Assignment 5

Abgabe bis zum 18. Mai 2021 (Dienstag)um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 18. 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.

Aufgabe 1: LadybugGame 4

Sie sind nun bei der letzten Ladybug Aufgabe angekommen. In der letzten Woche haben Sie die erste spielbare Version des Spiels programmiert, nun sollen Sie ein paar Features hinzufügen und alte Features mit sinnvolleren Programmierkonzepten umschreiben. Auch wenn Sie Ihre eigene Lösung weiterprogrammieren, empfehlen wir Ihnen einen Blick in die Musterlösung zu werfen.

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) Sie haben letzte Woche `Entity` als eine normale Klasse erstellt. Das ist keine wirklich elegante Lösung, da es möglich ist ein `Entity` Objekt zu erstellen. An dieser Stelle ist dies allerdings nicht sinnvoll.

Ändern Sie die `Entity` Klasse zu einem Interface, von dem `Tree`, `Clover` (und falls Sie es letzte Woche schon gemacht haben `Ladybug`) erben. Fügen Sie zwei Klassen `Start` und `End` hinzu, die auch von Entity erben. Jedes `Game` soll genau ein `Start` und genau ein `End` auf dem Spielfeld haben. `Start` und `End` sind der Start- und Endpunkt des Spiels. Game soll das Programm mit einer sinnvollen Nachricht beenden, wenn der Ladybug das `End`-Feld betritt. der Ladybug soll auf dem `Start` Feld starten.

Diese Modifkation der `Game` Klasse erfordert noch keine Anpassung des Kommandozeileninterfaces in der `main` Methode.

b) Modifizieren Sie `Ladybug` so, dass `Ladybug` eine abstrakte Klasse ist und implementieren Sie zwei neue Klassen: `GroundLadybug` und `FlyingLadybug`.

Die `Tree` Klasse soll nun eine Höhe als Eigenschaft bekommen. Die Höhe soll in ganzzahligen Metern (`int`) gespeichert werden. Ein Baum soll maximal 10 Meter hoch sein (und natürlich nicht kleiner als 0 Meter).

Ein `GroundLadybug` soll sich so verhalten wie der `Ladybug`, außer dass dieser, wenn möglich, zwei Schritte nach vorne gehen soll.

Der `FlyingLadybug` soll über Bäume, die bis zu 5 Meter hoch sind, fliegen können, also diese Felder begehen können.

Bearbeiten Sie Ihr Kommandozeileninterface so, dass Sie am Anfang entscheiden können, ob Sie ein `FlyingLadybug` oder ein `GroundLadybug` benutzen wollen. Sie finden zwei Beispiele für Programmabläufe im Anhang A: Beispielabläufe für Aufgabe 1.

c) Versehen Sie alle Methoden und Klassen mit Javadoc-Kommentaren nach den Konventionen aus Assignment 3.

Bestehenskriterien

- Ihr Code kompiliert.
- Ihr Code ist wie beschrieben mit Javadoc-Kommentaren versehen.
- Ihre Klassenstruktur ist, wie in der Aufgabe beschrieben, aufgebaut.
- Die Eigenschaften der verschiedenen Klassen sind korrekt implementiert.
- Ihr Kommandozeilen 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 2: Angestelltenverwaltung

In dieser Aufgabe sollen Sie ein Programm schreiben, das das Abspeichern von Angestellten in einem Array ermöglicht.

In dieser Aufgabe sollen alle Klassen, Interfaces, Methoden und Felder mit Javadoc-Kommentaren nach der Konvention aus Assignment 3 versehen werden (Author Tags an Methoden sind nicht nötig). 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 dasselbe Package zu legen). Sie sollen in der Lage sein, Ihre Packagestruktur Ihrem Tutor zu erklären.

a) Jede/r Angestellte hat folgende Eigenschaften:

- Jede/r Angestellte hat einen Vor-/Nachnamen
- Jede/r Angestellte hat einen Rang (z.B. “Senior Engineer”, “Junior Engineer” oder “CEO”)
- Jede/r Angestellte hat ein Gehalt (auf den Monat gerechnet).
- Jede/r Angestellte hat ein Urlaubskontingent (auf das Jahr gerechnet).

