Ziele des Informatikunterrichts in der Einführungsphase

Der Informatikunterricht in der Einführungsphase soll Dir einen Überblick über die wichtigsten Inhalte und Kompetenzen des Fachs Informatik geben. Alle Inhalte und Kompetenzen werden in der Qualifikationsphase wieder aufgegriffen und vertieft.

Ein Hauptaugenmerk wird in der Einführungsphase auf das Erlernen einer objektorientierten Programmiersprache gelegt. Eine solche Programmiersprache ist nötig, um informatische Aufgabenstellungen nicht nur theoretisch sondern auch praktisch lösen zu können – ganz ähnlich wie Du z. B. in der Mathematik nicht nur eine Formel bestimmst, sondern oftmals auch mit dem Taschenrechner konkrete Lösungen berechnest.

Java ist eine solche objektorientierte Programmiersprache. Mit Java kannst Du leistungsfähige Programme in ansprechendem, plattformunabhängigem Design erstellen. Z. B. lassen sich mit Java sogenannte Applets programmieren, welche auf einer Internetseite eingebunden werden können.

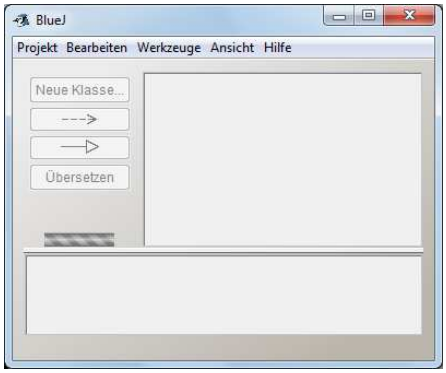
Im den folgenden Kapiteln sollen Dir die wesentlichen Befehle und grundlegenden Konzepte dieser Programmiersprache und deren Grundlagen (Hardware, Maschinensprache) näher gebracht werden. Zudem gibt es für Java zahlreiche Programmierumgebungen. In diesem Kurs wirst Du anfangs mit den Programmierumgebungen BlueJ und Greenfoot und später mit dem Java-Editor[1] arbeiten.

[1] **Installation von BlueJ:**
Es werden die Installationsdateien von BlueJ sowie der Software-Development-Kit (JDK) benötigt:
Quelle: BlueJ: http://www.bluej.org
Installation von Greenfoot:
Es wird zusätzlich die Installationsdatei von Greenfoot benötigt:
Quelle: Greenfoot: http://www.greenfoot.org
Installation des Java-Editors:
Es wird zusätzlich die Installationsdatei des Java-Editors benötigt.
Quelle: Editor: http://www.javaeditor.org/doku.php?id=en:download

Grundlagen der objektorientierten Analyse und Programmierung

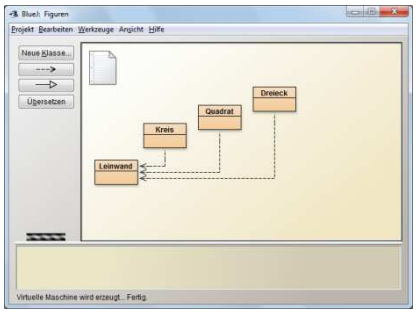
Klassen und Objekte in Java

Wenn Du BlueJ startest, so erhältst Du zunächst ein leeres Fenster, in dem lediglich die Menüleiste aktiviert ist.



Wir wollen zu Anfang mit einem schon fertigen Programm arbeiten, um uns die Funktionsweise der Programmiersprache Java besser verdeutlichen zu können.

Aufgabe: Öffne das BlueJ-Projekt *Figuren*. (Menübefehl *Projekt*|*Projekt öffnen...*).



BlueJ zeigt Dir vier **Klassen** für ein Dreieck, ein Quadrat, einen Kreis und eine Leinwand. Die Pfeile machen deutlich, dass sich alle Klassen auf die Leinwand beziehen, d. h., alle Objekte der Klassen werden auf einer Leinwand gezeichnet.

Die Klasse Kreis stellt hier allerdings nur den grundsätzlichen "Bauplan" eines Kreises dar. Wenn wir ein **Objekt** der Klasse Kreis haben wollen, so müssen wir dieses zuerst erzeugen.

Aufgabe: Drücke die rechte Maustaste auf die Klasse Kreis und erzeuge einen neuen Kreis mit der Methode *new Kreis()*. Gib dem Kreis einen sinnvollen Namen, z. B. *ball*. (Achtung: Gib Objekten immer einen Namen, der mit einem kleinen Buchstaben anfängt). Am unteren Fensterrand wird Dir nun Dein soeben erstelltes Objekt als roter Kasten angezeigt.

Wir kommen jetzt direkt zu einer der wichtigsten Unterscheidungen in der Programmiersprache Java. Wenn wir von **Klassen** sprechen, so meinen wir die **gelben Kästen**. Wenn wir von **Objekten** sprechen, so meinen wir die **roten Kästen**.

Eine **Klasse** stellt lediglich Informationen zur Verfügung, welche Eigenschaften (**Attribute**) und welche Fähigkeiten (**Methoden**) Objekte der Klasse haben. Z. B. würde eine Klasse *Auto* festlegen, dass Objekte der Klasse Auto immer genau vier Räder besitzen, in einer beliebigen Farbe lackiert sind und ein Kennzeichen besitzen. Darüber hinaus kann ein Auto z. B. sein Gewicht durch Zu- oder Entladung ändern.

