

1 Grundlagen der Objektorientierung

Dieses Kapitel stellt eine solide, pragmatische Einführung in die fundamentalen Konzepte der Objektorientierung dar. Es werden die Begriffe **Klasse**, **Objekt** und **Methode** vorgestellt. Obwohl ein erster Blick auf den Quelltext geworfen wird, gehe ich nicht auf Details der Java-Programmierung ein.

Da dieses Kapitel auch eine Einführung in die Entwicklungsumgebung **BlueJ** darstellt, werde ich den Text mit vielen Bildern begleiten. Dem Anfänger wird somit eine schnelle Orientierung geboten.

1.1 Objekt, Klasse

Wenn du ein Programm entwickelst, dann erstellst du immer ein Modell eines Ausschnitts der realen Welt. Dieser Ausschnitt setzt sich aus Objekten zusammen, die im Anwendungsbereich vorkommen. Diese Objekte müssen im zu erstellenden Computermodell geeignet repräsentiert werden.

Diskussion: Unser Gymnasium

Programm zur Stundenplanerstellung

Programm zur Verwaltung der Schülerdaten

Programm zur Verwaltung der Lernmittelfreien Bücherei

Objekte können klassifiziert bzw. kategorisiert werden. Eine Klasse beschreibt also alle Objekte einer Kategorie.

Diskussion: Klasse und Objekt

Welche Objekte gibt es an unserer Schule? Welche Klassen?

Ist der Begriff „Auto“ eine Klasse oder ein Objekt?

1.2 Objekte erzeugen

Bemerkung:

1. Die Begriffe *Instanz* und *Objekt* verwende ich im Unterricht synonym.
2. In diesem Kapitel werden die Projekte *Figuren* und *Zeichnung* verwendet. Vorgestellt und ausführlich besprochen werden diese beiden Projekte von **M. Kölling** und **D. Barnes** in ihrem Lehrbuch *Objektorientierte Programmierung mit Java (2. Auflage)* bzw. *Java lernen mit BlueJ (3. Auflage)*.

Übung 1.1:

Hole dir vom Schulserver die Datei *Figuren* auf den Desktop oder in einen geeigneten Ordner. Nun starte die Entwicklungsumgebung **BlueJ**. Unter dem Menüpunkt „*Project > Open Project ...*“ suche diese Datei *Figuren* und öffne sie.

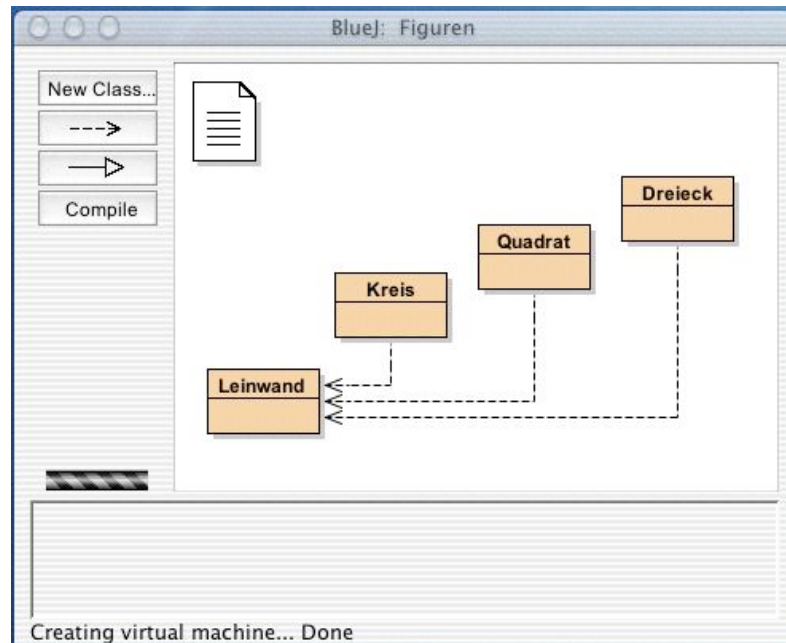


Abbildung 1.1: Projekt *Figuren* in **BlueJ**

Die **Klassen** dieses Projekts heißen *Kreis*, *Quadrat*, *Dreieck* und *Leinwand*.

Übung 1.2:

Klicke mit der rechten Maustaste auf die Klasse *Kreis* und wähle im Kontextmenü *new Kreis()*. Du hast gerade dein erstes Objekt mit dem Namen *kreis1* erzeugt. Es erscheint in der Objektleiste.



Abbildung 1.2: Objekt *kreis1*

Übung 1.3

Erzeuge ein weiteres Kreisobjekt. Erzeuge dann ein Quadratobjekt.

