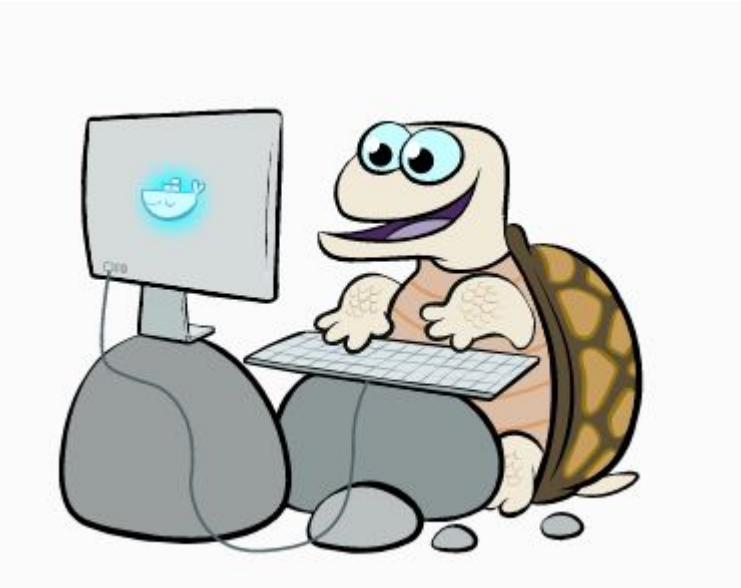
Quelle: Oracle.com

Skills

- Keine Skills Erforderlich

- Ziel ist Anfänger Skills in Java zu erlangen



Quelle Bild : Docker.com

Trainer Christopher Dierig

Ausbildung - 1und1 Internet AG

Junior Software Developer 1und1 Internet AG

Software Developer 1und1 Internet AG

Lead Developer Bisnode Informatics GmbH

Senior Consultant Sobek Innovations UG

(haftungsbeschränkt)

→ 13

Jahre Java Erfahrung

Java, die Sprache und ihre Entstehung

Die Entstehung

- Entwickelt 1990 von der Firma Sun Mircosystems
- Ziel : Entwicklung einer Allgemeinen Hochsprache
- Offizielle Vorstellung im Netz 1995
- System Unabhängig
- Auf dem Level C/C++

Die Virtuelle Maschine

- Die Plattformunabhängigkeit ist nur auf den Plattformen gegeben, für die eine virtuelle Maschine in der richtigen Version existiert!
- Was aber ist die virtuelle Maschine?
- Consumer Elektronik Bereich, d.h. zum Einsatz auf Microcontrollern entwickelt
- Spezieller Java Chip -> als Ausgang
- Dieser Wird von der Virtuellen Maschine emuliert

Vorteile / Nachteile

+ Programme laufen Stabiler
+ Plattform Unabhängigkeit
+ Java ist einfach

- Geschwindigkeitsverlust

- “Java ist Langsam”
  - Stimmt aber nicht!
  - Geschwindigkeitsunterschied Minimal

Eigenschaften von Java

Eine der herausragendsten Eigenschaften von Java ist die Plattformunabhängigkeit.

Dadurch ist es z.B. möglich Entwicklung und Ausführung von Programmen auf unterschiedlichen Betriebssystem auszuführen.

Dies ist vor allem bei Internet-Anwendungen interessant, da es für Windows-Betriebssysteme zwar die größte Auswahl an Entwicklungswerkzeugen gibt, die Applikationen meistens jedoch auf Unix/Linux-System eingesetzt werden.

Objektorientierung

- Um eine möglichst rasche Verbreitung zu erleichtern, wurde die Sprachsyntax und der Aufbau sehr stark an C++ angelehnt

- Auf einige Features von C++, die teilweise dem objektorientierten Gedanken widersprechen, oder hauptsächlich zur Fehlerhäufigkeit bei C++ Programmen beitragen, wurde dabei verzichtet.

Multithreaded

- Java ist eine Multithread fähige Programmiersprache

Beispielanwendung :

- Ein Thread kümmert sich um die Aktualisierung der Bildschirmausgabe.
- Ein Anderer um das Herunterladen von Daten von einem Webservice

Robust

- Verzicht auf den direkten Zugriff auf den Speicher
- Speicher Verwaltung übernimmt das System
- Strenge Objektorientierung und strenge Typenprüfung tragen zu einer starken Typsicherheit bei.
- Java verzichtet auf Operatoren Overloading
- Verzicht auf Mehrfachvererbung
- keine automatische Typumwandlungen

Einrichtung der IDE

- IDE nach Wahl (Eclipse, Netbeans, IntelliJ IDEA etc…)
- Java SDK
- APACHE Maven

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

https://netbeans.org/downloads/index.html

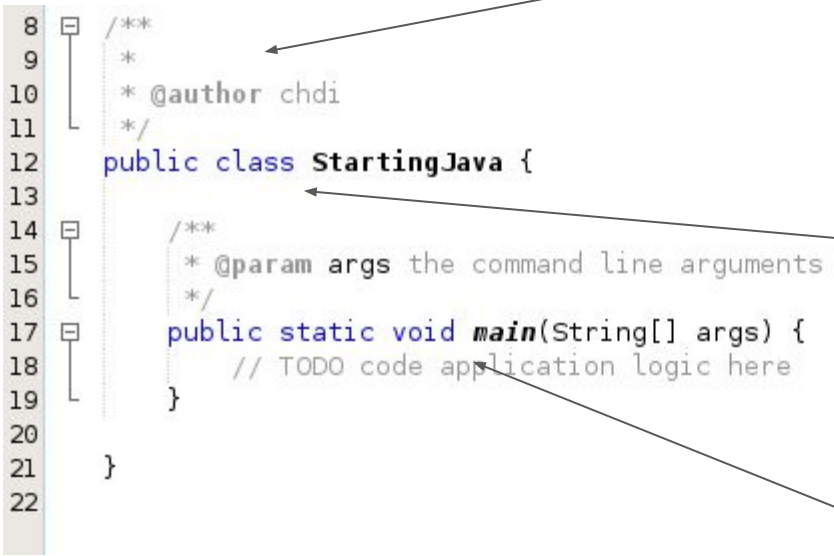
https://www.eclipse.org/downloads

https://www.jetbrains.com/idea/download/

https://maven.apache.org/download.cgi

https://www.mkyong.com/maven/how-to-install-maven-in-windows/

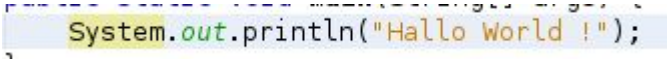
EINRICHTUNG IDE UND MAVEN



Java Kommentar
// Kommentiert eine Zeile
aus
/* Start Kommentarblock
*/ End Kommentarblock

Klassenname
Klassennamen
schreiben wir groß

Startklasse



Standard Ausgabe

Result

```
run:
Hallo World !
BUILD SUCCESSFUL (total time: 0 seconds)
```

Wir haben unsere Erste Java Application geschrieben!

Variablen

- Variablen werden benötigt um Daten während des Programmablaufs zu speichern.
- Um mit Variablen arbeiten zu können, wird für sie ein Name definiert.

```
variablenTyp variablenName;
```

- Der Variablentyp kann ein primitiver Datentyp sein oder eine Klasse.
- Variablen schreiben wir klein!

Wertzuweisung Variablen

Wertzuweisungen an Variablen werden mit dem '='-Operator durchgeführt.

```
variablenName = variablenWert;
```

Der variablenWert muss von dem Typ sein, der in der Variablen Vereinbarung für den variablenName festgelegt wurde. Die Wertzuweisung kann auch in einer Anweisung mit der Vereinbarung der Variablen durchgeführt werden.

```
variablenTyp variablenName = variablenWert;
```

Primitive Datentypen

Ein boolean Wert kann in Java nur zwei verschieden Werte annehmen:

true und false.

Eine Variablenvereinbarung eines boolean Wertes, kann mit folgender Anweisung implementiert werden:

```
boolean wahrOderFalsch;

boolean falschOderWahr = true;
```

Aufgabe - Wir erweitern unsere Ausgabe

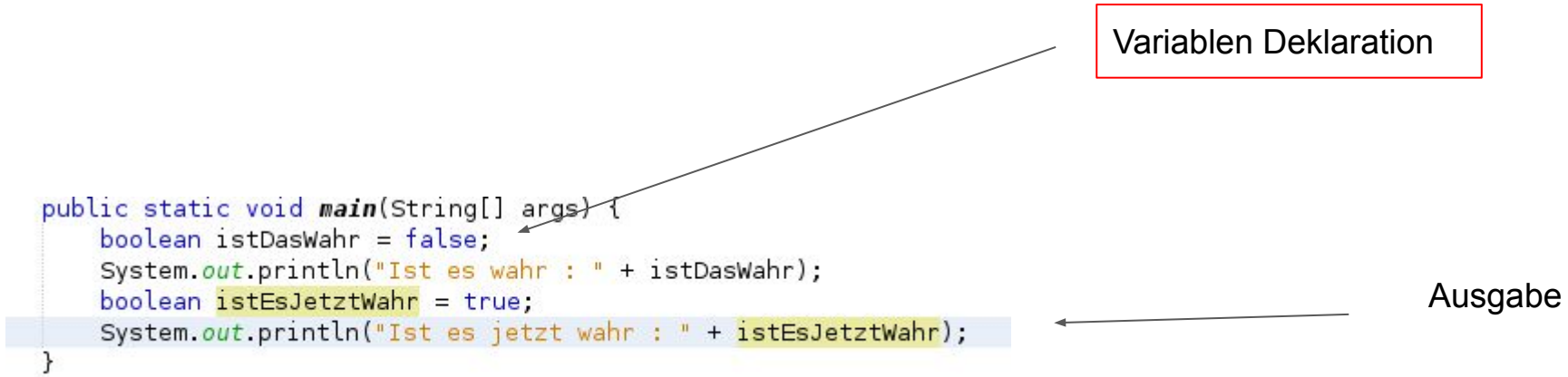
Ausgabe :

```
run:
Ist es wahr : false
Ist es jetzt wahr : true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Lösungshinweis :

In der Standard Ausgabe verketten wir die Ausgabe mit dem Text mit einem +

System.out.println(" ich bin ein Text " + variablenName);


Variablen Deklaration
public static void main(String[] args) {
boolean istDasWahr = false;
System.out.println("Ist es wahr : " + istDasWahr);
boolean istEsJetztWahr = true;
System.out.println("Ist es jetzt wahr : " + istEsJetztWahr);
}
Ausgabe

Aufgabe

Erstellen sie eine Programm welches Folgende Konsolenausgabe darstellt:

```
run:
Ich bin habe 8 Äpfel und ich habe 4.5 Birnen
Es ist so true das ich die Datentypen verstanden habe
BUILD SUCCESSFUL (total time: 0 seconds)
```

Daten Typen Übersicht

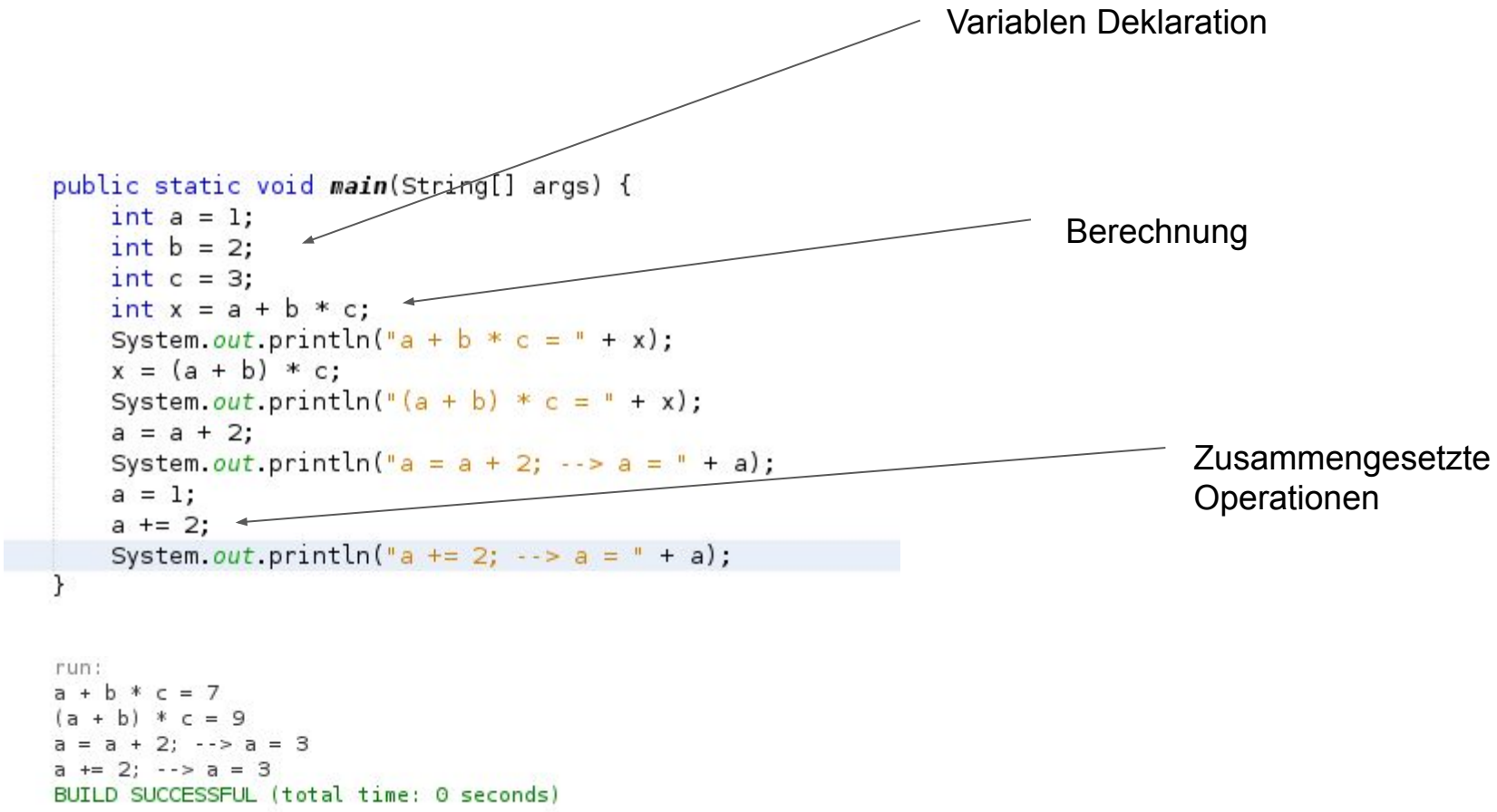
Typname	Größe	Wrapper-Klasse	Wertebereich	Beschreibung
boolean	undefiniert	java.lang.Boolean	true / false	Boolescher Wahrheitswert, Boolescher Typ
char	16 bit	java.lang.Character	0 ... 65.535 (z. B. 'A')	Unicode-Zeichen (UTF-16)
byte	8 bit	java.lang.Byte	-128 ... 127	Zweierkomplement-Wert
short	16 bit	java.lang.Short	-32.768 ... 32.767	Zweierkomplement-Wert
int	32 bit	java.lang.Integer	-2.147.483.648 ... 2.147.483.647	Zweierkomplement-Wert
long	64 bit	java.lang.Long	-2^{63} bis $2^{63}-1$, ab Java 8 auch 0 bis $2^{64}-1$	Zweierkomplement-Wert
float	32 bit	java.lang.Float	+/-1,4E-45 ... +/-3,4E+38	32-bit IEEE 754, es wird empfohlen, diesen Wert nicht für Programme zu verwenden, die sehr genau rechnen müssen.
double	64 bit	java.lang.Double	+/-4,9E-324 ... +/-1,7E+308	64-bit IEEE 754, doppelte Genauigkeit

Operatoren

Die Operatoren +, -, *, /, % führen arithmetische Operationen durch.

Dabei gilt: Multiplikation und Division wird vor Addition oder Subtraktion ausgeführt. Die Operationen sind dabei immer rechtsseitige Argumente.

richtig: c = a + b;
falsch: a + b = c;


Variablen Deklaration
Berechnung
Zusammengesetzte Operationen
public static void main(String[] args) {
int a = 1;
int b = 2;
int c = 3;
int x = a + b * c;
System.out.println("a + b * c = " + x);
x = (a + b) * c;
System.out.println("(a + b) * c = " + x);
a = a + 2;
System.out.println("a = a + 2; --> a = " + a);
a = 1;
a += 2;
System.out.println("a += 2; --> a = " + a);
}
run:
a + b * c = 7
(a + b) * c = 9
a = a + 2; --> a = 3
a += 2; --> a = 3
BUILD SUCCESSFUL (total time: 0 seconds)

Operatoren

Operator	Beschreibung	Kurzbeispiel
+	Addition	`int antwort = 40 + 2;`
-	Subtraktion	`int antwort = 48 - 6;`
*	Multiplikation	`int antwort = 2 * 21;`
/	Division,	`int antwort = 84 / 2;`
%	Teilerrest, Modulo-Operation, errechnet den Rest einer Division	`int antwort = 99 % 57;`
+	positives Vorzeichen, in der Regel überflüssig	`int j = +3;`
-	negatives Vorzeichen	`int minusJ = -j;`

Operator	Beschreibung	Kurzbeispiel
==	gleich	`3 == 3`
!=	ungleich	`4 != 3`
>	größer als	`4 > 3`
<	kleiner als	`-4 < -3`
>=	größer als oder gleich	`3 >= 3`
<=	kleiner als oder gleich	`-4 <= 4`

Quelle: Wikipedia

Simpler Vergleich false

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

        int a = 3;

        int b = 4;

        System.err.println(a == b);

    }

}
```

```
run:
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Simpler Vergleich true

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

    int a = 3;

    int b = 3;

    System.err.println(a == b);

}
```

```
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bedingungen

Die *if*-Anweisung führt eine Anweisung oder einen Anweisungsblock in Abhängigkeit von einer logischen Bedingung aus. Diese Bedingung muss den Ergebnistyp *boolean* besitzen.

```
if(Bedingung1) {

Ich tue etwas

}
```

Die Bedingung kann auch einen optionalen *else*-Zweig besitzen, der ausgeführt wird wenn die Bedingung das Ergebnis *false* hat.

```
if(Bedingung1) {

Ich tue etwas

}  else {

Ich tue auch dann etwas

}
```

Beispielaufgabe :

```
public static void main(String[] args) {
    double myDouble = Math.random();
    System.out.println("Der berechnete Wert ist " + myDouble);
    if (myDouble < 0.5) {
        System.out.println("Der Wert ist kleiner als 0.5");
    } else {
        System.out.println("Der Wert ist grösser oder gleich 0.5");
    }
}
```

```
run:
Der berechnete Wert ist 0.9351743212702978
Der Wert ist grösser oder gleich 0.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Aufgabe :

Schreibe ein Programm, welches sich eine Zufallszahl zwischen 10 und 30 berechnet. Zeige die berechnete Zahl an. Gib auf der Console aus, ob der berechnete Wert größer, gleich oder kleiner 20 ist.

Die Zufallszahl Berechnen sie wie Folgt :

```
double number = 10 + (Math.random() * 20);
```

Lösung

```
public static void main(String[] args) {
    double number = 10 + (Math.random() * 20);
    System.out.println("Es wurde die Zahl " + number + " berechnet.");
    if (number > 20) {
        System.out.println("Die berechnete Zahl ist grösser als zwanzig.");
    } else if (number == 20) {
        System.out.println("Die berechnete Zahl ist gleich zwanzig.");
    } else if (number < 20) {
        System.out.println("Die berechnete Zahl ist kleiner als zwanzig.");
    }
}
```

Switch-Case

Die *switch*-Anweisung dient dazu, verschiedene Anweisungsblöcke in Abhängigkeit vom Wert eines Ausdrucks auszuführen. Der Ausdruck wird Selektor genannt.

```
switch(selektor) {
   ...
   ...
}
```

der Selektor kann folgende Typen beinhalten char, byte, short, int. Seit Java 7 kann im Selektor auch ein String angeben werden

```
int selektor = intWert;
 switch(selektor) {
  case constant1:
   Anweisung11;
   ...
   Anweisung1m;
  case constant2:
   Anweisung21;
   ...
   Anweisung2m;
  case constant3:
   Anweisung31;
   ...
   Anweisung3l;
  ...
  ...
  ...
  case constantn:
   Anweisungn1;
   ...
   Anweisungnk;
 }
```

```
int x = 1;
switch (x) {
    case 0:
        System.out.println("x = 0");
    case 1:
        System.out.println("x = 1");
    case 2:
        System.out.println("x = 2");
}
```

Result :

x = 1

x = 2

Switch Case Default

```
int x = 1;
switch (x) {
    case 0:
        System.out.println("x = 0");
        break;
    case 1:
        System.out.println("x = 1");
        break;
    case 2:
        System.out.println("x = 2");
        break;
    default:
        System.out.println("x ist weder 0 noch 1 noch 2");
}
```

```
run:
x = 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abbruchs Befehl

Default Ausgang

Aufgabe :

Schreibe ein Programm welches eine ganzzahlige Zufallszahl zwischen 1 und 6 berechnet. Wird eine 3 gewürfelt, so soll der Text "Gewonnen, du hast eine 3 gewürfelt." ausgegeben werden. Bei 4, 5 und 6 wird der Text "Du hast eine [gewürfelte Augen] gewürfelt und darfst es noch einmal versuchen." ausgegeben. Bei allen anderen Zahlen soll der Text "Leider verloren, du hast eine [gewürfelte Augen] gewürfelt." ausgegeben werden.

Die Zufallszahl Erstellst du mit Folgendem Code :

```
int number = 1 + (int)(Math.random() * 6);
```

Lösung

```
public static void main(String[] args) {
    int number = 1 + (int) (Math.random() * 6);
    switch (number) {
        case 3:
            System.out.println("Gewonnen, du hast eine 3 gewürfelt.");
            break;
        case 4:
        case 5:
        case 6:
            System.out.println("Du hast eine " + number + " gewürfelt und darfst es noch einmal versuchen.");
            break;
        default:
            System.out.println("Leider verloren, du hast eine " + number + " gewürfelt.");
    }
}
```

Schleifen

Auf Schleifen greift man in der Programmierung dann zurück, wenn eine bestimmte Anweisungen oder Operationen beliebig oft wiederholt werden sollen. In Java gibt es drei relevante Schleifen-Typen. Je nach Einsatzszenario entscheidet man sich für eine der drei Schleifen:

1. For-Schleife
2. While-Schleife
3. Do-While-Schleife

For Schleife

Die For-Schleife nimmt man immer dann, wenn man die Anzahl der benötigten Schleifen-Durchläufe schon im Voraus kennt. Für die For-Schleife werden die folgenden drei Parameter benötigt:

1. Initialisierung
2. Zielwert
3. Schrittweite

```
for(Initialisierung; Zielwert; Schrittweite) {
        // Anweisungen
}
```

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

    for (int i = 0; i < 10; i++) {
        System.out.println("Hallo World " + i);
    }

}
```

```
run:
Hallo World 0
Hallo World 1
Hallo World 2
Hallo World 3
Hallo World 4
Hallo World 5
Hallo World 6
Hallo World 7
Hallo World 8
Hallo World 9
BUILD SUCCESSFUL (total time: 0 seconds)
```

While Schleife

Die While-Schleife wird so lange durchlaufen, bis die Bedingung ein False ergibt. Dabei steht die Bedingung am Anfang, ist sie also schon davor „falsch“, dann wird die Schleife kein einziges Mal ausgeführt, sondern übersprungen.

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

```java
public static void main(String[] args) {
    boolean var = true;
    while (var) {
        var = false;
        System.out.println("Hello World");
    }
}
```

```
run:
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Aufgabe Erweitere die Schleife

```java
public static void main(String[] args) {
    boolean var = true;
    while (var) {
        var = false;
        System.out.println("Hello World");
    }
}
```

Erweitern sie das Beispiel so dass die schleife 5 mal durchläuft und dann Abbricht.

Erinnern sie sich an die Bedingungen

Lösung

```java
public static void main(String[] args) {
    int i = 0;
    while (i < 5) {
        System.out.println("Hello World");
        i++;
    }
}
```

Do-While-Schleife

Wenn man weiß, dass die Schleife auf jeden Fall mindestens **einmal** ausgeführt werden soll, dann kann man statt einer While-Schleife auch eine Do-While-Schleife einsetzen. Eine **Do-While-Schleife** ist im Grunde genommen nichts anderes, als eine While-Schleife bei der die Schleifen-Bedingung am Ende der Schleife und nicht am Anfang steht.

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

Übungsaufgabe Schleifen

Erstelle ein Programm, welches 6000 mal eine Zufallszahl zwischen 1 und 6 berechnet. Verwende hierzu eine for-Schleife. Gebe nach Durchlaufen der Schleife die Anzahl der gewürfelten 6er aus.

Die Zufallszahl zum Würfeln erstellt ihr wie folgt :

```
int number = 1 + (int) (Math.random() * 6);
```

Lösung

```
public static void main(String[] args) {
    int counter = 0;
    for (int i = 0; i < 6000; i++) {
        int number = 1 + (int) (Math.random() * 6);
        if (number == 6) {
            counter++;
        }
    }
    System.out.println("Es wurde " + counter + " mal die 6 gewürfelt.");
}
```

```
run:
Es wurde 978 mal die 6 gewürfelt.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Aufgabe While Schleife

Berechne in einer while-Schleife solange eine ganzzahlige Zufallszahl zwischen 1 und 6, bis eine 6 gewürfelt wurde. Gebe auf der Konsole aus, wieviele Versuche hierzu benötigt wurden.

Der Code zum würfeln

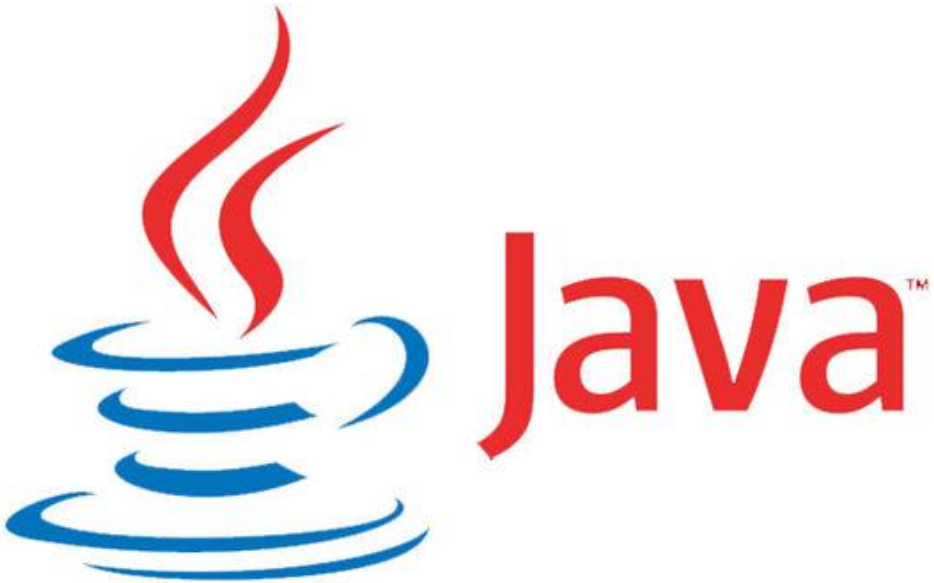
```
number = 1 + (int) (Math.random() * 6);
```

Lösung

```
public static void main(String[] args) {
    int number = 1 + (int) (Math.random() * 6);
    int counter = 1;
    while (number != 6) {
        counter++;
        number = 1 + (int) (Math.random() * 6);
    }
    System.out.println("Es wurde(n) " + counter + " Versuch(e) benötigt");
}
```

```
run:
Es wurde(n) 3 Versuch(e) benötigt
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tag 2



Quelle: Oracle.com

Knobel Aufgabe Schleifen und Abfragen

Erstellen sie mit Hilfe von schleifen und Abfragen Folgende Konsolenausgabe.

Ziel : Konsolenausgabe

Folgende Struktur ist Vorgegeben :

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

    String hash = "#";
    String space = " ";
    int zaehler = 9;

    for (int i = 0; i < 6; i++) {

        for (int j = 0; j < zaehler; j++) {

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

Lösung Knobel Aufgabe Tannenbaum

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

    String hash = "#";
    String space = " ";
    int zaehler = 9;

    for (int i = 0; i < 6; i++) {

        for (int j = 0; j < zaehler; j++) {
          if (j == (zaehler / 2) + i) {
                System.out.print(hash);
            } else if (j == (zaehler / 2) - i) {
                System.out.print(hash);
            } else if (i == 4) {
                System.out.print(hash);
            } else if(i == 5 && j == (zaehler / 2) ) {
                System.out.print(hash);
            }else{
                System.out.print(space);
            }
        }
        System.out.println();
    }
}
```

Arrays

Unter einem Array in Java versteht man ein Feld oder Container, das in der Lage ist, mehrere Objekte vom gleichen Typ aufzunehmen und zu verwalten.

