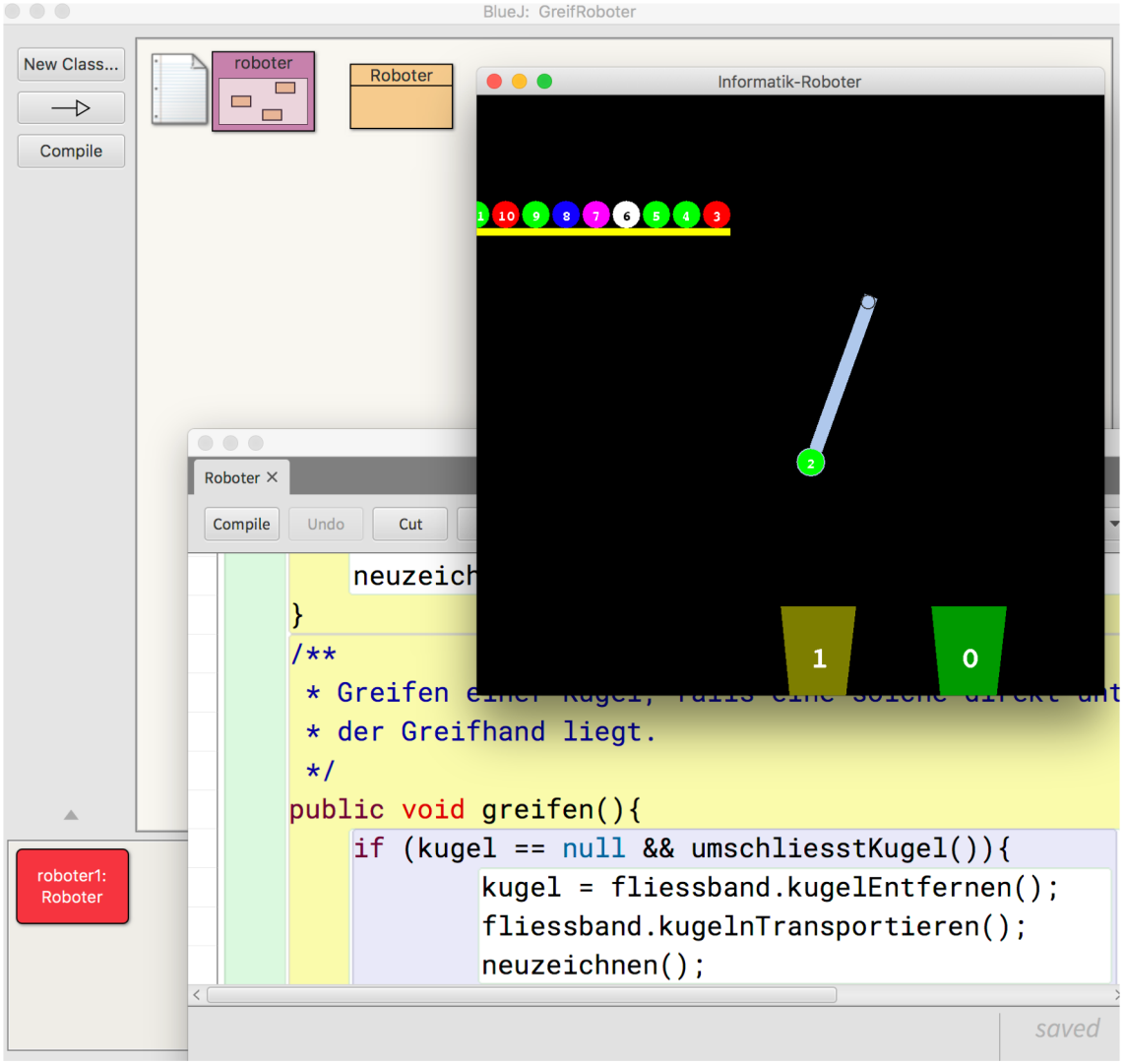
Informatik 10

Objektorientiertes Modellieren und Programmieren mit Java



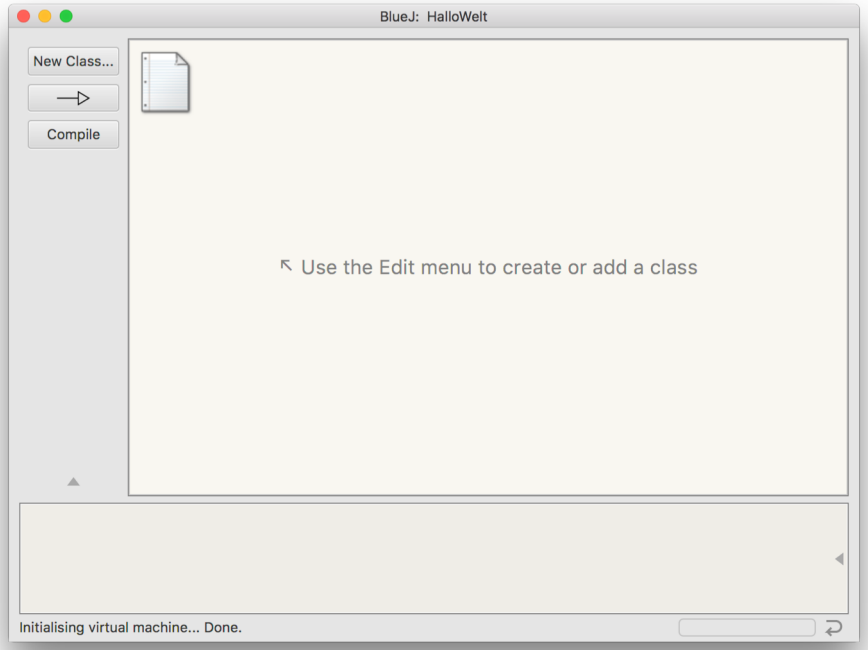
1. Klassen und Objekte

Zunächst wollen wir mit Hilfe eines Miniprogramms die Entwicklungsumgebung BlueJ kennen lernen.

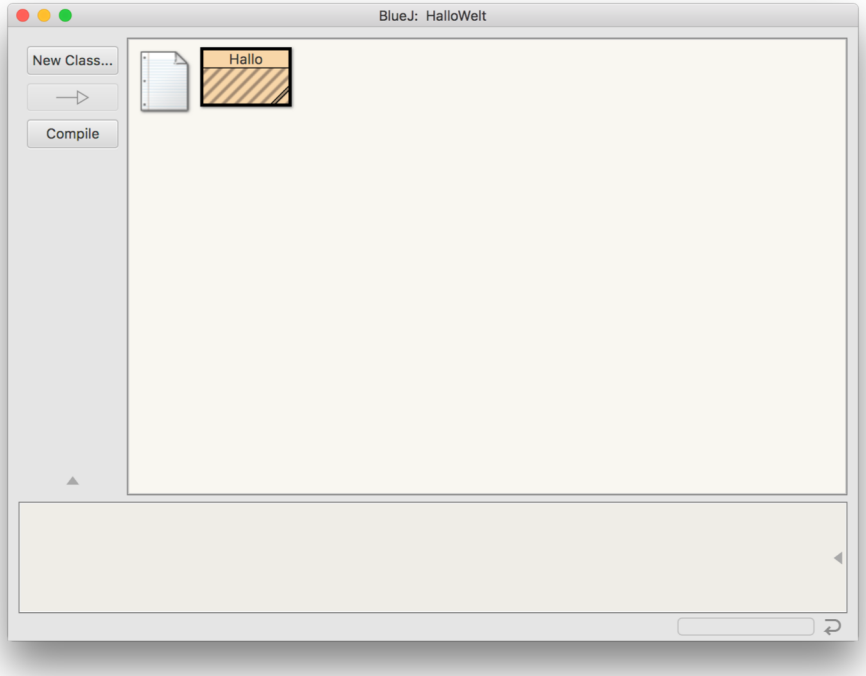
Die Installation von Java und BlueJ ist in der Datei Installation_Java_jdk_und_BlueJ.pdf im Ordner Material beschrieben.

Unter Informatikern ist es Tradition, zunächst ein Programm zu schreiben, das den Text "Hallo Welt!" auf dem Bildschirm ausgibt.

Öffne BlueJ und starte ein neues Projekt mit dem Namen HalloWelt:



Erzeuge eine neue Klasse mit dem Namen Hallo. Mit einem Doppelklick auf das Symbol öffnest du den Editor mit dem Java-Quelltext deiner Klasse.