1.3 Methoden aufrufen

Übung 1.4:

Ein Rechts-Klick auf das Objekt *kreis1* erzeugt wiederum ein Kontextmenü mit mehreren Einträgen. Wähle den Eintrag *void sichtbarMachen()*. Nun wird das Objekt *kreis1* in einem separaten Fenster sichtbar.

Probiere auch die Einträge *nachRechtsBewegen()*, *nachUntenBewegen()*, *unsichtbarMachen()* und *sichtbarMachen()* aus.

Diese Einträge im Kontextmenü des Objekts *kreis1* repräsentieren Operationen, mit denen das Objekt manipuliert werden kann. Diese Operationen heißen **Methoden**.

Übung 1.5:

Was geschieht, wenn du die Methode *nachUntenBewegen()* zweimal aufrufst? Oder dreimal? Was passiert, wenn du *unsichtbarMachen()* zweimal aufrufst?

1.4 Parameter

Rufe nun die Methode *horizontalBewegen()* auf. Es erscheint ein Dialog, der dich um eine Eingabe auffordert.

Übung 1.6:

- a) Untersuche die Methoden *vertikalBewegen()*, *langsamBewegen()* und *groesseAendern()*.
- b) Finde heraus, wie man den Kreis um 70 Einheiten (Bildschirmpunkte) nach links(!) bewegt.

Die zusätzlichen Werte, die manche Methoden benötigen, werden *Parameter* genannt. Eine Methode gibt an, welche Art von Parametern sie erwartet. Im Dialogfenster der Methode *horizontalBewegen()* erscheint die Zeile

void horizontalBewegen(int entfernung)

im oberen Bereich. Dieses bezeichnet man als *Signatur* einer Methode.

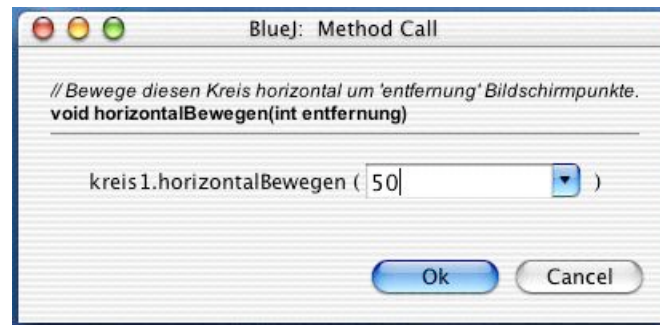


Abbildung 1.3: Methodenaufruf mit Parameter

1.5 Datentypen

Ein **Typ** gibt an, welche Art von Information übergeben werden kann. Zum Beispiel bezeichnet der Typ *int* ganze Zahlen.

Die Methode *farbeAendern()* erwartet einen Parameter vom Typ *String*.

Übung 1.7

- Rufe die Methode *farbeAendern()* deines Objekts *kreis1* auf und übergebe den Parameter „rot“ (mit Anführungszeichen).
- Was passiert, wenn du eine Farbe angibst, die nicht unterstützt wird?
- Rufe die Methode *farbeAendern()* auf und gib eine Farbe ohne Anführungszeichen an. Was passiert?

1.6 Eine Klasse, viele Objekte

Übung 1.8:

Erzeuge mehrere Kreis-Objekte auf der Objektleiste, indem du mehrfach den Eintrag *new Kreis()* im Kontextmenü auswählst. Mache diese Objekte sichtbar, bewege sie, mache einen Kreis groß und gelb, einen anderen klein und grün. Probiere auch die anderen Figuren und ändere deren Position, Größe und Farbe.

Sobald du eine Klasse hast, kannst du beliebig viele Objekte dieser Klasse erzeugen. Jedes dieser Objekte hat seine eigene Position, Farbe und Größe. Du änderst die Eigenschaft eines Objekts (z.B. die Farbe), indem du eine Methode an diesem Objekt aufrufst.

1.7 Zustand

Die Menge der momentanen Werte der Attribute, die ein Objekt definieren (wie x- und y-Position, Farbe, Durchmesser und Sichtbarkeit eines Kreises), wird

auch als *Zustand* des Objekts bezeichnet. In **BlueJ** lässt sich der Zustand eines Objekts mit Hilfe des *Objektinspektors* untersuchen.

