

Assignment 3

Abgabe bis zum 29. April 2025 um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 29. April 2025 um 23:59 Uhr auf folgender Webseite ab:

<https://assignments.hci.uni-hannover.de>

Ihre Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode und ggf. ein PDF-Dokument bei Freitextaufgaben enthält. Beachten Sie, dass Dateien mit Umlauten im Dateinamen nicht hochgeladen werden können. Entfernen Sie die daher vor der Abgabe die Umlaute aus dem Dateinamen. Überprüfen Sie nach Ihrer Abgabe, ob der Upload erfolgreich war. Bei technischen Problemen, wenden Sie sich an Sebastian Preuss. Es wird eine Plagiatsüberprüfung durchgeführt. Gefundene Plagiate führen zum Ausschluss von der Veranstaltung. In dieser Veranstaltung wird ausschließlich die Java/JavaFX Version 17 verwendet. Code und Kompilate anderer Versionen sind nicht zulässig. Ihre Abgaben sollen die vorgegebenen Dateinamen verwenden. Wiederholter Verstoß gegen diese Regel kann zu Punktabzügen führen.

Aufgabe 1: Jump 'n' Run

Innerhalb dieser Aufgabe entwickeln Sie ein simples 2D Jump 'n' Run Spiel. Weiter wird ein auf Regeln basierter sogenannter Non-Person-Character (NPC) implementiert, der versucht das Ende des Spiels zu erreichen.

Erstellen Sie für alle Klassen, Methoden und Member-Variablen JavaDoc Kommentare. Halten Sie sich dabei an die JavaDoc Konventionen, die Sie im Anhang finden. Weiter sind sinnvolle Getter/Setter-Methoden und Konstruktoren zu erstellen. Überlegen Sie sich eine Packagestruktur und begründen Sie diese innerhalb der Main-Methode.

Wie das fertige Spiel aussehen wird, können Sie unter <https://youtu.be/lcskiNa4qgQ> sehen. G steht hierbei für Ground, L für Lava, P für den Spieler, O für ein Hindernis und Z für das Ziel. Ihre Abgabe muss nicht die Konsole nach jeder Ausgabe leeren.

a) Erstellen Sie den Enum-Typen `GameObject`, welches verschiedene Spielobjekte innerhalb des Spieles repräsentieren soll. Der Enum-Typ kann die Werte `GROUND`, `LAVA`, `AIROBSTACLE`, `EMPTY` und `GOAL` annehmen. Jedes Spielobjekt wird durch ein einzigartiges Symbol vom Typ `'char'` dargestellt, welches innerhalb des Enums gespeichert wird. Implementieren Sie die Funktion `public char getSymbol()` innerhalb des Enum-Typs, welche das Symbol des Enum-Wertes zurückgibt.

b) Erstellen Sie die Klasse `NPC`, die den Non-Playable-Character darstellt. Der NPC kann entweder stehen (stand) oder kriechen (crouch). Ob der NPC gerade steht, wird in dem Attribut `isCrouching` gespeichert. Die X-Koordinate des NPCs im Spiel wird im Attribut `xCoordinate` gespeichert.

Die Klasse `NPC` hat dementsprechend folgende Attribute:

- `xCoordinate`: `int`
- `isCrouching`: `boolean`

Erstellen Sie außerdem die folgenden Methoden:

- `public NPC()` // Konstruktor

- `public void standUp()` // Setzt isCrouching, sodass der NPC steht
- `public void crouch()` // Setzt isCrouching, sodass der NPC kriecht

c) Nun soll sich der NPC in der Lage sein nach rechts zu laufen und zu 'dashaen'. Das Laufen soll in der Methode `public void walkRight()` umgesetzt werden. Wird diese Methode aufgerufen, dann wird die xCoordinate um 1 erhöht. Das Dashaen wird in der Methode `public void dashRight()` realisiert und ist nur im Stehen möglich. Allerdings wird die xCoordinate um 2 erhöht. Steht der NPC **nicht**, soll nichts passieren.

d) Die Klasse GameView stellt das Spiel dar. Intern nutzt es dafür 2D-GameObject-Array der Größe (x:20, y:3) und einen NPC. Erstellen Sie die Klasse samt Konstruktor. Initialisieren Sie im Konstruktor das GameObject-Array, wie in Figure 1. Beachten Sie, leere Felder mit GameObject.EMPTY zu füllen.

