I Einführung in JAVA mit BlueJ

1 Erste Schritte

1.1 Was ist JAVA bzw. BlueJ?

Wir müssen uns JAVA als eine Hilfe vorstellen, um bestimmte Probleme auf dem Computer zu lösen. Wollen wir z.B. auf dem Taschenrechner zwei Zahlen miteinander multiplizieren, so geht das ganz einfach. Mit JAVA ist das Einiges komplizierter, dafür versagt der Taschenrechner jedoch bei Problemen, die mit JAVA noch lösbar sind. So wie bei jeder Sprache müssen wir lernen, wie wir mit dem Computer auf „JAVA“ sprechen können. Und BlueJ, eine sogenannte Entwicklungsumgebung, hilft uns beim Anwenden dieser Sprache.

Grundlegend bei einer Sprache ist natürlich, dass der andere – in diesem Fall der Computer – versteht, was man sagen will. Damit das funktioniert muss man sich an bestimmte Vorgaben und Regeln halten. Dazu ein kurzes Programmbeispiel:

```
//Die Klasse KREIS
class KREIS{

    //Attribute
    int radius;
    int xPosition;
    int yPosition;

    //Konstruktor
    KREIS(){
        radius = 30;
        xPosition = 40;
        yPosition = 60;
    }

    //Weitere Methoden
    void nachrechts(){
        xPosition = 100;
        zeichnen();
    }
}
```

Im ersten Abschnitt sagen wir dem Computer grundsätzlich erst einmal, worum es geht. Ich erkläre also dem Computer mein Anliegen. Im obigen Beispiel geht es anscheinend um Kreise, genauer: um Objekte der Klasse `KREIS`. Deshalb die Befehlszeile `class KREIS{`

Nehmen wir an, ich würde dem Computer gerne etwas über Rechtecke beibringen, wie müsste dann wohl die erste Zeile lauten? (alles was Ihr an Schreibweisen bis zum aktuellen Zeitpunkt im Skript nicht versteht, schreibt Ihr einfach entsprechend anderer Beispiele ab):

```
class RECHTECK {
```

Die geöffnete geschweifte Klammer bedeutet, dass jetzt mehr zu dieser Klasse kommt. Vielleicht könnt Ihr Euch noch erinnern: Klasse ist eine Art Überbegriff, ihr gehören beliebig viele Objekte an (das Auto bei Euch daheim in der Garage ist ein Objekt der Klasse PKW).
Objekte, die einer gemeinsamen Klasse angehören, haben die gleichen Eigenschaften, besser: `Attribute`. Im obigen Beispiel erfahren wir also mehr über die `Attribute`, die für ein Objekt der Klasse `KREIS` wichtig sind.

Entsprechend: Was ist für ein Auto der Klasse `PKW` relevant? Welche Attribute hat es also?

PS, Farbe, Marke, vMax, Hubraum, ...

Diese Attribute schreibt man direkt nach der Klasse auf. Bei Rechtecken wäre das also z.B.:

Länge, Breite, Fuellfarbe, Position, Rand, ...

Weiterhin muss ich dem Computer sagen, von welcher Art die Attribute sind. Ganze Zahlen nennt man z.B. `int`. (Kurzform von integer, Englisch für „ganze Zahl“) D.h. beim Auto würde man sagen: `int PS;` da der PS-Wert meistens eine ganze Zahl ist.

Farbe ist ein Wort, auf gut JAVA ein `String` und wird bei der Eingabe in Anführungszeichen geschrieben weil der Computer natürlich nichts mit Buchstaben anfangen kann. Bei unserem Auto wäre das `String farbe;` da hier natürlich ein Wort, also eine Verkettung von Zeichen vorliegt. So zähle ich dem Computer also alle relevanten Attribute auf.

Dann gibt es noch den Attributtyp `boolean`, an den ihr euch hoffentlich erinnern könnt. Hier geht es um etwas das `wahr` oder `falsch` sein kann. So kann das Auto beispielsweise Allrad besitzen oder nicht, weshalb man hier schreiben würde `boolean allrad`.

Weiterhin gibt es noch `double`, was für reelle Zahlen verwendet wird. Man könnte also auch `double PS` definieren, da eben manchmal der PS-Wert auch keine ganze Zahl ist.

Später kommt noch der Datentyp `char` dazu; dieser ähnelt dem Datentyp `String`, besteht jedoch nur aus einem einzelnen Zeichen.

1.2 Die erste eigene Klasse

Zurück zu unserem Programm. Schreibe entsprechend dem Kreisbeispiel eine Klassendefinition zum Rechteck, also die „Kopfzeile“ und die nötigen Attribute (denk daran, dass ein Rechteck eine Farbe hat, aber auch unsichtbar sein könnte; welche Attributtypen verwendest Du?)

```
class RECHTECK {
    int laenge;
    int breite;
    String fuellfarbe;
    boolean istSichtbar;
```

1.3 Der Konstruktor

Als nächstes sollte man dem Computer sagen, welche Werte die einzelnen Attribute zu Beginn haben, man „konstruiert“ quasi einen Anfangszustand, daher der Name `Konstruktor`. Es handelt sich hier schon um eine spezielle Methode, die den gleichen Namen wie die Klasse hat, hier also auch `KREIS`. Man sagt also um welche Klasse es geht und gibt den zugehörigen Attributen durch eine einfache Zuweisung beliebige Werte (beim Auto z.B. Farbe = Silbermetallic). Wie lautet also ein möglicher Konstruktor für die Klasse Rechteck?

```
RECHTECK() {
    laenge = 70;
    breite = 40;
    fuellfarbe = "red";
    istSichtbar = true;  // oder false
    }
```

Dieser Konstruktor ist nicht unbedingt notwendig, macht aber schon Sinn. Beim Auto könnte man sich ein unlackiertes, unfrisiertes Grundmodell vorstellen, das man später noch lackiert oder aufpimpt.

1.4 Die erste eigene Methode

Bisher ist ja noch nicht viel passiert. Will der Computer nun mit den Objekten etwas machen, dann nennt man dies eine Methode. Einen PKW kann man z.B. lackieren, fahren, putzen usw. Natürlich muss man dem Computer auch diese Methoden erklären.

