

Objekt-Orientiertes Programmieren – Teil I

Das Spiel Ping-Pong

Eines der ersten Computerspiele war „Ping-Pong“. Zwei Rechtecke als Schläger, ein Kreis als Ball, der Monitor als Spielfeld ...

Wir wollen im Folgenden dieses uralte Spiel aus objektorientierter Sichtweise analysieren und es Stück für Stück programmieren.

Erinnern wir uns an die objektorientierte Analyse bei der Datenbank-Planung, so erinnern wir uns an **Klassen-Karten** und **Klassen-Diagramme**.



Wiederholung

In der heutigen Betrachtungsweise der Informatik besteht die Welt aus Objekten. Jedem Objekt liegt ein Bauplan zugrunde, nach dem es erstellt wurde. Diesen Bauplan nennt man **Klasse**. Eine Klasse legt fest, welche **Attribute** (Farbe, Breite, Höhe, ...) und welche **Methoden** (auf Klick reagieren, ...) solche Objekte haben sollen. Hat man nun mehrere Objekte einer Klasse erzeugt, so können sich diese durch ihre **Attribut-Werte** (Farbe "rot", Farbe "grün", ...) unterscheiden. Bei Methoden muss man manchmal Übergabe-Parameter mitgeben (neue Schriftgröße, neue Schriftfarbe, ...). Manche Methoden geben Antwort auf eine Frage (z.B. Frage nach Speicherplatz einer Datei über „Rechtsklick, Eigenschaften“).

Klassen-Karten

Der logische Aufbau einer Klasse wird in der Regel durch eine sog. **Klassen-Karte** veranschaulicht. Eine Klassen-Karte ist ein Rechteck, das von oben nach unten aus drei Bereichen besteht:

- Oben: Name der Klasse
- Mitte: Attribute und Daten-Typen
- Unten: Methoden und Übergabe-Parameter,

MENSCH	
geburtsdatum :	Datum
name :	Text
groesse :	Dezimalzahl
schlaeft :	Ja/Nein
...	
laecheln()	
schlafen(dauer : Zeit)	
trinken(was : Text , menge : Kommazahl)	
...	

Objekt-Karten

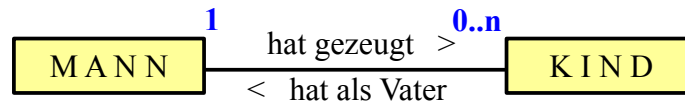
Verschiedene Objekte einer Klasse unterscheiden sich in der Regel in Ihren Attribut-Werten. Objekte werden durch eine **Objekt-Karte** veranschaulicht. Objekt-Karten haben im Vergleich zu Klassen-Karten abgerundete Ecken. Jede Objekt-Karte besteht aus zwei Bereichen:

- Oben: Name und Klasse des Objekts
- Unten: Attribute und deren Werte

susi : MENSCH	
geburtsdatum	= 27.04.1996
name	= "Susanne "
groesse	= 1.63
schlaeft	= Nein
...	

Klassen-Diagramme

Die Objekte stehen meist miteinander in irgendeiner Beziehung. Diese Beziehungen werden in einem sog. **Klassen-Diagramm** dargestellt. Dazu nimmt man nur den oberen Teil der Klassen-Karte (Klassen-Name) und zeichnet Verbindungslinien zu allen Klassen, zu denen diese Klasse eine Beziehung aufweist. Die Linien werden aussagekräftig beschriftet und die Beschriftung (nicht die Linie) zur besseren Lesbarkeit mit einem Richtungspfeil versehen. Oft gibt man auch noch **Kardinalitäten** an. Darunter versteht man Zahlen, die angeben, wie viele Objekte der einen Klasse in Beziehung zur anderen Klasse stehen können.



„Ein Mann kann kein Kind oder auch mehrere Kinder haben. Aber jedes Kind hat genau einen Mann als Vater.“

Objektorientierte Analyse – Teil I

- ? Welche Objekte kannst du im PingPong-Spiel erkennen?
In welche Klassen kannst du diese Objekte einteilen?
- ? Nenne Attribute dieser Klassen.
Von welchen Daten-Typen sind die Attribute?
- ? Nenne Methoden der Klassen.
Benötigen die Methoden Übergabe-Parameter? Welche Daten-Typen haben sie?
- 🚶 Fertige zusammen mit der Lehrkraft erweiterte Klassen-Karten.

Programmier-Umgebung BlueJ – Teil I

Damit du JAVA auf deinem Rechner programmieren kannst, benötigst du eigentlich nur vier Dinge:

1. Das **JAVA-Development-Kit** (JDK)
Lade die Datei (ca. 55 MB) von der Web-Site herunter und folge einfach den Installations-Anweisungen.
Starte anschließend deinen Rechner neu!
2. Eine Programmier-Umgebung wie z.B. **BlueJ**
Lade die Datei (ca. 6 MB) von der Web-Site herunter und folge ebenfalls einfach den Installations-Anleitungen.

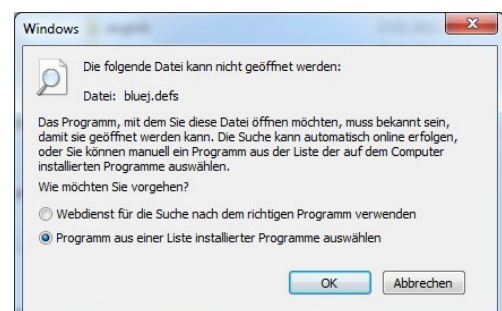


Sollte BlueJ nicht auf Deutsch starten, so musst du die Datei `bluej.defs` editieren.
Dazu gehst du folgendermaßen vor:

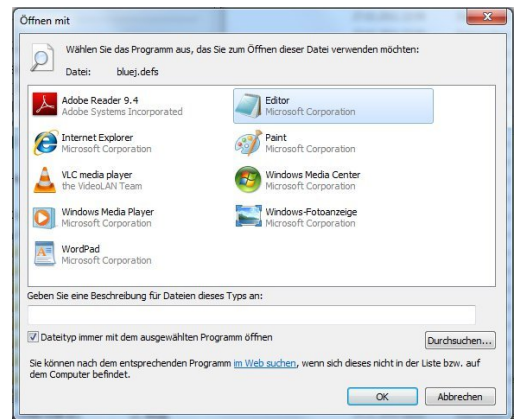
- Klicke doppelt auf die Datei `bluej.defs`.

Unter Windows findest du sie hier: „ `C: \ BlueJ \ lib \ bluej.defs` “.