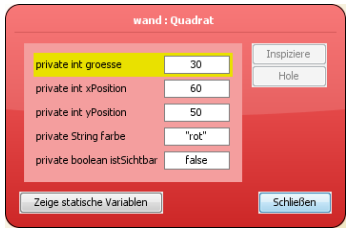
Ein **Objekt** der Klasse Auto würde dagegen konkrete Angaben zu Farbe und Kennzeichen machen. Z. B. würde das Objekt *golf* der Klasse Auto die Farbe rot und das Kennzeichen K-XY-456 haben.

In unserem Fall haben wir eben ein Objekt *ball* erzeugt, welches wir nun auf der Leinwand zeichnen lassen wollen.

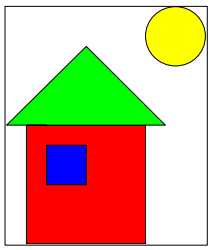
Aufgabe: Wähle aus dem Kontextmenü des Objekts *ball* den **Methode**naufruf *void sichtbarMachen()*. BlueJ erzeugt ein neues Objekt der Klasse Leinwand und zeichnet darauf Deinen selbst erstellten Kreis.

Aufgabe: Experimentiere mit den anderen **Methoden**, die das Objekt *ball* zur Verfügung stellt. Manchmal musst Du eine Zahl (z. B. *int entfernung*) eingeben, manchmal musst Du eine Zeichenkette (z. B. *String neueFarbe*) eingeben. Man nennt diese Eingaben auch **aktuelle Parameter**. Achte darauf, dass Du Zeichenketten stets in Anführungszeichen setzt (also z. B. "rot"). Experimentiere auch einmal mit negativen Eingabewerten.

Aufgabe: Erzeuge Objekte der Klassen Dreieck und Quadrat. Informiere dich über die intern gespeicherten Informationen eines Kreises, eines Dreiecks bzw. eines Quadrats. Verwende dafür den Kontextmenübefehl *Inspizieren*. Achte auf die verschiedenen **Datentypen** *int*, *String* und *boolean*. Welche Werte können in einem *int*, welche in einem *String* und welche in einem *boolean* gespeichert werden?



Aufgabe: Lösche alle vorhandenen Objekte (Kontextmenübefehl *Entfernen*). Erzeuge anschließend ein Bild von einem Haus, so wie es rechts dargestellt ist. Achte auf eine sinnvolle Bezeichnung der Objekte.



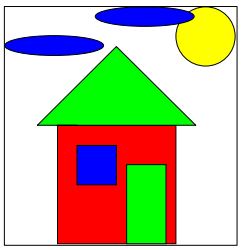
Die erste eigene Klasse

Es wäre schön, wenn wir eine rechteckige Tür einbauen könnten. Dafür benötigen wir allerdings eine neue Klasse Rechteck. Diese wollen wir einmal Schritt für Schritt neu erstellen.

Aufgabe: Führe folgende Schritte durch.

1. Erzeuge eine neue Klasse Rechteck (Button Neue Klasse...).
2. Kopiere die gesamten Programmzeilen der Klasse Quadrat in die Zwischenablage (Doppelklick auf die Klasse Quadrat, mit Strg+A alles markieren und mit Strg+C die markierten Programmzeilen kopieren).
3. Ersetze in der Klasse Rechteck den vorhanden Quelltext durch den Text in der Zwischenablage (mit Strg+A alles markieren und mit Strg+V die zuvor kopierten Programmzeilen anstelle der nun markierten einfügen).
4. Ersetze in dem Quelltext nun jedes Wort "`Quadrat`" durch das Wort "`Rechteck`".
5. Bisher gibt es nur ein Attribut `int groesse`, welches du ganz oben im Quelltext findest. Nenne dieses `groessex` und füge ein neues Attribut `int groessey` hinzu.
6. Benenne überall im Quelltext den Namen `groesse` in `groessex` um und füge eine sinnvolle Programmzeile für `groessey` hinzu. Im Einzelnen ist dies an folgenden Stellen nötig:
   Das erste Mal im so genannten Konstruktor *Rechteck()*, der immer aufgerufen wird, wenn ein neues Objekt der Klasse erzeugt wird. Setze `groessey` hier auf den festen Wert 60.
   Das zweite Mal in der Methode *groesseAendern(...)*. Benenne den Parameter `neueGroesse` auch in `neueGroessex` um und setze `groessey = 2*neueGroessex`.[2]
   Das dritte Mal in der Methode *zeichnen()*. Benenne den ersten Wert mit `groessex` und den zweiten mit `groessey`.
7. Übersetze den geänderten Quelltext.
8. Teste Deine Klasse, indem Du ein Objekt dieser Klasse erzeugst und davon die Methode *void sichtbarMachen()* aufrufst.

Aufgabe: Estelle eine neue Klasse Oval, mit der man später Wolken zeichnen könnte. Verfahre wie beim Rechteck, kopiere jetzt allerdings den Quelltext der Klasse *Kreis* in die Zwischenablage und verändere diesen so, dass ein Oval gezeichnet wird.
Erstelle anschließend Objekte und verändere deren Eigenschaften so, dass das rechts abgebildete Haus entsteht.



[2] Wenn Du schon Profi bist, so kannst Du auch einen neuen Parameter *int neueGroessey* im Kopf der Methode einfügen. Somit könnte man dann das Rechteck beliebig strecken.