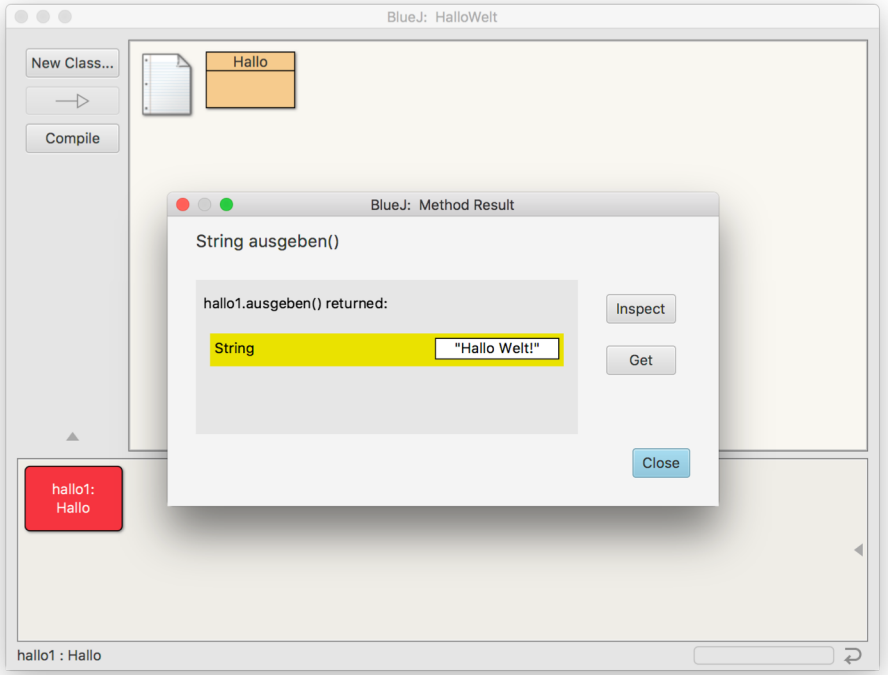


Lösche den gesamten vorgegebenen Text und gib dann im Editor folgenden Text ein. Achte dabei genau auf die Klammern, Zeichensetzung sowie Groß- und Kleinschreibung.

```
public class Hallo{
    String text = "Hallo Welt!";
    public String ausgeben(){
        return text;
    }
}
```

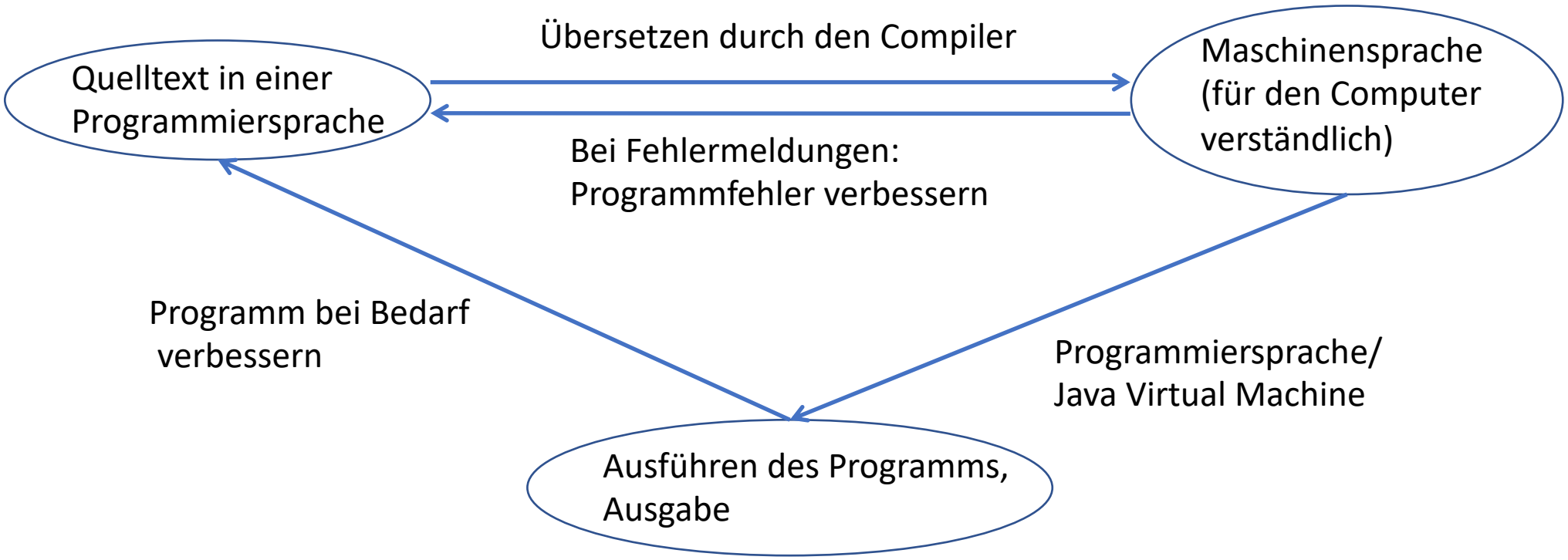
Klicke anschließend auf Compile. Damit wird der Text in eine für den Computer verständliche Maschinensprache übersetzt.

Erzeuge mit der rechten Maustaste und new Hallo() ein neues Objekt (Instance) der Klasse Hallo.
Das Objekt wird in BlueJ im unteren Bereich des Fensters dargestellt.
Mit der rechten Maustaste kannst du die Methode ausgeben() aufrufen.
Sie gibt den Wert des Attributs text zurück.



MERKE

Vorgänge im Computer beim Programmieren:



Ein Javaprogramm besteht aus mehreren **Klassen**.

Eine **Klasse** ist eine Art Bauplan, die festlegt, welche **Attribute** (Farbe, Breite, Höhe,...) und welche **Methoden** (ausgeben(), setzeFarbe(...), ...) die Objekte der Klasse haben sollen.

Von einer Klasse kann man verschiedene **Objekte** (**Instanzen**) dieser Klasse erzeugen, die sich in ihren **Attributwerten** unterscheiden können.

MERKE

Der logische Aufbau einer Klasse wird durch eine **Klassenkarte** veranschaulicht.

Eine Klassenkarte ist ein Rechteck, das aus drei Bereichen besteht:

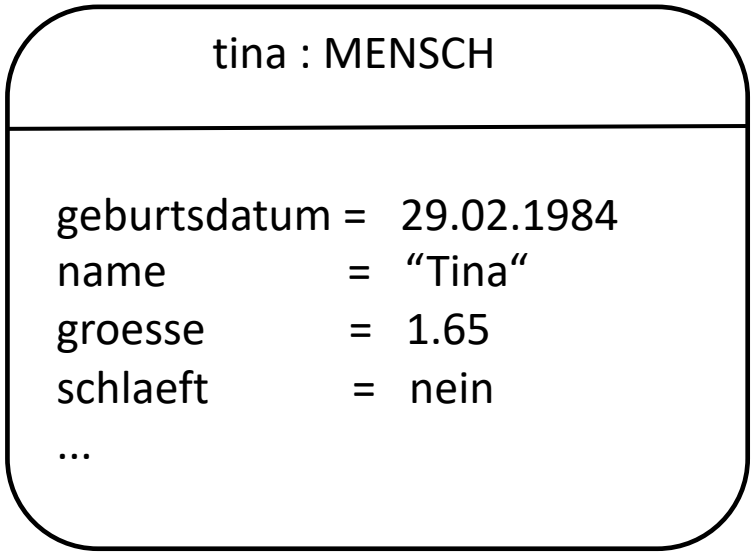
Oben: Name der Klasse
Mitte: Attribute und Datentypen
Unten: Methoden

MERKE

Objekte werden durch **Objektkarten** veranschaulicht.

Eine Objektkarte ist ein Rechteck mit abgerundeten Ecken,
das aus zwei Bereichen besteht:

Oben: Name und Klasse des Objekts
Mitte: Attribute und deren Werte



Objekte spricht man in der
Punktnotation an:

Objektname.Methodenaufruf

Beispiele:

tina.laecheln()
tina.schlafen(7 Stunden)

MERKE

Verschiedene **Arten von Methoden**:

Verändernde Methoden
bringen das Objekt in einen anderen **Zustand**: **setzeSchriftgroesse(12)**

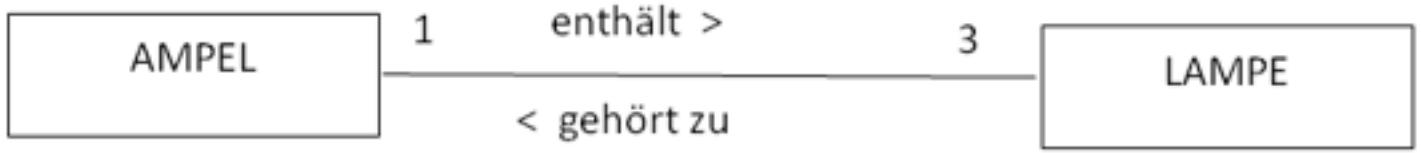
Sondierende Methoden
geben Informationen aus: **gibName()**

Methoden mit Übergabeparameter
Der Wert in den runden Klammern wird der Methode übergeben: **schlafen(7)**

Methoden ohne Übergabeparameter
erkennt man an den leeren runden Klammern **laecheln()**

MERKE

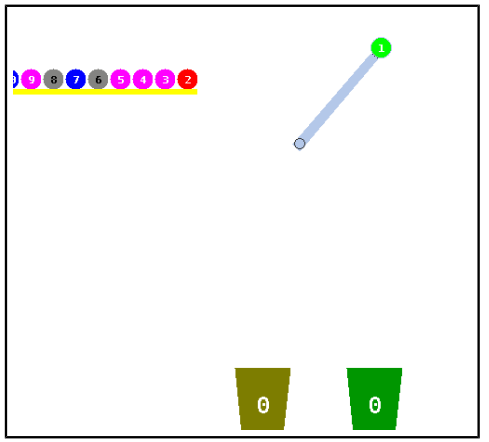
Beziehungen zwischen den Klassen kann man in einem **einfachen Klassendiagramm** darstellen:



Dazu nimmt man nur die oberen Teile der Klassenkarte und verbindet sie durch eine Linie, die aussagekräftig beschriftet wird.