Unter Linux findest du sie hier: „ `/ usr / share / bluej / bluej.defs` “.
- Wähle „Programm aus einer Liste installierter Programme auswählen“. Klicke OK.



- Wähle im folgenden Fenster „Editor“.
Klicke OK.



- Setze eine Raute # vor die Zeile mit der englischen Sprache.
- Entferne die Raute # vor der Zeile mit deutscher Sprache.
- Speichere die Datei und schließe den Editor.

```
#bluej. language=english
#bluej. language=afrikaans
#bluej. language=catalan
#bluej. language=chinese
#bluej. language=czech
#bluej. language=danish
#bluej. language=dutch
#bluej. language=french
bluej. language=german
```

3. Das Paket [engine-alpha_versionX.jar](#)

Lade die Datei (ca. 200 KB) von der Web-Site unter „Einzelne Bausteine“ herunter und speichere sie an einem beliebigen Ort auf deinem Computer.

Merke dir den Speicherort gut, du wirst ihn im nächsten Schritt noch einmal brauchen!

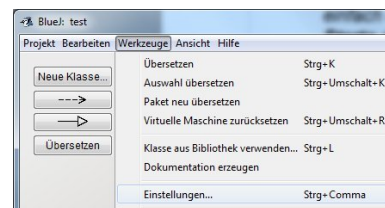
Lösche diese Datei nie!

Nun musst du dem Programm BlueJ die Datei `engine-alpha_versionX.jar` bekannt machen.

Dazu gehst du folgendermaßen vor:

WINDOWS:

- Hauptmenü „Werkzeuge“
- Untermenü „Einstellungen“

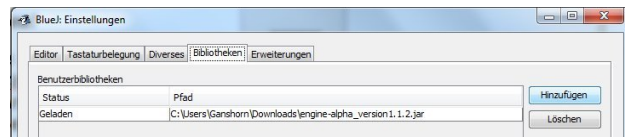


MAC:

- Menüleiste „BlueJ“
- Untermenü „Einstellungen“



- Reiter „Bibliotheken“
- Button „Hinzufügen“



- Wähle die vorher gespeicherte Datei `engine-alpha_versionX.jar` und klicke auf „öffnen“
- Schließe das „Einstellunen-Fenster“ durch Klick auf OK

4. Die BlueJ-Extension [Klassenkarte.jar](#)

Lade die Datei (ca. 60 KB) herunter und speichere sie im „extensions-Ordner“ deiner BlueJ-Installation.

Unter Windows findest du diesen Ordner unter „C: \ BlueJ \ lib \ extensions“.

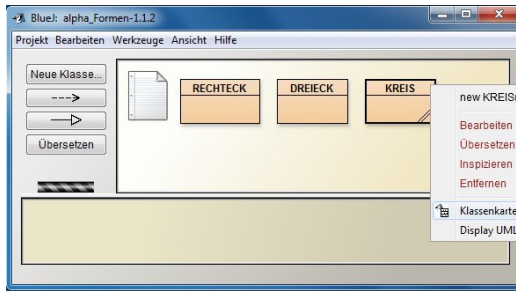
Unter Linux findest du diesen Ordner unter „/usr/share/bluej/extensions“.

Nach einem Neustart von BlueJ bist du nun endlich bereit zum Programmieren!



Öffne in BlueJ das Projekt [alpha_Formen](http://engine-alpha.org/html/unterricht/alpha_Formen.zip). Du siehst Klassen von geometrischen Figuren.
(Du kannst es unter http://engine-alpha.org/html/unterricht/alpha_Formen.zip herunterladen.)

1. Klicke mit rechts auf eine Klasse und wähle Klassen-Karte.
Erkunde die Methoden dieser Klasse.



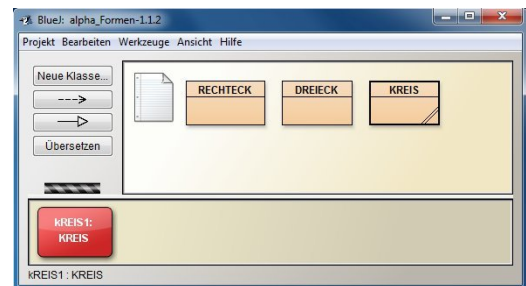
```

- M_x : int
- M_y : int
- farbe : String
- radius : int
- sichtbar : boolean
- symbol : KreisE

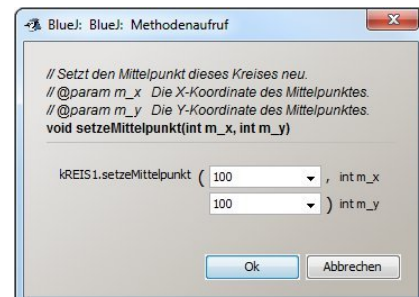
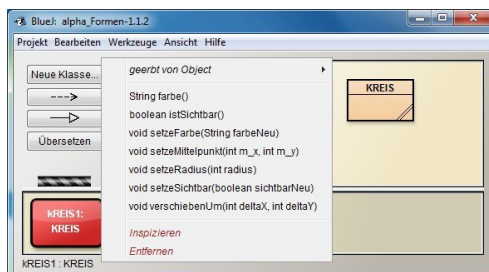
+ KREIS()
+ farbe() : String
+ istSichtbar() : boolean
+ setzeFarbe(farbeNeu : String) : void
+ setzeMittelpunkt(m_x : int, m_y : int) : void
+ setzeRadius(radius : int) : void
+ setzeSichtbar(sichtbarNeu : boolean) : void
+ verschiebenUm(deltaX : int, deltaY : int) : void

```

2. Klicke nochmals mit rechts auf eine Klasse und erzeuge ein Objekt mit **new ...**. Bestätige den Namensvorschlag. Es erscheint ein neues Fenster, in dem das Objekt grafisch veranschaulicht wird und im BlueJ-Fenster ist unten eine rote Objekt-Karte zu sehen.



3. Klicke nun mit Rechts auf die rote Objekt-Karte und wähle einen Methoden-Aufruf aus. Beobachte im anderen Fenster, wie sich die grafische Veranschaulichung verändert.



Die beiden Bilder veranschaulichen den Methoden-Aufruf:

`kreis1.setzeMittelpunkt(100,100)`

Du kannst die Punktnotation zu jedem Methoden-Aufruf direkt aus den Fenstern ablesen.

