PR1 – Formular für Lesenotizen 1
WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	6. Dezember 2020

a. Aus welchen Programmiersprachen hat Java wesentliche Elemente geerbt?
   C++ und Smalltalk
b. Wie formuliert man Algorithmen? **P**rogramm**AP**
   Sie werden in einer für Menschen verständlichen Sprache formuliert.

Schlüsselwörter

abstract	double	int	super
assert***	else	interface	switch
boolean	enum****	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto*	public	void
const*	if	return	volatile
continue	implements	short	while
default	import	static	_ (underscore)*****
do	instanceof	strictfp**	

Escape sequence

Ein String kann Spezialzeichen durch einen umgekehrten Schrägstrich (backslash) einleiten. Die Sequenz des backslash mit dem nachfolgenden Spezialzeichen nennt man auch escape sequence.
\t Tabulator-Zeichen
\n Zeichen für “neue Zeile”
\" Anführungszeichen
\\ Backslash

```
/**
   Grafikprogramm mit Kommentaren :D
 */
public class Graphic {                    //klasse Deklarieren
   public static void main(String[] args) {
      System.out.println(" _______");       //First line
      System.out.println(" /      \\ ");    //Backslash für Zeichen anstatt Operator
      System.out.println("/       \\ ");    //same
      System.out.println("-\"-'-\"-'-\"-");   //Backslash für Anführungsstriche
      System.out.println("\\       / ");    //same
      System.out.println(" \\_______/ ");    //same
   }
}
```

Programmierstruktur: Programmierkonventionen (Die Definition von Regeln zum programmieren)

- Sun style: öffnende geschweifte Klammer am Ende, schließende eigene Zeile
- „Klassennamen beginnen mit einem Großbuchstaben
- In Java: Camel Case – Class Namen Anfangsbuchstaben großschreiben und Methoden Namen Anfangsbuchstaben kleinschreiben
- Methoden 4 Zeilen einrücken & Statements 8 Zeilen
- „Jede Methode beginnt mit den Schlüsselwörtern public static void"
- „Bezeichner müssen exakt eingegeben werden"
- „Methoden, die aus mehreren Worten bestehen werden nicht mit einem Leerzeichen getrennt, sondern das nächste Wort wird groß geschrieben."

Primitive Datentypen (Java)

Datentyp	**bits**	**Wertebereich**
boolean	8 bit	True / false
byte	8 bit	-2^7 bis 2^7-1
short	16 bit	-2^{15} bis $2^{15}-1$
char	16 bit	0 bis 65535
int	32 bit	-2^{31} bis $2^{31}-1$
float	32 bit	+/-1,4E-45 bis +/-3,4E+38
long	64 bit	-2^{63} bis $2^{63}-1$
double	64 bit	+/-4,9E-324 bis +/-1,7E+308

Modulo:
In Java hat dr Operator das Vorzeichen des Dividenden z.B: 7 / -3 = 4 und -7 / 3 = -4.
In der Mathematik hat der Operator das Vorzeichen

Präzedenzregeln für Operator:
() ist höhergestellt als *, / und %,diese haben gleiche Präzedenz und höhergestellt als + und –

Ermittle die letzte Ziffer einer Zahl (Einerstelle)
Beispiel: Für 230857 ermittle die 7
Ausdruck: 230857 % 10

Ermittle die letzten 4 Ziffern der Sozialversicherungsnummer
Beispiel: Für 658236489 ermittle 6489.
Ausdruck: 658236489 % 10000

Ermittle die Zehnerstelle einer Zahl (zweitletzte Ziffer)
Beispiel: Für 7342 ermittle die 4.
Ausdruck: 7342 % 100 / 10

Ermittle, ob eine Zahl ungerade ist
Beispiel: Ist 6 ungerade?
Ausdruck: 6 % 2 (wenn das Ergebnis 0 ist, ist 6 gerade, sonst ungerade)

Warum lautet die Ausgabe von System.out.println(0.1 + 0.1 + 0.1); nicht 0.3 ?
Die Zahl 0.1 ist in der internen Bitdarstellung nicht genau dargestellt. Der Wert wird leicht gerundet und nun dreimal addiert. Dadurch kommt der Rundungsfehler
→ Wenn man mit double-Werten in Java rechnet folgen ungenaue Ergebnisse.

PR1 – Formular für Lesenotizen 2
WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

a. Was versteht man unter Programmierkonventionen?
   - Die Definition von Regeln zum programmieren

b. Was stimmt an dem folgenden Statement nicht?
   System.out.println("C:\Users\norbert\Documents\teafortwo");
   - „escape sequence“. Einmal „/n“ das gibt in der Ausgabe eine neue Zeile und „/t“ dadurch kommt eine Tabulator-Zeichen.

c. Warum gibt es hier einen Compilerfehler?
   public static void break(){ System.out.println("Spielabbruch"); }
   - „break“ ist ein Schlüsselwort in Java. „Bezeichner“ und Schlüsselwörter dürfen nicht identisch sein.

d. Mit welchem Ausdruck isoliert man die letzte Ziffer einer Zahl?
   - Man kann den %-Operator gut einsetzen, wenn man gezielt einzelne Stellen einer Zahl isolieren will.
     letzte Ziffer → % 10

e. Mit welchem Ausdruck isoliert man die vorletzte Ziffer einer Zahl?
   vorletzte Ziffer → % 100/10

f. Wie lautet die Ausgabe des folgenden Statements?
   System.out.println(1.3 * 5 – 5 / 2);
   - „4.5“

g. Warum lautet die Ausgabe von System.out.println(0.1 + 0.1 + 0.1); nicht 0.3 ?
   - Die Zahl 0.1 ist in der internen Bitdarstellung nicht genau dargestellt. Der Wert wird leicht gerundet und nun dreimal addiert. Dadurch kommt der Rundungsfehler
     Ans: 0.30000000000000004
   - Daraus folgt → Wenn man mit double-Werten in Java rechnet folgen ungenaue Ergebnisse.

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

(L.2.6-L2.8)

L.2.6 Variable und Zuweisung

<type> <name> = <value> ; Beispiel: double myGPA = 3.95;

Java 10: Typinferenz für lokale Variablen (der Compiler den Typ der Variablen aus dem Initialwert ableitet)
var myGPA = 3.95; *Vorsicht! Wenn var => int dann kann man daraus kein double machen*

L.2.7 Die for-Schleife

```
public static void main(String[] args){
    for(double i=0.00; i <= 3; i+=0.5){
        System.out.print(i + " ");
    }
}
```

L.2.7.6 Degenerierte for-Schleifen

Bezeichnet man Schleifen, die die Test-Bedingung in der Initialisierung nicht erfüllen oder Endlosschleifen

```
// a degenerate loop
for (int i = 10; i < 5; i++) {
    System.out.println("How many times do I print?");
   }
// another degenerate loop
for (int i = 10; i >= 0; i=i/2) {
    System.out.println(i);
   }
```

L.2.7.7 Zahlenfolgen erzeugen

(5 ist der Abstand zwischen den gewünschten Ausgaben)

Count	Gewünschte Ausgabe	5 * count	5 * count -3
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

L.2.8 Geltungsbereich

Variablen haben einen begrenzten Geltungsbereich (englisch: scope):
Der scope ist innerhalb der geschweiften Klammern

L.2.8.2 Lokale Variablen

Lokale Variable: Eine in einer Methode deklarierte Variable
Vorteile:

- So kommt es nicht so leicht zu Namenskollisionen.

