Praktikum 2

Aufgabe 1 (Funktionalität in abs. Oberklasse auslagern; Interface; toString und equals implementieren):

In dieser Aufgabe sollen die Gemeinsamkeiten der beiden Klassen *Payment* und *Transfer* in eine gemeinsame, abstrakte Oberklasse ausgelagert werden. Legen Sie hierfür zunächst eine neue Klasse *Transaction* im Java-Package *bank* an und passen Sie die drei Klassen entsprechend an:

- 1. Payment und Transfer sollen von der abstrakten Klasse Transaction erben.
- 2. Überlegen Sie, welche Klassenattribute sinnvollerweise in *Transaction* ausgelagert werden sollten und passen Sie ggf. die Sichtbarkeit der Klassenattribute an.
- 3. Passen Sie die Konstruktoren an, um redundaten Code zu vermeiden.
- 4. Passen Sie die Getter- und Settermethoden an, um redundaten Code zu vermeiden.

Im nächsten Schritt soll ein Interface *CalculateBill* im Java-Package *bank* angelegt werden, das eine einzige parameterlose Methode *calculate*(), die einen *double*-Wert zurückliefert, enthalten soll. Diese Methode soll von *Payment* und *Transfer* implementiert werden. Überlegen Sie hierfür, an welcher Stelle das *implements*-Statement am meisten Sinn ergeben würde.

Die Semantik der calculate()-Methode soll bei den beiden Klassen unterschiedlich implementiert werden:

Payment (in beiden Fällen fallen "Kosten" an, falls *interest* > 0):

Sollte der Betrag positiv sein und somit eine Einzahlung repräsentieren, soll der Wert des *incomingInterest*-Attributes prozentual von der Einzahlung abgezogen und das Ergebnis zurückgegeben werden.

Beispiel: 1000€ einzahlen, incomingInterest = 0.05 => 950€

Sollte der Betrag negativ sein und somit eine Auszahlung repräsentieren, soll der Wert des *outgoingInterest*-Attributes prozentual zu der Auszahlung "hinzuaddiert" und das Ergebnis zurückgegeben werden.

Beispiel: -1000€ auszahlen, outgoingInterest = 0.1 => -1100€

Die Ergebnisse sind immer aus Sicht der Bank bzw. des Bankkontos zu interpretieren.

Transfer:

Da in diesem Praktikumsszenario bei *Transfer*-Objekten keine eingehenden bzw. ausgehenden Zinsen anfallen sollen, soll die Methode *calculate()* den Betrag unverändert zurückgeben.

Im nächsten Schritt soll die Methode *printObject*() durch die Methode *toString*() der Klasse *Object* ersetzt werden. Überschreiben Sie diese Methode in allen drei Klassen (*Payment, Transfer* und *Transaction*) und vermeiden Sie redundanten Programmcode. Achten Sie darauf, dass der richtige (berechnete) Wert für *amount* (s. oben) ausgegeben wird.

Abschließend soll die Methode *equals(Object)* der Klasse *Object* ebenfalls in allen drei Klassen überschrieben und redundanter Programmcode vermieden werden. Für die Überprüfung mit *equals(Object)* sollen alle Klassenattribute verwendet und geprüft werden.

Aufgabe 2 (Dokumentation mit Hilfe von Javadoc):

Ersetzen Sie sämtliche Dokumentation aus Pratikum 1 mit korrektem *Javadoc* (dies gilt für Klassendefinitionen, Klassenattribute und Methoden) und verwenden Sie sinnvolle *Javadoc-Tags*, um eine möglichst aussagekräftige Dokumentation des gesamten Programmcodes zu gewährleisten.

Aufgabe 3 (Testen der Funktionalität):

In dieser Aufgabe soll die in Aufgabe 1 erstellte Funktionalität mit Hilfe einer *main-*Methode getestet werden.

Erstellen Sie hierfür eine Klasse *Main* im *default package* (oder verwenden Sie die alte Klasse wieder). Die Klasse *Main* soll eine *main*-Methode enthalten und somit als Einstiegspunkt für folgende Tests fungieren:

Erzeugen Sie Objekte der Klassen *Payment* und *Transfer* und verwenden Sie dabei alle zur Verfügung stehenden Konstruktoren (inkl. *Copy-Konstruktor*). Erzeugen Sie insb. auch Payment-Objekte für sowohl Ein- als auch Auszahlungen.

Testen Sie nun mit Hilfe der *calculate*()-Methode, ob die Berechnungen für *Payment-* und *Transfer-* Objekte korrekt funktionieren. Geben Sie hierfür die Werte auf der Konsole aus.

Abschließend sollten auch die in Aufgabe 1 implementierten toString()-Methoden getestet werden.

Aufgabe 4 (Erstellung von UML-Klassendiagrammen):

Erstellen Sie für alle bisher erstellten Klassen und Interfaces syntaktisch korrekte UML-Klassendiagramme. Achten Sie insbesondere auch auf mögliche Beziehungen zwischen den Klassen und Interfaces und stellen Sie diese UML-konform dar.

Für die Erstellung der UML-Klassendiagramme können Sie externe Tools/Programme verwenden oder diese selber auf einem Blatt Papier zeichnen und während der Praktikumsabnahme vorzeigen. (*Hinweis*: Diese UML-Klassendiagramme sollen im nächsten Praktikum ergänzt bzw. erweitert werden.)