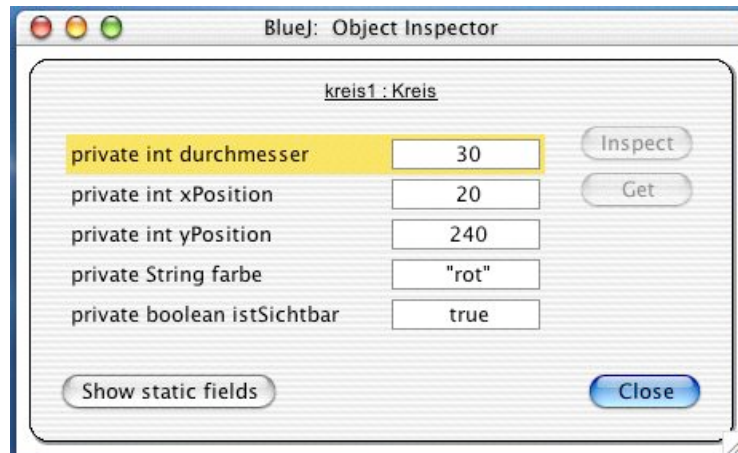


Abbildung 1.4: Objektinspektor zeigt den Zustand des Objekts *kreis1*

Übung 1.9:

Sorge dafür, dass mehrere Objekte auf der Objektleiste liegen und untersuche diese der Reihe nach. Versuche den Zustand eines Objekts zu ändern, während der Objektinspektor geöffnet ist.

In Java werden diese Attribute von Objekten *Datenfelder* oder einfach nur *Felder* genannt.

1.8 Das Innenleben eines Objekts

Alle Objekte derselben Klasse haben die gleichen Datenfelder. Das heißt, Anzahl, Typen und Namen der Datenfelder sind gleich, während der aktuelle Wert eines Datenfeldes in jedem Objekt anders sein kann.

Objekte unterschiedlicher Klassen haben dagegen unterschiedliche Datenfelder (Kreis: *durchmesser*; Dreieck: *breite* und *hoehe*)

Wenn also ein Objekt der Klasse *Kreis* erzeugt wird, dann hat dieses Objekt automatisch die Datenfelder *durchmesser*, *xposition*, *yPosition*, *farbe* und *istSichtbar*.

Die Werte der Datenfelder werden in dem Objekt gespeichert. Dies garantiert, dass jedes Objekt beispielsweise seine eigene Farbe kennt.

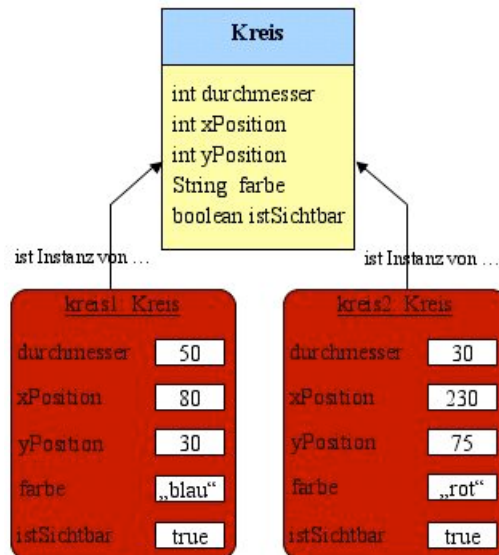


Abbildung 1.5: Eine Klasse und ihre Objekte mit Datenfeldern und Werten

Übung 1.10:

Erzeuge ein Bild ähnlich wie die unten gezeigte Abbildung 1.6.

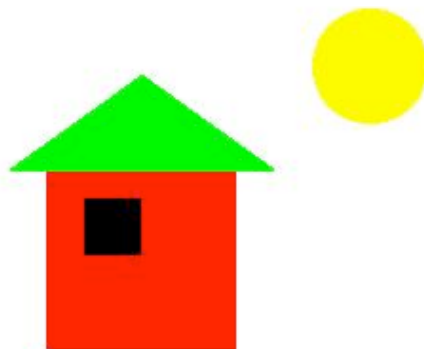


Abbildung 1.6: Darstellung in Übung 1.10

1.9 Objektinteraktionen

Übung 1.11:

Hole dir vom Schulserver das Projekt *Zeichnung*. Öffne in **BlueJ** dieses Projekt. Erzeuge ein Objekt der Klasse *Zeichnung* und rufe die Methode *zeichne()* auf. Probiere auch die beiden anderen Methoden.

Übung 1.12:

Wie zeichnet deiner Meinung nach die Klasse *Zeichnung* diese Bild?

Du hast in diesem Projekt eine zusätzliche Klasse *Zeichnung*. Diese Klasse ist so programmiert, dass sie automatisch das tut, was du in Übung 1.10 manuell getan hast.

Du erzeugst also ein Objekt dieser Klasse *Zeichnung*, und dieses erzeugt nun zwei Quadrat-Objekte, ein Dreieck-Objekt und eine Kreis-Objekt. Anschließend werden die Methoden jeweils dieser Objekte aufgerufen, so dass die Zeichnung wie oben aussieht.

Entscheidend ist hier:

Objekte können andere Objekte erzeugen und die Methoden dieser weiteren Objekte aufrufen. Man sagt auch, die Objekte können miteinander kommunizieren, indem sie gegenseitig ihre Methoden aufrufen.

