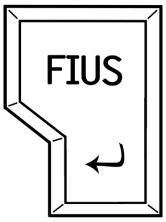
Universität Stuttgart



Übungsblatt 1

Java Vorkurs(WS 2021)

Aufgabe 1: Programmstart

How-To: Wie bekomme ich des Projekt

1. Lade zuerst das Maven-Projekt JavaVorkurs2020_EclipseProjekt.zip runter, man findet es hier:

   https://fg-inf.de/eclipse-project

2. Nachdem das Zip-File heruntergeladen wurde, muss man es in einem geeigneten Ordner entpacken.
   Windows: Entpacken funktioniert durch einen Rechtsklick auf die Datei und dann durch klicken auf Alle extrahieren... → Extrahieren.
   Linux: Am schnellsten entpackt man ein Zip-File über das Terminal mit dem Befehl:
   `> unzip JavaVorkurs2020_EclipseProjekt.zip`
   Apple: Nachdem man das Zip-File im Finder offen hat, entpackt man es durch einen einfachen Doppelklick.

HowTo: Projekt Import in Eclipse

1. Um ein Projekt zu importieren, klicke zuerst auf File → Import....
2. Wähle in der Auswahl Maven → Existing Maven Projects oder nutze das Suchfeld oben um Existing Maven Projects zu finden. Klicke dann auf Next >.
3. Drücke oben rechts auf Browse... und suche das Verzeichnis, in welchem die Datei JavaVorkurs2020_EclipseProjekt.zip entpackt wurde.
4. Stelle sicher, dass der Projektname im *Projects* Bereich des Fensters auftaucht.
5. Zu guter Letzt noch auf Finish drücken.
6. Nachdem sich das Fenster geschlossen hat, siehst du das Projekt im *Package Explorer* links an der Seite.
7. Damit euer Projekt richtig funktioniert, solltet ihr im *Package Explorer* das Projekt mit einem Rechtsklick auswählen und dann im Kontextmenü Maven → Update Project... → OK ausführen.

Öffne nun die Entwicklungsumgebung Eclipse und importiere das heruntergeladene Maven-Projekt JavaVorkurs2020_EclipseProjekt.zip wie oben in den beiden HowTo's beschrieben.

a) Finde und öffne die `Main` Klasse in dem Package Explorer von Eclipse. Die Klasse ist in dem Ordner `src/main/java` und dann in dem Package `de.unistuttgart.informatik.fius.jvk` in der Datei `Main.java`. Führe das Projekt aus, indem du den grünen Play-Button (links oben in der Werkzeugleiste, rechts neben dem Button mit dem Käfersymbol) von Eclipse drückst. Es sollten keine Fehler auftauchen, aber auch nichts passieren. Wir haben ja auch noch nichts programmiert.

b) Finde in der `Main` Klasse die Zeile mit `// implement task 1 (from sheet 1)here` und füge an seiner Stelle den fehlenden Code aus dem Bild ein. Aktuell musst du den Code noch nicht verstehen, es geht darum den Code Editor in Eclipse kennen zu lernen. Achte also darauf was passiert während du die fehlenden Zeilen eingibst.

```
31 public class Main {
32
33     /**
34      * The main entry point of the project
35      *
36      * @param args
37      *     the command line args; not used
38      */
39     public static void main(String[] args) {
40         Game demoGame = new Game("Hello world", new DemoTask(), new DemoTaskVerifier());
41         demoGame.run();
42     }
43 }
```

Wenn das Programm jetzt ausgeführt wird geht ein Fenster mit der Simulator Ansicht auf. Suche nun die (rote) Stop Taste in Eclipse um das Programm abzubrechen. Die Taste befindet sich in Eclipse unten in der Titelleiste der Console.

c) Versuche nun den Code so zu verändern, dass dein Name im Fenstertitel steht.

HowTo: Auskommentieren

Manchmal möchten wir, dass bestimmter Code nicht ausgeführt wird. Vielleicht möchten wir ihn später ausführen, oder einfach nicht verlieren. In diesen Fällen Kommentieren wir die Code-Zeilen einfach aus. Dies geht, indem wir der Code-Zeile zwei Schrägstriche (*eng.* slashes)(`//`) voranstellen. Als Beispiel betrachten wir folgenden Code.

```
1 public void run(Simulation sim) {
2     PlayfieldModifier pm =
3             new PlayfieldModifier(sim.getPlayfield());
4     pm.placeEntityAt(new Coin(), new Position(0, 0));
5 }
```

Wenn wir nun die letzte Zeile auskommentieren wollen, dann stellt sich der Code wie folgt dar.

```
1 public void run(Simulation sim) {
2     PlayfieldModifier pm =
3             new PlayfieldModifier(sim.getPlayfield());
4     //pm.placeEntityAt(new Coin(), new Position(0, 0));
5 }
```

Alles ab den zwei Schrägstrichen bis zum Zeilen Ende wird nun vom Computer ignoriert, was durch eine andere Färbung in Eclipse deutlich wird.

