Assignment 4

Abgabe bis zum 19. Mai 2020 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 19. Mai 2020 (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.

Hinweis

Ab diesem Assignment werden wir Ihnen mehr Freiheiten bezüglich Ihrer Implementierung der Aufgabe geben. Das bedeutet, dass wir Ihnen nicht immer zu 100% vorgeben, wie eine Klasse auszusehen hat, sondern, dass wir Ihnen eine Funktionalität vorgeben und Sie diese dann mit gewissen Freiheiten implementieren sollen. Das bedeutet aber auch, dass Sie evtl. Felder/Methoden implementieren müssen, die nicht explizit im Aufgabentext erwähnt werden. Dies dient dazu, Sie auf die Gruppenaufgabe vorzubereiten, in der Sie eine sehr offene Aufgabenstellung bekommen werden. Es gibt dabei nicht nur eine richtige Lösung. Wenn Sie Hilfe brauchen einen Lösungsansatz zu finden, können Sie gerne die Online-Sprechstunde in Anspruch nehmen.

Aufgabe 1: Inventory Management

In dieser Aufgabe soll es vorwiegend um Vererbung und um Eingaben per Kommandozeile gehen. Sie sollen ein kleines “Inventory Management” System entwickeln, das CDs, DVDs und Bücher organisieren kann.

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.aufgabe1` 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.

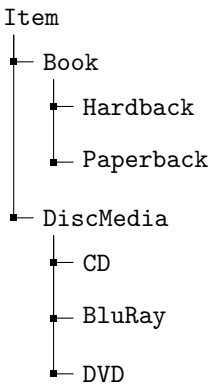
a) Erstellen Sie eine `Item` Klasse. Diese Klasse soll einen Gegenstand in einem Ladeninventar darstellen. Sie soll folgende Methoden enthalten:

- `String getTitle()`, die den Titel eines Gegenstandes zurückgibt.
- `String getID()`, die eine Identifikationsnummer eines Gegenstandes zurückgibt.
- `double getPrice()`, die den Preis eines Gegenstandes in EUR zurückgibt.

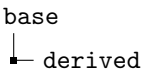
Schreiben Sie einen sinnvollen Konstruktor für die `Item` Klasse. Des Weiteren soll die `String Object.toString()` Methode in dieser Klasse überschrieben werden, sodass sie eine sinnvolle Repräsentation des Items als String zurückgibt.

Vorlesung: apl. Prof. Matthias Becker becker@hci.uni-hannover.de
Übung: Patric Plattner patric.plattner@hci.uni-hannover.de
Organisation: M.Sc. Dennis Stanke dennis.stanke@hci.uni-hannover.de

Als nächstes sollen Sie folgende Klassen implementieren: `Book`, `Paperback` (Taschenbuch), `Hardback` (Gebundenes Buch), `DiscMedia`, `CD`, `BluRay` und `DVD`. Die Klassen sollen in folgender Vererbungshierarchie aufgebaut sein:



Dabei ist dieser Graph wiefolgt 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 ISBN-13 Nummern). 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.

- Ein Buch hat immer eine einzigartige ISBN-13 Nummer (Hinweis zum Format von ISBN-13 in den Lösungshinweisen)
- Ein gebundenes Buch hat einen Standardpreis von 12.99EUR, ein Taschenbuch 9.99EUR. Der Preis kann aber abweichen.
- Der Titel eines gebunden Buches endet immer mit "`(Hardback Edition)`", bei einem Taschenbuch mit "`(Paperback Edition)`"
- Optische Medien (`DiskMedia`) haben eine ASIN Nummer (Hinweis zum Format von ASIN in den Lösungshinweisen)
- CDs haben einen Standardpreis von 8.99EUR, DVDs von 9.99EUR und BluRays von 12.99EUR. Der Preis kann aber abweichen.
- Der Titel einer BluRay endet immer mit "`(BD)`", bei einer DVD mit "`(DVD)`"

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 Komandozeileninterface soll ein Inventar (Array) an `Items` gemanaged werden. Die maximale Kapazität an `Items` soll 5 Einträge betragen. Folgende Kommandos sollen möglich sein:

- `add <spot> <paperback, hardback, bluray, dvd, cd>` Öffnet einen Dialog, in dem nach Titel, ISBN-13/ASIN und Preis gefragt wird (d für default) und der entsprechende Gegenstand in den angegebenen Inventarslot gelegt wird. Sollte dieser belegt sein, wird eine Fehlermeldung ausgegeben.