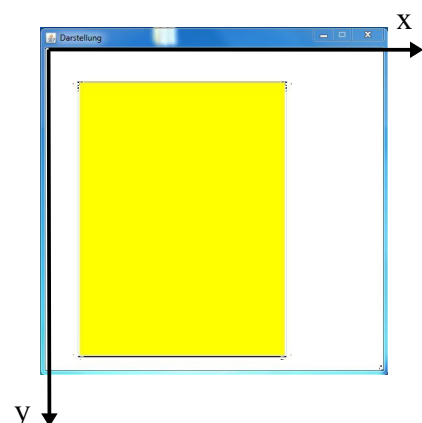
Beachte das Koordinaten-System:

Am Computer-Bildschirm ist die

y-Achse immer nach unten gerichtet

und der Ursprung ist links oben !

Die Maßeinheit ist Pixel (winzige Bildpunkte)





Versuche dein Spielfeld für PingPong ... ein Haus ... ein Auto ... zu zeichnen.

Beachte bei jedem (Rechts-)Klick, dass BlueJ dir die JAVA-Befehle zeigt, die du später in deinem Programm schreiben musst.

Beispiel: `wand = new RECHTECK();`

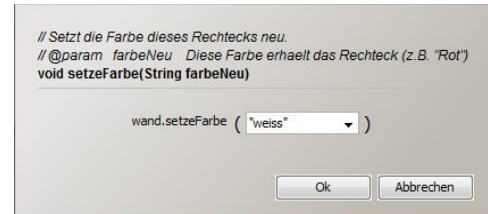


`wand.setzeFarbe("weiss");`

...

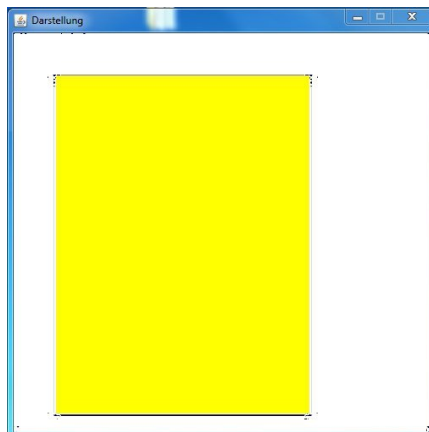
`dach = new DREIECK();`

...



Inspiziere einmal eines deiner Objekte ganz genau.

- Klicke dazu doppelt auf eine rote Objekt-Karte. Es öffnet sich der sog. **Objekt-Inspektor**. Er zeigt dir alle Attribute deines Objekts an.
- Lasse den Objekt-Inspektor eines Objekts geöffnet und rufe nun über einen Rechtsklick auf die Objekt-Karte eine Methode auf.
- Betrachte beim Bestätigen der Methode die Attribut-Werte im Objekt-Inspektor.

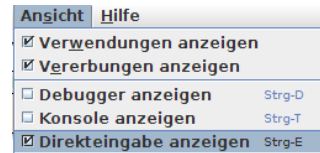


Teste später all deine selbst geschriebenen Klassen ausgiebig mit dem Objekt-Inspektor.

Nur so hast du eine Chance, versteckte Fehler zu finden !!!



Öffne nun in BlueJ die **Direkteingabe**. Klicke hierzu im Hauptmenü auf *Ansicht* und setze das Häkchen bei *Direkteingabe anzeigen*. Daraufhin erscheint im BlueJ-Fenster rechts unten ein weiterer Bereich.

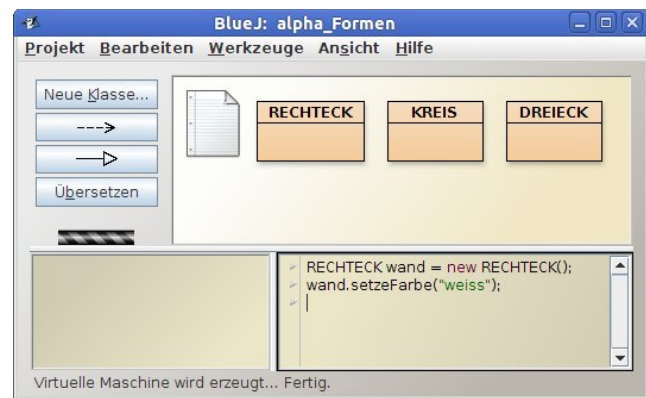


In der Direkteingabe kannst du nun die Befehle in Punktnotation ausprobieren – genau so wie du sie später in deinen Programmen schreiben wirst.

Schließe das Fenster mit deinem Haus (Auto). Zeichne es noch einmal indem du nun nur die Direkteingabe verwendest.

Dabei musst du beim Erzeugen eines neuen Objekts vor den Bezeichner (Namen) den Daten-Typ dieses Objekts angeben:

RECHTECK wand = new **RECHTECK**() ;



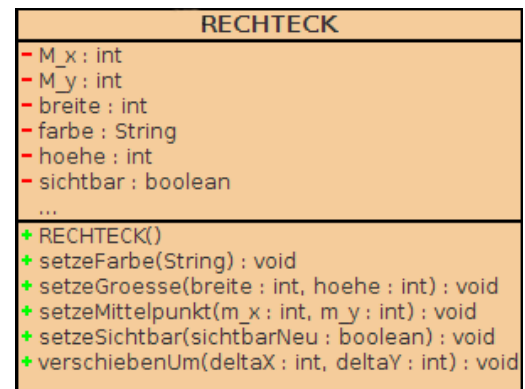
Solltest du dich nicht mehr an die Schreibweise der Methoden erinnern, so sieh dir einfach noch einmal die Klassen-Karte der entsprechenden Klasse an.

(Du erhältst sie durch Rechtsklick auf das Klassen-Symbol.)

Dabei steht *string* für *Text in Anführungszeichen*, z.B. **"gelb"**.

int steht für *eine ganze Zahl*, z.B. **100**.

(s. nächste Seite)



Erst wenn du dein Haus (Auto) mit der Direkteingabe erstellen kannst macht es Sinn, auf der nächsten Seite dieses Skripts weiter zu machen !!!

Die Programmier-Sprache JAVA – Daten-Typen und Klassen

JAVA ist eine weit verbreitete Programmier-Sprache. Sie ist verhältnismäßig einfach zu erlernen und unabhängig vom Betriebs-System (Windows, Linux, Mac OS).

