Assignment 4

Abgabe bis zum 9. Mai 2023 (Dienstag) um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 9. 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.

Für alle durch Sie erstellten Klassen, Methoden und Attribute sind JavaDoc Kommentare anzufertigen. Diese sollen der JavaDoc Konvention im Anhang entsprechen.

Aufgabe 1: Schaltungsimulation

Innerhalb dieser Aufgabe erstellen wir schrittweise einen Halbaddierer, welcher zwei Bits addieren soll. Im Folgenden sind boolean Werte als binäre Zahlen zu verstehen. Der Wert '1' wird durch *true* repräsentiert, der Wert '0' durch *false*.

a) Erstellen Sie die Klasse Gate. Implementieren Sie die Methode `public boolean evaluate(boolean a, boolean b)`, welche in jedem Fall *false* zurück gibt. Implementieren Sie die Methode `public void table()`, die eine Wahrheitstabelle wie in Fig. 1 ausgibt, in dem in zwei geschachtelte Schleifen über die Wahrheitswerte iterieren (überlegen Sie sich eine elegante Lösung) und das Ergebnis unter Benutzung von `evaluate()` errechnet und ausgegeben wird.

Die Klasse hat ein Attribut `String symbol` welches das Symbol der Operation enthält. Das Attribut soll auch zur Ausgabe der Wahrheitstabelle herangezogen werden.

b) Erstellen Sie die Klasse NotGate, welche von der Klasse Gate erbt. NotGate realisiert einen Inverter. Überschreiben Sie die Funktion `public boolean evaluate(boolean a, boolean b)`, sodass der invertierte Wert von 'a' zurückgegeben wird. Der Wert 'b' ist zu ignorieren.

Implementieren Sie ebenfalls die Klassen AndGate und OrGate, welche das Und-Gatter und das Or-Gatter darstellen und von Gate erben. Überschreiben Sie die Methode `public boolean evaluate(boolean a, boolean b)` entsprechend, sodass die Ver-UND-ung bzw. Ver-ODER-rung der beiden Werte 'a' und 'b' zurückgegeben wird.

c) Realisieren Sie die Klasse XorGate, welcher von Gate erbt und das Xor-Gatter darstellt. Machen Sie sich zunächst darüber Gedanken, wie man ein Xor-Gatter durch And-, Or und Not-Gattern ausdrücken kann. Hilfreich ist hierfür die Wahrheitstabelle in Figure 1. Erstellen Sie einen Konstruktor, der die benötigten Gatter innerhalb der Klasse speichert. Überschreiben Sie anschließend die Methode `public boolean evaluate(boolean a, boolean b)`, die das Xor-Gatter mithilfe der gespeicherten Gatter umsetzt. Nutzen Sie hierfür die evaluate-Methoden der gespeicherten Gatter.

A	B	Y = A ⊻ B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1: Wahrheitstabelle eines Xor-Gatters

d) Nun soll die Klasse HalfAdder erstellt werden, welche einen Halbaddierer darstellt. Dieser erbt **nicht** von Gate nutzt diese jedoch. Speichern Sie die benötigten Gates in einem Gate-Array `Gate[]`. Informieren Sie sich unter https://de.wikipedia.org/wiki/Halbaddierer, welche Gatter Sie für den Übertrag c und die Summe s benötigen. Anschließend realisieren Sie die Methode `public boolean[] add(boolean a, boolean b)`, welche zwei Binärzahlen addieren soll. Der erste Eintrag des Rückgabewerts speichert die Summe s, der zweite den Übertrag c.

Zusätzlich implementieren Sie eine Methode `void info()`, die alle gespeicherten Gates inklusive der Wahrheitstabellen ausgibt.

e) Implementieren Sie innerhalb der Main-Methode der Main-Klasse eine Konsoleneingabe. Das Programm fragt zunächst nach den zwei boolean Werten 'a' und 'b'. Anschließend sollen diese Werte mithilfe des Halbaddierers addiert und das Ergebnis auf der Konsole ausgeben werden. Anschließend beendet sich das Programm. Dies kann wie folgt aussehen.

```
1 Geben Sie den ersten boolean Wert an: true
2 Geben sie den zweiten boolean Wert an: true
3 Die Summe lautet 0, der Übertrag beträgt 1
```

Anschließend geben Sie noch die Information über die enthaltenen Gatter mittels `info()` aus.

