Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 1
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

1.1 Aufgabe*, Bibliothek

Betrachten Sie folgende Problemstellung:

> Entwickeln Sie ein Programm, dass Bibliothekare unterstützt. Es soll für jedes Buch Buchtitel, Preis (in €), Namen des Autors, Sprache und Erscheinungsjahr speichern.

Erstellen Sie ein Klassendiagramm für diese Anwendung auf Papier und schreiben Sie den Code für die Klasse in *ProfessorJ* auf. Schreiben Sie Code, um die folgenden Instanzen der Klasse zu erstellen:

1. *So Long and Thanks for all the Fish*; 8,45 €; Douglas Adams; Englisch; 1984
2. *Die Leber wächst mit ihren Aufgaben. Kurioses aus der Medizin*; 9,95 €; Eckart von Hirschhausen; Deutsch; 2008
3. *Für jede Lösung ein Problem*; 7,95 €; Kerstin Gier; Deutsch; 2007
4. *De bello Gallico / Der Gallische Krieg*; 12,80 €; Caesar (Autor), Marieluise Deissmann (Hg. / Übersetzung); Lateinisch / Deutsch; 1980
5. *Le voyage d'Hector. Ou la recherche du bonheur.*; 11,95 €; Francois Lelord; französisch; 2004
6. *Star Trek.The Klingon Dictionary: English/Klingon, Klingon/English*; 12,99 €; Marc Okrand; Englisch / Klingonisch; 1992

1.2 Aufgabe*, Bild

Fügen Sie der folgenden unvollständigen Klassendefinition einen Konstruktor hinzu und erstellen Sie ein Klassendiagramm:

```
1 //represent computer images
2 class Image {
3   int height; // pixels
4   int width; // pixels
5   String source; // file name
6   String quality; // informal
7   ... // bitte ergänzen
8 }
```

Diese Definition wurde für die folgende Problemstellung entwickelt:

> Entwickeln Sie ein Programm, dass eine Sammlung von Bildbeschreibungen enthält. Die Beschreibung soll jeweils die Breite und Höhe des Bildes, die Quelldatei, und die Qualität des Bildes enthalten.

1.3 Aufgabe, Apfel

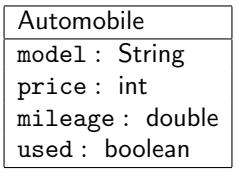
Erstellen Sie drei Instanzen der folgenden Klasse:

```
1  //introducing the concept of gravity
2  class Apple {
3    int x;
4    int y;
5    int RADIUS = 5;
6    int G = 10; // meters per second square
7
8    Apple(int x, int y) {
9      this.x = x;
10     this.y = y;
11   }
12 }
```

Wie viele Attribute besitzt jedes Objekt der Klasse Apple? Wie viele Argumente erhält der Konstruktor?

1.4 Aufgabe, Auto

Schreiben Sie für folgendes Klassendiagramm den Quellcode.



2.1 Aufgabe*, Wetter

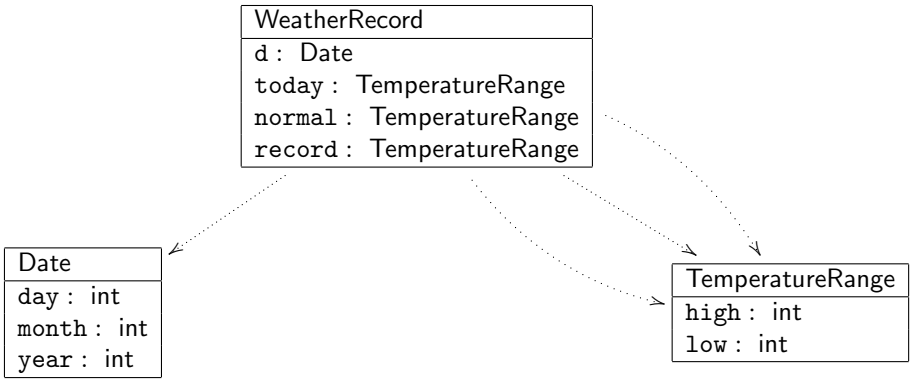


Abbildung 1: Wetterdaten

Betrachten Sie das Klassendiagramm aus Abbildung 1. Schreiben Sie die Klassendefinitionen und erzeugen Sie für jede Klasse jeweils zwei verschiedene Objekte.

2.2 Aufgabe*, Bibliothek - Version 2

Aufgabe 1.1 beschäftigt sich mit Buchdaten. Verändern Sie das Klassendiagramm so, dass es zusätzliche Informationen über den Autor enthalten kann (den Namen des Autors, das Geschlecht und den Geburtstag). Schreiben Sie für das veränderte Diagramm den Code in *ProfessorJ* auf und erzeugen Sie zwei Autorenobjekte und zwei Buchobjekte.