Erste Daten-Typen in JAVA

Daten-Typ	Bedeutung	Beispiel	Anmerkung
<code>int</code>	ganze Zahl	<code>123 ; -321</code>	
<code>double</code>	Kommazahl	<code>13.24</code>	<u>Dezimalpunkt</u>
<code>char</code>	einzelnes Zeichen	<code>'a' ; '5'</code>	<u>Hochkommata</u>
<code>String</code>	mehrere Zeichen	<code>"Hallo"</code>	<u>Anführungszeichen</u>
<code>boolean</code>	Wahrheitswert	<code>true ; false</code>	

Erste Klasse in JAVA

BALL
<code>radius : int</code>
<code>farbe : String</code>

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

neue Klasse deklarieren

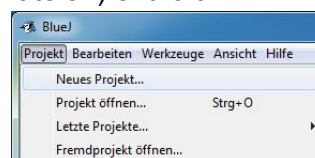
Attribute

deklarieren

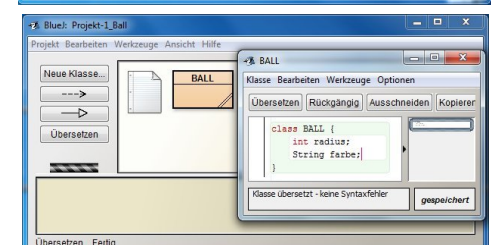
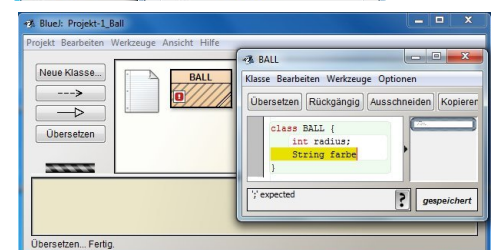
Programmier-Umgebung BlueJ – Teil II

Nun wollen wir unsere erste Klasse in JAVA selbst programmieren. In der Programmier-Umgebung BlueJ werden Klassen immer in einem sog. **Projekt** gespeichert. Ein Projekt ist im Prinzip ein Ordner, der eine oder mehrere Klassen (und andere Dateien) enthält.

- Erstelle ein neues Projekt:
Projekt / Neues Projekt ...
Pfad und Projektname eingeben ...
Button „Erzeugen“
- Erzeuge in dem Projekt eine neue Klasse:
Button „Neue Klasse“ ...
Klassennamen eingeben ...
Button „ok“
- Öffne den Editor:
Doppelklick auf die neu entstandene Klassen-Karte ...
Lösche den vorgegebenen JAVA-Code ...
Gib deinen eigenen Java-Code ein
- Übersetze deinen Java-Code:
Button „Übersetzen“ ...
Wenn Fehler auftreten, so lies die Fehlermeldung ...
Meist handelt es sich um Tippfehler ...
Beseitige den Fehler ...
Übersetze nochmal ...
Bis die Meldung „keine Syntaxfehler“ erscheint



Dateiname: Erzeugen
Dateityp: Abbrechen





1. Erstelle die oben erwähnte Klasse *BALL*.
Schreibe den entsprechenden JAVA-Code.
Übersetze die Klasse fehlerfrei.
2. Erzeuge ein erstes Objekt deiner Klasse *BALL*.
(Rechtsklick auf die Klassen-Karte ... *new* ...).
Inspiziere die Attribut-Werte deines ersten Objekts.
(Doppelklick auf die rote Objekt-Karte).

Was stört dich beim Anblick des Objekt-Inspektors?

Mit dieser Frage werden wir uns im nächsten Kapitel beschäftigen!



Du hast nun gesehen, wie man mit der Programmier-Umgebung BlueJ umgeht. Du kannst:

- Projekte Anlegen
- Klassen anlegen
- JAVA-Code schreiben und übersetzen
- Objekte erzeugen und inspizieren

Genau so solltest du alle weiteren Programmier-Arbeiten erledigen!

Alles muss inspiziert und ausführlich getestet werden ...

TIPP: **Kapselungs-Prinzip**

- Klassen erhalten immer den **Modifikator *public***.
Dadurch sind sie sichtbar für andere Klassen.
(Das ist später wichtig, damit man von überall aus auf diese Klasse zugreifen kann.)
- Attribute erhalten immer den **Modifikator *private***.
Dadurch sind die Attribute von Objekten nach außen hin unsichtbar und damit unantastbar.
Nur durch Methoden-Aufrufe kann ein Objekt selbst kontrolliert seine Attribut-Werte verändern.

Beispiel:

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

neue Klasse deklarieren
Attribute deklarieren



Deklarieren von Attributen in JAVA

Modifikator	Datentyp	AttributName	Strichpunkt
<i>private</i>	<i>String</i>	<i>farbe</i>	<i>;</i>
<i>private</i>	<i>int</i>	<i>radius</i>	<i>;</i>



Erweitere deine Klasse *BALL* um die Modifikatoren und übersetze neu.

TIPP: Übersetzen

- Das Übersetzen eines Programms wird auch **kompilieren** (engl.: to compile) genannt.
- *Was geschieht da?*
Dein Java-Code wird in sog. Byte-Code (lauter Einsen und Nullen) übersetzt, damit die *Java-Virtual-Machine* (vereinfacht gesagt: ein Prozessor) die Befehle verstehen kann.
Die Programmier-Sprache JAVA ist so eine Art Kompromiss zwischen der Alltags-Sprache, die du sprichst und den Einsen und Nullen, die der Computer versteht.

Die Programmier-Sprache JAVA – Methoden

Du erinnerst dich !? Der `radius` deines ersten Kreis-Objekts war `0` und seine `farbe` `"null"` !?

Das liegt daran, dass wir bisher zwar Attribute **deklariert** (Variablen „bestellt“) haben. Aber wir haben sie nicht **initialisiert**, d.h. ihnen Werte zugewiesen.

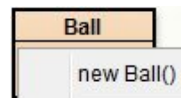


Unterscheide die Begriffe

- Variablen **deklarieren** (beantragen, z.B. `private int radius`)
- Variablen **initialisieren** (ihnen Werte zuweisen, z.B. `radius = 20`)

Die Konstruktor-Methode

Das anfängliche Initialisieren („einrichten“) eines Objekts bei seiner Erzeugung erledigt eine besondere Methode, der **Konstruktor**. Ihn rufen wir auch auf, wenn wir mit `new ...` ein neues Objekt erzeugen.



Einen Standard-Konstruktor, der das Objekt erzeugt, ohne die Attribute zu initialisieren, erstellt JAVA unsichtbar im Hintergrund automatisch. Sonst hätten wir unser erstes Objekt der Klasse *BALL* nicht erzeugen können.

Beispiel:

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

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

Klasse Ball deklarieren

*Attribute
deklarieren*

*Konstruktor-Methode:
Attribute
initialisieren*



Erstelle in deiner Klasse *BALL* einen Konstruktor:

Das Attribut `radius` soll für alle Objekte mit dem Wert `10` initialisiert werden.
Das Attribut `farbe` soll für alle Objekte mit dem Wert `"weiss"` initialisiert werden.

