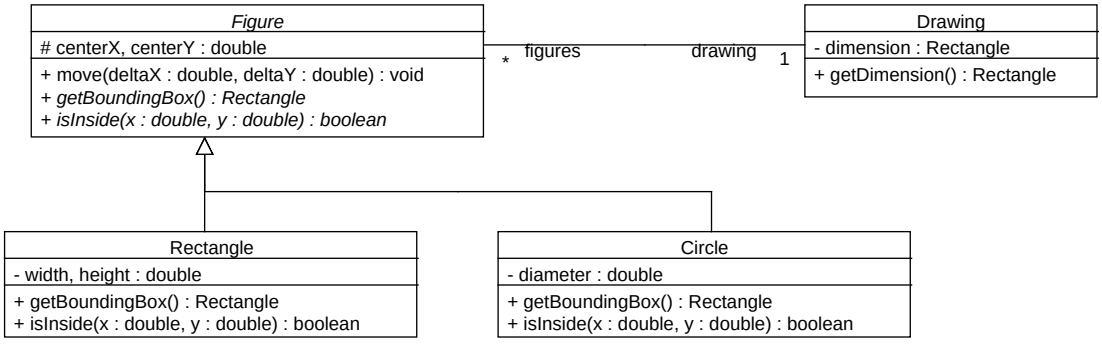
**Ludwig-Maximilians-Universität München** SS 2006
**Institut für Informatik** Übungsblatt 10
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

# Informatik II Musterlösung

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.

   **Lösung:**

   ```
   context Rectangle::isInside(real x, real y) : boolean
   pre:  true
   post: result = (centerX - x).abs() < width/2 and (centerY - y).abs() <
         height/2

   context Circle::isInside(real x, real y) : boolean
   pre:  true
   post: result = (centerX - x) * (centerX - x) + (centerY - y) * (centerY-y) <
         (diameter * diameter)/2
   ```

   Der Satz des Pythagoras heisst ja $a^2 + b^2 = c^2$, insofern kann auf eine Quadratwurzel in diesem Fall verzichtet werden. Man muss natürlich schauen, ob es jetzt Probleme mit vorher negativen Zahlen geben kann, aber der Durchmesser sollte sowieso positiv sein. Das kann man durch eine Invariante erzwingen:

   ```
   context Circle
   inv:  diameter >= 0
   ```

2. `move` soll eine Figur um die angegebene Distanz verschieben.

   **Lösung:**

**context** `Figure::move(real deltaX, real deltaY) : void`
**pre:** `true`
**post:** `centerX = centerX@pre + deltaX and centerY = centerY@pre + deltaY`

3. `getBoundingBox` soll ein Rechteck zurückliefern, daß alle Punkte der Figur enthält (wo liegt hier das Problem? Können Sie es beheben?)

   **Lösung:** Das Problem ist, daß folgende Lösung korrekt ist:

   **context** `Figure::getBoundingBox() : Rectangle`
   **pre:** `true`
   **post:** `result = drawing.getDimension()`

   (Denn dass dies eine Bounding Box ist, folgt aus der (später gegebenen) Invariante. Gemeint ist natürlich das kleinste Rechteck, daß die Figur umschliesst. Dies ist möglich mit

   **context** `Rectangle::getBoundingBox() : Rectangle`
   **pre:** `true`
   **post:** `result.centerX = centerX and result.centerY = centerY and result.width = width and result.height = height`

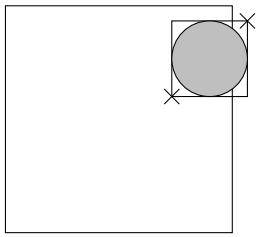
   **context** `Circle::getBoundingBox() : Rectangle`
   **pre:** `true`
   **post:** `result.centerX = centerX and result.centerY = centerY and result.width = diameter and result.height = diameter`

4. Zu jedem Zeitpunkt soll eine Figur innerhalb der Fläche der Zeichnung liegen, zu der sie gehört.

   **Lösung:**

   **context** `Figure`
   **inv:** `drawing.getDimension().isInside( getBoundingBox().centerX - getBoundingBox().width/2, getBoundingBox().centerY - getBoundingBox().height/2) and drawing.getDimension().isInside( getBoundingBox().centerX + getBoundingBox().width/2, getBoundingBox().centerY + getBoundingBox().height/2)`

   Dies genügt, wie man an folgender Grafik sieht:



**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`.

**Lösung:**

```
package testfirst;

public enum Unit {
```

```
    CM("cm", 1.0), M("m", 100.0), IN("in", 2.54), FT("ft", 30.48);

    private String printName;
    private double timesCm;

    private Unit(String printName, double timesCM) {
        this.printName = printName;
        this.timesCm = timesCM;
    }

    public double getTimesCm() {
        return this.timesCm;
    }

    public String getPrintName() {
        return this.printName;
    }

}
```

```
package testfirst;

public class Length {

    private double amountInCm;
    private Unit unit;

    public Length(double amount, Unit unit) {
        this.unit = unit;
        this.amountInCm = convert(amount, unit, Unit.CM);
    }

    private static double convert(double fromAmount, Unit fromUnit, Unit toUnit) {
        double cms = fromAmount * fromUnit.getTimesCm();
        return cms / toUnit.getTimesCm();
    }

    public Length(Length other) {
        this.amountInCm = other.amountInCm;
        this.unit = other.unit;
    }

    private void setUnit(Unit newUnit) {
        this.unit = newUnit;
    }

    public Length convertTo(Unit newUnit) {
        Length newLength = new Length(this);
        newLength.setUnit(newUnit);
        return newLength;
    }

    //------------------------ Standard Object methods

    @Override
    public int hashCode() {
        return (new Double(amountInCm).hashCode() * 3) + unit.hashCode();
```

```
    }
    @Override
    public boolean equals(Object other) {
        if (! (other instanceof Length)) return false;
        Length otherLength = (Length) other;
        return this.amountInCm == otherLength.amountInCm && this.unit == otherLength.unit;
    }
    @Override
    public String toString() {
        return Double.toString(convert(this.amountInCm, Unit.CM, this.unit))+unit.getPrintName();
    }
}
```

**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:

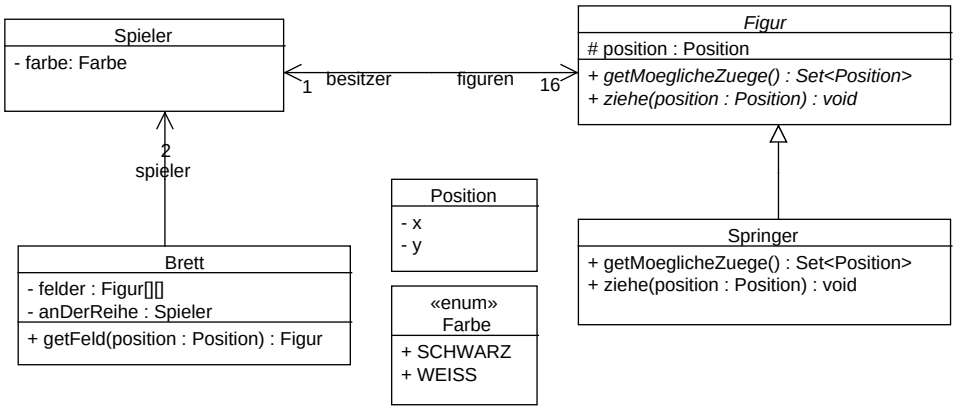
- Sie müssen nicht alle Figuren modellieren sondern nur den Springer (`http://de.wikipedia.org/wiki/Springer_(Schach)`).
- Allzu genaue Details sind nicht gefragt, mehr ein Überblick über den Problembereich.

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.

**Lösung:**




Responsibilities:

- Brett: Verwalte Brettinhalt und Spieler. Merke Dir, welcher Spieler gerade an der Reihe ist.

- Spieler: Hat eine Farbe und verwaltet seine Figuren.
- Figur (bzw. deren konkrete Unterklassen): Weiss die Züge, die sie machen kann, steht an einer bestimmten Position auf dem Feld, die man ändern kann.

Anmerkungen:

- Das ist ein gutes Beispiel für die Unsicherheit, die man immer beim objektorientierten Modellieren hat: Gehören die Figuren dem Spieler oder dem Brett oder gar niemandem? Zieht die Figur selbst oder eher der Spieler oder sollte das Ziehen nicht vom Brett übernommen werden? Hat der Spieler eine Farbe oder *ist* der Spieler die Farbe? Diese Fragen klären sich überlicherweise, je mehr man sich auf die Implementierung zubewegt.
- Assoziationen: Bei den Assoziationen wurde hier besonders hervorgehoben, wer was sehen kann (und weniger die Lebensdauer wie bei schwacher bzw. starker Aggegation).
- Getter und Setter: Wurden bewusst nicht ins Klassendiagramm geschrieben, da sie konzeptuell mit den Attributen zusammenfallen.

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