Fehler

Einige Aufgaben in Blatt 1 (und auch den anderen Blättern) verlangen von euch, den Code absichtlich kaputt zu machen. Das Ziel von diesen Aufgaben ist es, dass ihr die Fehler die dabei auftreten schonmal in einem kleinen übersichtlichen Beispiel seht. Achtet also bei solchen Aufgaben darauf, welche Änderung zu welchem Fehler geführt hat. Später kann euch dieses Wissen helfen, wenn ihr nur den Fehler seht und besser versteht, nach was ihr im Code suchen müsst, um den Fehler zu beheben.
Ihr braucht auch keine Angst haben, dass ihr etwas nachhaltig kaputt machen könntet. Eclipse hat eine „Undo“ -Funktion die ihr im Menu oben unter edit (deutsch Bearbeiten) oder über Strg + z aufrufen könnt. Falls ihr trotzdem Hilfe braucht, könnt (*und sollt!*) ihr euch gerne bei den Tutoren melden.

d) Finde heraus was passiert, wenn man die erste oder die zweite Zeile in der Main-Funktion auskommentiert.

   Hinweis: Nicht alle Varianten funktionieren zwangsläufig, Überlege warum dies der Fall sein könnte.

e) Vielleicht sind dir beim Ausprobieren in a) schon einige Dinge aufgefallen die der Eclipse Editor anders macht als zum Beispiel Word. Wenn du `demoGame.` eingibst erscheint ein Overlay mit möglichen Autocompletions. Wenn du dann mit dem Cursor an eine andere Stelle gehst verschwindet das Overlay wieder. Öffne das Overlay manuell, indem du den Cursor direkt nach `demoGame.` platzierst und Strg + Space (Space ist die Leertaste) drückst.

f) Wenn du die Maus über `Game` bewegst, siehst du ein Overlay mit der Dokumentation. Das funktioniert auch an anderen Stellen. Was steht im Overlay von `DemoTask()`? Sollte sich das Overlay nicht öffnen, dann kannst du F2

g) Wenn du Strg + Shift + f (Shift ist die Taste über Strg, mit der man Großbuchstaben schreibt) drückst, formatiert Eclipse den Code für dich. Füge an verschiedenen Stellen neue Zeilen und Leerzeichen ein und beobachte was passiert, wenn du Strg + Shift + f drückst

Optionale Aufgaben

Aufgaben die mit **(Optional)** markiert, sind müssen nicht bearbeitet werden. Sie setzen schon Vorkenntnisse in Java oder Programmieren vorraus und sind deshalb auch oft deutlich schwerer als die normalen Aufgaben. Wenn du also an einer optionalen Aufgabe festhängst, dann solltest du mit der nächsten normalen Aufgabe weitermachen. Später, wenn du genug Zeit oder Wissen hast um die optionale Aufgabe zu lösen, kannst du nochmal zu ihr zurückkehren.

f) **(Optional)** Finde eine Möglichkeit, den Fenstertitel nach der `demoGame.run();` Zeile zu ändern. Dafür benötigst du den folgenden Code, den du aber noch auf deinen Namen anpassen musst: `demoGame.getGameWindow().setWindowTitle("");`

g) **(Optional)** Benenne `demoGame` in `myDemoGame` um. Wenn du nur die Stelle `Game demoGame` anpasst, wirst du eine Fehlermeldung bekommen. Mache deine Änderung nochmal rückgängig und platziere den Cursor in dem Wort `demoGame`. Mit der Tastenkombination Shift + Alt + r kommst du in den „Refactor“ -Modus, in dem du alle Vorkommen des Namens gleichzeitig umbenennen kannst. Das gleiche geht auch über Rechtsklick > Refactor > Rename.

   Mit Strg + Shift + l kannst du eine Liste alle Tastenkombinationen ansehen.

h) **(Optional)** Versuche, drei Fenster gleichzeitig zu starten.