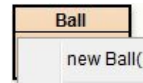
Übersetze anschließend die Klasse, bis keine Fehler mehr auftreten.

Erzeuge ein Objekt deiner Klasse *BALL* und inspiziere es im Objekt-Inspektor.

Der Sinn des Schlüsselworts `this` wird im nächsten Teil-Abschnitt geklärt.



- Der **Konstruktor** ist eine Methode, die aufgerufen wird, wenn ein neues Objekt erzeugt wird.
Im Konstruktor werden alle Attribute dieses Objekts mit Werten belegt.
- Der Name der Konstruktor-Methode muss genauso lauten, wie der Name der Klasse selbst.
- Du hast schon mehrfach ein Objekt mit `new ...` erzeugt.
Dabei wird der Konstruktor der entsprechenden Klasse aufgerufen.
- Nachdem alle Attribute deiner Klasse deklariert wurden, solltest du den Attributen immer das Schlüsselwort `this` voranstellen!**



Übergabe-Parameter

Bei unserem bisherigen Konstruktor werden die Attribut-Werte aller erzeugten Objekte immer exakt gleich initialisiert.

Manchmal möchte man aber zwei verschiedenen Objekten gleich bei der Erzeugung unterschiedliche Attribut-Werte zuweisen. Dies kann man erreichen, indem man der Konstruktor-Methode **Parameter** übergibt.

Parameter sind kurzlebige Variablen, die nach dem Beenden der Methode wieder gelöscht werden.

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

Konstruktor mit Parametern

Bei Parametern muss Daten-Typ und Parameter-Name angegeben werden



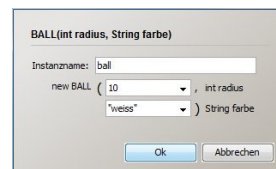
- Bei **Übergabe-Parametern** muss in den runden Klammern der Methode erst der Daten-Typ und danach der Name des Parameters angegeben werden.
- Bei **Wert-Zuweisungen** steht immer links das Attribut und rechts der neue Wert.
Bspl.: `Attribut = Wert`
- Das Schlüsselwort **this** kann dazu verwendet werden, um zwischen Attribut und Parameter zu unterscheiden, falls beide denselben Namen haben.
Bspl: `radius` Parameter (neuer Attribut-Wert)
 `this.radius` Attribut des späteren Objekts
 `this.radius = radius` (Attribut bekommt neuen Wert)

Nachdem man nun einen Konstruktor mit

`new BALL(10, "weiss")`

aufgerufen hat, wird dadurch

`this.radius = radius` zu `this.radius = 10` und
 `this.farbe = farbe` zu `this.farbe = "weiss"` .



TIPP: Überladen von Methoden

- Es ist zulässig, mehrere Konstruktor-Methoden zu deklarieren.
- Diese müssen sich aber unbedingt in der Anzahl der Parameter oder deren Daten-Typen unterscheiden. (sonst gibt es Fehler-Meldungen beim Übersetzen!)
- Hat man eine Methode auf mehrere Arten implementiert, so nennt man das **Überladen** einer Methode.



Erstelle in deiner Klasse *BALL* drei weitere Konstruktoren:

1. Der zweite Konstruktor soll zwei Übergabe-Parameter haben. Der Aufrufer des Konstruktors soll also beide Werte übergeben müssen und kann so entscheiden wie der Ball aussehen soll.
2. Der dritte Konstruktor soll nur den *radius* durch einen Übergabe-Parameter belegen, die *farbe* soll immer mit einem festen Wert "weiss" initialisiert werden.
3. Der vierte Konstruktor soll nur die *farbe* durch einen Übergabe-Parameter belegen, der *radius* soll immer mit einem festen Wert 10 initialisiert werden.



Auftrag zum selbständigen Anwenden:

1. Deklare in deiner Klasse *BALL* zwei weitere Attribute x und y (Koordinaten für den Ort).
2. Welche Daten-Typen sind dafür geeignet? (Tipp: am Bildschirm denkt man in Pixeln)
3. Ändere deine Konstruktoren so ab, dass sie auch diese neuen Attribute berücksichtigen.
4. Schreibe einen weiteren Konstruktor, der alle vier Attribute durch Übergabe-Parameter initialisiert.

Erzeuge mit jedem Konstruktor mindestens ein Objekt und inspiziere es im Objekt-Inspektor!

Sind alle Attribut-Werte so gesetzt, wie du es dir vorgestellt hast?

Methoden mit Rückgabe

Damit später unser Ball-Objekt an den Schläger-Objekten reflektiert werden kann, muss es mit diesen kommunizieren können. Ganz konkret muss der Ball bei jedem „kleinen Schritt“ die Schläger fragen, wo sie gerade sind. Genauso gut könnten die Schläger-Objekte den Ball fragen, wo er gerade ist.

Damit dies möglich ist, brauchen unsere Klassen Methoden, die eine Antwort geben. Dies erreicht man mit Hilfe des Schlüsselworts *return*. Den Daten-Typ der Antwort gibt man direkt vor dem Namen der Methode an.

Bspl.:

```
public int nenneRadius() {
    return this.radius;
}
```

Daten-Typ der Rückgabe

Rückgabe-Anweisung

Damit erhält man in der Klasse *BALL* eine Methode (unterhalb der Konstruktoren), welche eine ganze Zahl als Antwort gibt, nämlich den Wert des Attributs *radius* des Ball-Objekts. Führe dir an dieser Stelle noch einmal in Erinnerung, dass mit *this.radius* das Attribut *radius* des Objekts angesprochen wird, welches du oberhalb des Konstruktors deklariert hast.



- Bei einer **Methode mit Rückgabe** muss vor dem Methoden-Namen der Daten-Typ des Rückgabe-Werts stehen.
- Die eigentliche Rückgabe erfolgt als letzte Anweisung in der Methode mit dem Schlüsselwort *return*.



TIPP

Möchtest du in der Direkteingabe auch die Rückgabe „sehen“, so musst du am Ende des Methoden-Aufrufs den Strichpunkt weglassen!



Implementiere in deiner Klasse *BALL* folgende Methoden:

1. *nenneRadius()*
2. *nenneFarbe()*
3. *nenneX()*
4. *nenneY()*

Teste mit dem Objekt-Inspektor, ob die Antworten auch den Attribut-Werten entsprechen!



Auftrag zum selbständigen Anwenden:

Schreibe nun analog zur Klasse *BALL* eine Klasse *SCHLAEGER*.

