Arbeiten mit BlueJ

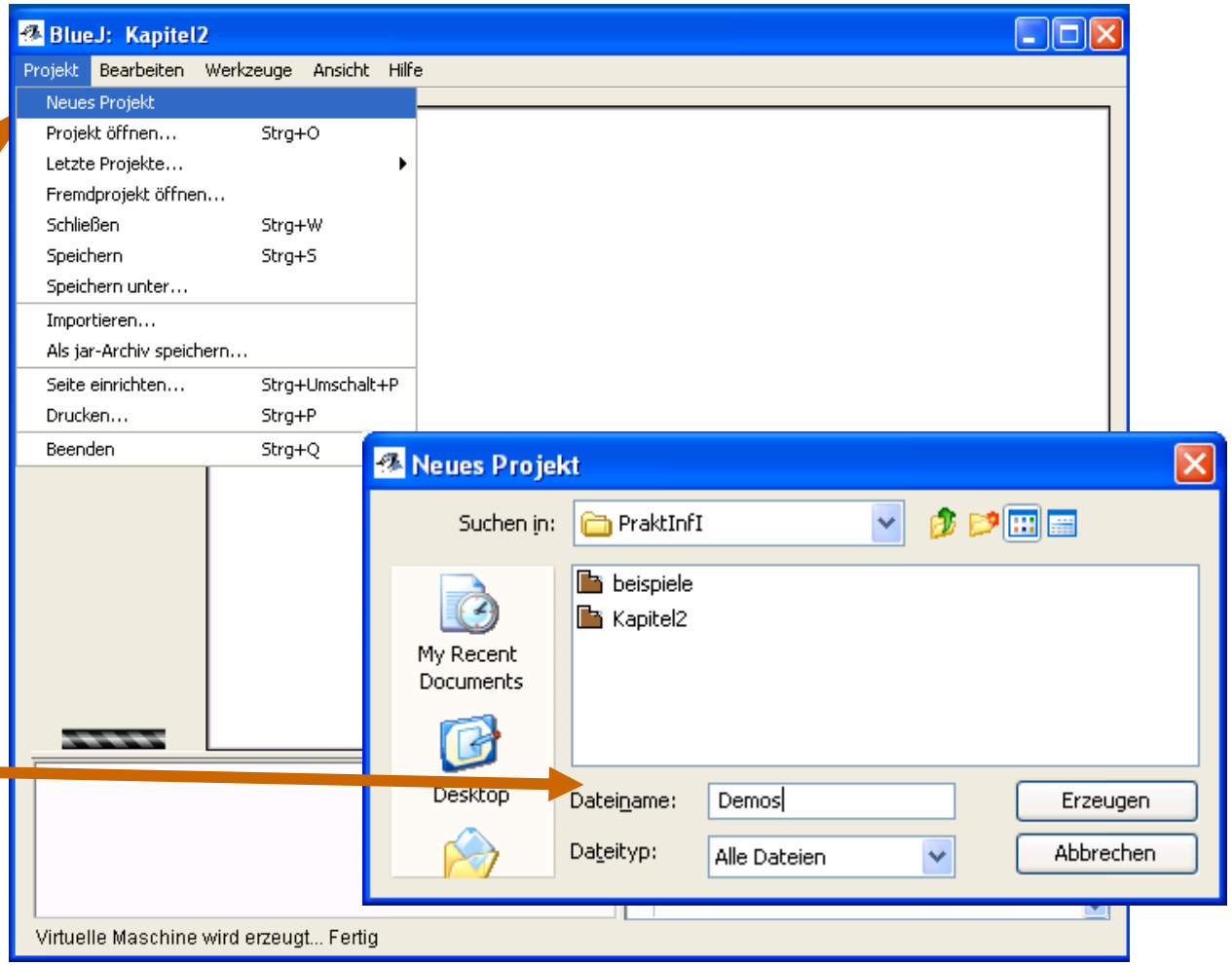
Compilieren, Editieren, Objekte erzeugen, Methoden aufrufen, Objekte inspizieren, vorhandene Klassen benutzen, eigene Klassen und Methoden erstellen

In BlueJ geht es einfach

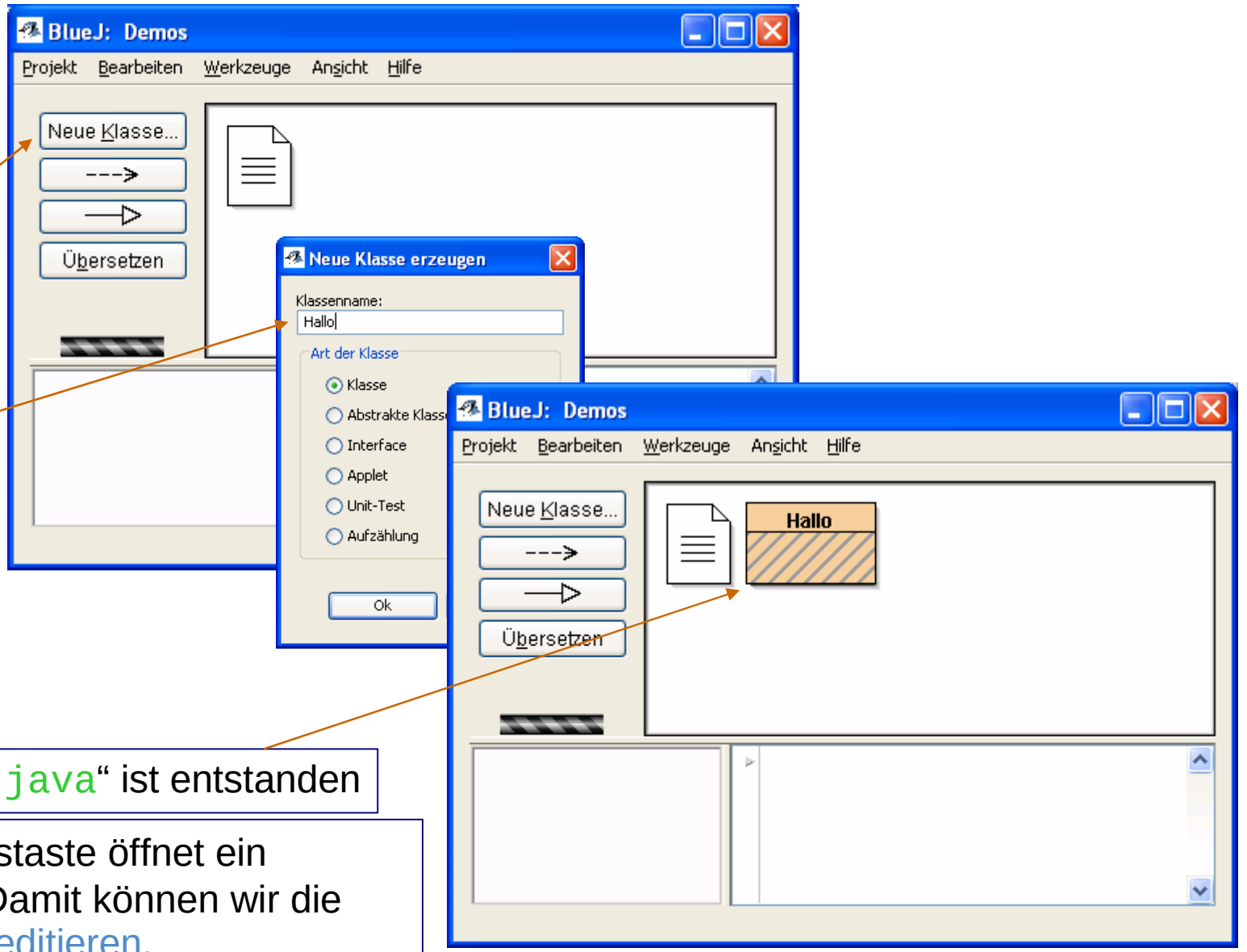
BlueJ durch Doppel-klick starten

Neues Projekt öffnen
Da passen dann viele Klassen hinein

Name ausdenken:
Demos



BlueJ, Erste Klasse ...



Neue Klasse

Name: `Hallo`

Klasse "`Hallo.java`" ist entstanden

Die rechte Maustaste öffnet ein Kontextmenü. Damit können wir die Klasse z.B. editieren.

Editieren, Compilieren

Der Editor enthält schon Beispielcode.

Meistens werden wir diesen verändern – editieren.

Wir können ihn aber auch schon gleich übersetzen – kompilieren

Shortcut: Strg-k

Hallo

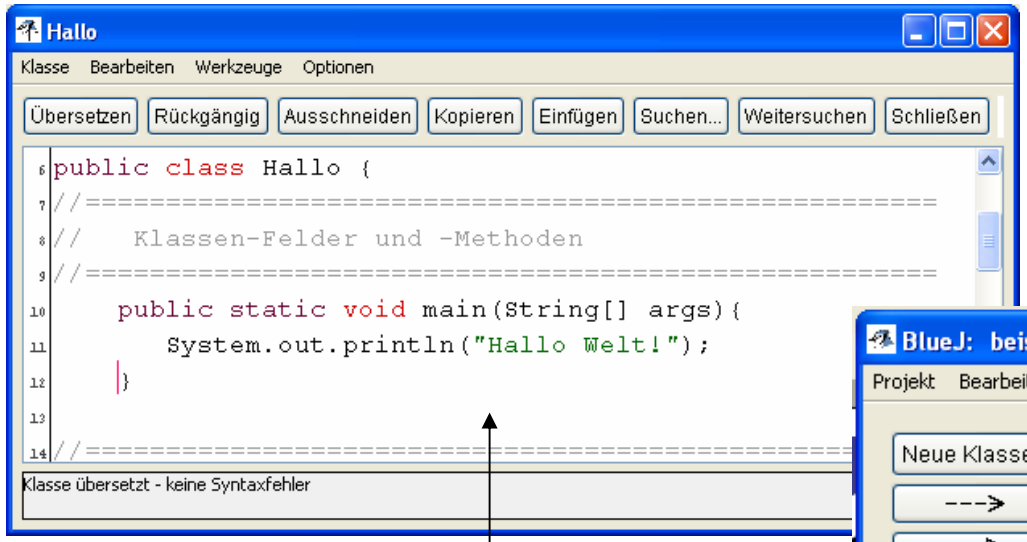
Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | Weitersuchen | Schließen