Implementieren Sie weiter die Methode `public String str()` innerhalb der GameView Klasse, die das aktuelle Spielfeld samt NPC als String zurückgibt. Der NPC wird durch den Charakter 'P' dargestellt. Steht der NPC, wird er durch zwei übereinander liegenden 'P' an seiner X-Koordinate dargestellt., anderenfalls nur durch ein 'P'.

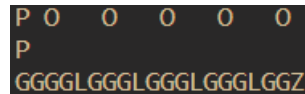


Figure 1: Beispielhaftes Spielfeld: G steht für GROUND, O für AIROBSTACLE, L für LAVA, Z für GOAL (Ziel) und P für NPC

e) Implementieren Sie die Methode `public void play()` innerhalb der GameView Klasse. Aufgabe dieser Methode ist es, das Spiel solange zu spielen, bis es endet. Dies geschieht durch regelbasierte Entscheidungen und wird durch den folgenden Pseudo-Code erklärt:

```

1 public void play() {
2     while(true) {
3         - Spielfeld ausgeben
4
5         - Steht der NPC auf Lava oder dem Ziel, ist das Spiel beendet.
6
7         - Steht der NPC vor einem AIROBSTACLE, soll er jetzt kriechen.
8
9         - Steht der NPC links vor einem LAVA-Objekt, soll er dashen
10
11        - Kriecht der NPC links vor einem LAVA-Objekt, soll er aufstehen.
12
13        - Sonst läuft der NPC nach rechts.
14    }
15 }
16

```

Testen Sie die Methode innerhalb der Main-Methode innerhalb der GameView Klasse.

Aufgabe 2: Marketplace 2

In dieser Woche sollen Sie Ihre Implementierung für die Marketplace Aufgabe aus der letzten Woche erweitern. Sollten Sie diese letzte Woche nicht bearbeitet haben oder anderweitig damit Probleme gehabt haben, können Sie eine Musterlösung zu Assignment 2 im Stud.IP Dateibereich der Übung unter *Marketplace.zip* herunterladen.

- a) Ergänzen Sie zu allen Klassen, Methoden und Attribute JavaDoc Kommentare. Halten Sie sich dabei an die JavaDoc Konventionen, die Sie im Anhang finden. Versehen Sie auch alle Klassen, Methoden und Attribute, die Sie in den anderen Teilaufgaben erstellen, mit JavaDoc Kommentaren, die den Konventionen aus dem Anhang folgen.
- b) Implementieren Sie den Enum-Typen `Category` innerhalb des `offerings` Package. Dieser Typ kann die Werte `FURNITURE`, `ELECTRONICS`, `OTHER` annehmen. Realisieren Sie innerhalb dieses Typs die Methode `public String str()`, die eine eindeutige String-Repräsentation des Enum-Wertes als String zurückgibt.
- c) Modifizieren Sie die `Item` Klasse, sodass diese ebenfalls die Kategorie als Attribut speichert. Implementieren Sie die Methode `public String str()` analog zur vorherigen Teilaufgabe.
- d) Erstellen Sie die Methode `public String filterMarket(Category category)` innerhalb der `Marketplace` Klasse. Diese Methode funktioniert wie die bereits existierende `public String str()` Methode, jedoch werden nur Items ausgegebene, die der übergebenen `Category` angehören.

Aufgabe 3: Debugging

In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitte bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Jedoch beschreiben Kommentare beschreiben immer das **korrekte** Verhalten des Programms. Sie dürfen den Code kompilieren und ausführen, um Fehler zu finden. Arbeiten Sie in der Template-Datei in der `Debug03.zip`.

- a) Der Code enthält 6 Fehler. Kompilieren und führen Sie den Code aus. Suchen Sie anhand der Fehlermeldungen des Compilers oder der Laufzeitfehlermeldungen Fehler im Code und korrigieren Sie diese. Beschreiben Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während der Fehlerkorrektur einen Blockkommentar unter dem Code. Schreiben Sie die Zeile(n) des Fehlers auf und beschreiben Sie den Fehler. Kopieren Sie die Fehlermeldung, sofern es eine gab, anhand welcher Sie den Fehler erkannt haben. Die Dokumentation soll wie in folgendem Beispiel aussehen:

```
1 public class Debug { //K: class falsch geschrieben (ckass)
2
3   ...
4
5   /*
6   ...
7   Zeile 1: class Keyword falsch geschrieben (ckass):
8   Fehlermeldung:
9   *****
10  Debug.java:1: error: class, interface, or enum expected
11  public ckass Debug {
12      ^
13  *****
14  Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
15      ↳ geschriebene ckass bekommen.
16  ...
17  */
```

Anhang A: JavaDoc Konventionen

Alle JavaDoc Kommentare sind in **Englisch** zu erstellen, da dies die Standardsprache für Code-Dokumentation ist.