1.10 Quelltext

Übung 1.13:

- a) Aktiviere das Kontextmenü der Klasse *Zeichnung* und wähle den Eintrag *Bearbeiten (Open Editor)*.
- b) Finde im Quelltext der Klasse *Zeichnung* den Abschnitt, der das eigentliche Zeichnen ausführt. Ändere ihn so, dass die Sonne blau statt gelb ist.
- c) Ändere das Programm derart, dass eine 2. Sonne hinzugefügt wird.
- d) Füge der Version von *Zeichnung* mit nur einer Sonne einen Sonnenuntergang hinzu. Hierzu soll die Sonne langsam untergehen, rot werden und dann verschwinden.
- e) Für den Sonnenuntergang soll eine eigene Methode *sonnenuntergang()* definiert werden.

1.11 Die Klasse Garten

Bisher hast du eine Klasse verwendet bzw. deren Quelltext verändert. Nun wirst du eine Klasse neu programmieren. Dazu verwendest du eine Kopie des Projekts *Figuren* und speicherst dieses unter dem Namen *Garten*.

Aufgabe des Projekts *Garten*:

Es soll ein Apfelbaum mit einigen roten Äpfeln und einem Vogelhaus gezeichnet werden. Ein Apfel fällt herunter und aus diesem kriecht dann ein kleiner Wurm.

1.11.1 Programmierung einer Klasse

Du kannst nun dieses Problem programmgesteuert erledigen?

1. Welche Aktionen sind notwendig?
 - a) Quadrate erzeugen und Namen geben *stammU, stammO, haus*
 - b) Kreise erzeugen und Namen geben *krone, apfel1, apfel2*
 - c) Dreiecke erzeugen und Namen geben *dach*
2. Wie kann man dies durch eine Java-Klasse erledigen?

Du benötigst eine Klasse *Baum*, die diese Bilder produziert.

Programmiere also nun zuerst die Klasse *Baum*. Dazu erzeugst du im Projekt *Garten* mit *New Class* eine neue Klasse, benennst sie *Garten* und editierst diese.

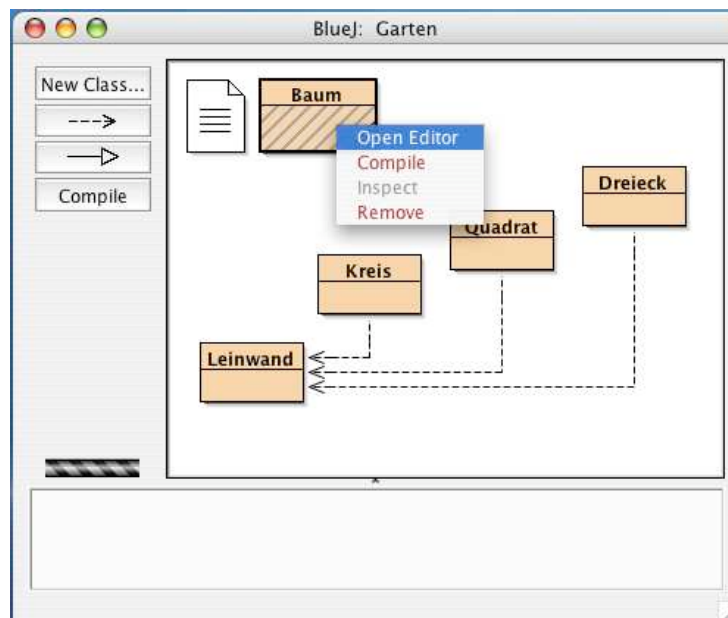


Abbildung 1.7: Projekt *Garten* mit der neu erstellten Klasse *Baum*

Im Editorfenster löschst du nun den vorgeschlagenen Quelltext bis auf die in Abbildung 1.8 dargestellten Zeilen.