Zusätzlich kann man noch die **Kardinalitäten** angeben. Dies sind Zahlen, die beschreiben, wie viele Objekte der einen Klasse in Beziehung zu der anderen Klasse stehen.

Übung 1 mit Beispiel-Objekten, Greifroboter



Öffne das BlueJ-Projekt „GreifRoboter“.

a) Erzeuge ein Objekt der Klasse Roboter und nenne es *greifi*.

b) Veranlasse *greifi*, die erste Kugel zu greifen.

c) Lasse *greifi* den Arm um 20° nach rechts drehen.

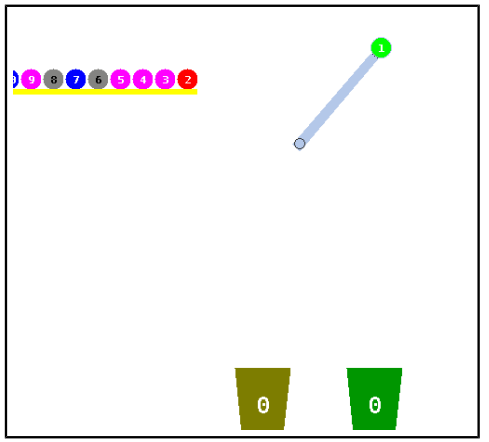
d) Frage *greifi* nach der Farbe der gegriffenen Kugel.

e) Frage *greifi* nach dem aktuellen Winkel.

f) Finde heraus, wo der Winkel 0° ist.

g) Schreibe die Methodenaufrufe von b) bis e) in Punktnotation auf.

Übung 2 mit Beispiel-Objekten, Greifroboter



Öffne das BlueJ-Projekt „GreifRoboter“.

a) Zähle alle Methoden mit Übergabeparameter auf.

b) Zähle alle Methoden ohne Übergabeparameter auf.

c) Zähle alle sondierenden Methoden auf.

d) Zähle alle verändernden Methoden auf.

e) Zu welchen Klassen hat der Roboter eine Beziehung? Zeichne dazu ein einfaches Klassendiagramm.

Übung 3 mit Beispiel-Objekten, JavaKarol

Öffne das BlueJ-Projekt „JavaKarol“.

a) Erzeuge ein Objekt der Klasse WELT mit Breite 10, Länge 10 und Höhe 5. Nenne sie *welt1*.

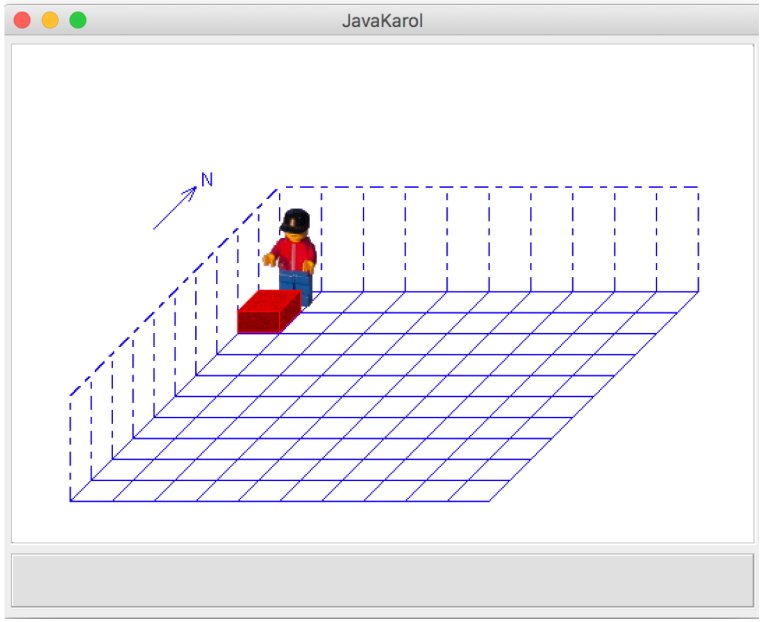
b) Erzeuge ein Objekt der Klasse ROBOTER, nenne es *rob* und setze *rob* in *welt1.*

c) Nenne alle sondierenden und alle verändernden Methoden.

d) Wodurch unterscheiden sich im Quelltext die beiden Methoden ROBOTER(...) wesentlich von den anderen Methoden? Was bewirken sie?

e) Die Klasse ROBOTER zeigt keine Attribute (sie werden von einer nicht sichtbaren Klasse geerbt). Welche Attribute könnte sie haben?

f) Zeichne eine Objektkarte von *rob*, so wie er im Bild gezeichnet ist.



MERKE

Für Attribute gibt es verschiedene **Datentypen**.
Wichtige Datentypen in Java sind:

Datentyp	Bedeutung	Beispiele
int	integer, ganze Zahl	27; 0; -1024
double	rationale Zahl	-3.14 ; 5.0E+9
char	character, Zeichen	’e’ ; ’D’ ; ’5’ ;
String	Zeichenkette	”Hallo Welt!” ; ”27”; ”e”; ” ” ;
boolean	logische Variable	true; false; 4<5 ; 4==5; 4!=5;

Übung 4: Objekte, Klassen, Datentypen

Öffne das BlueJ-Projekt „ alphaFormen_Zeichnung_KoSy“.

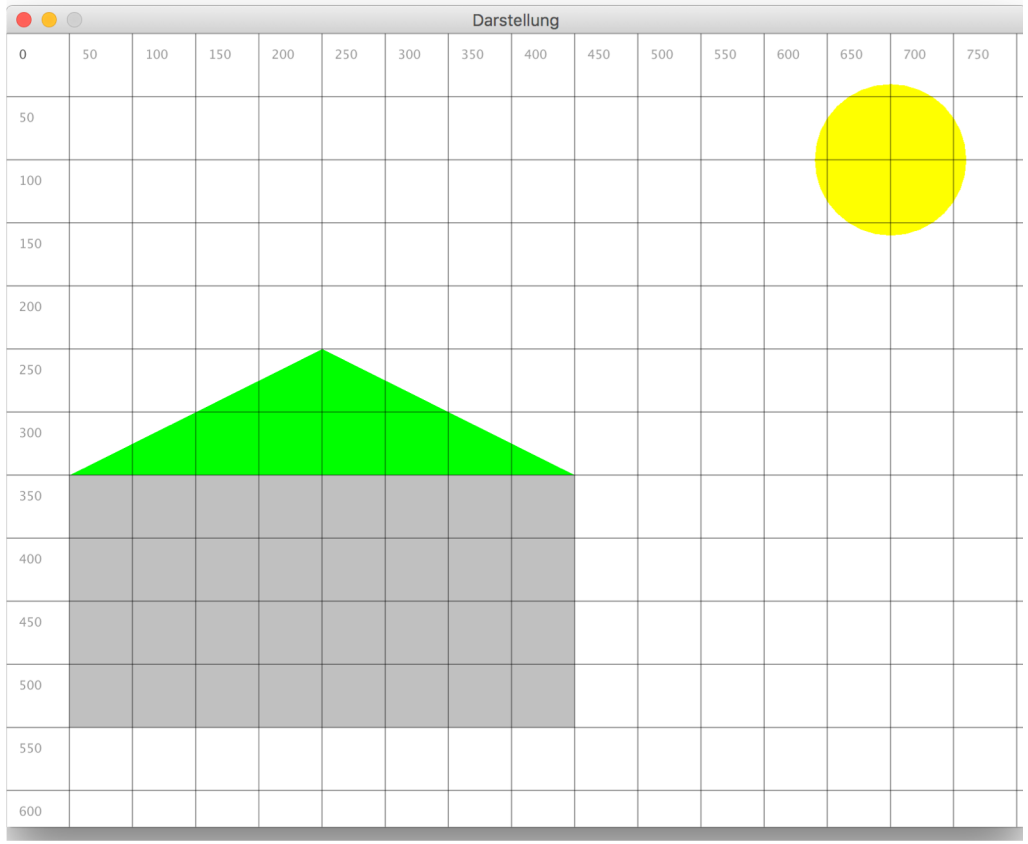
a) Erzeuge ein Objekt der Klasse KREIS und nenne es *sonne.*
Färbe es gelb und verschiebe es in der Zeichnung.
Erforsche dabei das Koordinatensystem.

b) Erzeuge ein Objekt der Klasse RECHTECK, nenne es *wand* und färbe es hellgrau.

c) Erzeuge ein Objekt der Klasse DREIECK, nenne es *dach* und färbe es grün .

d) Erstelle daraus ein Haus mit Sonne ähnlich wie in der Abbildung.

e) Notiere die benötigten Methodenaufrufe in Punktnotation.