Objekte von Objekten erzeugen lassen

Die letzte Aufgabe dauerte vermutlich schon ganz schön lange, bis man endlich alle Objekte an die richtige Stelle geschoben hatte, oder? Es wäre einfacher, wenn wir die Erzeugung der Objekte und deren Verschiebungen und Änderungen mit einem einzigen Befehl ausführen könnten. Wie dies geht, schauen wir uns nun im Projekt ***Zeichnung*** an.

Aufgabe: Öffne das BlueJ-Projekt *Zeichnung* und schaue Dir den Quelltext der Klasse *Zeichnung* genauer an:

1. Am Anfang der Klasse wird festgelegt, aus welchen Objekten eine Zeichnung besteht (hier: wand, fenster, dach, sonne). Dies sind die so genannten **Attribute**.
2. Im **Konstruktor** (`public class Zeichnung`) wird festgelegt, wie die Objekte auszusehen haben. Der Befehl `wand = new Quadrat();` erstellt dabei ein neues Objekt der Klasse Quadrat und nennt dieses *wand*. Die Methoden, die wir zuvor mit der rechten Maustaste ausgewählt haben, werden hier einfach mit einem Punkt getrennt hinter den Objektnamen geschrieben. Der Befehl `wand.vertikalBewegen(80)` verschiebt das Objekt mit dem Namen *wand* um 80 Pixel nach unten.
3. In der **Methode** `public void zeichne()` werden nun nur noch der Reihe nach alle Objekte sichtbar gemacht.

Aufgabe: Teste die Klasse Zeichnung, indem Du ein neues Objekt dieser Klasse erzeugst und die Methoden ausprobierst. Was macht die Methode *inSchwarzWeissAendern()* und wie könnte dies programmiertechnisch umgesetzt werden? Schau Dir zur Kontrolle den Quelltext noch einmal an.

Aufgabe: Füge dem Projekt Deine zuvor erstellten Klassen Rechteck und Oval hinzu (Menübefehl *Bearbeiten*|*Klasse aus Datei hinzufügen...*). Erweitere die Zeichnung dann um eine Tür und zwei Wolken, so wie in der letzten Aufgabe im Projekt *Figuren*.

Zusammenfassung und Vertiefung

So, dies war schon mal das erste große Projekt. Ganz nebenbei hast Du wichtige Erkenntnisse über Java gewonnen:

- Du kennst den Unterschied zwischen *Klasse* und *Objekt*.
- Du kannst den *Quelltext* einer Klasse ändern und Objekte dieser Klasse mit unterschiedlichen *Eigenschaften/Attributen* erzeugen.
- Du weißt, was ein *Attribut*, ein *Konstruktor* und eine *Methode* sind. Du weißt außerdem, dass Methoden mit *Parametern* aufgerufen werden können.
- Du hast bereits die wichtigsten *Datentypen* `int`, `String` und `boolean` verwendet.
- Du kannst Objekte im Programm erzeugen und deren Attribute durch Methodenaufrufe ändern.

Im Projekt *Zeichnung* wurden etliche neue Konzepte diskutiert, welche nun vertieft betrachtet werden sollen. Dazu verwenden wir das Projekt ***Laborkurse***, das Projekt *Zeichnung* solltest Du nun schließen.

Aufgabe: Öffne das BlueJ-Projekt *Laborkurse* und erzeuge mehrere Objekte der Klasse *Student* (man nennt diese auch **Instanzen** der Klasse *Student*). Neben dem Namen des Objekts wirst Du auch nach weiteren Parametern gefragt. Fülle die Felder sinnvoll aus. Inspiziere die erstellten Objekte (rechte Maustaste auf die roten Objekte und dann *Inspizieren*).

Wenn nun die Methode *gibName()* eines Studenten aufgerufen wird, so stellt man fest, dass BlueJ ein Ergebnis zurückgibt. Dies liegt daran, dass diese Methode einen *String* als Ergebnis zurückgeben möchte. Du erkennst dies an der **Signatur** der Methode:

```
public String gibName().
```

Im Gegensatz dazu schauen wir uns die Signatur der Methode *nameAendern()* genauer an:

```
public void nameAendern(String neuerName)
```

void bedeutet, dass kein Ergebnis zurückgeliefert wird. Stattdessen wird ein Parameter *neuerName* vom Datentyp *String* erwartet.

Methoden, welche ein Ergebnis zurückgeben, werden **sondierende Methode** genannt. Sie dienen in der Regel dazu, eine bestimmte Eigenschaft eines Objekts auszulesen.

Dagegen werden Methoden, denen ein Parameter zur Veränderung einer bestimmten Eigenschaft eines Objekts mitgegeben wird, **verändernde Methode** genannt. Diese beiden Begriffe werden Dir in den folgenden Aufgaben noch häufiger begegnen, doch vorher sollst Du mit den unterschiedlichen Methoden experimentieren.

Aufgabe: Erzeuge ein Objekt der Klasse Laborkurs. Experimentiere mit den Methoden und schaue Dir jeweils die Signaturen genau an. Welche Besonderheit fällt Dir bei der Signatur zur Methode *trageStudentEin(...)* auf?

Um Java-Programme schreiben zu können, müssen wir lernen, wie man Klassendefinitionen mit Attributen und Methoden schreibt und wie man diese Klassen geeignet miteinander kombiniert. Um dieses Ziel zu erreichen, müssen wir uns einerseits mit der Syntax und den Sprachelementen von Java beschäftigen, andererseits Strategien entwickeln, wie Problemstellungen analysiert werden können.