Dazu gibt man dem ganzen Vorgang einen (möglichst sinnvollen) Namen und anschließend sagt man dem Computer was er tun soll, wenn irgendjemand diese Methode anwenden will. Computer, mach das Rechteck sichtbar. Was muss der Computer tun?

```
void sichtbarMachen(){
    istSichtbar = true;
    zeichnen();
    }
```

Mit dem Befehl `zeichnen` (das ist selbst wieder eine Methode, aber darauf gehen wir später ein) wird dabei das nun veränderte Objekt neu gezeichnet, die Änderung wird also sichtbar. Vorerst schauen alle Methoden so aus. Das `void` bedeutet übrigens, dass der Computer uns keine direkte Rückgabe liefert, also kein Ergebnis. Ich möchte ja keine Antwort haben, anders als wenn ich fragen würde, was ist denn die Summe der Zahlen 2 und 3; hier will ich als Mathelehrer ja eine Antwort bekommen.

1.5 Wertänderungen

Wenn wir in BlueJ Werte ändern wollen, dann funktioniert das auch auf einem ganz speziellen Weg: **links** steht das, was einen neuen Wert bekommen soll, rechts der Rechenausdruck oder die Zahl, woraus sich der neue Wert ergibt. Nehmen wir an Du sollst ab nächste Woche 10 Euro Taschengeld bekommen. Dann lautet die Anweisung wie?

```
Taschengeld = 10;
```

Nehmen wir an, Dein altes Taschengeld soll um 10 Euro erhöht werden, egal wie hoch es zuvor war. Wie rechnest Du das aus? Da kann es also auch nur eine mögliche Anweisung geben. Und wie lautet die Anweisung, wenn sich das Taschengeld verdoppeln soll? Wie lautet die Anweisung, wenn ich die `flaeche` eines Rechtecks aus seinen Werten für `laenge` und `breite` berechnen will?

```
Taschengeld = Taschengeld + 10;
Taschengeld = Taschengeld * 2;
flaeche = laenge * breite;
```

Wenn Du das verstanden hast, dann erstelle nun eine Methode, welche Länge und Breite eines Rechtecks ganz allgemein verdoppelt.

```
void verdoppeln(){
    laenge = laenge * 2;
    breite = breite * 2;
    zeichnen();
    }
```

1.6 Parameter

Was, wenn ich nun das Rechteck nicht verdoppeln will, sondern ganz allgemein um einen beliebigen Faktor strecken will? Der Computerbenutzer soll also entscheiden, wie sehr das Objekt gestreckt oder gestaucht wird. Erstelle nun eine Methode `strecken`, welche die Länge und Breite eines Rechtecks ganz allgemein ver-`faktor`-facht (mit `faktor` ϵ IN).

D.h. ich brauche also eine Möglichkeit, dem Computer etwas einzugeben. Anders ausgedrückt: der Computer muss wissen, dass wenn die Methode `strecken` angewendet wird, man auch noch einen Streckfaktor eingeben muss. Natürlich muss dem Computer dabei bekannt sein, was für eine Art von Wert eingegeben wird. Und das steht, wie schon von z.B. EOS bekannt, in den Klammern am Ende der Methode. Hier habe ich also die Methode:

```
void strecken(int faktor){
    laenge = laenge * faktor;
    breite = breite * faktor;
    zeichnen();
    }
```

Was fällt auf? Richtig, bei dem Wert, den der Benutzer eingeben soll, steht noch ein `int` dabei, damit klar wird, welche Art von Eingabewert hier erwartet wird. Hier z.B. muss dann eine ganze Zahl eingegeben werden, etwas anderes akzeptiert das Programm nicht. Ach ja, etwas, was der Benutzer eingeben kann, nennt man übrigens einen Parameter.

Wie lautet denn nun der Quelltext (das sind die Programmzeilen) einer Methode `groesser`, welche Länge und Breite des Rechtecks um einen bestimmten Wert (`AenderWert`) erhöht?

```
void groesser(int AenderWert){
    laenge = laenge + AenderWert;
    breite = breite + AenderWert;
    zeichnen();
    }
```

Und wie könnte ich die Farbe des Rechtecks ändern (`farbeAendern`)? Achtung, von welcher Art ist hier der Eingabewert? In welchem Attribut steht die bisherige Farbe?

```
void farbeAendern (String andereFarbe){
    fuellfarbe = andereFarbe;
    zeichnen();
    }
```

2 Erste Praktische Anwendung mit BlueJ

2.1 Erste eigene Methoden

So, nachdem wir nun schon einige Beispiele theoretisch abgehandelt haben, nun ein paar weitere Beispiele mit BlueJ selbst. Dazu speichere bitte das verlinkte Projekt `BlueJ_Tutorial` auf Deinem Computer und öffne es mit BlueJ.

Schau Dir nun den Quelltext zur Klasse `RECHTECK` an (rechter Mausklick auf `RECHTECK`, dann `Bearbeiten`). Lass Dich nicht verwirren, wenn hier ab und zu ein **`public`** oder ein **`private`** dabei steht. Das können wir weglassen und wird für uns frühestens in der 11.Klasse relevant.

Du sollst nun weitere Methoden ergänzen. Diese kommen natürlich an die Stelle im Quelltext, an der **`\\Weitere Methoden`** steht. Ergänze nun entsprechend der im Quelltext schon vorhandenen Methode `nachRechts()` die Methode `nachLinks()`. #Lsg20

Anschließend den Quelltext compilieren (also quasi übersetzen) und es müsste funktionieren, d.h. Du müsstet diese Methode nun verwenden können.

2.2 Methoden können sich gegenseitig aufrufen

Jetzt erstelle bitte noch zwei weitere Methoden, und zwar:

`nachUnten()`
`nachOben()` #Lsg07

Wenn Du nun den Quelltext kompiliert, dann bekommst Du eine Fehlermeldung (Erklärung sämtlicher Fehlermeldungen im Anhang). Warum?