Übung 5 : Objekte, Klassen, Javacode schreiben

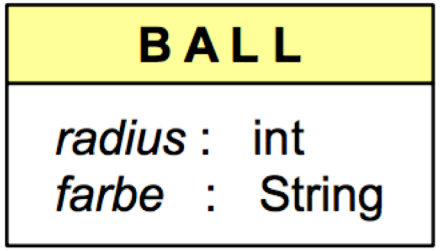
Öffne das BlueJ-Projekt „ alphaFormen_Zeichnung_KoSy“.

a) Öffne die Klasse ZEICHNUNG und sieh dir den Javacode in Ruhe an. Hast du Fragen?

b) Schreibe nun nach den Zeilen
// ... und legst ihr Aussehen fest
// (Fachsprache: initialisieren der Attribute)
die Methodenaufrufe, die du in der Übung 4 notiert hast.
Beende jede Zeile mit einem Strichpunkt.
Klicke nach jeder Zeile auf compile (übersetzen) und verbessere gegebenenfalls deine Fehler.

c) Erzeuge ein neues Objekt der Klasse ZEICHNUNG.
Sieht es so aus, wie du wolltest? Bessere bei Bedarf nach.

Eigene Klassen erstellen

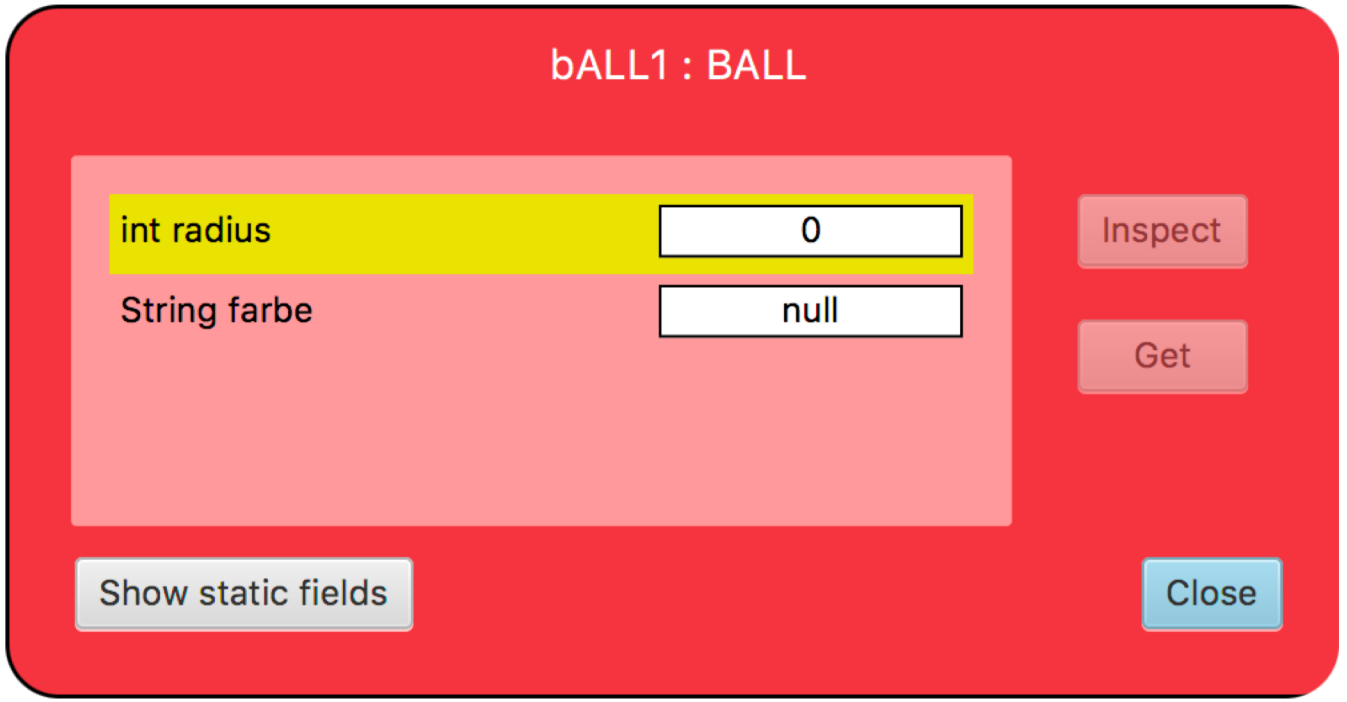


Öffne das BlueJ-Projekt „ alphaFormen“.
Erstelle in diesem Projekt eine neue Klasse mit dem Namen BALL und öffne den Editor.
Lösche den gesamten vorgegebenen Quelltext und schreibe im Editor folgenden Java-Code:

```
public class BALL{
    int radius;
    String farbe;
}
```

Eigene Klassen erstellen

Erzeuge ein Objekt der Klasse BALL und öffne den Objektinspektor (Doppelklick auf das Objektsymbol oder rechte Maustaste und inspect)



Eigene Klassen erstellen, Konstruktor

Die Attribute radius und farbe haben die Standardwerte 0 bzw. “null“.

Dies liegt daran, dass wir Attribute nur **deklariert** (bestellt, beantragt), aber noch nicht **initialisiert** haben, also ihnen noch keine Werte zugewiesen haben.

Dies kann bei einem neuen Objekt mit einer besonderen Methode, dem **Konstruktor**, erledigt werden.

Den Konstruktor ruft man auch auf, wenn man mit new BALL() ein neues Objekt erzeugt.

Eigene Klassen erstellen, Konstruktor

Ergänze nun im Quelltext den Konstruktor:

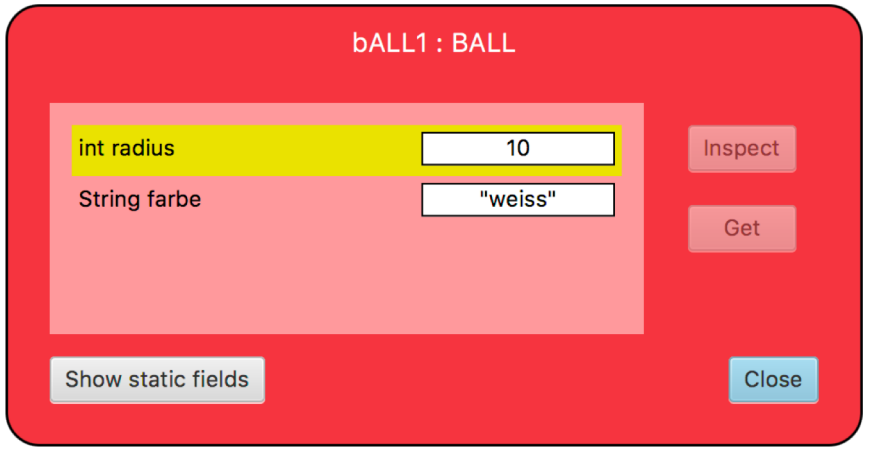
```
public class BALL{
   int radius;
   String farbe;

   public BALL(){
      this.radius = 10;
      this.farbe = "weiss";
   }
}
```

Konstruktor der Klasse BALL

Das Wort “this“ spricht das aktuelle Objekt der Klasse an. Man könnte es in diesem Beispiel auch weglassen.

Erzeuge ein neues Objekt und prüfe im Objektinspektor, ob die Attribute die von dir programmierten Werte haben.



Eigene Klassen erstellen, Konstruktor

Eine Klasse kann mehrere Konstruktoren besitzen:

```
public class BALL{
   int radius;
   String farbe;

   public BALL(int radiusNeu, String farbeNeu){
      this.radius = radiusNeu;
      this.farbe = farbeNeu;
   }
}
```

Konstruktor mit Übergabeparameter

Erzeuge auch mit diesem Konstruktor ein neues Objekt und prüfe im Objektinspektor.

Das Wort “this“ spricht wieder das aktuelle Objekt der Klasse an. Auch in diesem Beispiel könnte man es weglassen, es verdeutlicht aber die Zugehörigkeit der Attribute.

MERKE

Im **Kopf der Klasse** werden Attribute **deklariert** (bestellt, beantragt).

```
public class BALL{
   int radius;
   String farbe;

   //...
}
```

MERKE

Der **Konstruktor** ist eine spezielle Methode, in der Attribute **initialisiert**, d.h. ihnen Werte zugewiesen, werden.

```
public BALL(){
  this.radius = 10;
  this.farbe = "weiss";
}
```

Mit dem Wort **this** spricht man das akuelle Objekt der Klasse an.

Der Name des Konstruktors muss **genau so geschrieben** werden wie der Klassenname.

Er bekommt am Ende **runde Klammern**, um ihn als Methode zu kennzeichnen.