2.3 Aufgabe, Film

Sie sind Programmierer eines Webunternehmens und erhalten den Auftrag für folgende Problembeschreibung ein Klassendiagramm zu entwerfen:

> Unser Filmverleih will seine Webseite, die zur Zeit in einem katastrophalen Zustand ist, neu aufbauen. Uns ist wichtig, dass der Besucher der Webseite die Möglichkeit hat, unsere Filme (mit Titel, Jahr, 1. Schauspieler, 2. Schauspieler, 3. Schauspieler, Drehbuchautor, Regisseur) nach allen diesen Kriterien leicht zu durchsuchen. Wir speichern für jeden Schauspieler noch den Namen, Vornamen sowie das Geburtsdatum. Beim Drehbuchautor und Regisseur ist das Geburtsdatum nicht vorhanden.
>
> Für jeden Film soll festgehalten werden, ob er noch verfügbar ist, bzw. bis zu welchem Datum er ausgeliehen wurde. Falls ein Benutzer einen Film ausleihen will, kann er ihn über das Internet reservieren und innerhalb der nächsten 5 Stunden abholen. In diesem Zustand soll der Film nicht mehr für andere User zum Ausleihen verfügbar sein.

Erstellen Sie ein Klassendiagramm und schreiben Sie den Code in *ProfessorJ* auf.

3.1 Aufgabe, Zug

In der Vorlesung wurde ein Klassendiagramm für Zugfahrpläne gezeigt. Statt einen Zug mit einer String-Klassifizierung (`kind`) zu versehen, entfernen Sie dieses Attribut und erstellen Sie drei Klassen. Eine Klasse repräsentiert für S-Bahnen (SB), eine für Regionalbahnen (RB) und eine dritte ICEs. Der jeweilige Konstruktor soll bei diesen drei Klassen somit den kind-Parameter nicht erhalten.

Modifizieren Sie das Klassendiagramm entsprechend und schreiben Sie den Code für das Klassendiagramm auf. Erzeugen Sie jeweils zwei ICEs, RBs und SBs.

Damit alle Klassen als Züge behandelt werden können, erstellen Sie ein passendes Interface.

3.2 Aufgabe*, Bild - Version 2

Die image-Klasse aus Aufgabe 1.2 beschreibt Bilder anhand diverser Attribute. Erweitern Sie den dort gegebenen Ansatz, damit er folgender Aufgabenstellung zu gerecht wird:

> Ein Museum, das bis jetzt nur seine Bilddaten in einer Datenbank verwaltet hat, geht nun den Schritt zur kompletten Inventarverwaltung mit dem Computer. Aus diesem Grund fallen nun eine Menge von unterschiedlichen Daten an, die verwaltet werden müssen.
>
> 1. Bilder in Form von `*.gif` Dateien
> 2. Musik in Form von `*.mp3` Dateien
> 3. Text in Form von `*.txt` und `*.odt`[1]
> 4. Tabellen in Form von `*.ods`[2]
> 5. Archive in Form von `*.tar.gz`[3] oder `*.zip`
>
> Jede der Dateien hat einen Namen, ein Datum, an dem die Datei erzeugt wurde, ein Datum, zu der die Datei das letzte Mal verändert wurde, und einen Besitzer.

Entwerfen Sie ein Klassendiagramm, das die unterschiedlichen Dateitypen zusammen mit ihren Attributen verwaltet. Schreiben Sie den entsprechenden Code und erstellen Sie von jeder Klasse mindestens zwei Objekte.

3.3 Aufgabe, Telefonbuch

Ihr Handy besitzt ein Telefonbuch. Schauen Sie, welche Daten pro Eintrag im Telefonbuch möglich sind, und erstellen Sie eine entsprechende Java-Klasse und das passende Klassendiagramm. (Falls Sie kein Handy zur Hand haben, öffnen Sie die Anwendung `thunderbird` und betrachten Sie die Struktur des Adressbuchs.)

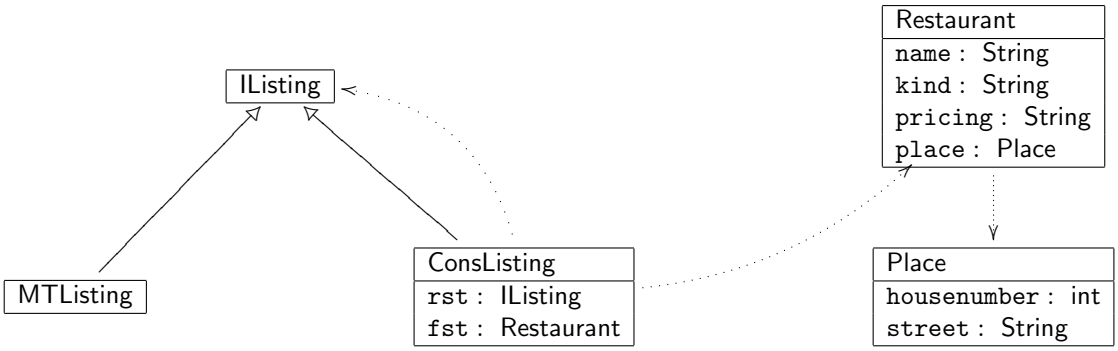
Erzeugen Sie 5 Einträge des Telefonbuchs.

Schreiben Sie das Ergebnis Ihrer Einträge so auf, wie *ProfessorJ* die Objekte darstellt.