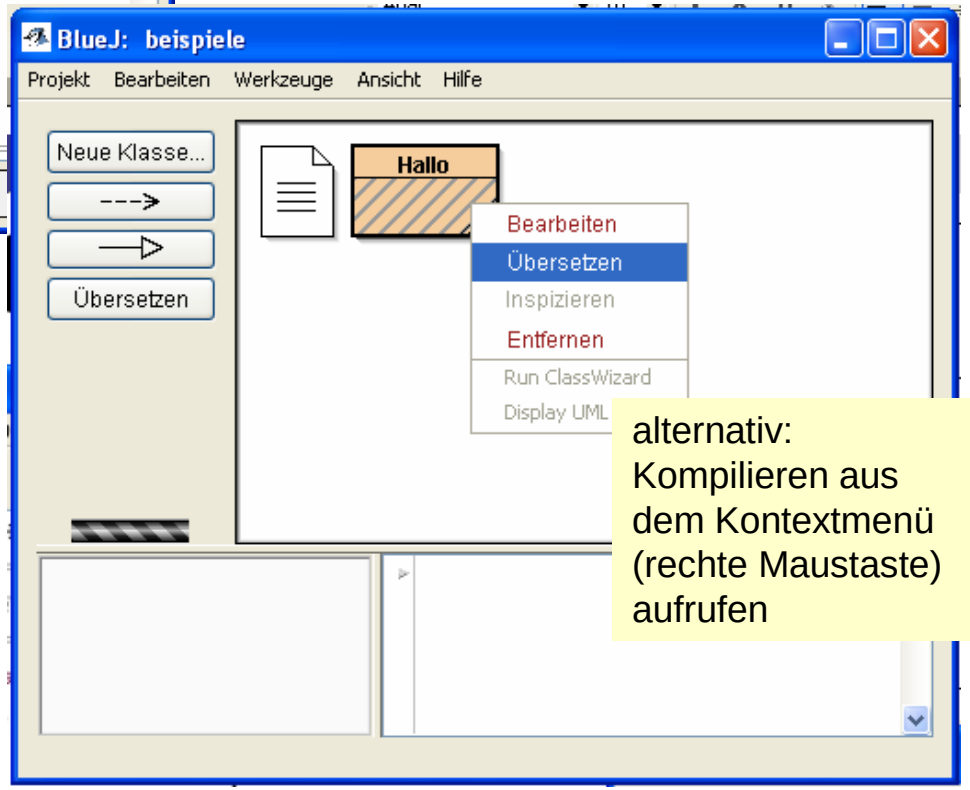
```
/** Beschreibung der Klasse Hallo.
 *
 * @author (Ihr Name)
 * @version (eine Versionsnummer oder ein Datum)
 */
public class Hallo {
//=========================================================
//   Klassen-Felder und -Methoden
//=========================================================
    public static void main(String[] args){
       System.out.println("Hal
//=========================================================
//   Objekt-Felder
//=========================================================
//=========================================================
//   Konstruktoren
//=========================================================
    /**
     * Default-Konstruktor für Objekte der Klasse Hallo
     */
      public Hallo(){
        // Initialisiere Objektfelder
      }
```

cursor

Editieren, Compilieren



Text hinzufügen, kompilieren (Ctrl-k)



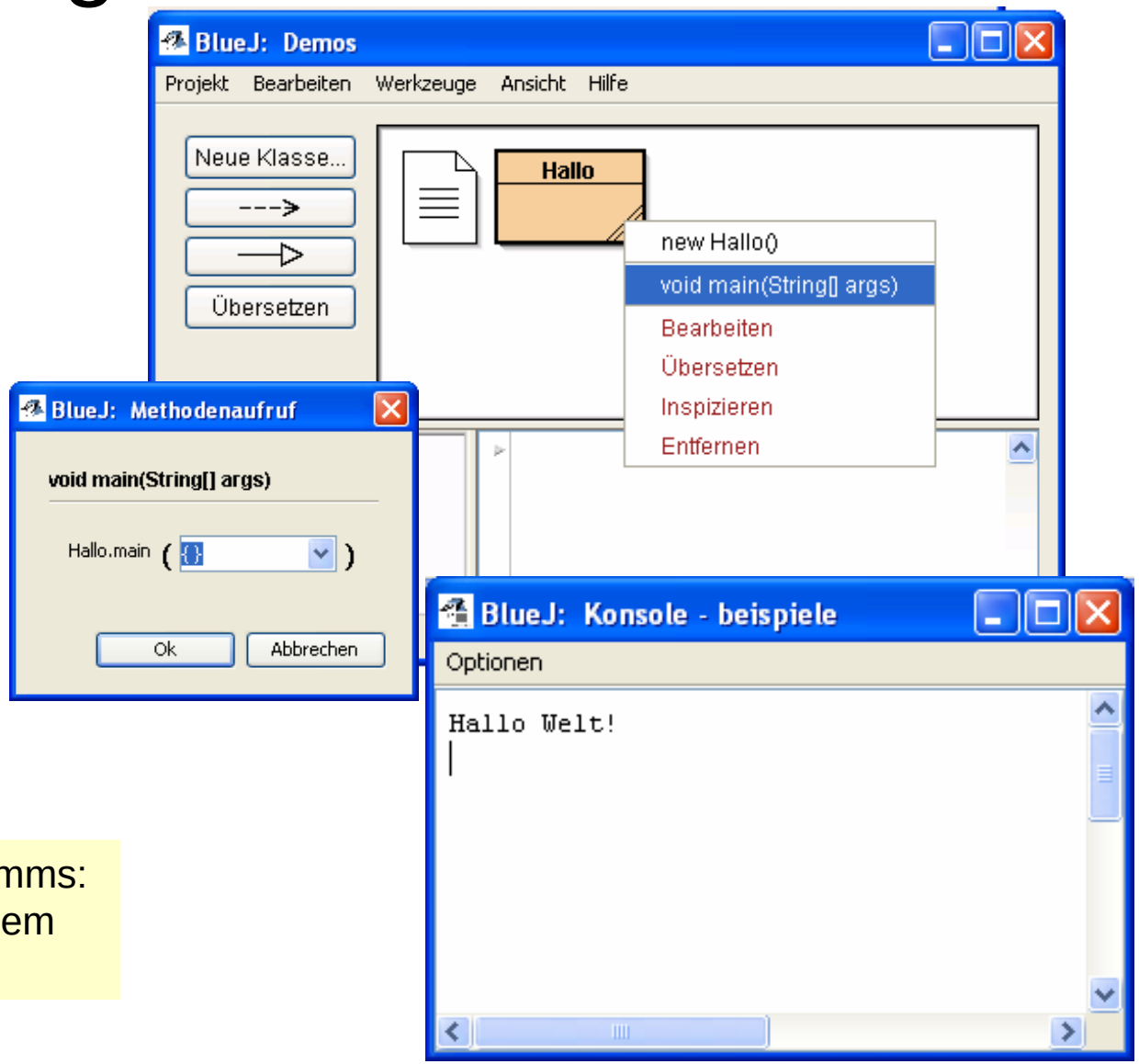
alternativ: Kompilieren aus dem Kontextmenü (rechte Maustaste) aufrufen

Aufruf und Ergebnis

Aufruf unserer Funktion
main
aus dem Kontextmenü

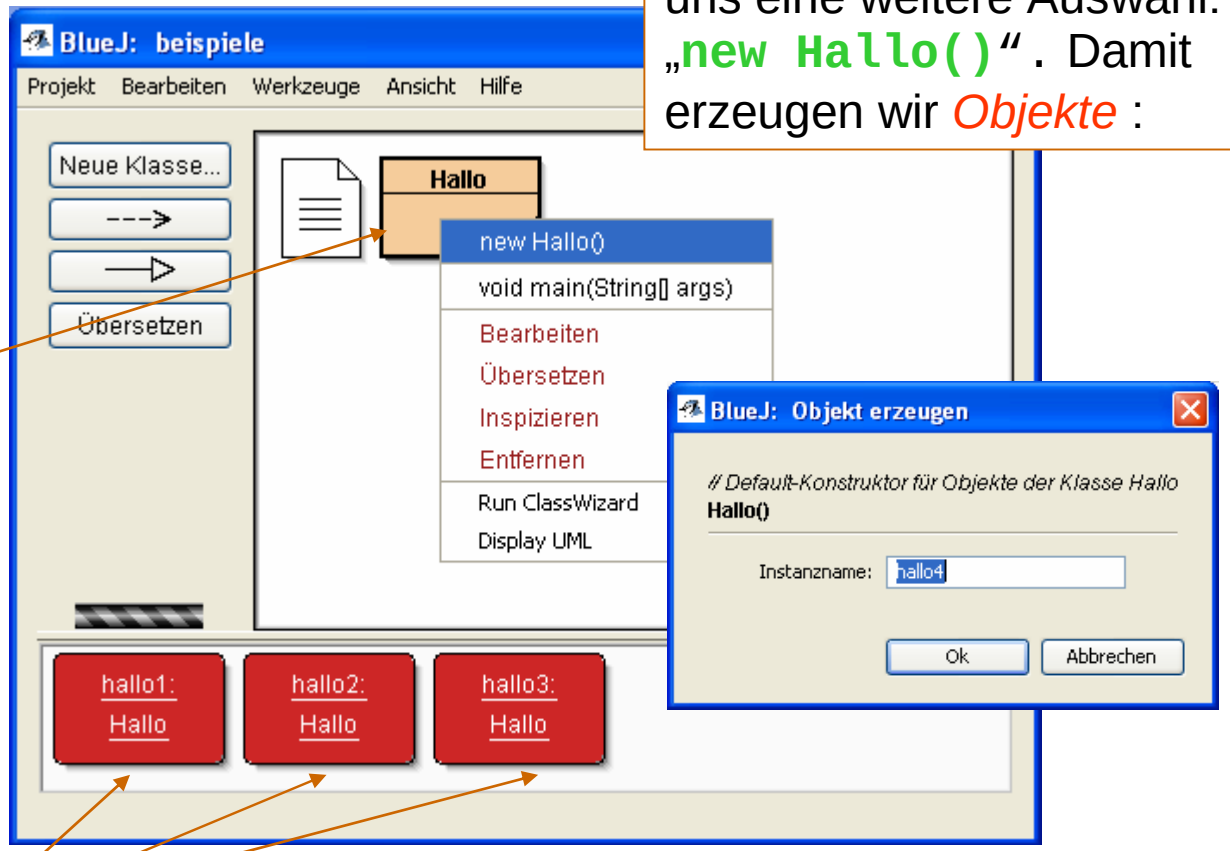
Angebot, einige Strings
als Parameter an
main
zu übergeben

