Prof. Dr. Thomas Breuel
Dipl.-Inf. Patrick Michel (AG ST)
Dipl.-Ing. Ludwig Schmidt-Hackenberg

TU Kaiserslautern
Fachbereich Informatik
AG Bildverstehen und Mustererkennung

Übungsblatt 12: Software-Entwicklung 1 (WS 2010/11)

Ausgabe: in der Woche vom 24.01. bis zum 28.01.11
Abgabe: in der Woche vom 31.01. bis zum 04.02.11
Abnahme: max. zwei Tage nach der Übung

Aufgabe 1 Objektorientierte Modellierung (Präsenzaufgabe)

a) Entwerfen Sie ein Klassendiagramm für einfach verkettete Listen vergleichbar zu `IntList` aus der Vorlesung. Tragen Sie in dieses Diagramm auch Multiplizitäten und Namen der Beziehungen ein.

b) Entwerfen Sie ein Klassendiagramm für das Geflecht eines fast vollständigen binären Suchbaums mit Objekten des Typ `T` an den Elementen des Baums. Tragen Sie in dieses Diagramm auch Multiplizitäten und Namen der Beziehungen ein.

c) Entwerfen Sie ein Klassendiagramm mit Multiplizitäten und Beziehungen für folgendes Beispiel: `Personen` in einem `Unternehmen` können angestellt sein, es leiten, einer `Abteilung` vorstehen, einer Abteilung zugeordnet sein oder Kunde des Unternehmens sein. Ein Unternehmen kann auch mit einem Unternehmen kooperieren.

Aufgabe 2 Objektorientierte Modellierung (Einreichaufgabe)

Ihnen liegen die folgenden Informationen über Transportmittel vor:

Transportmittel können sich bewegen und haben die Eigenschaften Höhe, Breite, Farbe, Höchstgeschwindigkeit und Durchschnittsgeschwindigkeit. Ein Landtransportmittel kann fahren, bremsen und ziehen und besitzt zusätzlich Angaben über die Zahl der Räder. Wassertransportmittel zeichnen sich zusätzlich durch die Angabe eines Bruttogewichts aus und können anlegen und ablegen. Lufttransportmittel können zusätzlich abheben und landen. Autos sind Landtransportmittel, die man betanken kann, mit zusätzlichen Angaben über die Leistung (in kW), Benzinverbrauch, Hersteller und Modell. Ein Zug ist ebenfalls ein Landtransportmittel, das abfahren und anhalten kann, mit einer Angabe über die Lokomotive und die Anzahl der Waggons. Ein PKW ist ein Auto und hat zusätzlich noch eine Angabe über die Anzahl der Türen und das Zulassungsjahr. Busse sind Autos mit einer zusätzlichen Angabe über die Anzahl der Sitzplätze. Gäste können zu ihnen zusteigen oder aussteigen. LKWs sind Autos, die beladen und entladen werden können, mit einer zusätzlichen Angabe über die Ladefläche.

a) Erstellen Sie mit Hilfe dieser Informationen ein Klassendiagramm, das die verschiedenen Transportmittel einschließlich ihrer Attribute und Methoden strukturiert darstellt.

b) Spezialisieren Sie die Klassen Wasser- und Lufttransportmittel, indem Sie mindestens fünf weitere Klassen, wie z.B. Motorboot und Flugzeug in das Klassendiagramm aus Aufgabenteil a) einfügen! Erweitern Sie die neuen Klassen um geeignete Attribute und Methoden.

c) Entwerfen Sie ein Klassendiagramm mit Multiplizitäten und Beziehungen für das Beispiel Mini-Facebook aus der Vorlesung.

Aufgabe 3 Objekte, Kapselung, Fehlerbehandlung (Einreichaufgabe)

Wir betrachten eine Klasse `Secret`, die einen geheimen Wert enthält, auf den von außerhalb nicht zugegriffen werden darf:

```
class Secret {
  private int secret;
  private int accessCounter = 0;

  int accessCount() { return accessCounter; }
  int unlock(int[] guess) throws WrongCombinationException {
    accessCounter++;
    ...
  }

  Secret()      { ... }
  Secret(int n) { ... }
}
```

