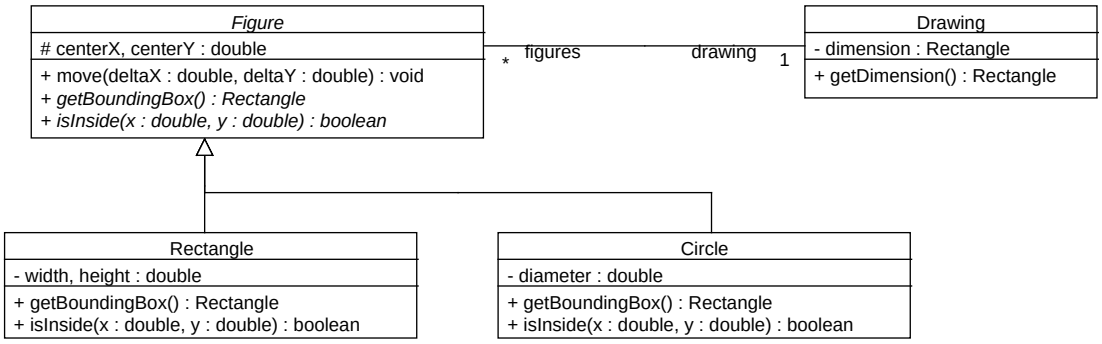
Ludwig-Maximilians-Universität München SS 2006
Institut für Informatik Übungsblatt 10
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

Informatik II

Zu jeder Aufgabe sind Dateien abzugeben, deren Namen rechts in der Aufgabenüberschrift stehen. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 10-1 OCL (12 Punkte, `ocl.txt`)

Das folgende UML-Diagramm gibt die Modellierung eines stark vereinfachten Grafikprogramms wieder, daß die aus der Klausur bekannten grafischen Elemente benutzt. Diese werden in einer Zeichnung verwendet. Eine Zeichnung hat eine gewisse Höhe und Breite, und in diesem Beispiel sollen grafische Elemente stets ganz innerhalb der Zeichnung liegen. Um dies überprüfen zu können, wird eine Methode `getBoundingBox` verwendet.



Spezifizieren Sie durch eine geeignete textuelle OCL-Spezifikation folgende Anforderungen an eine korrekte Implementierung:

1. `isInside` kann jederzeit aufgerufen werden, und soll genau dann `true` zurückliefern, wenn der Punkt $(\mathtt{x}, \mathtt{y})$ echt innerhalb der Figur liegt.
2. `move` soll eine Figur um die angegebene Distanz verschieben.
3. `getBoundingBox` soll ein Rechteck zurückliefern, daß alle Punkte der Figur enthält (wo liegt hier das Problem? Können Sie es beheben?)
4. Zu jedem Zeitpunkt soll eine Figur innerhalb der Fläche der Zeichnung liegen, zu der sie gehört.

Hinweise:

- In OCL-Ausdrücken ist es erlaubt, Methoden aufzurufen, sofern diese den Objekt-/globalen Zustand nicht modifizieren (man spricht dann von „queries“) und den Rückgabewert zu verwenden. In dem gegebenen Beispiel sind dies alle Methoden ausser `move`.
- Für einen OCL-Ausdruck t vom Typ `integer` oder `real` liefert t`.abs()` den Betrag von t. Eine Berechnung der Quadratwurzel brauchen Sie eigentlich nicht, aber Sie können notfalls auf `Math.sqrt` ausweichen, welche auch als Query aufgefasst werden kann.