Klar, in Eurer Methode hier steht ja nun eine Anweisung `bewegeVertikal`, d.h. Du rufst eine andere Methode auf. Diese Methode kennt der Computer bisher aber noch nicht. Erstelle deshalb noch die benötigte Methode `bewegeVertikal`. Nun müsste alles klappen. #Lsg01

Außerdem fehlt, wenn wir eine Methode `breiteSetzen` haben noch eine weitere Methode. Erstelle diese bitte. #Lsg11

Anschließend hätte ich gerne noch eine Methode `LaengeUndBreiteSetzen`, die beide Vorgänge auf einmal absolviert. #Lsg05

Zurück zur Kapitelüberschrift. Erstelle bitte eine Methode `groesser` wie schon im Kapitel 1. Verändere die Methode aber nun bitte so, dass sie funktioniert, indem sie selbst die Methoden `breiteSetzen` und `laengeSetzen` aufruft. #Lsg17

Zusatz: Erstelle folgende Methoden (Kontrolle der Funktionalität direkt am Computer)

1. Eine Methode Quadratmachen, die aus dem Rechteck ein Quadrat macht, indem es eine der beiden Seitenlängen durch die andere überschreibt. #Lsg22
2. Eine Methode Kippen, die ein Rechteck um 90° kippt #Lsg06
3. Eine Methode StreckenXY, die ein Rechteck um einen bestimmten Faktor in x-Richtung und um einen anderen Faktor in y-Richtung streckt. #Lsg15

3 Kommunikation mit Methoden

3.1 Was kommt heraus?

Wenn ich den Computer als Taschenrechner missbrauchen möchte, dann brauche ich natürlich auch eine Rückgabe, sozusagen das Display auf dem das Ergebnis steht. Das üben wir mal mit dem Satz des Pythagoras. Aus zwei gegebenen Seiten `a` und `b` soll die dritte berechnet werden.

Bevor Du Dich gleich an die Aufgabe machst – zum einen brauchst Du noch einen speziellen Befehl, mit dem Du die Wurzel aus einer Zahl ziehen kannst, nämlich `Math.sqrt(zahl)`, zum anderen musst du noch etwas importieren.

Was soll das denn schon wieder heißen? Also, irgendwelche Leute haben schon mal ein paar mathematische Methoden geschrieben und die unter dem Namen `java.lang.Math` zusammengefasst. Wenn du diese Methoden verwenden willst, musst Du das dem Computer anfangs mitteilen. Und das geht so:

```
Import java.lang.Math
```

Das kommt also ganz zu Beginn eines Programms. Erstelle bitte ein komplettes Programm in der Klasse `PYTHAGORAS` mit den entsprechenden Attributen (also den benötigten Variablen), einem `Konstruktor` mit Anfangswerten 0 für alle Parameter und einer Methode `cBerechnen`, in der das eigentliche Problem gelöst wird. Teste mit dem Inspektor! `#Lsg16`

Das Überprüfen der Korrektheit mit dem Inspektor ist natürlich immer etwas langwierig. Darum lassen wir uns vom Computer das berechnete Ergebnis gleich ausgeben. Dazu schreiben wir erst mal die Methode ein wenig um. Bisher sah die Kopfzeile einer Methode so aus:

```
void cBerechnen(double aWert, double bWert){
```

Eine Rückgabe wurde hier bisher vermieden (englisch `avoid`). Jetzt möchte ich etwas zurückhaben (engl. `return`). Und zwar den Wert, den Du berechnet hast, z.B. also Deinen `Ergebniswert`. Dazu brauchst Du in der Methode also einerseits die Angabe, von welcher Art der Rückgabewert ist und andererseits natürlich, welche Variable zurückgegeben werden soll. Diese Zeile heißt nun also:

```
double cBerechnen(double aWert, double bWert){
```

Zum zweiten brauchst Du noch eine Zeile, welche diese Rückgabe erledigt (mit `return`). Also:

```
return Ergebniswert
```

Verbessere nun Dein Programm noch entsprechend und teste es mit Hilfe von Dir bekannten Zahlentripeln (3-4-5 oder 12-5-?). `#Lsg03`

3.2 Pimp your program

Schönheitskorrekturen kann man vornehmen, indem man z.B. innerhalb der Methode schreibt:

```
System.out.println(cSeite);
```

Das ist der Befehl mit dem man etwas auf dem Bildschirm ausgibt. Besser wäre natürlich:

```
System.out.println("Die Hypotenuse ist " + cSeite);
```

So kann man also einen Text mit einer Variablen verknüpfen. Nicht das `+` vergessen! Verbessere Deinen Quelltext, sodass der Computer Dir die beiden eingegebenen Katheten und die berechnete Hypotenuse auf dem Bildschirm ausgibt.
Diese Ausgabe im Konsolenfenster, das ist das Fenster, welches sich beim Aufruf einer solchen `System.out.println` – Anweisung öffnet, hat natürlich nichts mit der `return` – Anweisung zu tun. Die muss trotzdem sein, wenn Du die Methode entsprechend geschrieben hast. Ansonsten kommt die Fehlermeldung: `missing return statement.` #Lsg19

3.3 Ein Gutscheinautomat

Als nächstes wollen wir einen Gutscheinautomaten betrachten. Der Anwender soll sagen können, welchen Gutscheinwert er haben möchte, anschließend wirft er Geld ein und am Ende kann er sich den Gutschein ausdrucken.

Schau Dir dazu bitte den Quelltext in der Klasse `GUTSCHEINE` an, probiere den Automaten aus und beantworte folgende Fragen:

1. Welche Wertanweisung muss bei der Methode `geldEinwerfen` ergänzt werden?
2. Wofür gibt es die Variable `wozu`?
3. Wozu dient die Methode `ausgabe`?
4. Was leistet die Methode `bisher`?
5. Welche Schwachstellen hat der Automat?

bisherGezahlt = bisherGezahlt + betrag

Sie berechnet die Gesamteinnahmen des Automaten

Hier wird der gewählte Gutscheinwert ausgegeben

Hier wird der bisher gezahlte Geldbetrag ausgegeben

Werte < 0, Druck bevor Wert erreicht, keine Rückgabe

Beantworte die Fragen 1-4 bitte auch im Quelltext Deines Programms. Daran erkennst Du auch, wie es möglich ist, in einem Programm nützliche Bemerkungen für den Benutzer einzugeben. Dabei gibt es zwei Varianten. BlueJ ergänzt hier selbständig die benötigten Zeichen, wenn Du die Zeichenfolge `/**` eingibst und anschließend die Eingabetaste betätigst. Oder Du verwendest die Zeichenfolge `//` für einen einzeiligen Kommentar.