Dabei wird in Java das Array als eine spezielle Klasse repräsentiert, was unter anderem mit sich bringt, dass man auf spezielle Methoden und Operationen bei Arrays zurückgreifen kann.

Der Umgang mit Arrays mag gerade am Anfang etwas schwerer sein und birgt viele Fehlerquellen, nach und nach wird man das System das hinter den Arrays steht aber gut nachvollziehen können.

Deklaration

Typ [ ] Name = new Typ[Anzahl];

Typ Name [ ] = new Typ[Anzahl];

int [ ] meinArray = new int [5];

Gleich Gefüllt :

int [ ] meinArray = {1,2,3,4,5};

Zugriff

```
int[ ] meinArray = {1,2,3,4,5};
System.out.println(meinArray[0])
```

```
int[] meinArray = {1,2,3,4,5};
for(int i=0; i‹meinArray.length; i++) {
        System.out.println(meinArray[i]);
}
```

-> Achtung wenn man auf ein Element außerhalb des Array Zugriff wird eine Exception geschmissen ! (IndexOutOfbound)

Aufgabe Arrays

Erstelle ein Programm mit Folgender Ausgabe :

```
Das Array ist 10 Groß !
An Stelle 0 ist folgender wert : 100
An Stelle 1 ist folgender wert : 200
An Stelle 2 ist folgender wert : 300
An Stelle 3 ist folgender wert : 400
An Stelle 4 ist folgender wert : 500
An Stelle 5 ist folgender wert : 600
An Stelle 6 ist folgender wert : 700
An Stelle 7 ist folgender wert : 800
An Stelle 8 ist folgender wert : 900
An Stelle 9 ist folgender wert : 1000
ende
```

Lösung

```
public static void main(String[] args) {
    int [] meinarray = new int[10];
    meinarray[0] = 100;
    meinarray[1] = 200;
    meinarray[2] = 300;
    meinarray[3] = 400;
    meinarray[4] = 500;
    meinarray[5] = 600;
    meinarray[6] = 700;
    meinarray[7] = 800;
    meinarray[8] = 900;
    meinarray[9] = 1000;

    System.out.println("Das Array ist " + meinarray.length +" Groß !");
    for (int i = 0; i < meinarray.length; i++) {
        System.out.println("An Stelle "+ i + " ist folgender wert : "+ meinarray[i]);
    }
    System.out.println("ende");
}
```

Mehrdimensionales Array

Bei Arrays gibt es die Möglichkeit mehrere Dimensionen anzugeben.

2D Array

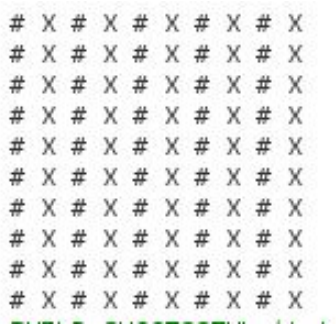
int [ ] [ ] 2darray = new int [10] [10]

oder auch 3D Array

int [ ] [ ] [ ] 3darray = new int [10] [10] [10]

Aufgabe Mehrdimensionales Array

Erstelle die Folgende Ausgabe :



Diese soll Dynamisch erstellt werden.

Hilfe : mit dem Operator % kann geprüft werden ob eine zahl ohne Rest Teilbar ist

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

    int[][] array = new int[10][10];

    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if(j % 2 == 0){
                array[i][j] = 0;
            } else{
                array[i][j] = 1;
            }
        }
    }

    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if(array[i][j] == 0){
                System.out.print("# ");
            }else{
                System.out.print("X ");
            }
        }
        System.out.println("");
    }
}
```

Knobel Aufgabe Array

Implementieren sie einen Bubblesort Algorithmus. Bubblesort ist der Simpelste Sortieralgorithmus. Nacheinander werden in einem Array immer zwei aufeinander folgende Elemente Verglichen und miteinander Vertauscht.

Der Algorithmus ist Vorbei wenn es nichts mehr zu Tauschen gibt.

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

    int[] unsortiert = {1, 5, 8, 2, 7, 4,9,11,45, 24,31,4,7,99,61};
    int[] sortiert = bubblesort(unsortiert);
}

public static int[] bubblesort(int[] zusortieren) {
    return zusortieren;
}
```

```
run:
1, 2, 4, 4, 5, 7, 7, 8, 9, 11, 24, 31, 45, 61, 99, BUILD SUCCESSFUL (total time: 0 seconds)
```

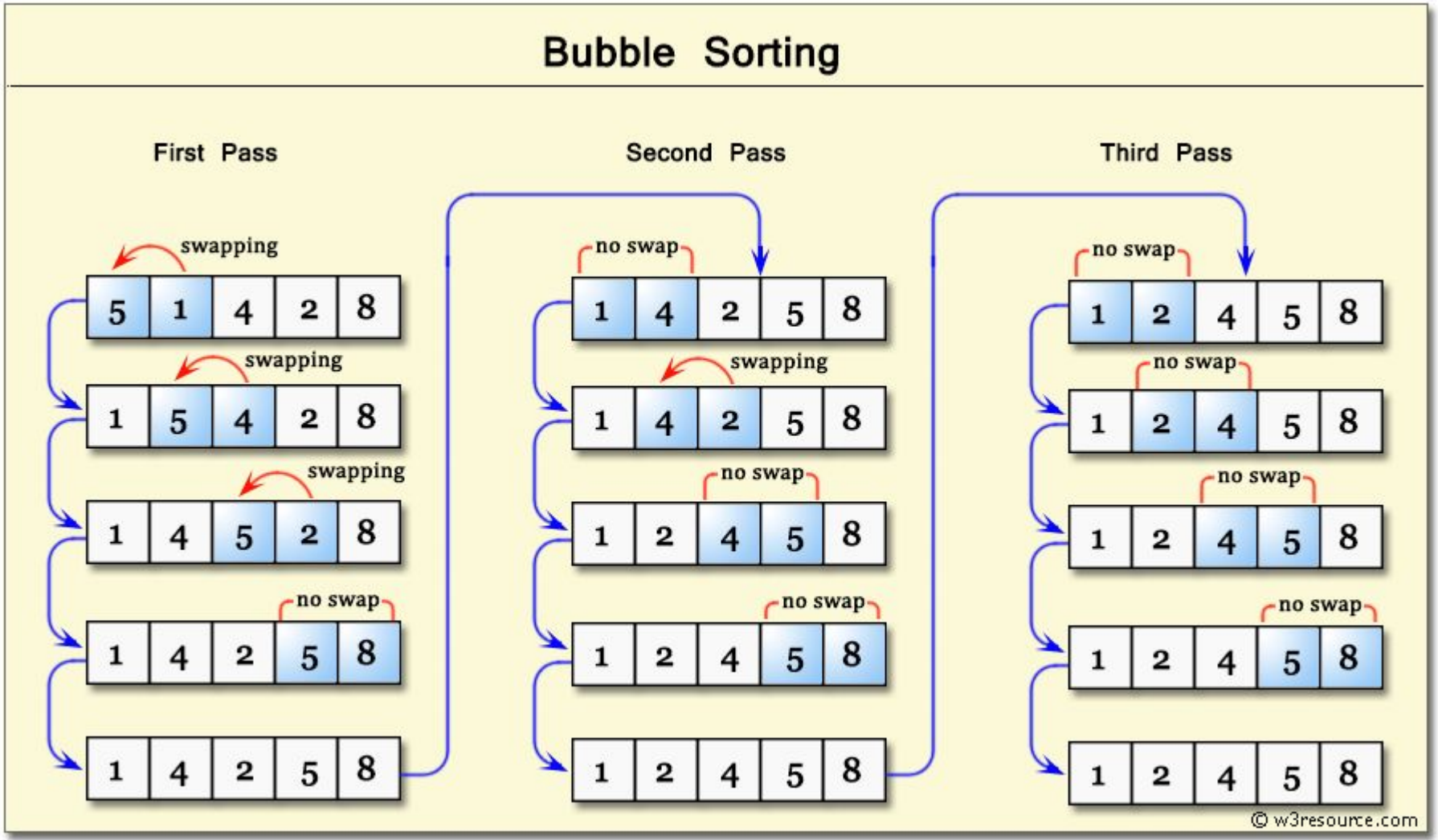
Knobel Aufgabe Array

Implementieren sie einen Bubblesort Algorithmus. Bubblesort ist der Simpelste Sortieralgorithmus. Nacheinander werden in einem Array immer zwei aufeinander folgende Elemente Verglichen und miteinander Vertauscht.

Der Algorithmus ist Vorbei wenn es nichts mehr zu Tauschen gibt.

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

    int[] unsortiert = {1, 5, 8, 2, 7, 4,9,11,45, 24,31,4,7,99,61};
    int[] sortiert = bubblesort(unsortiert);
}

public static int[] bubblesort(int[] zusortieren) {
    return zusortieren;
}
```

```
run:
1, 2, 4, 4, 5, 7, 7, 8, 9, 11, 24, 31, 45, 61, 99, BUILD SUCCESSFUL (total time: 0 seconds)
```

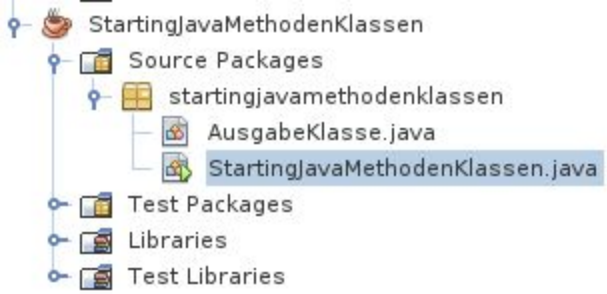

Bubble Sorting
First Pass
Second Pass
Third Pass
swapping
swapping
swapping
no swap
5 1 4 2 8
1 5 4 2 8
1 4 5 2 8
1 4 2 5 8
1 4 2 5 8
no swap
swapping
no swap
no swap
1 4 2 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
no swap
no swap
no swap
no swap
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
© w3resource.com

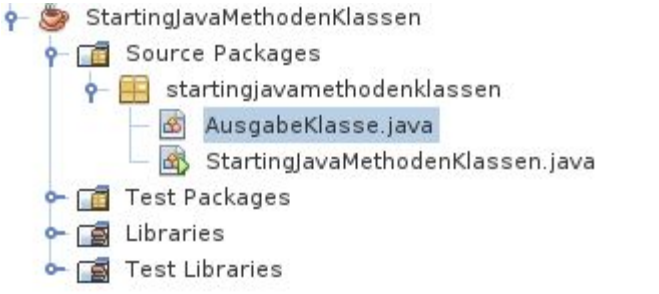
```java
public class Bubblesort {

    public static void main(String[] args) {

        int[] unsortiert = {1, 5, 8, 2, 7, 4,9,11,45, 24,31,4,7,99,61};
        int[] sortiert = bubblesort(unsortiert);

        for (int i = 0; i < sortiert.length; i++) {
            System.out.print(sortiert[i] + ", ");
        }

    }

    public static int[] bubblesort(int[] zusortieren) {
        int temp;
        for (int i = 1; i < zusortieren.length; i++) {
            for (int j = 0; j < zusortieren.length - i; j++) {
                if (zusortieren[j] > zusortieren[j + 1]) {
                    temp = zusortieren[j];
                    zusortieren[j] = zusortieren[j + 1];
                    zusortieren[j + 1] = temp;
                }

            }
        }
        return zusortieren;
    }

}
```

Klassen und Methoden

In **Java** geht nichts ohne **Klassen**. Alle Programme basieren auf **Klasse,** egal wie groß oder klein. In einer **Klasse** sind Methoden und Eigenschaften eines Objektes **definiert**. Die **Klasse** dienen als Templates, also Vorlagen, aus denen beliebig viele Objekte erzeugt werden können.

Definitions Quelle : https://de.wikibooks.org/wiki/Java_Standard





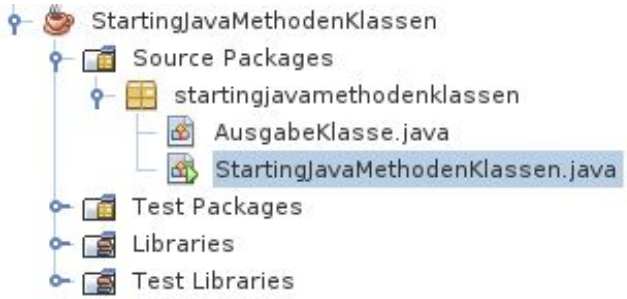

```java
public class StartingJavaMethodenKlassen {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        AusgabeKlasse ausgabeKlasse = new AusgabeKlasse();

        System.out.println(ausgabeKlasse.sayHallo("Peter"));
    }

}
```

```java
public class AusgabeKlasse {

    /**
     * Konstruktor
     */
    public AusgabeKlasse(){
    }
    /**
     *
     * @param name
     * @return String
     */
    public String sayHallo(String name){
        return "Hallo " + name;
    }
}
```

Aufgabe : Erweitern sie die Ausgabe

Erweitern sie die Klasse Ausgabe Klasse mit der Methode sayGoodbye.

Als Parameter soll ein String Übergeben werden.