Nachteil:

- es ist nicht so leicht, von jeder Methode auf die benötigten Daten direkt zuzugreifen.

a.) Gegeben ist folgender Code

```
public class Test {
    public static void main(String[] args) {
      int a=5, b, c=6;
      System.out.println(a+b+c);
    }
  }
```

- Der Code funktioniert nicht, b wurde nicht deklariert und ist Teil der Ausgabe. Dementsprechend funktioniert die Ausgabe auch nicht.

b.) Schreiben Sie eine for-Schleife, die die Funktionswerte der folgenden Parabel im Definitionsbereich [-10,10] im Abstand von Schritten der Schrittweite 0,5 ausgibt.f(x) = $4x^2 - 3x + 5$.

c.)
```
for (double i = -10; i <= 10; i+= 0.5 ) {
                System.out.println(4*i*i - 3*i + 5);
            }
```

d.) Gegeben ist folgende Schleife

```
for (int count = 1; count <= 5; count++) {
    ...
}
```

- Ergänzen Sie ein Statement, so dass die Ausgabe wie folgt lautet: 9 5 1 -3 -7

Count	Gewünschte Ausgabe	-4 * count	4 * count + 13
1	9	-4	9
2	5	-8	5
3	1	-12	1
4	-3	-16	-3
5	-7	-20	-7

```
for (int count = 1; count <= 5; count++) {
    System.out.print(4*count+13 + „“)
}
```

- Ergänzen Sie ein alternatives Statement, so dass die Ausgabe wie folgt lautet: 16 27 38 49 60

Count	Gewünschte Ausgabe	11 * count	1 * count + 5
1	16	11	2
2	27	22	7
3	38	33	12
4	49	44	17
5	60	55	22

```
for (int count = 1; count <= 5; count++) {
    System.out.print(1 * count + 5 + „“)
}
```

,,

Count	Gewünschte Ausgabe	8 * count	8 * count + 6
1	14	8	
2	22	16	
3	30		
4	38		
5	46		

Count	Gewünschte Ausgabe	-5 * count	-5 * count + 134
1	129	-5	
2	124	-10	
3	119	-15	
4	114		
5	109		

Count	Gewünschte Ausgabe	-5 * count	-5 * count + 134
1	1		
2	4		
3	9		
4	16		
5	25		

Count	Gewünschte Ausgabe	2 * count +2	-5 * count + 134
1	4	4	
2	8	6	
3	14	8	
4	22	10	
5	32	12	

PR1 – Formular für Lesenotizen
WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.2.9-L.2.Ende L.3.1-L.3.2 (einschl.)

Parametrisierung
Parameter: Ein vom Aufrufer an die Methode übergebener Wert.
Parametrisierte Methode: Eine Methode, die beim Aufruf eine Extra-Information erhält, z. B. die Anzahl zu zeichnendem Sternchen.
Eine parametrisierte Methode deklariert man in Java wie folgt:
public static void <name> (<type> <name>) {
    <statement(s)> ;
}

L.3.2 Methoden mit Rückgabewerte

Die Klasse Math

Methiode	Beschreibung
abs(value)	Absolutbetrag
ceil(value)	Aufrunden
cos(value)	Cosinus vom Bogenmaß
floor(value)	Abrunden
log(value)	Logarithmus zur Basis e
log10(value)	Logarithmus zur Basis 10
max(value1, value2)	der größere zweier Werte
min(value1, value2)	der kleinere zweier Werte
pow(basis, exponent)	basis potenziert zum exponent
random()	Zufallswert double >=0.0 und <1.0
round(value)	Kaufmännisches Runden auf die nächste ganze Zahl
sin(value)	Sinus vom Bogenmaß
sqrt(value)	Quadratwurzel
toRadians(value)	Umrechnung von Grad in Bogenmaß
toDegrees(value)	Umrechnung von Bogenmaß in Grad

Math.<method name> (<parameter(s)>)

Methoden mit Rückgabewert
Ein Rückgabewert (return value) ist ein Wert, der von einer Methode als Ergebnis bereitgestellt wird.

Ein Rückgabewert kann z. B. in einem Ausdruck verwendet werden oder einfach nur ausgegeben werden.
Ein Rückgabewert ist gewissermaßen das Gegenteil eines Parameters:
 Parameter liefern Information vom Aufrufer in die Methode.
 Rückgabewerte liefern Information aus der Methode an den Aufrufer.

```java
/**
klasse
*/
public class VierQuadrate {
    public static final int SIZE = 3;

    public static void main(String[] args) {
        drawbox(SIZE);
    }

    public static void drawbox (int size) {
        for (int i = 1; i <= 2; i++) {
            drawline(size);
            drawmid(size);
        }
        drawline(size);
    }

    public static void drawmid (int count) {
        for (int line = 1; line <= count; line++) {
            System.out.print("#");
            drawspace(count);
            System.out.print("|");
            drawspace(count);
            System.out.println("#");
        }
    }

    public static void drawline (int count) {
        for (int i = 0; i <= count; i++){
            System.out.print("=");
        }
        System.out.print("+");
        for (int i = 0; i <= count; i++){
            System.out.print("=");
        }
        System.out.println();
    }

    public static void drawspace(int count) {
        for (int i = 1; i <= count; i++){
            System.out.print(" ");
        }
    }
}
```

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.3.3-L.3.5 (einschl.)

Stringobjekte

String example = "speak friend and enter";
System.out.println(example.toUpperCase());

Methodenname	Beschreibung
charAt(index)	Zeichen an der gegebenen Indexstelle
indexOf(str)	Index, an dem der als Parameter gegebene String str in dem String-Objekt beginnt (-1, wenn er nicht vorkommt)
length()	Anzahl der Zeichen im String-Objekt
substring(index1, index2)	Die Zeichen von einschließlich index1 bis ausschließlich index2
toLowerCase()	Ein neuer String in Kleinbuchstaben
toUpperCase()	Ein neuer String in Großbuchstaben

3.4 Interaktive Programme

System.out	**System.in**
Für Ausgaben auf der Console	Für Eingaben auf der Console
Hat Methoden println und print	Hat Methoden für das Einlesen von Bytes
Kann sinnvoll direkt verwendet werden	Wird in der Regel nicht direkt verwendet.

L.3.4.2 Hilfsklasse Scanner

Methode	**Beschreibung**
nextInt()	Liest und gibt die Benutzereingabe als int zurück
nextDouble()	Liest und gibt die Benutzereingabe als double zurück
next()	Liest und gibt die Benutzereingabe als String zurück
nextLine()	Liest und gibt die nächste Eingabezeile als String zurück

```
import java.util.*;    // um Scanner benutzen zu können
public class Average {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Bitte drei Zahlen eingeben: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();
        int num3 = console.nextInt();
        double average = (double) (num1 + num2 + num3) / 3;
        System.out.println("Der Durchschnitt ist " + average);
    }
}
```

import java.util.*;

- Scanner als Parameter wird so angegeben: public static void methode1(Scanner console)

Tokens Scanner-Objekt liest Eingaben in Blöcken zwischen Leerraumstellen
23 John Smith 42.0 "Hello world" => 6 Tokens

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("How many numbers? ");
    int n= console.nextInt();
    int sum = readSum(console, n);
    System.out.println("The sum is " + sum);
}
public static int readSum(Scanner console, int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        System.out.print("Type a number: ");
        sum += console.nextInt();
    }
    return sum;
}
```

L.3.4.2 Pakete und Classpath

Pakete

Jede Java Klasse (java & class-Dateien) ist in einem Paket (package), dieser entspricht der Verzeichnisstruktur im Dateisystem. => 1. Zeile (bzw. nach Kommentar): packaga <paketname>;

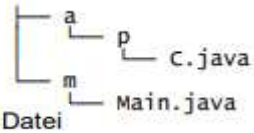
Classpath

Die Umgebungsvariable legt Compiler und Interpreter fest, wo i m Dateisystem nach Paketen und class-Dateien gesucht wird.
CLASSPATH kann mit Paketnamen verknüpft werden (sogar in einem anderen Verzeichnis)

- Beispiel: Verzeichnis /home/sch/prog1/bsp liegt Beispiel.java & .class mit package = prog1/bsp
  - ⇨ Man kann in der Konsole CLASSPATH = /home/sch und danach java /prog1/bsp die Datei ausführen

Modulepath

Konzept zur Strukturierung eines Programms, wichtig bei vielen Klasse



Wir übersetzen zuerst `C.java` und anschließend unter Angabe des CLASSPATH die Datei `Main.java`:

```
javac a/p/C.java
export CLASSPATH=a
javac m/Main.java
```

Anschließend führen wir das Hauptprogramm aus. Da sich dieses in einem weiteren Verzeichnis `m` befindet, ergänzen wir vorher den CLASSPATH:

```
export CLASSPATH=a;m
java Main
```

Wie unterscheidet man die Klassen im Programmtext?
Die Klassen im Programm werden durch Packages unterschieden, in denen sich die Classen befinden. Anschließend wird mit dem CLASSPATH das Programm Klasse für Klasse kompiliert und am Ende ausgeführt.

