Hochschule für Angewandte Wissenschaften Hamburg
University of Applied Sciences Hamburg
Fakultät Technik und Informatik
Department Informatik

HAW HAMBURG

Aufgabe A4.0 Vorbereitungsaufgabe: Vererbung, dynamische Bindung oder auch nicht ;-)

Diese Aufgabe ist eine Vorbereitungsaufgabe. Auch wenn die Bearbeitung dieser Aufgabe Pflicht ist, wird Ihre Lösung (u.U.) nicht im Labor kontrolliert. Sie können aber auf jeden Fall Verständnisfragen im Labor stellen.

Analog zu den in der Vorlesung durchgeführten Übungen finden Sie bei dieser Aufgabe mehrere Klassen vor. Schauen Sie sich alles gründlich an. Überlegen Sie sich bevor Sie die Methoden der Klassen des TestFrame1 und TestFrame2 anstarten, was für eine Ausgabe Sie erwarten und kontrollieren Sie was Sie wirklich bekommen. Klären Sie mögliche Unsicherheiten.

Versuchen Sie auch die Bedeutung von
- ….class
- ….getClass()
- ….getSimpleName()

zu ergründen.

Falls Sie hierzu weitere Informationen in der Java-API nachlesen wollen, die Methode getClass() "kommt" aus der Klasse Object und die Methode getSimpleName() "kommt" aus der Klasse Class.
Die API für Java7 finden Sie unter:
`https://docs.oracle.com/javase/7/docs/api/` verständlicher
und die API für Java8 finden Sie unter:
`https://docs.oracle.com/javase/8/docs/api/` aktueller
Das Klassen-Literal .class wird u.a. in der Java Language Specification (15.8.2. Class Literals) beschrieben.

Aufgabe A4.1 (Karten-)Hand

Entpacken Sie die komprimierte Zusammenstellung von gegebenen Code `Z4x1.zip` und lösen Sie die Aufgabe A4.1
Nach dem Entpacken finden Sie 4 Klassen vor:

- PalindromeTester ist ein Code-Template in dem Sie Ihre Lösung einbauen sollen.
- TestFrame ist eine Möglichkeit Ihre Lösung anzustarten und interaktiv zu testen.
- ProposalForYourTestFrame soll Ihnen helfen Ihre Lösung anzustarten und mit eigenen Tests reproduzierbar zu testen.
- UnitTestFrame ist ein einfacher Unit-Test, der als Akzeptanz-Test konzipiert ist und Ihnen eine gewisse Sicherheit vermitteln soll, dass Sie die Aufgabe auch wirklich gelöst haben.

Zum Zeitpunkt der Aufgabenstellung kennen wir noch keine "Collections" und insbesondere noch keine "ArrayList". Für diese Aufgabe dürfen Sie nur Arrays verwenden und insbesondere keine "ArrayList".

Die (Spiel-)Karten, die ein Spieler auf der Hand hält werden auch oft (Karten-)Hand genannt. Schreiben Sie eine Klasse **`Hand`**, die dies unterstützt. Die Klasse soll einen Konstruktor aufweisen, der beliebig viele Karten entgegen nehmen kann. Diese Karten sind dann (zunächst) die Hand.
Weiterhin sollen mit einer Prozedur **`add()`** beliebig viele Karten hinzugefügt werden können. Entweder als Karten selbst oder in Form einer anderen Hand.

Die sondierende Funktion **`isSuited()`** soll einen Wahrheitswert abliefern für die Aussage, ob alle Karten von einer Farbe sind.

Schließlich soll die Funktion **`getHandCards()`** alle Karten, die in der Hand enthalten sind, abliefern und die Prozedur **`setHandCards()`** die aktuelle Hand auf beliebig viele als Parameter übergebende Karten setzen.

Die Implementierung der Methoden **`equals()`**, **`toString()`** und **`hashCode()`** ist für die Klasse **`Hand`** bzw. diese Aufgabe nicht gefordert.

Aufgabe A4.2 Konten

In dieser Aufgabe geht es um Bankkonten (**`BankAccount`**). Es sollen Sparkonten (mit Zinsausschüttung - **`SavingsAccount`**), Girokonten (**`CurrentAccount`**) und Überweisungen zwischen Girokonten unterstützt werden. Für diese Aufgabe gilt! :

- Es wird mit Cent-Beträgen und auf den Cent genau gerechnet.
- Die Bank rundet immer zu ihrem Vorteil.
- Konten dürfen zu keinem Zeitpunkt negative Kontostände annehmen.

Implementieren Sie einen Daten-Typ **`BankAccount`**.
Dieser Typ soll einen Konstruktor aufweisen, der eine ID (z.B. IBAN) vom Typ **`String`** und einen Startbetrag in Cent vom Typ **`long`** entgegen nimmt.
Ferner soll es einen zweiten Konstruktor geben, der eine ID (z.B. IBAN) vom Typ **`String`** entgegen nimmt. In diesem Fall soll der Startbetrag 0 sein.
Es soll eine Methode **`withdraw()`** für das Abheben eines Betrags in Cent vom Typ **`long`** sowie eine Methode **`deposit()`** für das Einzahlen eines Betrags in Cent vom Typ **`long`** geben.

