Skript Programmieren

1. Die Basics

1. FOR-Schleife

Die for-Schleife ist eine Struktur, die dazu dient, eine oder mehrere Anweisungen solange wiederholt auszuführen, wie eine Bedingung wahr `(true)` ist.

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

2. WHILE-Schleife

Die while-Schleife dient dazu, eine oder mehrere Anweisungen so lange auszuführen, wie eine Bedingung wahr (`true`) ist.

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

3. DO-WHILE-Schleife

Die do-while-Schleife ist eine Struktur, in der Anweisungen wiederholt, mindestens jedoch ein Mal ausgeführt werden. Die Anzahl der Ausführungen ist abhängig von der Prüfung einer Abbruchbedingung.

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

4. Ternary

Der ternäre Operator kann eine if-else-Verzweigung ersetzen und weist meist einer Variablen einen Wert in Abhängigkeit vom Ergebnis einer Bedingungsprüfung zu.

Variable = bedingung ? wert1 : wert2;

5. IF-ELSE

Die if-Verzweigung dient dazu, eine oder mehrere Anweisungen auszuführen, je nachdem ob eine Bedingung wahr `(true)` oder unwahr `(false)` ist.

```
int i = new Random().nextInt(11);
if(i < 6) {
	System.out.println("i ist kleiner 6: " + i);
} else {
	System.out.println("i ist nicht kleiner 6: " + i);
}
```

6. SWITCH-CASE

Mit Hilfe einer switch-case-Verzweigung können, ähnlich wie in einer if-else-Verzweigung, Werte überprüft und selektiv Anweisungen ausgeführt werden.

```
int i = 2;
switch(i){
	case 0:
		System.out.println("i ist null");
		break;
	case 1:
		System.out.println("i ist eins");
		break;
	case 2:
		System.out.println("i ist zwei");
		break;
	default:
		System.out.println("i liegt nicht zwischen null und drei");
		break;
}
```

7. TRY-CATCH

Try-catch-Anweisungen dienen in Zusammenarbeit mit Exceptions hauptsächlich dem Abfangen von Programmfehlern.

```
int[] array = new int[5];
try {
    int x = int[10];
} catch (ArrayIndexOutOfBoundsException e) {
    e.printStackTrace();
} finally {
    System.out.println(“Fertig”);
}
```

TRY: zu prüfender Code
CATCH: bei einer Exception auszuführender Code
FINALLY: immer auszuführender Code, außer bei System.exit([Zahl]);

8. INSTANCEOF

Der instanceOf-Operator überprüft, ob eine Variable zu einer Klasse gehört. Hierbei gibt es drei mögliche Ergebnisse:

```
class A{};
class B extend A{};
class C extend A{};

new C() instanceOf B;   → kompiliert nicht, da nicht verwandt
new C() instanceOf A;   → true
new A() instanceOf B;   → false
```

9. Modulo

Restrechnung:
`4 % 2`

Prüfung, ob eine Zahl gerade ist:
`4 % 2 == 0` → `true oder false`

2. Primitive Datentypen

Name	Wertebereich
byte	-2^7 bis $2^7 - 1$ (-128...127) (Ganzzahlen)
short	-2^{15} bis $2^{15} - 1$ (Ganzzahlen)
char	Ein Zeichen
int	Hohe Zahlen (Ganzzahlen)
long	Mega hohe Zahlen (Ganzzahlen)
float	Hohe **Kommazahlen**
double	Verdammt hohe **Kommazahlen**
boolean	true/false

Wichtiger Fakt:
MAX_VALUE ist immer eine ungerade Zahl
MIN_VALUE ist eine gerade Zahl

3. ASCII-Tabelle

Dezimal	Hex	Zeichen	Dezimal	Hex	Zeichen
32	20	[Leerzeichen]	97	61	a
65	41	A	98	62	b
66	42	B	99	63	c
67	43	C	100	64	d
68	44	D	101	65	e
69	45	E	102	66	f
70	46	F	103	67	g
71	47	G	104	68	h
72	48	H	105	69	i
73	49	I	106	6A	j
74	4A	J	107	6B	k
75	4B	K	108	6C	l
76	4C	L	109	6D	m
77	4D	M	110	6E	n
78	4E	N	111	6F	o
79	4F	O	112	70	p
80	50	P	113	71	q
81	51	Q	114	72	r
82	52	R	115	73	s
83	53	S	116	74	t
84	54	T	117	75	u
85	55	U	118	76	v
86	56	V	119	77	w
87	57	W	120	78	x
88	58	X	121	79	y
89	59	Y	122	7A	z
90	5A	Z			

Chars können als int dargestellt werden!

```
char c = 65;
```

4. Wrapper-Klassen

10. Integer

Attribute & Konstanten

`static int`	**BYTES** The number of bytes used to represent a `int` value in two's complement binary form.
`static int`	**MAX_VALUE** A constant holding the maximum value an `int` can have, $2^{31}-1$.
`static int`	**MIN_VALUE** A constant holding the minimum value an `int` can have, -2^{31}.
`static int`	**SIZE** The number of bits used to represent an `int` value in two's complement binary form.
`static Class<Integer>`	**TYPE** The `Class` instance representing the primitive type `int`.