4 Fallunterscheidungen

4.1 Rückblick

Wir haben schon des Öfteren mit Fallunterscheidungen gearbeitet. Sei es in der 7.Klasse bei Robot Karol - nebenbei: was leistet dieses Programm denn?

```
wiederhole solange NichtIstWand
   wenn IstZiegel
      dann aufheben
      sonst hinlegen
   *wenn
   schritt
*wiederhole
```

Es invertiert eine Reihe bis zur Wand. Wo Ziegel waren sind nun keine und umgekehrt.

Oder in der 9. Klasse bei bedingten Anweisungen in den Zellen:

```
Wenn(B1 < 10; 0,10; 0,03)
```

Wenn der Wert in B1 kleiner 10 ist, dann ist das Ergebnis 0,10; sonst 0,03.

Wo könnten wir denn nun bei unserem Gutscheinautomaten eine Fallunterscheidung brauchen? Klar, einerseits bei der Eingabe eines Betrages innerhalb der `geldEinwerfen` – Methode, andererseits beim `gutscheinDrucken`, denn dies sollte natürlich nur möglich sein, wenn der komplette Betrag bezahlt wurde.

4.2 Fallunterscheidung bei BlueJ

Und wie schaut bei BlueJ so eine Fallunterscheidung aus? Ganz einfach. Wenn eine bestimmte Bedingung gilt (mit `if`), dann mache dies, ansonsten (`else`) mach' etwas Anderes.

```
if (bedingung){
    //Anweisung
}
else{
    //Alternativanweisungen
}
```

Ein ganz einfaches Beispiel vorab. Verbessere Dein Pythagoras-Programm, indem du es um eine Fallunterscheidung ergänzt, die Werte unter Null vermeidet. Das Ganze natürlich für die Seiten `a` und `b`, sodass insgesamt nur noch „richtige“ Werte zu einer Berechnung führen. `#Lsg21`

Zurück zu unserem Automaten. Öffne wieder die Klasse `GUTSCHEINE` und versuche die Methode `GeldEinwerfen` so abzuändern, dass sie nur Beträge über 0 akzeptiert. (Ansonsten irgendeine Ausgabe auf dem Bildschirm für den Benutzer). `#Lsg13`

So, das wäre erledigt. Aber ein weiteres Problem war, dass man den Gutschein ausdrucken konnte, auch wenn man ihn nicht voll bezahlt hatte. Deshalb bräuchten wir auch beim Aufruf der Methode `gutscheinDrucken` eine Fallunterscheidung, die dies verhindert.
Um die Kunden nicht zu verärgern, sollte der Automat auch noch Restgeld zurückgeben. Ändere die entsprechenden Befehle in der Methode `gutscheinDrucken` so ab, dass dem Nutzer auch noch ein Restbetrag angezeigt (und imaginär ausgezahlt) wird. `#Lsg2`

Übrigens, wenn man bei einem Vergleich z.B. innerhalb einer `if-Abfrage` untersuchen will, ob ein Wert genau gleich Null ist, so lautet die entsprechende Befehlszeile:

```
if (testwert == 0)...
```

Dabei wird das doppelte „=“ zur Unterscheidung von einer einfachen Wertanweisung: `a=0` verwendet. Weitere Vergleichsoperatoren lernen wir demnächst.

4.3 Die Mitternachtsformel

Versuche diesmal, den Quelltext zuerst auf einem Blatt zu notieren und erst anschließend auf den Computer zu übertragen.

- Die benötigten Variablen wollen wir dabei `aWert`, `bWert` und `cWert` nennen, die zu berechnenden Lösungen `lsg1` und `lsg2`. Die neu zu erstellende Klasse nennen wir `MITTERNACHT`.
- Der Benutzer gibt in einer Methode `werteEinlesen` die Werte für `a`, `b` und `c` ein.
- Mit Hilfe der Methode `anzahlLSG` führt der Computer eine Fallunterscheidung durch, um dem Benutzer zurückgeben zu können, wie viele Lösungen es gibt.
- Eine weitere Methode `loesungen` berechnet die möglichen Werte.
- Schließlich soll noch eine vierte Methode für die Bildschirmausgabe erstellt werden: `ausgabeLoesungen`. `#Lsg09`

Sicher hast Du schon bemerkt, dass eine solche Art des Programmierens nicht sinnvoll ist. Das Programm funktioniert nur, wenn die Methoden für ein Objekt der Klasse Mitternacht der Reihe nach aufgerufen werden. Das werden wir künftig vermeiden.

Zusätzlich hast Du vielleicht eine Kleinigkeit vergessen. Schau Dir die Mitternachtsformel an. Bei welchen Werten funktioniert sie denn (zusätzlich zu negativen Diskriminanten) auch nicht?
Richtig. Es sollte auch eine Abfrage stattfinden, ob denn auch der Wert für `a` ungleich 0 ist. Und um zwei Bedingungen gleichzeitig abfragen zu können verwendet man wie auch schon bei Excel ein OR. Dies wird bei BlueJ allerdings etwas anders geschrieben. Hier eine kurze Zusammenfassung von Bool’schen Vergleichen und Verknüpfungen in BlueJ:

Vergleiche:

<	kleiner als	if (a < b)
>	größer als	if (a > b)
<=	Kleiner oder gleich	if (a <= b)
>=	Größer oder gleich	if (a >= b)
==	Gleich	if (a == b)
!=	Ungleich	if (a != b)

Verknüpfungen:

&&	AND (und)	if (a < b && c < d)
||	OR (oder)	if (a > b || c < d)
!	NOT (nicht)	if (!(a < b))

4.4 Mehrfache Fallunterscheidung

Bei obigem Beispiel musste man zwischen drei Fällen unterscheiden. Da geht das schon noch mit `if...else`. Wie ist das aber, wenn man mehr als drei Fälle hat?

1. Beispiel: Währungsumrechner

Nehmen wir dazu einen Währungsrechner. Der Benutzer gibt einen Betrag ein und die Währung in welche er diesen umgerechnet haben will, der Computer erledigt den Rest. Wenn wir hier fünf verschiedene Währungen haben, z.B. DM, Schilling (ehemalige österreichische Währung), Peso (Argentinien), US-Dollar und Krone (Norwegen), dann wird das mit unserer bisherigen Anweisung schon schwierig.

Dazu gibt es die `case`–Anweisung. Um diese Anweisung kennen zu lernen, öffne bitte die Klasse `WAEHRUNG`. Probiere das Programm aus und überlege Dir zuerst einmal, wozu die `Default`-Anweisung dient!