MERKE

Ein Klasse kann mehrere Konstruktoren besitzen. Ein Konstruktor kann auch **Übergabeparameter** haben.

```
public class BALL{
  int radius;
  String farbe;

  public BALL(){
    this.radius = 10;
    this.farbe = "weiss";
  }

  public BALL(int radiusNeu, String farbeNeu){
    this.radius = radiusNeu;
    this.farbe = farbeNeu;
  }
}
```

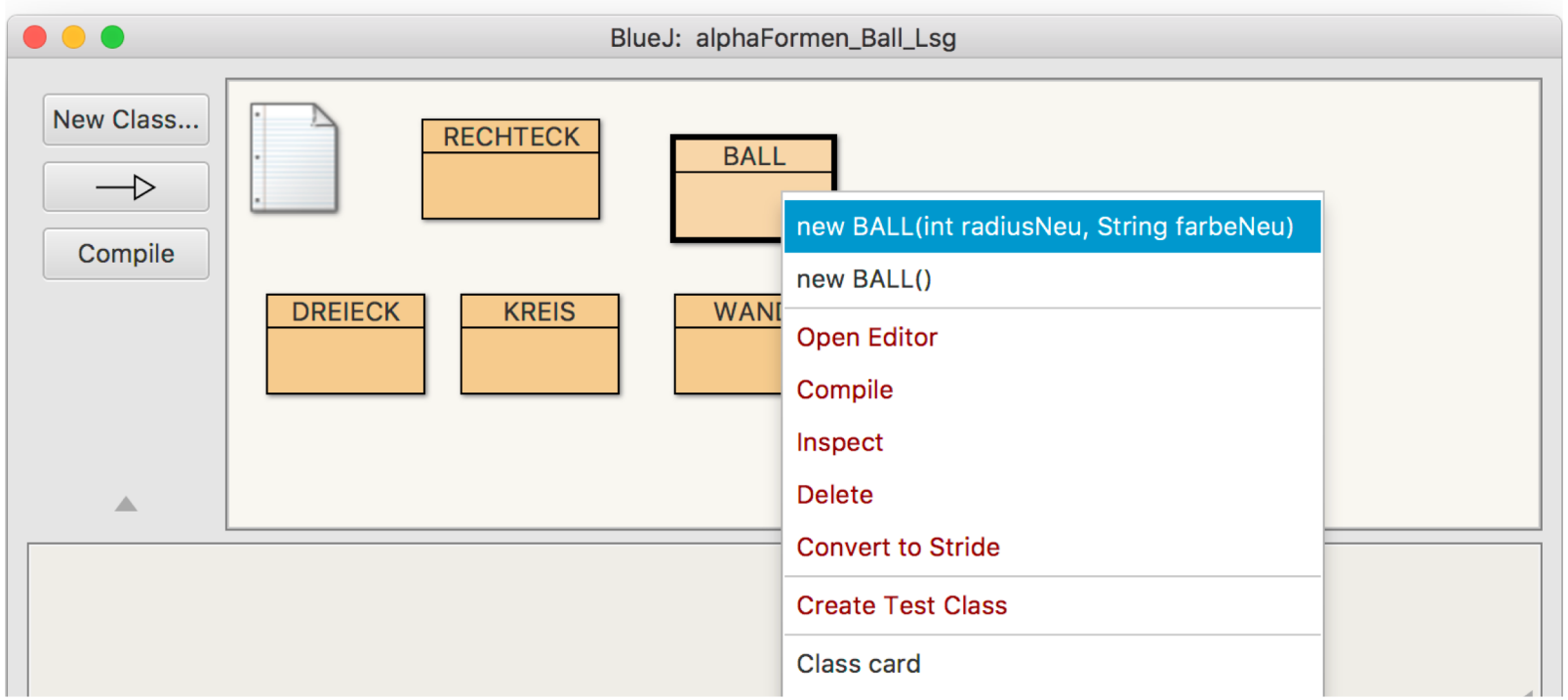
Konstruktor 1

Konstruktor 2

MERKE

Der Konstruktor wird mit dem **new**-Operator aufgerufen, wenn man ein neues Objekt der Klasse erzeugen möchte.

z.B. `new BALL()` oder `new BALL(20, "blau")`



Übung6: Eigene Klassen erstellen, Klasse WAND

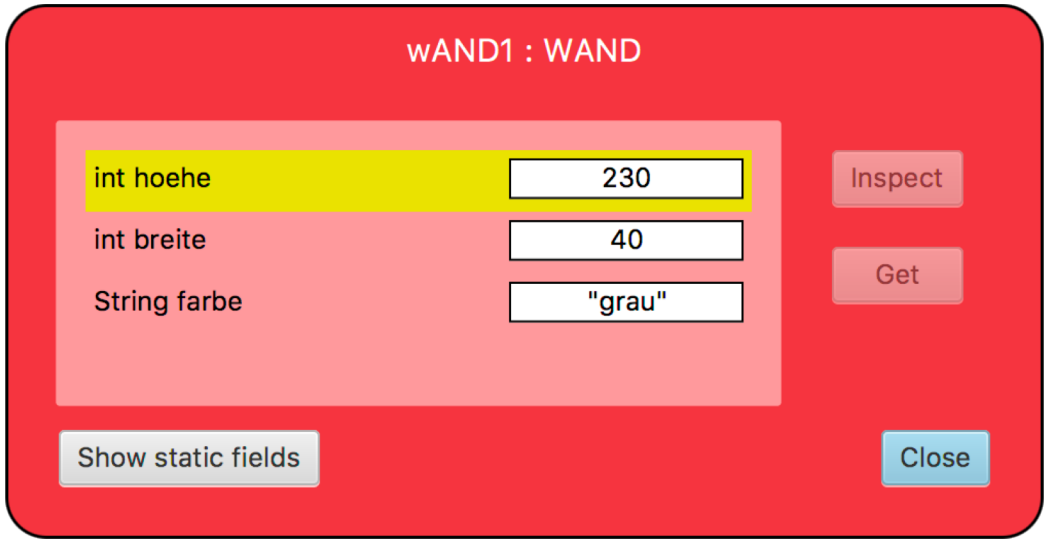
Schreibe analog zur Klasse BALL eine weitere Klasse WAND.

Deklariere die Attribute `hoehe, breite` und `farbe`.

Initialisiere die Attribute im Konstruktor mit den Werten `230, 40` und `“grau“.`

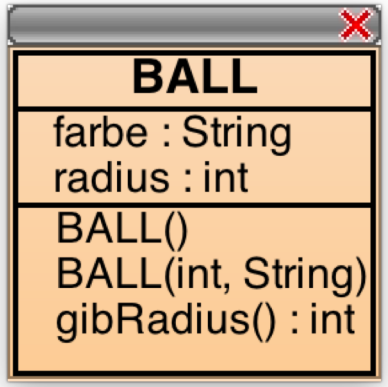
Schreibe einen zweiten Konstruktor mit Übergabeparameter.

Erzeuge ein Objekt von WAND und überprüfe die Werte.



Eigene Klassen erstellen, sondierende Methoden

Die Klasse BALL soll eine sondierende Methode haben, die den Radius des Objekts ausgibt:

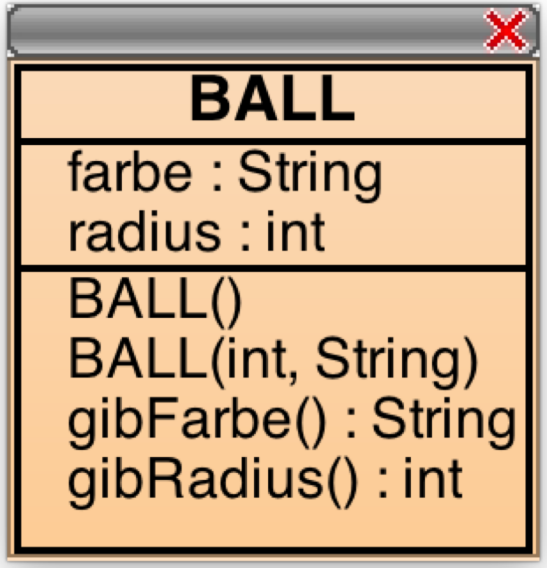


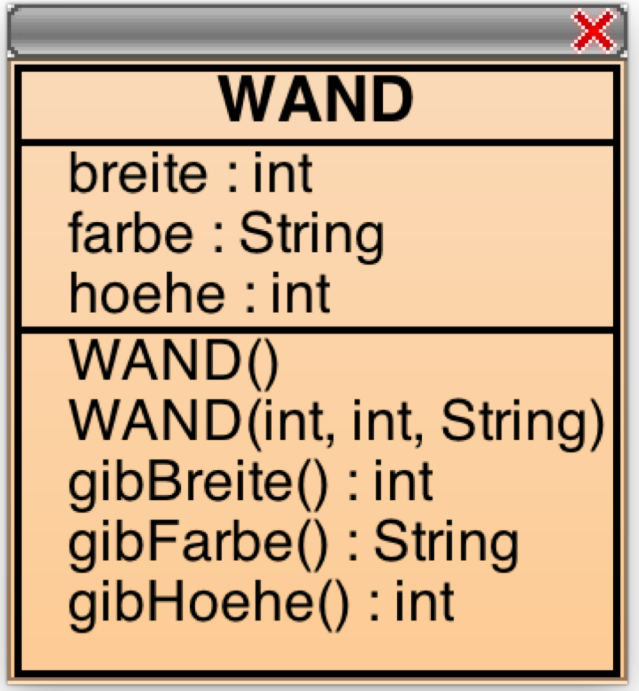
Ergänze in der Klasse BALL den Quelltext:

```
public int gibRadius(){
   return radius;
}
```

