Assignment 3

Abgabe bis zum 2. Mai 2023 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 2. Mai 2023 (Dienstag) 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 Kevin Schumann oder Dennis Stanke. 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 immer 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 ungefähr 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. Im Gegensatz zu dem Video muss Ihre Abgabe 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, AIR 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. Diese Klasse stellt den Non-Playable-Character dar. Der NPC kann stehen oder kriechen. Ob er gerade steht oder kriecht, wird innerhalb der Klasse unter dem boolean *isCrouching* gespeichert. Steht der Charakter, wird der NPC später innerhalb des Spielfelds durch zwei übereinander liegende 'P's dargestellt. Tut er das nicht nur durch ein 'P', wie im Video zu erkennen ist. Die Darstellung ist nur für Teilaufgabe c und d wichtig. Die NPC Klasse speichert außerdem die X- und Y-Koordinate des Charakterfußes, also dem unterem 'P'.

Implementieren Sie weiter die Methode `public void toogleCrouch()`, die das Umschalten zwischen stehen oder kriechen erlaubt.

Der Charakter ist auch in der Lage, nach rechts zu laufen und nach rechts zu 'dashen'. Beim Laufen bewegt er sich einen Block und beim 'dashen' direkt um zwei. 'Dashen' ist nur während des Stehens möglich. Implementieren Sie die Methoden `public void walkRight()` und `public void dashRight()`, die die X-Koordinate des NPC's entsprechend anpasst. Gehen Sie hierbei davon aus, dass der NPC die Aktion ausführen kann.

Vorlesung:	apl. Prof. Matthias Becker	xmb@hci.uni-hannover.de
Übung:	Kevin Schumann	k.schumann@stud.uni-hannover.de
Organisation:	M.Sc. Dennis Stanke	dennis.stanke@hci.uni-hannover.de

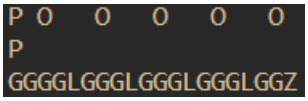

Figure 1: Beispielhaftes Spielfeld: G steht für GROUND, O für AIROBSTACLE, L für LAVA, Z für GOAL (Ziel) und P für NPC

c) Das Spielfeld wird durch die Klasse GameView dargestellt. GameView speichert eine 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, sodass sich am unteren Rand des Spielfeldes GameObject.GROUND befinden. Setzen Sie den NPC auf den linken Spielfeldrandboden. Beachten Sie, leere Felder mit GameObject.AIR zu füllen.

Implementieren Sie weiter die Methode `public String str()` innerhalb der GameView Klasse, die das aktuelle Spielfeld samt NPC als String zurückgibt.

d) 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:

```
public void play() {
    while(true) {
        Spielfeld ausgeben

        Wenn der NPC sich auf Lava oder Goal befindet, dann ist das Spiel beendet.

        Wenn ein Airobstacle das weitergehen hindert, dann muss der NPC kriechen.

        Ist der Boden neben dem NPC Lava, dann muss der NPC darüber 'dashen'. Steht der
            ↪ NPC nicht, muss er erst aufstehen.

        Sonst läuft der NPC nach rechts.
    }

}
```

e) Modifizeren Sie den Konstruktor, sodass:

- sich GameObject.AIROBSTACLES am oberen Spielfeldrand befinden
- ein GameObject.GOAL am rechten Spielfeldrand existiert
- einige GameObject.GROUND-Blöcke durch GameObject.LAVA ersetzt wurden

Das Spiel soll durch die oben genannten Regeln lösbar sein. Hardcoden ist hier explizit erlaubt. Sie können sich auch an Figure 1 orientieren. Testen Sie dies innerhalb der Main-Methode, welche sich in der GameView Klasse befindet.

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, SERVICES, CLOTHES, ANIMALS annehmen. Realisieren Sie innerhalb dieses Typs die Methode `public String str()`, die eine eindeutige String-Repräsentation des Enum-Wertes als String zurückgibt. Setzen Sie ebenfalls die Methode `public boolean isSameCategory(Item item)` um. Diese Methode gibt nur dann *true* zurück, wenn das übergebene Item derselben Kategorie wie der Enum-Wert angehört.

c) Modifizieren Sie die Item Klasse, sodass diese ebenfalls die Kategorie als Attribut speichert. Passen Sie die str() Methode an, sodass die Kategorie auch im String vorkommt. Ergänzen Sie auch eine Getter bzw. eine Setter Methode für die Kategorie.

d) Erstellen Sie die Methode `public String filterMarket(Category category)` innerhalb der Marketplace Klasse. Diese Methode funktioniert ähnlich wie die str() Methode derselben Klasse, jedoch werden nur diejenigen Items ausgegeben, die der übergebenen Kategorie 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        ^
```

```
**************
Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
    ↪ geschriebene ckass bekommen.
...
*/
```

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

```java
package a.b.c;

import java.lang.Math;

/**
 * This class provides multiple Methods to calculate integers in arrays.
 * This class also tracks the amount of Method calls on this class.
 *
 * @author  Patric Plattner&lt;patric.plattner@hci.uni-hannover.de&gt;
 * @version 2022 April 27
 */
public class Operators {
    /** Counts method calls on this class. Calls to getters of this Variable should be ignored.*/
    protected static int count_;


    /**
     * This Method adds the absolute values of given Values.
     * This Method has the side effect of altering the values of the input array to their Math.abs value.
     *
     * @author     Patric Plattner &lt;patric.plattner@hci.uni-hannover.de&gt;
     * @param  arr Array of integers to add up.
     * @return     Sum of all Math.abs values of the input array.
     */
    public static int addAbs(int[] arr) {
        Operators.count_++;
        int res = 0;
        for (int i = 0; i < arr.length; ++i) {
            arr[i] = Math.abs(arr[i]);
            res += arr[i];
        }
        return res;
    }


    /**
     * This Method multiplies the absolute values of given Values.
     * This Method has the side effect of altering the values of the input array to their Math.abs value.
     *
     * @author     Patric Plattner &lt;patric.plattner@hci.uni-hannover.de&gt;
     * @param  arr Array of integers to multiply.
     * @return     Product of all Math.abs values of the input array.
     */
    public static int mulAbs(int[] arr) {
        Operators.count_++;
        int res = 1;
        for (int i = 0; i < arr.length; ++i) {
            arr[i] = Math.abs(arr[i]);
            res *= arr[i];
        }
        return res;
    }

    /**
     * Getter for {@link a.b.c.Operators#count_}.
     * {@link a.b.c.Operators#count_} is a Variable that tracks how often Methods of this class were called.
     * @return count
     */
    public static int getCount() {
        return Operators.count_;
    }
}
```

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.
```