Aufgabe 2: Einschub: Objekte, Klassen, Klassendiagramm

HowTo: Klassen und Objekte auseinander halten

In Java (und auch in anderen Programmiersprachen) gibt es **Klassen** und **Objekte**. Dabei gehört ein Objekt immer zu einer Klasse. Genauer gesagt ist ein Objekt **eine Instanz** einer Klasse.
Die Klasse beschreibt dabei generell welche Eigenschaften und Operationen ein Objekt haben kann. Eigenschaften können zum Beispiel Daten (wie der Name oder die Farbe des Objekts) sein. Operationen wiederum sind Funktionen die das Objekt ausführen kann. Das Objekt, also eine konkrete Instanz der Klasse, hat dann zum Beispiel eine bestimmte Farbe (z.B. grün). Ein anderes Objekt der selben Klasse kann eine eigene Farbe (z.B. rot) haben.

Man muss sich dabei oft entscheiden, wie generisch oder genau man eine Klasse macht. Zum Beispiel kann man die Klasse Sportschuh und die Klasse High-Heels haben, oder man hat nur eine Klasse Schuh mit einer Eigenschaft, welche die Schuhart beschreibt.

a) Wir haben hier ein paar Klassen. Bitte erweitere sie um je drei weitere Eigenschaften. Der Stuhl ist ein Beispiel:

Stuhl	**Auto**	**Haus**	**Buch**
Marke	Marke	Fensteranzahl	Autor
Farbe	_____	_____	_____
Anzahl Beine	_____	_____	_____
Material	_____	_____	_____

b) Nun erstelle zu jeder Klasse drei verschiedene Objekte. Wenn ihr in einer Gruppe zusammenarbeitet sollte sich jeder selbst ein Objekte ausdenken.

c) Identifiziere im folgenden Text drei vorkommende Klassen mit jeweils drei Eigenschaften:

> Die BilligMöbel GmbH ist ein Unternehmen mit 1000 Mitarbeitern. Die Akte jedes Mitarbeiters enthält seinen Namen und Alter sowie sein Gehalt und seine Kontonummer. Aufgrund von Massenproduktion kann das Unternehmen seine Möbel sehr billig verkaufen. So werden beispielsweise Stühle für 10€, Liegen für 15€ und Schränke für 32€ angeboten. Alle Möbelstücke sind aus Plastik oder Holz sowie in verschiedenen Farben erhältlich. In seiner Kundenkartei vermerkt das Unternehmen jeweils das Geschlecht und die Postleitzahl ihrer Kunden sowie die Kosten des letzten Einkaufs.

HowTo: Operationen von Objekten

Klassen verfügen über Operationen, die von bzw. auf ihren Objekten ausgeführt werden können. Als Beispiel betrachten wir ein Auto. Es kann fahren, bremsen, abbiegen und den Motor an-/abschalten.

Dabei können Operationen auch sogenannte Parameter besitzen wie z.B. abbiegen(links) oder fahren(50) (für 50 km/h). Wie man genau bestimmt was diese Parameter bedeuten erklären wir noch.

d) Überlege dir für die folgenden drei Klassen Operationen. Es ist nicht notwendig Parameter zu verwenden. Der Stuhl ist wieder ein Beispiel:

Stuhl	**Ampel**	**Fernseher**	**Wecker**
Umfallen Ächzen Kaputt gehen	Grün werden _____ _____	Kanal wechseln _____ _____	_____ _____ _____

e) Überlege dir nun eine eigene Klasse und finde zu ihr 3 Eigenschaften und 3 Operationen. Dann überlege dir 3 Objekte dieser Klasse mit unterschiedlichen Eigenschaften. Wenn du in einer Gruppe arbeitest, könnt ihr gerne durchtauschen und jeder muss Objekte für die Klasse eines Kommilitonen finden.

Info: Objekte vs. reale Objekte

Wichtig zu verstehen ist, dass Klassen und Objekte keine physischen Objekte repräsentieren müssen. Man kann auch z.B. eine Klasse für Useraccounts oder eine Klasse für Personengruppen haben. Auch werden Klassen verwendet, um technische Teile des Programms zu repräsentieren, wie z.B. das Fenster, welches angezeigt wird, oder ein Paket, das durchs Internet geschickt wird.

f) Hier findet ihr ein Quiz, dass das gelernte abfragt: **Falls ein Link bereits aufgebraucht ist, verwendet einen anderen (es ist 3 mal das selbe Quiz)**

`https://kahoot.it/challenge/05570785?challenge-id=9fbcfde6-f12d-4f90-a71a-78962e44d1e0_1633206271656`

`https://kahoot.it/challenge/09632907?challenge-id=9fbcfde6-f12d-4f90-a71a-78962e44d1e0_1633206304242`