Aufgabe 2: Marketplace 3

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 *Marketplace2.zip* herunterladen.

a) Erstellen Sie die Klasse Admin, welche von der User Klasse erbt. Damit man unterscheiden kann, ob es sich um einen User oder Admin handelt, wird der User Klasse das Attribut isAdmin hinzugefügt, welches stets *false* ist. Innerhalb der Admin Klasse wird es jedoch auf *true* gesetzt.

Modifizieren Sie die Main-Methode, sodass des Marketplace ebenfalls ein Admin hinzugefügt wird.

b) Erstellen Sie die Methode `public User login()`. Diese Methode fragt liest bis zu 3-mal einen Benutzernamen und Password ein. Ist ein User mit dem gleichen Benutzernamen und dem gleichen Password im Marketplace vorhanden, soll diese Methode diesen User zurückgeben. Schlägt der Login 3-mal fehl, soll sich das Programm beenden. Dies kann mit `System.exit(0)` erreicht werden.

c) Nun soll ein User in der Lage neue Items dem Marketplace hinzuzufügen, sie zu entfernen, den Marketplace (u.a auch nur einzelne Kategorien) anzusehen oder das Program zu beenden. Implementieren Sie die Methode `public void handleUser()`, welche diese Aktionen über die Konsole umsetzt. Dies könnte wie folgt aussehen:

```
Wählen Sie aus den folgenden Optionen:
1. Item hinzufügen
2. Item entfernen
3. Marketplace anssehen
4. Programm beenden

3

Welche Kategorie wollen Sie sich anschauen:
1. Alle
2. CLOTHES
3. ELECTRONICS
...
X. Programm beenden

Es wurden folgende Kleidung gefunden:
[Item: Tasche; Preis: 10.99; Verkäufer: Max Mustermann; Beschreibung: Eine Tasche,
    ↪ Kategorie: CLOTHES]

1. Item hinzufügen
2. Item entfernen
3. Marketplace anssehen
4. Programm beenden
```

d) Admins sind in der Lage Items und User aus dem Marketplace zu entfernen. Implementieren Sie die Methode `public void handleAdmin()`, welche diese Aktionen über die Konsole umsetzt. Dies könnte wie folgt aussehen:

```
Wählen Sie aus den folgenden Optionen:
1. Nutzer entfernen
2. Item entfernen
3. Programm beenden

2

Es wurden folgende Items gefunden, welches Item wollen Sie entfernen?
1. [Item: Tasche; Preis: 10.99; Verkäufer: Max Mustermann; Beschreibung: Eine Tasche,
    ↪ Kategorie: CLOTHES]
2. [Item: Nachhilfe; Preis: 10.99; Verkäufer: Max Mustermann; Beschreibung: Es gebe
    ↪ Nachhilfe für Oberstufenschüler, Kategorie: SERVICES]
...
X. Programm beenden

2

Item 2 wurde erfolgreich entfernt.

1. Nutzer entfernen
2. Item entfernen
3. Programm beenden
```

e) Passen Sie die Main-Methode innerhalb der Marketplace-Klasse derartig an, sodass abhängig von dem Login die handleUser- bzw. handleAdmin-Methode aufgerufen wird.

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 `Debug04.zip`.

a) Der Code enthält 5 fehlende bzw. fehlerbehaftete Codezeilen. 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        ^
13 **************
14 Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
       ↪ geschriebene ckass bekommen.
15 ...
16 */
```

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  Max Musterstudierender
 * @version 2023 May 03
 */
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      Max Musterstudierender
     * @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      Max Musterstudierender
     * @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.
```