Das Ergebnis des Programms:
Ein Konsolenfenster mit dem
gewünschten Output



Objekte erzeugen

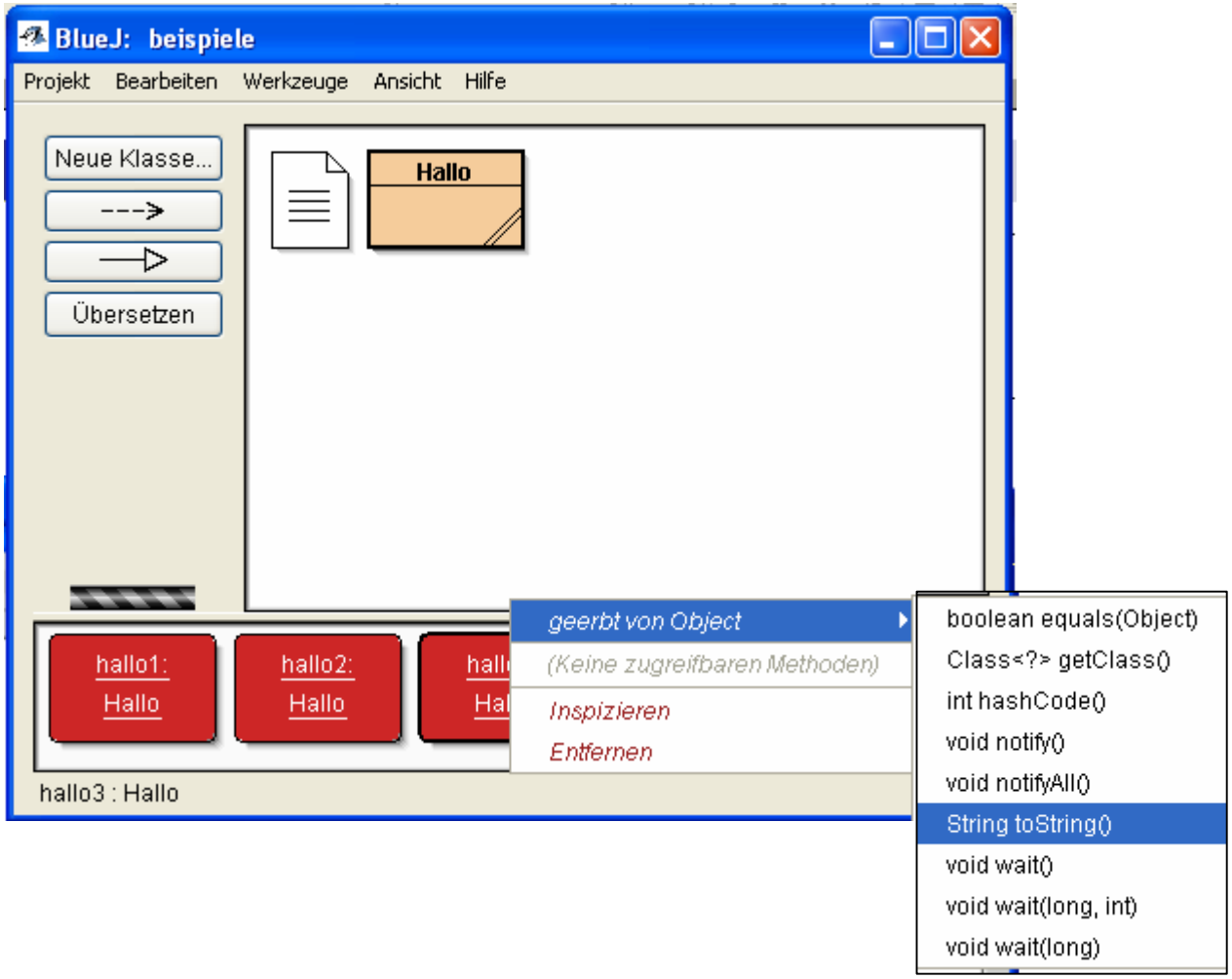
Das KontextMenü bietet uns eine weitere Auswahl: „`new Hallo()`“. Damit erzeugen wir *Objekte* :



Der Klasse im BlueJ-Fenster sieht man an, dass sie erfolgreich compiliert ist – sie ist jetzt nicht mehr gestreift.

Hier sind schon drei Objekte erzeugt worden

Objekte – unser Ziel



Objekte sind die Helden des Objektorientierten Programmierens.

In BlueJ können alle Fähigkeiten der Objekte durch Kontextmenüs direkt aufgerufen werden.

Zusätzlich kann man die Innereien der Objekte inspizieren (*Inspect*), oder das Objekt entfernen (*Remove*).

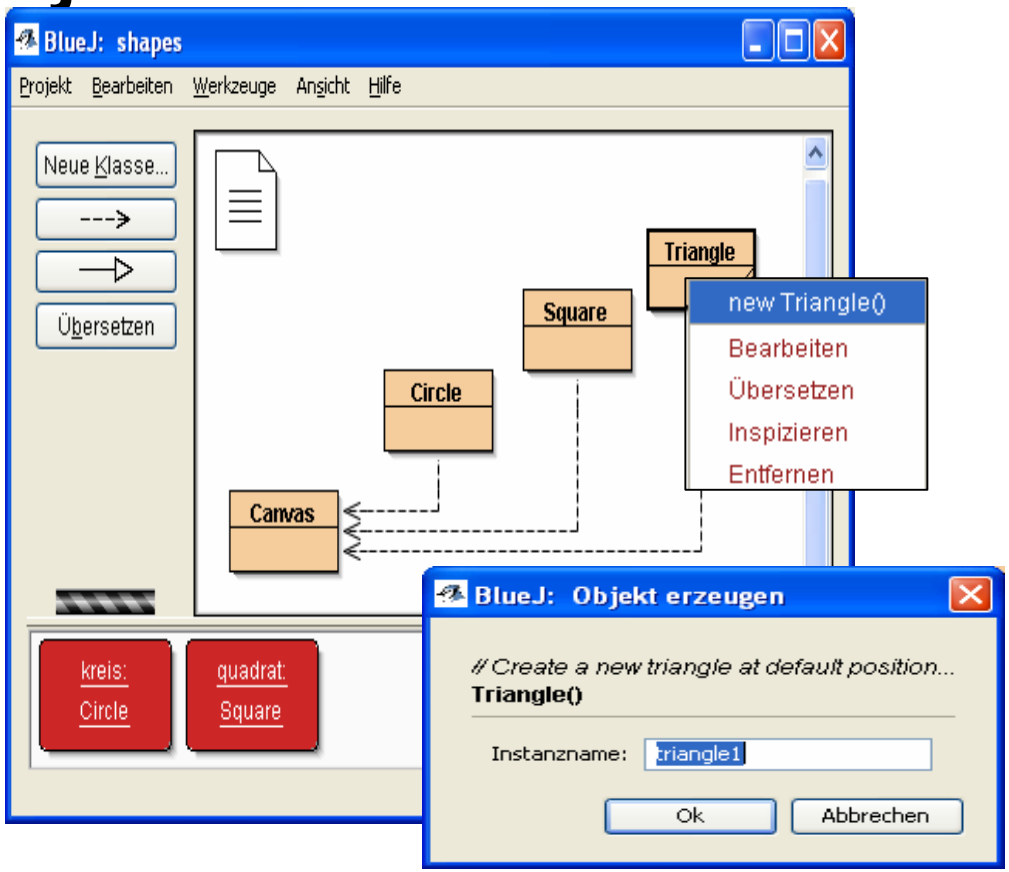
Figuren, als Objekte modelliert

Wir öffnen das vorhandene Projekt „**shapes**“ und erzeugen uns Objekte der *Klassen*
- Ø **Circle**
- Ø **Square**
- Ø **Triangle**.

Wir können den *Objekten* Namen geben, z.B.:
- § **kreis**
- § **quadrat**

oder die Vorgabe-Namen
- § **circle_1**
- § **circle_2**
- § **triangle_1** ...

akzeptieren.

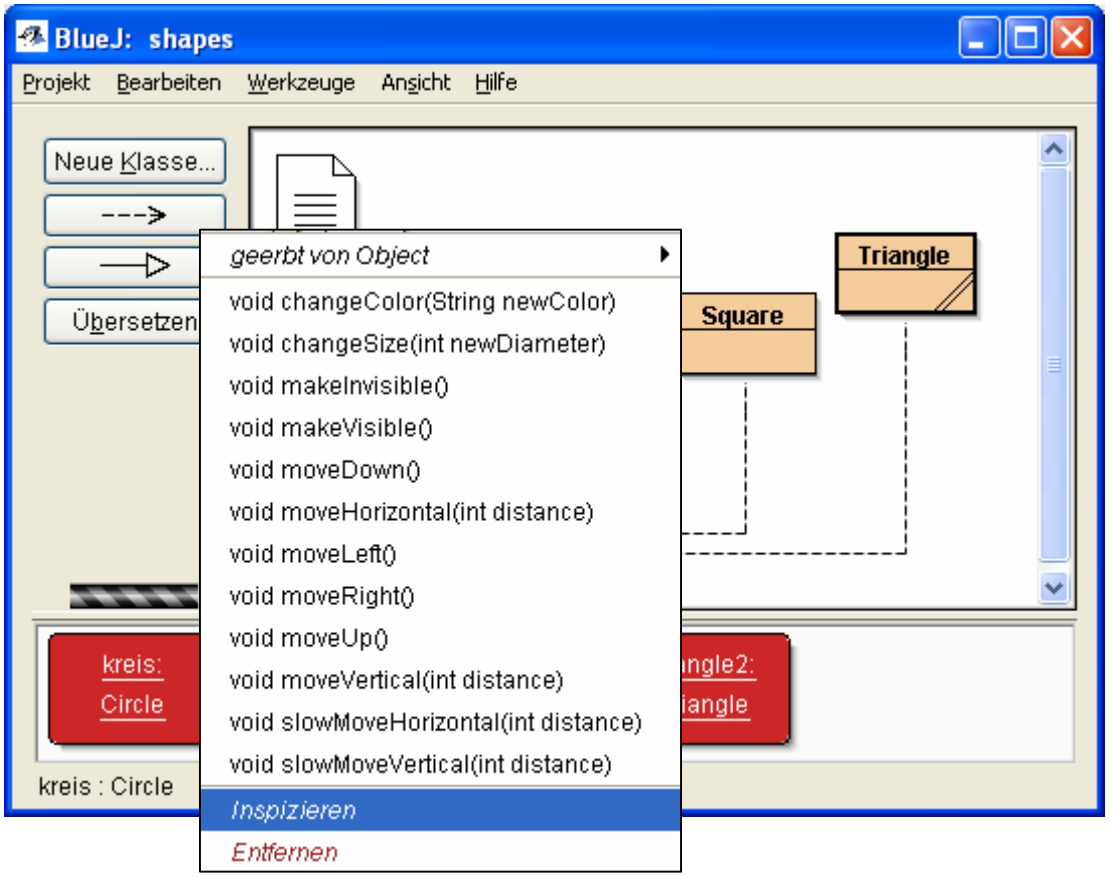


Wir halten uns an den allgemein akzeptierten Standard:

Namen für Objekte beginnen immer mit Kleinbuchstaben !!!

Namen für Klassen beginnen mit einem Großbuchstaben !!!

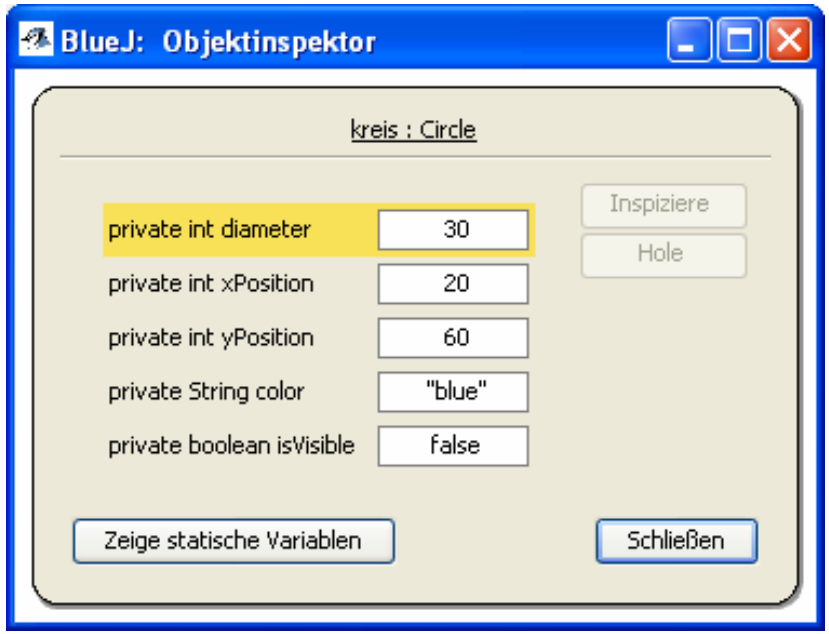
Warum sehen wir keinen Kreis ?



