Assignment 4

Abgabe bis zum 11. Mai 2021 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 11. Mai 2021 (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 Patric Plattner 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 11 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.

Info

Ab dieser Woche werden die Aufgabenstellungen offener sein, als in den Vorwochen. Wir wollen Ihnen die Möglichkeit geben, mehr eigene Problemlösungen zu finden, statt nach Anleitung zu Programmieren. Deshalb möchten wir Sie dazu ermutigen, sich zunächst eigene Gedanken zu einer Aufgabenstellung zu machen. Sollten Sie weiterhin Hilfe brauchen, wenden Sie sich bitte an das Stud.IP Forum der Übungsveranstaltung oder an die Discord Beratungstutoren.

Wir möchten ihnen nahelegen, Ideen für die Bearbeitung von offeneren Aufgaben im Forum zu diskutieren. Das Posten von Lösungen zu Fragen und Codeausschnitten generell ist verboten, lediglich das Besprechen von Lösungs**ansätzen** ist erlaubt.

Aufgabe 1: LadybugGame 3

Ab dieser Woche soll das Spiel langsam Form annehmen. Das Spiel soll nun über die Kommandozeile spielbar werden. Sie sollen auch `Clover` in das Spiel übernehmen. Dies sollen Sie erreichen, indem Sie sich Vererbung zu Nutze machen.

Die Musterlösung der Ladybug Game Aufgabe der Vorwoche finden Sie im Stud.IP. Sie können entweder die Musterlösung erweitern, Ihre eigene Lösung der Vorwoche weiterentwickeln, oder Ihre eigene Lösung mithilfe der Musterlösung verbessern und darauf aufbauen.

a) Erstellen Sie eine `Entity` Klasse. Ändern Sie `Tree` und `Clover` ab, sodass diese von `Entity` erben. `Ladybug` darf auch von `Entity` erben, muss sie aber nicht. Entscheiden Sie das je nachdem wie das in ihre bisherige Implementierung passt.

Ihr Spielfeld aus `Tree` Objekten aus letzter Woche soll nun ein Spielfeld aus `Entity` Objekten sein. Passen Sie auch den Rest Ihrer `Game` Klasse an, sodass die Funktionalität aus letzter Woche erhalten bleibt. Sie dürfen dabei natürlich auch Methoden aus letzter Woche abändern, solange die Voraussetzungen aus letzter Woche erhalten bleiben (z.B. können Sie eine `turnLeft()` und `turnRight()` mit einer `setDirection()` Methode austauschen und vice versa).

b) Wenn Sie dies letzte Woche noch nicht gemacht haben: implementieren Sie ein Kommando, sodass der `Ladybug` einen `Clover` aufheben kann, das in der Aufgabe **c)** der LadybugGame Aufgabe der letzten Woche erwähnt wurde.

Die Methode, die den Status des Spielfeldes als String zurückgegeben hat, soll unter dem Spielfeld auch in einer Zeile den aktuellen Punktestand des `Ladybugs` und wie viele `Clover` der `Ladybug` hat ausgeben.

c) Nutzen Sie die `java.util.Scanner` Klasse um ein Kommandozeilen Interface zu erstellen.

Das Spielfeld soll nicht über das Kommandozeilen Interface veränderbar sein. Das Befüllen des Spielfeldes soll weiterhin in die `main` Methode hardcoded sein.

Setzen Sie für das Kommandozeilen Interface die Kommandos aus der Aufgabe **c)** der LadybugGame Aufgabe der letzten Woche inklusive dem Kommando zum Aufheben von `Clover` Objekten.

Sie können den Wall Follower Algorithmus aus der letzten Woche als Kommando implementieren, müssen es aber nicht. Wenn der Wall Follower nicht als Kommando implementiert wurde, entfernen Sie ihn aus der `main` Methode.

Fügen Sie dazu ein Kommando zum Beenden des Programms hinzu.