[1] OpenOffice Text
[2] OpenOffice Sheet
[3] Archivformat von `tar`

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 4
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

4.1 Aufgabe, Restaurant



Schreiben Sie den Code für dieses Klassendiagramm und erstellen Sie eine Liste von Restaurants mit mindestens vier Einträgen aus Freiburg und Umgebung.

4.2 Aufgabe, Typen

```
1  interface SalesItem {}
2
3  class DeepDiscount implements SalesItem {
4    int originalPrice;
5    ...
6  }
7
8  class RegularDiscount implements SalesItem {
9    int originalPrice;
10   int discountPercentage;
11   ...
12 }
```

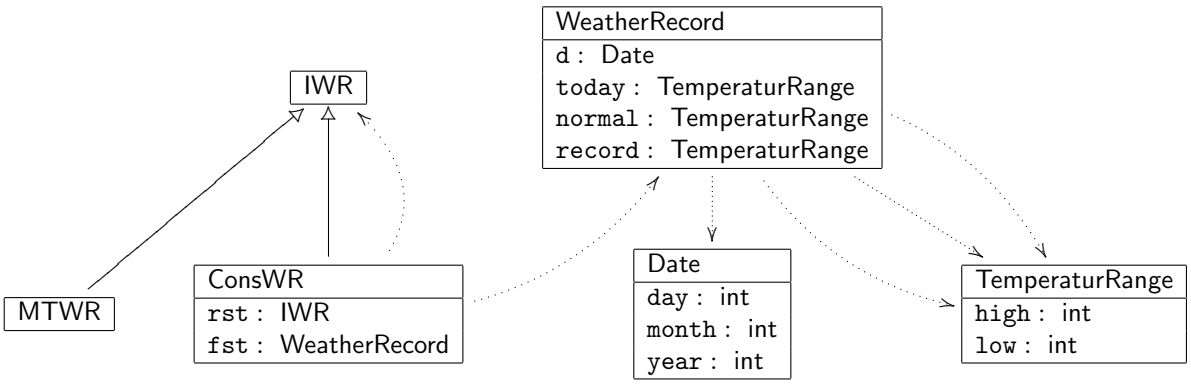
Stellen Sie in diesem Zusammenhang die Typen der Variablen `s`, `t` und `v` fest, falls sie wie in den drei folgenden Zeilen definiert werden.

```
1 SalesItem s = new DeepDiscount(9900);
2 SalesItem t = new RegularDiscount(9900,10);
3 RegularDiscount u = new RegularDiscount(9900,10);
```

Falls jemand die folgenden Zeilen aufgeschrieben hat, welche von Ihnen enthalten Typfehler und wo sind diese Typefehler?

```
SalesItem v = new SalesItem();
RegualrDiscount v1 = new RegularDiscount(9900,10);
DeepDiscount v2 = new RegularDiscount(9900,10);
DeepDiscount v3 = new DeepDiscount(9900,10);
DeepDiscount v4 = new RegularDiscount(9900);
DeepDiscount v5 = new DeepDiscount(9900);
RegularDiscount v6 = new RegularDiscount(9900);
RegularDiscount v7 = new DeepDiscount(9900);
RegularDiscount v8 = new RegularDiscount(9900,10);
RegularDiscount v9 = new DeepDiscount(9900,10);
```

4.3 Aufgabe*, Wetter - Version 2



Das oben angegebene Klassendiagramm definiert Wetterberichte. Es erweitert das Diagramm aus Aufgabe 2.1 Beschreiben Sie, aus welchen Daten ein Wetterbericht besteht und schreiben Sie Code für das gegebene Klassendiagramm.

Erstellen Sie zwei Wetterberichte, einen für Freiburg und einen für Ihr Lieblingsreiseziel mit jeweils mindestens fünf Einträgen.

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 5
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

5.1 Aufgabe, Kuehlschrank [optional]

Sie wollen einen futuristischen Kühlschrank entwerfen. Er soll später in der Lage sein, Überblick über seinen Inhalt zu haben, alle nicht vorhandenen Lebensmittel automatisch übers Internet nachzubestellen und Ihnen auf einem Display in der Kühlschranktür Rezepte zum Kochen anzeigen. Überlegen Sie, welche Informationen der Kühlschrank über die Lebensmittel benötigt, die er aufbewahrt, und erstellen Sie ein Klassendiagramm und Code für den Kühlschrank, die Lebensmittel und die Rezepte.

5.2 Aufgabe, Laufen [optional]

Entwerfen Sie, ohne die Folien der Vorlesung zu verwenden, das Klassendiagramm für die Laufdatenbank.

5.3 Aufgabe, Raumplan [optional]

Sie sollen ein Raumplanungsprogramm entwerfen, mit dem es möglich ist, Veranstaltungen an der Uni Räumen zuzuweisen. Überlegen Sie, welche Eigenschaften ein solches System benötigt, um gut bedienbar zu ein. Einige der Eigenschaften sollten sein:

1. Die Uni besteht aus einer Reihe Fakultäten. Da Sie Ihr Programm nicht für jede Uni neu programmieren wollen, gehen Sie davon aus, dass die Anzahl der Fakultäten beliebig ist.
2. Jede Fakultät besitzt eine Reihe von Gebäuden, in denen die Veranstaltungen stattfinden können.
3. Jedes Gebäude besitzt eine Anzahl von Räumen, die zur Verfügung stehen.
4. Jeder Raum hat eine Ausstattung, wie z.B. Platzanzahl, Beamer, Micro, usw.
5. Da Sie nicht jedes Semester der Universität einen teuren Fachman schicken wollen, der ein neues Semester anlegt, muss die Anzahl der Semester ebenfalls beliebig sein.
6. Gleiches gilt für die vorhandenen Lehrveranstaltungen.
7. Die Belegungsslots bestehen aus einem Wochentag, einer Startzeit, einer Endzeit und einem Semester.

