Übungsaufgaben „Objektorientierte Programmierung“, SS2018, Georg Ringwelski

Stand: 29.1.19

Durch die die Bearbeitung und Präsentation der Übungen erhalten Sie die Prüfungsvorleistung (VT), die Sie berechtigt an der Klausur zum Modul teilzunehmen. Für die VT ist die erfolgreiche Präsentation Ihrer Lösung der Aufgaben 3 in KW 17 und der zu Aufgabe 6 in KW 25 erforderlich.

Aufgabe 1 (KW 13): Implementierung Währungsrechner

Implementieren Sie eine Java Applikation, mit der Geldbeträge in unterschiedliche Währungen umgerechnet werden können.

- Entwerfen Sie als Skizze auf Papier eine graphische Oberfläche für die Applikation. Dabei können durchaus mehrere verschiedene Fenster zum Einsatz kommen. Folgende Anforderungen sollen erfüllt werden
  - Eingabe neuer Währungen
  - Änderung von Wechselkursen
  - Umrechnung beliebiger Beträge in beliebige Währungen
- Implementieren Sie diese Oberfläche mit JavaFX.
- Implementieren Sie getrennt davon die Programmlogik, in der die nötigen Umrechnungen durchgeführt werden.
- Implementieren Sie Tests für die Programmlogik.
- Verbinden Sie die graphische Oberfläche mit der Programmlogik, um eine funktionierende Applikation zu erhalten.

Aufgabe 2 (KW 15): Deployment und Sichtbarkeiten

Machen Sie ein auslieferbares Produkt aus Ihrer Software

- Extrahieren Sie eine möglichst kompakte Schnittstelle (interface) für Ihre Programmlogik und integrieren diese so in Ihren Code, dass im GUI ausschließlich diese Schnittstelle zur Interaktion mit der Programmlogik verwendet wird.
- Definieren Sie je ein package für Ihre Applikation und für Ihre Tests.
- Definieren Sie für alle Attribute, Methoden und Klassen in Ihrer Applikation minimale Sichtbarkeiten.
- Erstellen Sie für die Applikation eine ausführbare jar-Datei, die sich per Doppelklick als Windows-Applikation starten lässt
- (nur für Cracks) Versuchen Sie das Deployment dieser Applikation innerhalb einer HTML-Seite in einem web-Browser auf dem lokalen Computer. Erstellen Sie dazu eine geeignete JNLP-Datei.

Aufgabe 3 (KW 17): Änderung der Implementierung: Aufzählungstyp, Daten speichern und Dokumentation

Präsentation 1 für die Prüfungsvorleistung

Erweitern Sie die Funktionalität Ihrer Applikation, verbessern ihr Softwaredesign und dokumentieren sie.

- Implementieren Sie die Schnittstelle aus Aufgabe 2 mit einem Aufzählungstyp (enum), in dem alle Währungen eingetragen sind. Überprüfen Sie mit Ihren Tests, ob auch mit dieser Implementierung alles richtig funktioniert.
- Erweitern Sie Ihre GUI um die Möglichkeit zur dynamischen Auswahl einer Implementierung der Programmlogik (enum oder Klassen aus Aufg. 1). Bei der Verwendung des enum ist das Hinzufügen von neuen Währungen nicht möglich. Deaktivieren Sie also bei dieser Auswahl die entsprechende Eingabemöglichkeit im GUI.
- Realisieren Sie Ihre Applikation (nur die ausführbare jar) so, dass geänderte Wechselkurse beim Beenden der Applikation in einer Datei gespeichert und beim nächsten Start wieder geladen werden. Achten Sie auf einen kontrollierten Programmablauf, auch wenn die Datei nicht existiert.
- Dokumentieren Sie Ihre Applikation mit javadoc in HTML-Dateien. Achten Sie auf die Verlinkung mit der Dokumentation der Java API von oracle.

Aufgabe 4 (KW20): Generics

Vergleichen Sie die Performanz verschiedener Implementierungen von java.util.Collection<T> bzgl. ihrer Methoden add und remove

Definieren Sie dazu für jede der beiden Methoden eine separate generische Klasse als Testszenario. Der zu testende Typ T muss dabei als Typparameter angegeben werden, z.B.

```
class AddSzenario<S, T  extends Collection<S>> {
     void testfall(T t, S s){…  t.add(s) …}

…}
```

- In den Szenarien implementieren Sie einen geeigneten Performanztest (Laufzeit in msec messen) und speichern die Ergebnisse in Attributen ab. (Hinweis: eine Analyse, aus der sich allgemeine Aussagen über das Laufzeitverhalten ableiten lassen ist nicht notwendig. Wählen Sie Ihren Testfall aber möglichst so, dass unterschiedliche Werte für die unterschiedlichen Collections entstehen.)
- Definieren Sie eine Anwendung zum Testen der Szenarien für die Klassen HashSet<T>, TreeSet<T>, ArrayList<T> und LinkedList<T>. Dabei muss jeder Datentyp in jedem Szenario getestet werden, bevor aus den Attributen der verschiedenen Szenario-Objekte, Laufzeiten ausgelesen und übersichtlich dargestellt werden.

Aufgabe 5 (KW 22): Multithreading

Wiederholen Sie den Performanztestest aus Aufgabe 4 in einer nebenläufigen Anwendung. Beobachten Sie die Laufzeit und die Ergebnisse mit und ohne Synchronisierung

- Refaktorisieren Sie Ihre Applikation aus Aufgabe 4, so dass alle Tests nebenläufig ausgeführt werden. Vergleichen Sie die Testergebnisse (also die Laufzeiten) dieser Realisierung mit denen aus Aufgabe 4.
- Um sinnvolle Ergebnisse zu erhalten ist die Synchronisierung der Threads nötig. Die Threads müssen (in beliebiger Reihenfolge) nacheinander ausgeführt werden. Realisieren Sie das durch Synchronisation.

Aufgabe 6 (KW 25): Innere Klassen, Multithreading

Präsentation 2 für die Prüfungsvorleistung

Implementieren Sie eine GUI mit Buttons zum Starten und Beenden von 5 nebenläufigen „Stoppuhren“. Durch einen Klick auf einen Start-Button beginnet ein Thread in seinem zugeordneten Textfeld seine eigene Laufzeit auszugeben, bis der zugehörige Stop-Button gedrückt wird. Dann steht die Uhr und man kann die abgelaufene Zeit ablesen. Bei erneutem Start läuft die Uhr weiter. Alle Uhren können parallel laufen und unabhängig voneinander gestartet oder angehalten werden.

Realisieren Sie für jeden der 5 Threads den EventHandler auf unterschiedliche Weise:

a) Als Separate Klasse in separater Datei
b) Als nicht-statische Innere Klasse
c) Als lokale Innere Klasse
d) Als anonyme innere Klasse
e) Als Lambda-Ausdruck