```java
    public String sayGoodbye (String name){
        return "Goodbye " + name;
    }


public static void main(String[] args) {

    AusgabeKlasse ausgabeKlasse = new AusgabeKlasse();

    System.out.println(ausgabeKlasse.sayHallo("Peter"));

    System.err.println(ausgabeKlasse.sayGoodbye("Peter"));
}
```

```
run:
Hallo Peter
Goodbye Peter
BUILD SUCCESSFUL (total time: 0 seconds)
```

Übungsaufgabe Registrierkasse

- Schleifen
- Klassen
- Methoden
- Datentypen

TODO

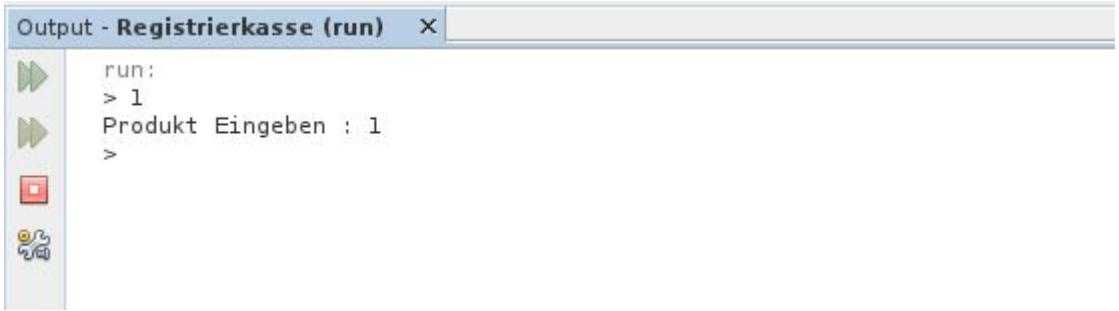
- Erstellen sie die Klasse Produkt mit den zwei Attributen
  - String name
  - double Preis
- Erstellen sie die Main Klasse Registrierkasse
  - Erstellen Sie ein Array von Produkten
  - Erstellen Sie 3 Produkt Objekte und fügen Sie diese dem Array zu.
  - Übergebe die Objekte im Konstruktor der Klasse **Kasse**
- Erstellen die Klasse Kasse
  - Erstellen Sie die Methode Loop die eine While (true) Schleife enthält
  - Die Produkte Sollen über den Konstruktor übergeben werden.

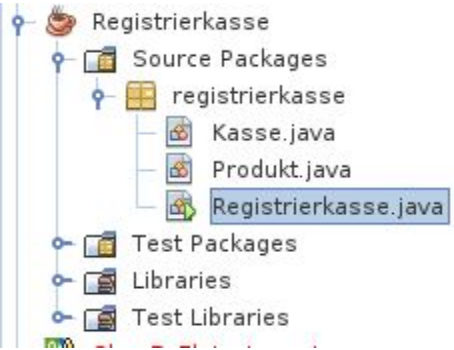

```java
package registrierkasse;

/**
 *
 * @author chdi
 */
public class Produkt {

    String name;

    double preis;


}

```


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

    Produkt produkt1 = new Produkt();
    produkt1.name = "Schokolade";
    produkt1.preis = 1.99;
    Produkt produkt2 = new Produkt();
    produkt2.name = "Cola";
    produkt2.preis = 0.99;
    Produkt produkt3 = new Produkt();
    produkt3.name = "Hundefutter";
    produkt3.preis = 3.99;

    Produkt produkte [] = {produkt1,produkt2,produkt3};

    Kasse kasse = new Kasse(produkte);


        kasse.kassenloop();


}
```

Erweitern sie das Programm

Erweitern sie das Programm das Folgende Ausgabe erscheint. Es soll möglich sein Beliebig viele Produkte einzugeben.





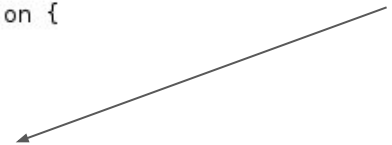

```java
public class Kasse {

    Produkt [] produkts = new Produkt[100];

    public Kasse(Produkt [] produkt) {
            produkts = produkt;
    }


    public void kassenloop() throws IOException {
        boolean isRunnung = true;

        while (isRunnung) {
            wareEingeben();
        }
    }

    public String wareEingeben() throws IOException {
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("> ");
        return input.readLine();
    }


}
```

Konstruktor

Eingabe

Exception

```java
public class Kasse {

    Produkt [] produkts = new Produkt[100];

    public Kasse(Produkt [] produkt) {
            produkts = produkt;
    }


    public void kassenloop() throws IOException {
        boolean isRunnung = true;

        while (isRunnung) {
           String id = wareEingeben();

            System.out.println("Produkt Eingeben : "+ id);
        }
    }

    public String wareEingeben() throws IOException {
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("> ");
        return input.readLine();
    }


}
```




```java
public class Registrierkasse {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)  {

        Produkt produkt1 = new Produkt();
        produkt1.name = "Schokolade";
        produkt1.preis = 1.99;
        Produkt produkt2 = new Produkt();
        produkt2.name = "Cola";
        produkt2.preis = 0.99;
        Produkt produkt3 = new Produkt();
        produkt3.name = "Hundefutter";
        produkt3.preis = 3.99;

        Produkt produkte [] = {produkt1,produkt2,produkt3};

        Kasse kasse = new Kasse(produkte);

        try {
            kasse.kassenloop();
        } catch (IOException ex) {
            System.err.println(ex);
        }

    }

}
```

Fehler fangen und behandeln

Erweitern sie das Programm

```
run:
> 1
Aktuelle Summe : 1.99
> 2
Aktuelle Summe : 2.98
> 3
Aktuelle Summe : 6.970000000000001
> 4
Produkt mit der ID 4 ist nicht vorhanden !
Aktuelle Summe : 6.970000000000001
>
```

- Die Eingabe soll ausgewertet werden
- Es soll die Summe Addiert werden
- Bei einer fehlerhaften ID soll eine Exception Geschmissen werden ! Diese ist zu fangen und zu behandeln

Lösungshinweis : einen String wandelt man wie Folgt in einen int wert . Integer.parseInt(String wert).

```
double summe = 0.0;

public Kasse(Produkt [] produkt) {
        produkts = produkt;
}


public void kassenloop() throws IOException {
    boolean isRunnung = true;

    while (isRunnung) {
       String id = wareEingeben();
        try {
        summe+=getPreis(id);
        } catch (Exception e) {
            System.err.println("Produkt mit der ID "+ id +" ist nicht vorhanden !");
        }


       System.out.println("Aktuelle Summe : "+ summe);
    }
}

public String wareEingeben() throws IOException {
    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("> ");
    return input.readLine();
}

public double getPreis(String id){

    for(int i = 0; i < this.produkts.length; i++) {
        if(this.produkts[i].id == Integer.parseInt(id)){
            return this.produkts[i].preis;
        }
    }
    throw new RuntimeException("ID ist nicht Vorhanden");
}
```

Erweitern sie das Programm

```
run:
> 1
> 2
> z
Aktuelle Summe : 2.98
> 3
> 3
> z
Aktuelle Summe : 10.96
> x
Aktuelle Summe zu Zahlen :10.96
> 12
Betrag : 10.96
Gezahlt : 12.0
--------------------
Rückgeld : 1.0399999999999991
BUILD SUCCESSFUL (total time: 30 seconds)
```

- bei z/Z soll die Aktuelle Summe ausgegeben werden.
- bei x/X soll die eingabe beendet werden und die Zahlung entgegengenommen werden
- Zuschuss soll das Rückgeld berechnet werden

Lösung

```java
public void kassenloop() throws IOException {
    boolean isRunning = true;

    while (isRunning) {

        String id = wareEingeben();

        switch (id.toUpperCase()) {
            case "X":
                isRunning = false;
                break;
            case "Z":
                System.out.println("Aktuelle Summe : " + summe);
                break;
            default:
                try {
                    summe += getPreis(id);
                } catch (Exception e) {
                    System.err.println("Produkt mit der ID " + id + " ist nicht vorhanden !");
                }
            break;
        }
    }

    System.out.println("Aktuelle Summe zu Zahlen :" + summe);
    double zahlung = Double.parseDouble(wareEingeben());
    System.out.println("Betrag : " + summe );
    System.out.println("Gezahlt : " + zahlung);
    System.out.println("--------------------");
    System.out.println("Rückgeld : " + (zahlung - summe));


}
```

```
run:
> 1
> 2
> z
Aktuelle Summe : 2.98
> 3
> 3
> z
Aktuelle Summe : 10.96
> x
Aktuelle Summe zu Zahlen :10.96
> 12
Betrag : 10.96
Gezahlt : 12.0
--------------------
Rückgeld : 1.0399999999999991
BUILD SUCCESSFUL (total time: 30 seconds)
```

In Java Braucht man das Rad nicht neu Erfinden

java.util

Das Paket java.util enthält viele **Hilfsklassen** für den alltäglichen Gebrauch. Von **generischen Datenstrukturen** wie Vector, Stack und Hashtabelle, über **Kalender** und Datum bis hin zum Collections Framework, der **Internationalisierung** und einem Bit-Array bietet das Paket alles, was man sich für die tägliche Arbeit wünscht. Auch ein **Timer** ist vorhanden.

In diesem Abschnitt werden mehrere Themen ausführlicher beschrieben, die für alltägliche Programmieraufgaben von Nutzen sein können. Hier folgt nun ein Überblick.

java.Math

Die Klasse Math ist das Matheobjekt mit allen Operationen für einfache numerische Berechnungen. Neben Konstanten PI und E werden auch viele mathematische Operationen wie Wurzelziehen, Exponentialzahlen, Sinus und Cosinus zur Verfügung gestellt. Alle Konstanten und Methoden in der Math-Klasse sind **static**, damit man kein eigenes Math-Objekt für jede Berechnung anlegen muss. Der Ergebnistyp fast aller Operationen ist **double**.

Nutzt :

https://docs.oracle.com/javase/8/docs/api/

Aufgabe Arbeiten mit der JAVA API

https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html

das IO Package (Input Output)

Erstellen sie einen Kassenbon. Dieser soll als Bon.txt auf ihrem Desktop gespeichert werden.

Lösung

```
                try {
                    summe += getPreis(id);
                    einkauf[count] = getProdukt(id);
                } catch (Exception e) {
                    System.err.println("Produkt mit der ID " + id + " ist nicht vorhanden !");
                }
                break;
            }

            count++;
        }

        System.out.println("Aktuelle Summe zu Zahlen :" + summe);
        double zahlung = Double.parseDouble(wareEingeben());
        System.out.println("Betrag : " + summe);
        System.out.println("Gezahlt : " + zahlung);
        System.out.println("--------------------");
        System.out.println("Rückgeld : " + (zahlung - summe));

        createKassenBon(einkauf, summe, zahlung);
```

```
public void createKassenBon(Produkt einkauf[], double summe, double zahlung) throws IOException {

    File file = new File("/home/chdi/Dokumente/git/test.txt");

    if (!file.exists()) {
        if (file.createNewFile()) {
            System.out.println("KassenBong Neun Erstellt");
        }
    }
    FileWriter fw = new FileWriter(file, true);
    BufferedWriter ausgabe = new BufferedWriter(fw);
    ausgabe.write("Wilkommen im Laden ");
    ausgabe.newLine();
    ausgabe.write("--------------------");
    ausgabe.newLine();
    for (int i = 0; i < einkauf.length; i++) {
        Produkt produkt = einkauf[i];
        if (produkt != null) {
            ausgabe.write(produkt.name + "       " + produkt.preis);
            ausgabe.newLine();
        }
    }
    ausgabe.write("--------------------");
    ausgabe.newLine();
    ausgabe.write("Betrag : " + summe);
    ausgabe.newLine();
    ausgabe.write("Gezahlt : " + zahlung);
    ausgabe.newLine();
    ausgabe.write("--------------------");
    ausgabe.write("Rückgeld :" + (zahlung - summe));

    ausgabe.close();

}
```

Public Private Protected - Kapselung meiner Logic

In der Objektorientierung kennt man so genannte Zugriffsmodifizierer (engl. *access modifier*), die die Rechte anderer Objekte einschränken (Kapselung) oder die ein bestimmtes Verhalten von einem Unterobjekt verlangen (Abstraktion/Vererbung).

	Die Klasse Selbst innere Klassen	Klassen im selben Package	Unterklassen	Sonstige Klassen
private	Ja	Nein	Nein	Nein
(default)	Ja	Ja	Nein	Nein
protected	Ja	Ja	Ja	Nein
public	Ja	Ja	Ja	Ja

Public Private Protected

Der Zugriffsmodifizierer `public` gestattet sämtlichen Klassen Zugriff auf den betreffenden Member. Er ist der freizügigste und zugleich der am häufigsten eingesetzte Zugriffsmodifizierer.

Die Zugreifbarkeit `public` findet man hauptsächlich bei wichtigen Methoden, die von anderen Klassen verwendet werden sollen, bei Konstruktoren und bei Datentypen.

`private` ist der restriktivste Zugriffsmodifizierer. Er verbietet jeglichen Zugriff von außerhalb der Klasse auf den entsprechend modifizierten Member. Auf eine private Variable kann nur die Klasse selbst zugreifen, ebenso auf einen privaten Konstruktor, eine private Methode oder einen privaten geschachtelten Datentyp.