Dazu wird ein weiteres, neues Projekt betrachtet, der *Ticketautomat*.

Klassendefinitionen und einfache Kontrollstrukturen

Der Ticketautomat

Auf Bahnhöfen und U-Bahnstationen stehen üblicherweise Ticketautomaten, die ein Ticket ausdrucken, sobald ein Kunde einen korrekten Betrag eingeworfen hat. Das Projekt ***Ticketautomat*** simuliert eine stark vereinfachte Version eines solchen Automaten, der in einigen Details noch stark von realen Ticketautomaten abweicht. Zunächst soll daher das Verhalten des Ticketautomaten untersucht werden und später aus den daraus gewonnenen Erkenntnissen eine verbesserte Version der Klasse Ticketautomat entwickelt werden.

Aufgabe: Öffne das BlueJ-Projekt *Ticketautomat* in BlueJ und erzeuge eine Instanz von *Ticketautomat* (dazu musst Du den Ticketpreis in Cent angeben). Untersuche das Verhalten des Objekts, wenn Du verschiedene Methoden aufrufst. Ist das Verhalten des Automaten realistisch? Was passiert, wenn z. B. zu viel Geld eingeworfen wird? Was passiert, wenn nicht genügend Geld eingeworfen wird und trotzdem ein Ticket gedruckt wird?

Untersuchung der Klassendefinition

Betrachte den Quelltext der Klasse *Ticketautomat*. Der Quelltext kann in zwei Hauptbestandteile unterteilt werden: eine schmale äußere Klammer, welche die Klasse benennt, und den meist umfangreicheren Innenteil, der die ganze Arbeit erledigt. Hier sehen die äußeren Klammern so aus:

```
public class Ticketautomat
{
  //  Innenteil der Klasse, hier ausgelassen
}
```

Im Innenteil einer Klasse werden die Attribute, Konstruktoren und Methoden definiert, die den Instanzen der Klasse ihre Struktur und ihr Verhalten geben.

Attribute: Speichern die Daten und Eigenschaften des Objekts.

Konstruktoren: Setzen neu erzeugte Objekte in einen vernünftigen Anfangszustand.

Methoden: Regeln das Verhalten der Objekte.

Im Folgenden wollen wir kurz die einzelnen Bestandteile einer Klasse beschreiben:

Attribute oder auch Instanzvariablen:

Ein Ticketautomat muss über drei Informationen verfügen, welche in den jeweiligen Attributen gespeichert werden:

1. der festgelegte Preis eines Tickets (`preis`)
2. der bereits eingeworfene Geldbetrag (`bisherGezahlt`)
3. die Summe aller Geldbeträge, die eingeworfen wurde (`gesamtsumme`)

Das Schlüsselwort `private` sagt aus, dass das Attribut eine private Eigenschaft des Objekts ist und nicht beliebig geändert werden darf. Schließlich soll nicht jeder einfach den Preis des Ticketautomaten ändern dürfen.

Es ist sinnvoll, jedem Attribut einen Kommentar hinzuzufügen, der beschreibt, was im Attribut gespeichert wird. Einzeilige Kommentare werden mit `//` eingefügt. Ausführlichere Kommentare, die über mehrere Zeilen reichen, werden üblicherweise mit

Mehrzeilenkommentaren vorgenommen. Diese beginnen mit dem Zeichenpaar `/*` und enden mit dem Paar `*/`.

Aufgabe: Welchen Typ haben die folgenden Attribute Deiner Meinung nach? Welchen Namen haben sie?
- *private int zaehler;*
- *private Student sprecher;*
- *private Server zentral;*
- *private boolean erledigt;*
- *private Spiel spiel;*

Konstruktoren

Konstruktoren sind dafür verantwortlich, dass ein Objekt unmittelbar nach seiner Erzeugung in einen gültigen Anfangszustand versetzt wird. Diese Aufgabe wird als ***Initialisierung*** bezeichnet; der Konstruktor initialisiert das Objekt. **Ein Konstruktor heißt stets so wie die Klasse, in der er definiert ist.**

Wir wissen bereits: Sowohl Konstruktoren als auch Methoden erhalten Werte über Parameter. Diese werden im Kopf eines Konstruktors oder einer Methode definiert. Der Konstruktor der Klasse Ticketautomat verfügt über einen **formalen Parameter** `ticketpreis`, mithilfe dessen das Attribut `preis` einen Wert zugeordnet bekommt (**Zuweisung**: `preis = ticketpreis`). Nach Abarbeitung des Konstruktors existiert die Variable `ticketpreis` nicht mehr.

Beispiel: `public Ticketautomat(int ticketpreis)`

- Die Variable *ticketpreis* ist ein ***formaler Parameter*** des Konstruktors der Klasse *Ticketautomat*. Formale Parameter sind Variablen, die nur innerhalb ihres zugehörigen Konstruktors oder ihrer zugehörigen Methode benutzt werden können.
- Ein ***aktueller Parameter*** ist der Wert, der dem formalen Parameter von außen zugewiesen wird. Z. B. die Zahl 200, die man beim Methodenaufruf eingeben kann.
- Die Lebensdauer eines formalen Parameters ist auf die Ausführungszeit der Methode/des Konstruktors beschränkt. Soll der Wert des aktuellen Parameters nicht verloren gehen, muss er in einem Attribut zwischengespeichert werden.