Wir *inspizieren* das Objekt **`kreis`** , indem wir aus dem Kontextmenü *Inspect* wählen.

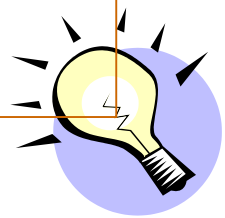
isVisible = false

Es öffnet sich ein Inspektor:



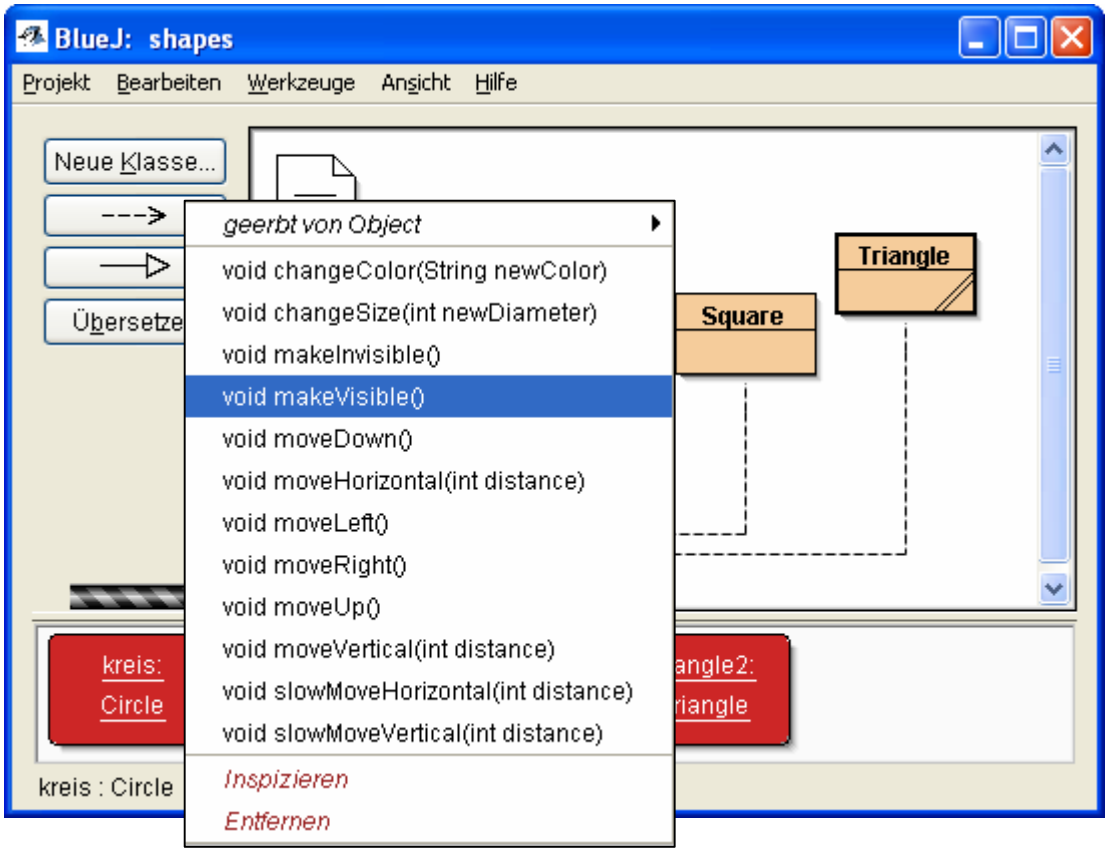
Der **kreis** hat also
- Ø diameter 30
- Ø xPosition 20
- Ø color „blue“

und die Eigenschaft
- Ø isVisible false

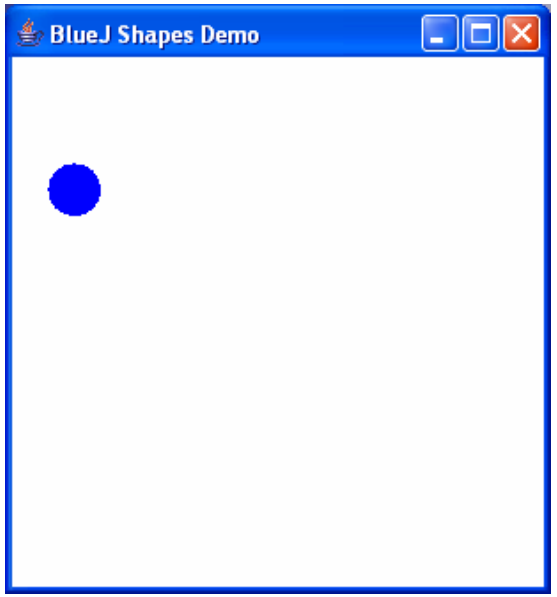


Wir müssen also diese Eigenschaft verändern.
Aber wie ?

Die Methode `makeVisible()`

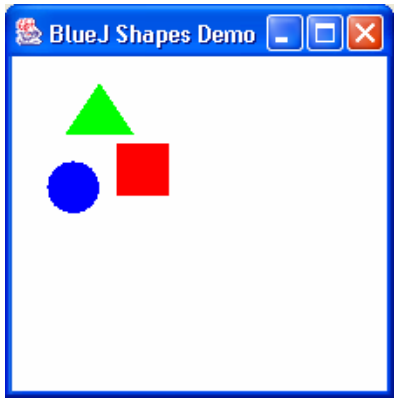


Im *KontextMenü* von **`kreis`** finden wir die Methode **`makeVisible()`**. Wir klicken diese an und … der Kreis wird in einem Fenster sichtbar



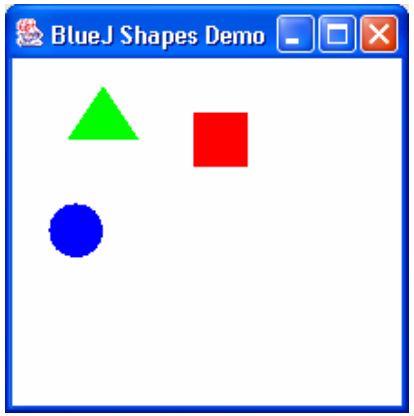
Methodenerkundung

Auf die gleiche Weise machen wir auch **`quadrat`** und **`triangle_2`** sichtbar.

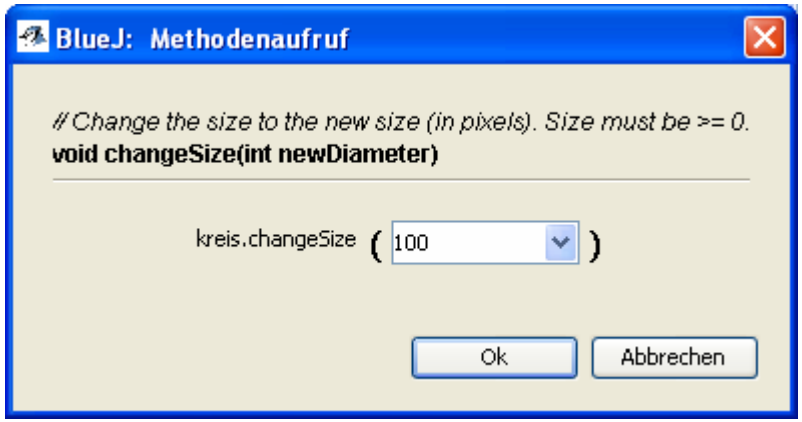


Anschliessend probieren wir weitere Methoden, z.B.

- Ø von **`square`** die Methode **`moveRight`**( )
- Ø von **`kreis`** die Methode **`moveDown`**( )
- Ø von **`square`** die Methode **`moveUp`**( )
- Ø von **`square`** die Methode **`moveRight`**( )



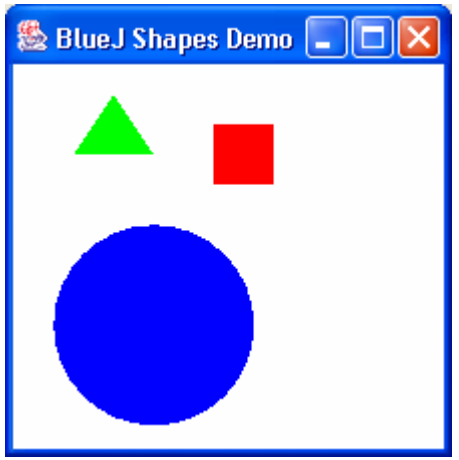
Methoden mit Argumenten



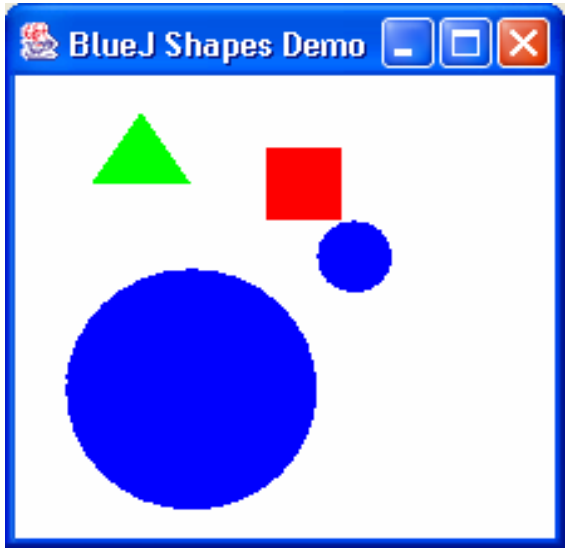
Jetzt wollen wir die Größe des Quadrats ändern und wählen die Methode **`changeSize(int)`**

Es öffnet sich ein InputFenster, das uns auffordert, einen neuen Durchmesser einzugeben ….

`int` bedeutet, dass eine ganze Zahl verlangt ist. In dem Kästchen steht aber noch ein Kommentar *//Change size ... Size must be >= 0.*
Wir geben 100 ein und ….



Weitere Objekte



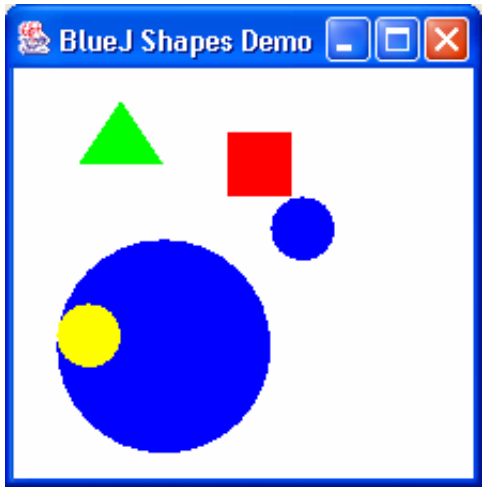
- Wir fügen zwei weitere Kreise hinzu
  - bewegen einen 100 nach rechts
  - den anderen um 50 nach unten
  - machen beide sichtbar
- … wir sehen nur zwei blaue Kreise.
  Wo ist der dritte ?