Mit dem Zugriffsmodifizierer `protected` ist der Zugriff nicht nur Klassen aus dem selben Package (wie "default"), sondern auch Subklassen der Klasse erlaubt. Dies gilt auch, wenn die betreffenden Subklassen aus einem anderen Package sind als die Klasse des betreffenden Members.

Der Zugriffsmodifizierer `protected` wird verwendet, wenn es nur für Subklassen Sinn hat, den betreffenden Member zu verwenden.

Diese Zugriffsmodifizierer findet in der API-Programmierung Einsatz. Auch Muster wie das Schablonenmuster (engl. *template pattern*) nutzen diesen Mechanismus.

Qullen : https://de.wikibooks.org/wiki/Java_Standard:_Klassen

Welcher Zugriffsmodifizierer?

Welchen Zugriffsmodifizierer soll man nun verwenden?

Im einfachsten Fall verwendet man `private` für Variablen sowie `public` für Methoden, Konstruktoren und Datentypen.

Eine Regel, die sich in der Praxis sehr bewährt hat, lautet

"so streng wie möglich, so freizügig wie nötig".

Beispiel

```
public Kasse(Produkt[] produkt) {
    produkts = produkt;

    produkts[0].preis = 100.00;

}
```

```
> 1
> x
Aktuelle Summe zu Zahlen :100.0
>
```

```
Produkt produkt1 = new Produkt();
produkt1.name = "Schokolade";
produkt1.preis = 1.99;
produkt1.id = 1;
Produkt produkt2 = new Produkt();
produkt2.name = "Cola";
produkt2.preis = 0.99;
produkt2.id = 2;
Produkt produkt3 = new Produkt();
produkt3.name = "Hundefutter";
produkt3.preis = 3.99;
produkt3.id = 3;
```

```java
public class Produkt {

    private int id;

    private String name;

    private double preis;

}
```



```java
Produkt produkt1 = new Produkt();
produkt1.name = "Schokolade";
produkt1.preis = 1.99;
produkt1.id = 1;
Produkt produkt2 = new Produkt();
produkt2.name = "Cola";
produkt2.preis = 0.99;
produkt2.id = 2;
Produkt produkt3 = new Produkt();
produkt3.name = "Hundefutter";
produkt3.preis = 3.99;
produkt3.id = 3;
```

Getter und Setter Methoden

```java
public class Produkt {

    private int id;

    private String name;

    private double preis;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPreis() {
        return preis;
    }

    public void setPreis(double preis) {
        this.preis = preis;
    }
```

```java
public static void main(String[] args)

    Produkt produkt1 = new Produkt();
    produkt1.setName("Schokolade");
    produkt1.setPreis(1.99);
    produkt1.setId(1);
    Produkt produkt2 = new Produkt();
    produkt2.name = "Cola";
    produkt2.preis = 0.99;
    produkt2.id = 2;
    Produkt produkt3 = new Produkt();
    produkt3.name = "Hundefutter";
    produkt3.preis = 3.99;
    produkt3.id = 3;

    Produkt produkte [] = {produkt1 produ
```

Aufgabe Refactoring

Bauen sie das Programm so um, so dass keine Produkte mehr manipuliert werden können. Überprüfen sie auch ob es Methoden in der “Kasse.java” gibt, die wir auf “private” setzen und so vor Zugriff von außen schützen wollen.

```java
Produkt produkt1 = new Produkt();
produkt1.setName("Schokolade");
produkt1.setPreis(1.99);
produkt1.setId(1);
Produkt produkt2 = new Produkt();
produkt2.setName("Cola");
produkt2.setPreis(0.99);
produkt2.setId(2);
Produkt produkt3 = new Produkt();
produkt3.setName("Hundefutter");
produkt3.setPreis(3.99);
produkt3.setId(3);
```

```java
public void kassenloop() throws IOException {
    boolean isRunning = true;
    Produkt[] einkauf = new Produkt[10];
    int count = 0;
    while (isRunning) {

        String id = wareEingeben();

        switch (id.toUpperCase()) {
            case "X":
                isRunning = false;
                break;
            case "Z":
                System.out.println("Aktuelle Summe : " + summe);
                break;
            default:
                try {
                    summe += getPreis(id);
                    einkauf[count] = getProdukt(id);
                } catch (Exception e) {
                    System.err.println("Produkt mit der ID " + id + " ist nicht vorhanden !");
                }
                break;
        }

        count++;
    }

    System.out.println("Aktuelle Summe zu Zahlen :" + summe);
    double zahlung = Double.parseDouble(wareEingeben());
    System.out.println("Betrag : " + summe);
    System.out.println("Gezahlt : " + zahlung);
    System.out.println("--------------------");
    System.out.println("Rückgeld : " + (zahlung - summe));

    createKassenBon(einkauf, summe, zahlung);
}

private String wareEingeben() throws IOException {...5 lines }

private double getPreis(String id) {...9 lines }

private Produkt getProdukt(String id) {...9 lines }

private void createKassenBon(Produkt einkauf[], double summe, double zahlung) throws IOException {...34 lines }
```

Arbeiten mit Listen

Es gibt neben arrays auch die Sogenannte Verkettete liste. Diese unterscheidet sich in einigen Punkten. So entscheidet man während dem Entwickeln einer Software was man einsetzten möchte.

Listen
+ Dynamisch (kann wachsen)
+ Viele Methoden
- höhere Speicherverbrauch
- Performance

Array
+ genauerer Speicherverbrauch
+ Zugreifen auf einzelne elemente
- Nicht Dynamisch
- geringe “Fertige” Methoden”

In der Heutigen zeit spielt aber die Performance oftmals keine Rolle mehr. So werden aus meiner erfahrung fast nur noch Listen Verwendet bei Komplexeren Aufgaben.

Nutzen sie Java Doc um Folgende Fragen zu Beantworten

https://docs.oracle.com/javase/8/docs/api/java/util/List.html

- Wie erhalte ich die Größe einer Liste ? Nennen Sie die Methode
- Ist es Möglich ein Array komplett in eine Liste einzufügen? Zeigen sie den Code
- Wie durchläuft man eine Liste ? zeigen Sie ein Codebeispiel
- Wie füge ich ein Element hinzu? zeigen Sie ein Codebeispiel
- Wie würde eine ArrayListe für unser Kassen Beispiel aussehen?

- Wie erhalte ich die Größe einer Liste ? Nennen sie die Methode

  Liste <foo> bar = new ArrayList<foo>; bar.size();

- Ist es Möglich ein Array Komplett in eine Liste einzufügen ? Zeigen sie den code

  List<Produkt> produkteasList = new ArrayList<>(Arrays.asList(produkte));

- Wie durchläuft man eine Liste ? zeigen sie ein Codebeispiel

```
List<Produkt> produkts = new ArrayList<Produkt>();

        for (Produkt produkt : produkts) {

         produkt..

      }

      for (int i = 0; i < produkts.size(); i++) {

         Produkt get = produkts.get(i);

       }
```

- Wie füge ich ein Element Hinzu ? zeigen sie ein Codebeispiel

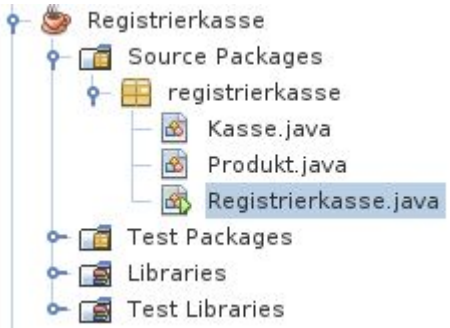
  Liste <foo> bar = new ArrayList<foo>; bar.add(new Foo());

- Wie würde eine ArrayListe für unser Kassen Beispiel aussehen ?

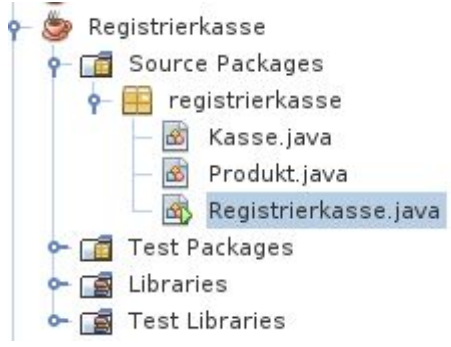
  List<Produkt> produkts = new ArrayList<Produkt>();

Aufgabe Refactoring

Bauen sie unsere Beispiel Application so um dass wir statt Arrays eine Liste verwenden.

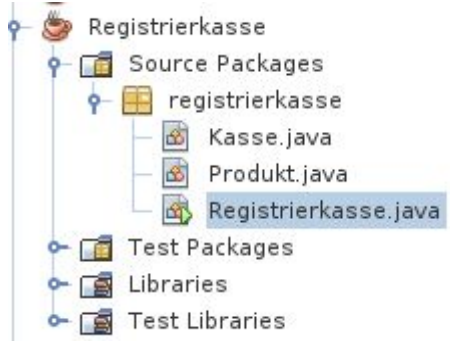



```
public static void main(String[] args)  {

    Produkt produkt1 = new Produkt();
    produkt1.setName("Schokolade");
    produkt1.setPreis(1.99);
    produkt1.setId(1);
    Produkt produkt2 = new Produkt();
    produkt2.setName("Cola");
    produkt2.setPreis(0.99);
    produkt2.setId(2);
    Produkt produkt3 = new Produkt();
    produkt3.setName("Hundefutter");
    produkt3.setPreis(3.99);
    produkt3.setId(3);

    Produkt produkte [] = {produkt1,produkt2,produkt3};

    List<Produkt> produkteasList = new ArrayList<>(Arrays.asList(produkte));


    Kasse kasse = new Kasse(produkteasList);

    try {
        kasse.kassenloop();
    } catch (IOException ex) {
        System.err.println(ex);
    }

}
```




```java
List<Produkt> produkts = new ArrayList<Produkt>();

double summe = 0.0;

public Kasse(List<Produkt> produkts) {
    this.produkts = produkts;

}
```

```java
public void kassenloop() throws IOException {
    boolean isRunning = true;
    List<Produkt> einkauf = new ArrayList<Produkt>();
    while (isRunning) {

        String id = wareEingeben();

        switch (id.toUpperCase()) {
            case "X":
                isRunning = false;
                break;
            case "Z":
                System.out.println("Aktuelle Summe : " + summe);
                break;
            default:
                try {
                    summe += getPreis(id);
                    einkauf.add(getProdukt(id));
                } catch (Exception e) {
                    System.err.println("Produkt mit der ID " + id + " ist nicht vorhanden !
                }
                break;
        }

    }

    System.out.println("Aktuelle Summe zu Zahlen :" + summe);
    double zahlung = Double.parseDouble(wareEingeben());
    System.out.println("Betrag : " + summe);
    System.out.println("Gezahlt : " + zahlung);
    System.out.println("--------------------");
    System.out.println("Rückgeld : " + (zahlung - summe));

    createKassenBon(einkauf, summe, zahlung);
}
```




```java
private double getPreis(String id) {

    for (Produkt produkt : produkts) {
        if (produkt.getId() == Integer.parseInt(id)) {
            return produkt.getPreis();
        }
    }
    throw new RuntimeException("ID ist nicht Vorhanden");
}

private Produkt getProdukt(String id) {

    for (Produkt produkt : produkts) {
        if (produkt.getId() == Integer.parseInt(id)) {
            return produkt;
        }
    }
    throw new RuntimeException("ID ist nicht Vorhanden");
}

private void createKassenBon(List<Produkt> einkauf, double summe, double zahlung) throws IOException {

    File file = new File("/home/chdi/Dokumente/git/test.txt");

    if (!file.exists()) {
        if (file.createNewFile()) {
            System.out.println("KassenBong Neun Erstellt");
        }
    }
    FileWriter fw = new FileWriter(file, true);
    BufferedWriter ausgabe = new BufferedWriter(fw);
    ausgabe.write("Wilkommen im Laden ");
    ausgabe.newLine();
    ausgabe.write("--------------------");
    ausgabe.newLine();

    for (Produkt pro : einkauf) {
        ausgabe.write(pro.getName() + "       " + pro.getPreis());
        ausgabe.newLine();
    }
```

JDBC

JDBC - ist die “Java Database Connectivity”

Es ermöglicht uns mit einer Datenbank zu interagieren.

Wir benötigen hierfür eine sogenannte Treiber Klasse

https://dev.mysql.com/downloads/file/?id=13597


Registrierkasse
Source Packages
registrierkasse
Kasse.java
Produkt.java
Registrierkasse.java
Test Packages
<default package>
Libraries
Add Project...
Add Library...
Add JAR/Folder...
Properties
ShopBy
Starting
StartingJavaMethodenKlassen
Registrierkasse
Source Packages
registrierkasse
Kasse.java
Produkt.java
Registrierkasse.java
Test Packages
<default package>
Libraries
MySQL JDBC Driver - mysql-connector-java-5.1.23-bin.jar
JDK 1.8 (Default)
Test Libraries

Datenbank Vorbereiten

https://www.apachefriends.org/de/index.html

mysql> create database kasse

mysql> CREATE TABLE Produkt (id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,produktid VARCHAR(30) NOT NULL,name VARCHAR(30) NOT NULL,preis VARCHAR(30) NOT NULL)

mysql> insert into Produkt (produktid,name,preis) values("4","Kekse","0.99") ;


```java
package registrierkasse.db;

import java.sql.*;



/**
 *
 * @author chdi
 */
public class DatabaseConnection {

    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/kasse";

    //  Database credentials
    static final String USER = "root";
    static final String PASS = "root";

    public void connection() {...62 lines }

}
```

```
public void connection() {

    Connection conn = null;
    Statement stmt = null;

    try {
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);

        //STEP 4: Execute a query
        System.out.println("Creating statement...");
        stmt = conn.createStatement();
        String sql;
        sql = "SELECT produktid,name, preis FROM Produkt";
        ResultSet rs = stmt.executeQuery(sql);

        //STEP 5: Extract data from result set
        while (rs.next()) {
            //Retrieve by column name
            String produktid = rs.getString("produktid");
            String name = rs.getString("name");
            String preis = rs.getString("preis");

            //Display values
            System.out.print(", Produktid : " + produktid);
            System.out.print(", Name: " + name);
            System.out.println(", Preis: " + preis);
        }
        //STEP 6: Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
    } catch (SQLException se) {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch (Exception e) {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        //finally block used to close resources
        try {
            if (stmt != null) {
                stmt.close();
            }
        } catch (SQLException se2) {
        } finally {
            //finally block used to close resources
            try {
                if (stmt != null) {
                    stmt.close();
                }
            } catch (SQLException se2) {
            }// nothing we can do
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException se) {
                se.printStackTrace();
            }//end finally try
        }//end try

        System.out.println("Goodbye!");

    }
```

```
run:
Connecting to database...
Creating statement...
, Produktid : 4, Name: Kekse, Preis: 0.99
Goodbye!
>
```

Erstellen eines Eintrags

```
public void createMysqlValue(String produktId,String name,String preis) throws SQLException{
  Statement stmt = null;
  Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
  stmt = conn.createStatement();
  String sql = "insert into Produkt (produktid,name,preis) values(\""+produktId+"\",\""+name+"\",\""+preis+"\")" ;

  stmt.executeUpdate(sql);

}
```

Refactor

Führen Sie ein Refactoring der Klasse DatabaseConnection.java durch.