Sie soll alle übrigen falschen Eingaben abfangen

Informiere Dich im Internet über die aktuellen Wechselkurse der oben angegebenen Währungen und notiere Dir die Umrechnungsfaktoren in Dein Heft.

Die entscheidenden Anweisungen innerhalb einer solchen Mehrfachauswahl sind:

```
switch (Auswahlwert){
    case 0: {
        //Anweisungen;
    } break;
    ...
    ...
    default: {
        //Anweisungen;
    } break;
}
```

Ergänze nun in Deinem Programm die restlichen Fälle.

Eine Fallunterscheidung in BlueJ kann aber noch Einiges mehr. Die Fälle müssen beispielsweise nicht durchnummeriert sein. Es ist auch möglich, auf die Eingabe eines Zeichens (`char`) zu warten und aufgrund dieses Zeichens die Fallunterscheidung anzusetzen. Hier lautet die Fallunterscheidung also:

```
    case 'D': {
```

Ändere die Unterscheidung nun ab, sodass der Benutzer den Anfangsbuchstaben der gewünschten Währung eingeben muss, um die Umrechnung zu erhalten. #Lsg04

2. Beispiel: Versandkostenrechner.

Eine Anwendung der `Case-Fallunterscheidung` ist auch bei einem Versandkostenrechner möglich. Erstelle in einer Klasse `VERSANDRECHNER` ein Programm, das nach Eingabe des Paketgewichts die entsprechenden Versandkosten ausgibt.

Achtung, hier hilft ein Trick: teile den Bestellwert durch 10 (Division mit Rest) und verwende das Ergebnis als Fall für Deine Fallunterscheidung. Bestellungen mit über 40 Euro musst du allerdings getrennt betrachten.

Jedenfalls benötigst Du eine Methode, um aufgrund des Bestellwerts den richtigen Fall zu ermitteln und aus dieser Methode heraus rufst Du dann eine zweite Methode auf, die Dir aufgrund einer Fallunterscheidung die korrekten Versandkosten zurückgibt. #Lsg14

Bestellwert		Versandkosten
< 10 Euro	→	3,97 Euro
< 20 Euro	→	2,97 Euro
< 30 Euro	→	1,97 Euro
< 40 Euro	→	0,97 Euro
ab 40 Euro	→	keine

5 Wiederholungen in BlueJ

5.1 Rückblick

Auch mit Wiederholungen haben wir uns schon des Öfteren beschäftigt. An welche Arten der Wiederholung kannst Du Dich denn erinnern?

Wiederholung mit fester Anzahl

Wiederholung mit Anfangsbedingung

Wiederholung mit Endbedingung

5.2 Wiederholung mit fester Anzahl

Die Anweisung für eine solche Wiederholung besteht aus einer Zeile. In dieser Zeile teilt man dem Computer die Zählvariable mit, ihren Anfangs- und Endwert, sowie die Schrittweite. Dadurch erhält man die gewünschte Anzahl an Durchläufen. Mit Anfangswert 2, Endwert 10 und Schrittweite 2 lautet der Befehl beispielsweise:

```
for (int i = 2; i < 10; i = i + 2){
   //Anweisungen
}
```

Wie oft werden in diesem Fall die Anweisungen durchlaufen?

4 mal

Sinnvoller ist natürlich der Anfangswert 0 und Schrittweite 1. Dann kann man als Endwert einfach die Anzahl der gewünschten Durchläufe nehmen. Wie lautet der Befehl, wenn man 15 Durchläufe will?

```
for (int i = 0; i < 15; i = i + 1)
```

Speziell für Schrittweite 1 gibt es eine Kurzform, man kann auch schreiben:

```
for (int i = 0; i < 15; i++)
```

Wie lautet also der Befehl für Anfangswert 5, Endwert kleiner 45 und Schrittweite 3? Wie oft wird die Wiederholung durchlaufen? Und für Anfangswert 0, Schrittweite 1 und gewünschten 22 Durchläufen?

for (int i = 5; i < 45; i = i + 3)

for (int i = 0; i < 22; i++)

Warum aber hat man überhaupt die Möglichkeit einer variablen Schrittweise? Ein ganz einfaches mathematisches Beispiel wäre, wenn man sich vom Computer z.B. alle Vielfachen der Zahl 17 ausdrucken lassen will, bis hin zur Quadratzahl von 17. Wie lautet hier die entsprechende Wiederholung? #Lsg10

Und was leistet diese kleine Methode? Spiele dazu das Programm mit den Eingabetupeln `(2;5)` und `(5;3)` durch.

```
public int was_mach_ich (int Zahl; int wert){
   int ergebnis = 1;
   for (int i = 0; i < wert; i++){
      ergebnis = ergebnis * Zahl;
   }
   return ergebnis;
}
```

5.3 Wiederholung mit Anfangsbedingung

Eine solche Wiederholung wird durchlaufen, solange eine zu Beginn abgefragte Bedingung gilt. Ganz allgemein schaut eine solche Wiederholung wie folgt aus:

While (Anfangsbedingung){

//Anweisungen

}

Ein kleines Beispiel (Zahl sollte dabei eine beliebig einzugebende natürliche Zahl sein). Verwende die Werte `3` und `7` für `Zahl` um herauszufinden was dieses Programm leistet!

```
public int und_was_bewirke_ich (int Zahl){
   int i = 0;
   int ergebnis = 0;
   while (i < Zahl){
      ergebnis = ergebnis + 5;
      i = i + 1;
   }
   return ergebnis;
}
```

Die Methode liefert als Ergebnis 5*Zahl

5.4 Wiederholung mit Endbedingung

Eine solche Wiederholung wird durchlaufen, bis die am Ende der Wiederholung stehende Bedingung ein Ende bewirkt. Somit wird diese Wiederholung mindestens einmal durchlaufen.

```
do {
   //Anweisungen
}while (Bedingung)
```

Ein kurzes Beispiel. Was bewirkt dieses Programm und warum ist es nicht sehr sinnvoll?

```
do {
   system.out.println(„Passwort eingeben“);
   passwort = eingabestring;
} while (passwort != „Abrakadabra“);
```

Es wird auf die richtige Passworteingabe gewartet;

man hat jedoch unendlich viele Möglichkeiten.