1. Ein Schläger soll die Attribute *x*, *y*, *breite*, *hoehe* und *farbe* haben.
2. Es soll verschiedene Konstruktoren geben. Solche ohne Übergabe-Parameter, aber auch welche mit Übergabe-Parametern.
3. Zu jedem Attribut soll es eine „nenne-Methode“ geben, welche den entsprechenden Attribut-Wert zurück gibt.

Übersetze deine Klasse nach jedem kleinen Teilschritt um eventuelle Fehler sofort zu bemerken!

Teste ausgiebig mit dem Objekt-Inspektor um sicher zu stellen, dass deine Objekte auch so funktionieren, wie du es geplant hast.

Methoden ohne Rückgabe

Nachdem man ein Objekt erzeugt hat, möchte man oft nachträglich die Attribut-Werte verändern. Du kennst das z.B. aus der Textverarbeitung: *zeichen3.setzeSchriftart("Arial")*

Solche Methoden haben allerdings keine Rückgabe. Statt eines Daten-Typs für eine Rückgabe verwendet man das Schlüsselwort **void** (engl.: Fehlstelle, Leere), was andeuten soll, dass hier keine Rückgabe erfolgt. Entsprechend gibt es hier auch KEIN *return*. Der neue Wert für das Attribut wird dann genau wie beim Konstruktor gesetzt.

Bspl.:

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

KEINE Rückgabe



- **Methoden ohne Rückgabe** haben vor dem Methoden-Namen das Schlüsselwort **void**.
- Die einzige Ausnahme bildet der Konstruktor. Hier fehlt dieses Schlüsselwort.



Implementiere in deinen Klassen *BALL* und *SCHLAEGER* für jedes Attribut eine „setze-Methode“.

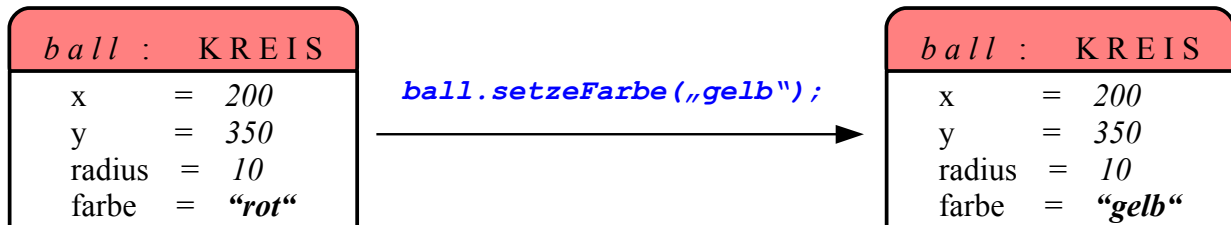
1. *BALL*: *setzeRadius(...)*, *setzeFarbe(...)*, *setzeX(...)*, *setzeY(...)*
2. *SCHLAEGER*: *setzeBreite(...)*, *setzeHoehe(...)*, *setzeX(...)*, *setzeY(...)*, *setzeFarbe(...)*



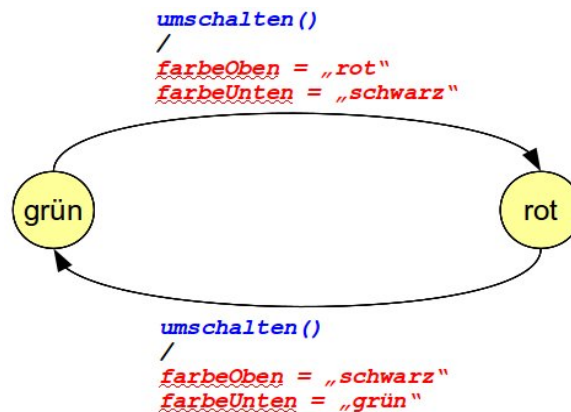
Zeichne ausführliche Klassen-Karten zu deinen beiden ersten selbst geschriebenen Klassen.
Gib auch alle Daten-Typen mit an.

Methoden als Zustands-Übergänge von Objekten

Durch die exakte Angabe der Werte von allen Attributen eines Objekts ist ein klar definierter **Zustand** dieses Objekts definiert. Ändert man einen oder mehrere Attribut-Werte, so befindet sich das Objekt in einem anderen Zustand. Man sagt auch, das Objekt ist von einem Zustand in einen anderen übergegangen. Solche **Zustands-Übergänge** werden in der Regel durch Aufrufe von Methoden des Objekts ausgelöst.



Wenn es keine Zweifel gibt, von welchem Objekt man spricht, so kann man die Darstellungsform vereinfachen. Ein Zustand wird dann nur durch einen Kreis dargestellt, der einen treffenden Namen trägt. Bei den Übergängen werden manchmal zu den auslösenden Ereignissen (z.B. Methodenaufrufen) auch noch die ausgelösten Aktionen (z.B. Änderung der Attribut-Werte) angegeben. Man spricht dann von einem **Zustands-Übergangs-Diagramm**.



Zustands-Übergangs-Diagramm einer Fußgänger-Ampel

Zustands-Übergangs-Diagramme werden oft verwendet, um kompliziertes Verhalten eines Objekts zu modellieren, bevor man dies programmieren kann. Wir werden dies später bei der Bewegung der Schläger und des Balls sehr ausgiebig verwenden. An dieser Stelle sollst du nur das Prinzip verstehen.



- Ein **Zustand** beschreibt einen exakt festgelegten Satz von Attribut-Werten eines Objekts
- Bei einem **Zustands-Übergang** wird mindestens ein Attribut-Wert verändert.
- Zustände und Übergänge werden in einem **Zustands-Übergangs-Diagramm** veranschaulicht.
- Zustands-Übergangs-Diagramme dienen der Beschreibung von komplexen Vorgängen.



Erstelle zur Übung ein Zustands-Übergangs-Diagramm einer Auto-Ampel (rot-grün-gelb).

Dokumentation ist wichtig

Wenn mehrere Personen an einem gemeinsamen Projekt arbeiten, aber auch wenn du alleine über einen längeren Zeitraum an einem Projekt arbeitest, dann sollten alle Programmierer den Code auch **dokumentieren**. JAVA stellt hierfür sog. **JAVA-DOC-Kommentare** zur Verfügung.

Aus diesen Kommentaren kann mit nur einem Knopfdruck eine **HTML-Seite** (Web-Seite) erstellt werden, in der du alles nachlesen kannst, was du über eine Klasse, ein bestimmtes Attribut oder eine Methode wissen möchtest.