Übung 7:
Eigene Klassen erstellen, sondierende Methoden

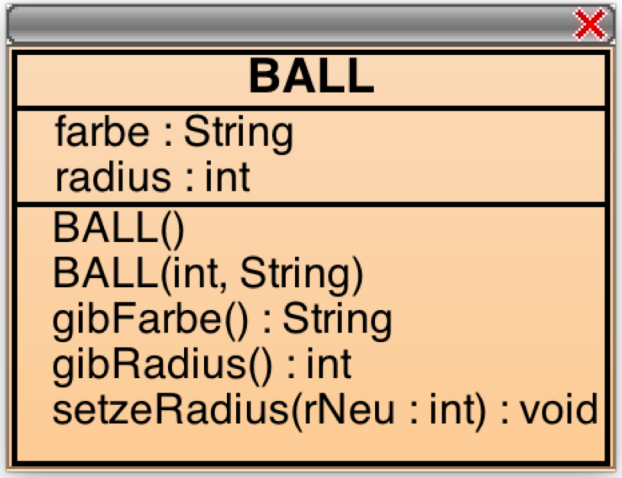
Ergänze die Klassen BALL und WAND um die sondierenden Methoden:





Eigene Klassen erstellen, verändernde Methoden

Die Klasse BALL soll nun auch eine verändernde Methode haben, die den Radius des Objekts aktualisiert:

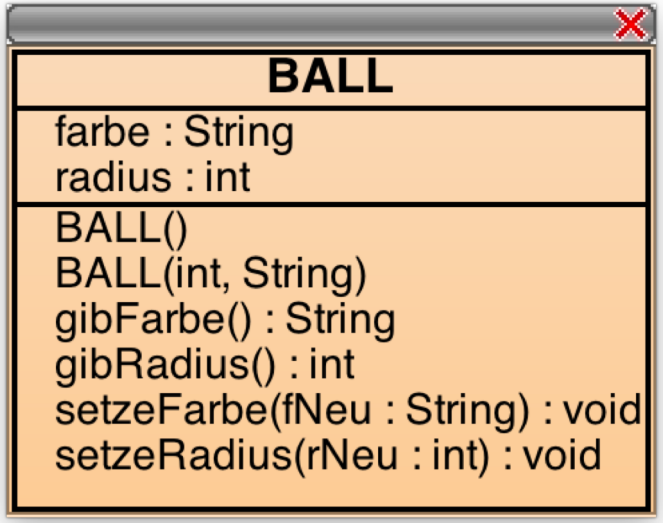


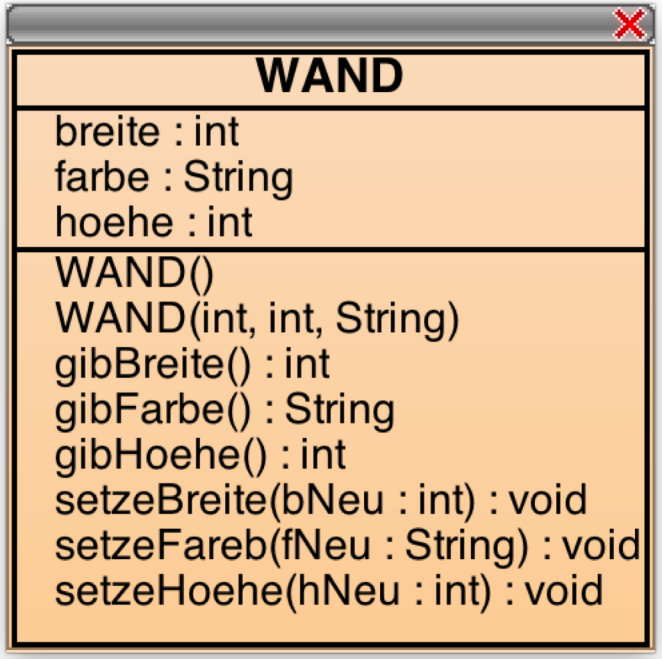
Ergänze in der Klasse BALL den Quelltext:

```
public void setzeRadius(int rNeu){
   radius = rNeu;
}
```

Übung 8:
Eigene Klassen erstellen, verändernde Methoden

Ergänze die Klassen BALL und WAND um die verändernden Methoden:





MERKE

Eine sondierende Methode hat den folgenden Aufbau:

```
public Datentyp-der-Antwort Name-der-Methode () {
   return Attribut;
}
```

Beispiel:

```
public String gibFarbe(){
   return farbe;
}
```

MERKE

Eine verändernde Methode hat den folgenden Aufbau:

```
public void Name-der-Methode (Datentyp Parameter) {
   Attribut = Parameter;
}
```

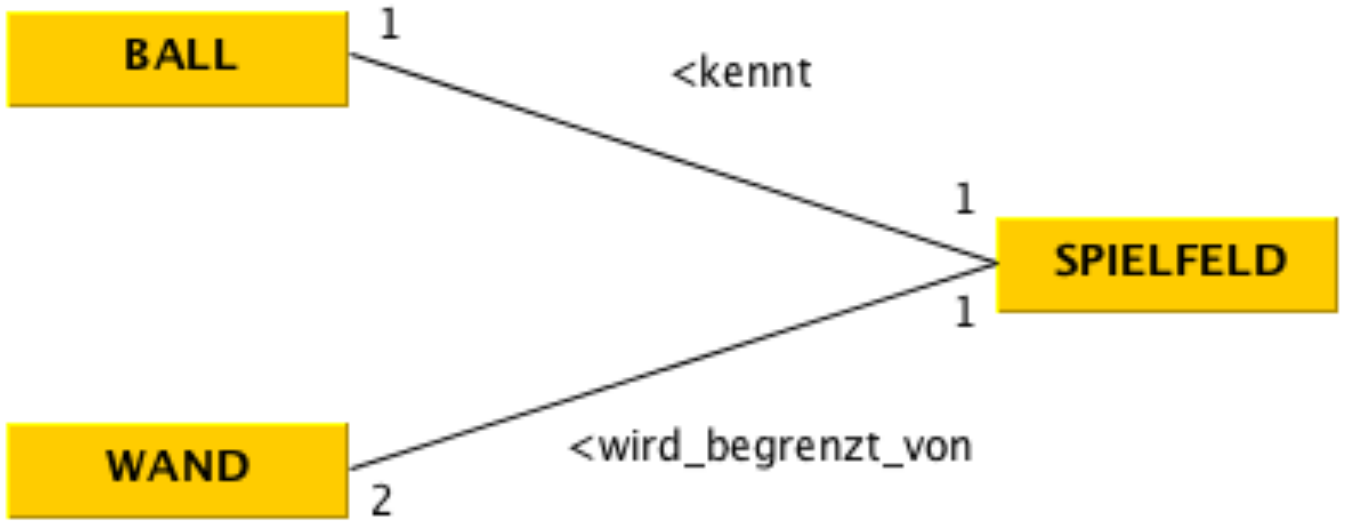
Beispiel:

```
public void setzeHoehe(int hNeu){
   hoehe = hNeu;
}
```

Eigene Klassen erstellen, Referenzattribute

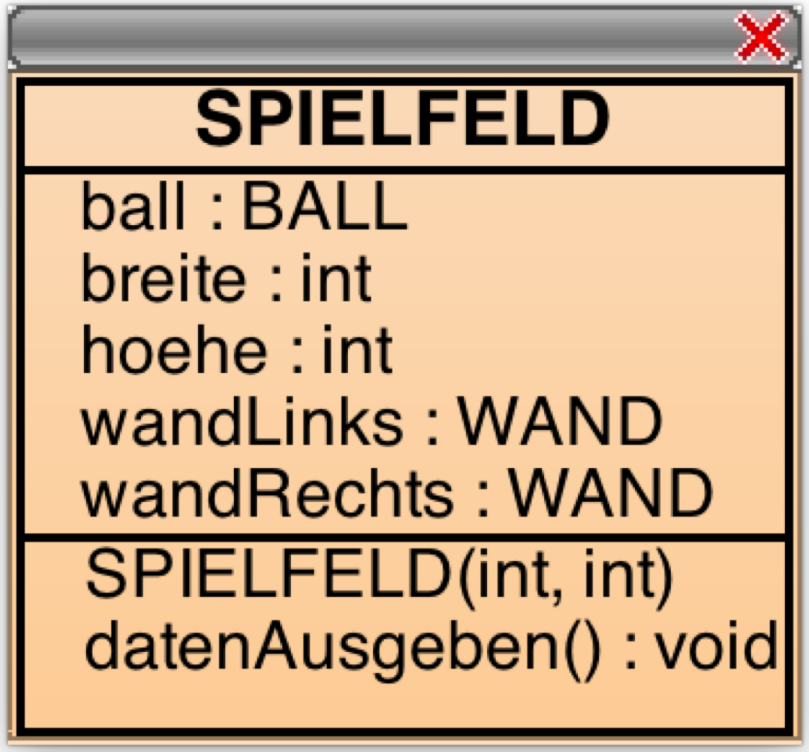
Verwende das BlueJ Projekt „alphaFormen_Ball_Lsg“.