5.5 Aufgaben zu Wiederholungen

1. Erstelle in einer neuen Klasse `WIEDERHOLUNGEN` eine Methode `summeBisHundert`, welche die Summe der Zahlen 1 bis 100 berechnet. Erstelle in der gleichen Klasse anschließend eine weitere Methode `summeBisZahl`, welche die Summe der Zahlen 1 bis zur eingegebenen natürlichen Zahl berechnet. `#Lsg08`

2. Erstelle außerdem eine Methode `fakultaet`, mit deren Hilfe man die Fakultät einer Zahl `n (Produkt 1*2*…*(n-1)*n)` berechnen kann. `#Lsg12`

6 Felder

6.1 Einführung

Als letzten Konstrukt benötigen wir noch den Datentyp Feld (array). Hier geht es darum, dass mehrere Daten gleichen Typs abgespeichert werden. So wie man in der Mathematik Indizes verwendet: z.B.: $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, $f_4 = 3$, $f_5 = 5$, $f_6 = 8$, $f_7 = 13$ usw. (diese Folge kennen wir schon – wie heißt sie?), so speichert man hier die Daten in einem indizierten Feld ab, was man sich wie eine Zeile einer Tabelle vorstellen kann, oder wie einen Schrank mit durchnummerierten Fächern.

p[1]	p[2]	p[3]	P[4]	p[5]	p[6]	p[7]	…

Was ist alles nötig, um ein solches Feld mit 100 Eintragmöglichkeiten zu deklarieren und einem speziellen Feld, z.B. dem sechsten Feld, den Wert 10 zuzuweisen?

```
int[] feldname;
feldname = new int[100];
feldname[5] = 10;
```

Aufgepasst! Der Computer beginnt bei 0 zu zählen. Jetzt Ihr noch einmal. Wie wären also die Programmzeilen, um ein Feld mit Namen `Uebung` der Größe 60 zu erstellen und dem 15.Platz den Wert 50 zuzuweisen?

```
int[] uebung;
uebung = new int[60];
uebung[14] = 50;
```

Wie schaut also ein Programm aus, das alle Werte der Fibonacci-Folge bis zu einem einzugebenden n-ten Folgenglied berechnet und in einem Feld speichert?

```
class FIBOFOLGE{
    int[] fibo;

    FIBOFOLGE(){
        fibo = new int[20];
        fibo[1] = 1;
        fibo[2] = 1;
    }

    void fibonacci (int bisWert){
        for (int i = 3; i<= bisWert; i++){
            fibo[i] = fibo[i-1] + fibo[i-2];
        }
    }
}
```

Öffne dazu nun bitte die Klasse `FIBOFOLGE` und ergänzt die Methode `fibonacci` um eine Bildschirmausgabe. Überprüfe Deine Lösung, indem Du verschiedene Werte eingibst. Was passiert bei Werten über 20?

Besser wäre es also, die Größe des Feldes dynamisch zu gestalten, sie also erst festzulegen, wenn bekannt ist, wie viele Zahlen berechnet werden sollen. Ändere das vorliegende Programm nun entsprechend ab. Achte bei der Änderung des Quelltextes darauf, dass ein Feld der Größe 100 Belegungen von 0 bis 99 hat! `#Lsg18`

6.2 Verschiedene Möglichkeiten der Felddeklaration

Also zusammenfassend: um ein Feld zu erzeugen muss man es zuerst deklarieren und anschließend anlegen. Außerdem kann man es dann auch gleich noch belegen. Die Befehle, mit denen man also ein Feld anlegt, in dem alle Monate gespeichert werden, lauten ausführlich:

```
String[] monate;
monate = new String[11]
```

Anschließend könnte man das Feld dann belegen. Die Kurzform für obige Befehle lautet:

```
String[] monate = new String[11]
```

Will man das Feld deklarieren, anlegen und gleich belegen, so schreibt man:

```
String[] monate = {„Januar", „Februar", ... "Dezember"}
```

7 Arbeiten mit mehreren Klassen

7.1 Einführungsbeispiel

Nehmen wir nun mal eine Bank an, die mehrere Kunden führt. Hier wäre es sinnvoll, für jeden der Kunden ein Objekt der Klasse `KUNDE` anzulegen. Der Konstruktor legt also ein Konto mit einzugebenden Werten für `Kontostand` und `Besitzer` (des Kontos) an. Es soll Methoden `einzahlen` und `abheben` geben, sowie eine Möglichkeit, Kontostand (`standgeben`) und Besitzer (`namegeben`) abzufragen. Erstelle die Klasse und vergleiche sie mit der Vorgabe im Projekt `konto01`.

Anschließend legt man in einer Klasse BANK ein Feld `kundendatei` vom Typ `KUNDE` an, weil in der Bank ja auch die Kundendaten verwaltet werden. Das heißt nun also, dass hier Objekte gespeichert werden, die eben nicht vom Typ `int` oder `String`, sondern vom Typ `KUNDE` sind. Die Bank braucht außerdem einen Zähler um mitzuzählen, wie viele Konten schon vergeben sind.

Kommt jemand in die Bank, so soll man für ihn ein neues Konto anlegen können, indem man die Konstruktormethode der Klasse KONTO aufruft und ihr die Werte des aktuellen Kundens übergibt. Dieses Konto wird dann auch gleich in der `kundendatei` an der entsprechenden Stelle abgespeichert. Außerdem sollen in der Bank alle vier Methoden für ein Objekt der Klasse `KONTO` ausgeführt werden können. Wir nennen sie zur Unterscheidung `einzahlvorgang`, `abhebvorgang`, `kontostandGeben` und `kundennameGeben`.

Wie ausführlich Du dabei die Methoden schreibst, bleibt Dir überlassen.

Zusatz:

1. Ergänze Deine Klasse Konto um einen Kreditrahmen. Beim Abhebevorgang wird geprüft, ob dies überhaupt möglich ist.

2. Ergänze Deine Bank um eine Methode, die alle Kundendaten mit Kontonummer, Name und Kontostand ausgibt.

3. Ergänze eine Testmethode, die 3 Konten anlegt, alle Kundendaten ausgibt, dann zwei Abhebungen und drei Einzahlungen vornimmt und anschließend erneut alle Kundendaten ausgibt.

7.2 Schuldatenbank

Erstelle nun gemäß dem Kontoführungsprojekt ein weiteres, das zur Verwaltung von Schülerdaten verwendet werden soll.

