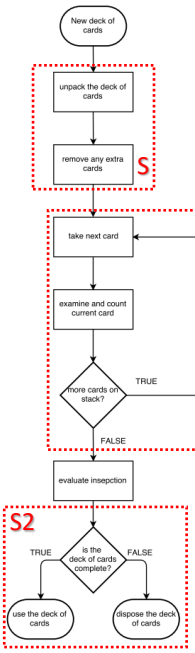
Modulinhalte

Anleitung, Flowchart, Algorithmus, Sequenz, Iteration, Selektion



Aus der Beschäftigung mit Anleitungen und Flowcharts wurden die Bezeichnungen der wesentlichen Strukturelemente von Programmen entwickelt:

Ein **Algorithmus** ist eine endliche Folge (Ablauf) aus eindeutigen und ausführbaren Anweisungen. Ein Programm ist ein Algorithmus, der in einer formalisierten Sprache abgefasst ist und maschinell ausgeführt werden kann.

Eine (S) **Sequenz** ist eine Folge einzelner Ablaufschritte.

Eine (S2) **Selektion** ist eine von einer Bedingung abhängige Verzweigung.

(I) **Iteration** bezeichnet eine bedingte Wiederholung von Anweisungen.

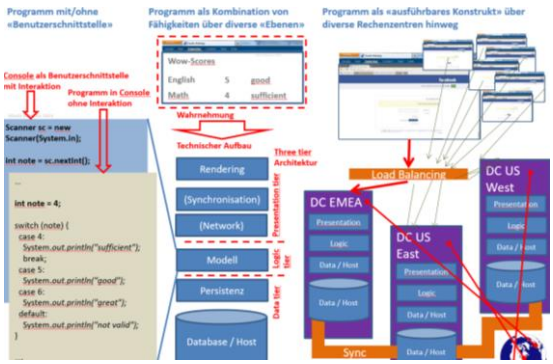
Iterationen und Selektion ist gemeinsam, dass sie durch Bedingungen gesteuert werden. In Java können beide je mit zwei unterschiedlichen Kontrollstrukturen für unterschiedliche Aufgabenstellungen umgesetzt werden.

Die Geschichte von Java

Java wurde wurde ursprünglich auch als **Oak** (Object Application Kernel) bezeichnet, und **1991/1992** als The Green Project bei Sun Microsystems entwickelt. Wichtiger **Haupt-Entwickler war James Gosling**. Hoffnungen auf Nutzung in Haushalts- und Freizeitgeräten zerschlugen sich, stattdessen wurde das Konzept der **Java-Applets entwickelt, mit dem Ziel, diese im gerade entstehenden Internet** zu nutzen. Doch auf dem Client wurde Java nie besonders stark, sondern setzte sich v.a. im Bereich der Server-Programme durch. **2010 wurde Sun durch Oracle aufgekauft**. Java ist seit Jahrzehnten auf den ersten Plätzen unterschiedlicher Listungen populärer Sprachen.

Programmausschnitt & Java-Programme

Die ersten Programme in diesem Modul sind so einfach, dass dies ev. zur Frage führt, ob dies überhaupt Programme, so wie man diese kennt, sind. Tatsächlich zeigt die folgende Abbildung ganz links, dass **in diesem Modul nur ein Ausschnitt «vollständiger» Programme** (in der zweiten Spalte, 3. Spalte zudem Verteilung) angesehen wird.



Programme entstehen traditionell durch Übersetzung eines **Quelltextes in Maschinencode**. In **Java hingegen** wird aus dem Quelltext durch einen Compiler ein sog. **Bytecode** (.class-Dateien) erzeugt, der dann von einer **virtuellen Maschine** (JVM) ausgeführt wird. Diese JVM muss pro Kombination von Betriebssystem und Hardware nur einmal programmiert werden, damit auf dieser «Plattform» dann alle Java-Programme ausgeführt werden können. Die JVM wird in zwei Varianten angeboten, als **JDK**, Java Development Kit und **JRE**, Java Runtime Environment. Zudem gibt es unterschiedliche Varianten je nach Nutzungsbereich.