`https://kahoot.it/challenge/08343706?challenge-id=9fbcfde6-f12d-4f90-a71a-78962e44d1e0_1633206320477`

g) **(Optional)** Versuche erneut drei Klassen mit je drei Eigenschaften zu identifizieren. Diese Aufgabe soll etwas schwieriger sein als die c)

> Du betrittst die Gebäude und schaust dich um. Es gibt sehr viele Menschen hier. Doch niemanden den du kennst, also bewegst du dich direkt auf den Aufzug zu. Doch bevor du in dein Stockwerk fahren kannst, musst du deine Personalkarte vor den Leser halten. Nachdem deine Sicherheitsstufe ausgelesen wurde kannst du den Knopf für das 5. Stockwerk drücken. Langsam fährt der Aufzug los. Langsamer als sonst. Larry wird ihn wohl wieder neu eingestellt haben.
>
> Im 5. Stockwerk angekommen erwartet dich ein Sicherheitsposten. Er ist nur mit einer statt mit zwei Personen besetzt, doch das macht dir keine Sorgen. Immerhin ist es Sonntag. Der Mann fragt nach deiner Personalkarte und scannt sie. Auf seinem Bildschirm taucht ein Foto auf, welches er mit deinem Gesicht vergleicht. Doch ihm fällt nichts auf. Also grüßt er dich und lässt dich vorbei. Für einen Moment fragst du dich, woher er deinen Namen kennt, doch der wurde vermutlich einfach mit dem Foto angezeigt. Heute ist einfach alles auf diesen Karten gespeichert.
>
> Du schlenderst durch das Labyrinth aus Gängen, bis du zu deinem Ziel kommst. Die Tür ist unscheinbar und das Büro dahinter ist es auch. Kaum 10 Quadratmeter groß. Und die Fenster zeigen auf nichts Beachtenswertes. Aber das ist dir egal. Dich interessiert der Tresor in der obersten Schublade des Schreibtischs. Ein Model 4452 von SuperSecure.

Du versicherst dich kurz, dass die Tür zu ist und dann machst du dich an das Knacken des digitalen Schlosses. Es dauert kaum 2 Minuten, dann schwingt die 3cm dicke Stahltür auf und ermöglicht die Sicht auf einige USB-Sticks. Schnell suchst du den blauen mit 16 GB Kapazität und lässt ihn in deiner Hosentasche verschwinden.

Jetzt nur noch den gleichen Weg zurück.

Aufgabe 3: Syntax und Codestil

Aufgabe 3.1 Syntax

In dieser Aufgabe zeigen wir euch die Syntax von Java. Die Syntax beschreibt, wie der Code geschrieben werden muss, sodass er von einem Computer verstanden wird.
Es kommen jetzt viele Definitionen, ihr müsst euch aber keine Sorgen machen, wenn ihr nicht alles direkt versteht. Nehmt diese Aufgabe als Referenz zum Nachschauen, wenn ihr euch später bei etwas nicht sicher seid.

a) **Schreibt** zu jedem Teil selbst zwei Beispiele in einem Editor.

b) Hier findet ihr ein Quiz, dass das gelernte abfragt: **Falls ein Link bereits aufgebraucht ist, verwendet einen anderen (es ist 3 mal das selbe Quiz)**

Macht das Quiz nachdem ihr Aufgabe a) gemacht habt.

`https://kahoot.it/challenge/02297621?challenge-id=9fbcfde6-f12d-4f90-a71a-78962e44d1e0_1633208690696`

`https://kahoot.it/challenge/06631442?challenge-id=9fbcfde6-f12d-4f90-a71a-78962e44d1e0_1633208706556`

`https://kahoot.it/challenge/03438017?challenge-id=9fbcfde6-f12d-4f90-a71a-78962e44d1e0_1633208720874`

Syntax

In diesem Teil zeigen wir euch die Syntax von Java. Die Syntax beschreibt, wie der Code geschrieben werden muss, sodass er von einem Computer verstanden wird.
Es kommen jetzt viele Definitionen, ihr müsst euch aber keine Sorgen machen, wenn ihr nicht alles direkt versteht. Nehmt diesen Teil als Referenz zum Nachschauen, wenn ihr euch später bei etwas nicht sicher seid.

Kommentare:

Kommentare sind Teile des Codes, die vom Computer ignoriert werden. Sie werden genutzt, um anderen (aber auch euch selbst) kenntlich zu machen, was im Code passiert, was noch verändert werden sollte oder um allgemein Anmerkungen zu hinterlassen.