Sinnvoll ist natürlich zuerst eine Klasse `SCHUELER` zur Verwaltung der Daten der einzelnen Schüler. Hier benötigen wir den Namen (`name`), die Anzahl der Verweise (`verweise`) und seinen Gesundheitszustand (`status`). Ein Schüler kann nur krank oder gesund sein, welchen Variablentyp verwendet man hier also am besten?

Der Konstruktor legt diesmal also einen Schüler mit einzugebenden Werten an. Weiterhin benötigt man die Methoden `statusGeben`, `nameGeben` und `verweisStand`, um die jeweiligen Attributwerte abzurufen. Außerdem benötigt man eine Methode `statusAendern`, die einen kranken Schüler gesund und einen gesunden Schüler krank macht, sowie eine Methode `verweisErteilen`, welche natürlich die Anzahl der Verweise hochzählt.

Anschließend legt man in einer Klasse `SCHULE` ein Feld `schuelerliste` an (von welchem Typ ist diese natürlich wieder?) Hier erzeugt der Konstruktor wieder eine entsprechende Liste mit 50 freien Plätzen. Dann benötigen wir eine Methode `schuelerAufnehmen`, welche wieder die Konstruktormethode in `SCHUELER` aufruft und einen neuen Schüler anlegt. Dabei wieder mit einem Zähler mitzählen! Außerdem benötigen wir auch in dieser Klasse die Methoden `verweisErteilen` und `statusAendern`, die ihrerseits für den entsprechenden Schüler die benötigte Methode aufruft. Als Eingabe genügt dabei ein `int`-Wert für die Position im Feld `schuelerliste`.
Als letztes erstellt noch eine Methode `krankeZaehlen`, die am Bildschirm ausgibt, wie viele Kranke es aktuell gibt, sowie eine Methode `schuelerDaten`, welche die Daten aller Schüler ausgibt.

II Fehlermeldungen

';' expected: Naja, er will halt dass Du nen Strichpunkt setzt. Könnte auch sein, dass ihm eine {-Klammer fehlt.

')' expected: Vielleicht hast Du bei einer Methode, die auf zwei Eingabewerte wartet, dazwischen einen Strichpunkt gesetzt, wobei da ein Komma hingehört. Oder es fehlt eben einfach eine)-Klammer.

... in ... cannot be applied to (): Ihr ruft eine Methode auf und übergebt dabei einen Parameter falsch.

Cannont find symbol – variable Muster: da hat sich wohl ein Schreibfehler beim Bezeichner einer Variablen eingeschlichen (vielleicht sogar nur ein Groß- statt Kleinbuchstabe).

Cannont find symbol – method MusterBeispiel(Int): da hat sich entweder ein Schreibfehler beim Bezeichner einer Methode eingeschlichen (evtl. wieder nur ein Groß- statt Kleinbuchstabe) oder der Parameter wurde noch nicht definiert.

Class, interface or enum expected: wahrscheinlich habt ihr eine Klammer zu } an der falschen Stelle stehen. Diese weglöschen!

Illegal start of type: vielleicht auch wieder, weil ihr vorher eine Klammer nicht geschlossen habt.

Incompatible Types – fount void but expected int: irgendwie habt ihr verschiedene Datentypen miteinander vermischt. Schaut mal ob da Alles passt!

Missing return statement: Der Computer denkt, Deine Methode sollte etwas als Ergebnis zurück liefern (kein `void` in der Methodenzeile), Du sagst ihm aber nicht was, oder Du sagst es in der falschen Zeile, also erst noch eine Klammer mehr oder weniger schließen.

Not a statement:

Possible loss of expression: möglicher Genauigkeitsverlust) Du rechnest irgendwas aus als double-Wert und weist es dann einem Integer-Parameter zu.

Reached end of file while parsing: (während des Übersetzens das Ende der Datei erreicht) Folglich denkt der Computer, das Programm ist noch nicht zu Ende. Warum? Weil eine schließende }-Klammer fehlt.

III Lösungen

(1)
```
void bewegeVertikal(int strecke){
    yPosition = yPosition + strecke;
    zeichnen();
}
```

(2)
```
void gutscheinDrucken(){
     // Den Ausdruck eines Tickets simulieren.
     if (bisherGezahlt < preis) {
         System.out.println("Erstmal ganz bezahlen, Geizhals");
     }
     else {
         System.out.println("###################");
         System.out.println("# Ihr Gutschein");
         System.out.println("# hat den Wert");
         System.out.println("# " + preis + " Euro.");
         System.out.println("# Ihr Restgeld beträgt");
         System.out.println("# " + (bisherGezahlt-preis) + " Euro.");
         System.out.println("###################");
         System.out.println();
         // Hier passiert wieder etwas mit "wozu"
         wozu = wozu + preis;
         // Die bisherige Bezahlung zurücksetzen.
         bisherGezahlt = 0;
      }
}
```

(3)
```
import java.lang.Math;

class PYTHAGORAS{
    double aSeite;
    double bSeite;
    double cSeite;

    PYTHAGORAS{
        aSeite = 0;
        bSeite = 0;
        cSeite = 0;
    }

    double cBerechnen(double aWert, double bWert){
        cSeite = aWert * aWert + bWert * bWert;
        cSeite = sqrt(cSeite);
        return cSeite;
    }
}
```

(4)
```
double WerteEingeben(int Eingabewert, char Waehrung){
    switch (Waehrung){
        case 'D': {
            Ausgabewert = Eingabewert * 1.95583;
        } break;

        // alle folgenden Werte sind vom 14.03.2017
        case 'S': {
            Ausgabewert = Eingabewert * 7.01;
        } break;
        case 'P': {
            Ausgabewert = Eingabewert * 16.54;
        } break;
        case 'D': {
            Ausgabewert = Eingabewert * 1.06;
        } break;
        case 'K': {
            Ausgabewert = Eingabewert * 9.16;
        } break;
        default: {
            System.out.println("Diese Währung gibt es nicht");
            Ausgabewert = 0;
        } break;
    }
    return Ausgabewert;
}
```

(5)
```
void LaengeUndBreiteSetzen(int lwert, int bwert){
    breite = bwert;
    laenge = lwert;
    zeichnen();
}
```

(6)

```
void kippen(){
    int Hilfsvar;
    Hilfsvar = laenge;
    laenge = breite;
    breite = Hilfsvar;
    zeichnen();
}
```

