Assignment 5

Abgabe bis zum 26. Mai 2020 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 26. 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. Stattdessen geben wir Ihnen eine Funktionalität vor, welche Sie 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 Gruppenaufgabe werden Sie eine sehr offene Aufgabenstellung bekommen. 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: Sortable Stack

In dieser Aufgabe sollen Sie ein Java Interface für einen sortierbaren Stack entwerfen und zwei Implementierungen für dieses Interface schreiben. Eine der Implementierungen für das Interface basiert auf der Listenaufgabe aus Assignment 3, weshalb Sie eine Musterlösung für die relevanten Teile dieser Aufgabe bekommen.

Eine Erklärung zu Stacks ist hier zu finden: https://de.wikipedia.org/wiki/Stapelspeicher

In dieser Aufgabe sollen alle Klassen, Interfaces, Methoden und Felder mit Javadoc Kommentaren nach der Konvention aus Assignment 3 versehen werden (Author Tags bei 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.aufgabe1` 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) Erstellen Sie ein Java Interface `public interface ISortableIntStack` das folgende Funktionalität vorgibt:

- Das Interface soll einen Stack aus ausschließlich *positiven* `int`s vorgeben. Sollte versucht werden, einen negativen Integer auf den Stack zu legen, soll der Betrag des Integers auf den Stack gelegt werden.
- Das Interface soll die Standardoperationen für einen Stack vorgeben, also `push`, `pop` und `peek`.
- Falls eine Methode fehlschlagen kann, soll diese einen Fehlschlag am Rückgabewert signalisieren können.

- Das Interface soll eine Methode zum **aufsteigenden** Sortieren vorgeben. Das kleinste Element soll somit oben auf dem Stack liegen.
- Das Interface soll einen **unendlich wachsenden** Stack vorgeben, der keine vordefinierte maximale Größe hat.

Sollte die Funktionalität nicht durch die Methoden des Interfaces vorgegeben werden können, so soll diese Funktionalität in entsprechenden Javadoc Kommentaren der Methoden oder des Interfaces vermerkt werden.

b) Wandeln Sie Ihre ListHead Klasse aus Assignment 3 so ab, dass sie `ISortableIntStack` implementiert. Implementieren Sie dazu neue Methoden oder wandeln Sie bereits vorhandene Methoden so ab, dass Sie das Interface implementieren. Schreiben Sie die `public String toString()` Methode so um, dass sie die Liste als Stack formatiert ausgibt (Beispiel in Teilaufgabe d).

c) Erstellen Sie eine `ArrayIntStack` Klasse, die das `ISortableIntStack` Interface implementiert. Der Stack soll mithilfe von einem Array (explizit `int[]`) realisiert werden. Hier ein paar Tipps zu der Implementierung:

- Ein Stack auf einem Array kann mithilfe eines Stackpointers realisiert werden, also ein Integer, der den Index des obersten Elements des Stacks angibt.
- Der Stack soll **unendlich** wachsen können, Arrays sind in Java aber fixed Size. Sie müssen also einen lösung für dieses Problem finden.
- Beachten Sie beim Sortieren des Arrays, dass der Stack aus der Sicht des Nutzers aufsteigend sortiert werden soll, also das Element, das oben auf dem Stack liegt soll das kleinste sein und das Element, das ganz unten liegt, soll am größten sein.

Die Klasse soll die `public String toString()` Methode überschreiben, sodass Sie den Stack in genau demselben Format wie `ListHead` ausgibt.

d) 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. Die erste Eingabe soll abfragen, welche Art von Stack genutzt werden soll. Danach soll der Stack über die Kommandozeile befüllt/geleert werden können. Folgende Kommandos sollen möglich sein:

- `put <number>` Legt Nummer auf den Stack.
- `pop` Nimmt oberste Nummer von dem Stack und gibt sie auf die Kommandozeile aus.
- `peek` Gibt oberste Nummer von dem Stack auf die Kommandozeile aus, ohne sie zu entfernen.
- `sort` Sortiert den Stack aufsteigend.
- `print` Printet den gesamten Stack auf die Kommandozeile.

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

```
What stack type should be used <list, array>: [list]

Command: [push 1]
Pushed 1

Command: [push 3]
Pushed 3

Command: [print]
<TopOfStack>
3
1
<BottomOfStack>

Command: [push 2]
Pushed 2

Command: [sort]
Sorted the Stack

Command: [print]
<TopOfStack>
1
2
3
<BottomOfStack>

Command: [pop]
Popped 1

Command: [peek]
Peeked 2

Command: [pop]
Popped 2

Command: [pop]
Popped 3
```

Bestehenskriterien

- Ihr Code kompiliert.
- Ihr Code ist wie beschrieben mit Javadoc Kommentaren versehen.
- Ihre Klassen implementieren das Interface, so wie in der Aufgabe beschrieben.
- 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: Employee List

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) Jeder Angestellter hat folgende Eigenschaften:

- Jeder Angestellter hat einen Vor-/Nachnamen
- Jeder Angestellter hat einen Rang (z.B. "Senior Engineer", "Junior Engineer" oder "CEO")
- Jeder Angestellter hat ein Gehalt (auf den Monat gerechnet).
- Jeder Angestellter 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 an Mitarbeitern: 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 Früh- und Spätschicht $\frac{1}{4}$ Urlaubstage und für jede Nachtschicht $\frac{1}{2}$ Urlaubstage hinzu.

IT-Personal

- Es gibt "Senior Developer" und "Junior Developer" als Ränge.
- Die Arbeitszeit des IT-Personals wird in Wochenstunden angegeben.
- Jedes IT-Personal hat einen Stundenlohn abhängig vom Rang (Junior Developer 25EUR/h, Senior Developer 32EUR/h).
- Die bezahlten monatlichen Arbeitsstunden werden berechnet, indem man die Wochenstunden mit 4.5 multipliziert. Mithilfe dieser Stundenzahl wird das monatliche Gehalt berechnet.
- Ein IT-Angestellter hat immer 24 Urlaubstage das Jahr.

Management

- Es gibt "Project Manager" und "CEO" als Ränge.
- Ein Manager hat ein Basisgehalt, das von seinem Rang abhängig ist(Project Manager: 8000Euro/Monat, CEO: 10000Euro/Monat).
- Ein Manager hat eine Zahl an erfolgreich abgeschlossenen Projekten. Für jedes erfolgreich abgeschlossene Projekt bekommt ein Manager eine Gehaltssteigerung von 10%. Diese Gehaltsteigerung arbeitet ohne Zinseszins. Also hat ein Manager nach dem 2. Projekt 120% seines Gehaltes, nach dem 3. Projekt 130%, usw.
- Ein Manager 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 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 einen neuen Manager.
- `show_employees` Listet alle Angestellten auf.

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; 28 offdays.

Command: [new_it]

Adding new IT-personel:
Firstname: [Erika]
Lastname: [Mustermann]
Rank ("JuniorDev" or "SeniorDev"): [SeniorDev]
```

```
Weekhours: [40]
Added Mustermann, Erika to company.

Command: [show_employees]

Mustermann, Max (Guard): Shifts 8/8/8; Monthly salary = 2880 EUR; 28 offdays.
Mustermann, Erika (Senior Developer): Weekhours = 40; Monthly Salary = 5760 EUR; 24
    ↪ offdays.
```

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 Angestllten 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.