Konstruktoren

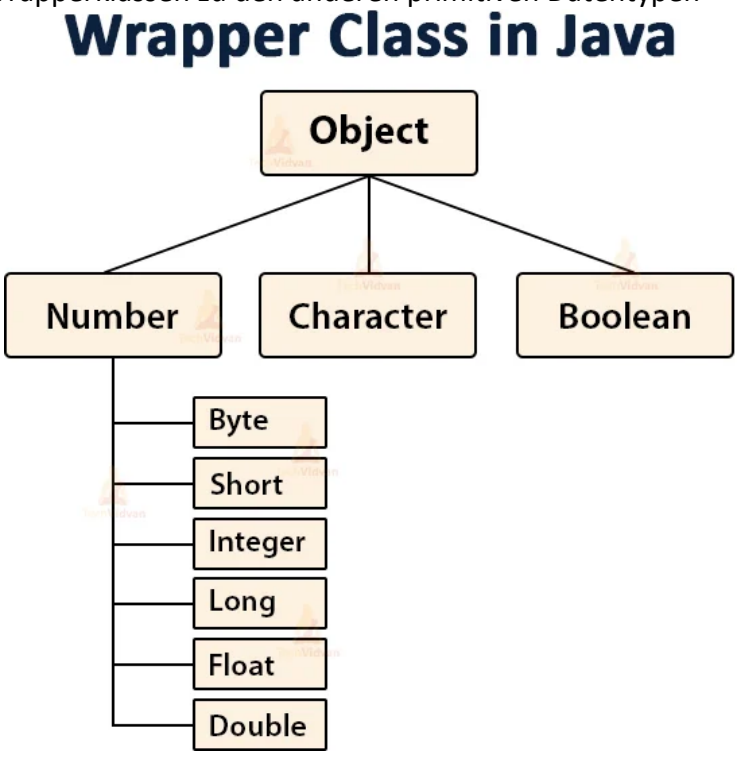
`Integer``(int value)`
Constructs a newly allocated `Integer` object that represents the specified `int` value.

`Integer(String s)`
Constructs a newly allocated `Integer` object that represents the `int` value indicated by the `String` parameter.

Methoden (die wichtigsten)

`static int parseInt(String s)`	Konvertiert einen String s zu einem int
`String toString()`	Konvertiert den Integer zu einem String

Analog dazu die Wrapperklassen zu den anderen primitiven Datentypen



5. Codes

1. BMI

```
public class BMI {

    private double weight, height, bmi;

    public BMI(double weight, double height) {
        this.weight = weight;
        this.height = height;
        bmi = weight / (height * height);
    }
    public double getBmi() {
        return bmi;
    }
    public double getWeight() {
        return weight;
    }
    public double getHeight() {
        return height;
    }
    public void setWeight(double weight) {
        this.weight = weight;
        bmi = weight / (height * height);
    }
    public void setHeight(double height) {
        this.height = height;
        bmi = weight / (height * height);
    }
    public static void main(String[] args) {
        BMI mauri = new BMI(80, 1.20);
        System.out.println(mauri.getBmi());
        mauri.setWeight(90);
        System.out.println(mauri.getBmi());
    }

}
```

2. Fakultät - rekursiv

```
// REKURSIV @thomas
static int fakultaet(int obereGrenze) {
    if(obereGrenze <= 1) {
        return 1;
    } else {
        return obereGrenze * fakultaet(obereGrenze - 1);
    }
}
```

3. Fakultät – nicht rekursiv

```
// NICHT REKURSIV @thomas
static int fakultaetNichtRekursiv(int obereGrenze) {
    int ergebnis = 1;
    while(obereGrenze > 1) {
        ergebnis = ergebnis * obereGrenze;
        obereGrenze--;
    }
    return ergebnis;
}
```

4. Normale Summenfunktion (für Thomas)

```
static int summe(int a, int b) {
    int ergebnis = a + b;
    return ergebnis;
}
```

5. Summenfunktion für unbestimmte Anzahl an Summanden

```
 static int summe(int ... a) {
    int summe = 0;
    for(int x : a) {
        summe += x;
    }
    return summe;
   }
```