Kommentar Beispiel

```
1  // Dies ist ein einfacher Kommentar über eine Zeile.
2  // Er wird mit // gestartet und nimmt die gesamte Zeile hinter sich ein.
3  someCode.run(); //Kommentare sind auch nach Operationen möglich.
4  /*
5  * Dies ist ein Block Kommentar.
6  * Diese können sich auch über mehrere Zeilen erstrecken.
7  * Um die Lesbarkeit zu verbessern, werden die Zeilen dazwischen meist
8  * mit einem * begonnen.
9  */
```

Klassen:

Klassen sind Baupläne für Objekte. In diesen können Operationen definiert werden, welche dann den Objekten zur Verfügung stehen.
Eine Klasse beginnt immer mit dem **class** Schlüsselwort, gefolgt von ihrem Namen. Diesen könnt ihr fast frei wählen, nur spezielle Schlüsselwörter wie zum Beispiel **class** sind verboten. Anschließend folgt ein Paar geschweifter Klammern { }, in denen der Inhalt der Klasse geschrieben wird. In Java muss der Dateiname gleich dem Klassennamen sein. Z.B. Klasse „Hund“ → Datei „Hund.java“ .

Klassen Syntax

```
1   // < > wird immer durch das ersetzt was in < > beschrieben wird
2   public class <NameDerKlasse>{ // Beginn der Klasse
3       /*
4       * Inhalt der Klasse. Um die Lesbarkeit zu verbessern wird
5       * Code innerhalb geschweifter Klammern eingerückt.
6       */
7       // Definition einer Operation (Siehe nächsten Eintrag)
8       public <Rückgabetyp> <OperationsName>(<Typ> <ParameterName>, ...){
9           // Inhalt der Operation
10      }
11      ...
12  }
```

Klassen Beispiel

```java
public class Hund{
    public void wuff(){
        // Code um zu bellen
    }
}
```

Operationen

Eine Operation besitzt einen Namen, einen Rückgabewert und kann Parameter besitzen (sie besitzen auch noch eine Sichtbarkeit, wir verwenden im Vorkurs immer **public** um es zu vereinfachen).
Operationen beinhalten Code, der mithilfe des Operationsnamen über die Objekte einer Klasse aufgerufen werden kann. Der Rückgabewert gibt an, ob und was die Operation zurückgeben wird. Sollte eine Operation nichts zurückgeben so hat sie den Rückgabewert **void**.
Parameter sind Variablen, die es ermöglichen einen Wert zu übergeben. Diese Variable ist nur innerhalb der Operation im Code gültig. Parameter werden in den Klammern nach dem Namen der Operation angegeben. Danach kommt in geschweiften Klammern der Inhalt der Operation.
Wenn man eine Operation aufruft und ein Objekt dort übergibt, wird dieses Objekt Argument genannt.

Bei der Definition einer Operation werden die Variablen, die festlegen, was man übergeben, kann **Parameter** genannt. Die Objekte, die man beim Aufrufen einer Operation übergibt, werden **Argumente** genannt.

Operations Syntax

```java
public <RückgabeTyp> <OperationsName>(){
    // Inhalt der Operation
}
public <RückgabeTyp> <OperationsName>(<Typ> <ParameterName>, ...){
    // Inhalt der Operation
}
```

Operations Beispiel

```java
public void springen(){
    // Code um zu springen
}

public Hund sucheEinenHundAus(String wunschFarbe){
    // Code um einen Hund auszusuchen
}
```

Objekte erzeugen:

Objekte sind Instanzen von Klassen. Um ein Objekt zu erzeugen wird das **new** Schlüsselwort verwendet. Man schreibt dann **new** gefolgt von dem Klassennamen und ein paar Klammern ().

Objekt erstellen Syntax

```java
new <KlassenName>(<MöglicheParameter>);// Ein Objekt wird erzeugt
```

Objekt erstellen Beispiel

```
1  new Hund("Border Collie");
```

Variablen:

Variablen werden genutzt, um Werte zu speichern. Sie sind in den Fällen nützlich, in denen ihr einen bestimmten Wert mehrfach im selben Code nutzen wollt. Außerdem auch wenn ein Wert beim Starten des Programms noch nicht fest steht oder sich noch ändert. Variablen besitzen immer einen Typ und einen Namen. Mit einem = kann einer Variable ein Wert zugewiesen werden. Wenn man ein Objekt erzeugt, weist man es oft direkt einer Variable zu, damit man es später im Code noch benutzen kann.

