

Musterlösung 6

Aufgabe 1 (Vererbung)

Hier die zentralen Punkt der Lösung:

- Es existiert eine Klasse Bank, welche die Konten, Filialen und das Protokoll verwaltet.
- Die Klasse Filiale wiederum verwaltet Kassen, Beratungsschalter, Geldautomaten und Kunden.
- Kassen, Beratungsschalter und Geldautomaten werden abgeleitet von der Klasse AbstrakterSchalter, welche gemeinsame Fähigkeiten kapselt. Der Unterschied zwischen einem Beratungsschalter auf der einen und Kassen und Geldautomaten auf der anderen Seite liegt in der gemeinsamen Schlange aller Beratungsschalter. Diese gemeinsame Schlange wird als Attribut in der Klasse Filiale gehalten.
- Die Klassen Kasse und Geldautomat unterscheiden sich dann nur noch im Konstruktor und der Methode `bearbeite()`.

Sie finden die Jar-Datei unter:

<http://www.inf.fu-berlin.de/lehre/WS05/ALP3/loesungen/U6.A1.jar>

(entpacken und den Testfall U6.A1.Test mittels JUnit ausführen)

Aufgabe 2 (Vergleichen)

```
/**
 * Nachbildung von Haskell's Typklasse Eq. Es muss entweder equal oder unequals
 * ersetzt werden.
 */
abstract class Eq<A> {
    boolean equal(A a) {
        return !unequal(a);
    }

    boolean unequal(A a) {
        return !equal(a);
    }
}

/**
 * Nachbildung von Haskell's Typklasse Ord. Ein Klasse, die von Ord erbt, drückt
 * damit aus, dass ihre Objekte mit A-Objekten vergleichbar sind. Es muss
 * mindestens lessOrEq und entweder equal oder unequal implementiert werden
 * (sonst zyklische Aufrufbeziehung).
 */
abstract class Ord<A extends Ord<A>> extends Eq<A> {
    public enum Ordering {
        EQ, LT, GT
    }

    Ordering compare(A a) {
        if (equals(a))
            return Ordering.EQ;
        if (lessOrEq(a))
            return Ordering.LT;
        return Ordering.GT;
    }

    boolean lessOrEq(A a) {
        return compare(a) != Ordering.GT;
    }

    boolean less(A a) {
        return compare(a) == Ordering.LT;
    }

    boolean greaterOrEq(A a) {
        return compare(a) != Ordering.LT;
    }

    boolean greater(A a) {
        return compare(a) == Ordering.GT;
    }

    Ord<A> max(A a) {
        return greaterOrEq(a) ? this : a;
    }

    Ord<A> min(A a) {
        return lessOrEq(a) ? this : a;
    }
}
```

```
/** Beispielklasse Cardinal */  
  
class Cardinal extends Ord<Cardinal> {  
    private int value;  
  
    Cardinal(int c) {  
        if (c < 0)  
            throw new IllegalArgumentException("Cardinal muss > 0 sein.");  
        value = c;  
    }  
  
    boolean equal(Cardinal c) {  
        return c.value == value;  
    }  
  
    boolean lessOrEq(Cardinal c) {  
        return value <= c.value;  
    }  
  
    public String toString() {  
        return Integer.toString(value);  
    }  
}
```