```
import java.awt.Point;
public class Swap {
public static void main(String[] args) {
        Point p1= new Point(5, 2);
        Point p2= new Point(-3, 6);
        swapPoints(p1, p2);
        System.out.println("p1: ("+p1.x+", "+p1.y+")");
        System.out.println("p2: ("+p2.x+", "+p2.y+")");
    }
    public static void swapPoints(Point p1, Point p2) {
      Point tmp = (Point) p1.clone();
      p1.x = p2.x;
      p1.y = p2.y;
      p2.x = tmp.x;
      p2.y = tmp.y;
    }
}
```

```
public class Wurzel {
    public static void main(String[] args) {
        System.out.println(zahlHochlDurchn(256, 4));
    }
    public static double zahlHochlDurchn(double a, double n) {
        double ans = Math.pow(a,1/n);
        return ans;
    }
}
```

PR1 – Formular für Lesenotizen
WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.4.1-L.4.3 (einschl.)

L.4.1.1 double Vergleiche (immer falsch)

- Vermeidbar: if (Math.abs(<double Wert> - <Betrag>) < epsilon(0.001))

```
int z;
if (x > y) {
    z = x;
} else {
    z = y;
}
```

Der vorstehende Code ist ersetzbar durch:

```
int z = Math.max(x, y);
```

L.4.1.2 Objekte miteinander vergleichen

Objekte (wie String, Point) werden durch Aufruf der Methode equals auf Gleichheit geprüft.

"A" Datentyp String = > Objekt mit Methoden
'A' Char = > primitiver Datentyp ohne eigene Methoden

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Bond")) {
    System.out.println("Shaken, not stirred!");
}
```

Methode	Beschreibung
`equals(str)`	Prüft, ob zwei Strings genau die gleichen Zeichen enthalten.
`equalsIgnoreCase(str)`	Dito, nur wird Groß-/Kleinschreibung dabei ignoriert.
`startsWith(str)`	Prüft, ob ein String am Anfang genau die Zeichen eines anderen Strings enthält
`endsWith(str)`	Prüft, ob ein String am Ende genau die Zeichen eines anderen Strings enthält

L.4.2 Mehrfachauswahl mit switch

Einschränkungen der switch-Anweisung

- Der Ausdruck in der *switch*-Anweisung muss vom Typ *char, byte, short, String* oder *int* sein Sog. enum-Typen sind ebenfalls möglich.

Die allgemeine Syntax des `switch`-Statements ist:

```
switch (<expression>) {
case <const expression> :
  <statement(s)> ;
  break;
...
case <const expression> :
  <statement(s)> ;
  break;
default:
  <statement(s)> ;
}
```

```
char geschlech
switch (geschl
case 'M':
    System.out
    break;
case 'W':
    System.out
    break;
default:
    System.out}
}
```

```
Scanner console = new Scanner(System.in);
int max = Integer.MIN_VALUE;
for (int i = 1; i <= 10; i++) {
    System.out.print("Zahl " + i + ": ");
    int n = console.nextInt();
    if (n > max) {
        max = n;
    }
```

L.4.3 Textverarbeitung mit String und char

Dies sind gültige Vergleichsoperationen für char-Werte (nicht Strings!): 'a' < 'b' oder 'Q' != 'q'
Methode der Klasse Character

Methode	Beschreibung	Beispiel
`getNumericValue(ch)`	Wandelt ein Zeichen, das aussieht wie eine Ziffer, in eine Zahl um	`Character.getNumericValue('6')` liefert 6
`isDigit(ch)`	Prüft, ob ch eines der Zeichen '0' bis '9' ist	`Character.isDigit('X')` liefert `false`
`isLetter(ch)`	Prüft, ob ch ein Buchstabe ist	`Character.isLetter('f')` liefert `true`
`isLowerCase(ch)`	Prüft, ob ch ein Kleinbuchstabe ist	`Character.isLowerCase('q')` liefert `true`
`isUpperCase(ch)`	Prüft, ob ch ein Großbuchstabe ist	`Character.isUpperCase('q')` liefert `false`
`toLowerCase(ch)`	Liefert den zugehörigen Kleinbuchstaben	`Character.toLowerCase('Q')` liefert 'q'
`toUpperCase(ch)`	Liefert den zugehörigen Großbuchstaben	`Character.toUpperCase('q')` liefert 'Q'

Chat viel mehr zahl als text charatker
Mit Zeichen kann man sogar rechnen. Der Ausdruck 'a'+10 hat den Wert 107. Das Zeichen wird hierbei in einen int-Wert umgewandelt und anschließend wird die +-Operation ausgeführt.

```java
import java.util.*;
public class Reverse {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Bitte geben Sie Ihren vollen Namen ein: ");
        String name = console.nextLine();

        int leer = 0;
        while (name.charAt(leer) != ' ') {
            leer = leer + 1;
        }
        String vorname = name.substring(0, leer);
        String nachname = name.substring(leer + 1);

        System.out.println("Ihr Name in umgekehrter Schreibweise ist: "
            + nachname + ", " + vorname);
    }
```

Joe Cocker
Cocker, Joe

vertikal

```java
public class Vertical {
    public static void main(String[] args) {
        vertical("Gut Holz!");
    }
    public static void vertical(String str){
        for (int i = 0; i<str.length();i++){
            char c = str.charAt(i);
            System.out.println(c);
        }
```

Switch befehl

```java
public class test{
    public static void main(String[] Args){
        Scanner console = new Scanner(System.in);
        int monat = console.nextInt();
        switch (monat){
            case 1:
                System.out.println("31");
            case 2:
                System.out.println("28");
            case 3:
                System.out.println("30");
    }
}
```

rechnen

```java
import java.util.*;
public class KumulProd {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Wieviele Zahlen? ");
        int n = console.nextInt();

        int product = 1;
        int temp = 1;
        for ( int i = 1; i <= n; i++){
            System.out.print( i + "-te Zahl: ");
            temp = console.nextInt();

            product *= temp;
        }
        System.out.println("Das kumulative Produkt ist: " + product );
```

Min-max

```java
public class Range {
    public static void main(String[] args) {
        printRange(5,10);
    }
    public static void printRange(int eingabe1, int eingabe2){

        if (eingabe1 == eingabe2) {
            System.out.println("["+ eingabe1 + "]");
        } else if (eingabe1 >= eingabe2) {
            System.out.print("[" + eingabe1 + ", ");
            for (int i = eingabe1 -1; eingabe2 <= i -1;i--){
                System.out.print(i + ", ");
            }
            System.out.println(eingabe2 + "]");
        } else if (eingabe1 <= eingabe2) {
            System.out.print("[" + eingabe1 + ", ");
            for (int i = eingabe1 + 1; i <= eingabe2 -1;i++){
                System.out.print(i + ", ");
            }
            System.out.println(eingabe2 + "]");
```

Übrigens: der Grund, weshalb man das Ergebnis explizit in ein char umwandeln muss, ist, dass der Zieltyp einen kleineren Wertebereich (hier char, also 2 Byte) als der Quellausdruck hat (hier int, also 4 Byte). Bei dem Ausdruck 'a' + 10 wäre dies vielleicht nicht erforderlich, denn hier sehen wir ja mit einem Blick, dass das Ergebnis 107 noch klein genug ist, um in 2 Byte dargestellt werden zu können. Was aber ist mit dem Ausdruck 'a' + 65439 ? Ist das Ergebnis noch ein gültiges Zeichen oder ist der Wertebereich von 2 Byte überschritten? Da das Ergebnis der +-Operation zwischen einem char-Wert und einem intWert potentiell zu groß werden kann, erwartet der Compiler, dass man explizit eine Typumwandlung programmiert. Die explizite Typumwandlung kann dazu führen, dass das Ergebnis ungenauer ist als das Original, weil zu wenig Bits im Ziel-Datentyp zur Verfügung stehen. Man nennt eine solche Typumwandlung eine narrowing conversion, zu Deutsch eine Konvertierung, die den Wertebereich einschränkt. Einschränkende Konvertierungen müssen in Java immer explizit programmiert werden.

Die +-Operation verhält sich außerdem etwas unerwartet, wenn sie auf zwei Zeichen losgelassen wird. Dann erhält man nicht etwa eine Zeichenkette der Länge 2, sondern die Summe der beiden beteiligten Codes: System.out.print('A'+'A'); gibt den Wert 130 auf der Console aus.

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.4.4-L.4.6 (einschl.) L.5.1-L.5.3 (einschl.)

L00 Return

- Alle if/else Pfade müssen einen return Befehl enthalten
- Compiler kann nicht Bedingungen vergleichen z.B.: if (a <= b) u. else if (a > b) return Statements besitzen kommt es trotzdem zu einer Fehlermeldung. Immer ende Methode return
- For-schleife mit if/else muss auch return berücksichtigt werden wenn Schleife/if kein Mal läuft

L.4.4 Der ternäre Operator

Syntax des Operators: **<varname> = <boolean expr> ? <expr1> : <expr2>**
Bsp: *String text = geschlecht == 'w' ? "weiblich" : geschlecht == 'm' ? "männlich" : "Keine Angabe"*

phase = alter < 1? " Baby ": alter < 3? " Kleinkind ": alter < 6? „Vorschulkind“: alter < 13? „Schulkind“: alter < 20? „Tennager“: „Erwachsen“

L.4.5 Exceptions erzeugen

Exceptions, Programm auf einen Fehler stößt
Syntax: **throw new <exception-class>(<message>)**
Bsp: throw new IllegalArgumentException("jahre muss >= 0 sein.")