(7)

```
void nachOben(){
    bewegeVertikal(-20); // andersrum wär's die falsche Richtung
}
void nachUnten(){
    bewegeVertikal(20);
}
```

(8)

```
int summeBisHundert(){
    int summe = 0;
    for (int i=1; i<=100; i++){
        summe = summe + i;
    }
    return summe;
}
int summeBisZahl(int zahl){
    int summe = 0;
    for (int i=1; i<=zahl; i++){
        summe = summe + i;
    }
    return summe;
}
```

(9)

```
import java.lang.Math;
class MITTERNACHT{
    double aWert;
    double bWert;
    double cWert;
    int AnzahlLSG;
    double Lsg1, Lsg2;
    MITTERNACHT(){
        aWert = 0;
        bWert = 0;
        cWert = 0;
    }
    void werteEinlesen(double a, double b, double c){
        aWert = a;
        bWert = b;
        cWert = c;
    }
    int anzahlLSG(){
         if (bWert*bWert-4*aWert*cWert < 0 || aWert == 0){
            System.out.println("Keine Lösung");
            AnzahlLSG = 0;
        }
        else {
            if (bWert*bWert-4*aWert*cWert == 0) {
                System.out.println("Genau eine Lösung");
                AnzahlLSG = 1;
            }
            else {
                System.out.println("Zwei Lösungen");
                AnzahlLSG = 2;
            }
        }
        return AnzahlLSG;
    }
    void loesungen(){
        if (AnzahlLSG == 1) {
            Lsg1 = -bWert/(2*aWert);
            Lsg2 = -bWert/(2*aWert);
        }
        else {
            if (AnzahlLSG > 1) {
                Lsg1 = ((-bWert + Math.sqrt(bWert*bWert-4*aWert*cWert))/(2*aWert));
                Lsg2 = ((-bWert-Math.sqrt(bWert*bWert-4*aWert*cWert))/(2*aWert));
            }
        }
    }
    void ausgabeLoesungen(){
        System.out.println("**********************");
        System.out.println("** Lösung 1 ist: " + Lsg1);
        System.out.println("** Lösung 2 ist: " + Lsg2);
        System.out.println("**********************");
        System.out.println();
    }
}
```

(10)

```
for (int i = 1; i <= 17; i++){
   wert = i*17;
   System.out.println(wert);
}
```

(11)

```
void laengeSetzen(int neueLaenge){
    laenge = neueLaenge;
    zeichnen();
}
```

(12)

```
int fakultaetVonZahl(int Eingabe) {
    int ergebnis = 1;
    for (int i = 1 ; i <= Eingabe; i++) {
        ergebnis = ergebnis * i;
    }
    return ergebnis;
}
```

(13)

```
void geldEinwerfen(int betrag){
     if (betrag < 0) {
          System.out.println("Wert über Null eingeben, Hirni");
     }
     else {
         bisherGezahlt = bisherGezahlt + betrag;
     }
}
```

(14)

```
class VERSANDRECHNER{
    /**Versandpauschalen
     * bis unter 10 Euro 3,97 Euro
     * bis unter 20 Euro 2,97 Euro
     * bis unter 30 Euro 1,97 Euro
     * bis unter 40 Euro 0,97 Euro
     * ab 40 Euro 0 Euro
     **/
    double preis;
    int klasse;

    void klasseberechnen(int bestellwert){
        klasse = bestellwert/10;
        if (klasse > 4){
            klasse = 4;
        }
        double kohle = preisBerechnen(klasse);
        System.out.println(kohle);
    }

    private double preisBerechnen(int klasse){
        switch(klasse){
            case 0:{
                preis = 3.97;
            }break;
            case 1:{
                preis = 2.97;
            }break;
            case 2:{
                preis = 1.97;
            }break;
            case 3:{
                preis = 0.97;
            }break;
            case 4:{
                preis = 0;
            }break;
            default:{
                System.out.println("Fehleingabe");
            }break;
        }
        return preis;
    }
}
```

(15)

```
void StreckenXY(int xFaktor, int yFaktor){
    breite = breite * xFaktor;
    laenge = laenge * yFaktor;
    zeichnen();
}
```

(16)
```
import java.lang.Math;

class PYTHAGORAS{
    double aSeite;
    double bSeite;
    double cSeite;

    PYTHAGORAS(){
        aSeite = 0;
        bSeite = 0;
        cSeite = 0;
    }
    void cBerechnen(double aWert, double bWert){
        cSeite = aWert * aWert + bWert * bWert;
        cSeite = Math.sqrt(cSeite);
    }
}
```

(17)
```
void groesser(int AenderWert){
    breiteSetzen(breite + AenderWert);
    laengeSetzen(laenge + AenderWert);
    zeichnen();
}
```

(18)
```
class FIBOFOLGE{
    int[] fibo;

    void fibonacci (int bisWert){
        fibo = new int[bisWert+1];
        fibo[1] = 1;
        fibo[2] = 1;
        for (int i = 3; i<= bisWert; i++){
            fibo[i] = fibo[i-1] + fibo[i-2];
        }
        for (int i = 1; i <= bisWert; i++){
            System.out.println("Die " +i+ "-te Fibonaccizahl ist " + fibo[i]);
        }
    }
}
```

(19)
```
double cBerechnen(double aWert, double bWert){
    cSeite = aWert * aWert + bWert * bWert;
    cSeite = Math.sqrt(cSeite);
    System.out.println("Die eine Kathete ist " + aSeite);
    System.out.println("Die andere Kathete ist " + bSeite);
    System.out.println("Die Hypotenuse ist " + cSeite);
    return cSeite;
}
```

(20)
```
void nachLinks(){
    bewegeHorizontal(-20);
}
```

(21)
```
double cBerechnen(double aWert, double bWert){
    if (aWert < 0){
        System.out.println("Der Wert von Seite a ist kleiner Null");
    }
    else {
        if (bWert < 0) {
            System.out.println("Der Wert von Seite a ist kleiner Null");
            }
        else {
            cSeite = aWert * aWert + bWert * bWert;
            cSeite = Math.sqrt(cSeite);
            System.out.println("Die eine Kathete ist " + aWert);
            System.out.println("Die andere Kathete ist " + bWert);
            System.out.println("Die Hypotenuse ist " + cSeite);
            }
        }
    return cSeite;
}
```

(22)
```
void quadratmachen(){
    laenge = breite;   //oder umgekehrt
    zeichnen();
}
```