6.1 Aufgabe, Bild - Version 3

Erweitern Sie die image-Klasse aus Aufgabe 3.2 um die folgenden Methoden:

- isPortrait stellt fest, ob die Höhe des Bildes größer ist als die Breite
- size berechnet die Anzahl der Pixel des Bildes.
- isLarger gibt an, ob das Bild größer ist als das andere Bild.

Zeichnen Sie als erstes das entsprechende Klassendiagram. Füllen Sie dann den Code mit den Methodensignaturen aus. Schreiben Sie dann zwei Testfälle je Methode und füllen Sie erst anschließend den Rumpf der Methode mit dem passenden **return**-Statement.

6.2 Aufgabe, Bibliothek - Version 3

Erweitern Sie das Diagramm aus Aufgabe 2.2 und den geschriebenen Code um die folgenden Methoden:

1. thisYear. Diese Methode soll anzeigen, ob das Buch in dem übergebenen Jahr publiziert wurde.
2. thisAuthor. Diese Methode soll anzeigen, ob das Buch von dem übergebene Autor geschreiben wurde.
3. sameAuthor Diese Methode soll angeben, ob das Buch von demselben Autor geschrieben wurde wie ein als Parameter übergebenes Buch.

Zeichen Sie zuerst das Klassendiagramm, schreiben Sie dann die Methodensignaturen auf, erstellen Sie dann zwei Testfälle und füllen Sie den Rumpf als letztes mit den passenden **return**-Statements.

6.3 Aufgabe, Datum

Schreiben Sie für die date-Klasse (siehe vorherigen Blättern) zwei Methoden, die feststellen, welches von zwei Datumswerten kleiner ist.

1. earlierWithoutIf. Verwenden Sie keine **if**-Bedingungen, sondern errechnen Sie das Ergebnis.
2. earlierWithIf. Schreiben Sie dieselbe Methode diesmal mithilfe von **if**-Bedingungen, und dafür ohne Rechnungen.

Schreiben Sie zwei Testfälle für die beiden Methoden.

7.1 Aufgabe, Niederschlag

```
//the daily percipitation of three consecutive days
class Precipitation {
  int day1;
  int day2;
  int day3;

  Precipitation(int day1, int day2, int day3) {
    this.day1 = day1;
    this.day2 = day2;
    this.day3 = day3;
  }

  // how much did it rain during these three days?
  int cumulative() {
    return this.day1 + this.day2 + this.day3;
  }
}

// collect examples of precipitations
class PrecipitationExamples {
  p1 = new Precipitation(1,1,1);
  p2 = new Precipitation(2,1,2);
  p3 = new Precipitation(3,2,4);

  boolean testOne = check this.p1.cumulative() expect 3;
  boolean testTwo = check this.p2.cumulative() expect 5;
  boolean testThree = check this.p3.cumulative() expect 9;
}
```

Fügen Sie der Klasse eine Methode average hinzu. Folgen Sie hier den Entwurfkonzepten der Vorlesung und verwenden Sie vorhandene Methoden, falls dies sinnvoll ist. Fügen Sie ebenfalls zwei Tests zu der Beispielklasse hinzu.

7.2 Aufgabe, Wetter - Version 3

Erweitern Sie die Klasse aus Aufgabe 4.3 um eine Eigenschaft precipitation. Dieses Attribut soll die Regenmenge aufnehmen und Kommazahlen speichern.

Erweitern Sie anschließend die Klasse WeatherRecord um die folgenden Methoden:

1. tempDiff. Diese Methode soll die Temperaturschwankung des Tages berechnen.
2. withinRange. Diese Methode stellt fest, ob die heutigen Messwerte innerhalb der normalen Schwankungen liegen.
3. rainyDay. Diese Methode stellt fest, ob die Regenmenge größer als ein übergebener Wert ist.
4. recordDay. Diese Methode stellt fest, ob die Aufzeichnung den Temperaturrekord (noch oben oder unten) verursacht hat.

Schreiben Sie für die WeatherRecord Klasse drei Testfälle.

7.3 Aufgabe, Sterne

Kennen Sie das Märchen *Die Sterntaler* der Gebrüder Grimm? Aufbauend auf diesem Märchen sollen Sie ein Spiel entwickeln, indem Sie die Bewegung vom Himmel fallender Sterne simulieren.

```
//represent a falling star on a 100x100 canvas
class Star {
  int X = 20;
  int y;
  int DELTA = 5;

  Star(int y) {
    this.y = y;
  }
  ... // bitte ergänzen
}
```

Schreiben Sie passend zu der Klasse eine Methode drop, die Ihren Stern jeweils um DELTA Pixel nach unten bewegt, bis er den Boden erreicht hat. Die Methode drop soll hierbei jeweils einen neuen Stern zurückgeben, und den alten Stern unverändert lassen.