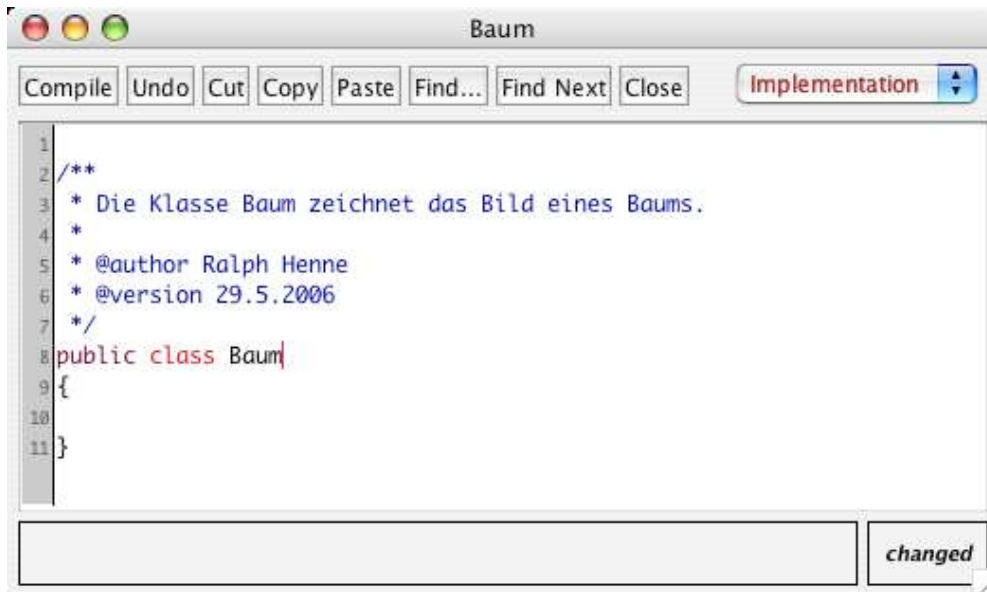


Abbildung 1.8: Das Editorfenster der Klasse *Baum*

Jeder Baum, d.h. jedes Objekt der Klasse *Baum* muss

1. Quadrate, Kreise und Dreiecke erzeugen und jedem dieser Elemente einen Namen geben
2. eine Methode z.B. *zeichneBaum()* besitzen, die die Objekte an die richtige Stelle verschiebt, sie vergrößert bzw. verkleinert, sichtbar macht, ihre Farbe wechselt.

1.11.2 Erzeugung von Objekten

Die Erzeugung von Objekten besteht aus zwei Schritten:

1. Ein Name für das noch zu schaffende Objekt wird registriert.
2. Ein neues Objekt wird erzeugt und unter diesem Namen gespeichert.

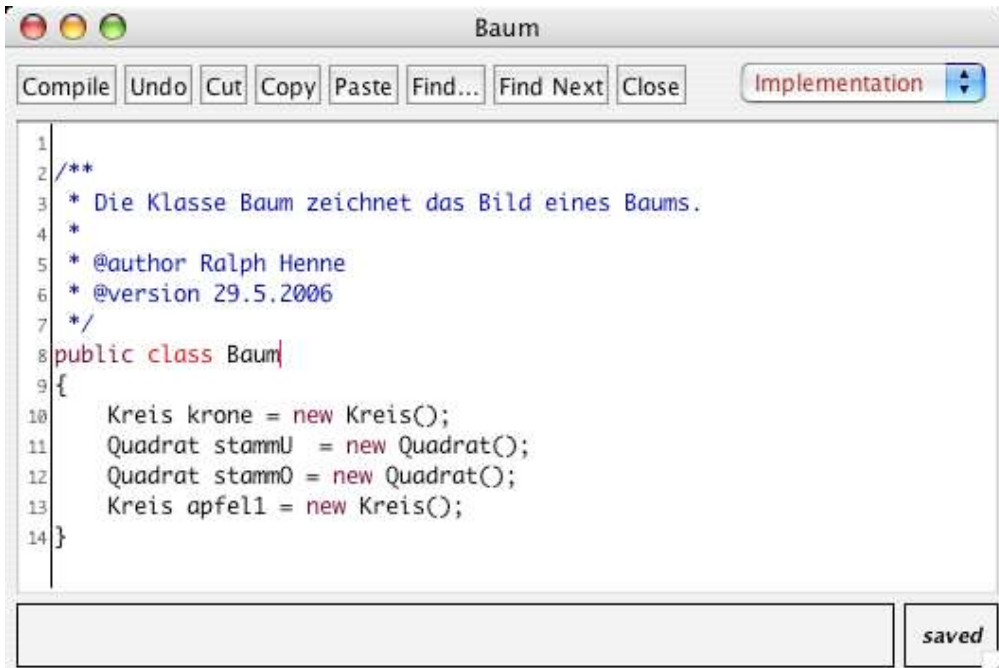
Kreis krone; Registriere den Namen *krone* für ein noch zu schaffendes Objekt der Klasse *Kreis*.

krone = new Kreis(); Konstruiere ein neues Objekt der Klasse *Kreis* und nenne es *krone*.

Diese beiden Schritte kannst du zusammenfassen durch folgende Anweisung:

| | |
|---|---|
| <code>Kreis krone = new Kreis();</code> | |
| Registriere <code>krone</code> als Name für ein Objekt der Klasse <code>Kreis</code> . | Ein <code>neues</code> konstruiertes Objekt der Klasse <code>Kreis</code> . |
| = | |
| Unter dem soeben registrierten Namen <code>krone</code> wird das neu konstruierte Objekt der Klasse <code>Kreis</code> abgespeichert. | |

Also programmierst du nun diese benötigten Objekte in der Klasse *Garten*.



```
1
2 /**
3  * Die Klasse Baum zeichnet das Bild eines Baums.
4  *
5  * @author Ralph Henne
6  * @version 29.5.2006
7  */
8 public class Baum
9 {
10     Kreis krone = new Kreis();
11     Quadrat stammU = new Quadrat();
12     Quadrat stammO = new Quadrat();
13     Kreis apfell = new Kreis();
14 }
```

Abbildung 1.9: Klasse *Baum* erzeugt weitere Objekte

Die Klasse *Baum* lässt sich schon kompilieren und man kann bereits Objekte erzeugen. Im Kontextmenü der Klasse *Baum* findest du die Methode `new Baum()` und kannst damit das Objekt *baum1* erzeugen.