Erstellen Sie eine abstrakte Klasse `Employee`, die Methoden bietet um diese Informationen bereitzustellen. `Employee` soll auch angemessene Konstruktoren haben. Es gibt allerdings drei verschiedene Arten von Mitarbeitenden: Sicherheitspersonal, IT-Personal und Management. Diese unterscheiden sich darin, wie sich ihr Gehalt und ihr Urlaubskontingent errechnen. Erstellen Sie die Methoden, die diese Informationen bereitstellen, als abstrakte Methoden in `Employee`.

Sicherheitspersonal:

- Jedes Sicherheitspersonal hat den Rang “Guard”.
- Das Sicherheitspersonal kann Nacht-, Früh- und Spätschichten belegen (immer auf einen Monat gerechnet). Jede Schicht dauert 8 Stunden.
- Das Gehalt berechnet sich durch die Anzahl der jeweiligen Schichten. Pro Früh- oder Spätschicht gibt es 100 Euro, für eine Nachtschicht gibt es 160 Euro
- Jeder startet mit 20 Urlaubstagen und bekommt für jede Schicht $\frac{1}{4}$ Urlaubstage hinzu (am Ende wird abgerundet).

IT-Personal

- Es gibt “Senior Developer” und “Junior Developer” als Ränge.
- Die Arbeitszeit des IT-Personals wird in Wochenstunden angegeben.
- Jedes IT-Personal hat das Jahr der Einstellung vermerkt. Ein Jahr zählt als absolviert, wenn die Person zum Ende des Kalenderjahres eingestellt war (z.B. bei einem Beitrittsjahr von 2010 hat die Person 11 absolvierte Jahre, von 2010 bis 2020).
- Jedes IT-Personal hat einen Stundenlohn abhängig vom Rang (Junior Developer 25EUR/h, Senior Developer 32EUR/h). Ein “Senior Developer” bekommt pro absolviertem Jahr einen 1EUR/h oben drauf.

- Die bezahlten monatlichen Arbeitsstunden werden berechnet, indem man die Wochenstunden mit 4 multipliziert. Mithilfe dieser Stundenzahl wird das monatliche Gehalt berechnet.
- Ein IT-Angestellter hat immer 22 Urlaubstage im Jahr. Ein "Senior Developer" bekommt je drei absolvierte Jahre einen Urlaubstag dazu, bis zu einem Maximum von 5.

Management

- Es gibt "Project Manager" und "CEO" als Ränge.
- Ein/e Manager/in hat ein Basisgehalt, das von dem zugehörigen Rang abhängig ist(Project Manager: 8000Euro/Monat, CEO: 10000Euro/Monat).
- Ein/e Manager/in hat eine Zahl an erfolgreich abgeschlossenen Projekten. Für jedes erfolgreich abgeschlossene Projekt bekommt ein/e Manager/in eine Gehaltssteigerung von 10%. Diese Gehaltsteigerung arbeitet ohne Zinseszins. Also erhält ein/e Manager/in nach dem 2. Projekt 120% des ursprünglichen Gehaltes, nach dem 3. Projekt 130%, usw.
- Ein/e Manager/in startet mit 20 Urlaubstagen und bekommt für jedes erfolgreich abgeschlossene Projekt 2 Tage dazu.

Schreiben Sie für jede Art an Angestellten eine Klasse, die von `Employee` erbt, die abstrakten Methoden von `Employee` implementiert, angemessene Konstruktoren und ggf. zusätzliche Methoden um spezielle Informationen auslesen zu können enthält. Implementieren Sie eine `public String toString()` Methode auf den drei abgeleiteten Klassen, die eine String-Repräsentation des/der Angestellten ausgibt.

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. Es sollen bis zu 15 Angestellte erstellt werden können. Folgende Kommandos sollen möglich sein:

- `new_guard` Erstellen Sie eine neue Wache (Sicherheitspersonal).
- `new_it` Erstellen Sie ein neues IT-Personal.
- `new_manager` Erstellen Sie eine/n neuen Manager/in.
- `show_employees` Listet alle Angestellten auf.

Sie finden einen Beispielaufruf des Programms in Anhang B: Beispielabläufe für Aufgabe 2.