Die Methode `unlock` bietet Zugriff auf den geheimen Wert, insofern man die richtige Zahlenkombination als Array übergibt. Ist die Kombination hingegen falsch, wird eine `WrongCombinationException` geworfen, die vom Aufrufer behandelt werden muss. Werden zu viele oder zu wenig Stellen übergeben, werden ebenfalls passende Ausnahmen ausgelöst. Schließlich wird auch eine Ausnahme ausgelöst, falls man an einer Stelle einen Wert der außerhalb des gültigen Bereichs liegt übergibt.

```
class TooManyDigitsException extends RuntimeException {}
class TooFewDigitsException  extends RuntimeException {}
class InvalidDigitException  extends RuntimeException {}

class WrongCombinationException extends Exception {
  private int p, d;
  WrongCombinationException(int p, int d) { this.p = p; this.d = d; }

  int correctPlace() { return p; }
  int correctDigit() { return d; }
}
```

Erzeugt wird ein zufälliges Geheimnis mit dem *default*-Konstruktor `Secret()`, ein festes Geheimnis mit dem Konstruktor `Secret(int n)`. Eine Kombination zum Schutz des Geheimnis wird automatisch aus dem Geheimnis generiert und niemandem mitgeteilt. Alles was man darüber weiß ist, dass es mindestens vier Stellen lang ist und jede Stelle Eingaben ab `0` akzeptiert. Außerdem ist der maximale Wert einer Stelle größer oder gleich der Anzahl der Stellen.

Leider handelt es sich hierbei wohl um eins der schlechtesten Zahlenschlösser aller Zeiten, da jeder vergebliche Versuch es zu öffnen eine Menge an Informationen über die richtige Kombination preisgibt.

a) Laden Sie sich das Archiv "`Secret.zip`" von der Vorlesungsseite und entwicklen Sie eine Klasse `UncleKracker` mit folgenden Eigenschaften: Die Klasse bekommt im Konstruktor ein Secret übergeben, mit dem sie arbeiten soll. Zusätzlich soll sie eine Methode `int[] crack()` besitzen, die das Schloss knackt und die richtige Kombination zurück gibt. Achten Sie darauf, dass niemand auf die Internas Ihrer Klasse zrugreifen kann.

   Zum Knacken soll die Methode `crack` die öffentliche Schnittstelle von `Secret` verwenden, also im wesentlichen die Methode `unlock`. Fangen Sie alle möglichen Ausnahmen entsprechend ab und reagieren Sie sinnvoll darauf, so dass Sie nach einer endlichen Anzahl an Iterationen das Schloss geknackt haben.

b) Testen Sie Ihre Implementierung mit verschiedenen Schlössern. Da die Schlösser sehr groß werden können, sollten Sie Ihre Tests mit kleinen, einfachen Schlössern durchführen. Dazu können Sie die folgenden Seeds benutzen, die alle ein Schloss mit vier Stellen und sechs Möglichkeiten pro Stelle lieferen:[1]

[1] Sollten diese Seeds auf Ihrem System nicht diesen Effekt haben, müssen Sie selbst ermittlen hinter welchen Seeds sich kleine Schlösser verbergen.

```
7422    111972  209338  265094  306944  499357  551273  779423  921639  978011
20025   146574  213305  266831  368652  509463  580264  783807  934502
82725   194839  222197  277350  381932  524567  665598  812081  938630
103080  204798  248655  294337  403542  538126  692080  825105  965144
```

c) *(freiwillige Zusatzaufgabe)* Die Klasse `Secret` protokolliert die Anzahl der Zugriffe auf die Methode `unlock`. Optimieren Sie Ihren Algorithmus zum Knacken von Schlössern, um im Mittel über viele zufällige Beispiele eine möglichst niedrige Zahl zu erreichen!

Aufgabe 4 Grafikeditor (Präsenzaufgabe)

Lesen Sie die Teilaufgaben von Aufgabe 5 und entwerfen Sie ein Klassendiagramm mit allen Klassen, Methoden, Attributen und Beziehungen.

Aufgabe 5 Grafikeditor (Einreichaufgabe)

In dieser Aufgabe geht es darum, einen Editor für graphisch darstellbare Komponenten zu implementieren. Der Editor hat ein Zeichenfeld, das über x und y-Koordinaten angesprochen wird. Die Bedienung des Editors erfolgt Menü gesteuert in der Kommandozeile. Eine mögliche Sitzung könnte wie folgt aussehen:

```
 1) Neues Polygon anlegen
 2) Fokus ändern
 3) Verschieben
Ihre Wahl: 1
Polygon eingeben:
Name: Dreieck
Anzahl der Ecken (min 3): 3
Ecke 1:
x-Koordinate: 10
y-Koordinate: 10
Ecke 2:
x-Koordinate: 10
y-Koordinate: 20
Ecke 3:
x-Koordinate: 60
y-Koordinate: 30
 1) Neues Polygon anlegen
 2) Fokus ändern
 3) Verschieben
Ihre Wahl:
```