Im Projekt soll es eine weitere Klasse SPIELFELD geben, die mit Objekten der Klassen BALL und WAND arbeitet:



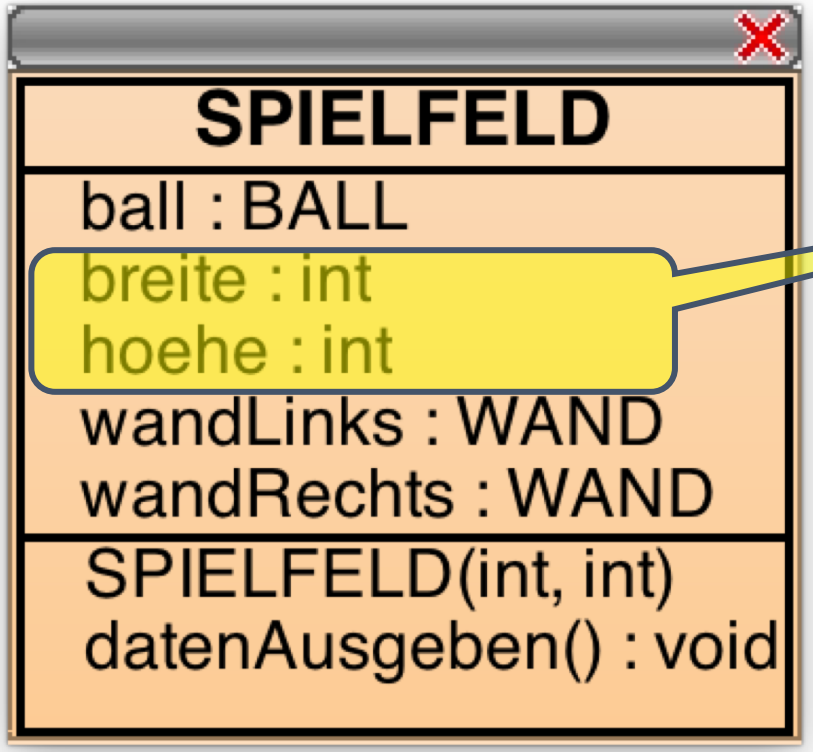
Eigene Klassen erstellen, Referenzattribute

Klassenkarte von SPIELFELD



Eigene Klassen erstellen, Referenzattribute

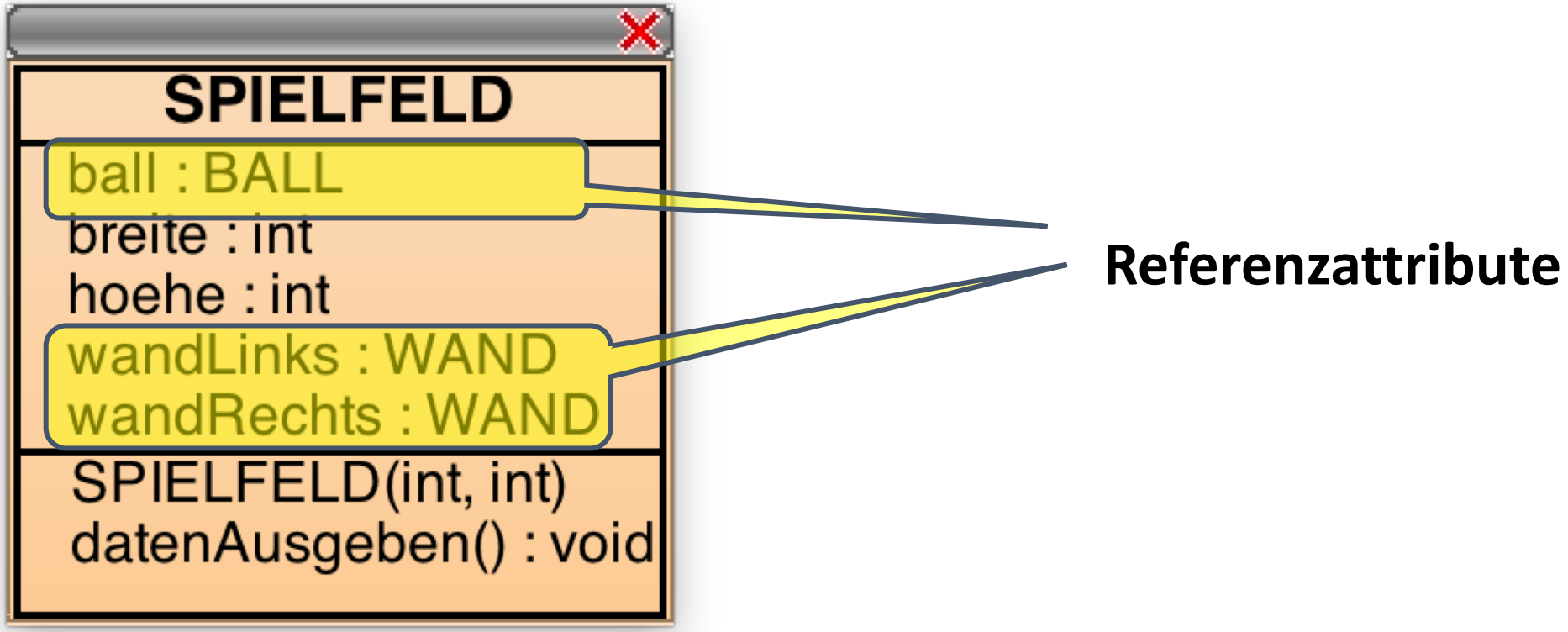
Klassenkarte von SPIELFELD



Einfache Attribute

Eigene Klassen erstellen, Referenzattribute

Klassenkarte von SPIELFELD



MERKE

Referenzattribute sind Attribute, die sich auf eine andere Klasse beziehen.

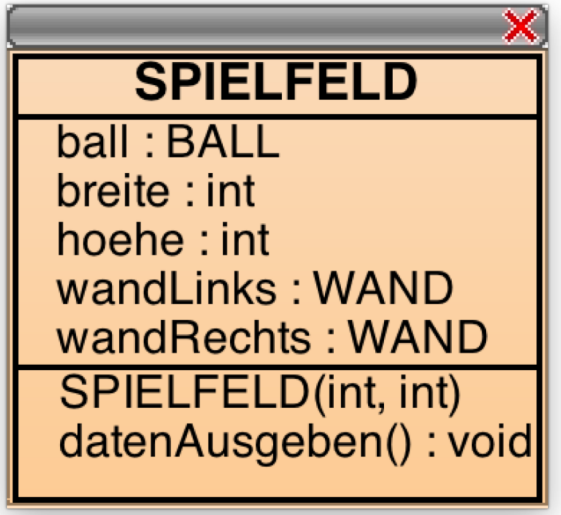
Im Gegensatz zu einfachen Attributen werden von einem Referenzattribut Objekte erzeugt.

Beispiel:

```
public class SPIELFELD {
    WAND wandLinks;                  //Referenzattribut
    ...
    public SPIELFELD(){
       wandLinks = new WAND(); //Erzeugen eines Objekts
    }
```

Übung 9:
Eigene Klassen erstellen, Referenzattribute

Implementiere die Klasse SPIELFELD mit den vorgegebenen Attributen.



Der Konstruktor hat zwei Übergabeparameter hoehe und breite
(in dieser Reihenfolge) .

Der Wert von hoehe soll dabei jeweils an wandLinks und wandRechts weitergegeben werden.Das Attribut breite der Objekte von WAND soll jeweils den Wert 10 haben.

Außerdem wird im Konstruktor von SPIELFELD ein Objekt ball der Klasse BALL erzeugt.

Der Aufruf new SPIELFELD (300, 500) erzeugt also die Objekte wandLinks und wandRechts, für die gilt wandLinks.hoehe=300, wandLinks.breite=10, wandRechts.hoehe=300 und wandRechts.breite=10 sowie ein Objekt ball der Klasse BALL.

Die Methode datenAusgeben() wird auf den nächsten Seiten erklärt.

Eigene Klassen erstellen; die Methode System.out.print(String text)

Anstelle der sondierenden Methoden sollen diesmal einige der Attributwerte in einem Textfenster ausgegeben werden.

BlueJ: Terminal Window - alphaFormen_Ball_Wand_Spielfeld

```
Hoehe= 300
Breite= 500
Hoehe WandLinks= 300
Breite WandRechts= 10
Hoehe WandLinks= 300
Breite WandRechts= 10
Radius Ball= 10
Farbe Ball= weiss
```

Can only enter input while your programming is run

Eigene Klassen erstellen; die Methode System.out.print(String text)

Dies erfolgt durch die Methode **print** oder auch **println.**
Bei println wird nach der Ausgabe eine neue Zeile begonnen, bei print nicht.
Übergabeparameter ist jeweils eine Zeichenkette.