Variablen Syntax

```
1  <KlassenName> <Name>; // Eine Variable wird erzeugt, ihr ist noch kein
       ↪ Wert zugewiesen.
2
3  <KlassenName> <Name> = <Wert>; // Eine Variable wird erzeugt und ihr
       ↪ wird ein Wert zugewiesen.
4
5  <Name> = <Wert>; // Einer bestehenden Variable wird ein neuer Wert
       ↪ zugewiesen.
6
7  <KlassenName> <Name> = new <KlassenName>(); // Einer neuen Variable wird
       ↪  eine neue Instanz eines Objekts zugewiesen.
8
9  <KlassenName> <Name> = <NameEinerAnderenVariable>; // Einer neuen
       ↪ Variable wird der Wert einer anderen Variable zugewiesen.
```

Variable Beispiel

```
1  String etwasText = "Dies ist nun der Wert der Variable";
2
3  Hund meinHund = new Hund("Deutscher Schäferhund");
```

Kommandos und Abfragen:

Kommandos und Abfragen sind zwei Varianten von Operationen einer Klasse. Kommandos beschreiben alle Operationen, die etwas an dem Objekt, auf dem sie aufgerufen werden, verändern. Abfragen sind alle Operationen, die nichts verändern, dafür aber eine Eigenschaft als Wert zurückgeben.
Um Operationen eines Objekts aufzurufen, wird zuerst der Objektname geschrieben, gefolgt von einem Punkt und dem Operationsname. Am Ende stehen paar Rundeklammern „( )“ , in denen Parameter stehen können.
Wichtig: Operationen können nur auf Objekten und nicht auf Klassen aufgerufen werden. Es gibt hier einen Sonderfall, diesen werdet ihr in PSE kennenlernen.

Kommando/Abfrage Syntax

```
1  <ObjektName>.<OperationsName>();
2
3  <ObjektName>.<OperationsName>(<Argument>);
```

Kommando/Abfrage Beispiel

```
1  // meinHund.sitz(); soll dafür sorgen dass der Hund sich hinsetzt, es
       ↪ wird also etwas am Objekt geändert und somit ist diese Operation
       ↪ ein Kommando.
2  meinHund.sitz();
3
4  // meinHund.getAlter() soll wiedergeben wie alt der Hund ist, es wird
       ↪ also etwas abgefragt wonach es sich um eine Abfrage handelt.
5  meinHund.getAlter();
```

Aufgabe 3.2 Namen und Codestyle

Wie ihr wahrscheinlich schon gemerkt habt, kann Code schnell überfordern und verwirren. Da man dieses Problem in der Informatik oft antrifft wenn man mit Anderen zusammen arbeitet, gibt es Konventionen wie der Code geschrieben werden sollte, um die Lesbarkeit und das Verständnis zu verbessern.

Namen:

Die Java-Syntax verbietet bereits, dass:

- Schlüsselwörter(**void**, **class**, ...) als Name genutzt werden.
- Namen Leerzeichen beinhalten.
- Namen mit Zahlen beginnen.

Formulierungen auf die man achten sollte:

- Variablennamen sollten sprechend sein → bestDog statt bD
- Operationen sollten Verben im Namen haben → ”makeNoise” statt ”laut”
- Abfragen sollten mit get beginnen
- Kommandos, die einen Wert setzen, sollten mit set beginnen
- Keine Umlaute

Allgemein sollten alle Namen auf Englisch formuliert werden. Variablen und Operationen werden immer mit einem Kleinbuchstaben angefangen. Falls ein Name aus mehreren Wörtern besteht, schreibt man die Anfangsbuchstaben ab dem zweiten Wort groß. Diese Schreibweise nennt man auch camelCase. Sie sollten jeweils so benannt sein, dass erkennbar ist, was sie tun bzw. wofür sie stehen.
Für Klassen gelten fast die selben Richtlinien, nur ist hier auch das erst Wort groß. Das nennt man dann PascalCase oder UpperCamelCase.

Beispiel gute Namensgebung

```
1  class DogOwner{
2
3      void talkAboutTheDog(Dog dog){
4          String ownerName = "FIUS";
5          // Talk about the Dog until stopped
6      }
7  }
```

Einrückungen:

Um dafür zu sorgen, dass erkennbar ist, ob man sich gerade in einer Operation oder in einer Klasse befindet, wird allgemein jeder Code-Block innerhalb von geschweiften Klammern um eine Stufe weiter eingerückt als der Block, in dem die Klammern stehen. Wichtig ist, dass ihr immer alle Klammern wieder schließt, die ihr öffnet. Dabei müsst ihr auch die Reihenfolge der schließenden Klammern beachten!