6. Klasse mit Instanzvariablen und Getter/Setter

```
public class Cat {

   private String name;
   private int alter;
   private int weight;

   public Cat(String name, int age, int weight) {
       this.name = name;
       this.age = age;
       this.weight = weight;
   }
   public Cat() {
   }
   public void sayMeow() {
       System.out.println("Miau!");
   }
   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }
   public int getAge() {
       return age;
   }
   public void setAge(int age) {
       this.age = age;
   }
   public int getWeight() {
       return weight;
   }
   public void setWeight(int weight) {
       this.weight = weight;
   }
}
```

7. SUMME – rekursiv

```
public class SummeRekursiv {
   public static void main(String[] args) {
      int sum = sum(4);
   }
   public static int sum(int a){
      int summe = 0;
      summe += a;
      if (a>1){
         summe += sum(a-1);
      }
      return summe;
      // Oder One-Liner:
      return a + ((a-1 == 1) ? 1 : sum(a-1));
   }
}
```

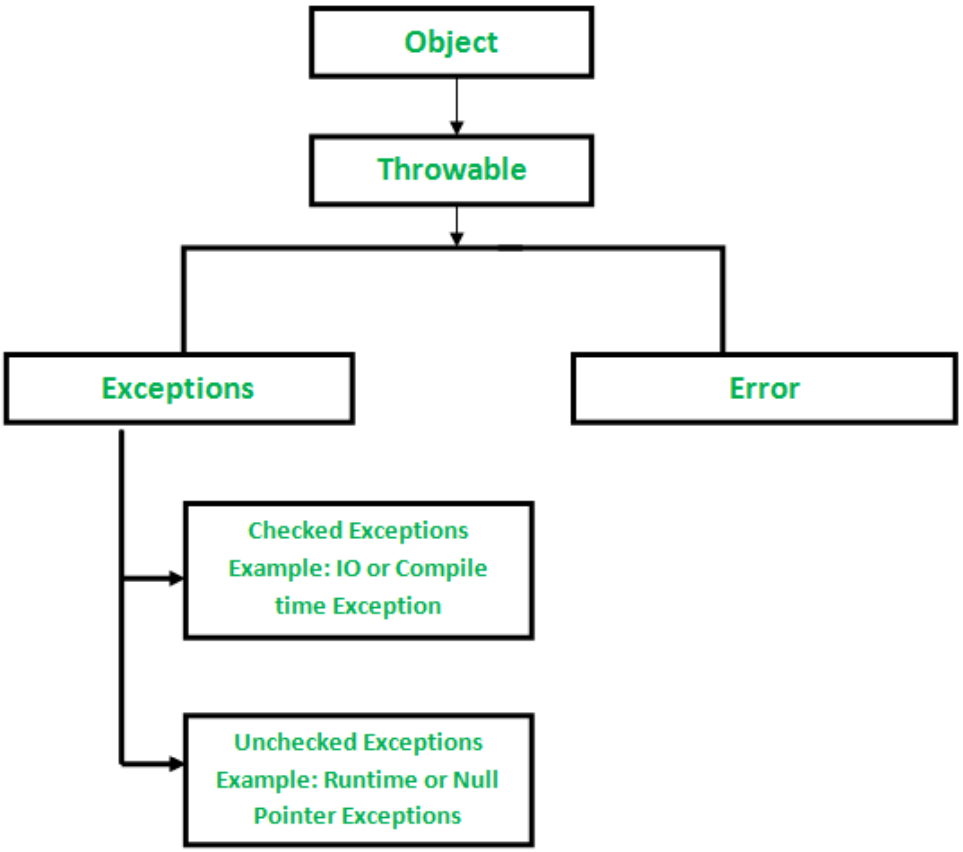
8. Zahlen aus einem String entfernen

```
public static String stringBereinigen(String eingabe) {
        StringBuilder sb = new StringBuilder();
        for(char c : eingabe.toCharArray()) {
            if(c >= 48 && c <= 57) continue;
            /* Oder:
            if(c == '1') continue;
            if(c == '2') continue;
             */
            sb.append(c);
        }
        return sb.toString();
  }
```

// Der Dezimalzahlenkorridor 48-57 repräsentiert die Unicodezeichen ‘0‘ bis ‘9‘

6. Exceptions

1. Übersicht



2. ClassCastException

Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
So, for example, when one tries to cast an Integer to a String, String is not a subclass of Integer, so a ClassCastException will be thrown.

```
Object i = Integer.valueOf(42);
String s = (String) i;            // ClassCastException thrown here
```

Vll bessere Erklärung:

Let's think of a collection of classes.

```
class A {...}
class B extends A {...}
class C extends A {...}
```

1. You can cast any of these things to *Object*, because all Java classes inherit from *Object*.
2. You can cast either B or C to A, because they're both "kinds of" A
3. You can cast a reference to an A object to B *only if* the real object is a B.
4. You can't cast a B to a C even though they're both A's.