L.4.6 Random - Erzeugung Intervall
min. – max.: nextInt(max – min + 1) + min
Bsp. Random rand= new Random();
*int a= rand.**nextInt**(MAXNUM)+1;*

Buchstaben v1. **If Anweisungen** |
v2. **<String>***“ABCD“.***char.At(z);** *(int z = rand.nextInt(5);*

```
public static char zufallsVokal() {
    Random rand = new Random();
    String vokale = "aeiou";
    int laenge = vokale.length();
    char c = vokale.charAt(rand.nextInt(laenge));
    return c;
}
```

L.5.1 while-Schleifen

***Definite Schleife**: Schleifen, deren Anzahl der Durchläufe vorher bekannt*.

Indefinite Schleife - while-Schleife

Sentinel-Schleife: Schleife, die bis zu Sentinel-Wert läuft. Nutzt häufig das Zaunpfahlstil

Sentinel-Werte
• Ein spezieller (Eingabe-)Wert, der das Ende einer Folge von Daten(-eingaben) signalisiert

```
24 public static final int SENTINEL = -1;
25     public static int einlesen(Scanner console) {
26         System.out.print("Enter a number (" + SENTINEL + " to quit): ");
27         return console.nextInt();
28     }
29     public static void main(String[] args) {
30         Scanner console = new Scanner(System.in);
31         int sum = 0;
32         int inputNumber = einlesen(console);
33         while (inputNumber != SENTINEL) {
34             sum += inputNumber*inputNumber;
35             System.out.println("Sum of squares: "+sum);
36             inputNumber = einlesen(console);
37         }
38         System.out.println("Total sum of squares is " + sum);
39     }
```

```
<initialization>;
while (<condition>) {
<statement(s)> ;
 <update>;
}
```

L.5.3.1 Logische Ausdrücke

&& = und **|| = Oder (Inklusiv)**

^ = Xor (exklusives Oder) **! = Nicht**

Operator		&&	||	^	!
Beschreibung		Und	Oder (inklusives)	Xor (exklusives Oder)	Nicht
Beispiel		(9!=6)&&(2<3)	(2==3)||(-1<5)	(9!=6)^(2<3)	!(7>0)
Ergebnis		true	true	false	false
p	q	p && q	p || q	p ^ q	!p
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

L.5.3.2 Präzedenzregeln für Operatoren

Operator	Rang	Typ	Beschreibung
++, --	1	Arithmetisch	Inkrement / Dekrement
+, -	1	Arithmetisch	Unäres Plus und Minus
!	1	boolean	Negation
(Typ)	1	Jeder	Typumwandlung
*, /, %	2	Arithmetisch	Multiplikative Op.
+, -	3	Arithmetisch	Additive Op.
+	3	String	String-Konkatenation
<, <=, >, >=	5	Arithmetisch	Numerische Vergleiche
==, !=	6	Primitiv	Gleich-/Ungleichheit von Werten
==, !=	6	Objekt	Gleich-/Ungleichheit von Referenzen
^	8	boolean	Logisches exkl. Oder
&&	10	boolean	Logisches Und
||	11	boolean	Logisches Oder
=	13	Jeder	Zuweisung
*=, /=, %=, +=, -=	14	Jeder	Zuweisung mit Operation

```
8     public static double log(double arg, double base) {
9         if ( arg <= 1 || base <= 1){
10            throw new IllegalArgumentException("arg || base müssren größer als 1 sein.");
11        }
12        return Math.log(arg) / Math.log(base);
13    }
```

```
1   import java.util.*;
2 public class test {
3     public static void main(String[] args){
4         Scanner console = new Scanner(System.in);
5         int count = 0;
6         String pw = "";
7         while(pw.length() <= 8){
8             System.out.print("Ihr neues Passwort:");
9             pw = console.nextLine();
10            System.out.println("Bitte ein längeres Passwort eingeben (mind. 8 Zeichen)");
11            count ++;
12        }
13        System.out.println("Sie haben " + count + " Versuche gebraucht");
14    }
15 }
```

c. Schreiben Sie ein Programm, das eine sich zufällig entwickelnden Zahlenfolge simuliert:

d. Schreiben Sie ein Programm, das den Benutzer nach einem neuen Passwort fragt, und zwar so lange, bis die Eingabe mindestens 8 Zeichen lang ist. Am Ende soll die Anzahl der Versuche ausgegeben werden. Gewünschter Beispielablauf:

Ihr neues Passwort: test
Bitte ein längeres Passwort eingeben (mind. 8 Zeichen)
Ihr neues Passwort: test243
Bitte ein längeres Passwort eingeben (mind. 8 Zeichen)
Ihr neues Passwort: geheim99
Sie haben 3 Versuche gebraucht

e. Was stimmt mit den folgenden Ausdrücken nicht?
a != 5 || 6 5 <= i <= 8

f. Welchen Wert haben die folgenden Ausdrücke (nehmen Sie an, dass die Variable lottogewinn vom Typ boolean existiert)?

17 < 15 && lottogewinn
17 > 15 || lottogewinn
13 != 12 ^ 12 != 11
!(Math.PI < 4)

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.6.1-L.6.3 (einschl.)

L.5.4 Das continue-Statement

break-Anweisung: Beendet eine Schleife unmittelbar.
continue-Anweisung: Springt unmittelbar zum Ende des Schleifenblocks.

L.6.1 I/O und File-Objekte

Methode	Beschreibung
`canRead()`	Prüft, ob Datei gelesen werden kann
`delete()`	Löscht Datei
`exists()`	Prüft, ob Datei auf dem Datenträger existiert
`getAbsolutePath()`	Gibt den Pfad im Dateisystem zurück (z. B. `"/home/stud/user/datei.txt"`)
`getName()`	Gibt den Dateinamen zurück
`isDirectory()`	Prüft, ob es sich um ein Verzeichnis handelt
`isFile()`	Prüft, ob es sich um eine Datei handelt
`length()`	Liefert die Größe der Datei in Bytes
`mkdirs()`	Erzeugt das repräsentierte Verzeichnis, falls nicht schon vorhanden.
`renameTo(`*file*`)`	Benennt die Datei um in *file*

L.6.2 Dateien öffnen und schließen / Ausnahmen ignorieren

Syntax: `Scanner` *<name>* `= new Scanner(new File("`*<file name>*`"));`

Exception/Ausnahme: ein Objekt, das einen Laufzeitfehler anzeigt.

Überprüfungsbedürftige Ausnahmen (checked exceptions)
Eine Ausnahme, deren Prüfung programmiert werden muss entweder catch oder throws

Nicht überprüfungsbedürftige Ausnahmen (unchecked exceptions)
Eine Ausnahme, deren Prüfung nicht programmiert werden muss

Throws
Schlüsselwort im Methodenkopf, dass aussagt, dass die Methode potenziell eine Ausnahme wirft z.B.: `public static void main(String[] args) throws FileNotFoundException {`

Dateien schließen
Um Speicherverbrauch und den nicht Zugang zur Datei zu vermeiden mit `scanner.close();`

Berücksichtigung der Locale: `input.useLocale(new Locale("en", "US"));`

Das Statement new File("datei.txt"); führt nicht zum Anlegen einer neuen Datei.
- Erstellt ein File Objekt, dass eine Datei repräsentiert und damit Metadaten wie Name, Länge usw. zugreifen kann, jedoch nicht auf den Inhalt. Dafür müsste man ein Scanner Objekt erstellen