Aufgabe: Zu welcher Klasse gehört der folgende Konstruktor?

```
public Buch(String titel, double preis)
```

Wie viele Parameter hat er und welche Typen haben diese?

Hinweis: `double` steht für den Datentyp Dezimalzahl, d. h. eine Variable vom Datentyp double kann nicht nur ganze Zahlen, sondern auch Dezimalzahlen aufnehmen.

Aufgabe: Welche der folgenden Zuweisungen sind korrekt, welche nicht?

```
preis = 17;
```
(wenn *preis* eine Variable vom Typ *int* ist)

```
zahl = 13.1;
```
(wenn *zahl* eine Variable vom Typ *String* ist)

```
„Guten Tag“ = begruessung;
```
(wenn *begruessung* eine Variable vom Typ *String* ist)

```
int summe = 35;
```

Aufgabe: Nimm an, in einer Klasse *Haustier* gibt es ein Attribut *name* vom Typ *String*. Schreibe einen Konstruktor, dem ein aktueller Parameter für den Namen des Haustieres mitgegeben werden kann. Das Attribut *name* soll mit dem Wert des aktuellen Parameters initialisiert werden.

Methoden

Alle Methoden bestehen aus zwei Teilen:

Kopf der Methode: Der Kopf einer Methode besteht aus dem Kommentar zur Methode sowie aus der Signatur. Der Kommentar beschreibt, was die Methode tut. Du solltest Dir angewöhnen, jeder Methode einen sinnvollen Kommentar voranzustellen.

Wie bei den Attributen gibt es auch ein Schlüsselwort, hier jedoch `public`. Dieses sagt aus, dass die Methode von außen sichtbar ist und von anderen Objekten genutzt werden darf. Man kann Methoden auch als `private` deklarieren. Dies bedeutet, dass die Methode nur von der Klasse selbst genutzt werden darf – doch dazu später mehr.

Rumpf der Methode: Der Rumpf der Methode ist alles, was dem Kopf der Methode folgt – also die eigentlichen Programmzeilen.

Wie Du bereits aus dem Projekt *Laborkurse* weißt, gibt es Methoden, die einen Wert zurückgeben, und Methoden, die ein Attribut verändern.

Methoden, die einen Wert zurückgeben, nennt man **sondierende Methoden** oder auch **Funktionen**. Der Datentyp der Rückgabe steht am Anfang in der Signatur der Methode. Mit `return` wird der Rückgabewert zurückgegeben.

Methoden, die etwas verändern, aber keinen Wert zurückgeben, nennt man **verändernde Methoden** oder auch **Prozeduren**. Mithilfe eines Parameters kann man den Wert mitgeben, mit dem ein Attribut verändert werden soll.

Aufgabe: Gib aus der Klasse *Ticketautomat* jeweils ein Beispiel für beide Arten von Methoden an. Definiere anschließend Methoden für die folgenden Aufgaben:

1. eine sondierende Methode, welche die Gesamtsumme der eingeworfenen Gelder zurückliefert.
2. eine verändernde Methode, welche dem Ticketautomaten das Geld entnimmt.
3. eine verändernde Methode, welche einen Preis für ein Ticket festlegt.
4. einen parameterlosen Konstruktor. Dieser legt den Ticketpreis auf 200 Cent fest.
5. eine Methode, welche den aktuellen Kassenstand (Attribut `gesamtsumme`) auf dem Bildschirm ausgibt

(z. B. so: `Kassenbestand: 560 Cent`).
Schaue Dir dazu die Methode *ticketDrucken()* genauer an. Dort erkennst du, wie man eine Bildschirmausgabe mit dem Befehl `System.out.println("...")` programmiert.

Bewertung des Ticketautomaten – Fehlerbeschreibung

Wir haben bereits gesehen:

- Der Ticketautomat druckt Tickets ohne Beachtung des durch den Kunden eingeworfenen Betrags.
- Der Ticketautomat gibt kein Wechselgeld zurück.
- Der Ticketautomat lässt zu, dass unsinnige Beträge (z. B. negative Geldbeträge) eingeworfen werden können.
- Der Ticketautomat lässt negative Ticketpreise zu.

Zuerst kümmern wir uns um das Problem, dass negative Geldbeträge eingeworfen werden können. Innerhalb der Methode *geldEinwerfen(int betrag)* müsste man umgangssprachlich wie folgt vorgehen:

> Wenn *betrag* > 0, dann erhöhe *bisherGezahlt* um den *betrag*, andernfalls gib eine Fehlermeldung auf dem Bildschirm aus.

In Java könnte dies wie folgt formuliert werden:

```
public void geldEinwerfen(int betrag)
{
  if (betrag > 0) {
    bisherGezahlt += betrag;  // das gleiche wie
                              // bisherGezahlt=bisherGezahlt+betrag
  } else {
     System.out.println("Bitte nur positive Beträge verwenden!");
  }
}
```

Mit der Anweisung `if… else…` bietet Dir Java die Möglichkeit, in Abhängigkeit davon, ob eine Bedingung zutrifft, eine von zwei unterschiedlichen Anweisungen ausführen zu lassen. Die allgemeine Syntax der `if`-Anweisung lautet dabei wie folgt:

```
if (<Bedingung>)
{     <Anweisung 1>
            ...
      <Anweisung n>
} else {
      <Anweisung 1>
            ...
      <Anweisung n>
}
```

Dabei kann man auch auf den `else`-Teil verzichten, wenn dieser nicht nötig ist. Wird nur eine Anweisung benötigt, so kann man sogar die geschweiften Klammern weglassen.