Schreiben Sie mindestens 4 Testfälle, um zu prüfen, ob Ihre Sterne sich korrekt verhalten.

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 8
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

8.1 Aufgabe*, Geometrie

```
// Interface für geometrische Figuren
interface IShape {
  double area ();
  double distTo0();
  BoundingBox bb();
}

// Hilfsklasse
class CartPt {
  int x;
  int y;
  CartPt(int x, int y) {
    this.x = x;
    this.y = y;
  }
  ... // bitte ergänzen, falls nötig
}

// Punkt
class Dot implents IShape {
  CartPt loc;
  Dot(CartPot loc) {
    this.loc = loc;
  }
  double area() { return 0; }
  ... // bitte ergänzen
}

// Rechteck
class Rectangle implements IShape {
  CartPt tlCorner;
  int width; int height;
  Rectangle(CartPt loc, int w, int h) {
    this.tlCorner = loc;
    this width = w; this.height = h;
  }
  ... // bitte ergänzen
}
```

```
// Quadrate
class Square implements IShape {
  CartPt loc;
  int size;

  Square(CartPt loc, int size) {
    this.loc = loc;
    this.size = size;
  }

  double area() {
    return this.size * this.size;
  }
  ... // bitte ergänzen
}

// Kreise
class Circle implements IShape {
  CartPt loc;
  int radius;

  Circle(CartPt loc; int radius) {
    this.loc = loc;
    this.radius = radius;
  }

  double area () {
    return (Math.PI * this.radius * this.radius);
  }
  ... // bitte ergänzen
}
```

Füllen Sie die leeren Stellen, markiert durch *... //bitte ergänzen*. Betrachten Sie dazu das Interface und überlegen Sie, welche Methoden wie aufeinander aufbauen könnten. Die Vorlesung hat Ihnen hier schon einen großen Teil der Arbeit abgenommen.

Die Klasse BoundingBox kennen Sie aus der Vorlesung. Verwenden Sie den Code dieser Klasse, damit Sie keine Fehlermeldung in *ProfessorJ* erhalten.

8.2 Aufgabe*, Geometrie - Version 2

Fügen Sie die Klasse SuperImp, wie aus der Vorlesung bekannt, zu den Klassen aus Aufgabe 8.1 hinzu.

Sie können in *ProfessorJ* zeichnen. Hierzu gibt es eine Libary, die sie mit den Zeilen:

```
import draw.*;
import colors.*;
import geometry.*;
```

laden können. Wählen Sie hier bitte das Sprachlevel `ProfessorJ: Zwischenstufe` aus. Es steht Ihnen dann eine Canvas Klasse zur Verfügung, deren Funktionalität Sie dem folgenden Klassendiagramm entnehmen können:

Canvas
`width` : int `height` : int
`show`() : boolean `close`() : boolean `drawCircle`(`p` : Posn, `r` : int, `c` : IColor) : boolean `drawDisk`(`p` : Posn, `r` : int, `c` : IColor) : boolean `drawRect`(`p` : Posn, `width` : int, `height` : int, `c` : IColor) : boolean `drawString`(`p` : Posn, `s` : String) : boolean

Hinweis: In der DrScheme Version 371 müssen Sie statt IColor den Namen AColor verwenden.

Als Beispiel finden Sie unten ein paar Zeilen Code die Ihnen die Funktion der Canvas Klasse verdeutlicht. Kopieren Sie den Code von der Homepage in den oberen Teil von *ProfessorJ* und tippen Sie im unteren Abschnitt z.B. **new** Test1() ein, um zu sehen, was gezeichnet wird.

```
import draw.*;
import colors.*;
import geometry.*;

class Test1 {
  Canvas c = new Canvas(500,500);
  Test1() {
     c.show();
     c.drawString(new Posn(10,50),"Hallo Welt!");
  }
}

class Test2 {
  Canvas c = new Canvas(500,500);
  Test2() {
    c.show();
    c.drawCircle(new Posn(250,250),100,new Red());
  }
}

class Test3 {
  Canvas c = new Canvas(500,500);
  Test3() {
    c.show();
    c.drawRect(new Posn(100,100),100,25,new Red());
  }
}

class Test4 {
  Canvas c = new Canvas(500,500);
  Test4() {
    c.show();
    c.drawDisk(new Posn(100,100),25,new Red());
  }
}
```

9.1 Aufgabe*, Zeichnen

In Aufgabe 8.2 haben Sie ein Menge von graphischen Objekten modelliert. Schreiben Sie nun zu jeder der Klasse die Methode draw, die einen Canvas als Parameter erhält, und das Objekt entsprechend zeichnet. Erweitern Sie hierzu die Klassen jeweils um die Instanzvariable color (der Typ der Variable ist IColor).

Vergessen Sie nicht, dass Sie bei diesem Prozess das Interface IShape entsprechend anpassen sollten. Ansonsten können Sie in der Klasse SuperImp die Methode draw nicht schreiben.