Genau, der große blaue verdeckt den kleinen blauen – oder umgekehrt …
Zum Glück finden wir im KontextMenü von `circle_2` die Methode
`changeColor(String)`

String-Argumente

Wir sollen einen *String*, also einen Text in Anführungszeichen “ “ eingeben. Der Programmierer hat bemerkt, dass “red“, “yellow“, “blue“, etc. erlaubt sind :

Jetzt sehen wir auch **`circle_2`** :



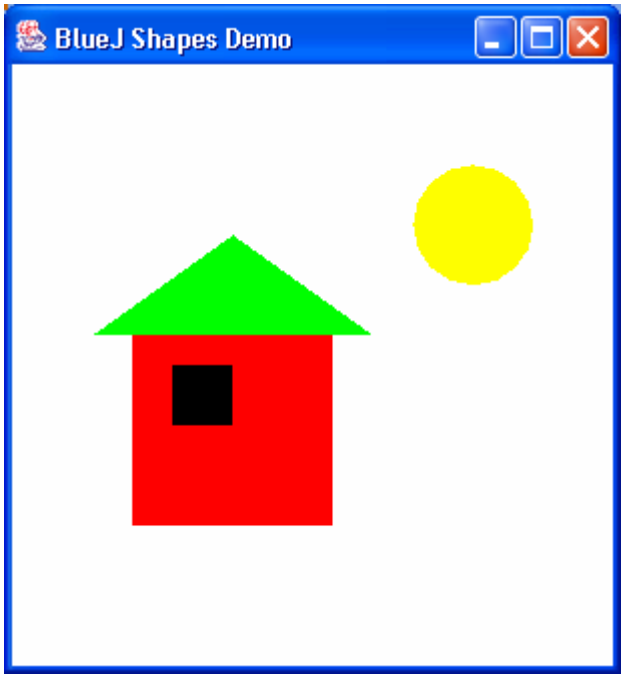
Nach-Inspektion

- Wir schauen noch einmal im Inspektor für **`kreis`** nach und erkennen, dass die Felder
  - **`diameter`**
  - **`xPosition`**
  - **`yPosition`**
  - **`isVisible`**

  neue Werte haben.

- Offensichtlich kommen drei Typen von Werten vor
  - **`int`** steht für ganzzahlige Werte, z.B. 1, 20, -300, 0, 42
  - **`String`** steht für Text, z.B. “blue“, “OttoKar“, “ich habe fertig !“
  - **`boolean`** steht für einen der Werte true oder false (ohne “ “).

Ein Haus für BlueJ

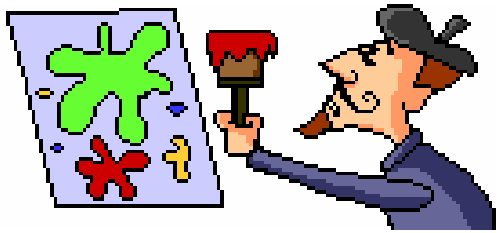


- Versuchen Sie, diese idyllische Szene nachzuzeichnen
- Entwerfen Sie Ihr eigenes Haus
- Malen Sie Ihr eigenes Bild
  - z.B. Fische im Aquarium
  - Smileys
  - eine Lokomotive
  - ...

Ein Maler als Java-Programm

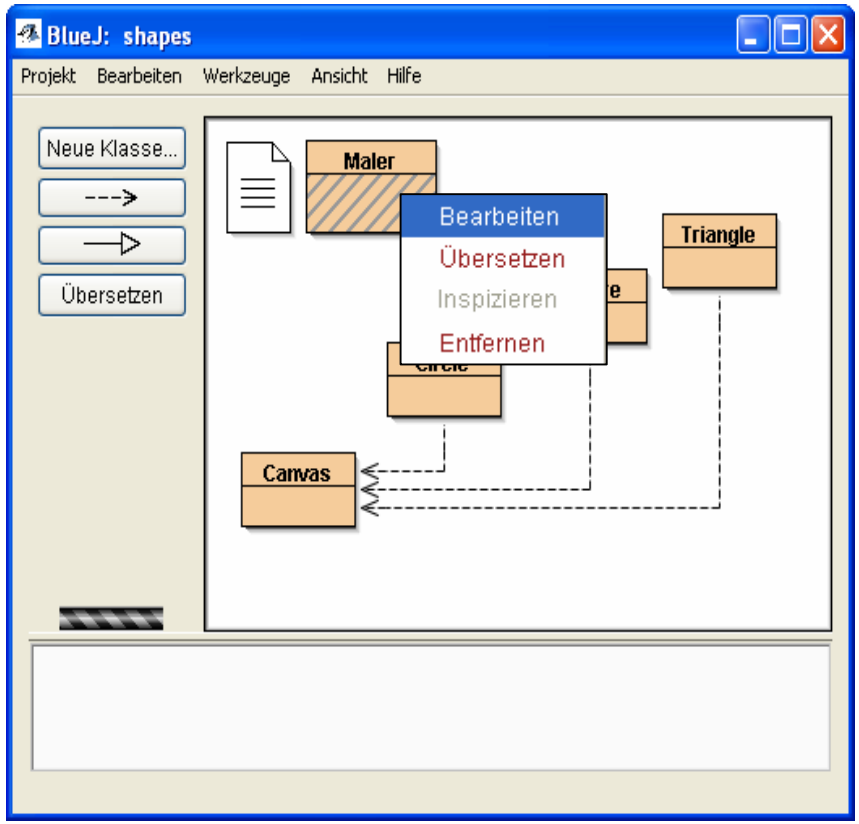
Bisher haben wir alle Manipulationen interaktiv erledigt.
Kann man das auch programmgesteuert machen ?

- Welche Aktionen waren notwendig:
  - Kreise erzeugen und Namen geben – z.B. kreis, circle_1, circle_2
  - Quadrate erzeugen und Namen geben – z.B. quadrat
  - Dreiecke erzeugen und Namen geben – z.B. triangle_1, triangle_2.
- Wie kann man das durch eine Java-Klasse erledigen ?
  - Wir benötigen eine Klasse um Bilder zu produzieren,
    - z.B. eine Klasse **Maler**



Neue Klasse: Maler

- Öffnen Sie in BlueJ das Projekt **shapes** und erzeugen Sie eine neue Klasse mit Namen **Maler**
- Editieren Sie diese



Die Klasse Maler

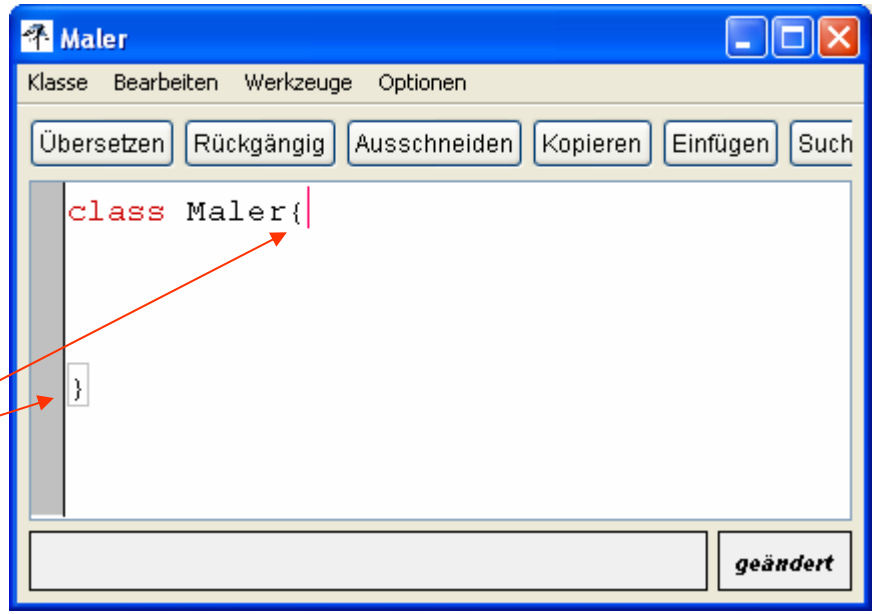
- Ausnahmsweise löschen wir alles, bis auf
  - die Zeile

```
class Maler
```

  - Und die geschweiften Klammern

```
{

}
```



Die Klammern sind die „Wände“ unserer Malerklasse. Dazwischen kommen die Instruktionen, wie ein Maler konstruiert wird.

Was braucht ein Maler

- Jeder Maler – d.h. jedes Objekt der Klasse Maler – muss ...
  - Kreise, Quadrate und Dreiecke erzeugen und jedem einen Namen geben
  - ... eine Methode z.B. `zeichneHaus( )` haben, die
    - die Objekte an die richtige Stelle verschiebt
    - sie vergrößert/verkleinert
    - sichtbar macht
    - ihre Farbe wechselt

Konstruktion der Objekte

- Die Erzeugung von Objekten besteht aus zwei Schritten
  - Ein Name für das Objekt wird registriert
  - Ein neues Objekt wird erzeugt und unter diesem Namen gespeichert

```
Circle sonne;
```

Registriere den Namen **sonne** für ein noch zu schaffendes Objekt der Klasse **Circle**.

```
sonne = new Circle( );
```

Konstruiere ein neues Objekt der Klasse Circle und nenne es **sonne**

Vorsicht:

Java unterscheidet zwischen Groß- und Kleinschreibung.

... CIRCLE ≠ Circle ≠ circle, sonne ≠ Sonne ≠ SOnne

Registrierung mit Erzeugung

- Jedes Objekt wird erzeugt und getauft. Wir können das auch in einem Schritt erledigen, z.B. ein neues Objekt der Klasse Circle erzeugen und diesem den Namen **sonne** geben:

```
Circle sonne = new Circle( );
```

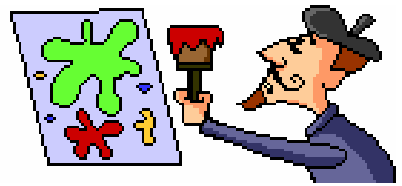
Registriere **sonne** als Name für ein Objekt der Klasse **Circle**

Ein neues konstruiertes Objekt der Klasse Circle.

```
=                          ;
```

Unter dem soeben registrierten Namen **sonne** wird das neu konstruierte Objekt der Klasse **Circle** abgespeichert.

Die Klasse Maler mit allen Objekten



Die Klasse lässt sich schon compilieren, und man kann bereits Objekte erzeugen...