Aufgabe 10-2 **Test-First-Entwicklung** (12 Punkte, `Unit.java, Length.java`)

Gegeben seien die folgenden Unit-Tests:

```
package testfirst;

import junit.framework.TestCase;

public class LengthTest extends TestCase {
    public void testConversion() {
        assertEquals(new Length(1.0, Unit.M), new Length(100.0, Unit.CM).convertTo(Unit.M));
        assertEquals(new Length(1.0, Unit.FT), new Length(12.0, Unit.IN).convertTo(Unit.FT));
        assertEquals(new Length(2.54, Unit.CM), new Length(1.0, Unit.IN).convertTo(Unit.CM));
    }
    public void testEquals() {
        assertEquals(new Length(1.0, Unit.IN), new Length(2.54, Unit.CM).convertTo(Unit.IN));
        // Längen sollen nur dann gleich sein, wenn sie auch die selbe Masseinheit haben:
        assertFalse(new Length(1.0, Unit.IN).equals(new Length(2.54, Unit.CM)));
    }

    public void testToString() {
        assertEquals("3.0cm", new Length(3.0, Unit.CM).toString());
        assertEquals("3.0m", new Length(3.0, Unit.M).toString());
        assertEquals("3.0in", new Length(3.0, Unit.IN).toString());
        assertEquals("3.0ft", new Length(3.0, Unit.FT).toString());
    }
}
```

Implementieren Sie die Enum `Unit` und die Klasse `Length`, so dass sie diese Tests erfüllen. Hinweise zur Implementierung von Standard-Methoden von `Object` (siehe auch und insbesondere deren JavaDoc):

- Wer `equals()` implementiert, muss auch `hashCode` implementieren!
- `hashCode`: Kann man über die Hashcodes der Attribute implementieren, wenn man jede Komponente mit einer Primzahl multipliziert (so dass sie im Ergebnis nicht zu sehr „vermischt“ werden):

  ```
  return (this.attrib1.hashCode()
          + (this.attrib2.hashCode() * 3)
          + (this.attrib3 * 5));
  ```

- Man kann in einer Enum u.a. folgendes selbst definieren: (private) Konstruktoren, Attribute und Methoden.

Sonstige Bemerkungen:

- Der erste Schritt, ist, dass die zu implementierenden Klassen (bzw. deren Skelett) übersetzt werden können. Wird eine Methode als noch nicht vorhanden in Eclipse angezeigt, wird zur Fehlerbehebung angeboten, eine leere Implementierung dieser Methode zu generieren. Danach sollte der Compiler keine Fehler mehr anzeigen. Als nächstes lassen Sie den Test zum ersten Mal laufen und überzeugen sich, dass er auch wirklich scheitert (ansonsten wären Sie fertig).
- Arbeiten Sie nun die Tests der Reihe nach ab (indem sie entweder scheiternde Tests ignorieren oder auskommentieren).
- Google hilft einem beim Umrechnen von Einheiten. So kann man beispielsweise nach „1in in mm“ suchen und bekommt die Antwort, dass `1 in = 25.4 millimeters`.

Aufgabe 10-3 Objekt-orientierter Entwurf (Keine Punkte, `schach-klassen.txt`, `schach-uml.pdf,ps,jpg`)

Diese Aufgabe wird zwar korrigiert, aber nicht bewertet.

Entwerfen Sie ein Schachspiel. Im folgenden die bewusst ungenau gehaltenen Anforderungen eines Anwenders:

> Es gibt zwei Spieler, die abwechselnd mit jeweils einer von ihren 16 Figuren ziehen. Jede Figur weiss, wie sie ziehen kann.

Hinweise:

- Sie müssen nicht alle Figuren modellieren sondern nur den Springer (`http://de.wikipedia.org/wiki/Springer_(Schach)`).
- Allzu genaue Details sind nicht gefragt, mehr ein Überblick über den Problembereich.

Beim Entwurf sollen Sie wie folgt vorgehen:

a) Klassen und deren Aufgaben: Führen Sie alle Klassen auf und schreiben Sie zu jeder Klasse, welche Aufgaben Sie hat. Diese Verantwortlichkeiten sind eine traditionelle Idee im objekt-orientierten Entwurf und werden im Englischen auch als *responsibilities* bezeichnet.

b) Erstellen Sie auf dieser Grundlage ein Klassendiagramm in UML.

Abgabe: Per UniWorx, bis spätestens Montag, den 17.7.2006 um 9:00 Uhr.