9.2 Aufgabe, Haus

Schreiben Sie eine Klasse HouseDrawing. Sie soll über eine Methode draw verfügen. Diese Methode soll keinen Parameter als Argument erhalten, sondern ein Haus auf den Canvas zeichnen. Die Parameter zum Zeichnen des Hauses sollen dem Konstruktor der Klasse HouseDrawing übergeben wurde. Überlegen Sie, welche Informationen ein Haus fürs Zeichnen benötigt, und fügen Sie der Klasse entsprechende Instanzvariablen hinzu.

Zeichen Sie ein Klassendiagramm der Klasse HouseDrawing.

9.3 Aufgabe, Sterne - Version 2

Schreiben Sie eine Klasse Sky, die eine Liste von Sternen aufnehmen kann (siehe Aufgabe 7.3 für die Sterne, und für die Listen z.B. Aufgabe 4.1). Schreiben Sie eine Methode draw für die Klasse Star und anschließend für die Klasse Sky, damit Sie den schönen Nachthimmel mit den Sternen betrachten können.

Schreiben Sie eine Funktion step, die aus Ihrer Welt eine neue Welt erzeugt, und alle Sterne Ihrer alten Welt um die entsprechend Pixel verschiebt.

Infos zum Zeichnen:

1. Canvasgröße: 500x500
2. Hintergrund: Blau
3. Sterne: Wie * und gelb

Hinweis: Definieren Sie eine Methode zum Zeichnen von beliebigen Linien.

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 10
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

10.1 Aufgabe, Geometrie - Version 3

Aus Aufgabe 8.1 kennen Sie die Klassen im Umfeld der Geometrieobjekte. In der Vorlesung wurde für dieses Umfeld die Klassen AShape eingeführt, um Gemeinsamkeiten der Klassen zu nutzen (siehe Vorlesung, S. 13, 3. Tag).

Bei den Beispielen in der Vorlesung wurden jeweils nur Codeteile gezeigt. Füllen Sie alle Lücken dieser Klassen.

10.2 Aufgabe, Typen - Ball

```
class Ball {
  double RADIUS = 4.0;
  int DIAMETER = this.RADIUS;
  String w;
  Ball(int w) {
    this.w = w;
  }
}
```

Finden Sie die Typfehler in der aufgelisteten Klasse. Erklären Sie die Fehler auf Papier und geben Sie ein weiteres Beispiel für entsprechende Fehler an.

10.3 Aufgabe, Typen - Rocket

```
interface IWeapon {
  boolean launch();
}

class Rocket implements IWeapen {
  ...
  boolean launch(int countDown) { ... }
}
```

Finden Sie die Typfehler in der aufgelisteten Klasse. Erklären Sie die Fehler auf Papier und geben Sie ein weiteres Beispiel für entsprechende Fehler an.

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 11
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

11.1 Aufgabe, Vererbung - Strings

```
abstract class A {
  String a;
  A(String a) {
    this.a = a;
  }
  String bla(String blub) {
    return this.a.concat(blub);
  }
  String getAValue() {
    return this.a;
  }
}


class B extends A {
  B(String b) {
    super("Dies ist ein B: ".concat(b));
  }
  String bla(String blub) {
    return ("Hurra!".concat(super.bla(blub)));
  }
  String blub(String blub) {
    return ("Test".concat(this.bla(blub)));
  }
}


class C extends B {
  C(String c) {
    super(new String("Dies ist ein C: ").concat(c));
  }
  String bla(String blub) {
    return "Ich bin ein C:".concat(blub).concat(", ").concat(this.getAValue());
  }
}
```

Welche Ergebnisse liefern die folgenden Zeilen, falls Sie diese im *ProfessorJ* eingeben? Erklären Sie die Ausgabe! Versuchen Sie die Aufgabe zuerst auf Papier zu lösen, bevor Sie Ihr Ergebniss in *ProfessorJ* überprüfen.

```
> B b = new B("t");
> b.bla("g")
... ???
> C c = new C("t");
> c.bla("h");
... ???
> B d = new C("hub");
> d.blub("trut");
... ???
```

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 12
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

12.1 Aufgabe, Koordiantenkreuz

Sie wollen ein Programm schreiben, dass es Ihnen erlaubt Funktionen zu zeichnen. Überlegen Sie sich eine Datenrepräsentation für einfache Funktionen, und schreiben Sie eine Funktion, die den Wert eines Terms ausrechnet. Die Syntax der Terme wird anhand folgender Beispiele klar:

1. `f(x) = x + 2`
2. `f(x) = x^2 + 3`
3. `f(x) = 9x^3 + 3x^2 + x + 3`
4. `f(y) = sin(y)`
5. `f(u) = cos(u)`

12.2 Aufgabe, Fahrzeuge - Version 2

Entwerfen Sie eine Klassenhierachie für einen Fahrzeugpark. Es gibt

1. Autos
2. LWKs
3. Motoräder
4. Roller
5. Wohnwagen

Überlegen Sie sich eine gute Struktur, und setzen Sie abstrakte Klassen ein.