Bestehenskriterien

- Ihr Code kompiliert.
- Ihr Code ist wie beschrieben mit Javadoc-Kommentaren versehen.
- Ihre Klassenstruktur ist, wie in der Aufgabe beschrieben, aufgebaut.
- Die Eigenschaften der verschiedenen Arten an Angestellten sind korrekt implementiert.
- Ihr Kommandozeilen 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) {

    IntBinTreeNode treeSorted = new SortedIntBinTreeNode(1, null, null);
    IntBinTreeNode treeUnsorted = new UnsortedIntBinTreeNode(1, null, null);
    int[] toInsert = { 4, 2, 6, 8, 0, 2, 1, 5, 1 };
    // for each i in toInsert
    for (int i : toInsert) {
      treeSorted.insert(i);
      treeUnsorted.insert(i);
    }
    System.out.println(treeSorted);
    System.out.println(treeUnsorted);
  }
}
```

`de.uni_hannover.hci.aufgabe3.model.IntBinTreeNode`

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

// Abstract class that represents a binary tree. Inserting and searching are handled
    ↪ by derived classes.
public abstract class IntBinTreeNode {
  protected int content_;
  protected IntBinTreeNode left_, right_;

  public IntBinTreeNode(int content, IntBinTreeNode left, IntBinTreeNode right) {
    this.content_ = content;
    this.left_    = left;
    this.right_   = right;
  }

  public int getContent() {
    return this.content_;
  }

  public void setContent(int content) {
    this.content_ = content;
  }

  public IntBinTreeNode getLeft() {
    return this.left_;
  }
```

```
25
26    public void setLeft(IntBinTreeNode left) {
27      this.left_ = left;
28    }
29
30    public IntBinTreeNode getRight() {
31      return this.right_;
32    }
33
34    public void setRight(IntBinTreeNode right) {
35      this.right_ = right;
36    }
37
38    /**
39     * Inserts integer into appropriate place in tree.
40     * Inserting methodology is dictated by derived classes.
41     *
42     * @param i Integer to insert.
43     */
44    public abstract void insert(int i);
45
46    /**
47     * Looks up, whether integer is contained within tree.
48     * Searching methodology is dictated by derived classes.
49     *
50     * @param i Integer to search for.
51     * @return  True if integer is in tree.
52     */
53    public abstract boolean contains(int i);
54
55    @Override
56    public String toString() {
57      return String.format(
58        "(%s %d %s)",
59        this.left_ == null ? "_" : this.left_.toString(),
60        this.content_,
61        this.right_ == null ? "_" : this.right_.toString()
62      );
63    }
64  }
```

de.uni_hannover.hci.aufgabe3.model.UnsortedIntBinTreeNode

```
1   package de.uni_hannover.hci.aufgabe3.model;
2
3   import java.util.Random;
4
5   //Inserts randomly left or right. Needs to search through both subtrees, as
6   //there is no way of knowing where the node containing i might be.
7   public class UnsortedIntBinTreeNode extends IntBinTreeNode {
8     private static Random rand;
9
10    /**
11     * Randomly inserts to left or right.
12     * If left/right is not null, call insert on that child node.
13     *
14     * @param i Integer to insert.
15     */
16    @Override
```

```
17    public void insert(int i) {
18      if (UnsortedIntBinTreeNode.rand.nextBoolean()) {
19        if (super.left_ == null)
20          super.left_ = new UnsortedIntBinTreeNode(i, null, null);
21        else
22          super.left_.insert(i);
23      } else {
24        if (super.right_ == null)
25          super.right_ = new UnsortedIntBinTreeNode(i, null, null);
26        else
27          super.right_.insert(i);
28      }
29    }
30
31    /**
32     * Looks if integer is in tree.
33     * Looks through both subtrees, as they are not sorted.
34     *
35     * @param i Integer to search for.
36     * @return  True if integer is in tree.
37     */
38    @Override
39    public boolean contains(int i) {
40      return super.content_ == i || (super.left_ != null && super.left_.contains(i))
          ↪ || (super.right_ != null && super.right_.contains(i));
41    }
42  }
```

de.uni_hannover.hci.aufgabe3.model.SortedIntBinTreeNode

```
1   package de.uni_hannover.hci.aufgabe3.model;
2
3   //Inserts integers in a sorted way. Only needs to search in one subtree as a
4   //result.
5   public class SortedIntBinTreeNode extends IntBinTreeNode {
6
7     public SortedIntBinTreeNode(int content, SortedIntBinTreeNode left,
        ↪ SortedIntBinTreeNode right) {
8       super(content, left, right);
9       if ((left != null && left.content_ >= content) || (right != null &&
            ↪ right.content_ <= content)) {
10        System.err.println("Trying to create invalid sorted tree.");
11        System.exit(2);
12      }
13    }
14
15    /**
16     * Inserts integer into the sorted tree.
17     * Smaller Integers will be placed into the left subtree, larger ones into
18     * the right subtree. Equal ones will be ignored
19     *
20     * @param i Integer to insert.
21     */
22    @Override
23    public void insert(int i) {
24      if (i > super.content_) {
25        if (super.left_ == null)
26          super.left_ = new SortedIntBinTreeNode(i, null, null);
27        else
```

```
28          super.left_.insert(i);
29      } else if (i < super.content_) {
30        if (super.right_ == null)
31          super.right_ = new SortedIntBinTreeNode(i, null, null);
32        else
33          super.right_.insert(i);
34      }
35    }
36  }
```