3. NullPointerException

Zugriff auf ein Objekt, das *null* ist.

```
String str = null;
str.toLowerCase();            // NullPointerException
```

4. ArrayIndexOutOfBoundsException

Zugriff auf einen nicht existenten Index bei einem Array.

```
int[] array = new int[5];
array[10]                     // ArrayIndexOutOfBoundsException
```

5. ArithmeticException

Mathematischer Fehler, wie z.B. Dividieren durch 0.

7. Probeklausurlösungen

```
String myString = "Do";
myString.concat(" not").concat(" ever ");          --> Kompiliert nicht, wegen 4 und 6
myString.substring(4,6).toUpperCase();
System.out.println(myString);
```

```
String myString = "Do";
myString.concat(" not").concat(" ever ");
myString.toUpperCase();
myString + "Dowa Didi damm didi doo".substring(0,3);        -->Kompiliert, ergebnis "Do Drugs"
myString += " drink and hugs!".replace("ink and h", "");
System.out.println(myString)
```

```
StringBuilder mySB = new StringBuilder("012");
mySB.append((new String("89"))).replace(3,7 , new String("567")).append("2");
System.out.println(mySB);                                    -->Kompiliert, ergebnis "0125672"
```

```
String myString = "hi";
StringBuilder mySB = new StringBuilder();
mySB.append("hi");
        int counter = 0;
if (myString == new String("hi"))
        ++counter;
        System.out.print(counter++ + " ");                   -->Kompiliet, ergebnis "0 0 true 1 1"
if (mySB.equals(new StringBuilder(mySB)))
        System.out.print(counter++ == 0 | ++counter < 3);
        System.out.print(--counter*2 + " ");
if (mySB.toString().equals((new StringBuilder(mySB)).toString())){
        System.out.print(counter++ == 0 || --counter < 3);
        System.out.print(" " + counter++ + " ");
```

```
String myString = "42.0";
switch ((new Double(Double.parseDouble(myString)).intValue())%5){
        default: System.out.println("Falsche Reihenfolge");
        case 2: System.out.println("Zwei");   -->Kompiliert, ergebnis "Zwei Eins Null"unterein
        case 1: System.out.println("Eins");
        case 0: System.out.println("Null");
```

```
String arr[] = {"A", "Bc", "Cde", "Dfgh"};
for( int i = 0; i< arr[2].length(); i++)          -->Kompiliert, ergebnis "A Bc Cde" untereinander
System.out.println(arr[i]);
```

```
String arr[] = {"A", "Bc", "Cde", "Dfgh"};
for( int i = 0; i< arr.length(); i++)  -->Kompiliert nicht weil Klammer zu viel
System.out.println(arr[i]);


String arr[] = {"A", "Bc", "Cde", "Dfgh"};
int b = arr.length;
for( int i = 0; i<= b ; i++)                    -->Kompiliert
System.out.println(arr[i]);


String arr[] = {"A", "Bc", "Cde", "Dfgh"};
int b = arr.length;
for( int i = 0; i<= b ; i++)                              -->Kompiliert aber arrayindexoutofbounce
System.out.println(arr[i]);


String arr[] = new String[3];
try{
for( int i = 0; i< 3 ; i++)
System.out.println(arr[i].length());}        --> Kompiliert 4 aber nullPointerException
finally{
System.out.println(4);
}


int arr[] = new int[3];
int i ;
try{
for(i = 0; i< 3 ; i++)
System.out.println(42/arr[i]);}
catch(RuntimeException e){       --> Kompiliert nicht, weil i kein default wert hat
System.out.print(i);
}
finally{
System.out.println(2);
}


StringBuilder sb = new StringBuilder();
char myChar = 65;
int myCounter = 4;
for(; ;){
while(true){
sb.append(myChar++);              -->Kompiliert zu "A AB ABC ABCD ABCDE" untereinander
if(sb.toString().contains("A") )
break; }
System.out.println(sb) ;
```

```
if((myCounter--) == 0)
break;

int [] arr1 = {97,98};
int [] arr2 = {65,66};
int [][] arr3= { arr1, arr2};
for (int [] tmp1: arr3){
for (int tmp2: tmp1){                          -->Kompiliert zu "false false false true"
if ((int)tmp2 > tmp1.length )
System.out.print((tmp2 == new Character('B')) + " " );
}
}
```