige Wechselwegnahme</i> genannt. Eingangsgrößen sind zwei natürliche Zahlen <i>a</i> und <i>b</i> . Bei der Berechnung verfährt man nach Euklid wie folgt:	
1.	Setze $m = a$ ; $n = b$ .
2.	Ist $m < n$ , so vertausche <i>m</i> und <i>n</i> .
3.	Berechne $r = m - n$ .
4.	Setze $m = n$ ; $n = r$ .
5.	Ist $r \neq 0$ , so fahre fort mit Schritt 2.
Nach Ablauf des Verfahrens hat man mit <i>m</i> den ggT von <i>a</i> und <i>b</i> gefunden.	

Überzeugen Sie sich durch geeignete Tests von der Korrektheit Ihrer Implementation. Versuchen Sie auch den Algorithmus strukturell so zu formulieren, dass man die obige verbale Beschreibung wiedererkennen kann. Sorgen Sie dafür, dass im Programm geprüft wird, ob die beiden als Argumente übergebenden Zahlen tatsächlich echt positiv (also  $> 0$ ) sind. Ansonsten funktioniert der oben beschriebene Ablauf vielleicht gar nicht richtig!? Das macht man in Java beispielsweise so:

```
// die Bedingung muessen Sie natuerlich geeignet formulieren:
if (someCondition) {
    System.out.println("eine passende Ausschrift");
    System.exit(-1); // Programm sofort beenden
    // negative Parameter bei exit weisen den Rufer
    // auf ein abnormales Ende hin
}
```

Abschließend hier noch ein paar Beispielaufrufe, die zeigen, wie sich Ihr Programm verhalten soll - offenbar müssen Sie sich NICHT darum kümmern, dass tatsächlich zwei Zahlen als Argumente übergeben werden.

```
$ java GGT 12 14
ggT(12, 14) = 2
$ java GGT 36 12
ggT(36, 12) = 12
$ java GGT 36 -12
nur positive ganze Zahlen als Argumente erlaubt
$ java GGT -36 12
nur positive ganze Zahlen als Argumente erlaubt
$ java GGT 666
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 1
    at GGT.main(GGT.java:6)
$ java GGT 6 sieben
Exception in thread "main" java.lang.NumberFormatException: For input string:
"sieben" at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:492)
    at java.lang.Integer.parseInt(Integer.java:527)
    at GGT.main(GGT.java:6)

(dabei können die angegebenen Zeilennummern 65, ... natürlich andere sein: sie beziehen sich auf Ihren Quelltext
und auf die jeweilige Implementation der Java-Bibliothek)
```

## ACHTUNG

Um die Korrektur der Aufgaben durch die Tutoren zu erleichtern, gelten (auch bei allen weiteren Aufgaben) die folgenden Regeln:

Eingesandte Java-Programme, die nicht vom Java-(Referenz-)Compiler akzeptiert werden (d.h. Fehler bei der Übersetzung liefern) werden ohne weitere Inspektion mit 0 Punkten bewertet (keine Abgabepunkte und erst recht keine Leistungspunkte)!

Halten Sie sich bei der Benennung von Klassen, Funktionen und Dateien immer an die Vorgaben und achten Sie darauf, dass Sie **keine** Packages verwenden (nur *default package*). Alle Abweichungen von diesen Regeln führen zu zusätzlichem Aufwand bei der Bewertung und daher auch zu Punktabzug. Vergessen Sie nicht den abschließenden Test auf der Referenzplattform!

Die vom implementierten Programm erzeugten Ausgaben müssen das geforderte Format exakt einhalten (incl. Zeilenstruktur und Leerzeichen). Soll und Ist werden automatisch verglichen. Sofern Abweichungen festgestellt werden, gibt es Punktabzug, über dessen Höhe dann die Tutoren entscheiden! Die einzig zulässige Abweichung (eine die man bei der Textansicht des Programms gar nicht feststellen kann) besteht in der Codierung der Zeilenenden, die unter Windows mit zwei Zeichen (carriage return [/x0d] + line feed [/x0a]) unter Unix dagegen nur mit einem Zeichen (newline [/x0a]) dargestellt werden.

## FAQ

1. Q:

Was sind denn Abgabe- und Leistungspunkte?

A:

Wenn Sie die [Praktikumsordnung](#) gelesen haben, sollte klar sein, dass Sie alle Aufgaben abgeben müssen. Die Abgabepunkte erhalten Sie dann, wenn Sie eine erkennbar ernst gemeinte Abgabe eingereicht haben. Die eigentlich in der jeweiligen Aufgabe geforderte Leistung wird dann in Abhängigkeit von der Erfüllung mit Leistungspunkten bewertet.

**Bitte schauen Sie vor der Abgabe immer noch einmal online in die FAQs - falls es Korrekturen oder Modifikationen an der Aufgabestellung geben sollte, werden diese dort explizit erläutert!**

Das war die 1. (richtige) Praktikumsaufgabe! - Viel Erfolg bei der Bearbeitung

Letzte Änderung 27. Oktober 2022

Inhaltliche Fragen/Korrekturen bitte an: [Ahrens](#)