Für die Bedingung kannst Du verschiedene Vergleichsoperatoren verwenden. Den größer-Operator kennst Du ja jetzt. Es gibt noch größer-gleich (>=), gleich (==), kleiner (<), kleiner-gleich (<=) und ungleich (`!=`). Außerdem kannst Du mehrere Bedin-

gungen mit UND (&&) bzw. ODER (||) verknüpfen. Probiere in den folgenden Aufgaben einfach einmal verschiedene Vergleichsoperatoren und Verknüpfungen aus.

Aufgabe: Verbessere den Ticketautomaten, so dass die oben beschriebenen Problemfälle abgefangen werden. Z. B. soll nur ein Ticket gedruckt werden, wenn genügend Geld eingeworfen wurde, andernfalls soll die Meldung "Es fehlt noch Geld!!!" ausgegeben werden[3].

Aufgabe: Implementiere eine sondierende Methode `int wechselgeldAusgabe()` für die folgende Aufgabe:
Wurde genügend Geld eingezahlt, so soll das Wechselgeld berechnet und zurückgegeben werden; außerdem soll der Wert für *bisherGezahlt* auf 0 gesetzt werden. Wurde noch nicht genügend Geld eingeworfen, so soll der Wert 0 zurückgegeben werden. Füge einen sinnvollen Methodenaufruf in der Methode *ticketDrucken()* ein.

Die letzte Aufgabe ist nur zu lösen, wenn Du eine neue Variable zur Berechnung des Wechselgeldes einführst. Eine mögliche Lösung könnte wie folgt aussehen:

```
public int wechselgeldAusgabe()
{
  if (bisherGezahlt >= preis) {
    int wechselgeld;
    wechselgeld = bisherGezahlt - preis;
    bisherGezahlt = 0;
    return wechselgeld;
  } else {
    return 0;
  }
}
```

Die Variable `wechselgeld` ist kein Attribut, da Attribute außerhalb der Methoden deklariert werden, aber auch kein Parameter, da Parameter im Kopf der Methode deklariert werden (innerhalb der Klammern). Da die Variable nur kurzzeitig (lokal) verwendet wird, nennt man eine solche Variable eine **lokale Variable**. Die Gültigkeit einer lokalen Variablen bezieht sich nur auf den Teil innerhalb der geschweiften Klammern, in der sie definiert wurden.

Aufgabe: Stelle tabellarisch Gemeinsamkeiten und Unterschiede der drei Variablenarten gegenüber. Lege dazu in Deinem Heft eine Tabelle wie folgt an und fülle alle Felder aus.

	Attribute	**Formale Parameter**	**Lokale Variablen**
Deklaration			
Ort der Deklaration			
Lebensdauer			
Gültigkeit / Sichtbarkeit			

[3] Wenn du schon Profi bist, so kannst Du ja als Fehlermeldung ausgeben lassen: "Es fehlen noch xyz Cent!", wobei xyz für die fehlenden Cent steht.

Aufgaben zur Selbstüberprüfung

Projekt Bücherei

Aufgabe Öffne das Projekt *Buecher*. Führe dann die folgenden Aufgaben durch:

a) Füge sondierende Methoden hinzu, mit denen man den Autor (`gibAutor`) und den Titel (`gibTitel`) auslesen kann. Man nennt diese Methoden, die den Wert eines Attributs ausgeben, auch **Getter**.

b) Implementiere zwei Methoden `schreibTitel` und `schreibAutor`, mit denen der Titel bzw. der Autor des Buches auf der Konsole ausgegeben werden kann.

c) Erweitere die Klasse `Buch` um ein Attribut `seiten`. Die Seitenzahl sollte dem Konstruktor mitgeteilt werden können.

d) Implementiere eine Methode `detailAusgabe`, welche alle Buchinformationen strukturiert auf der Konsole ausgibt.

e) Füge ein Attribut `signatur` vom Typ `String` hinzu. Standardmäßig sollte der Wert mit `""` initialisiert sein (Konstruktor ändern). Füge anschließend eine verändernde Methode `setzeSignatur` (so genannter **Setter**, der den Wert eines Attributs setzt) und einen Getter `gibSignatur` ein.

f) Erweitere die Methode `setzeSignatur` um eine Überprüfung, ob die eingegebene Signatur richtig ist. Eine korrekte Signatur besteht aus genau 10 Zeichen. **Hinweis:** Die Länge eines Strings kannst Du mit der Methode `length` der Klasse `String` herausfinden. Hier wäre das der Befehl `signatur.length()`.

g) Passe Deine Methode für die `detailAusgabe` an. Sollte noch keine Signatur für das Buch vergeben worden sein, so soll auf dem Bildschirm "unbekannt" erscheinen.

h) In Objekten der Klasse `Buch` soll gespeichert werden, wie oft das Buch entliehen wurde. Füge dafür einen Zähler `ausgeliehen` (Standardwert 0) der Klasse hinzu. Implementiere dann einen Getter (`gibAusgeliehen`), sowie eine Methode `ausleihen`, welche den Zähler um eins erhöht. Erweitere außerdem die Methode `detailAusgabe`.

i) Jedes Buch soll eine eigene Ausleihzeit besitzen. Diese ist durch einen Zahlencode verschlüsselt. 0 = keine Ausleihe möglich, 1 = Ausleihe übers Wochenende, 2 = Ausleihe für 7 Tage, 3 = Ausleihe beliebig lange möglich. Erweitere Deine Klasse Buch um

- **1)** ein Attribut `ausleihZeit`
- **2)** einen Getter und einen Setter
- **3)** eine Standardinitialisierung mit dem Code 3
- **4)** eine angepasste `detailAusgabe`, welche die Ausleihzeit in Worten angibt.