Innerhalb der JVM können zudem noch diverse **andere Sprachen** ausgeführt werden, wie Groovy, Scala, etc.

Wichtige **Eigenschaften von Java sind**: Portabilität, Einfachheit, Objektorientierung, Verteilung, etc.

Modulinhalte

Java Syntax lesen und schreiben mit HelloWorld

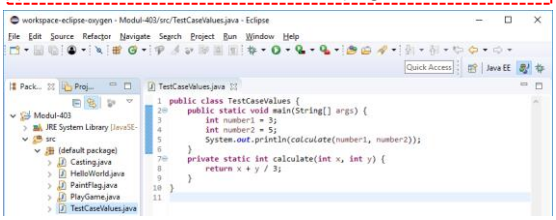
Das Einstiegsprogramm in Java, HelloWorld:

```
public class HelloWorld {
  public static void main (String[]args) {
    System.out.println("Hello World");
  }
}
```

Wichtige Prinzipien sind **lesen können von Klammern**, erkennen der Paare, **Zusammengehörigkeit** und folglich erkennen können des Klassenkörpers und des **Körpers** der Main-Methode. Anstelle von Körper wird auch von Block gesprochen.

Java Editoren, IDE's und Eclipse

Eine der populärsten Programmierumgebungen, oder eben **IDE's (Integrated development environment)**, so genannt, da diese Programme kompilieren, ausführen und mehr können, ist Eclipse. Das liegt daran, dass Eclipse Open Source ist, und viele Plug-Ins dafür bestehen. Eine andere populäre IDE ist IDEA von der Firma JetBrains. Für Eclipse ist ein **Workspace** anzulegen, in dem dann Java-Projekte angelegt werden. Die Klassen darin lassen sich in sog. **Packages** (ähnlich Ordnern für z.B. Office Dokumente) organisieren.



Einlesen

Im Kontext des Moduls 403 erfolgt die «Interaktion» mit dem Programm jeweils über die sog. **Console**. Die Ausgabe auf die Console ist oben aus dem Beispiel HelloWorld ersichtlich.
Für das Einlesen von Daten dient die MyTools.jar Bibliothek mit der **Helper**-Klasse, um verschiedene Datentypen einzulsen:

```
double value = Helper.readDouble();
Char c = Helper.readChar();
```

Datentypen

Die folgende Liste zeigt die **primitiven Datentypen** in Java, und zeigt die Deklaration (erste Zeile fett: Datentyp und Variablenname) und die optionale Initialisierung (ab Gleich-Zeichen) jeweils mit einem zulässigen Wert:

```
boolean isMember = true; //or: false
char letter = 'a';
byte smallNumber = 127;
short mediumNumber = 32767;
int number = 2147483647;
long bigNumber = 9223372036854775807;
float weight = 23.4;
double amount = 12.25;
```

Ausser **Konstanten (final)** kann einer Variablen jederzeit wieder ein neuer Wert zugewiesen werden.

Der Datentyp **String** ist notwendig, um mit alphanumerischen (a…z, 0…9) **Zeichenfolgen** zu arbeiten, es lassen sich darin z.B. ein Name speichern:

```
String name = "Petra";
```

Je nach Datentyp stehen Operatoren wie *, +, - und / bereit oder eben nicht, und wirken entsprechend dem Datentyp, so dient + bei Zahlen dem Rechnen, bei String hingegen der Verkettung. Bei Zahlen gibt es die **Spezialoperatoren ++ und --** zum hoch und runterzählen.