Schreiben Sie nun eine Methode draw. Diese Methode soll das Fahrzeug auf einer übergebenen Canvas zeichnen. Die Position, an der sich das Fahrzeug befindet soll zusammen mit der Orientierung des Fahrzeugs der Methode übergeben werden.

Modellieren Sie nun einen Parkplatz mit b x l m^2 Platz. Schreiben Sie eine entsprechende Klasse CarPark.

Der Klasse sollen Sie mithilfe der Methode add neue Fahrzeuge hinzufügen können. Als Rückgabe der Methode erhalten Sie die Koordinaten, an denen das Fahrzeug abgestellt wurde.

Die Klasse (CarPark) soll immer sicherstellen, dass keine Fahrzeuge übereinander stehen. Außerdem soll der Parkplatz zusammen mit den Fahrzeugen gezeichnet werden können, indem die Methode draw aufgerufen wird.

1. CartPt add(IVehicle v)
2. draw(ICanvas c)

12.3 Aufgabe, Fahrzeuge - Version 3

Erweitern Sie alle Fahrzeuge um die Methode toString, die eine Stringdarstellung des Fahrzeugs liefert. Achten Sie hier auch darauf, die abstrakten Klassen einzusetzen.

Der Fahrzeugpark soll anschließend eine String liefern, der durch Semikolon getrennt alle Fahrzeuge darstellt und die Größe des Platzes ausgibt. Ein Beispiel:

```
Parkplatz: Größe: 100m x 100m,
  Fahrzeuge:
   [Auto: BMW, 100 KW, grün, 3.20 m x 1.9 m, P: 10/10, O: 0 Grad;
    Auto: Opel, 80 KW, gelb, 2.80 m x 1.8 m, P: 15/20, O: 90 Grad;
    Roller : Opel, 15 KW, rot, 1.20 m x 0.4 m; P: 40/40, O: 180 Grad;
    LKW: Mercedes, 500 KW, schwarz, 20 m x 2.7 m, P: 80/80, O: 0 Grad]
```

Hierbei soll das P für die Position des Fahrzeugs stehen, und das O für die Orientierung des Fahrzeugs auf dem Parkplatz.

12.4 Aufgabe, Fahrzeuge - Version 4 [optional]

Für diese Aufgabe wissen Sie nicht, wie groß Ihr Parkplatz ist. Sie können davon ausgehen, dass er rechteckig ist, und dass Sie die Länge und Breite des Platze durch den Konstruktoraufruf erfahren.

Erweitern Sie die Klasse CarPark um eine Funktion, die sicherstellt, dass der Parkplatz noch benutzbar ist. D.h. Sie sollen sich einen Algorithmus überlegen, der überprüft, ob jedes Fahrzeug den Parkplatz verlassen kann, ohne das andere Fahrzeuge umgestellt werden müssen.

Halten Sie dabei folgende Eigenschaften ein:

1. Jedes Fahrzeug ist ein Rechteck, das der Breite und Länge der Fahrzeuge entspricht. Es kann hierbei jeweils um 90 Grad gedreht werden. Schräg geparkte Fahrzeuge müssen Sie nicht berücksichtigen.
2. Der Parkplatz ist nicht eingeteilt in Parkflächen. Jedes Auto kann somit von Ihnen frei positioniert werden.
3. Sie wissen nicht, in welcher Reihenfolge Autos den Parkplatz verlassen oder betreten wollen.
4. Die Linke untere Ecke des Platzes ist die Ein- und Ausfahrt des Platzes.
5. Alle Fahrzeuge können sich auf der Stelle drehen, indem Sie nur ein paar cm nach vorne fahren, dann wieder nach hinten, usw.

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 13
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

13.1 Aufgabe, Zeichenen von Linien

```
import draw.*;
import colors.*;
import geometry.*;

class Line {
  Canvas theCanvas = new Canvas(500,500);
  AColor color;

  Line(AColor c) {
    this.color = c;
  }

  boolean show() {
    return this.theCanvas.show();
  }

  void draw(Posn p1, Posn p2) {
    // compute width and hight
    int dx = p1.x - p2.x;
    int adx = Math.abs(dx);
    int dy = p1.y - p2.y;
    int ady = Math.abs(dy);
    // bei p1 = p2 brauchen wir nicht zeichnen
    if ((ady == 0) && (adx == 0)) {
      return;
    }
    // p1 != p2, d.h. ady != 0 oder adx != 0
    if (ady < adx) {
      // dx != 0, da 0 <= |dy| < |dx|
      double ratio = (double) dy / dx;
      Posn pstart;
      if (p1.x < p2.x) {
        pstart = p1;
      } else {
        pstart = p2;
      }
      drawHorz(pstart,ratio,0,adx);
    } else {
      // dy != 0, da |dy| >= |dx| und einer != 0
      double ratio = (double) dx / dy;
      Posn pstart;
      if (p1.y < p2.y) {
        pstart = p1;
      } else {
        pstart = p2;
      }
      drawVert(pstart,ratio,0,ady);
    }
  }
```