Projekt Bodymaßindex-Waage

Aufgabe: Erzeuge ein neues Projekt *Bodymaßwaage* mit einer neuen Klasse `Waage`. Objekte dieser Klasse sollen es möglich machen, Personen zu wiegen, zu messen und deren Bodymaßindex zu berechnen.

a) Ändere die Projektbeschreibung und Kommentare entsprechend ab. Lösche alle überflüssigen Attribute und Beispiel-Methoden.

b) Objekte der Klasse `Waage` haben die Attribute `gewicht` und `groesse`. Erweitere die Klasse um Getter und Setter

c) Implementiere zwei verschiedene Konstruktoren. Mit einem Konstruktor soll die Angabe des Gewichts und der Größe direkt möglich sein, der andere soll beide Werte auf 0 setzen.

d) Entwickle eine Methode zur Berechnung des Bodymaßindex `public double bodyMassIndex()`. Der bmi berechnet sich mit der Formel $bmi = \frac{gewicht}{größe^2}$. Achte darauf, dass bei nicht erfolgter Größen-Eingabe eine Berechnung nicht möglich ist (Division durch 0).

e) Implementiere eine Methode `detailAusgabe`, welche alle Informationen übersichtlich auf der Konsole ausgibt.

f) Mit Hilfe des bmi können Informationen über Untergewicht, Normalgewicht oder Übergewicht ausgegeben werden. Informiere dich im Internet über die Grenzen (z. B. ist ein bmi < 20 mit Untergewicht gleichzusetzen) und erweitern Sie die Methode `detailAusgabe` um diese Information. Gehe zuerst davon aus, dass nur männliche Personen die Waage nutzen.

g) Erweitere Deine Klasse um ein Attribut `geschlecht` vom Typ `String`. Dieses soll beim Konstruktor mitgegeben werden können, aber auch anschließend durch eine Setter-Methode verändert werden können. Natürlich sollen nur die Geschlechter "m" und "w" zugelassen werden. Achte bei der Implementierung der Setter-Methode darauf.
Ändere anschließend die `detailAusgabe` dahingehend, dass die Informationen über das Gewicht abhängig vom Geschlecht ausgegeben werden.
Hinweis: Zur Überprüfung, ob in dem Attribut `geschlecht` der Wert "m" steht, benötigst Du einen neuen Befehl: `if (geschlecht.equals("m")) { ... }`
Die Methode `equals` vergleicht den String des aktuellen Parameters (hier "m") mit dem String des Attributs `geschlecht`.

Projekt Heizungssteuerung

Aufgabe: Erzeuge ein Projekt *Heizungssteuerung* mit einer Klasse `Heizung`. Lösche alle überflüssigen Attribute, Methoden und Kommentare.

a) Die Klasse `Heizung` besitzt ein Attribut `double temperatur`, welches Temperaturen mit einer Genauigkeit von zehntel Grad aufnimmt, die im Bereich zwischen 0° C und 45° C liegen.

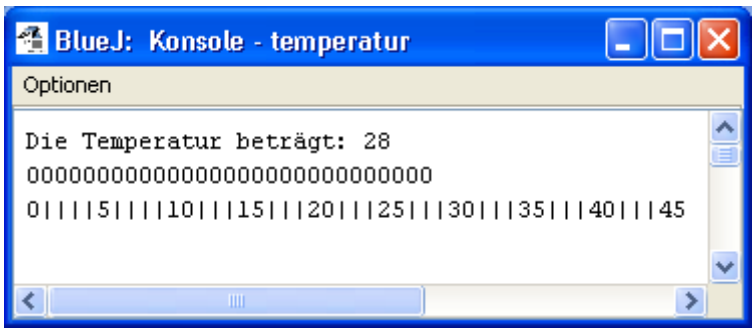
b) Definiere einen Konstruktor, welcher die Temperatur auf einen Anfangswert von 15,5° C setzt. (In Java wird statt des Kommas ein Punkt verwendet: `temperatur = 15.5`)

c) Implementiere eine Getter-Methode, welche die Temperatur zurückgibt.

d) Entwickle zwei verändernde Methoden `waermer` und `kaelter`, die die Temperatur um jeweils 0,5° C erhöhen bzw. senken. Erweitere die Klasse um zwei weitere Methoden `schnellWaermer` und `schnellKuehler`, welche die Temperatur um 5° C verändern.
Hinweis: Achte bei allen Methoden darauf, dass die Temperatur im Intervall [0; 45] bleibt.

e) Implementiere eine Methode, welche die aktuelle Temperatur auf der Konsole ausgibt. Die Temperatur soll dabei auf volle Grad gerundet werden. Verwende dafür den folgenden Befehl: `int gerundet = (int)temperatur;`
Hinweis: Mit `(int)...` wird eine Dezimalzahl in eine ganze Zahl umgewandelt, indem die Nachkommastellen einfach weggelassen werden. Dies entspricht nicht der mathematisch korrekten Rundung.

f) Entwirf eine analoge Temperaturanzeige, welche je nach Temperatur einen unterschiedlich langen Balken ausgibt.
Hinweis: Hier könnte Dir eine Zählschleife weiterhelfen. Informiere dich im Internet über die Syntax der `for`-Schleife, welche in Java als Zählschleife verwendet wird.