Maler

Klasse Bearbeiten Werkzeuge Optionen

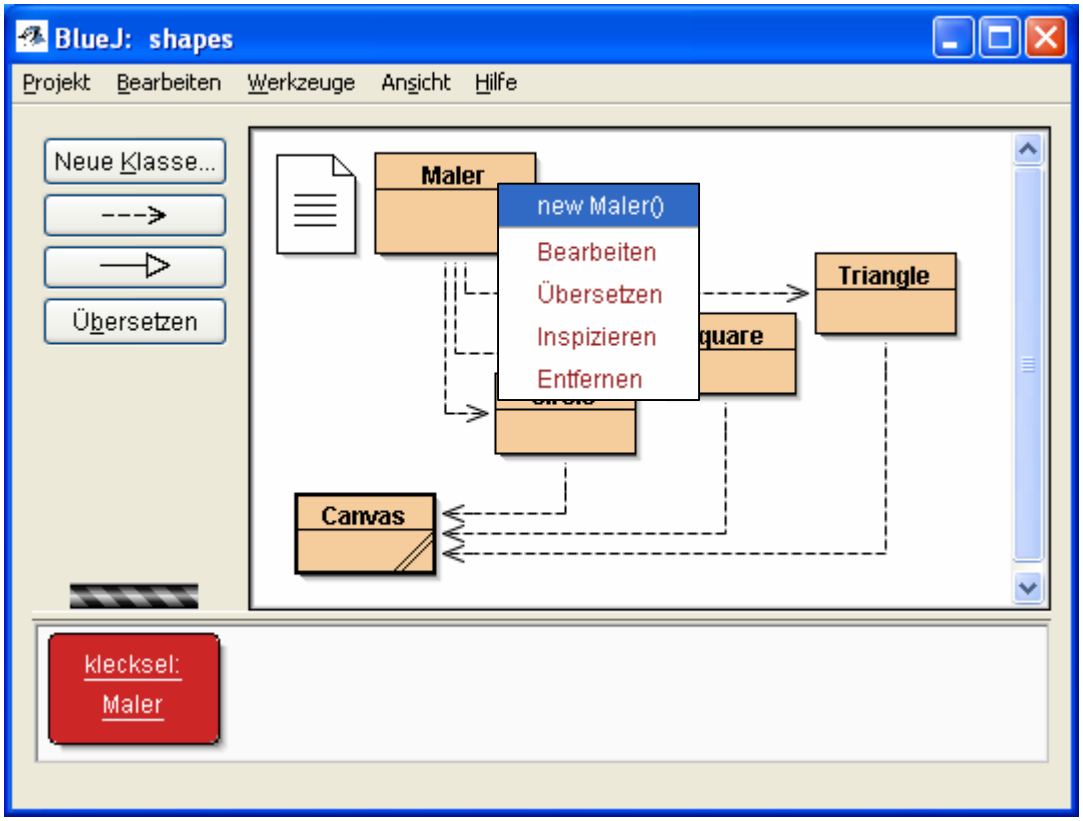
Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Sucher

```
class Maler
{
  Circle sonne = new Circle( );
  Square wand = new Square( );
  Square fenster = new Square( );
  Triangle dach = new Triangle( );
}
```

Klasse übersetzt - keine Syntaxfehler

gespeichert

Maler `klecksel`



Im Kontextmenü der Klasse Maler finden wir die Methode new und erzeugen `klecksel`

Dann inspizieren wir das Objekt `klecksel`

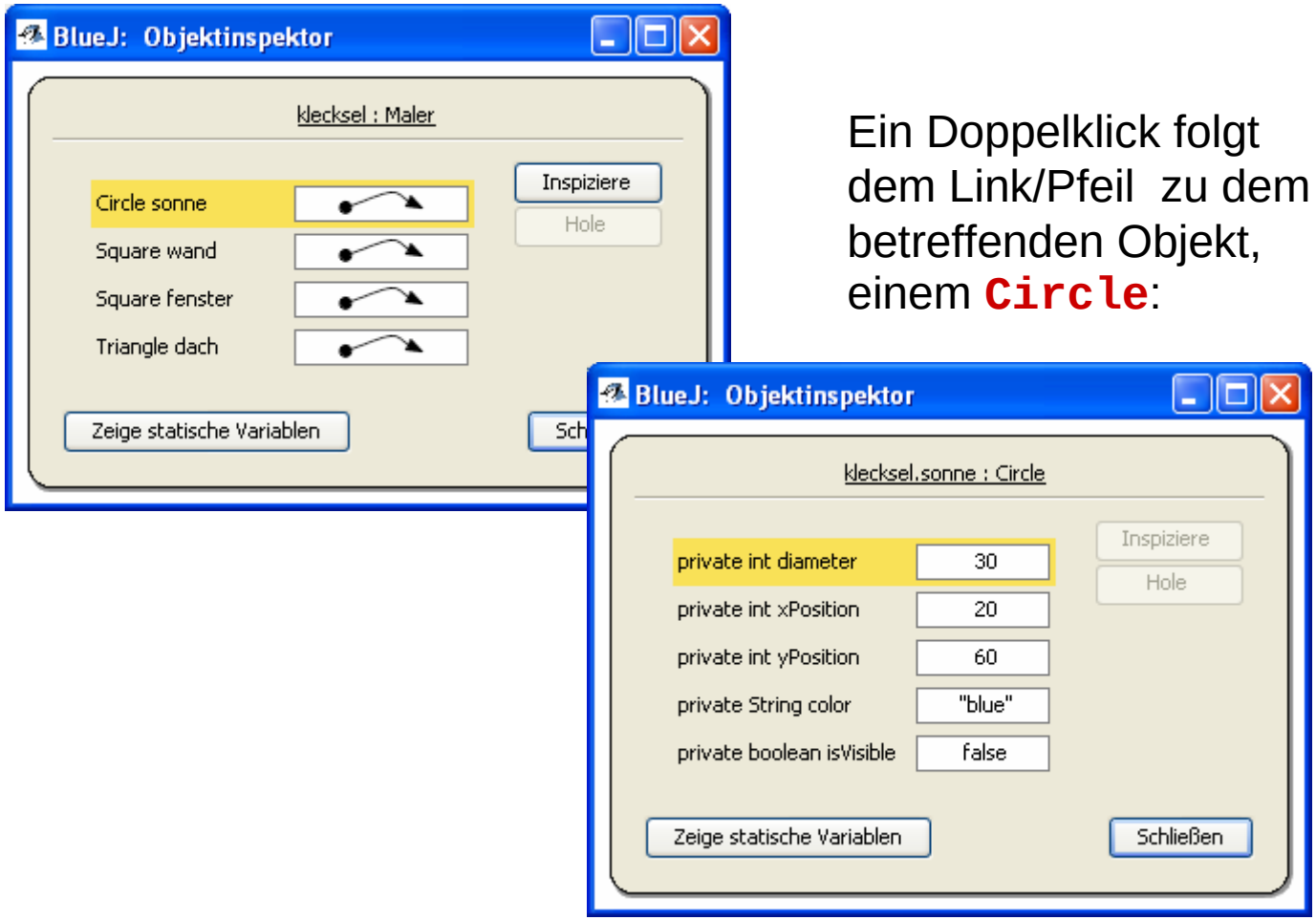
Inspektion von `klecksel`

Der Inspektor findet bei klecksel die Felder

- **`sonne`**
- **`wand`**
- **`fenster`**
- **`dach`**

Dies sind keine Zahlen, sondern Objekte der Klassen

- **`Circle`**
- **`Square`**
- **`Square`**
- **`Triangle.`**



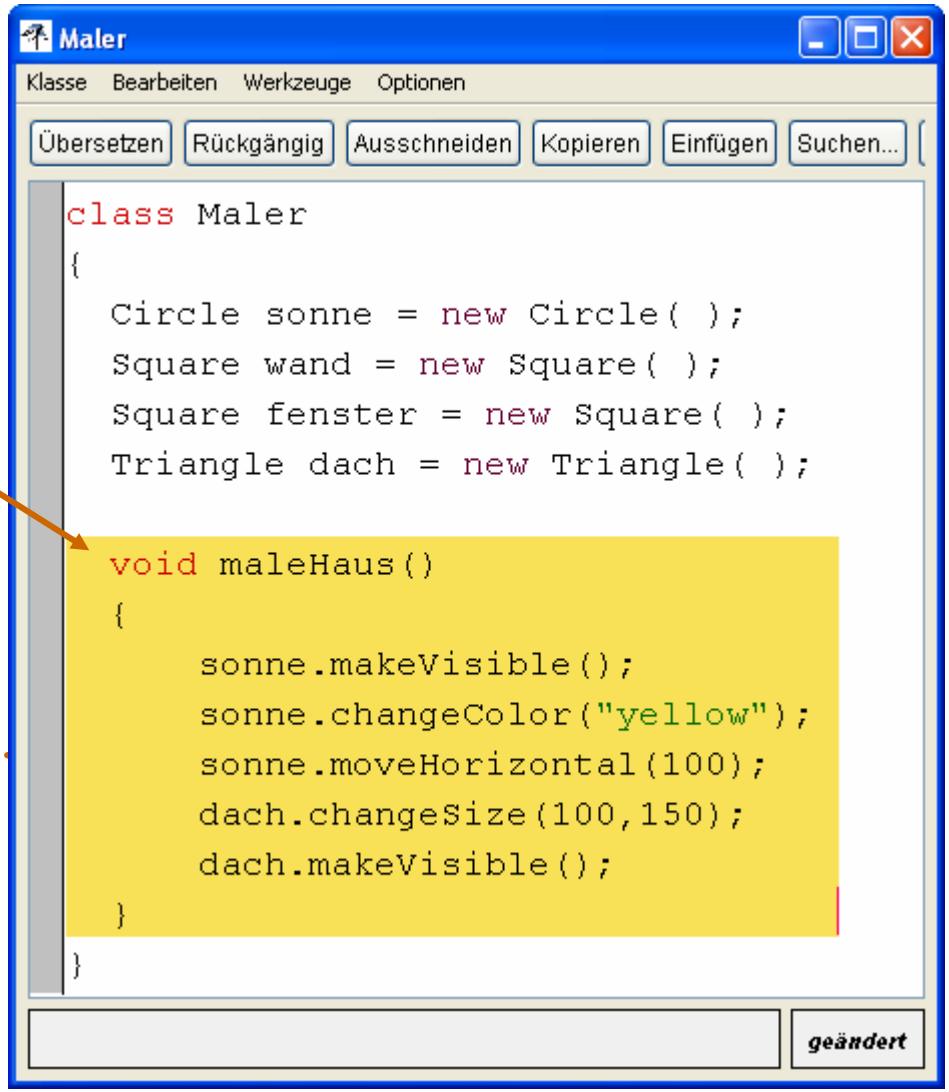
Ein Doppelklick folgt dem Link/Pfeil zu dem betreffenden Objekt, einem **`Circle`**:

Jetzt soll der Maler malen

Wir müssen dem Maler eine Methode zur Verfügung stellen, ein Haus zu malen.

- Eine Java-Methode hat stets einen Kopf bestehend aus
  - einen ***Ergebnistyp*** – hier **`void`**
  - einen ***Namen*** – z.B. **`maleHaus`**
  - eine ***Parameterliste*** - hier leer, also **`( )`**

- ... und einen Rumpf, in dem steht, was sie tun soll, z.B.
  - **`sonne.makeVisible();`**
  - **`sonne.changeColor("yellow");`**
  - **`sonne.moveHorizontal(100);`**
  - **`dach.changeSize(200);`**
  - **`dach.makeVisible();`**
  - ... etc. ...



Java-Methoden – der Kopf

- Der Kopf einer Java-Methode:

```
void maleHaus( )
```

Typ des Berechnungs-ergebnisses. Wir wollen keinen Wert berechnen, daher void

Selbstgewählter Name. Konvention: kleingeschrieben

Liste der Parameter
Darf nicht fehlen, auch wenn kein Parameter benötigt.