Fortsetzung nächste Seite →

```
  void drawHorz(Posn p1,double ratio, int x, int w) {
    if (x <= w) {
      int y = (int) Math.round(p1.y + x * ratio);
      Posn p = new Posn(p1.x + x, y );
      this.theCanvas.drawRect(p,1,1, this.color);
      drawHorz(p1,ratio,x+1,w);
    }
  }

  void drawVert(Posn p1,double ratio, int y, int h) {
    if (y <= h) {
      int x = (int) Math.round(p1.x + y * ratio);
      Posn p = new Posn( x , p1.y + y );
      this.theCanvas.drawRect(p,1,1, this.color);
      drawVert(p1,ratio,y+1,h);
    }
  }
}
```

Der obriege Code (Voller Sprachumfang) zeichnet Linien zwischen zwei Punkten. Malen Sie auf Ihrem Blatt ein 10x10 großes Feld, und simulieren Sie den Algrithmus für die Linien von (2,3) nach (5,7) und von (1,1) nach (1,5).

Geben Sie bei jedem Aufruf von drawRect die Werte der Variablen (p,x,y,(h oder w),ratio,p1) an.

Programmierzertifikat Objekt-Orientierung mit Java, Blatt Nr. 14
http://proglang.informatik.uni-freiburg.de/teaching/java/2008/

14.1 Aufgabe, Tetris

Ziel: Ein Tetrisspiel

1. Wir beginnen mit einer einfachen Version, in der alle Steine nur quadratisch sind, und sich nach unten bewegen. Hier soll es noch nicht möglich sein, die Steine mit den Tasten zu verschieben. Für diesen ersten Schritt beginnen wir mit folgender Vorlage:

```
import idraw.*;
import colors.*;
import geometry.*;

class Tetris extends World {
  AColor BACKBGROUND = new White();

  Tetris() { ... }

  // change the World, i.e. move the Block downwards
  public void onTick() {
    return;
  }

  // draw the world, do not change the world here
  public void draw() {
    ...
  }

  // change the world
  public void onKeyEvent(String key) {
    if (key.equals("q")) {
      this.theCanvas.close();
      this.endOfWorld("Spiel zuende");
    }
  }

}
```

   Erstellen Sie Klassen, die Ihre Tetriswelt und die Informationen, die Sie benötigen werden, speichern können. Zeichen Sie diese als Klassendiagramm auf. Fragen Sie nun einen Tutor, ob dieses Klassendiagramm sinnvoll ist, bevor Sie fortfahren.

   Folgende Klassen und Interface sollten Sie erstellen (eventuell nicht vollständige Liste):

   a) Tetris (das gesammte Spiel)
   b) Fild (das Spielfeld)
   c) IBlock, Block (ein Block)
   d) IStone, Stone (ein Spielstein, zusammengesetzt aus 4 Blöcken)
   e) IDropBlock, DropBlock (Blöcke, die sich nach unten bewegen)
   f) BlockList (die Liste der Blöcke des Feldes)

2. Schreiben Sie die Klasse Block, die einen Block darstellt, und diesen auf eine Canvas zeichnen kann.

   Testen Sie den Code, indem Sie eine Block in der Welt zeichnen.

3. Schreiben Sie eine Klasse DropBlock, die von der Klasse Block abgeleitet ist, und die sich pro Zeiteinheit eine feste Anzahl Pixel nach unten bewegt. Hierzu sollte die Klasse ein onTick Methode erhalten.

   Binden Sie nun einen Block in Ihr Spiel ein, und rufen Sie die onTick Methode immer aus der onTick Methode der Weltklasse aus auf.

Nun sollte Sie eine erste Version besitzen, in der eine DropBox nach unten fällt, bis sie nicht mehr zu sehen ist.

4. Schreiben Sie einen Test, mit dem Sie sicherstellen, dass die Boxen nicht aus der Zeichenfläche fallen, und dass sie nicht übereinander zu liege kommen.

   Fügen Sie hier zuerst der Klasse Block eine Methode hinzu, die prüft, ob sich zwei Blöcke schneiden.

   Schreiben Sie anschließend eine Methode, die für alle in dem Spielfeld vorkommenden Blöcke ein übereinanderliegen feststellt, .d.h. true liefert, falls eine Überlagerung vorhanden ist, und false, falls es keine Überlagerung gibt.

5. Falls eine Box den Boden oder eine andere Box erreicht hat, soll diese an der Stelle verweilen, und eine neue Box soll oben in der Mitte des Spielfelds erscheinen.

   Vergessen Sie nicht die Box, die nun fest ist, geeignet zu speichern. Sonst können Sie bei den folgenden Boxen keine Kollision abfragen oder testen, ob eine Reihe mit Boxen angefüllt ist.

6. Fragen Sie ab, ob der Benutzer die Taste rechts oder links gedrückt hat, und verschieben Sie dann die Box, die sich nach unten bewegt, in die entsprechende Richtung.

7. Falls eine Reihe voll ist, löschen Sie diese, und bewegen Sie alle Boxen eine Reihe nach unten.

8. Vergeben Sie Punkte für gelöschte Reihen

9. Erstellen Sie nun Steine, die andere Formen haben

10. Schreiben Sie die Methode rotate für die Steineklasse.

11. Zeigen Sie eine Vorschau an, damit der Spieler den nächsten Stein sehen kann

12. ...