- `list` Listet alle Inventarplätze mit deren Inhalt auf.
- `take <spot>` Nimmt den Gegenstand an dem gegebenen Slot aus dem Inventar und gibt ihn auf die Konsole aus.
- (OPTIONAL) `stat` Gibt eine Statistik der Gegenstände aus dem Inventar auf die Konsole aus. Die Statistik soll daraus, wie hoch der Gesamtpreis ist und wieviele Gegenstände aus jeder Kategorie im Inventar sind, bestehen.

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

```
Please enter your command: [list]

1: Empty
2: Empty
3: Empty
4: Empty
5: Empty

Please enter your command: [add 2 paperback]

Adding a paperback book to the inventory.
Please enter the title:                [Harry Potter and the Deathly Hallows]
Please enter the ISBN-13 code:         [978-0545139700]
Please enter a price (d for default): [d]
The book was entered into the inventory.

Please enter your Command: [list]

1: Empty
2: Harry Potter and the Deathly Hallows (Paperback Edition); ISBN-13=978-0545139700;
    ↪ Price=9.99EUR
3: Empty
4: Empty
5: Empty

Please enter your command: [add 2 paperback]

This slot is already taken by:
Harry Potter and the Deathly Hallows (Paperback Edition)

Please enter your command: [add 1 bluray]

Adding a BluRay to the inventory.
Please enter the title:                [Harry Potter and the Deathly Hallows Part 1&2]
Please enter the ASIN code:            [B008UZ94F4]
Please enter a price (d for default): [25.49]
The bluray was entered into the inventory.

Please enter your command: [list]

1: Harry Potter and the Deathly Hallows Part 1&2 (BD); ASIN=B008UZ94F4;
    ↪ Price=25.49EUR
2: Harry Potter and the Deathly Hallows (Paperback Edition); ISBN-13=978-0545139700;
    ↪ Price=9.99EUR
3: Empty
4: Empty
5: Empty
```

```
45
46 Please enter your command: [stat]
47
48 Your inventory is worth 35.48EUR;
49 Your inventory contains:
50 Books:       1
51   Paperback: 1
52   Hardback:  0
53 DiscMedia:   1
54   CD:        0
55   DVD:       0
56   BluRay:    1
57
58 Please enter your command: [take 2]
59
60 You have taken the Book "Harry Potter and the Deathly Hallows (Paperback Edition)"
       ↪ with the ISBN-13 "978-0545139700" out of the inventory. It costs 9.99EUR.
61
62 Please enter your command: [take 2]
63
64 This slot is empty.
```

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

- Information zu ISBN-13 und ASIN finden Sie hier: https://www.amazon.com/gp/seller/asin-upc-isbn-info.html
- 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 2: Sudoku

In dieser Aufgabe sollen Sie ein rudimentäres Sudoku-Spiel schreiben. Sie sollen hier keine Sudoku Felder generieren, diese bekommen Sie von uns vorgegeben.

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, au0er 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 `SudokuField` Klasse. Diese soll ein SudokuFeld darstellen. Sie bekommen die Beispielfelder im Anhang dieses Assignments in Form eines `byte[][]` Arrays gegeben (-1 steht für leer). Schreiben Sie einen Konstruktor, der ein Spielfeld anhand dieser Arrays initialisiert, oder das Programm mit `System.exit(2)` beendet falls das Array ein nicht valides Sudokufeld beinhaltet. Erstellen Sie dann folgende Methoden:

- Eine Methode, die das Spielfeld als String zurückgibt.
- Eine Methode, die den Inhalt einer einzelnen Spielfelde-Zelle setzt und eine, die ihn zurückgibt. Sollte versucht werden, den Inhalt einer Zelle auf einen invaliden Wert zu setzen, dann soll dies am Rückgabewert signalisiert werden und der Inhalt der Zelle unverändert bleiben.
- Eine Methode, mit der Zahlen für eine Zelle vorgemerkt werden können (invalide Zahlen müssen hier nicht gecheckt werden), eine mit der vorgemerkte Zahlen von einer Zelle entfernt werden können und eine, die die vorgemerkten Zahlen einer Zelle ausgibt.

Implementieren Sie dabei so viele Helfermethoden oder Klassen, wie Sie brauchen.

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 ein Sudoku-Spiel gespielt werden. Wenn das Spielfeld verändert wurde (vorgemerkte Zahlen zählen nicht), soll es ausgegeben werden. Folgende Kommandos sollen möglich sein:

- `mark <row> <column> <value>` Merken Sie einen Wert für die gegebenen Zelle vor. Geben Sie eine Fehlermeldung aus, wenn die Zahl bereits vorgemerkt war.
- `unmark <row> <column> <value>` Löschen Sie die Vormerkung einer Zahl für die gegebenen Zelle. Geben Sie eine Fehlermeldung aus, wenn die Zahl noch nicht vorgemerkt war.
- `viewmarks <row> <column>` Geben Sie die vorgemerkten Zahlen für ein Feld aus.
- `enter <row> <column> <value>` Tragen Sie eine Zahl in die angegebene Zelle ein. Sollte diese Zelle bereits belegt sein, überschreiben Sie den alten Wert. Sollte die Eingabe invalide sein, geben Sie eine Fehlermeldung zurück und lassen Sie das Spielfeld unverändert.