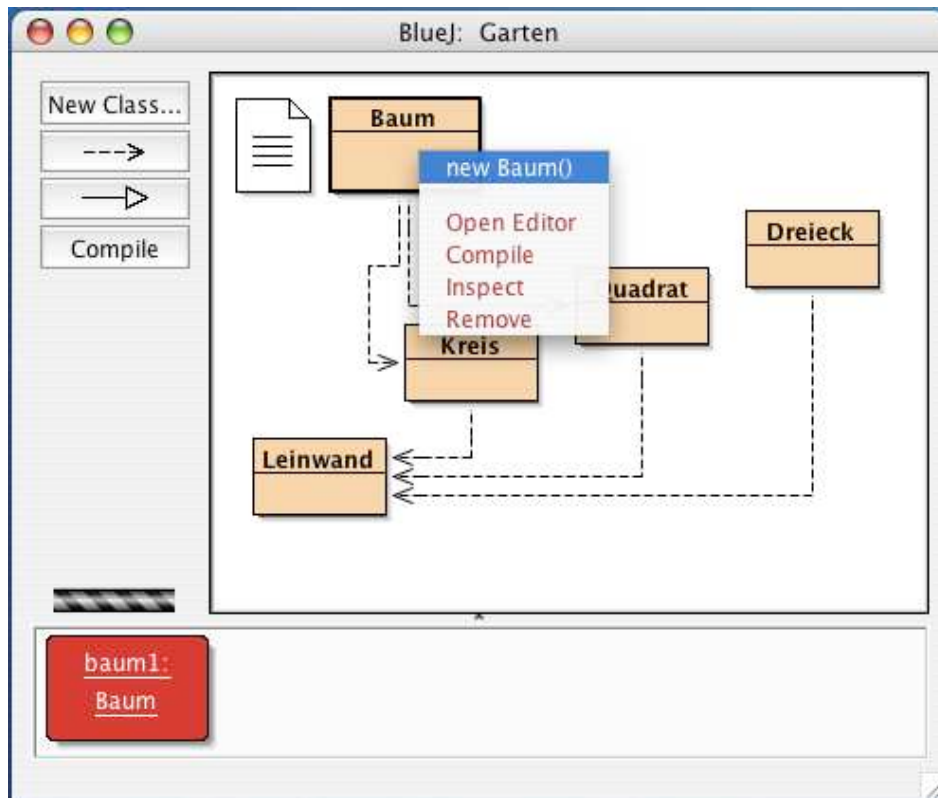


Abbildung 1.10: Projekt *Garten* mit erzeugtem Objekt *baum1*

Untersuche nun das Objekt *baum1* mit Hilfe des Inspektors:

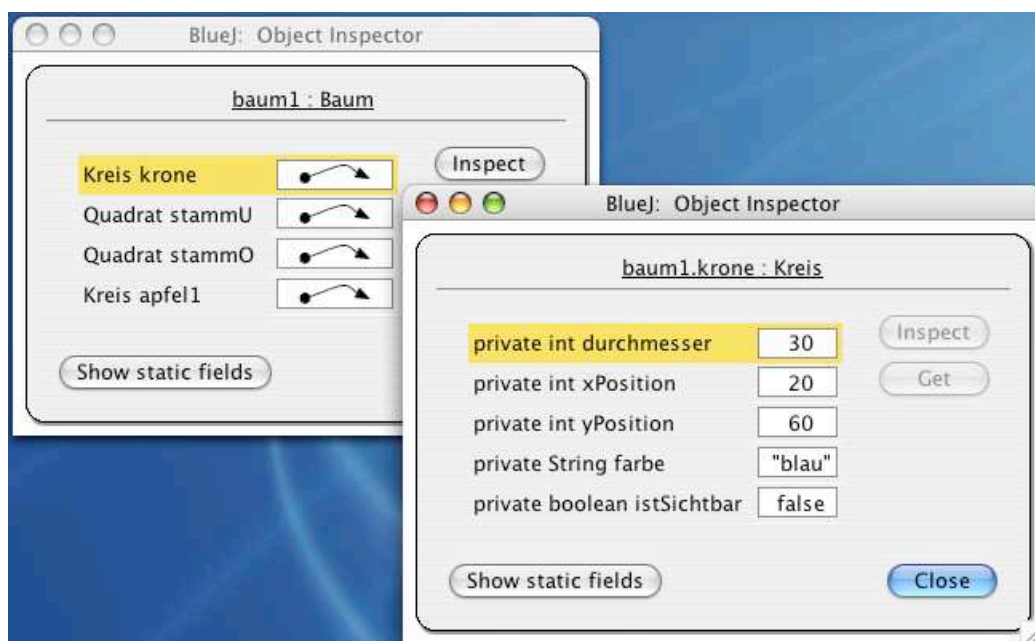


Abbildung 1.11: Inspektor des Objekts *baum1*

Der Inspektor findet bei *baum1* die Datenfelder *krone*, *stammU*, *stammO* und *apfel1*. Diese sind keine Zahlen, sondern Objekte der Klassen *Kreis*, *Quadrat*,

Quadrat und *Kreis*. Ein Doppelklick folgt dem Link zu dem betreffenden Objekt, hier in der Abbildung 1.11 zum Objekt *krone* der Klasse *Kreis*.

1.11.3 Aktionen mit Hilfe von Methoden

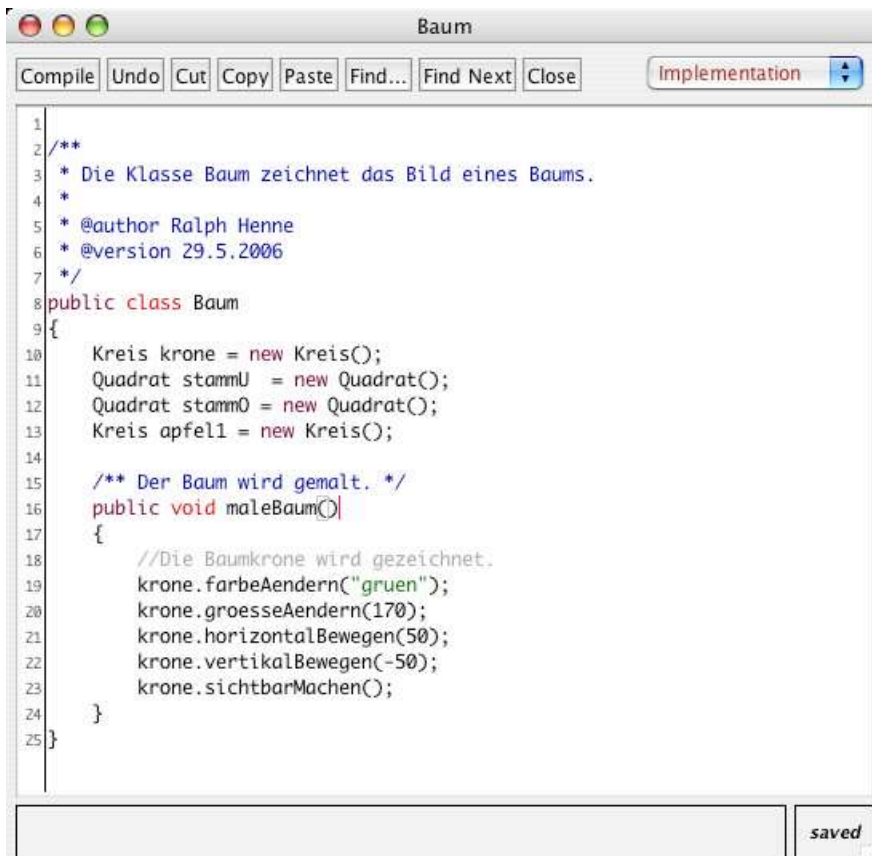
Nun soll dieser Baum gezeichnet werden. Du musst also der Klasse *Baum* eine Methode zur Verfügung stellen, die den Apfelbaum malen kann.

Eine Methode in Java hat stets einen *Kopf*, bestehend aus

- einem Ergebnistyp – hier *void* (engl.: leer)
- einem Namen – hier *maleBaum*
- einer Parameterliste – hier leer, also *()*

und einem **Rumpf**, in dem steht, was die Methode tun soll

- krone.farbeAendern(„gruen“);*
- krone.groesseAendern(140);*
- ... etc. ...



```
1
2 /**
3  * Die Klasse Baum zeichnet das Bild eines Baums.
4  *
5  * @author Ralph Henne
6  * @version 29.5.2006
7  */
8 public class Baum
9 {
10     Kreis krone = new Kreis();
11     Quadrat stammU = new Quadrat();
12     Quadrat stammO = new Quadrat();
13     Kreis apfeli = new Kreis();
14
15     /** Der Baum wird gemalt. */
16     public void maleBaum()
17     {
18         //Die Baumkrone wird gezeichnet.
19         krone.farbeAendern("gruen");
20         krone.groesseAendern(170);
21         krone.horizontalBewegen(50);
22         krone.vertikalBewegen(-50);
23         krone.sichtbarMachen();
24     }
25 }
```

Abbildung 1.12: Die Klasse *Garten*

Hinweis:

Da du im oben abgebildeten Beispiel die Methoden der Klasse *Kreis* benötigst, ist es hilfreich, in einem weiteren Fenster den Quelltext dieser Klasse *Kreis* geöffnet zu haben.