Sie finden einen Beispielablauf in Anhang A: Beispielablauf zu Aufgabe 1.

d) Schauen Sie sich alle Methoden und Member Variablen dieser Aufgabe an und entscheiden Sie für jede Methode oder Membervariable, ob diese auf `public`, `protected`, `private` oder Package Private (also kein modifier) sein sollte. Sie müssen ihre Entscheidung nicht in den Kommentaren begründen, sollten aber Fragen dazu vor ihrem Tutor beantworten können (wobei Kommentare natürlich helfen).

Versehen Sie alle Methoden, Member Variablen und Klassen, die Sie hinzugefügt haben und die auf `public` oder `protected` gesetzt sind, mit JavaDoc Kommentaren.

Aufgabe 2: Price Comparator

In dieser Aufgabe soll es vorwiegend um Vererbung und um Eingaben per Kommandozeile gehen. Sie sollen eine Software für einen imaginären Streaming Dienst entwickeln, in dem ein Nutzer eingeben kann was für Musik oder Filme er oder sie sich kaufen würde. Die Software soll dem Nutzer mitteilen, wie viel Geld dieser sparen würde und ob es Qualitätsunterschiede gibt, wenn der Nutzer stattdessen den Streamingdienst abonnieren würde.

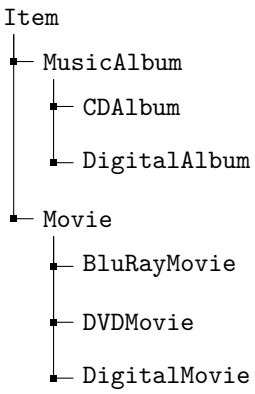
In dieser Aufgabe sollen alle Klassen, Methoden und Felder mit Javadoc Kommentaren nach der Konvention aus Assignment 3 versehen werden. Es werden keine Sichtbarkeiten für Methoden, Felder oder Klassen vorgegeben. Diese müssen Sie selbst entscheiden und wie in Assignment 3 Ihre Begründung dafür in das Javadoc Kommentar schreiben. Auch die Package Struktur wird nicht vorgegeben, außer dass Sie im root package `de.uni_hannover.hci.ihr_name.aufgabe2` arbeiten sollen. Sie sollen sich dabei eine sinnvolle Package Struktur überlegen (es ist explizit verboten alle Klassen in dassslbe Package zu legen). Sie sollen in der Lage sein, Ihre Packagestruktur Ihrem Tutor zu erklären.

a) Erstellen Sie eine `Item` Klasse. Diese Klasse soll ein Medium (Musik oder Video) in einem Shop darstellen, den es auch auf dem imaginären Streaming Dienst gibt. Ein `Item` soll folgende Eigenschaften haben:

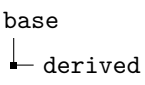
- Einen Titel.
- Einen Identifier.
- Einen Preis.

Schreiben Sie einen sinnvollen Konstruktor für die `Item` Klasse. Des weiteren soll die `String Object.toString()` Methode in dieser Klasse überschrieben werden, sodass sie eine sinnvolle Repräsentation des Items als String zurückgibt.

Als nächstes sollen Sie folgende Klassen implementieren: `MusicAlbum`, `CDAlbum`, `DigitalAlbum`, `Movie`, `BluRayMovie`, `DVDMovie` und `DigitalMovie`. Die Klassen sollen in folgender Vererbungshierarchie aufgebaut sein:



Dabei ist dieser Graph wie folgt zu lesen:



Überschreiben Sie gegebenenfalls Methoden und implementieren Sie neue Konstruktoren in den abgeleiteten Klassen, sodass sie folgendes Verhalten aufweisen. Insbesondere sollen invalide Informationen vermieden werden (z.B. doppelte Identifier). Allerdings lassen sich einige dieser Bedingungen durch das Abwandeln der Informationen im Getter, anstatt durch das Abfangen von Fehlinformation im Konstruktor gelöst werden. Sollte versucht

werden, ein invalides `Item` (oder eine abgeleitete Form) zu erstellen, soll das Programm eine Fehlermeldung per `System.err.println(fehlerMeldung);` ausgeben und per `System.exit(2);` beendet werden.