Zur Bearbeitung dieser Aufgabe können Sie sich die Datei "`SEGraphics.java`" von der Vorlesungsseite laden und übersetzen. `SEGraphics` bietet Ihnen folgendes:

- Der Konstruktor von `SEGraphics` legt eine Zeichenfläche an.
- Die Methode `clear` löscht den Inhalt einer Zeichenfläche.
- `drawCircle`, `drawString` und `drawLine` werden zum Zeichnen auf der Zeichenfläche benutzt.

Folgende Teilaspekte sollen bearbeitet werden:

a) Entwerfen Sie eine Klasse `Point`. Der Konstruktor soll die Koordinaten des Punktes als Parameter nehmen. Schreiben Sie eine statische Methode `construct`, die vom Benutzer die Koordinaten des Punkts erfragt, ein Objekt vom Typ `Point` erzeugt und zurückgibt. Sie werden auch Methoden brauchen, die einen Punkt verschieben.

b) Ein Polygon besteht aus mindestens drei verschiedenen Punkten einer Ebene, die durch Linien miteinander verbunden sind. Entwerfen Sie eine Klasse `Polygon`, mit der Sie beliebige Polygone verwalten können. Um Polygone später identifizieren zu können, soll jedes Polygon einen Namen haben.

   Schreiben Sie auch hier eine zusätzliche statische Methode mit dem Namen `construct`, die vom Benutzer den Namen und die Koordinaten der Eckpunkte erfragt und ein neues Polygon zurückgibt. Außerdem brauchen Sie Methoden und Attribute, um die Position und Farbe des Polygons zu ändern, und um das Polygon auf ein `SEGraphics`-Objekt zu zeichnen.

   Verwenden Sie für Farben Objekte der Klasse `Color` der Java-API. Informationen dazu finden Sie unter http://download.oracle.com/javase/6/docs/api/java/awt/Color.html.

c) Entwerfen Sie eine Klasse `Editor`, die eine Menge von Polygonen und eine Zeichenfläche verwaltet. Der Editor soll eine Methode haben, die es dem Benutzer erlaubt, interaktiv Polygone zu erstellen, zu verschieben und zu löschen.

   Als Besonderheit soll der Editor über einen Fokus verfügen, so dass man ein Polygon auswählt, d.h. ihm den Fokus gibt, und anschließend z.B. mehrere Verschiebungen nacheinander auf diesem Polygon ausführen kann.

d) Schreiben Sie ein Hauptprogramm, das einen neuen Editor erzeugt und die Interaktionsmethode aufruft.

Aufgabe 6 Basiswissen zur Vorlesung

Kreuzen Sie an, ob folgende Aussagen wahr oder falsch sind. Bereiten Sie diese Aufgabe bis zu Ihrer nächsten Übungsstunde vor, so dass Sie bei Unklarheiten nachfragen und die Antworten diskutieren können.

wahr	falsch	
☐	☐	Eine Ordnung ist genau dann noethersch, wenn jede aufsteigende Kette stationär wird.
☐	☐	$(\mathbb{N}, <)$ ist keine noethersche Ordnung.
☐	☐	$(\mathbb{Z}, \leq)$ ist keine noethersche Ordnung.
☐	☐	$(2\mathbb{N}, \leq)$ ist keine noethersche Ordnung. $(2\mathbb{N} = \{2n \mid n \in \mathbb{N}\})$
☐	☐	Einer Programmvariablen können mehrere Speichervariablen zugeordnet sein.
☐	☐	Ein vollständiger gerichteter Graph hat genau doppelt so viele Kanten wie ein vollständiger ungerichteter Graph.
☐	☐	Eine Bedingung, die im Vorzustand einer Prozedur gelten muss, kann nicht unbedingt über den Rumpf der Prozedur als gültig angenommen werden.
☐	☐	Aus der Vorbedingung **`true`** lässt sich die Nachbedingung **`false`** für keine Prozedur beweisen.
☐	☐	Ein Hoare-Tupel verbindet eine Vorbedingung P mit einer Nachbedingung Q.
☐	☐	Kann die Spezifikation einer Prozedur mit Schleife bewiesen werden, gibt es automatisch unendlich viele Schleifeninvarianten, mit denen der Beweis geführt werden kann.