Implementieren Sie einen Daten-Typ **`SavingsAccount`**, der ein Sparkonto beschreibt. Sparkonten sind Bankkonten! Dieser Typ soll einen Konstruktor aufweisen, der eine ID (z.B. IBAN) vom Typ **`String`**, einen Startbetrag in Cent vom Typ **`long`** und einen Zinsatz in Promille vom Typ **`int`** entgegen nimmt.
Es soll eine Methode **`giveInterest()`** für eine Zinsausschüttung geben. Der Aufruf von **`giveInterest()`** soll den jeweiligen Kontostand um die Zinsen erhöhen.

Implementieren Sie einen Daten-Typ **`CurrentAccount`**, der ein Girokonto beschreibt. Girokonten sind Bankkonten! Dieser Typ soll einen Konstruktor aufweisen, der eine ID (z.B. IBAN) vom Typ **`String`** und einen Betrag für eine Standardgebühr in Cent vom Typ **`int`** entgegen nimmt.
Auf/mit Girokonten sind Überweisungen möglich. Allerdings wird für jedes Abheben (auch als Teil einer Überweisung) die Standardgebühr für das jeweilige Konto fällig, die sofort vom Konto abgezogen wird.

Implementieren Sie einen Daten-Typ **`TransferManager`**, dessen Objekte es ermöglichen Geld von einem Girokonto auf ein anderes Girokonto zu überweisen.
Es soll eine Methode **`transfer()`** für das konkrete Überweisen geben. Diese Methode soll als Parameter das Quell-Giro-Konto, das Ziel-Giro-Konto und den zu überweisenden Betrag in Cent vom Typ **`long`** aufweisen.

Alle von Ihnen zu implementierenden Klassen, die Bankkonten beschreiben, sollten die Methoden ~~**`equals()`**~~, **`toString()`** ~~und **`hashCode()`**~~ aufweisen.
Wie in der Vorlesung besprochen, darf

- ~~die Methode **`hashCode()`** in der (unzulässig) vereinfachten Form implementiert werden, da **`hashCode()`** bisher noch nicht in der Vorlesung besprochen wurde.~~
- die Methode **`toString()`** in der vereinfachten Form implementiert werden.

In der **`toString()`**-Methode werden Sie vermutlich **`String.format()`** verwenden wollen. Wie in der Vorlesung besprochen, können Sie Ihr Wissen von **`printf()`** übertragen bzw. Sie können sich

```
System.out.printf( parameter );
```

als

```
System.out.print( String.format( parameter ));
```

Vorstellen.

Aufgabe A4.3 Längste (Satzfragment-)Palindrom finden

Sie dürfen die Ergebnisse von A3.4 wieder verwenden bzw. geeignet modifiziert nutzen.

Schreiben Sie eine Klasse `PalindromeFinder`, die sowohl einen parameterlosen Konstruktor als auch einen Konstruktor mit einem String als Parameter unterstützen soll.
Die von Ihnen zu schreibende Methode `String getLongestPalindrom()` hat die Aufgabe in einem Text, der entweder

mit dem String-Parameter im Konstruktor

oder

mit der von Ihnen zu schreibenden Methode `void setText( String )`

gesetzt werden kann, das längste "Textfragment-Palindrom" zu bestimmen.
Bei der Bestimmung des "Textfragment-Palindroms" sollen alle(!) Zeichen innerhalb des gegebenen Textes berücksichtigt werden. Lediglich die Groß-/Klein-Schreibung der Buchstaben soll ignoriert werden. Sollte es nicht möglich sein, ein Palindrom mit einer Mindestlänge von einem Zeichen zu bestimmen, dann soll `null` zurückgegeben werden. Sollte die Lösung nicht eindeutig sein, so können Sie ein beliebiges der existierenden längsten Textfragment-Palindrome als Ergebnis zurückgeben.
Die im TestFrameForA4x3 gegeben Methode doTest() markiert im ersten Testtext alle "Textfragment-Palindrom" mit einer Mindestlänge von zwei Zeichen. (Für die Markierung wurde abweichend von der Aufgabenstellung eine Mindestlänge von zwei Zeichen gewählt, weil sonst alles hätte markiert werden müssen und damit die Markierung als Verdeutlichung wertlos gewesen wäre)
Schließlich soll die Methode `String getText()` den aktuell zu untersuchenden Text abliefern. Also den Text der entweder über den Konstruktor oder mit setText() gesetzt wurde.

Wir können noch keine Rekursion – rekursive Lösungen werden nicht akzeptiert.

Die Implementierung der Methoden **`equals()`**, **`toString()`** und **`hashCode()`** ist für die Klasse **`PalindromeFinder`** bzw. diese Aufgabe nicht gefordert.

Freiwillige Zusatzaufgaben

Es folgen freiwillige Zusatzaufgaben. D.h. diese Aufgabe ist freiwillig ;-).
Wenn Sie diese freiwillige Zusatzaufgabe freiwillig lösen, dann haben Sie den "Gewinn", dass Sie mehr geübt haben und dass Sie Ihre Lösung für diese freiwillige Zusatzaufgabe im Labor besprechen können (sofern Zeit ist – Pflichtaufgaben haben Vorrang).