Aufruf der Methode:

```
System.out.println("Diese Zeichenkette wird ausgegeben.");
```

Man kann mehrere Zeichenketten durch + miteinander verbinden.
Auch das Leerzeichen wird als Zeichenkette behandelt.

```
System.out.println("Ich" + " " + "programmiere.");
```

Dasselbe Ergebnis erhält man natürlich auch so:

```
System.out.println("Ich " + "programmiere.");
```

Eigene Klassen erstellen; die Methode System.out.print(String text)

Einfache Datentypen wie z.B. int , char oder boolean werden automatisch als Zeichenkette ausgegeben:

```
System.out.println(“Breite des Spielfeldes: “ + breite);
```

Dies liefert z.B. die Ausgabe: Breite des Spielfeldes: 500

Man kann dabei auch sondierende Methoden aufrufen:

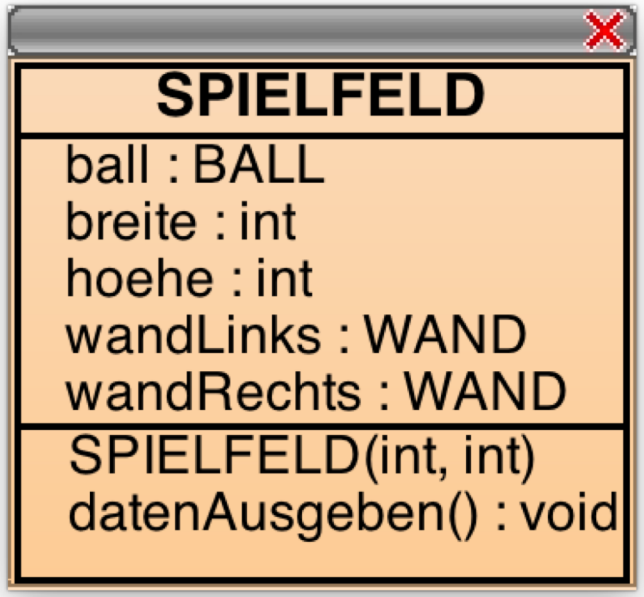
```
System.out.println(“Höhe wandLinks: “ + wandLinks.gibHoehe());
```

Übung 10:
Eigene Klassen erstellen, Textausgabe

Implementiere in der Klasse SPIELFELD nun die Methode

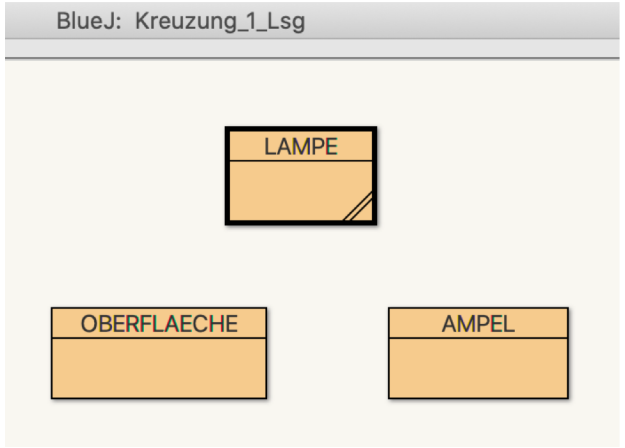
```
public void datenAusgeben(){
   System.out.println("Breite: " +breite);
   ...
}
```

Lass möglichst viele Attributwerte ausgeben.



Übung 11(*):
Steuerung von Ampeln an einer Kreuzung

Zur graphischen Darstellung und Steuerung einer Kreuzung mit Ampeln gibt es die Klassen OBERFLAECHE,LAMPE und AMPEL:



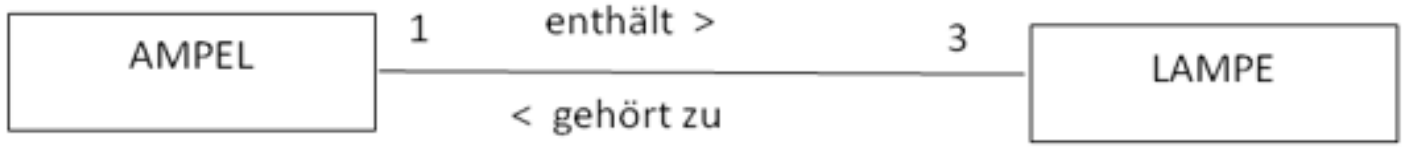
Die Klassen OBERFLAECHE und LAMPE sind vorgegeben.
Öffne dazu das BlueJ Projekt Kreuzung_1_Vorlage.

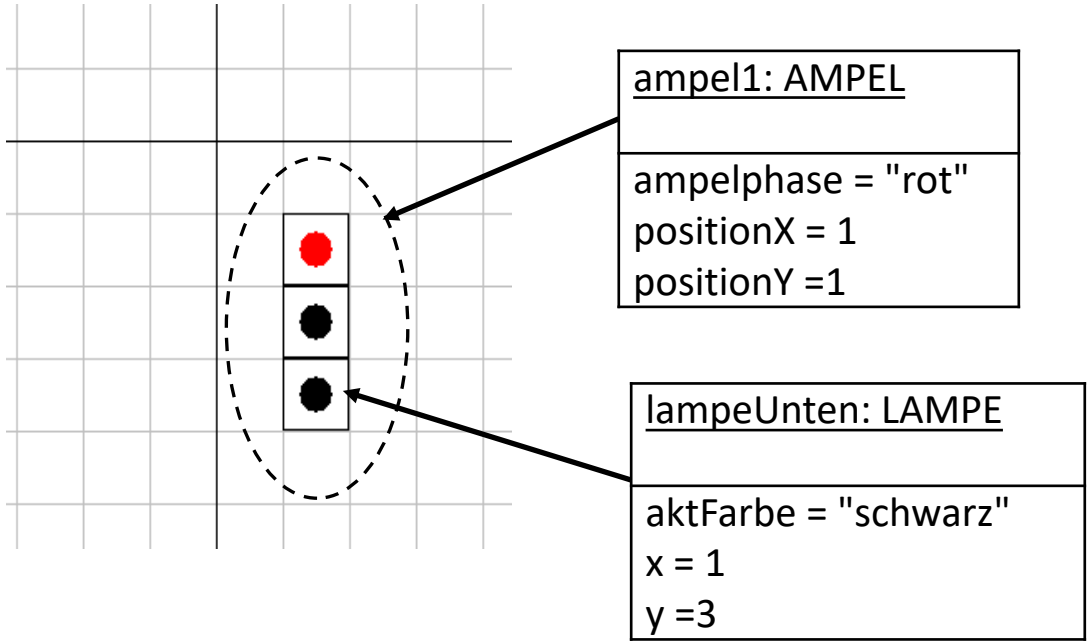
Ein Objekt der Klasse LAMPE erzeugt dabei ein Objekt von OBERFLAECHE, in dem graphisch ein Koordinatengitter angezeigt wird.
Im Gitter wird zur Darstellung der Lampe ein Quadrat mit einem Kreis gezeichnet.

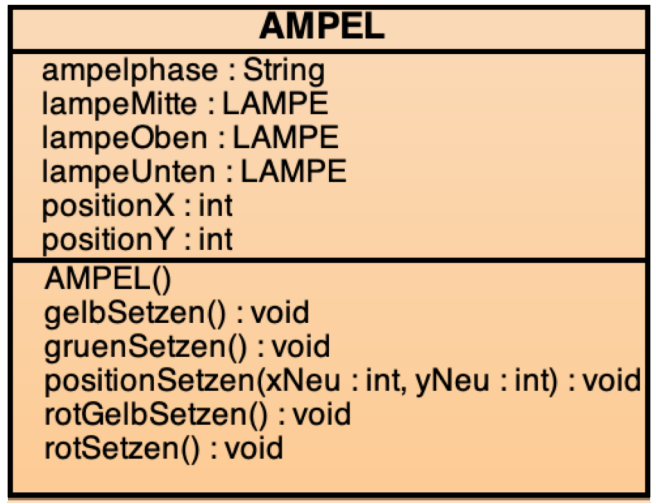
Erzeuge ein Objekt von LAMPE und untersuche damit die gegebenen Attribute und Methoden.

Übung 11(*):
Steuerung von Ampeln an einer Kreuzung

Klassendiagramm, Objekt – und Klassenkarten:

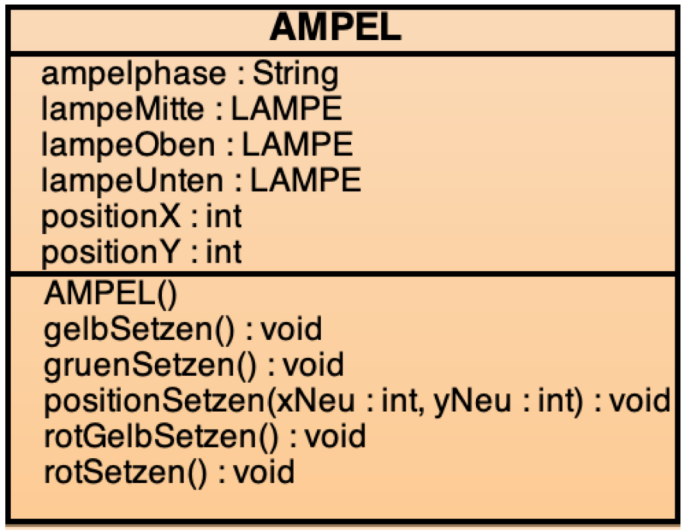






Übung 11(*):
Steuerung von Ampeln an einer Kreuzung

Ergänze den Quelltext der Klasse AMPEL.



Erläuterung der Methoden:

Konstruktor AMPEL():
Erzeugt die drei Objekte von LAMPE, ruft die Methoden rotSetzen() und positionSetzen(1,1) auf.

rotSetzen():
Setzt die Farbe von lampeOben auf "rot", lampeMitte und lampeUnten auf "schwarz" und das Attribut ampelphase auf den Wert "rot".

gelbSetzen(), gruenSetzen() und rotGelbSetzen() analog zu rotSetzen().

positionSetzen(int xNeu, int yNeu):
Setzt die Attribute positionX und positionY auf die Werte xNeu und yNeu und ruft von lampeOben, lampeMitte und lampeUnten die Methode PositionSetzen mit den passenden Übergabeparametern auf. Diese Übergabeparameter hängen von positionX und positionY ab.