Im Konstruktor soll die DatabaseConnection Hergestellt werden.

Erstellen Sie eine Methode die die DB Connection zurückgibt.

Erstellen Sie eine separate Eingabemethode

Erstellen Sie eine separate Ausgabemethode.

```
    private Connection conn;

    public DatabaseConnection() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
        } catch (Exception e) {
        }

    }

    public void readAllValues() throws SQLException {
        Statement stmt = conn.createStatement();
        String sql;
        sql = "SELECT produktid,name, preis FROM Produkt";
        ResultSet rs = stmt.executeQuery(sql);

        //STEP 5: Extract data from result set
        while (rs.next()) {
            //Retrieve by column name
            String produktid = rs.getString("produktid");
            String name = rs.getString("name");
            String preis = rs.getString("preis");

            //Display values
            System.out.print(", Produktid : " + produktid);
            System.out.print(", Name: " + name);
            System.out.println(", Preis: " + preis);
        }
        //STEP 6: Clean-up environment
        rs.close();
        stmt.close();
    }

    public void createMysqlValue(String produktId, String name, String preis) throws SQLException {
        Statement stmt = null;
        Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
        stmt = conn.createStatement();
        String sql = "insert into Produkt (produktid,name,preis) values(\"" + produktId + "\",\"" + name + "\",\"" + preis + "\")";

        stmt.executeUpdate(sql);

    }
```

Aufgabe: Erweitern sie das Programm

Ändern sie die Kassen Routine. Die Produkte die eingekauft werden können, sollen nun aus der Datenbank gelesen werden. Und nicht mehr aus einer Liste.

```java
public static void main(String[] args) throws SQLException  {


    DatabaseConnection databaseConnection = new DatabaseConnection();
    List<Produkt> prod = databaseConnection.readAllValues();
    databaseConnection.closeConnection();


    Kasse kasse = new Kasse(prod);

    try {
        kasse.kassenloop();
    } catch (IOException ex) {
        System.err.println(ex);
    }

}
```

```java
public List<Produkt> readAllValues() throws SQLException {
    Statement stmt = conn.createStatement();
    String sql;
    sql = "SELECT produktid,name, preis FROM Produkt";
    ResultSet rs = stmt.executeQuery(sql);

    List <Produkt> result = new ArrayList<>();

    //STEP 5: Extract data from result set
    while (rs.next()) {
        //Retrieve by column name
        Produkt tmp = new Produkt();
        tmp.setId(Integer.parseInt(rs.getString("produktid")));
        tmp.setName(rs.getString("name"));
        tmp.setPreis(Double.parseDouble(rs.getString("preis")));

        result.add(tmp);
    }
    //STEP 6: Clean-up environment
    rs.close();
    stmt.close();

    return result;
}
```

Aufgabe Erweitern sie das Programm

bei der Eingabe der Kasse soll es möglich sein mittels der eingabe Admin eine Sub Routine zu öffnen. Diese soll die eingabe der Produkte in die Datenbank ermöglichen. Das neu Eingegebene Produkt soll direkt in der Kasse Verfügbar sein.

```
> admin
Connecting to database...
Willkommen im Admin Berreich
Bei 1 geben sie ein neues Produkt ein
Bei 2 reinizialisieren sie die Produkte
Bei 3 Beenden sie den Admin Berreich
> 1
ID eingeben
> 6
> Alwa wasser
> 3.99
Connecting to database...
Willkommen im Admin Berreich
Bei 1 geben sie ein neues Produkt ein
Bei 2 reinizialisieren sie die Produkte
Bei 3 Beenden sie den Admin Berreich
> 2
Connecting to database...
Willkommen im Admin Berreich
Bei 1 geben sie ein neues Produkt ein
Bei 2 reinizialisieren sie die Produkte
Bei 3 Beenden sie den Admin Berreich
> 3
> 6
> x
Aktuelle Summe zu Zahlen :3.99
```

Lösung

```
case "ADMIN":
    boolean adminisRunning = true;
    while (adminisRunning) {

        DatabaseConnection db = new DatabaseConnection();
        System.out.println("Willkommen im Admin Berreich");
        System.out.println("Bei 1 geben sie ein neues Produkt ein ");
        System.out.println("Bei 2 reinizialisieren sie die Produkte");
        System.out.println("Bei 3 Beenden sie den Admin Berreich");

        int idadmin = Integer.parseInt(wareEingeben());

        switch (idadmin) {
            case 1:
                System.out.println("ID eingeben");
                String idEingabe = wareEingeben();
                System.out.println("Name eingeben");
                String name = wareEingeben();
                System.out.println("Preis eingeben");
                String preis = wareEingeben();

                db.createMysqlValue(idEingabe, name, preis);
                break;
            case 2:
                this.produkts = null;
                this.produkts = db.readAllValues();
                break;
            case 3:
                adminisRunning = false;
                db.closeConnection();
                break;
            default:
                System.out.println("Ihre Eingabe ist Falsch");
        }
    }
    break;
```

Team Aufgabe Minesweeper

1. Team - Frontend

2. Spiel logic

3. Highscore Service

Aufgabe:

Erstellen sie ein Spiel Namens MinesWeeper, hierfür bilden sie 3 Teams.

Team Aufgabe Minesweeper

1 Team - Frontend

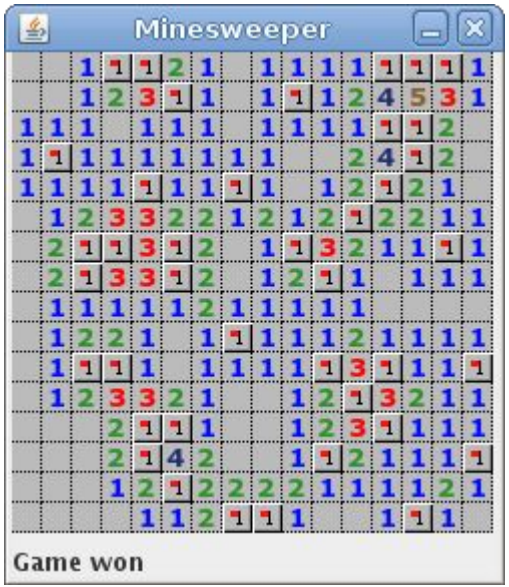
2 Spiel logic

Aufgabe:

Erstellen sie ein Spiel Namens MinesWeeper, hierfür bilden sie 2 Teams.

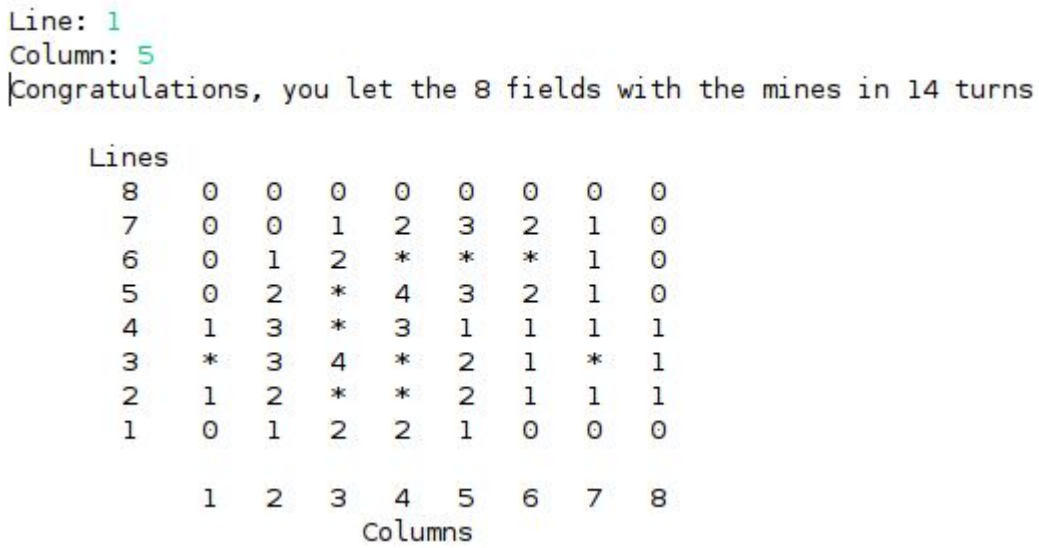
Gui Team

Das Guid Team hat die Aufgabe eine Spieloberfläche in Java Swing zu entwickeln.
Beispiel Bild :



Spiele Logic Team

Das Spiele Logic Team hat sich mit dem Spiel dem Algorithmus und mit der Behandlung der Events bei dem Minesweeper Spiel zu beschäftigen.



Aufgabe Spiele Logic

- Aufbau des Spielfeld - Wo sind Minen
- Aufbau der Nachbarn - Wo sind Zahlen und wo nicht
- Auf klicken - wo nichts ist Öffnet es sich bis zu den Zahlen

Highscore Service

Das Highscore Service Team entwirft einen Microservice & den Dazugehörigen web client. Hier kann sich der Spieler nach einem Spiel Eintragen. Der Service soll die Daten in eine Mysql Datenbank schreiben.

Build Tool Maven

- Maven ist ein Build Tool
- Maven sorgt für das Bauen einer Software
- Maven Übernimmt das Dependency Management
- Maven bietet viele Plugins die Arbeiten im Release und Build Management übernehmen

Maven Lebenscyclus

- Validate - Validiert das Projekt
- compile - Kompiliert das Projekt
- test - führt den Test aus
- package - baut ein JAR / WAR
- verify - lässt alle anderen Checks durchlaufen / und auch Integrations Tests
- install - installiert das package in das lokale Maven Repo
- deploy - kopiert das Paket in ein Remote Repository

Infos

Der Wichtigste Link rund um Maven ist

http://maven.apache.org

Darf jederzeit benutzt werden.