Werte eines Datentyps mit höherer Genauigkeit lassen sich nur mittels sog. **Casting** zuweisen womit Java signalisiert werden muss, dass der Genauigkeitsverlust ausdrücklich erwünscht ist:

```
double exactValue = 2.33;
int value = (int) exactValue;
```

Werden **mehrere Werte die einen inneren Zusammenhang** haben (z.B. Begriffe in einer Einkaufsliste, Rundenzeiten bei einem Sportlauf) erfasst, ist es häufig umständlich bis nicht möglich, z.B. für jeden Läufer eine eigene Variable zu erstellen. In diesen Situation wird ein sog. **Array** angelegt. Unter einem Namen werden dann mehrere gespeicherte **Werte mittels eines Indexes (Beginn bei Null!) angesprochen**:

```
String jobList[] = new String[5];
jobList[0] = "Bäcker";
jobList[1] = "Astronaut";
System.out.println(jobList[0]);
```

Modulinhalte

Kontrollstrukturen, Operatoren und die Wahrheit

Selektionen, also Entscheidungen, und Iterationen, Wiederholungen, können in Java je mit zwei Varianten programmiert werden. Sie alle werden Kontrollstrukturen genannt, und **abhängig von Bedingungen (resp. Werten bei switch)** verhält sich ein Programm unterschiedlich:

```
import java.util.Scanner;

public class Fraction {

    public static void main(String[] args) {
        System.out.println("Please enter the price:");
        double price = Helper.readDouble();

        if(price < 100) {
            price = price * 0.9;
        }

        System.out.println("Your price " + price);
        sc.close();
    }
}
```

Die fett hervorgehobene Zeile leitet hier die if-Kontroll-struktur ein. In der Klammer hinter dem Schlüsselwort if steht eine **Bedingung, die einen der Wahrheits-, boolean-Werte true oder false ergeben muss!** In der Klammer könnte folglich auch der Wert true oder false stehen, was die Flexibilität des if unnütz werden liesse, und nur erwähnt ist, um zu verdeutlichen, welche Werte dort resultieren.

Andere **Operatoren** sind aus folgender Liste ersichtlich:

==	Prüfung auf Gleichheit
>, <=, >=	grösser, kleiner gleich, grösser gleich
!=	ungleich
!	logisches Gegenteil (true wird false)

Mehrere **boolean Ausdrücke kombinieren**:

&&	Und-Verkettung, true, wenn beide true
||	Oder-Verkettung, true, wenn einer true

Kontrollstrukturen if, while, for, switch

Selektionen, also Entscheidungen, und Iterationen, Wiederholungen, können in Java je mit zwei Varianten programmiert werden. Sie alle werden Kontrollstrukturen (KS) genannt. Mit Ausnahme von switch, werden alle KS durch Bedingungen gesteuert, d.h. Ausdrücke, die true oder false ergeben.

if-Kontrollstruktur: das erste if kann durch weitere else if oder nur ein else ergänzt werden.

```
int price = 105;
if(price < 100) {
            price = price * 0.9;
        } else if(price >= 100 && price < 1000) {
            price = price * 0.8;
        } else {
            price = price * 0.7;
}
```

while-Kontrollstruktur: hier die **kopfgesteuerte** Variante; **fussgesteuert (Block mind. 1x durchlaufen)** beginnt es mit do und das while steht hinter der schliessenden Klammer und dahinter ein Semikolon!

```
int i = 0;
while(i < 10) {
            i++;
            System.out.println("i = " + i);
}
```

for-Kontrollstruktur: die Besonderheit hier sind die drei Abschnitte: 1. Deklaratаion und Initialisierung Variable, 2. Bedingung, 3. Rechnen:

```
for(int a = 0 ; a < 10 ;  a++) {
        x++;
        System.out.println(x + ": Hopp Schwiiz");
}
```

switch-Kontrollstruktur: die Besonderheit hier ist, dass hinter dem Schlüsselwort eine Variabel steht, abhängig von deren Wert wird ein Case-Block ausgeführt, dessen Verlassen mittels break erzwungen wird; default wird ausgeführt, wenn kein anderer Wert passt.

```
switch (note) {
         case 1:
          System.out.println("ungenügend");
          break;
          ...
         default:
          System.out.println("keine gültige Note");
}
```

