

Wo ist eigentlich was?

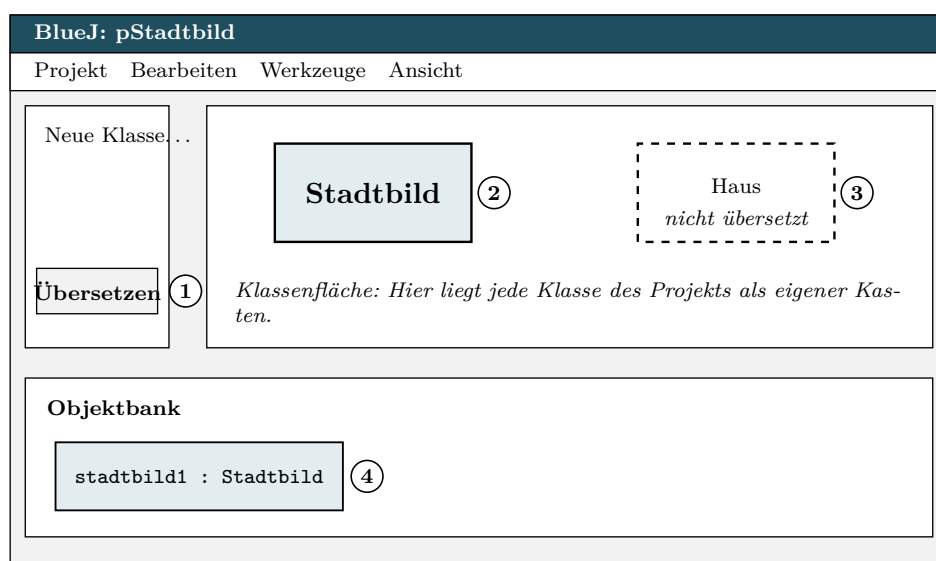
BlueJ, die Bibliothek *Stifte und Mäuse* und der Aufbau einer Klasse

Doppelstunde am Rechner

BlueJ ist keine besonders hübsche Entwicklungsumgebung, aber eine ehrliche: Man sieht darin sofort, was in der objektorientierten Programmierung eigentlich passiert. Klassen sind Kästen. Objekte liegen unten in einer Ablage und lassen sich anfassen. Wer eine Methode ausprobieren will, klickt sie an — ganz ohne Hauptprogramm.

Genau darum arbeiten wir damit. Diese Stunde geht es nur ums Verstehen der Oberfläche und darum, wie eine Java-Klasse von innen aussieht. Ab nächster Woche baut ihr euer erstes eigenes Bild.

Das Fenster



- 1 **Übersetzen.** Java-Quelltext muss vor jedem Start übersetzt werden. Solange ein Kasten gestreift dargestellt ist, läuft nichts. Fehlermeldungen erscheinen unten im Editorfenster — immer die *oberste* zuerst lesen, die anderen sind oft Folgefehler.
- 2 **Eine übersetzte Klasse.** Doppelklick öffnet den Editor. *Rechtsklick* zeigt den Konstruktor: `new Stadtbild()`. Genau so entstehen Objekte.
- 3 **Eine noch nicht übersetzte Klasse.** Erst nach dem Übersetzen lassen sich Objekte davon erzeugen.
- 4 **Ein Objekt.** Es trägt einen selbst vergebenen Namen und den Namen seiner Klasse. Rechtsklick darauf listet alle `public`-Methoden auf, und jede lässt sich einzeln anklicken — ohne Hauptprogramm. Das ist der schnellste Weg, etwas auszuprobieren.

Der wichtigste Handgriff: Rechtsklick auf einen Klassenkasten → `new Stadtbild()`. Damit entsteht ein Objekt, das unten in der Objektbank landet. Rechtsklick auf dieses Objekt zeigt alle Methoden, die es hat. Ihr könnt jede einzeln aufrufen und sofort sehen, was passiert. Kein Hauptprogramm, kein Umweg.

Wie eine Klasse von innen aussieht

Jede Klasse, die ihr in diesem Halbjahr schreibt, hat denselben Grundriss. Es lohnt sich, ihn einmal richtig anzuschauen, weil danach jede Fehlermeldung halb so rätselhaft ist.

```
import sum.kern.*; // 1 Bibliothek dazuholen

public class Stadtbild // 2 Name der Klasse - genau wie die Datei
{
    Bildschirm derBildschirm; // 3 Attribute: was das Objekt besitzt
    Buntstift meinStift;
```

```
public Stadtbild() // 4 Konstruktor
{
    derBildschirm = new Bildschirm(1000, 700);
    derBildschirm.nachVorn();
    meinStift = new Buntstift();
}

public void zeichneAlles() // 5 Methoden: was das Objekt kann
{
    // ...
}
}
```