- Köpfe anderer Methoden

```
void maleAquarium( )
void makeVisible( )
void changeColor(String farbe)
void moveHorizontal(int distance)
```

Eine mögliche Methode für Maler

Eine Methode von von Circle, Square, ..

Eine Methode mit String Parameter

Eine Methode mit int Parameter

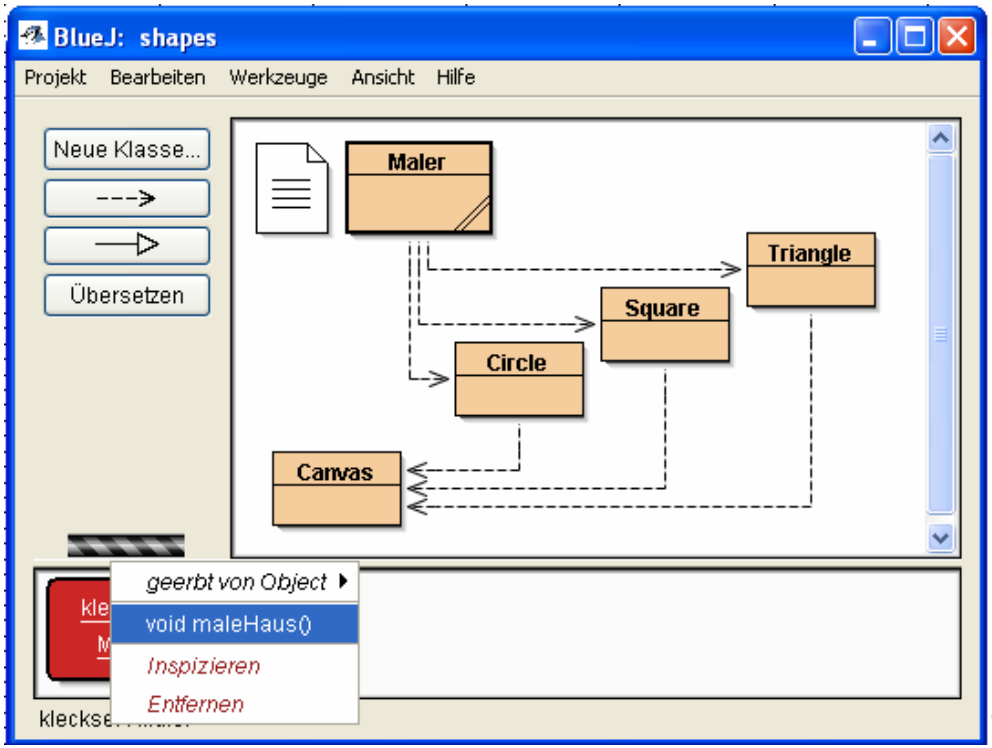
Java Methoden – der Rumpf

Der *Rumpf* einer Java-Methode heißt auch ***Block***. Er besteht aus

1. einer öffnenden Klammer : **{**
2. einer Folge von Anweisungen, z.B.:
   - Variablendeklarationen
   - Zuweisungen
   - Schleifen
   - Aufrufen von **void**-Methoden, z.B.:
     - `sonne.makeVisible();`
     - `dach.changeSize(100,150);`
     - `sonne.moveHorizontal(100);`
     - ...
3. einer schließenden Klammer: **}**

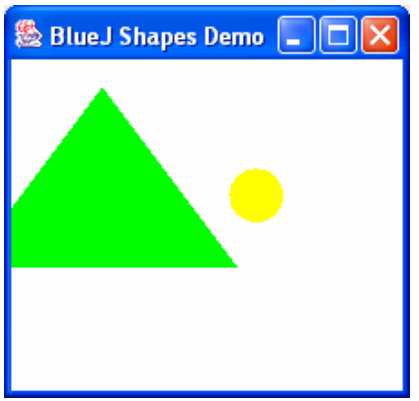
```
{
    sonne.makeVisible();
    sonne.changeColor("yellow");
    sonne.moveHorizontal(100);
    dach.changeSize(100,150);
    dach.makeVisible();
}
```

Erste Malversuche von `klecksel`



Wir erzeugen ein Objekt der Klasse **`Maler`** und nennen es **`klecksel`**.

Im Kontextmenü von **`klecksel`** finden wir die Methode **`void maleHaus( )`**
Und siehe da ...

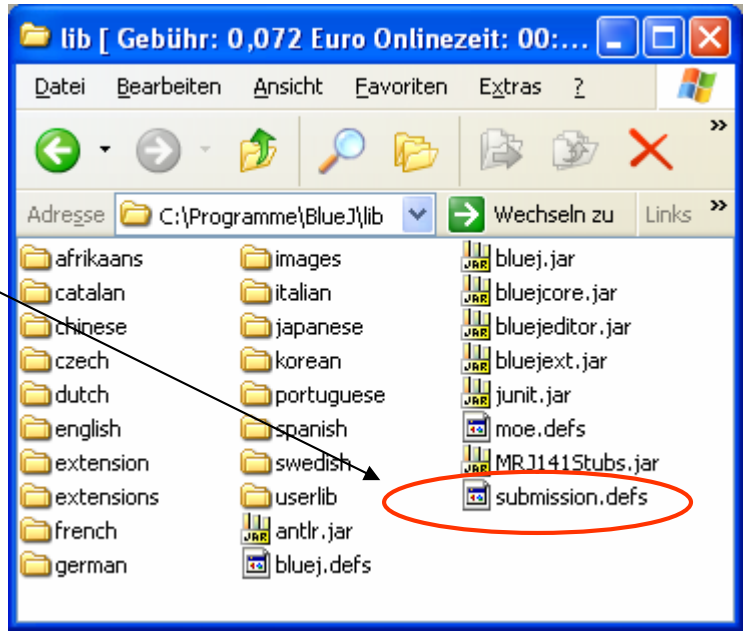


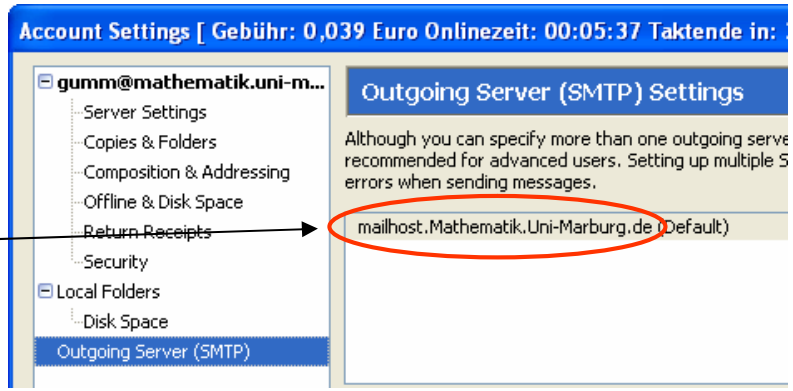
Na ja, für den Anfang geht's doch, oder .... ?

Hausaufgaben …

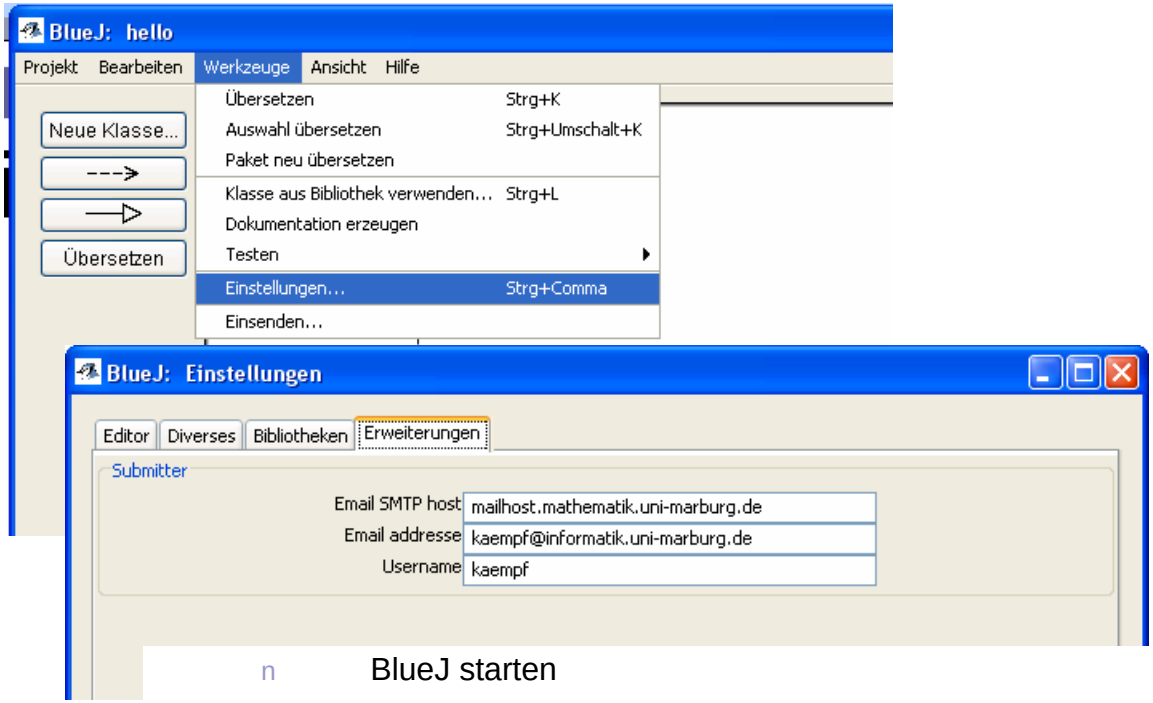
- … *werden **direkt aus BlueJ*** an Ihren Tutor geschickt

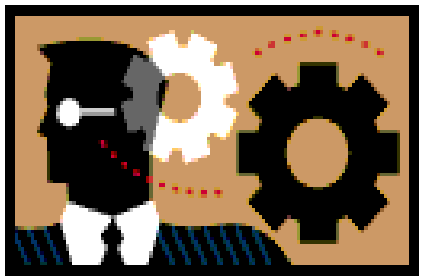
  - Einmalige Vorbereitung notwendig:
    - Datei submission.defs von Übungsseite laden
    - in Ihrem BlueJ\lib-Verzeichnis speichern
  - Adresse des *Postausgangsserver* Ihres email-providers herausfinden. Zum Beispiel:
    - aus dem Netz des Fachbereiches Informatik:
      - mailhost.mathematik.uni-marburg.de
    - aus T-Online:
      - mailto.t-online.de
    - aus web.de:
      - smtp.web.de
    - aus ….
  - Notfalls
    - in Ihrem Mailprogramm nachschauen
      - z.B. unter Tools/Account Settings
    - komplette Liste deutscher Anbieter unter www.patshaping.de/hilfen_ta/pop3_smtp.htm