Modulinhalte

Methoden

Im Modul 403 wird mit Methoden ein erstes Gliederungselement vorgestellt, welches für grössere Programme unverzichtbar ist. Dabei ist bisher immer bereits die Main- Methode vor Augen und ein Muster:

```
public static void main(String[] args) {
   ...
}
```

Eigene Methoden können nach dem gleichen Muster geschrieben werden.

```
01: public class Methoden {
02:  public static void main(String[] args) {
03:    printHello();
04:    printMessage ("World");
05:    int a = 5;
06:    int b = 2;
07:    int c = calculate(a, b);
08:    System.out.println(c);
09:   }

10:  public static void printHello() {
11:    System.out.println("Hello");
12:  }

13:  public static void printMessage(String message) {
14:    System.out.println(message);
15:  }

16:  public static int calculate(int b, int a) {
17:    return b * 5 / a;
18:  }
19: }
```

Die Main-Methode ruft verschiedene Methoden innerhalb der Klasse auf, die um es einfach zu halten auch alle public sind.

Zeile 10: denkbar **einfachste Methode, die über den Namen aufgerufen wird** (siehe Zeile 3). Weil diese Methode in der Klasse selbst steht, steht davor kein Variablennamen, wie dies z.B. bei Verwendung des Scanners der Fall ist, z.B.: sc.nextInt();

Zeile 13: **Methode mit einer Parameter-Deklaration**, und Aufruf von Zeile 4, welche die Vorschrift *erster Parameter ist ein String-Wert* erfüllt.

Zeile 16: Eine **Methode mit zwei Parametern, einer sog. Parameterliste**, bei der die Reihenfolge, nicht Benennung im Vergleich mit der Aufrufzeile 7 entscheidend ist. In der Methode hat also b den Wert 5, das Ergebnis ist also 12 (gerundet!). Vor dem Ergebnis steht **return, der Befehl an Java, den nachfolgenden Wert an die Stelle des Aufrufs zurück zu senden**, was auf Zeile 7 passiert, wobei der Wert in c gespeichert wird, und in der nächsten Zeile ausgegeben wird.

Fehler, Testen, Testfälle, Debuggen

Fehler: Es gibt verschiedene Fehlerkategorien, die von harmlos, z.B. Erschwernis bei der Eingabe bis hin zu schwerwiegend, Programm stürzt durch ungültige Werte ab, reichen.

Testen, Testarten: Tests lassen sich nach Zeitpunkt und Umfang unterscheiden, wie z.B. Einzeltest, Integrationstest, Benutzertest. Wichtig sind auch die Begriffe **Black-Box-Test** (es wird nur der Input, gewählte Aktion und Output beobachtet) und **White-Box-Test** (das Verhalten des Programms mit Input und Aktion wird im Quelltext untersucht). Dann sind auch automatische und manuelle Tests zu unterscheiden.

Testfälle: Sind **tabellarische Auflistungen** von Input, Aktion resp. Befehl, erwartetem Output, tatsächlichem Output und Angabe, ob die Output-Wert übereinstimmen. Bei mehreren Testfall-Tabellen, kombiniert mit unterschiedlichen Testpersonen und definierten Testumgebungen spricht man von einem Testdrehbuch.

Testautomatisierung: Die Testautomatisierung kann auf zwei Arten realisiert werden: 1. **Record- und Playback-Software**, die für End-to-End-Tests die Benutzerinteraktion mittels Programm oder Browser aufzeichnet und später wieder abspielen kann. **2. Programmierte Tests** auf unterschiedliche Stufen, angefangen von End-To-End-Tests bis hin zu Tests auf Methodenebene. Tests von Methoden wurden zusammen mit agilen Entwicklungs und Buildmethoden und Java besonders JUnit Bibliothek gängig und z.B. als Nunit auch in andere Sprachen übertragen, und damit **automatische Regressionstest** möglich.

Debuggen: Fehler können in der IDE untersucht werden, in dem **Haltepunkte** im Quelltext definiert werden, an denen dann z.B. der Status von Variablen und Bedingungen untersucht werden kann. Ab dort kann das Programm schrittweise weiter laufen.