- 1 Ohne diese Zeile kennt Java weder **Bildschirm** noch **Buntstift**. Die zweite Zeile `import sum.werkzeuge.*;` braucht ihr erst, wenn **Uhr** oder **Rechner** dazukommen.
- 2 Der Klassenname beginnt groß und muss exakt so heißen wie die Datei. Wer die Klasse umbenennt, muss auch die Datei umbenennen — BlueJ macht das beim Umbenennen im Editor mit.
- 3 **Attribute** stehen *außerhalb* aller Methoden, direkt unter der geschweiften Klammer der Klasse. Sie leben so lange wie das Objekt. Alles, was das Objekt sich dauerhaft merken soll, gehört hierhin.
- 4 Der **Konstruktor** heißt *genau* wie die Klasse und hat **kein void**. Er läuft ein einziges Mal: in dem Moment, in dem das Objekt entsteht. Seine Aufgabe ist, die Attribute mit Leben zu füllen — hier also Bildschirm und Stift wirklich zu erzeugen.

Steht dort versehentlich `public void Stadtbild()`, ist es kein Konstruktor mehr, sondern eine gewöhnliche Methode, die zufällig so heißt. BlueJ bietet beim Rechtsklick dann kein *new* mit Parametern an. Daran merkt man es.
- 5 Alles Weitere sind **Methoden**. Sie stehen *nebeneinander*, niemals *ineinander*. **public** heißt: von außen aufrufbar und im Rechtsklickmenü sichtbar. **private** heißt: nur die Klasse selbst darf das.

Zum Ausprobieren

Ihr arbeitet ab heute in **Zweierteams** an einem Rechner. Wechselt euch dabei ab — wer tippt, tippt zehn Minuten, dann wird getauscht. Wer nicht tippt, liest mit und denkt mit. Das ist keine Schikane: Wer nur zuschaut, kann es zwei Wochen später nicht.

1 Ein Projekt anlegen

Legt in eurem persönlichen Verzeichnis ein Projekt `pStadtbild` an und darin eine Klasse `Stadtbild`. Tippt das Gerüst von der Vorderseite ab — *abtippen*, nicht kopieren. Ihr werdet dabei mindestens einen Tippfehler machen, und genau dessen Fehlermeldung wollen wir sehen.

2 Übersetzen und erzeugen

Drückt *Übersetzen*. Wenn die Schraffur verschwindet, ist alles in Ordnung. Erzeugt dann per Rechtsklick ein Objekt. Es erscheint ein Fenster, in dem ihr zeichnen könnt — noch leer.

3 Den Stift anfassen

Schreibt in `zeichneAlles()` ein paar Zeilen und probiert aus, was passiert. Ideen zum Anfangen:

```
meinStift.runter();  
meinStift.bewegeBis(400, 300);  
meinStift.zeichneKreis(50);  
meinStift.setzeFarbe(Farbe.ROT);  
meinStift.zeichneRechteck(120, 60);
```

Ruft die Methode über den Rechtsklick auf euer Objekt auf. Ändert Zahlen, tauscht Zeilen, lasst etwas weg. Ihr könnt nichts kaputt machen.

Notiert unterwegs drei Beobachtungen, die euch überrascht haben. Egal wie klein.

4 Fehler provozieren

Baut absichtlich Fehler ein und schaut euch an, was BlueJ dazu sagt. Notiert die Meldung in Stichworten — diese vier Meldungen werdet ihr dieses Halbjahr noch oft sehen.

Das baut ihr ein	Was BlueJ sagt
Ein Semikolon am Zeilenende weglassen	
<code>meinStift</code> zu <code>meinstift</code> ändern	
Eine schließende Klammer <code>}</code> löschen	
Die <code>import</code> -Zeile löschen	

Und die eine Sache, die man nicht rät: Löscht das `new Buntstift()` aus dem Konstruktor, übersetzt neu und ruft die Methode auf. Das Programm übersetzt sauber und stürzt trotzdem ab. Was steht in der Meldung?

Merken für den Rest des Halbjahres. Ein Attribut zu *deklarieren* (`Buntstift meinStift;`) und es zu *erzeugen* (`meinStift = new Buntstift();`) sind zwei verschiedene Dinge. Das erste sagt nur, dass es das Attribut geben soll — das zweite füllt es. Wer das zweite vergisst, bekommt beim Ausführen eine `NullPointerException`. Das ist die mit Abstand häufigste Fehlermeldung im Anfängerjahr.

Nächste Woche geht es los mit dem Stadtbild. Bringt mit: ein Foto oder eine klare Erinnerung an einen Ort, den ihr zeichnen wollt. Es muss nicht Kreuztal sein.