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
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 geschriebene ckass bekommen.
15 ...
16 */
```

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: Beispielabläufe für Aufgabe 1:

Die Legende für das Spielfeld ist:

- `^`, `>`, `v` und `<` = `Ladybug`
- `*` = `Tree` $\leq$ 5m
- `X` = `Tree` $>$ 5m
- `S` = Startfeld
- `G` = Endfeld
- `O` = `Clover`

Beispielablauf mit `GroundLadybug`: In diesem Beispielablauf wählt der Nutzer eine `GroundLadybug`. Nutzen Sie dieses Beispiel auch falls Sie nicht wissen, wie sich eine `GroundLadybug` verhalten soll:

```
Do you want to play a flying ladybug or a ladybug that walks on the ground?
Options(flying,ground): [ground]

XvXXXXX
X *  OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X *  OX
Xv  * X
XXXXXGX
Points=0

> [turnLeft]

XSXXXXX
X *  OX
X>  * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X *  OX
X  >* X
XXXXXGX
Points=0

> [forward]
```

```
XSXXXXX
X *  OX
X  >* X
XXXXXGX
Points=0

> [turnLeft]

XSXXXXX
X *  OX
X  ^* X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X *^ OX
X   * X
XXXXXGX
Points=0

> [turnRight]

XSXXXXX
X *> OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X *  >X
X   * X
XXXXXGX
Points=0

> [pickUp]

XSXXXXX
X *  >X
X   * X
XXXXXGX
Points=11

> [turnRight]

XSXXXXX
X *  vX
X   * X
XXXXXGX
```

```
> [forward]

XSXXXXX
X *   X
X   * X
XXXXXvX

You arrived at the goal.
```

Beispielablauf mit `FlyingLadybug`: In diesem Beispielablauf wählt der Nutzer eine `FlyingLadybug`. Nutzen Sie dieses Beispiel auch falls Sie nicht wissen, wie sich eine `FlyingLadybug` verhalten soll:

```
Do you want to play a flying ladybug or a ladybug that walks on the ground?
Options(flying,ground): [ground]

XvXXXXX
X *  OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
Xv*  OX
X   * X
XXXXXGX
Points=0

> [turnLeft]

XSXXXXX
X>*  OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X >  OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X *> OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X * >OX
X   * X
XXXXXGX
Points=0

> [forward]

XSXXXXX
X *  >X
X   * X
XXXXXGX
Points=0
```

```
> [pickUp]

XSXXXXX
X *  >X
X   * X
XXXXXGX
Points=11

> [turnRight]

XSXXXXX
X *  vX
X   * X
XXXXXGX
Points=11

> [forward]

XSXXXXX
X *   X
X   *vX
XXXXXGX
Points=11

XSXXXXX
X *   X
X   * X
XXXXXvX
Points=11

You arrived at the goal.
```

Anhang B: Beispielabläufe für Aufgabe 2:

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

```
Command: [new_guard]

Adding new guard
Firstname: [Max]
Lastname: [Mustermann]
Night shifts (per month): [8]
Early shifts (per month): [8]
Late shifts (per month): [8]
Added Mustermann, Max to company.

Command: [show_employees]

Mustermann, Max (Guard): Shifts 8/8/8; Monthly salary = 2880 EUR; 26 offdays.

Command: [new_it]

Adding new IT-personel:
Firstname: [Erika]
```

```
Lastname: [Mustermann]
Rank ("JuniorDev" or "SeniorDev"): [SeniorDev]
Year joined: [2010]
Weekhours: [40]
Added Mustermann, Erika to company.

Command: [show_employees]

Mustermann, Max (Guard): Shifts 8/8/8; Monthly salary = 2880 EUR; 26 offdays.
Mustermann, Erika (Senior Developer): Weekhours = 40; Monthly Salary = 6880 EUR; 25
    ↪ offdays.
```