Was du bis jetzt können musst (Teil 1)

# Allgemeines

* Den Unterschied zwischen \*.java und \*.class-Dateien erklären können.*\*.java-Dateien sind das Drehbuch, \*.class-Dateien der Film*
* Den Begriff kompilieren erklären können.  
  *Durch das Kompilieren wird aus dem Drehbuch der Film erstellt.*
* Erläutern können, warum ein Java-Programm kompiliert werden muss.  
  *Nur der Film kann später von anderen auch angeschaut bzw. benutzt werden.*
* Erläutern können, warum nicht auf jedem Computer Java-Programme ablaufen.  
  *Da es nicht für jedes Betriebssystem einen Projektor gibt.*
* Das Programm BlueJ kennen und starten können
* Ein- und mehrzeilige Kommentare einfügen können und wissen, was kommentiert werden muss.  
  *Ein einzeiliger Kommentar beginnt mit //... , ein mehrzeiliger wird „geklammert“ durch /\*...\*/ .  
  Es wird das Ziel jeder Methode angegeben, außer diese erschließt sich vollständig aus dem Methodennamen.  
  Es wird jedes Attribut und jede Variable erklärt, außer wenn sich dies vollständig aus dem Namen erschließt.  
  Der Methodenrumpf wird durch einzeilige Kommentare, die als Zwischenüberschriften dienen, gegliedert.*
* Wissen, was das Wort „implementieren“ bedeutet.  
  *„umsetzen“, hier im Sinne von „programmieren“. Wichtig für schriftliche Tests.*

# Die Grundbegriffe des Programmierens[[1]](#footnote-1)

* Eine Klasse definieren   
  *public – class – EigenerName – { – }[[2]](#footnote-2)*
* Attribute definieren   
  *private[[3]](#footnote-3) – Datentyp[[4]](#footnote-4) – eigenerName ;*
* Eine Variable definieren können  
   *Datentyp – eigenerName ;*
* Alle wichtigen, primitive Datentypen angeben und erklären können  
  *int (Ganze Zahlen), double (Kommazahlen: 3.14), String (Zeichenketten), char (1 Zeichen), boolean (Wahrheitswert: true, false)  
  String ist genau genommen kein primitiver Datentyp, da er aus char gebildet wird. Daher auch das große S bei String.  
  Es gibt noch mehr primitiven Datentypen, aber diese genügen am Anfang.*
* Eine Variable mit eine Startwert initialisieren können.  
  *eigenerName - = - Wert ;   
  Beispiel: alter = 18; name = „Max“;  
  Variablen, die nur innerhalb von Methoden auftauchen sollten bei der Definition bereits mit einen Startwert initialisiert werden, um Fehler zu vermeiden.  
  Beispiel: Datentyp – eigenerName - = - Wert ;*
* Zeichenketten verknüpfen können  
  *Beispiel: String antwort = „Ich heisse: “ + name;*
* Welche Eigenschaft wird durch „public“ und „private“[[5]](#footnote-5) definiert und was bedeuten sie?  
  *Sichtbarkeit oder Zugriffsrecht: Bei public darf jeder auf das Atrribut / Methode zugreifen. Bei private nur das Objekt selbst.*
* Grundlegende Programmierrichtlinien kennen (Einrückungen, Groß/kleinschreibungen, KamelHöckerschreibweise)  
  *Alles, was Teil von etwas anderem ist, wird um 1 Tabulator eingerückt.  
  Klassennamen beginnen immer mit einem Großbuchstaben, Methoden (Funktionen und Prozeduren) und Attribute sowie Variablen immer mit einem Kleinbuchstaben.  
  Da Klassen- und Methodennamen keine Leerzeichen (keine Sonderzeichen) enthalten dürfen, aber sprechende Namen wichtig sind, beginnt jedes innere Wort eines Namens mit einem Großbuchstaben: alleDatenAusgeben()*
* Die Aufgabe eines Konstruktors angeben können  
  *Ein Konstruktur erzeugt gemäß des Bauplans (Klasse mit Attributen, Methoden) ein Objekt der Klasse und sollte alle Attribute initialisieren.*
* Standardkonstruktoren .....  
  *Ein Standardkonstruktor hat keine Parameterliste und initialisiert alle Attribute mit fest vorgegebenen Startwerten.*
* ...... und Konstruktoren mit Parametern definieren  
  *Diese Konstruktoren initialisieren ebenfalls alle Attribute, allerdings benutzen sie dafür die Wert aus der Parameterliste.*
* Die Werte der Eingangsparameter in den Attributen speichern können  
  *this[[6]](#footnote-6).attributName - = - parameterWert ;  
  Beispiel: this.alter = neuesAlter;*
* Wissen, was ein Methodenkopf, ...

public void alterAusgeben() {

System.out.println(alter)