Ein JavaDoc Kommentar zu einer Klasse soll immer folgende Informationen enthalten:

- Der erste Satz (*brief description*) soll eine kurze Beschreibung der Klasse enthalten.
- Der Text unter der *brief description* soll eine detailliertere Beschreibung dazu haben, was die Klasse genau darstellt, aber **keine Implementierungsdetails**.
- Der @author Tag soll den Namen der Person, die für diese Klasse zuständig ist, beinhalten.
- Der @version Tag soll normalerweise eine Versionsnummer und das Datum der letzten Änderung beinhalten. Da Versionsnummern in der Einzelübung nicht relevant sind, soll hier nur das Datum der letzten Änderung stehen.

Ein JavaDoc Kommentar zu einer Membervariable soll immer folgende Informationen enthalten:

- Eine kurze Beschreibung, was diese Variable darstellt.

Ein JavaDoc Kommentar zu einer Methode soll immer folgende Informationen enthalten:

- Der erste Satz (*brief description*) soll eine kurze Beschreibung der Funktionalität der Methode enthalten.
- Der Text unter der *brief description* soll (falls nötig) eine genauere Beschreibung der Funktionalität der Methode beinhalten. Dazu gehören ggf. mögliche unerwartete Nebenwirkungen der Methode, allerdings **keine Implementierungsdetails**.
- Der @author Tag soll den Namen der Person, die diese Methode zuletzt bearbeitet hat, beinhalten.
- Die @param Tags sollen die Eingabe parameter beschreiben.
- Der @return Tag soll den Rückgabewert beschreiben.

Anhang B: JavaDoc Beispiel

```
1 package a.b.c;
2
3 import java.lang.Math;
4
5 /**
6  * This class provides multiple Methods to calculate integers in arrays.
7  * This class also tracks the amount of Method calls on this class.
8  *
9  * @author Patric Plattner&ltltpatric.plattner@hci.uni-hannover.de>>;
10  * @version 2022 April 27
11  */
12 public class Operators {
13     /** Counts method calls on this class. Calls to getters of this Variable should be ignored.*/
14     protected static int count_;
15
16
17     /**
18      * This Method adds the absolute values of given Values.
19      * This Method has the side effect of altering the values of the input array to their Math.abs
20      * ↪ value.
21      *
22      * @author Patric Plattner <patric.plattner@hci.uni-hannover.de>;
23      * @param arr Array of integers to add up.
24      * @return Sum of all Math.abs values of the input array.
25      */
26     public static int addAbs(int[] arr) {
27         Operators.count_++;
28         int res = 0;
29         for (int i = 0; i < arr.length; ++i) {
30             arr[i] = Math.abs(arr[i]);
31             res += arr[i];
32         }
33         return res;
34     }
35
36     /**
37      * This Method multiplies the absolute values of given Values.
38      * This Method has the side effect of altering the values of the input array to their Math.abs
39      * ↪ value.
40      *
41      * @author Patric Plattner <patric.plattner@hci.uni-hannover.de>;
42      * @param arr Array of integers to multiply.
43      * @return Product of all Math.abs values of the input array.
44      */
45     public static int mulAbs(int[] arr) {
46         Operators.count_++;
47         int res = 1;
48         for (int i = 0; i < arr.length; ++i) {
49             arr[i] = Math.abs(arr[i]);
50             res *= arr[i];
51         }
52         return res;
53     }
54
55     /**
56      * Getter for {@link a.b.c.Operators#count_}.
57      * {@link a.b.c.Operators#count_} is a Variable that tracks how often Methods of this class
58      * ↪ were called.
59      *
60      * @return count
61      */
62     public static int getCount() {
63         return Operators.count_;
64     }
65 }
```

Anhang C: Beispiel zu `switch` Statements

```
1 class Demo {
2     public static void main(String[] args) {
3         String switchString = "A";
4         //String switchString = "B";
5         switch(switchString) {
6             case "A":
7                 System.out.println("A");
8                 break;
9             case "B":
10                System.out.println("B");
11                break;
12        }
13    }
14 }
15 //Prints "A" if line 4 is commented out, prints "B" if line 3 is commented out.
```