- Der Titel eines Albums besteht dabei aus dem Namen des Albums und dem Artist (z.B. “OK Computer - Radiohead”)
- Ein Musikalbum hat eine einmalige Identifikation in Form einer MBID (Mehr Infos zu MBID in den Lösungshinweisen). Beachten Sie dabei, dass es zu einem Album mehrere Versionen auf verschiedenen Formaten geben kann, diese aber alle eine unterschiedliche MBID haben.
- Eine CD hat einen Standardpreis von 14.99EUR und ein digitales Album hat einen Standardpreis von 12.99EUR
- Der Titel eines digitalen Albums endet immer in `(Digital Edition)`.
- Filme haben eine einmalige IMDB-ID (Mehr Infos zu IMDB-IDs in den Lösungshinweisen). Beachten Sie dabei, dass es zu einem Film mehrere Versionen auf verschiedenen Datenträgern geben kann, die sich die IMDB-ID teilen.
- Eine BluRay hat einen Standardpreis von 16.99EUR, ein digitaler Download 12.99EUR und eine DVD 9.99EUR.
- Der Titel eines digitalen Films endet immer mit `(Digital Edition)` und der Titel einer BluRay endet immer mit `(BD Edition)`.

Sie müssen hier noch nicht checken, ob eine eingegebene MBID oder IMDB-ID valide ist.

b) Erstellen Sie eine `Main` Klasse, die eine `main` Methode enthält. In dieser Methode sollen Sie Kommandozeileneingaben mit einem `Scanner` Objekt entgegennehmen. Nachdem eine Eingabe bearbeitet wurde, sollen Sie die nächste Eingabe entgegennehmen. In diesem Kommandozeileninterface soll eine Liste (Array) an `Items` befüllt werden. Die maximale Kapazität an `Items` soll 5 Einträge betragen. Folgende Kommandos sollen möglich sein:

- `add <CDAlbum, DigitalAlbum, BluRayMovie, DVDMovie, DigitalMovie>` Öffnet einen Dialog, in dem nach Titel, MBID/IMDB-ID und Preis gefragt wird und der entsprechende Gegenstand in die Liste gelegt werden soll. Sollte die Liste voll sein sein, wird eine Fehlermeldung ausgegeben.
- `eval` Schaut nach, wieviel Geld der Nutzer mehr oder weniger ausgeben würde, wenn der Nutzer statt die Listenelemente einmal zu kaufen, einem Monat dem Streamingdienst abonieren würde.

Als Information für `eval`: Der Streamingdienst kostet 9.99EUR pro Monat.

Hier ist ein Beispielablauf der Programmbenutzung (Nutzereingaben sind mit `[]` gekennzeichnet und die Klammern sind in tatsächlichen Eingaben nicht enthalten):

```
Please enter your command: [add DigitalAlbum]

Adding a paperback book to the inventory.
Please enter the album name:          [OK Computer]
Please enter the album artist:        [Radiohead]
Please enter the MBID code:           [30702389-5c67-4438-9ea0-2351c8de0f1d]
Please enter a price (d for default): [d]

You added the digital album "OK Computer - Radiohead" with
    ↪ MBID=30702389-5c67-4438-9ea0-2351c8de0f1d and a price of 14.99EUR to the list.

```

```
Please enter your command: [add DVDMovie]

Adding a BluRay to the inventory.
Please enter the title:                [Harry Potter and the Deathly Hallows Part 2]
Please enter the IMDB-ID code:         [tt1201607]
Please enter a price (d for default): [6.99]
The bluray was entered into the inventory.

You added the DVD movie "Harry Potter and the Deathly Hallows Part 2" with
    ↪ IMDB-ID=tt1201607 and a price of 6.99EUR to the list.

Please enter your command: [eval]

You would save 11.99EUR by subscribing for one month instead of buying these items
    ↪ individually.
```

c) (optional) Checken Sie im Konstruktor der abgeleiteten Klassen auch, ob die IMDB-ID oder MBID valide ist. Dazu müssen Sie nur nach dem Format schauen:

Eine IMDB-ID für einen Film besteht aus einem `tt` gefolgt von einer 7-elementigen Ziffernfolge. Wenn Sie einen Film auf IMDB anschauen, ist die ID in der URL hinter `title/` (Beispiel: https://www.imdb.com/title/tt0241527/ gehört zu der IMDB-ID `tt0241527`)

Eine MBID(MusicBrainzID) besteht aus 32 Hexadezimalziffern (eine Hexadezimalziffer ist ein Ziffer von 0 bis 9 oder einer der Buchstaben a,b,c,d,e und f) und 4 Dashes ("-"). Sie folgt dem Format "8-4-4-4-12" wofür die Zahlen für die Anzahl der Hexadezimalziffern steht. Sie können auf https://musicbrainz.org/ nach MBIDs suchen. Wenn Sie ein Album gefunden haben, finden Sie die MBID im Details Bereich. Hier ist ein Beispiel für den ersten CD Release von OK Computer von Radiohead: https://musicbrainz.org/release/425d8db9-88a3-31e2-bd23-b1a967626dd5/details

Bestehenskriterien

- Ihr Code kompiliert.
- Ihr Code ist wie beschrieben mit Javadoc Kommentaren versehen.
- Ihre Vererbungsstruktur um die `Item` Klasse ist so wie in der Aufgabe beschrieben.
- Ihre abgeleiteten Klassen implementieren die Funktionalität wie in der Aufgabe beschrieben.
- Ihr Kommandozeileninterface funktioniert so, wie in der Aufgabe beschrieben. Das genaue Format kann von dem Beispiel abweichen, es sollte aber dieselbe Funktionalität erfüllen. Die Kommandos sollen aber exakt, wie in der Aufgabe angegeben, umgesetzt werden

Lösungshinweise

- Sie können den `instanceof` Operator nutzen um zu schauen, ob ein Objekt von einem bestimmten Typ ist.
- Sie können die `String.split()` Methode nutzen um den Eingabestring in mehrere kürzere Strings zu teilen. Dafür wird ein Trennsymbol definiert. Beispiel: `"add_1_bluray".split("_")//== {"add", "1", "bluray"}` In diesem Beispiel wurde ein Unterstrich als Trennsymbol verwendet.
- Vergleichen Sie Strings mit `str1.equals(str2)` (Beispiel: `"abc".equals("abc")//is true`).

- Um einzelne Character aus einem String zu lesen, nutzen Sie `str.charAt(i)` (Beispiel: `"abc".charAt(0)== 'a'//is true`).
- Um die Länge von Strings auszulesen, nutzen Sie `str.length()` (Beispiel: `"abc".length()== 3 //is true`).

Aufgabe 3: Debugging

In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitt bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Es kann sich dabei um Compiler- oder Laufzeitfehler handeln. Sie dürfen den Code kompilieren und ausführen um die Fehler zu finden. Arbeiten Sie in der Template-Datei `Debug.zip`. Diese finden Sie im Stud.IP. Nutzen Sie Kommentare, um die Fehler zu Kennzeichnen. Gegeben ist der folgende Codeabschnitt:

`de.uni_hannover.hci.aufgabe3.Debug`

```java
package de.uni_hannover.hci.aufgabe3;

import de.uni_hannover.hci.aufgabe3.model.*;

public class Debug {
  public static void main(String[] args) {
    Animal[] animals = new Animal[3];
    animals[0] = new Animal("Kangaroo Bob", 2, 2);
    animals[1] = new Dog("Barks");
    animals[2] = new Monkey("King Kong");
    for (int i = 0; i < animals.length; ++i) {
      System.out.println(animals[i]);
    }
  }
}
```

`de.uni_hannover.hci.aufgabe3.Animal`

```java
package de.uni_hannover.hci.aufgabe3;

class Animal {
  private String name_;
  private int legs_;
  private int arms_;


  Animal(String name, int legs, int arms) {
    this.name_ = name;
    this.legs_ = legs;
    this.arms_ = arms;
  }

  public Animal(String name) {
    this(name, 0, 0);
  }


  public String getName() {
    return this.name_;
  }