In Ihrer main-Methode rufen Sie die folgende Methode auf:

public static void openFile throws FileNotFoundException { … }

- Was bedeutet das throws … im Methodenkopf?
  - Schlüsselwort, dass die Methode potenziell eine Ausnahme wirft, ignoriert es
  - Kann man entweder main oder eigene Methode verwenden (oder beides, wenn nötig)
- Was müssen Sie in Ihrer eigenen main-Methode programmieren, um das Programm übersetzen zu können? - public static void main(String[] args) throws FileNotFoundException {

```java
import java.util.Scanner;
public class Quadratic {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Geben Sie Parameter für die Gleichung
ax² + bx + c = 0 ein: ");
      double a = ask(scanner);
      double b = ask(scanner);
      double c = ask(scanner);

      quadratic(a, b, c);
   }
   public static double ask(Scanner scanner) {
      return scanner.nextDouble();
   }

   public static void quadratic(double a, double b, double c) {
      double d = diskriminante(a, b, c);
      String plus = "+";
      String minus = "-";
      String none = "";

      if (a == 0) {
         System.out.println("a muss ungleich 0 sein");
      // double Wert ungenau deshalb sicherstellen das der negarive
Wert groß ist
      } else if (d < 0 && Math.abs(d) >= 0.0001) {
         System.out.println("Keine reelle Lösung");
      // dasseble um Rundungsfehler zu vermeiden statt nomarlerweise d == 0
      } else if (Math.abs(d) < 0.0001) {
         double x1 = 0;
         if (0 <= d) {
            x1 = cal(a, b, c, plus, minus, plus);
         } else {
            x1 = cal(a, b, c, plus, minus, none);
         }
         System.out.println("Lösung: " + round1(x1));
      } else {
         double x1 = cal(a, b, c, plus, minus, plus);
         double x2 = cal(a, b, c, plus, minus, minus);
         print(x1, "Erste");
         print(x2, "Zweite");
      }
   }
   /** Lösungsmenge bestimmen mit der Mitternachtsformel (abc-Formel). ZunächstAbfrage von Paramter s zur Unterscheidung der
beiden Lösungen */
   public static double cal(double a, double b, double c, String plus, String minus, String s) {
      if (s.equals(plus)) {
         return (-b + Math.sqrt(b*b-4*a*c))/(2*a);
      } else if (s.equals(minus)) {
         return (-b - Math.sqrt(b*b-4*a*c))/(2*a);
      } else {
         return -b/(2*a);
      }
   }/** einfacher print Befehl wobei die Lösung x gerundet wird*/
   public static void print(double x, String s) {
      System.out.println(s + " Lösung = " + round1(x));
   }/** return die Diskriminante (der Wurzelausdruck in der Gleichung)*/
   public static double diskriminante(double a, double b, double c) {
      return b*b-4*a*c;
   }/** Rundet eine gegebene Zahl auf eine Nachkommastelle. */
   public static double round1(double value) {
      if (Double.isNaN(value)) return value;
      if (Double.isInfinite(value)) return value;
      return Math.round(value * 10) / 10.0;
   }
```

```java
public class FirstWord {
    public static void main(String[] args) {
        System.out.println(firstWord("    "));
    }

    public static String firstWord(String s) {
        int start= 0;
        while (start < s.le           .charAt(start) == ' ') {
            start++;
        }

        int stop= start;
        while (stop < s.length() && s.charAt(stop) != ' ') {
            stop++;
        }
        return s.substring(start, stop);
    }
}
```

```java
import java.io.*;
import java.util.Scanner;
public class Aufd {
    public static void main(String[] args) throws FileNotFoundException {
        Scanner scanner = new Scanner(new File("names.txt"));
        int count = 0;
        String name = "";
        while (scanner.hasNext()) {
            name = scanner.next();
            if (Character.toLowerCase(name.charAt(0)) == 'x') {
                count++;
            }
        }
        System.out.println("Anzahl Namen mit X: " + count);
    }
}
```

```java
import java.util.Random;
public class RandomText {
   /** main Methode */
   public static void main(String[] args) {
      Random random = new Random();
      int row = random.nextInt(4) + 5;
      String vocals = "aeiou";
      for (int i = 0; i < row; i++) {
         int chars = random.nextInt(3) + 4;
         for (int j = 0; j < chars; j++) {
            char vocal =
vocals.charAt(random.nextInt(vocals.length()));
            System.out.print(vocal);
         }
         System.out.println();
```

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.5.4 und L.5.6 |L.6.6-L.6.7 (einschl.)

L.5.4 Token-basierte Verarbeitung von String

• Scanner <name> = new Scanner(<String>);

Abwechselndes Token- und Zeilenbasiertes Einlesen

• Möglichst vermeiden, da häufig Fehler auftreten, wenn man z.B.: bei der Konsole nach scanner.next() danach scanner.nextLine() abfragt, da nextLine() nur den Zeilenumbruch liest

L.6.6 Dimensionierung des try-Blocks

Syntax:

```
try {
   <statement(s) >;
} catch (<exception-type> <name>) {
   <statement(s)>;
} catch (<exception-type> <name>) {
   <statement(s)>;
} finally {
   <statement(s)>;
}
```

`<name> = e` Scanner-Objekte vor dem try Block deklarieren & initialisieren mit Scanner
`<name> = null` wegen Geltungsbereich

finally-Block lediglich darum, eine offene Datei zu schließen

L.6.7 Ausgabe in Dateien

In der Vorlesung haben wir die folgende Möglichkeit kennen gelernt, eine Datei zum Schreiben zu öffnen:

```
PrintStream <name> = new PrintStream(new File("<file name>"));
```

FileOutputStream: Dateien können dabei zum Anhängen geöffnet werden.
Beispiel:

```
PrintStream output = new PrintStream(new FileOutputStream(new File("output.txt"), true));
output.println("Appended Text"); output.close();
// Das true weist das FileOutputStream-Objekt an, die Datei nicht zu überschreiben, sondern Daten anzuhängen.
```

```
public static void verarbeite(Scanner erloeseScanner) {
   int gesamteinheiten= 0;
   double gesamterloes = 0.0;
   while (erloeseScanner.hasNextInt()) {
      int verkauft= erloeseScanner.nextInt();
      double einzelpreis= 0.0;
      try {
         einzelpreis= erloeseScanner.nextDouble();
      } catch (NoSuchElementException e) {
         System.out.print("Datenfehler im Einzelpreis: ");
         System.out.println(erloeseScanner.next());
         return;
      }
      gesamteinheiten += verkauft;
      gesamterloes += verkauft * einzelpreis;
   }
   double durchschnittspreis;
   if (gesamteinheiten != 0) {
      durchschnittspreis= gesamterloes / gesamteinheiten;
   } else {
      durchschnittspreis= 0.0;
   }
   System.out.println(" Einheiten : "+gesamteinheiten);
```

Lernzielfragen

```
import java.util.Scanner;
public class test{
   public static void main (String[] args){
      Scanner sc = new Scanner(System.in);
      int sum = 0;
      int input = 0;
      while(true){
         System.out.println("Geben sie eine Zahl ein (mit -1 beenden sie das)");
         input = sc.nextInt();
         if( input == -1){
            break;
         } else if (input %5 ==0){
            sum += input;
         }
      }
      System.out.println("Summe = " + sum);
   }
}
```

Beschreiben Sie den Ablauf zur Laufzeit beim Erzeugen, Weiterreichen und Abfangen einer Exception. Skizzieren Sie ein Beispiel mit mindestens zwei beteiligten Methoden und unterscheiden die Fälle von überprüfungsbedürftigen und nicht überprüfungs-bedürftigen Exceptions.

- Bei der Erzeugung einer Exception kann sie entweder sofort behandelt werden mit **catch** oder weitergereicht werden mit **throws** wie bei überprüfungsbedürftige Exceptions.
- Nicht überprüfungsbedürftige Exceptions muss man nicht throwen und somit catchen, jedoch es sich solche Fehler zu behandeln

Schreiben Sie ein Programm, das den Benutzer nach einem Dateinamen und einer Textzeile fragt. Hängen Sie die Textzeile ans Ende der Datei an.

```
import java.io.*;
import java.util.Scanner;
public class Auff {
    public static void main(String[] args) throws FileNotFoundException{
        Scanner scanner = new Scanner(System.in);
        System.out.print("Dateinamen angeben: ");
        String name = scanner.nextLine() + ".txt";
        PrintStream output = new PrintStream(new FileOutputStream(new File(name), true));
        System.out.print("Geben Sie Inhalt an: ");
        String text = scanner.nextLine();
        output.println(text);
        output.close();
        scanner.close();
    }
}
```

Schreiben Sie ein Programm, das aus einer Datei paarweise zwei ganze Zahlen ausliest.
Beispieldatei: 8 9 4 2 3 0 6 2
Das Programm soll jeweils die erste durch die zweite Zahl dividieren (Ganzzahldivision) und das Ergebnis ausgeben. Division durch 0 sollen Sie mit try/catch abfangen. Schließen Sie die Datei auch im Fehlerfall.

```java
import java.io.*;
import java.util.Scanner;
public class Aufg {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        Scanner input = null;
        String name = null;
        do {
            System.out.print("Geben Sie einen Dateinamen an: ");
            name = console.nextLine() + ".txt";
            try {
                input = new Scanner(new File(name));
            } catch (FileNotFoundException e) {
                System.out.println("Diesen Dateinamen gibt es nicht");
            }
        } while(input == null);
        int number1 = 0;
        int number2 = 0;
        int ergebnis = 0;
        while (input.hasNextLine()) {
            number1 = input.nextInt();
            number2 = input.nextInt();
            try {
                ergebnis = number1 / number2;
                System.out.println(number1 + " / " + number2 + " = " + ergebnis);
            } catch (ArithmeticException e) {
                System.out.println(number1 + " / " + number2 + ": Unerlaubte Division duch 0!");
            }
        }
        input.close();
        console.close();
    }
}
```

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	06.12.2020

L.7.1-L.7.2 (einschl.) L.8.1-L.8.2 (einschl.)