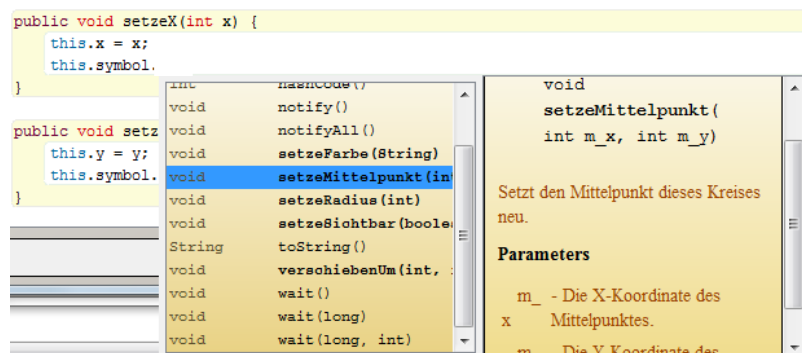
Method Summary	
void	setzeFarbe (java.lang.String farbeNeu) Setzt die Farbe dieses Kreises neu.
void	setzeMittelpunkt (int m_x, int m_y) Setzt den Mittelpunkt dieses Kreises neu.
void	setzeRadius (int radius) Setzt den Radius dieses Kreises neu.
void	setzeSichtbar (boolean sichtbarNeu) Setzt, ob dieser Kreis sichtbar sein soll.
void	verschiebenUm (int deltaX, int deltaY) Verschiebt diesen Kreis um eine Verschiebung - angegeben durch ein "Delta X" und "Delta Y".

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

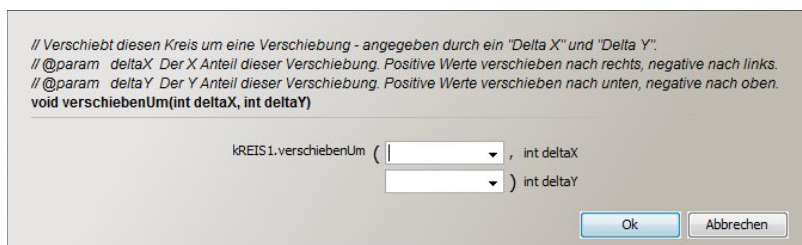
Constructor Detail
KREIS public KREIS() Konstruktor der Klasse KREIS. Erstellt einen neuen Kreis.

Method Detail
setzeFarbe public void setzeFarbe(java.lang.String farbeNeu) Setzt die Farbe dieses Kreises neu. Parameters: farbeNeu - Diese Farbe erhaelt der Kreis (z.B. "Rot")

Diese Kommentare erscheinen auch, wenn du im Editor von BlueJ mit der Tastenkombination „**Strg + Leertaste**“ den **Code vervollständigen** möchtest ...



... oder wenn du grafisch die **Methode eines Objekts aufrufen** möchtest, mit einem Rechtsklick auf eine Klassenkarte.

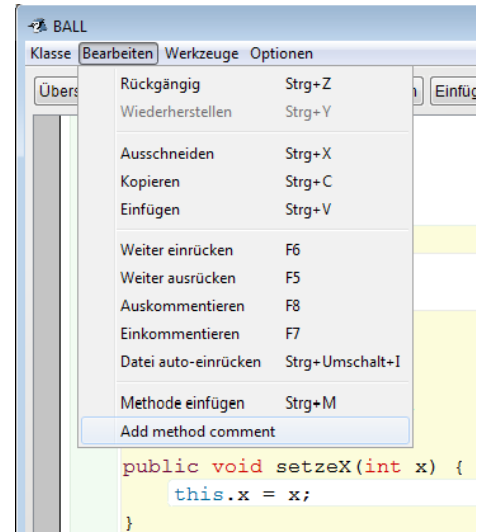


- Stelle im Editor den Cursor vor das *public* der Methode *setzeX(...)*.

Wähle nun im Hauptmenü des Editors

Bearbeiten / add method comment

```
/**
 * Method setzeX
 *
 * @param x A parameter
 */
public void setzeX(int x) {
    this.x = x;
}
```



- Ersetze den Kommentar-Text durch deinen eigenen

```
/**
 * Methode zum Setzen der x-Koordinate des Balls
 *
 * @param x Die neue x-Koordinate des Balls
 */
public void setzeX(int x) {
    this.x = x;
}
```

- Verfahre ebenso mit der Methode *nenneX()*.

```
/**
 * Methode zum Nennen der x-Koordinate des Balls
 *
 * @return x x-Koordinate des Balls
 */
public int nenneX() {
    return this.x;
}
```



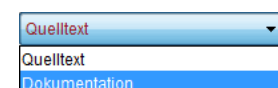
- Ein **JAVA-DOC-Kommentar** beginnt mit den Zeichen **/****
- Jede weitere Zeile muss mit ***** beginnen.
- Die letzte Kommentar-Zeile beginnt mit ***/**
- Nach einem **@param** kann man Informationen zu Übergabe-Parametern geben.
- Nach einem **@return** kann man Informationen zu Rückgabe-Werten geben.



Kommentiere nun alle *nenne*- und *setze*-Methoden in deinen Klassen BALL und SCHLAEGER.

Betrachte die dein Ergebnis, indem du rechts oben im Editor-Fenster auf Dokumentations-Ansicht wechselst.


Benutze die Hyperlinks und lies deine eigene Dokumentation ...



Erzeuge Objekte deiner Klassen und rufe durch Rechtsklick auf die Objekt-Karten *nenne*- und *setze*-Methoden auf. Beobachte deine eigene Dokumentation und beurteile, ob sie einem Dritten helfen würde, deine Klasse und ihre Methoden zu verstehen.


Fülle auch den JAVA-DOC-Kommentar oberhalb der Klasse aus und beobachte die Wirkung ...


Übungsaufgaben um das Gelernte zu festigen

-  Nimm dir etwas Zeit und versuche alle neu gelernten Fachbegriffe in eigenen Worten zu erklären:


deklarieren, initialisieren, Wert-Zuweisung, Konstruktor, Parameter, Überladen einer Methoden, Methoden mit Rückgabe, Methoden ohne Rückgabe, Java-Doc-Kommentare

Lege dir hierzu ein „Vokabelheft“ an ... ergänze bei Bedarf eigene Beispiele ...

-  Verfasse zusammen mit der Lehrkraft noch einmal eine Klassen-Karte deiner Klasse *BALL*, die nun auch alle Methoden enthält.

-  Zeichne die Klassen-Karte einer Klasse *KATZE*.