}

* was ein –rumpf ist.
* Die Bestandteile eines Methodenkopfes in der richtigen Reihenfolge angeben können.  
  Sichtbarkeit – Datentyp – methodenName ( ev. Parameterliste )
* Den Unterschied zwischen Prozedur und Funktion kennen  
  *Eine Prozedur liefert keine Antwort an den Aufrufenden zurück, daher ist der Rückgabewert void. Alltagsbeispiel: vonDerTafelAbschreiben() ist eine Prozedur von Schüler, da sie keine Antwort an den auffordernden Lehrer zurückgeben. Aber Sie produzieren einen Hefteintrag.  
  Eine Funktion gibt dem Aufrufenden eine Antwort. Die Art der Antwort (Datentyp) muss dabei vorher festgelegt werden. Alltagsbeispiel wäre bei einem Schüler die Funktion entschuldigungGeben() bei der der Schüler in Form einer Zeichenkette (String) eine Entschuldigung an den Lehrer zurückgibt.*
* Eine Ausgabe in das Editorfenster schreiben können  
  *Beispiel: System.out.println(„Hallo Welt“)  
  Ohne Zeilenumbruch am Ende: System.out.print(„Hallo Welt“)*
* Den Debugger benutzen können  
  *Stopp setzen und dann mit Step, Step into den Programmablauf nachvollziehen können.*
* Ein Objekt im Quelltext erzeugen können  
  *new – Konstruktorname – ( - Werteliste - )  
  Beispiel:   
  Hausaufgabe neueHA = new Hausaufgabe(„M“, „20161101“, „B: S 12/1“);*
* Ein Methode eines Objektes im Quelltext aufrufen können.  
  wer . was ( Werteliste );  
  Beispiel: *String alleInfos =* *neueHA.getInfos();* *neueHA.setDate(„20161101“);*

# Kontrollstrukturen

* Eine bedingte ein/zweiseitige Verzweigung angeben können.  
  if – ( - Ja/Nein-Frage - ) – {  
   //Ja-Fall  
  } else {   
   //Nein-Fall; der nicht kommen muss  
  }
* Du kennst alle Vergleichsoperatoren  
  *>, <, <=, >=, !=*
* *(Du kennst alle booleschen Operatoren)  
  Und && bzw. Oder ||  
  Beispiel* if (alter >= 16 && alter <18)  
   System.out.println(„Bier ist erlaubt.“);  
   }

# Arbeiten mit einer GUI

* Du weißt, wofür die Abkürzung GUI steht  
  *Graphical User Interface (-> User Interface Guidelines)*
* Du weißt, wo du das Programm JavaEditor im Schulnetzwerk findest
* Du kennst den Unterschied zwischen einer \*.jfm-Datei und der gleichnamigen \*.java-Datei  
  *name.jfm-Datei enthält die geometrische Beschreibung.   
  name.java, den zugehörigen Java-Quelltext.*
* Du kannst eine GUI mit

JLabels, JTextFields, JNumberFields, JTextArea, JCheckBox, JButton

anlegen

* Du kannst die GUI in dein BlueJ-Projekt übertragen.  
  *WICHTIG: Dateien dürfen ausschließlich nur in 1 Editor geöffnet werden.*
* Du kannst die Inhalte aller Eingaben in der Konsole ausgeben.  
  System.out.println(jTextField1.getText());
* Du kannst die vom Benutzer eingegebenen Daten an eine andere Klasse weitergeben.  
  MeineKlasse meinObjektname = new MeineKlasse(jTextField1.getText());  
  meinObjektname.setzeWert(jTextField1.getText());
* Die kannst den Rückgabewert einer Methode in der GUI anzeigen lassen  
  jTextField1.setText(meinObjektname.gibWert());

1. in Java [↑](#footnote-ref-1)
2. Diese Schreibweise ist an Syntaxdiagramme angelehnt. Eingrahmte Begriffe sind mehr oder minder frei vom Programmierer wählbar. Die – geben an, dass an dieser Stelle ein Leerzeichen zu sein hat. [↑](#footnote-ref-2)
3. Attribute sollte in der Regel immer *private* sein. Es gibt hierzu aber auch andere Meinungen. In der Programmiersprache Python sind z.B. alle Attribute standardmäßig *public* und nur, wenn man es möchte, kann man sie *private* setzen. [↑](#footnote-ref-3)
4. Der Datentyp ist vom Programmierer frei wählbar, allerdings auf bestimmte Datentypen mehr oder eingeschränkt. [↑](#footnote-ref-4)
5. „protected“ spielt im Anfangsunterricht aus meiner Erfahrung heraus keine Rolle. [↑](#footnote-ref-5)
6. Die Verwendung von this ist zwar eigentlich nicht nötig, betont aber nochmals den Unterschied zwischen Attribut und lokaler Variable bzw. Parameter. Zudem werden Zeilen wie alter = alter; verhindert, die nur zu unklaren Fehlern führen. [↑](#footnote-ref-6)