http://maven.apache.org/download.cgi

Maven Plugin Eclipse

http://www.eclipse.org/m2e/

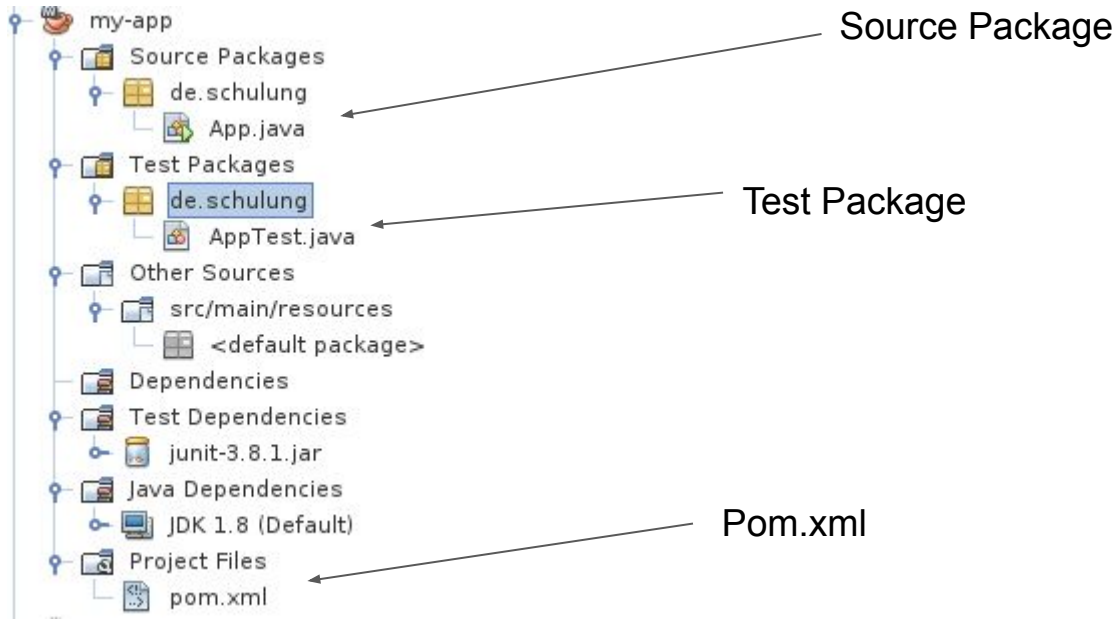
Erstellen eines Maven Projektes

- Projekte aus maven werden aus sogenannten “Archetype” erstellt.

```
mvn archetype:generate -DgroupId=de.schulung -DartifactId=my-app
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Quelle : maven.org

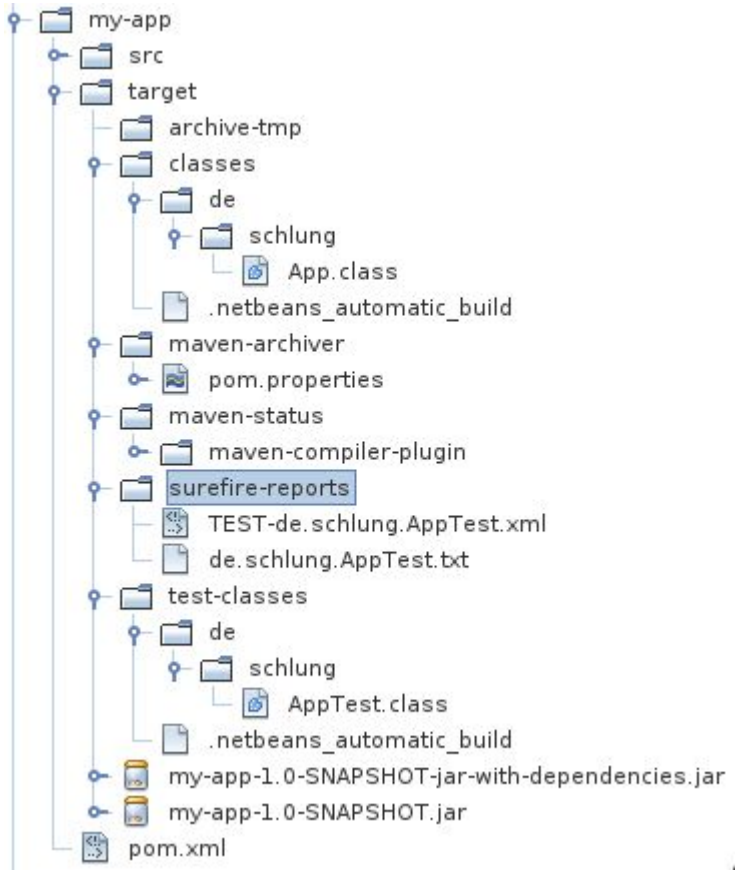
```
chdi@chdi-laptop:~/DevTools/apache-maven-3.3.9/bin$ ./mvn archetype:generate -DgroupId=de.schlung -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.pom (703 B at 10.6 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-bundles/2/maven-archetype-bundles-2.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-bundles/2/maven-archetype-bundles-2.pom (2 KB at 32.9 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/archetype/maven-archetype-parent/1/maven-archetype-parent-1.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/archetype/maven-archetype-parent/1/maven-archetype-parent-1.pom (2 KB at 24.2 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/4/maven-parent-4.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/4/maven-parent-4.pom (10 KB at 238.1 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/apache/3/apache-3.pom
Downloaded: https://repo.maven.apache.org/maven2/org/apache/apache/3/apache-3.pom (4 KB at 50.0 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar (5 KB at 113.4 KB/sec)
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: /home/chdi/DevTools/apache-maven-3.3.9/bin
[INFO] Parameter: package, Value: de.schlung
[INFO] Parameter: groupId, Value: de.schlung
[INFO] Parameter: artifactId, Value: my-app
[INFO] Parameter: packageName, Value: de.schlung
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/chdi/DevTools/apache-maven-3.3.9/bin/my-app
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.899 s
[INFO] Finished at: 2017-09-17T21:12:11+02:00
[INFO] Final Memory: 16M/205M
[INFO] ------------------------------------------------------------------------
chdi@chdi-laptop:~/DevTools/apache-maven-3.3.9/bin$
```


my-app
Source Packages
de.schulung
App.java
Test Packages
de.schulung
AppTest.java
Other Sources
src/main/resources
<default package>
Dependencies
Test Dependencies
junit-3.8.1.jar
Java Dependencies
JDK 1.8 (Default)
Project Files
pom.xml
Source Package
Test Package
Pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.schulung</groupId>
    <artifactId>my-app</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>my-app</name>
    <url>http://maven.apache.org</url>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <archive>
                                <manifest>
                                    <mainClass>
                                        de.schulung.App
                                    </mainClass>
                                </manifest>
                            </archive>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Build an Projekt

> mvn clean package



Play maven

Probieren sie Folgende Commands aus:

> mvn clean

> mvn clean package

> mvn clean install

Das Lokale Maven Repository

```
chdi@chdi-laptop:~/.m2$ ls
repository
chdi@chdi-laptop:~/.m2$ cd repository/
chdi@chdi-laptop:~/.m2/repository$ ls
antlr  asm  backport-util-concurrent  classworlds  com  commons-cli  commons-codec  commons-collections  commons-io  commons-lang  commons-loggi
chdi@chdi-laptop:~/.m2/repository$ ls -la
insgesamt 92
drwxrwxr-x 23 chdi chdi 4096 Sep 17 21:21 .
drwxrwxr-x  3 chdi chdi 4096 Mai 16 15:16 ..
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 antlr
drwxrwxr-x  8 chdi chdi 4096 Sep 17 21:07 asm
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 backport-util-concurrent
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 classworlds
drwxrwxr-x  4 chdi chdi 4096 Sep 17 21:07 com
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 commons-cli
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 commons-codec
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 commons-collections
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 commons-io
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 commons-lang
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 commons-logging
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 de
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 dom4j
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 doxia
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 jdom
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 junit
drwxrwxr-x  2 chdi chdi 4096 Mai 19 17:44 .locks
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:21 log4j
drwxrwxr-x  4 chdi chdi 4096 Sep 17 21:07 net
drwxrwxr-x  8 chdi chdi 4096 Sep 17 21:07 org
drwxrwxr-x  3 chdi chdi 4096 Sep 17 21:07 xml-apis
chdi@chdi-laptop:~/.m2/repository$
```

Aufgabe : Refactoring

Erstellen Sie mit Hilfe des Maven Archetype ein Neues Projekt und ziehen Sie das Kassen Projekt in ein Maven Projekt um.

Dependencies Eintragen

> https://search.maven.org/ hier finden sie Alle Public Dependencys wie Mysql driver

```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.6</version>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

JUNIT

JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (Klassen oder Methoden) geeignet ist. Anfänglich wurde **JUnit** von Erich Gamma und Kent Beck entwickelt. Es basiert auf Konzepten, die ursprünglich unter dem Namen SUnit für Smalltalk entwickelt wurden.

Junit ist gut geeignet für :

- Methoden mit Rückgabewert
- Methoden die Fehler schmeißen können
- Testen komplexer Units

Junit ist ungeeignet für :

- void Methoden (außer das ergebnis ist prüfbar)
- Integrations Tests

Quelle Definition: Wikipedia

Wie sieht eine Testklasse aus ?

```java
import org.junit.*;
import org.junit.rules.ExpectedException;

/**
 *
 * @author chdi
 */
public class KassenTest {

    Kasse classUnderTest;

    @Rule
    public ExpectedException thrown = ExpectedException.none();

    @Before
    public void init() {

        Produkt testProdukt = new Produkt();
        testProdukt.setId(1);
        testProdukt.setName("Hundefutter");
        testProdukt.setPreis(9.99);

        List<Produkt> testProdukts = new ArrayList<Produkt>();
        testProdukts.add(testProdukt);
        classUnderTest = new Kasse(testProdukts);

    }

    @Test
    public void testKonstruktor() {
        Assert.assertEquals(1, classUnderTest.produkts.size());
    }

    @Test
    public void testGetId() {
        Produkt result = classUnderTest.getProdukt("1");

        Assert.assertEquals("Hundefutter", result.getName());
    }

    @Test
    public void testGetId_NotFound() {
        thrown.expect(RuntimeException.class);

        Produkt result = classUnderTest.getProdukt("2");
    }
```

Zu Testende Klasse

Test Regel

“Setup” Methode

Unit Test

Test Erweitern

Erstellen sie für die für die Methode getPreis() einen Positiven Testfall und einen Exception Testfall

Lösung Junit

```
@Test
public void testPreisId() {
    double result = classUnderTest.getPreis("1");

    Assert.assertEquals(9.99, result,0.00);
}

@Test
public void testPreisId_NotFound() {
   thrown.expect(RuntimeException.class);

   double result = classUnderTest.getPreis("3");
}
```

Trainer Christian Gahlert

Wirtschaftsinformatik DHBW Karlsruhe (2010-2013)

Build Engineer bei Fiducia & GAD IT AG (2013-2015)

Lead Software Developer bei Sophos Technology GmbH (2015-2016)

Freelancer: Software Developer bei ISB AG (2016-2017)

Freelancer: Lead Software Developer bei 1&1 Mail & Media (seit 2017)

Kontakt: www.chrisgahlert.com

Agenda: Tag 4 & 5

- Unit-Tests Wiederholung und Einführung in Mockito
- Spring
  - Dependency Injection Grundlagen
  - XML based Configuration
  - Annotation based Configuration
  - Spring Boot und Spring MVC
- Java 8
- Wahl-Themen oder Gruppenprojekt

Aufgabe: Wiederholung Unit-Tests

- Öffnen Sie Beispiel-Projekt *1-dependency-injection*
- Schreiben Sie die Tests für den *CreditCardProcessor* (soweit möglich)

Design Patterns: Singleton

- Methode um jederzeit an die gleiche Instanz einer Klasse zu kommen.

```java
public class PaymentServiceImpl implements PaymentService {
    private static PaymentService singletonInstance = null;

    private List<Transaction> transactions = new ArrayList<>();

    private PaymentServiceImpl() {
    }

    public static PaymentService getInstance() {
        if(singletonInstance == null) {
            singletonInstance = new PaymentServiceImpl();
        }

        return singletonInstance;
    }
```

=>

```java
this.paymentService = PaymentServiceImpl.getInstance();
```

Dependency Injection

- Inversion of Control (IoC)
- Hollywood-Prinzip: “Don’t call us, we call you”
- Erhöhung der Testbarkeit
- Entkoppelung von Klassen (Loose coupeling)

Dependency Injection: Beispiel “Constructor Injection”

Vorher:

```java
public class CreditCardProcessor {

    private final PaymentService paymentService;

    public CreditCardProcessor() {
        // VORHER: Ich erstelle/hole mir die Instanz
        this.paymentService = PaymentServiceImpl.getInstance();
    }
```

Nacher:

```java
public class CreditCardProcessor {

    private final PaymentService paymentService;

    public CreditCardProcessor(PaymentService paymentService) {
        // NACHER: Ich lasse mir die Instanz reinreichen
        this.paymentService = paymentService;
    }
```

Aufgabe: Unit-Tests mit Constructor Injection

- Öffnen Sie Beispiel-Projekt *1-dependency-injection*
- Stellen Sie den *CreditCardProcessor* auf Constructor Injection um
- Erstellen Sie eine Test-Implementierung des *PaymentService*, z.B. *PaymentServiceTest*
- Schreiben Sie die Tests für den *CreditCardProcessor* unter Verwendung der Test-Implementierung

Test-Framework “Mockito”

- Statt eigene Test-Implementierungen zu erstellen ist es häufig komfortabler zu “mocken”
- Mockito ist ein Framework, welches erlaubt innerhalb einer JUnit-Test-Methode Mocking-Anweisungen zu schreiben

```
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class CreditCardProcessorTest {

    @InjectMocks
    private CreditCardProcessor creditCardProcessor;

    @Mock
    private PaymentService paymentService;

    @Test
    public void myTest() {
        // ...
    }
}
```

Mockito: Wenn, dann

```
@Test
public void myTest() {
    when(paymentService.getCurrentBalance( ccNumber: "123")).thenReturn(100);
    when(paymentService.getCurrentBalance( ccNumber: "456")).thenThrow(IllegalArgumentException.class);
    when(paymentService.getCurrentBalance( ccNumber: "456")).thenCallRealMethod();
}
```

Mockito: Assertions

```
@Test
public void myTest() {
    verify(paymentService).getCurrentBalance( ccNumber: "456");
    verify(paymentService, times( wantedNumberOfInvocations: 1)).findTransaction(UUID.fromString("..."));
    verify(paymentService, never()).addTransaction( ccNumber: "456", anyInt(), anyString(), any(Transaction.class));
}
```

Aufgabe: Unit-Test mit Mock

- Öffnen Sie Beispiel-Projekt *1-dependency-injection*
- Fügen Sie die aktuelle Version von Mockito Ihrem Projekt als Maven-Dependency hinzu. (Tipp: “mockito-core” auf www.maven.org)
- Ersetzen Sie die Test-Implementierung durch ein Mockito-Mock und passen Sie die Tests entsprechend an. (Tipp: www.mockito.org)
- *Hinweis: Die Test-Implementierung noch nicht löschen! Diese wird später noch benötigt. Machen Sie ebenfalls eine Sicherung der Test-Klasse.*
- Überlegen Sie, ob noch weitere Tests fehlen…

Spring: Dependency Injection Framework

- Das Management der Dependencies wird ausgelagert
- Als Entwickler kann man sich auf das “was” konzentrieren und das “wie” wird vom Framework übernommen.