Beispiel gute Namensgebung

```
1   // Code außerhalb einer Klammerung, nicht eingerückt
2   public class RandomClass{
3       public void operation1(){
4           // Operationsinhalt 2-mal eingerückt
5       }
6       // Klasseninhalt 1-mal eingerückt
7       public void operation2(){
8           // Operationsinhalt 2-mal eingerückt
9       }
10  }
```

Aufgabe 4: DemoTask und UI

a) Nun wollen wir wie in Aufgabe 1 ein Spiel starten. Dieses Mal nur mit dem `DemoTask` und noch ohne den `DemoTaskVerifier`:

```
1 Game myGame = new Game("Hello World", new DemoTask());
```

Wir starten das Spiel in der Variable `myGame`, indem wir die Operation `run()` darauf aufrufen.

```
1 myGame.run();
```

Starte das Programm in Eclipse, mit dem kleinen Play-Button oben in der Werkzeugleiste und schau dich ein wenig im Fenster das dann aufgeht um.

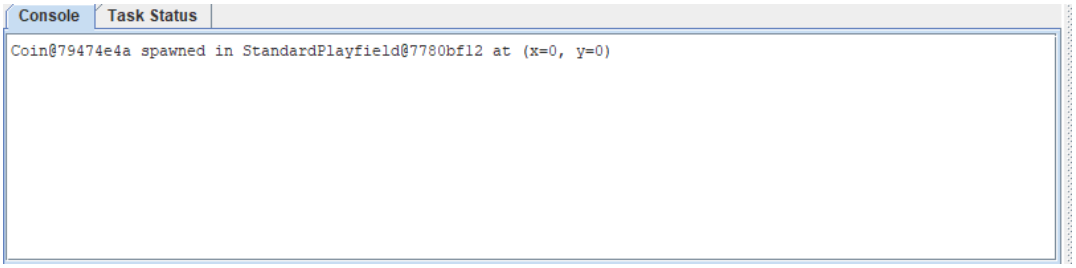
b) Nun erweitern wir den Konstruktor von `Game` um einen Parameter. Erweitere den Konstruktor um ein TaskVerifier Objekt, so wie unten im Bild zu sehen. Starte nun das Spielfenster neu. Was verändert sich im Spielfenster? Finde den „Task Status“ Tab und drücke den Refresh Button.

```
1 Game myGame = new Game("Hello World", new DemoTask(), new
    ↪ DemoTaskVerifier());
```

Der Refresh Button

Wenn ihr überprüfen wollt, ob ihr eure Aufgabe erledigt habt, müsst ihr den Task Status Tab unter dem Spielfeld öffnen und dann auf Refresh klicken.
Wichtig: Der Task Status aktualisiert sich nicht automatisch.

c) Finde sowohl im Spiel als auch in Eclipse die Konsole. Dies ist ein Feld in dem Text ausgegeben wird:



In der Konsole siehst du, dass eine Münze (`Coin`) gespawnt (platziert) wurde. Finde die Koordinaten des Feldes, auf welchem die Münze gespawnt wurde.

d) Nun suche nach der Stelle im Code der Klasse `DemoTask`, in dem die erste Münze erzeugt wird.

Kleiner Tipp: Wenn du Strg drückst während du auf einen Klassennamen oder einen Operationsnamen im Code klickst, öffnet Eclipse die entsprechende Java-Datei. Alternativ kannst du über den PackageExplorer in das Paket `de.unistuttagrt.informatik.fius.jvk.tasks` navigieren und dort die Datei `DemoTask.java` mit einem Doppelklick öffnen.

Aufgabe 5: Eine API verwenden / Doku lesen

a) Wie in Aufgabe 4 bereits vorgestellt, erstellen wir wieder ein neues Spiel. Dafür müsst ihr wieder die `Main` Klasse bearbeiten. Wir geben dem `Game` Konstuktor den Fenstertitel, den Task `Sheet1Task5` und den Verifier `Sheet1Task5Verifier` mit.

```
1 Game myGame = new Game("Task 5", new Sheet1Task5(), new
    ↪ Sheet1Task5Verifier());
```

Benennung der Task Klassen

Wir werden im weiteren Verlauf der Blätter das Namensschema wie eben gezeigt verwenden. Das heißt für Aufgabe **X** auf Blatt **Y** sollt ihr die Klassen `SheetYTaskX` und `SheetYTaskXVerifier` benutzen.
Für jede Aufgabe müsst ihr zuerst in der `Main` Klasse die Klassen `SheetYTaskX` und `SheetYTaskXVerifier` im `Game` Konstruktor eintragen. Den Code für die Aufgabe werdet ihr (fast) immer in der Klasse `SheetYTaskX` schreiben.

b) Navigiere jetzt in die Klasse `Sheet1Task5` um dort alle Änderungen vorzunehmen. Jetzt wollen wir selber Münzen auf dem Spielfeld platzieren. Erzeuge mindestens 5 Münzen und platziere sie auf beliebigen Feldern. Was fällt dir auf, wenn du mehrere Münzen auf einem Feld platzierst? Wie genau korrelieren Position und Koordinatensystem?