```

```
24    public int getArms() {
25      return this.arms_;
26    }
27
28    public int getLegs() {
29      return this.legs_;
30    }
31
32    @Override
33    public String toString() {
34      return String.format("%s is an animal with %d legs and %d arms.", this.getName(),
          ↪ this.getLegs(), this.getArms());
35    }
36  }
```

de.uni_hannover.hci.aufgabe3.model.Dog

```
1   package de.uni_hannover.hci.aufgabe3.model;
2
3   // A monkey always has 0 arms and 4 legs
4   public class Dog extends de.uni_hannover.hci.aufgabe3.Animal {
5
6     public Dog(String name) {
7       super(name);
8     }
9
10
11    @Override
12    public String getName() {
13      return super.name_;
14    }
15
16    @Override
17    public int getArms() {
18      return 0;
19    }
20
21    @Override
22    private int getLegs() {
23      return 4;
24    }
25
26    @Override
27    public String toString() {
28      return String.format("%s is a dog with %d legs and %d arms.", this.getName(),
          ↪ this.getArms(), this.getLegs());
29    }
30  }
```

de.uni_hannover.hci.aufgabe3.model.Monkey

```
1   package de.uni_hannover.hci.aufgabe3.model;
2
3   // A monkey always has 2 arms and 2 legs
4   public class Monkey extends de.uni_hannover.hci.aufgabe3.Animal {
5
6     public Monkey(String name) {
7       super(name);
```

```
  }


  @Override
  public String getName() {
    return super.name_;
  }

  @Override
  public int getArms() {
    return 0;
  }

  @Override
  private int getLegs() {
    return 4;
  }

  @Override
  public String toString() {
    return String.format("%s is a dog with %d legs and %d arms.", this.getName(),
        ↪ this.getLegs(), this.getArms());
  }
}
```

Der Codeabschnitt enthält zwischen 3-5 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. Kommentieren Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während des Fehlerkorrektur einen Blockkommentar unter dem Code, in dem Sie die Fehler dokumentieren. 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:

```
public class Debug { //K: class falsch geschrieben (ckass)

...

/*
...
Zeile 1: class Keyword falsch geschrieben (ckass):
Fehlermeldung:
**************
Debug.java:1: error: class, interface, or enum expected
public ckass Debug {
       ^
**************
Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
    ↪ geschriebene ckass bekommen.
...
*/
```

Bestehenskriterien

- Ihr korrigierter Code funktioniert so wie in den Kommentaren im Code beschrieben.

- Alle korrigierten Fehler haben einen kurzen Kommentar, der beschreibt, warum die Stelle im alten Code nicht funktioniert hat.
- Alle Fehlermeldungen (Compiler und Laufzeit) sind in einem Blockkommentar `/* ... */` unter dem Code in der jeweiligen Java-Datei.

Lösungshinweise

- Im Code vorgegebene Kommentare beschreiben immer das **korrekte** Verhalten des Programms.

Anhang A: Beispielablauf zu Aufgabe 1

```
XSXXXXX
XOX   X
X X X X
X   X X
XXXXX X
Points=0

Enter your command> [forward]
X XXXXX
XSX   X
X X X X
X   X X
XXXXX X
Points=0

Enter your command> [pickup]
X XXXXX
XSX   X
X X X X
X   X X
XXXXX X
Points=10

Enter your command> [forward]
X XXXXX
X X   X
XSX X X
X   X X
XXXXX X
Points=10

Enter your command> [forward]
X XXXXX
X X   X
X X X X
XS  X X
XXXXX X
Points=10

Enter your command> [turnLeft]
X XXXXX
X X   X
X X X X
XE  X X
XXXXX X
Points=10

Enter your command> [forward]
X XXXXX
X X   X
X X X X
X E X X
XXXXX X
Points=10
...
```