Spring: XML-Configuration

```xml
<beans>

    <bean class="com.example.MyService" id="myService" />

    <bean class="com.example.MyOtherService">
        <constructor-arg value="hello" />
        <property name="who" value="world" />
        <property name="myService" ref="myService" />
    </bean>

</beans>
```

```java
public class MyOtherService {

    private String salutation;
    private String who;
    private MyService myService;

    public MyOtherService(String salutation) {
        this.salutation = salutation;
    }

    public void setWho(String who) {
        this.who = who;
    }

    public void setMyService(MyService myService) {
        this.myService = myService;
    }
}
```

Spring: XML-Configuration

```xml
<beans>

    <bean class="com.example.MyService" id="myService" />

    <bean class="com.example.MyOtherService">
        <constructor-arg value="hello" />
        <property name="who" value="world" />
        <property name="myService" ref="myService" />
    </bean>

</beans>
```

```java
public class MyApplication {

    public static void main(String[] args) {
        ApplicationContext applicationContext = createApplicationContext();

        MyService myService1 = (MyService) applicationContext.getBean( s: "myService");
        MyService myService2 = applicationContext.getBean(MyService.class);
    }
```

Integrations-Tests mit Spring

- Unit-Tests sollten nur einzelne Klassen testen und sehr schnell sein.
- Um das Zusammenspiel von mehreren Klassen zu testen, kann z.B. ein Integrations-Test erstellt werden, der den Spring-Context verwendet.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class CreditCardProcessorIntegrationTest {

    @Autowired
    private CreditCardProcessor creditCardProcessor;

    @Autowired
    private PaymentService paymentService;

    @Test
    public void myTest() throws Exception {
        creditCardProcessor.charge( ccNumber: "123",  amountInEuroCents: 50);
    }
}
```

Aufgabe: Spring mit XML

- Öffnen Sie Beispiel-Projekt *2-dependency-injection-spring*
- Schreiben Sie die Bean-Definitionen in die Datei *src/main/resources/applicationContext.xml*
- Führen Sie die Java-Klasse *SpringMain* aus, um die Bean-Definitionen zu testen.
- Erstellen Sie die Datei *src/test/resources/testContext.xml* und überschreiben Sie den *PaymentServiceImpl* mit Ihrer Test-Implementierung.
  (Tipp: *<import />*)
- Befüllen Sie die Klasse *CreditCardProcessorIntegrationTest* mit Ihren zuvor gesicherten Tests.

Spring: Annotation-Configuration

```
public class MyOtherService {

    private String salutation;
    private String who;
    private MyService myService;

    public MyOtherService(String salutation) { this.salutation = salutation; }

    public void setWho(String who) { this.who = who; }

    @Autowired
    public void setMyService(MyService myService) {
        this.myService = myService;
    }

}
```

Spring: Annotation-Configuration

```xml
<context:component-scan base-package="com.example" />

<bean class="java.lang.String" id="salutation">
    <constructor-arg value="hello" />
</bean>

<bean class="java.lang.String" id="who">
    <constructor-arg value="Mark" />
</bean>
```

```java
@Component
public class MyService {
}
```

```java
@Service
public class MyOtherService {

    private String salutation;
    private String who;
    private MyService myService;

    @Autowired
    public MyOtherService(@Qualifier("salutation") String salutation) {
        this.salutation = salutation;
    }

    @Autowired
    public void setWho(@Qualifier("who") String who) {
        this.who = who;
    }

    @Autowired
    public void setMyService(MyService myService) {
        this.myService = myService;
    }

}
```

Aufgabe: Spring mit Annotations

- Öffnen Sie Beispiel-Projekt *2-dependency-injection-spring*
- Stellen Sie die Beans auf Annotations um. Verwenden Sie Component-Scan, um die entsprechenden Beans zu finden.
- Führen Sie die Java-Klasse *SpringMain* aus, um die Bean-Definitionen zu testen.
- Passen Sie auch den Test-Context an und verwenden Sie auch hier Component-Scan. Verwenden Sie Spring-Profile, um zwischen den verschiedenen Implementierungen zu unterscheiden.

Spring: Java-Config

VORHER:

```xml
<beans>

    <bean class="com.example.MyService" id="myService" />

    <bean class="com.example.MyOtherService">
        <constructor-arg value="hello" />
        <property name="who" value="world" />
        <property name="myService" ref="myService" />
    </bean>

</beans>
```

NACHER:

```java
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyService();
    }

    @Bean
    public MyOtherService myOtherService(MyService myService) {
        MyOtherService myOtherService = new MyOtherService( salutation: "hello");
        myOtherService.setWho("world");
        myOtherService.setMyService(myService);
        return myOtherService;
    }
}
```

```xml
<bean class="com.example.AppConfig" />
```

Aufgabe: Spring mit Java-Config

- Öffnen Sie Beispiel-Projekt *2-dependency-injection-spring*
- Erstellen Sie eine neue Config-Bean und annotieren Sie diese mit *@Configuration.* In der XML-Datei bleibt lediglich ein Verweis auf die Config-Bean.
- Migrieren Sie die übrigen Beans auf Java-Config
- Führen Sie die Java-Klasse *SpringMain* aus, um die Bean-Definitionen zu testen
- Entfernen Sie die Datei src/test/resources/testContext.xml und erstellen Sie stattdessen eine TestConfiguration-Klasse. Diese sollte die zuvor erstellte Config-Bean importieren und von dieser ableiten. Nutzen Sie auch hier Spring-Profile. (Tipp: *@Import*)

Spring MVC

- Framework, mit dessen Hilfe Webanwendungen auf Basis von Spring erstellt werden können.

```java
@Controller
public class MyWebController {

    @RequestMapping(method = RequestMethod.GET, value = "/hello")
    public String sayHello(@RequestParam("name") String name, Model model) {
        model.addAttribute( s: "message",  o: "Hello " + name);
        return "mytemplate";
    }

    @RequestMapping(value = "/hello2")
    @ResponseBody
    public String noTemplate() {
        return "hello without template";
    }
}
```

Spring Boot

- Es ist teilweise sehr kompliziert, bestimmte Spring-Komponenten einzubinden.
- XML-Configuration hat sich als schwerfällig herausgestellt
- Spring Boot ist ein Framework, dass die Komplexität von Spring versteckt und das Schreiben neuer Applikation deutlich erleichtert.
- Es ist mit vielen Automatismen ausgestattet, das Auto-Configuration ermöglicht.
- Da Spring-Boot Anwendungen sehr leicht zu erstellen sind, sind diese besonders für Micro-Services geeignet.

Aufgabe: Spring Boot Anwendung erstellen

- Gehen Sie auf start.spring.io und erstellen Sie eine neue Anwendung. Verwenden Sie das Modul “web” und eine beliebige Template-Engine.
- Öffnen Sie das Projekt in der IDE und sehen Sie sich das Projekt genau an. Fragen?
- Erstellen Sie einen neuen *@Controller* und eine Web-Methode, die per *GET* aufgerufen werden kann. Nutzen Sie *@ResponseBody*, damit die Antwort sofort angezeigt wird. Als Query-Parameter sollte der Name akzeptiert werden und in der Ausgabe verwendet werden.
- Erstellen Sie eine weitere Methode unter Verwendung der gewählten Template-Engine.

Aufgabe: Spring Boot Anwendung erweitern

- Schreiben Sie ihren Namen in eine beliebige Property unter *src/main/resources/application.properties* und verwenden Sie *@ConfigurationProperties* um diese auszulesen.
- Auch die URL des Nachbars sollte in einer solchen Property stehen.

- Erstellen Sie eine REST-Methode, die zählt, wie oft sie aufgerufen wurde.
- Sorgen Sie dafür, dass der Zähler auch dann fortgeführt wird, wenn die Anwendung neugestartet wird.
- Bonus: Binden Sie den *spring-boot-starter-validation* ein und sorgen Sie dafür, dass der Name aus der vorherigen Aufgabe immer mit einem “J” beginnt. Lesen Sie hierfür das Kapitel über “Validation” aus der Spring Boot Reference Documentation.

Java 8: Default Methods in Interfaces

```
interface Formula {
    double calculate(int a);

    default double sqrt(int a) {
        return Math.sqrt(a);
    }
}
```

Java 7: Wiederholung Anonyme Klassen

```
interface Person {
    void sayHello();
}
```

```
Person person = new Person() {
    @Override
    public void sayHello() {
        System.out.println("Hello there");
    }
};

person.sayHello();
```

Java 8: Lambda expressions

Beispiel in Java 7:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return b.compareTo(a);
    }
});
```

Java 8: Lambda expressions

Ab Java 8:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, (String a, String b) -> {
    return b.compareTo(a);
});
```

Java 8: Lambda expressions

Es geht noch kürzer:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, (String a, String b) -> b.compareTo(a));
```

Java 8: Lambda expressions

Und noch kürzer:

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return b.compareTo(a);
    }
});
```

```
List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");

Collections.sort(names, (a, b) -> b.compareTo(a));
```

Java 8: Lambda expressions

Dafür darf die anonyme Klasse nur eine Methode haben (kann - muss aber nicht - mit *@FunctionalInterface* forciert werden):

```
@FunctionalInterface
interface Converter<F, T> {
    T convert(F from);
}
```

```
Converter<String, Integer> converter = (from) -> Integer.valueOf(from);
Integer converted = converter.convert( from: "123");
System.out.println(converted);    // 123
```

Java 8: Method references

Ab Java 8 können auch Methoden referenziert werden:

```
Function<String, Integer> stringToInt = Integer::valueOf;

Integer onetwothree = stringToInt.apply( t: "123");

Function<String, Integer> stringToIntSquared = stringToInt.andThen((i) -> i * i);

Integer bigNumer = stringToIntSquared.apply( t: "123");
```

Java 8: Method references

Es geht auch mit mehr Parametern:

```
interface Addition<A, B, R> {
    R apply(A a, B b);
}
```

```
Addition<Integer, Integer, Integer> addTwoNumbers = (a, b) -> a + b;
Integer three = addTwoNumbers.apply( a: 1,  b: 2);
```

Java 8: Supplier und Consumer

Supplier:

```
Supplier<Person> personFactory1 = PersonImpl::new;
Person p1 = personFactory1.get();

Supplier<Person> personFactory2 = () -> new PersonImpl();
Person p2 = personFactory2.get();

Supplier<Integer> intGetter = () -> new Random().nextInt();
Integer randomInt = intGetter.get();
```

Consumer:

```
Consumer<Person> helloMethod = Person::sayHello;
helloMethod.accept(p1);

Consumer<String> sysout = System.out::println;
sysout.accept( t: "hello there");

Consumer<Integer> ageSetter = p2::setAge;
ageSetter.accept( t: 18);
```

Java 8: Streams

Kann auf beliebigen Collections ausgeführt werden:

```
List<String> stringCollection = new ArrayList<>();
stringCollection.add("ddd2");
stringCollection.add("aaa2");
stringCollection.add("bbb1");
stringCollection.add("aaa1");
stringCollection.add("bbb3");
stringCollection.add("ccc");
stringCollection.add("bbb2");
stringCollection.add("ddd1");

stringCollection
        .stream()
        .filter((s) -> s.startsWith("a"))
        .forEach(System.out::println);
```

Java 8: Streams

Es kann sortiert und “gemapped” werden:

```
stringCollection
        .stream()
        .map(String::toUpperCase)
        .sorted((a, b) -> b.compareTo(a))
        .forEach(System.out::println);

List<Integer> intList = stringCollection
        .stream()
        .map((s) -> Integer.valueOf(s) + 10)
        .collect(Collectors.toList());
```

Java 8: Streams

Es kann gesucht werden:

```
boolean anyStartsWithA =
        stringCollection
                .stream()
                .anyMatch((s) -> s.startsWith("a"));

System.out.println(anyStartsWithA);      // true

boolean allStartsWithA =
        stringCollection
                .stream()
                .allMatch((s) -> s.startsWith("a"));

System.out.println(allStartsWithA);      // false

boolean noneStartsWithZ =
        stringCollection
                .stream()
                .noneMatch((s) -> s.startsWith("z"));

System.out.println(noneStartsWithZ);      // true
```

Java 8: Streams

Es kann auch gezählt werden:

```
long startsWithB =
        stringCollection
                .stream()
                .filter((s) -> s.startsWith("b"))
                .count();

System.out.println(startsWithB);    // 3
```

Java 8: Streams

Und zuletzt können die Elemente kombiniert werden:

```
List<String> stringCollection = new ArrayList<>();
stringCollection.add("ddd2");
stringCollection.add("aaa2");
stringCollection.add("bbb1");
stringCollection.add("aaa1");
stringCollection.add("bbb3");
stringCollection.add("ccc");
stringCollection.add("bbb2");
stringCollection.add("ddd1");

Optional<String> reduced =
        stringCollection
                .stream()
                .sorted()
                .reduce((s1, s2) -> s1 + "#" + s2);

reduced.ifPresent(System.out::println);
// "aaa1#aaa2#bbb1#bbb2#bbb3#ccc#ddd1#ddd2"
```

Aufgabe 1: Java 8

Schreiben Sie mit Hilfe von Java 8 den folgenden Code neu.

```java
// Aufgabe 1
// Alle alphabetisch sortieren
List<String> names = Arrays.asList("Walter White", "Jessie Pinkman", "Saul Goodman");

Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return a.compareTo(b);
    }
});

for(String name : names) {
    System.out.println(name);
}
```

Aufgabe 2: Java 8

Schreiben Sie mit Hilfe von Java 8 den folgenden Code neu.

```java
// Aufgabe 2
// Anzahl der Elemente berechnen, die ein grosses oder kleines S haben
List<String> names = Arrays.asList("Walter White", "Jessie Pinkman", "Saul Goodman");

int count = 0;
for(String name : names) {
    if(name.contains("S") || name.contains("s")) {
        count++;
    }
}

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

Aufgabe 3: Java 8

Schreiben Sie mit Hilfe von Java 8 den folgenden Code neu.

```java
// Aufgabe 3
// Fuer jeden Namen, der mit "n" endet, zurueckgeben, wieviele Zeichen er
// hat. Diese Ausgabe sollte sortiert nach der Laenge (absteigend) sein.
List<Integer> lengths = new ArrayList<>();
for(String name : names) {
    if(name.endsWith("n")) {
        lengths.add(name.length());
    }
}
Collections.sort(lengths, new Comparator<Integer>() {
    @Override
    public int compare(Integer a, Integer b) { return b.compareTo(a); }
});
System.out.println("Lengths:");
for(Integer length : lengths) {
    System.out.println(length);
}
```

Weitere Themen

- Design Patterns:
  - Factory-Pattern, Builder-Pattern, MVC, MVP, MVVM, ...
- Jenkins:
  - Aufsetzen und Konfigurieren einer CI-Umgebung auf Basis von Jenkins
- Groovy:
  - Ein Blick auf eine andere JVM-Sprache
- Gradle:
  - Ein Crash-Kurs in diesem alternativen (besseren? :D) Build-System
- Wünsche?

Vielen Dank

Fragen?