L.7 Arrays

Standard initialisiert `(int 0, double 0.0, boolean false, char '\0', String null)`

L.7.2.1 null-Referenz

null: eine Referenz, die auf kein Objekt verweist

Methode	**Beschreibung**	**Beispiel** String s = "long book";
`toCharArray()`	Separiert den String in ein Array von einzelnen Zeichen	`s.toCharArray() liefert {'l', 'o', 'n', 'g', ' ', 'b', 'o', 'o', 'k'}`
`split(`*`begrenzer`*`)`	Separiert den String anhand des gegebenen Begrenzers in ein Array von Teilstrings	`s.split(" ") liefert {"long", "book"} s.split("o") liefert {"l", "ng b", "", "k"}`
`String.join (`*`begrenzer, array`*`)`	Setzt die Elemente des Arrays zu einem String zusammen	`String[] arr = {"a","b","c"}; String.join("-", arr) liefert "a-b-c"`

Mehtoden	**Beschreibung**
`(List.)add(value)`	Fügt den gegebenen Wert am Ende der Liste an
`add(index, value)`	Fügt den gegebenen Wert in der Liste vor dem gegebenen Index ein
`clear()`	Entfernt alle Elemente
`contains(value)`	**liefert true, wenn das Element in der Liste ist - Elemente suchen**
`get(index)`	Liefert den Wert an der gegebenen Indexposition
`indexOf(value)`	Liefert den kleinsten Index, an dem der gegebene Wert in der Liste vorkommt (oder -1, wenn nicht gefunden)
`lastIndexOf(value)`	Liefert den größten Index, an dem der gegebene Wert in der Liste vorkommt (oder -1, wenn nicht gefunden)
`remove(index)`	Entfernt das Element an der gegebenen Indexposition und liefert es zurück. Nachfolgende Elemente rücken auf.
`set(index, value)`	Ersetzt das Element an der gegebenen Indexposition
`size()`	Liefert die aktuelle Anzahl der Elemente in der Liste

L.7.2.2 NullPointerException

Wenn Sie dennoch versuchen, eine Methode via null-Referenz aufzurufen, wird eine NullPointerException erzeugt.

L.7.2.3 Zweiphasen-Initialisierung von Objekt-Arrays

```
Point[] coords = new Point[3];           // phase 1
for (int i = 0; i < coords.length; i++) {
coords[i] = new Point(0, 0);             // phase 2
```

Methode	Bedeutung || **`Klasse Array:`** `import java.util.Array`
binarySearch(array, wert)	Index von wert zurück (oder -1) & Array muss sortiert sein
equals(array1, array2)	true, wenn Arrays gleiche Elemente in gleicher Reihenfolge
fill(array, wert)	Setzt jedes Element im Array auf den gegebenen Wert
sort(array)	Sortiert Elemente von Array in aufsteigender Reihenfolge
toString(array)	Liefert eine Zeichenkette für die Ausgabe
compare(array1, array2)	Lexikografischer Vergleich

```
if (numbers.indexOf(<value>) != numbers.lastIndexOf(<value>))
```
zweimal in der schleife

```
areAnagrams("tonne", "noten") liefert true
public static boolean areAnagrams(String a, String b){

char[] zeichena= a.toCharArray();
char[] zeichenb= b.toCharArray();
Arrays.sort(zeichena);
Arrays.sort(zeichenb);

boolean ergebnis= Arrays.equals(zeichena, zeichenb);

return ergebnis;
}
```

```
import java.io.*;
import java.util.Scanner;
public class Diff {
   public static void main(String[] args) {
      diff("text1.txt", "text2.txt");
   }
   public static void diff(String s1, String s2) {
      Scanner text1 = null;
      Scanner text2 = null;
      String text1Str = null;
      String text2Str = null;
      String error = null;
      int counter = 1;

      try {
         text1 = new Scanner(new File(s1));
         text2 = new Scanner(new File(s2));
      } catch (FileNotFoundException e) {
         if (text1 == null) {
            System.out.println("Kann Datei nicht finden: " + s1);
         } else {
            System.out.println("Kann Datei nicht finden: " + s2);
         }//System.exit(-1); darf man nicht benutzen, da Graja damit Probleme hat
:/
      }// wegen Graja muss ich deshalb checken ob beide Dateien existieren//
damit der nächste Part anderenfalls nicht ausgeführt wird
      if (text1 != null && text2 != null) {
         while (true) {
            if (!(text1.hasNextLine() || text2.hasNextLine())) {
               break;
            }

            if (text1.hasNextLine()) {
               text1Str = text1.nextLine();
            } else {
               text1Str = "";
            }

            if (text2.hasNextLine()) {
               text2Str = text2.nextLine();
            } else {
               text2Str = "";
            }

            if (!(text1Str.equals(text2Str))) {
               if (error != null) {
                  error = error + "Zeile " + counter + ":\n< " + text1Str + "\n> " +
text2Str + "\n\n";
               } else {
                  error = "Zeile " + counter + ":\n< " + text1Str + "\n> " + text2Str +
"\n\n";
               }
            }
            counter++;
         }
         if (error != null) {
            System.out.print("Unterschiede gefunden\n" + error);
         } else {
            System.out.print("Keine Unterschiede gefunden\n");
         }
         text1.close();
         text2.close();
```

```
import java.io.*;
import java.util.*;

public static class DoubleSpace {
   public static void main(String[] args) throws FileNotFoundException {
      Scanner input= new Scanner(new File("…"));
      PrintStream output= new PrintStream(new File("…"));

      doubleSpace(input, output);
      output.close();
      input.close();

   }

   public static void doubleSpace(String file1, String file2) throws
FileNotFoundException {

      Scanner input = null;
      PrintStream output = null;
      try {
         input = new Scanner(new File(file1));
         output = new PrintStream(new File(file2));

         String outputS = read(input);
         output.print(outputS + "\n");

         while (input.hasNextLine()) {
            outputS = "\n" + read(input);
            output.println(outputS);
         }

      } catch (FileNotFoundException e) {
         System.out.println("Datei nicht gefunden");
         System.out.println(e.getMessage(test));
         System.exit(1);
      }

      while (input.hasNextLine()) {
         String line = input.nextLine();

         output.println("");
         output.println(line);
      }
      input.close();
      output.close();
   }
```

```
import java.util.ArrayList;
public class Lb {
    public static void main(String[] args) {
        String s = "Monat/Tag/Jahr";
        String[] arrS = s.split("/");
        s = arrS[0];
        for (int i = 1; i < arrS.length; i++) {
            s = s + "." + arrS[i];
        }
    }
}
```

```
ng read(Scanner s) {
ine()) {
```

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	08.12.2020

L.8.3-L.8.4 (einschl.) L.9.1 (einschl.)

L.8 Deklaration eines mehrdimensionalen Arrays

Rechteckiges Array

<type> [] [] <name> = new <type> [<length>] [length];

Jagged Array:

Ein Array, dessen Elemente ungleich große Arrays sind.

```
public static void print(double[][] grid) {
  for (int i=0; i<grid.length; i++)  {
    for (int j=0; j<grid[i].length; j++) {
      System.out.print(grid[i][j] + " ");
    }
  System.out.println();
}
```

```
for (String s : array) {
sum += s.length();
}
```

L.8.1 ArrayList

ArrayList<String> list = new ArrayList<String>();

In ArrayList ohne Typparameter <E> kann man beliebige Objekte speichern.

Wrapper-Klassen

ArrayList<Integer> list = new ArrayList<Integer>();

list.add(13); `//autoboxing: liste.add(Integer.valueOf(15));`

Int i= liste.get(0); `//autounboxing: liste.get().intValue();`

Primitiver Typ	Wrapper-Klasse
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

L.8.2 Collections

(Data) Collection / Datensammlung:

Behälter zur Aufnahme von Daten.

Die in der Collection gespeicherten Objekte heißen Elemente.