Hausaufgabe:

Vervollständige das Projekt *Garten*. Es soll ein Baum mit mehreren Äpfeln gezeichnet werden. Ein Apfel fällt herunter auf den Boden und aus diesem kriecht ein Wurm. Im Baum sitzt ein Vogel.

Erstelle ein neues Projekt *Aquarium*. In diesem befinden sich mehrere Fische. Ein Fisch schwimmt hinter einem Stein. Ein anderer Fisch frisst einen weiteren. Oder erfinde etwas Ähnliches

Erstelle ein weiteres Projekt Deiner Wahl. Sei kreativ!!

Speichere dein Projekt auf einen Memory-Stick o.ä. und stelle es in der Informatikstunde vor.

Du kannst auch dein Projekt komprimieren an deinen Informatik-Lehrer senden unter folgender Adresse:

henne.info@gmx.de

Dabei bestehen zwei Möglichkeiten:

1. Möglichkeit:

Komprimiere den Ordner *Garten* mit einem Programm wie z.B. „WinZip“ und versende diese komprimierte Datei als Anhang einer E-Mail.

2. Möglichkeit:

Speichere Dein Projekt mit Quelltext in einem Verzeichnis

- Wähle dazu aus *Project > Create Far File ...*
- Speichere es als **jar-Datei** (komprimiertes Archiv) **mit Quelltext** (source code)
- Wähle als Verzeichnisnamen Deinen Nachnamen

und versende diese komprimierte Datei als Anhang einer E-Mail.

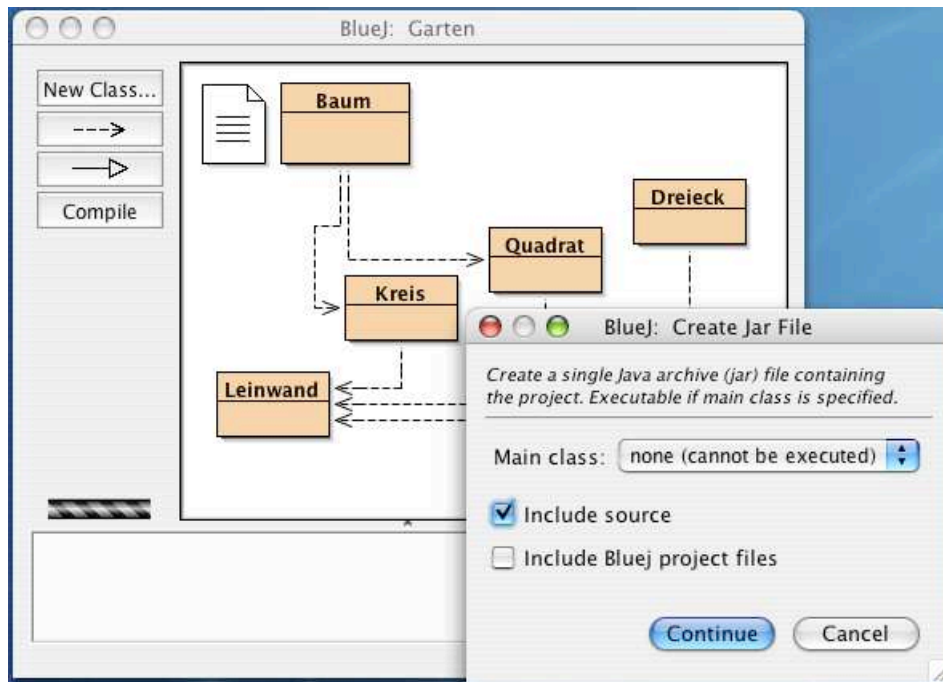


Abbildung 1.13: Erstellen einer jar-Datei



Abbildung 1.14: Name des erstellten jar-Datei

Hinweis:

Mit Hilfe der 2. Möglichkeit können „stand alone Applikationen“ erstellt werden. Hierzu muss jedoch zusätzlich diejenige Klasse angegeben werden, in der sich die Methode *main()* befindet. Solche so genannten **jar-Dateien** lassen sich per E-Mail versenden und mit Doppelklick auf das entsprechende Icon starten.

Die Methode *main()* wirst du später kennen lernen.

Im Unterricht ergaben sich weitere Fragen zu diesem Kapitel. Diese werden besprochen im Zusatzkapitel „01zGarten“