Einstellungen

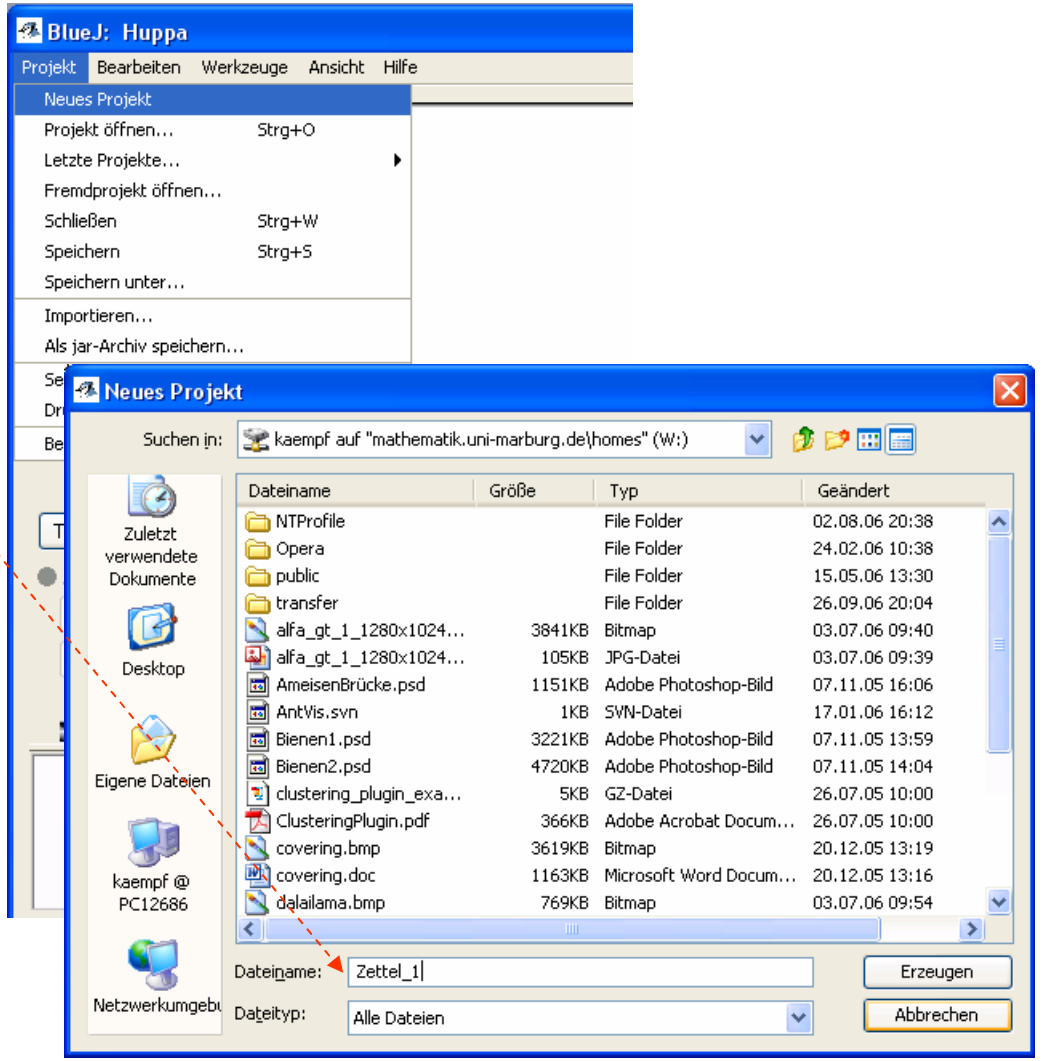




- BlueJ starten
- Unter Werkzeuge/Einstellungen/Erweiterungen eintragen:
  - Postausgangsserver(SMTP host)
  - Eigene E-Mail Adresse
  - Benutzernamen des Mail–Kontos
- Damit sind die Vorbereitungen abgeschlossen

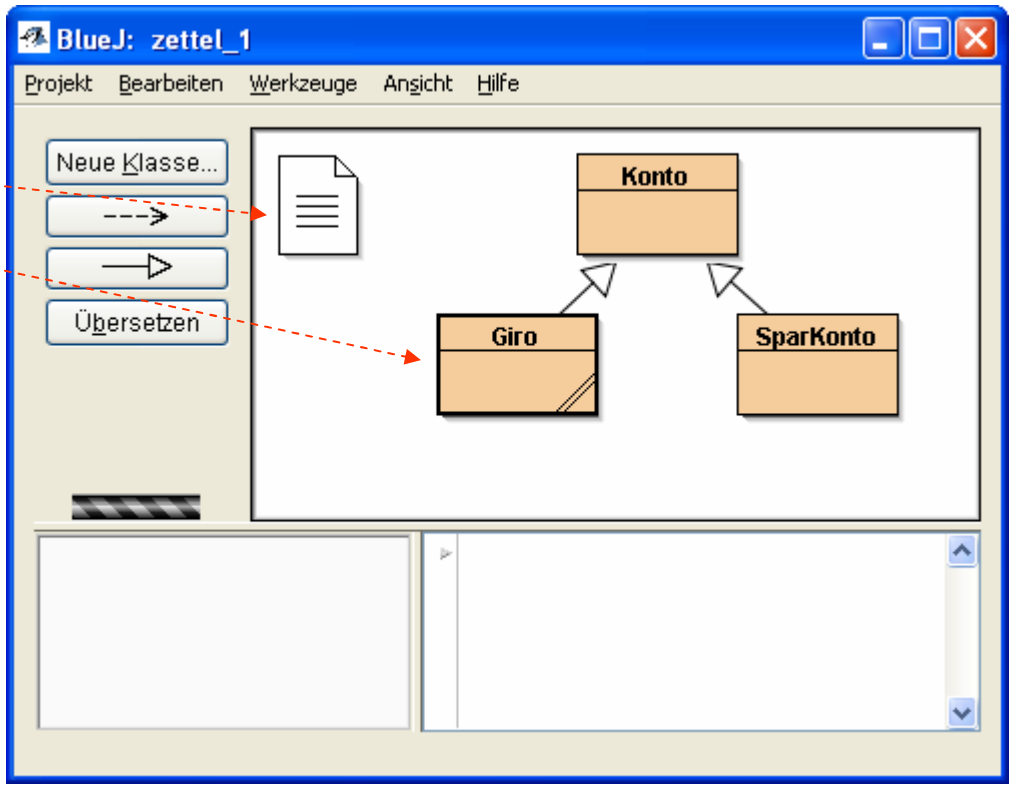
An die Arbeit …

- Erzeuge *BlueJ-Projekt*
  z.B. mit dem Namen Zettel_1:
  - *Neues Projekt* anklicken
  - *Projektnamen* wählen (z.B.: Zettel_1)
  - *Erzeugen* klicken
- BlueJ erstellt Verzeichnis mit Namen des Projekts
  - Hier: Zettel_1



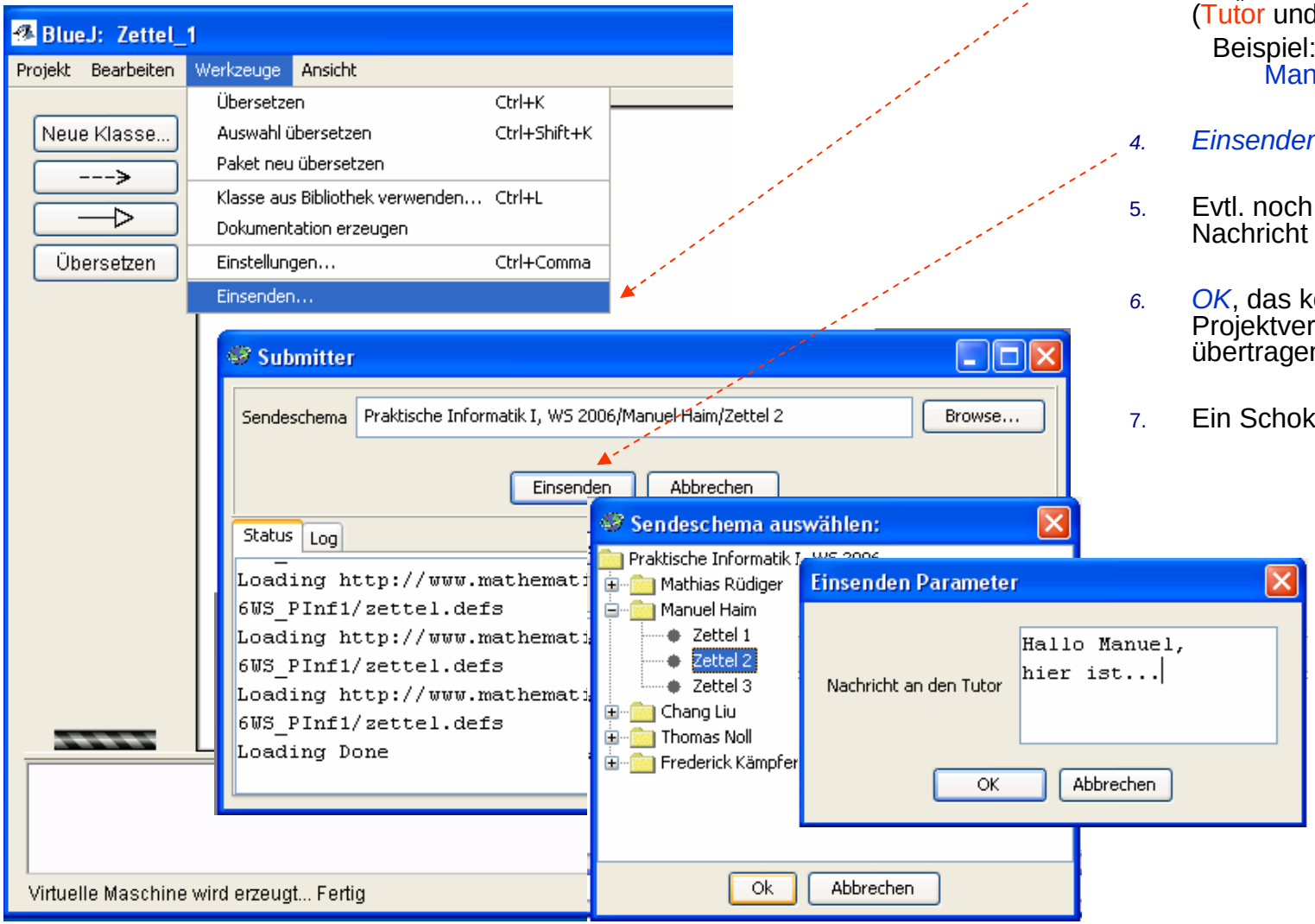
… an die Arbeit …

- Im Projektverzeichnis legt BlueJ alle zugehörenden Dateien ab
  - Eine Datei Readme.txt
  - Ihre Java-Klassen
  - sonstige Dateien
- Hier können Sie jetzt mit **BlueJ** die Lösung erstellen
  - Hauptsächlich: Java-Klassen programmieren
- Weitere Hinweise an den Tutor bitte in Readme.txt schreiben



Falls Sie weitere Dateien mitsenden wollen, speichern Sie diese in dem Projektverzeichnis

Lösung einsenden



1. Mit Internet verbinden
2. *Werkzeuge/Einsenden*
3. Mit „*Browse*“ Sendeschema (Tutor und Zettel) auswählen.
   Beispiel:
   Manuel Haim/Zettel 2
4. *Einsenden*
5. Evtl. noch eine kurze Nachricht an den Tutor
6. *OK*, das komplette Projektverzeichnis wird übertragen
7. Ein Schokoladeneis essen