Methode	Beschreibung
`add(`*value*`)`	Wert hinzufügen
`addAll(`*collection*`)`	Alle Elemente einer als Parameter gegebenen Collection zu dieser Collection hinzufügen
`remove(`*value*`)`	Entfernt den Wert (nur das erste Vorkommen) aus der Collection
`clear()`	Alle Elemente entfernen
`contains(`*value*`)`	liefert `true`, wenn der gegebene Wert enthalten ist
`containsAll(`*collection*`)`	`true`, wenn diese Collection alle Elemente der als Parameter gegebenen Collection enthält
`isEmpty()`	`true`, wenn diese Collection keine Elemente enthält
`removeAll(`*collection*`)`	Entfernt aus dieser Collection alle Werte der als Parameter gegebenen Collection.
`retainAll(`*collection*`)`	Entfernt aus dieser Collection alle Werte, die nicht in der als Parameter gegebenen Collection enthalten sind.
`size()`	Liefert die Anzahl der Elemente
`toArray()`	Liefert ein Array der Elemente
`iterator()`	Liefert ein besonderes Objekt für den Durchlauf durch alle Elemente der Collection.

L.8.3 Mengen (Sets)

TreeSet (Strings: alphabetisch)

HashSet (keine Duplikate)

Werden nicht in der Reihenfolge Saved

L.8.4 Maps

Map (Abbildung):

eine Collection, die eine Menge von Schlüsseln mit einer Ansammlung von Werten assoziiert.

(keine Schlüsselduplikate)

Ein Schlüssel =genau einen Wert.

Eine Map basiert aus zwei Collections

Map<String, Double> areaMap
= new HashMap<String, Double>();

Eine **TreeMap** speichert Schlüssel in der natürlichen Sortierung (Strings: alphabetisch).

for (String key : areaMap.keySet()) {
System.out.println(key + " => " + areaMap.get(key)); }

Methode	Beschreibung
`clear()`	Entfernt alle Schlüssel und Werte
`containsKey(`*key*`)`	liefert `true`, wenn der gegebene Schlüssel in der Map existiert
`containsValue(`*value*`)`	liefert `true`, wenn der gegebene Wert in der Map existiert
`get(`*key*`)`	Liefert den Wert, der zum gegebenen Schlüssel gehört (`null`, falls nicht gefunden)
`isEmpty()`	liefert `true`, wenn die Map keine Schlüssel oder Werte enthält
`keySet()`	Liefert eine Menge aller Schlüssel
`put(`*key*`, `*value*`)`	Ordnet dem gegebenen Schlüssel den gegebenen Wert zu
`putAll(`*map*`)`	Fügt alle Schlüssel-Wert-Paare aus der gegebenen Map in diese Map ein
`remove(`*key*`)`	Löscht den gegebenen Schlüssel und den zugehörigen Wert
`size()`	Liefert die Anzahl der Schlüssel-Wert-Paare in der Map
`values()`	Liefert eine Collection aller Werte

```java
import java.util.*;
import java.io.*;
public class WordLengths {

    public static void main(String[] args)
throws FileNotFoundException {
        WordLengths("text.txt");
    }
    public static void WordLengths(String file1)
throws FileNotFoundException {
        Scanner input = new Scanner(new File(file1));
        String word = null;
        int[] size = new int[81];

        while (input.hasNext()) {
            word = input.next();
            size[word.length()]++;
        }
        for ( int i =1; i< 81; i++) {
            while( size[i] != 0){
                System.out.print(i + ": " +
 size[i] +" ");

                for( int j =0; j<size[i];j++) {
                    System.out.print("*");
                }
                System.out.println("");
                break;
            ```
```

```java
import java.util.Scanner; import java.util.Arrays;
public  class Max {
    public static void main (String[] args) {
        boolean Systemexit = false;
        if (args.length != 3) {
        System.out.println("Bitte genau 3 Parameter angeben"
);
            Systemexit = true;
        }
        if (Systemexit != ture) {
            int number = 0;
            int max = Integer.MIN_VALUE;
            for(int i = 0; i<3; i++) {
                if (input.hasNextInt()){
                    number = input.nextInt();
                    if (number > max) {
                        max = number;
                    }
                } else {
                    Systemexit = true;
                    System.out.println("ungültiger Parameter
");
                }
            }
            if(Systemexit != false) {
                System.out.println("Maximum: " + max);
            }
}
```

```java
import java.util.*;
public class Pascal {
    public static void main (String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Wieviele Zeilen?");
        int n = console.nextInt();
        int[][] pascal = calPascal(n);
        drawPascal(pascal);
    }
    public static int[][] calPascal(int n) {
        int[][] dreieck = new int[n][];
        for (int i=0; i < n; i++) {
            dreieck[i]= new int[i+1];
            dreieck[i][0]= 1;
            dreieck[i][i]= 1; // Letzter Wert ist die 1
            for (int j = 1; j<i;j++) {
                dreieck[i][j]= dreieck[i-1][j-
1]+dreieck[i-1][j];
            }
        }
        return dreieck;
    }
    public static void drawPascal(int[][] pascal) {
        for (int i=0; i<pascal.length; i++)  {
            for (int j=0; j<pascal[i].length; j++) {

                System.out.print(pascal[i][j] + " ");
            }
            System.out.println();
        }
    }
```

```java
import java.util.*;
import java.io.*;
public class verdoppeln {
    public static void main (String [] args) {
        ArrayList<String> s= new ArrayList<String>();
        Collections.addAll(s, "Ich", "studiere", "in",
"Hannover");
        System.out.print(s.toString());
        verdoppeln(s);
    }
    public static void verdoppeln(ArrayList<String> s)
{
        int len = s.size();
        for(int i = 0;i < len;i++) {
            s.add("");
        }
        for ( int i = ((s.size()/2) -1 ); 0 <= i; i--
) {
            if(i >= 1) {
                s.set(2*i,s.get(i));
                s.set(2*i+1,s.get(i));
            } else {
                s.set(1,s.get(0));
            }
        }
        System.out.println(s.toString());
    }
}
```
```

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	16.12.2020

L.8.5 | L.9.2-L.9.3 (einschl.) | L.10.1

L.8.6 Iteratoren

Geheimnisprinzip: Verbergen von Daten/Informationen vor dem Zugriff von außen.

Obertyp Iterable = von Arraylist,Treeset[…]

```
Iterator<String> i= words.keySet().iterator();
while (i.hasNext()) {
   String word= i.next();
   System.out.println(word+": "+words.get(word));
}
```

L.9.1 Einführung in Rekursion

Programmiertechnik = eine Methode sich selbst aufruft

Rekursive Methoden:

• Einfacher Fall – Problem ohne Rekursion lösbar => & Abbruchbedingung

• Rekursiver Fall – Problem durch Rekursion lösbar => bis zur Abbruchbedingung

- Nachteil: hoher Speicherverbrauch, viel mehr (als Iteration)

Iteration: bei der man Wiederholung durch Schleifen umsetzt

L.9.2.1 Unendliche Rekursion - Abbruchkriterium vergessen

Es gibt keine Endlosrekursion, sondern wenn der stack frame die Grenze des verfügbaren RAM erreicht

L.9.3 Backtracking

Problemlösungstechnik, häufig implementiert durch Rekursion. Beim Backtracking werden alle Lösungen durch schrittweise Erweiterung von Teillösungen durchprobiert. Wenn eine Teillösung absehbar nicht zur Lösung führt, werden die letzten Schritte zurückgenommen und mit alternativen Teillösungen fortgefahren. Backtracking ist eine langsame, aber einfach zu implementierende Problemlösungstechnik auf der Grundlage von Versuch und Irrtum (trial and error).

Fibonacci-Folge:

```
public static int fib(int i) {
   if (i<=1) return 1;
   int a=1, b=1;
   for (int k=2; k<=i; k++) {
      int c= a+b;
      a=b;
      b=c;
   }
   return b;
}
```

L.9.4 Ineffizente Rekursion

Rekursion mit überlappenden Teilproblemen führt zu längeren Laufzeiten.

L.10.1.1 Umwandlung von Zahlen in Strings

Integer.toString(<zahl>) => geht auch für Double usw.

L.10.1.3 Umwandlung von Strings in Zahlen

int zahl= Integer.parseInt(s, (<zahl>)); || geht auch für Double usw.

String s= "true"; || boolean b= Boolean.parseBoolean(s);

Formatierung von Zahlen in Strings

```
int i= 78973;
String is= Integer.toString(i);
int ziffern= is.length();
String prefix= "00000000".substring(0, 8-ziffern);
is= prefix + is; System.out.println(is);
```

L.10.1.4 StringBuilder

Klasse für veränderbare Zeichenketten

- Anhängen ans Ende:
  `sb.append("mehr Text");`
- Einfügen:
  `sb.insert(pos, "einzufügender Text");`
- Ersetzen:
  `sb.setChar(pos, 'X');`
- Konvertierung in eine konstante Zeichenkette
  `String s = sb.toString();`
- Mehr Details finden Sie hier [2].

L.10.1.5 String.split

- Reguläre Ausdrücke erwartet String.split
  - ⇨ "[a-z]" beschreibt die kleinen Buchstaben von a-z
  - ⇨ "\d+" beschreibt Anzahl der Ziffern
  - ⇨ ".{5}" beschreibt Menge, die aus 5 Zeichen bestehen. Punkt bedeutet beliebige Zeichen
  - ⇨ "\\." Beschreibt Punkte