Eine Katze hat einen Namen und ein Alter. Eine neu geborene Katze ist 0 Jahre alt. Ihren Namen soll man im Konstruktor übergeben können. Die Katze soll Methoden haben, um über ihre Attribut-Werte Auskunft geben zu können. Außerdem soll es je eine Methode geben, um das Alter zu setzen und den Namen zu setzen.

-  Implementiere die Klasse *KATZE* mit BlueJ in JAVA.


Denke dabei auch an die JAVA-DOC-Kommentare ...

-  Zeichne die Klassen-Karte einer Klasse *HUND*.

Ein Hund hat einen Namen, ein Alter und einen Gemütszustand. Ein neu geborener Hund ist 0 Jahre alt und gut gelaunt. Den Namen soll man im Konstruktor übergeben können. Der Hund soll Methoden haben, um über seine Attribut-Werte Auskunft geben zu können. Es soll auch jeder Attribut-Wert neu gesetzt werden können. Außerdem soll es eine Methode *bellen()* geben mit der Rückgabe „wauwauwau“ und eine Methode *altern(...)*, die das Alter um Eins erhöht.

-  Implementiere die Klasse *HUND* mit BlueJ in JAVA.

Denke dabei auch an die JAVA-DOC-Kommentare ...

-  Zeichne die Klassen-Karte einer Klasse *AUTO*.

Ein Auto ist von einer bestimmten Marke und entweder gestartet oder nicht. Das Auto hat eine gewisse Geschwindigkeit. Alle Attribut-Werte sollen genannt und gesetzt werden können. Es soll eine Methode *beschleunigen()* geben, bei der das Auto um 10 km/h schneller wird. Einer weiteren Methode *beschleunigen(...)* soll man einen Wert übergeben können, um den das Auto schneller wird. Es soll auch eine Methode *bremsen* geben, bei der das Auto um 10 km/h langsamer wird.

-  Implementiere die Klasse *AUTO* mit BlueJ in JAVA.

Denke dabei auch an die JAVA-DOC-Kommentare ...

Selbstkontrolle: Hast du auch alles verstanden?

... dann kannst du bestimmt die folgenden Fragen ohne Probleme beantworten.

Wenn nicht, dann arbeite das entsprechende Teil-Kapitel noch einmal durch ...

- ❓ Was wird in Klassen-Karten dargestellt?
Was wird in Klassen-Diagrammen dargestellt?
Worin unterscheiden sich Klassen-Karten und Objekt-Karten?
Was versteht man unter dem Begriff Kardinalität?
- ❓ Erkläre deutlich den Unterschied zwischen den Begriffen *Attribut*, *Attribut-Wert* und *Parameter*.
- ❓ Wozu dient die Direkteingabe?
Wie kann man sie sichtbar machen?
- ❓ Mit welchem Schlüsselwort erzeugt man in JAVA ein neues Objekt?
Welche Methode der Klasse wird dabei aufgerufen?
- ❓ Erkläre deutlich den Unterschied zwischen den Begriffen *deklarieren* und *initialisieren*.
- ❓ Nenne mindestens 5 Daten-Typen der Programmier-Sprache JAVA und gib ihre Abkürzungen an.
- ❓ Was versteht man unter *Modifikatoren* und welche kennst du?
Was bewirken diese Modifikatoren?
- ❓ Erkläre die Bedeutung des Schlüsselworts *this*.
Was kann man damit unterscheiden?
- ❓ Wie gibt man in einer Methode Übergabe-Parameter an?
- ❓ Erkläre den Aufbau einer Methode mit Rückgabe-Wert.
Erkläre den Aufbau einer Methode ohne Rückgabe-Wert.
Was weißt du über die Konstruktor-Methode?
- ❓ Was versteht man unter *Überladen von Methoden*?
- ❓ Wie kann man in BlueJ eine Klassenkarte einer Klasse anzeigen lassen?
Wie kann man in BlueJ die Attribut-Werte eines Objekts genauer betrachten?
Wie ruft man in BlueJ eine Methode eines Objekts auf?
Was bedeutet der Begriff *übersetzen* oder *kompilieren*?
- ❓ Was versteht man unter dem Zustand eines Objekts?
Was geschieht bei einem Zustands-Übergang?
Wozu dienen Zustands-Übergangs-Diagramme?
- ❓ Wozu sind Java-Doc-Kommentare gut?
Wie sehen sie aus?
Was können sie alles enthalten?

Zusammenfassung

Dies war der erste Schritt auf dem Weg zu deinem eigenen PingPong-Spiel.

Du hast nun die erste einfache **logische Struktur** der Klassen *BALL* und *SCHLAEGER* modelliert und in der Programmiersprache JAVA implementiert.

Du solltest nun folgendes geschafft haben:

BALL	SCHLAEGER
- farbe : String - radius : int - x : int - y : int	- breite : int - farbe : String - hoehe : int - x : int - y : int
+ BALL(radius : int) + BALL(farbe : String) + BALL(radius : int, farbe : String) + BALL(x : int, y : int) + BALL(x : int, y : int, radius : int, farbe : String) + BALL() + nenneFarbe() : String + nenneRadius() : int + nenneX() : int + nenneY() : int + setzeFarbe(farbe : String) : void + setzeMittelpunkt(x : int, y : int) : void + setzeRadius(radius : int) : void + setzeX(x : int) : void + setzeY(y : int) : void	+ SCHLAEGER(x : int, y : int, farbe : String) + SCHLAEGER(x : int, y : int) + SCHLAEGER(x : int, y : int, breite : int, hoehe : int, farbe : String) + SCHLAEGER() + nenneBreite() : int + nenneFarbe() : String + nenneHoehe() : int + nenneX() : int + nenneY() : int + setzeBreite(breite : int) : void + setzeFarbe(farbe : String) : void + setzeHoehe(hoehe : int) : void + setzeMittelpunkt(x : int, y : int) : void + setzeX(x : int) : void + setzeY(y : int) : void

Die **roten Minus-Zeichen** auf den Klassen-Karten bedeuten *private*, die **grünen Plus-Zeichen** bedeuten *public*.

Sollte in deiner eigenen Arbeit irgend etwas fehlen, so hole es jetzt nach!

Hast du irgend etwas mehr programmiert, so wird es nicht stören.

Ausblick

Im nächsten Kapitel werden wir

- den logischen Strukturen nun eine **grafische Darstellung** folgen lassen,
- **Fallunterscheidungen** kennen lernen

Dabei wird alles, was du bisher programmiert hast wieder verwendet werden. Es kommt nur neues hinzu ...