Übungen zur Vorlesung Programmieren 2
FHWS – Fakultät für Informatik und Wirtschaftsinformatik

Übungsblatt 14

Aufgabe 1: (Klasseninvarianten)

a) Was ist eine Klasseninvariante (in einem Satz)?

b) Gegeben sei eine Klasse Person mit den Attributen *vorname* und *nachname*. Der Softwarevertrag zu der Klasse beinhaltet, dass Vorname und Nachname immer gesetzt sein müssen. Formulieren Sie die entsprechende Klasseninvariante.

c) Gegeben sei folgende Klasse.

```
public class Person
{
  private String vorname;
  private String nachname;

  public Person() {
  }
  public void init(String id) {
    …
  }
  public String getVorname() {
    return vorname;
  }
  public String getNachname() {
    return nachname;
  }
}
```

Durch Aufruf der init()-Methode wird *vorname* und *nachname* des Personenobjekts anhand der übergebenen id durch Daten aus einer Datei befüllt.

Weshalb wird die Klasseninvariante durch obige Implementierung verletzt?

d) Die Personen-Klasse aus Aufgabenteil c) wurde wie folgt verändert **(fett hervorgehoben)**:

```
public class Person
{
    private String vorname;
    private String nachname;

    public Person(String id) {
      init(id);
    }
    public void init(String id) {
      …
    }
    public String getVorname() {
      return vorname;
    }
```

```
    public String getNachname() {
      return nachname;
    }
    public void setVorname(String vorname) {
      this.vorname = vorname;
    }
    public void setNachname(String nachname) {
      this.nachname = nachname;
    }
}
```

Wie kann die Klasseninvariante jetzt verletzt werden?

Aufgabe 2: (Komplexe Datentypen vs. Klassen)

a) Legen Sie ein package *prozedural* an!
   Erstellen Sie je einen komplexen Datentypen für einen Kreis und für ein Quadrat in dem package *prozedural*. Schreiben Sie ferner eine Klasse Flächenberechner, die eine Methode berechneFläche anbietet. Die Methode soll ein Object entgegennehmen. Egal ob das übergebene Object ein Kreis oder ein Quadrat ist, soll die entsprechende Fläche korrekt berechnet und zurückgegeben werden.
   Legen Sie in einem Hauptprogramm ein Objekt von Kreis an und lassen Sie die Fläche berechnen.

   Hinweis:
   Ob ein Objekt von der Klasse Kreis ist, lässt sich bspw. wie folgt überprüfen:

```
Object o;
…
if(o.getClass() == Kreis.class) …
```

b) Legen Sie ein package *oo* an!
   Legen Sie eine Klasse Circle und eine Klasse Square in dem package *oo* an. Die Klassen sollen von einem gemeinsamen Interface Shape, die Methode calculateArea() (korrekt) implementieren.
   Legen Sie in einem Hauptprogramm ein Objekt von Circle an und lassen Sie die Fläche berechnen.
c) Fügen Sie einen weiteren komplexen Datentyp namens Rechteck im package *prozedural* ein.
   Erweitern Sie Ihre Klassen im package um die Berechnung der Fläche des Rechtecks.
   Erweitern Sie Ihre Klassen im package, um die Berechnung des Umfangs für Rechteck, Kreis und Quadrat.
d) Ergänzen Sie im package oo eine Klasse Rectangle.
   Erweitern Sie Ihre Klassen im package um die Berechnung der Fläche des Rectangle.
   Erweitern Sie Ihre Klassen im package, um die Berechnung des Umfangs für Square, Circle und Rectangle.
e) Vergleichen Sie beide Ansätze!

Aufgabe 3: (Vorbedingungen und Nachbedingungen)

Was sind sinnvolle Vor- und Nachbedingungen für die folgenden Methoden/Klasse?

a)

```
public static int potenz(int basis, int exponent)
{
  int ergebnis = 1; //neutrales Element der Multiplikation
  for (int i = 1; i <= exponent; i++)
  {
    ergebnis = ergebnis * basis;
  }
  return ergebnis;
}
```

b)

```
public static void bubbleSort(int... numArray)
{
  int n = numArray.length;
  int temp;

  for (int i = 0; i < n; i++)
  {
    for (int j = 1; j < n-i; j++)
    {
      if (numArray[j-1] > numArray[j])
      {
        temp = numArray[j-1];
        numArray[j-1] = numArray[j];
        numArray[j] = temp;
      }
    }
  }
}
```

Hinweis: int… ist ein varargs-Argument.

c)

```
public class Rechteck
{
  double laenge;
  double breite;

  public Rechteck(double laenge, double breite)
  {
    this.laenge = laenge;
    this.breite = breite;
  }

  public double berechneFlaeche()
  {
    return laenge * breite;
  }
}
```

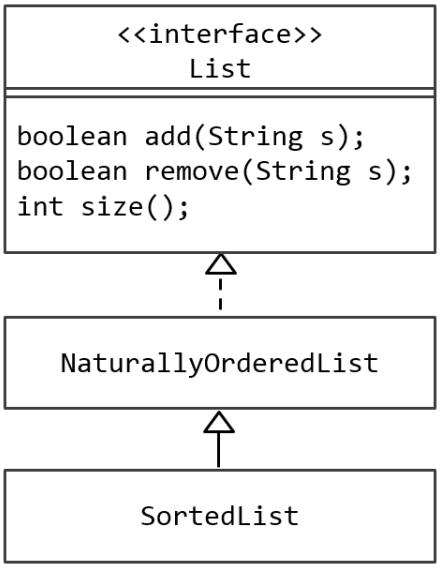
Aufgabe 4: (Liskovsches Substitutionsprinzip)
Gegeben sei ein Interface List mit den Methoden

- add, die den übergebenen String in die Liste einfügt.
- remove, die den übergebenen String aus der Liste entfernt.
- size, die die Länge der Liste zurückgibt.

Weiterhin gebe es eine Implementierung des Interfaces namens NaturallyOrderedList, die das Interface wie folgt umsetzt:

- add: fügt einen String an das Ende der Liste ein
- remove: entfernt den ersten String, der dem übergebenen String entspricht, aus der Liste.
- size: gibt die Länge der Liste zurück

Sie wollen nun eine Liste (SortedList) implementieren, die die Strings nach dem Alphabet (bzw. Unicode) sortiert vorhält. Dazu erben Sie von NaturallyOrderedList und überschreiben die add-Methode in der folgenden Weise:
Sie rufen zunächst die add-Methode der Oberklasse auf. Danach sortieren Sie die gesamte Liste.



Verstößt diese Implementierung gegen das Liskovsche Substitutionsprinzip (LSP)? Wenn ja, warum und wie kann man diesen Verstoß umgehen? Wenn nein, begründen Sie, warum das LSP eingehalten wird!