Tipp: Wenn du nicht genau weißt, wie man eine Münze spawnt, schau dir den Code, welchen du in Aufgabe 4 d) gefunden hast, nochmal an.

c) Nachdem wir nun die Operation `placeEntityAt()` aus der Klasse `PlayfieldModifier` kennengelernt haben, beschäftigen wir uns nun etwas genauer mit dem `PlayfieldModifier`. Insbesondere wollen wir nun herausfinden, welche anderen Operationen von dieser Klasse bereitgestellt werden. Es gibt noch zwei weitere Operationen, welche wir zum Platzieren von Münzen (`Coin`), Wänden (`Wall`) und Spielfiguren verwenden können. Versuche entweder mit Autocomplete (wie in Aufgabe 1 e) beschrieben) von Eclipse oder mit der Dokumentation auf `https://fius.github.io/ICGE2/master/` harauszufinden, wie diese heißen.

Die Dokumentation der `PlayfieldModifier` Klasse findest du direkt unter dem Link `https://fius.github.io/ICGE2/master/de.unistuttgart.informatik.fius.icge.simulation/de/unistuttgart/informatik/fius/icge/simulation/tools/PlayfieldModifier.html`.

Autocompletion

Sobald wir ein Objekt instanziiert haben und in eine Variable gespeichert haben, können wir hinter den Variablennamen einen Punkt '.' eingeben. Jetzt wird eine Liste von allen möglichen Attributen und Operationen angezeigt, die zu dem Objekt gehören. So kann man recht schnell alle möglichen Operationen durchsuchen, die mit diesem Objekt möglich sind.

Alle Objekte in Java haben die Operationen `equals(...)`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()` und `wait(...)`. Diese Operationen könnt ihr meistens ignorieren. Am häufigsten werden von diesen Operationen die Operationen `equals(...)` und `toString()` benötigt.

d) Jetzt, da uns weitere Operationen zur Verfügung stehen, können wir einfach 20 Münzen in einer Zelle platzieren. Dazu musst du das Kommando vom `PlayfieldModifier` benutzen, welches mehrere Entities auf dem selben Feld platziert. Für den ersten Parameter (`entityFactory`) kannst du `new CoinFactory()` verwenden. Wenn dir nicht klar ist, wie dieses Kommando genau funktioniert, dann lies dir nochmal die Dokumentation der Kommandos der Klasse `PlayfieldModifier` durch.

Dokumentation / JavaDoc

Normalerweise arbeiten wir nicht alleine an unserem Code, deshalb ist es wichtig zu dokumentieren wie unser Code funktioniert. Das heißt, mit einer guten Dokumentation sollte direkt ersichtlich sein, was bei dem Aufruf einer Operation passiert und was sie für Parameter erwartet. Diese Dokumentation können wir zum Beispiel online nachschauen oder direkt in unserer IDE. Wenn wir den Mauszeiger über den Namen einer Operation bewegen, wird uns ein Kasten mit der Dokumentation dieser Operation angezeigt.

e) Platziere nun 3 horizontale Reihen aus jeweils 7 Münzen. Dazu solltest du das Kommando vom `PlayfieldModifier` benutzen, das Entities an mehreren Positionen platzieren kann. Um eine Reihe zu erzeugen kannst du die Klasse `Line` unserer Shape Sammlung verwenden, welche eine Start und Endposition erwartet. Das `Line` Objekt kannst du dem Kommando als zweiten Parameter (`positions`) übergeben. Die Linien sollten nicht über Felder gehen die schon Münzen enthalten, damit die Erkennung im Task Status richtig funktioniert. Am besten haltet ihr ein Feld Abstand zwischen den Linien und anderen Münzen.

Übrigens: In dem Paket `de.unistuttgart.informatik.fius.jvk.provided.shapes` findest du die anderen Shapes aus unserer Shape Sammlung.

```
1 // Beispiel: eine Reihe von der Position (0,0) bis (5,0)
2 Position start = new Position(0,0);
3 Position end = new Position(5,0);
4
5 Line myLine = new Line(start, end);
```