Syntaktisch inkorrekte Eingaben werden nicht vorkommen.

Hier ein Beispielverlauf:

```
 |0 1 2|3 4 5|6 7 8|
-+-----+-----+-----+
0|   6 |     |4 2 5|
 |     |     |     |
1|5 7  |     |  8 1|
 |     |     |     |
2|     |4 3  |9    |
-+-----+-----+-----+
3|  5  |9 2  |  7 4|
 |     |     |     |
4|     |3 8 4|     |
 |     |     |     |
5|7 4  |5 6 7|  9  |
-+-----+-----+-----+
6|    2|  1  |     |
 |     |     |     |
7|  3 9|    6|7   8|
 |     |     |     |
8|     |    9|6    |
-+-----+-----+-----+
Command: [enter 3 5 1]
 |0 1 2|3 4 5|6 7 8|
-+-----+-----+-----+
0|   6 |     |4 2 5|
 |     |     |     |
1|5 7  |     |  8 1|
 |     |     |     |
2|     |4 3  |9    |
-+-----+-----+-----+
3|  5  |9 2 1|  7 4|
 |     |     |     |
4|     |3 8 4|     |
 |     |     |     |
5|7 4  |5 6 7|  9  |
-+-----+-----+-----+
6|    2|  1  |     |
 |     |     |     |
7|  3 9|    6|7   8|
 |     |     |     |
8|     |    9|6    |
-+-----+-----+-----+
Command: [enter 3 6 1]
Invalid entry.
Command: [mark 3 6 8]
Marked Value 8 for (3:6)
Command: [mark 3 6 8]
Value 8 was already marked for (3:6)
Command: [mark 3 6 3]
Marked Value 3 for (3:6)
Command: [viewmarks 3 6]
[3, 8]
Command: [unmark 3 6 8]
Unmarked Value 8 for (3:6)
Command: [viewmarks 3 6]
[3]
```

Bestehenskriterien

- Ihr Code kompiliert.
- Ihr Code ist wie beschrieben mit Javadoc Kommentaren versehen.
- Die `SudokuField` Klasse implementiert die Funktionalität, die in der Aufgabe beschrieben wurde.
- Ihr Komandozeilen Interface 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 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 {
```

```
4     private String name_;
5     private int legs_;
6     private int arms_;
7
8
9     Animal(String name, int legs, int arms) {
10      this.name_ = name;
11      this.legs_ = legs;
12      this.arms_ = arms;
13    }
14
15    public Animal(String name) {
16      this(name, 0, 0);
17    }
18
19
20    public String getName() {
21      return this.name_;
22    }
23
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);
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.getLegs(), this.getArms());
29    }
30  }
```

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:

```
1   public class Debug { //K: class falsch geschrieben (ckass)
2
3   ...
4
```

```
/*
...
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

```
/*
Field:
 |0 1 2|3 4 5|6 7 8|
-+-----+-----+-----+
0|  6  |     |4 2 5|
 |     |     |     |
1|5 7  |     |  8 1|
 |     |     |     |
2|     |4 3  |9    |
-+-----+-----+-----+
3|  5  |9 2  |  7 4|
 |     |     |     |
4|     |3 8 4|     |
 |     |     |     |
5|7 4  |5 6 7|  9  |
-+-----+-----+-----+
6|    2|  1  |     |
 |     |     |     |
7|  3 9|    6|7   8|
 |     |     |     |
8|     |    9|6    |
-+-----+-----+-----+
*/
byte[][] sudoku1 = {
{-1,  6, -1, -1, -1, -1,  4,  2,  5},
{ 5,  7, -1, -1, -1, -1, -1,  8,  1},
{-1, -1, -1,  4,  3, -1,  9, -1, -1},
{-1,  5, -1,  9,  2, -1, -1,  7,  4},
{-1, -1, -1,  3,  8,  4, -1, -1, -1},
{ 8,  4, -1,  5,  6,  7, -1,  9, -1},
{-1, -1,  2, -1,  1, -1, -1, -1, -1},
{-1,  3,  9, -1, -1,  6,  7, -1,  8},
{-1, -1, -1, -1, -1,  9,  6, -1, -1}
}
```