```
String str = "moin moin";
StringBuilder sb = new StringBuilder(str);
for (int i=0; i < sb.length(); i++) {
   if (sb.charAt(i) == 'o') {
      sb.setCharAt(i,'e');
   }
}
str = sb.toString();
```

```
    public static void abfrage() {
        Scanner eingabe = new Scanner(System.in);
        TreeMap<Integer, TreeMap<Double, Integer>> map = new TreeMap<>();
        while (true) {
            System.out.print("Ihre Wahl? ");
            int key = eingabe.nextInt();
            if (key == 0) {
                System.out.print("Statistik (in der obigen Sortierung)\n\n");
                break;
            }
            System.out.print("Zu welchem Preis? ");
            double preis = eingabe.nextDouble();
            System.out.print("Wie viele Einheiten? ");
            int einheiten = eingabe.nextInt();
            System.out.println("");
            if (!(map.containsKey(key))) {
                TreeMap<Double, Integer> map2 = new TreeMap<>();
                map.put(key, map2);
                map.get(key).put(preis,einheiten);
            } else {
                if(!(map.get(key).containsKey(preis))) {
                    map.get(key).put(preis,einheiten);
                } else {
                    map.get(key).put(preis,map.get(key).get(preis)+einheiten);
                }
            }
        }
        draw(map);
    }
```

```
public static boolean subMapSchehat(HashMap<String,
String> map1, HashMap<String, String> map2) {
    if (map1.size() > map2.size()) {
        return false;
    }
    int num = 0;
    for (String n : map1.keySet()) {
        if (map2.containsKey(n)) {
            if (map1.get(n).equals(map2.get(n))) {
                num++;
            } else {
                return false;
            }
        }
    }
    return num == map1.size();
}
```

```
public static long fakultaet(int n) {
    if (n==0) return 1;
    return n * fakultaet(n-1); }
    Eingabe einer nicht-negativen Zahl

public static int readNonNegInt(){
    while (true) { // will break on successful input
        int number= readInt();
        if (number >= 0) return number;
        System.out.println("Illegal input. Must be >=0.");
    }
}
public static int readInt() {
    while (true) {         // will break on successful inpu
t
        try {
            System.out.print("Your number: ");
            return Integer.parseInt(console.next());
        } catch(NumberFormatException e){
            System.out.println("Not a number.");
        }
```

PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	03.01.2021

L.8.6 L.9.4-L.9.5 (einschl.)

Aufzählungstypen enum

name()	liefert den deklarierten Attributnamen als String `String name = Spielkarte.HERZ.name(); // "HERZ"`
ordinal()	liefert die Position einer Enum innerhalb der Deklaration. `int idx = Spielkarte.HERZ.ordinal(); // 1`
valueOf(String)	Liefert Enum-Objekt zum Attributnamen. `Spielkarte k= Spielkarte.valueOf("HERZ"); // Spielkarte.HERZ`
values()	Liefert ein Array aller Enum-Objekte. `for (Spielkarte k : Spielkarte.values()) …`

```
public enum Spielkarte {
   KARO,
   HERZ,
   PIK,
   KREUZ //kann optional mit einem ; enden
}
```

java.util.Formatter;

Ausgabeformatierung

- Ein `Formatter`-Objekt arbeitet (wie ein `Scanner`) Locale-spezifisch.
- Beispiel:

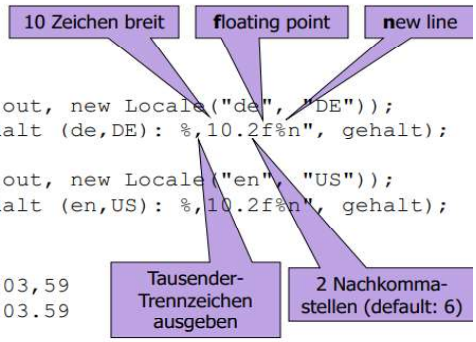
```
double gehalt= 1203.59;
Formatter formatter;

formatter = new Formatter(System.out, new Locale("de", "DE"));
formatter.format("Monatliches Gehalt (de,DE): %,10.2f%n", gehalt);

formatter = new Formatter(System.out, new Locale("en", "US"));
formatter.format("Monatliches Gehalt (en,US): %,10.2f%n", gehalt);
```

- Ausgabe:

```
Monatliches Gehalt (de,DE):    1.203,59
Monatliches Gehalt (en,US):    1,203.59
```



Formatangaben

b	Boolescher Wert
c	Einzelnes Zeichen
d	Ganzzahl in Dezimaldarstellung
o	Ganzzahl in Oktaldarstellung
x	Ganzzahl in Hexadezimaldarstellung
X	Ganzzahl in Hexadezimaldarstellung, mit großer
f	Fließkommazahl
e	Fließkommazahl mit Exponent
E	Fließkommazahl mit Exponent, mit großem "E"
g	Fließkommazahl in gemischter Schreibweise
G	Dito, ggf. mit großem "E"
t	Präfix für Datums-/Zeitangaben
s	Strings und andere Objekte
n	Zeilenumbruch

Variable Parameterlisten – public static void printes(String … name) {}
Die Ausgabe macht daraus ein Array name.length

```
public class Kurse2 {
public static void main(String[] args) throws FileNotFoundException{
    Locale enUS= new Locale("en", "US");
    Locale deDE= new Locale("de", "DE");
    PrintStream output= new PrintStream(new File("Kurse.en.csv"));
    Scanner input= new Scanner(new File("Kurse.csv"));
    String line= input.nextLine(); // Erste Zeile überlesen
    output.println("date,call price,offer price");
    while (input.hasNextLine()) {
        line= input.nextLine();
        Scanner lineScan= new Scanner(line); // Zeile m. Tokenscanner a
barbeiten
        lineScan.useLocale(deDE).useDelimiter(";");
        output.print(convertGermanDateToUS(lineScan.next()));
        for (int i=1; i<=2; i++) {
            output.format(enUS, "%s%.2f", ",", lineScan.nextDouble());
        }
        output.println();
        lineScan.close();
    }
    input.close();
    output.close();
    }
    public static String convertGermanDateToUS(String date){
        String[] arr= date.split("\\.");
        String us= arr[1] + "/" + arr[0] + "/" + arr[2];
        return us;
    }
}
```

```java
public class Raten {
    public static void main(String[] args) {
        int wert= -1;
        long start, stop;
        final int MAX= 1000000000;

        System.out.println("Brute Force Version:");
        start= System.currentTimeMillis();
        wert= ratenIterativ(MAX);
        stop= System.currentTimeMillis();
        System.out.println("Zahlenkombination ist: " + wert);
        System.out.println("Dauer: " + (stop-start)/1000.0 + "s");
        System.out.println("Anfragen: " + Ratespiel.anfragen());
        System.out.println();

        System.out.println("Rekursive Version:");
        start= System.currentTimeMillis();
        wert= ratenRekursiv(0, MAX);
        stop= System.currentTimeMillis();
        System.out.println("Zahlenkombination ist: " + wert);
        System.out.println("Dauer: " + (stop-start)/1000.0 + "s");
        System.out.println("Anfragen: " + Ratespiel.anfragen());
    }

    public static int ratenIterativ(int max) {
        for (int i=0; i<max; i++) {
            if (Ratespiel.rate(i)) {
                return i;
            }
        }
        return -1;
    }

    public static int ratenRekursiv(int min, int max) {
        if (min > max) {
            return -1;
        }
        int vermutung= (min+max)/2;
        if (Ratespiel.istGroesserAls(vermutung)) {
            return ratenRekursiv(vermutung+1, max);
        }
        if (Ratespiel.rate(vermutung)) {
            return vermutung;
        }
        return ratenRekursiv(min, vermutung-1);
    }
}
```

```java
public class Vorhang {
    public static void main(String[] args) {
        vorhang(4, "|----");
    }
    public static void vorhang(int tiefe, String pr
efix) {
        if (tiefe == 1) {
            System.out.println(prefix + "O");
        } else {
            vorhang(tiefe-1, prefix+"---");
            System.out.println(prefix + "O");
            vorhang(tiefe-1, prefix+"---");
        }
    }
}
```