Die letzte Aufgabe hatte es schon ein bisschen in sich, oder? Ich hoffe, Du konntest dich im Internet über die allgemeine Syntax der `for`-Schleife informieren. In unserem Fall könnte die Methode *temperaturAusgabe* wie folgt programmiert werden:

```
public void temperaturAusgabe()
{
  int gerundet = (int)temperatur;
  System.out.println("Die Temperatur beträgt: "+gerundet);
  for (int i = 0; i <= temperatur; i++) {
    System.out.print("O");
  }
  System.out.println();
  System.out.println("0|||5|||10|||15|||20|||25|||30|||35");
}
```

An dem fettgedruckten Befehl erkennst Du die allgemeine Struktur einer **Zählschleife** in Java, also einer `for`-Schleife:

```
for ( <Initialisierungs-Anweisung> ; <Bedingung> ; <Aktualisierungs-Anweisung> )
{   <Anweisung >
        ...
    <Anweisung n>
}
```

Hierbei können die geschweiften Klammern weggelassen werden, wenn nur eine Anweisung durchgeführt werden soll.

Mit der *Initialisierungs-Anweisung* wird im Normalfall eine Schleifenvariable auf einen Anfangswert gesetzt (hier soll die Variable i mit 0 starten). Die *Bedingung* gibt an, in welchem Fall die Schleife weiter durchgeführt werden soll (hier: solange i kleiner oder gleich der Temperatur ist). Die *Aktualisierungs-Anweisung* wird in der Regel dazu genutzt, die Schleifenvariable zu erhöhen oder zu erniedrigen (hier: die Variable i wird um 1 erhöht). In *Anweisung 1* bis *Anweisung n* steht dann, was während der Durchführung der Schleife geschehen soll (bei uns ist es nur die Ausgabe-Anweisung für ein "O").

Mit dieser `for`-Schleife kann man in Java fast alle Wiederholungs-Anweisungen durchführen, dennoch werden wir bald auch andere wichtige Wiederholungs-Anweisungen kennen lernen.

Neben einer neuen Anweisung hast Du in der letzten Aufgabe auch einen neuen Datentyp kennen gelernt. Es ist an der Zeit, Dir die wichtigsten primitiven Datentypen genauer vorzustellen:

Datentyp	Beschreibung	mögliche Werte	Länge in Bits
`boolean`	**Wahrheitswert**	**true, false**	**1**
`char`	Zeichen, Buchstabe	'c', '\n', '\u0035'	16
`byte`	ganze Zahl	-128=-27 bis 127=27-1	8
`short`	ganze Zahl	-32768 =-215 bis 32767=215-1	16
`int`	**ganze Zahl**	**-2^{31} bis $2^{31}-1$**	**32**

`long`	ganze Zahl	-2^{63} bis $2^{63}-1$	64
`float`	Gleitkommazahl	-3.40282347E+38 bis 3.40282347E+38	32
`double`	**Gleitkommazahl**	**-1.7976931348623157E+308 bis 1.7976931348623157E+308**	**64**

Größtenteils kommt man mit den fettgedruckten Datentypen aus. Vermutlich vermisst Du den Datentyp String in dieser Tabelle, oder? Der Datentyp String ist eine Klasse, ähnlich wie z. B. die Klasse *Rechteck* aus unserem ersten Projekt. Dies erkennt man im Übrigen auch daran, dass der Datentyp mit einem großen Buchstaben beginnt. Du hast sogar bereits Methoden der Klasse String verwendet – erinnere dich an *equals* oder an *length*. Im Moment wollen wir uns aber mehr um die primitiven Datentypen kümmern. Doch zuvor nochmals eine kleine

Zusammenfassung

In diesem Kapitel hast Du folgende wichtige Dinge kennen gelernt:

- Du hast eigene *Klassen* modelliert und implementiert und *Objekte* dieser Klassen manuell oder zur Laufzeit erzeugt.
- Du kennst den Unterschied zwischen *Attributen*, *lokalen Variablen* und *formalen Parametern*. Du kennst deren Verwendung, deren *Deklaration* und deren *Gültigkeitsbereiche*.
- Du kennst die Unterschiede zwischen *Konstruktoren* und *Methoden*. Du hast die Begriffe der *sondierenden* und *verändernden Methode* kennen gelernt.
- Du kennst bereits verschiedene Arten von *Anweisungen*: *Zuweisungen*, *bedingte Anweisungen* (`if ... else`) und *Wiederholungsanweisungen* (`for(...)`). Sowohl die bedingte Anweisung als auch die Schleifenanweisung benötigen *Bedingungen*, sogenannte *boolesche Ausdrücke*, deren Wert *true* oder *false* ergeben kann.
- Du kannst Deinen Quelltext mit *Kommentaren* und sinnvollen *Bildschirmausgaben* versehen.
- Du hast den primitiven Datentyp *double* verwendet und in Werte vom Typ *int* verwandelt.