Objektorientiertes Design

- Zerlegung einer Problemstellung in Klassen
- Packages
- Schnittstellen
- Vor- und Nachbedingungen, Klasseninvarianten

Fallstudie: Random Walk

Wir simulieren die Irrfahrt des Odysseus auf den Weltmeeren.

Man zeichne ein Gitter bestehend aus rechteckigen Feldern.

Odysseus befindet sich zu Beginn im mittleren Feld.

Eine bestimmte Zahl von Malen lasse man Odysseus zufällig eine Richtung (N, S, O, W) wählen, bewege ihn in diese Richtung und zeichne ihn im neuen Feld.

Zusatzbedingungen

Zunächst nehmen wir quadratische Felder. Später soll es möglich sein, auch rechteckige Felder, die nach außen hin immer kleiner werden, zu nehmen. Das Design soll dem bereits Rechnung tragen.

Es sollen nur wenige “uses” Beziehungen zwischen den Klassen vorhanden sein. Die vorhandenen sollen möglichst hierarchisch sein (keine Schleifen).

[Mögliche Lösung, siehe WWW-Seite]

Faustregeln

- Jedes Substantiv der Problembeschreibung ist ein Kandidat für eine Klasse.
- Enge Kopplung durch “uses”-Beziehungen ist zu vermeiden.
- Eine Klasse sollte diejenigen Methoden bereitstellen, die zu ihr passen, nicht solche, die sich aus ihrem Verwendungskontext ergeben. Double enthält keine Methode zur Anzeige im Grafikfenster.

Schnittstellen

Eine Schnittstelle (*interface*) besteht aus Methodensignaturen, d.h. Methodendefinitionen ohne den `{...}` Block.

Syntax:

```
public interface NamederSchnittstelle {
   MethodenSignatur1;
   MethodenSignatur2;
   ...
}
```

Mit einer Schnittstelle lassen sich Objekte, die ähnliche, gleichnamige Methoden implementieren, zusammenfassen.

`NamederSchnittstelle` ist jetzt ein Typ, der überall dort verwendet werden kann, wo Typen vorkommen (Deklaration von Variablen, Rückgabewerten).

Sprites

Beispiel:

```
public interface Sprite {
    /** Zeichnen des Sprite in ein gegebenes
        Rechteck */
    public void zeichnen(Graphics2D g, Rectangle r);
}
```

Die Schnittstelle `Sprite` fasst Objekte zusammen, die “sich” in ein gegebenes Rechteck zeichnen können.

Engl.: *sprite* = kleiner Waldgeist.

Bezeichnet kleine Figuren, die im Rahmen eines Computerspiels oder einer Animation sich auf dem Bildschirm umherbewegen.

Anderes Beispiel:

```
public interface Measurable{
   double getMeasure();
}
```

Die Schnittstelle `Measurable` fasst Objekte zusammen, die ein Double-wertiges “Maß” haben.

Implementierung

Durch die Klausel `implements NamederSchnittstelle` in einer Klassendefinition wird angezeigt, dass Objekte dieser Klasse die Schnittstelle implementieren.

In der Klasse müssen dann alle Methoden der Schnittstelle implementiert werden.

Objekte der Klasse haben dann automatisch auch den Typ `NamederSchnittstelle`.

Implementiert eine Klasse die Methoden einer Schnittstelle, fehlt aber die `implements`-Klausel, so haben Objekte der Klasse *nicht* den Typ der Schnittstelle.

In UML zeichnet man von der Klasse zur Schnittstelle einen Pfeil mit kreisrunder, hohler Spitze `-----o`.

Beispiel

```
public class Blob implements Sprite {
   private double temperatur; // Beliebige Instanzvariable
   void zeichnen(Graphics2D g, Rectangle box) {
         g.setColor(Color.red);
         g.fill(Ellipse2D.Double(box.x, box.y,
               box.width, box.height);
   }
   void erhoehen(); // beliebige andere Methode
}
```

Packages

Man kann mehrere Klassen in eine *package* zusammenfassen.

- Diese müssen dann in einem Unterverzeichnis liegen, dessen Name der Packagename ist.
- Jede Datei der Package muss mit `package <name>;` beginnen, wobei `<name>` der Packagename ist.
- Auf Klassen der Package greift man durch Vorschalten von `<name>.` zu.
- Alternativ kann man `import <name>.*;` verwenden.
- Ist eine Klasse oder Methode oder Instanzvariable weder `public` noch `private`, so ist sie nur innerhalb ihrer Package sichtbar.
- Packages vom übergeordneten Verzeichnis kompilieren und ausführen.