JAVA-Zusammenfassung

OOP1

Basiscs

```java
//Instanzvariablen
int a = 3;
String b = "miau";
int[] array = {1,2,3,6};
//Array mit Länge init
double[] array2 = new double[6];
//Methode
addition(a);
```

Methoden

```java
//static da keine Instanz erstellt
//& direkter Aufruf aus main()
1 usage
public static void addition(int a) {
    System.out.println(a + 4);
}
```

Klassen

```java
//Doppelrolle:
//>Nuterdef. Datentyp
//>Erzeugung Objekte

//Basisklasse zur Erzeugung
//von Items mit Konstruktor
10 usages  3 inheritors
abstract public class Item {
    //Beschreibung Nicht veränderbar
    8 usages
    final String description;

    //Konstruktor: Erählt description
    //aus konkreter Instanzierung
    3 usages
    public Item(String description) {
        this.description = description;
    }
    //this. für Erstellung konkretes
    //Objekt aus Instanz
    7 usages  3 implementations
    public abstract double getPrice();
    2 usages
    protected String getDescription() {
        return description;
    }
```

Vererbung

```java
//Subklasse erbt von Item
2 usages
public class ProductItem extends Item {
    2 usages
    int amount;
    2 usages
    double pricePerUnit;

public ProductItem(String description,
                   double pricePerUnit,
                   int amount) {
    super(description);
    this.pricePerUnit = pricePerUnit;
    this.amount = amount;
}
//zusätzlich zu den zwei "neuen"
//Var. MUSS mit super auf Besch-
//reibung aus Basisklasse instan-
//ziert werden.

//mit getter können aus anderen
//Klassen easy Werte abgerufen
//werden. @Override heisst,
//methode überschreibt gleiche
// Methode in Basisklasse.
7 usages
@Override
public double getPrice() {
    return pricePerUnit * amount;
}

//Initialisierung konkretes Objekt
ProductItem television4k = new ProductItem
                ( description: "Samsung 4K",
                  pricePerUnit: 2600.00,
                  amount: 1);
```

Sichtbarkeit von Instanz- Variablen und Methoden

Keyword	Sichtbar für
public	Alle Klassen
protected	Package* und alle Sub-Klassen** (aus anderen Packages)
(keines)	Alle Klassen im selben Package*
private	Nur eigene Klasse

	Lokale Variablen (in Methoden, Blöcken, Konstruktoren)	Instanz-Variablen (und Klassen-Variablen)
Wert nach Deklaration	undefiniert	Default-Wert
Initialisierung	muss explizit geschehen	Default-Wert, danach explizit zuweisbar

Static

Variablen

Static	Instanz
Pro Klasse	Pro Objekt
Zugriff per Klassenname	Zugriff per Referenz

Methoden

Static	Instanz
Aufruf per Klassenname	Aufruf per Referenz
Kein this	this definiert
Statische Bindung	Dynamic Dispatch

Interfaces

```java
//Ähnlich zu Vererbung.
//Schnittstellen definieren
// eine Sammlung von Methoden-
// signaturen, die von einer
// Klasse implementiert
// werden können. Klasse
// mehrere Schnittstellen
// implementieren
8 usages  1 implementation
public interface Visualization {
    1 usage  1 implementation
    void drawRectangle(int left, in
    1 usage  1 implementation
    void drawCircle(int x, int y, i
    1 usage  1 implementation
    void drawText(int x, int y, Str
    1 usage  1 implementation
    void clear();
}

public class Circle implements Shape {
    //Diese Klasse muss nun alle Meth-
    //oden implentieren, welche im In-
    //terface vorausgesetzt werden.
Collections_
//Liste von Elementen, wie Array
ArrayList<String> data =
        new ArrayList<>();
data.add("Floyd"); //Neu
data.add( index: 3,  element: "Queen"); //Neu
data.size(); //get Length
data.get(2); //get Value at position
data.set(3, "HIHI"); //Pos. überschreiben
data.remove( index: 3); //Löschen index
data.remove( o: "Floyd");//Delete Objektref.
```

```java
//Map -> Abbildung Schlüssel Werte
Map<Integer, Double> numbers =
        new HashMap<>();
numbers.put(1, 2.3); //Neu
numbers.get(1); //get Pos where Key
numbers.size(); //Anzahl werte
numbers.remove( key: 1); //Lösche an Index
//Set -> Menge von Elementen, NO Duplikate
//Schnell, keine Reihenfolge
Set<String> objects = new HashSet<>();
//Objektverwaltung wie List
```

Loops

```java
// While loop example
int i = 1; // Initialize loop
System.out.println("While Loop:");
while (i <= 5) { // Loop condition
    System.out.println("Iteration " + i);
    i++; // Update loop control variable
}
// For loop example
System.out.println("\nFor Loop:");
for (int j = 1; j <= 5; j++) {
    System.out.println("Iteration " + j);
}
// Do-while loop example
System.out.println("\nDo-While Loop:");
int k = 1; // Initialize loop control variable
do {
    System.out.println("Iteration " + k);
    k++; // Update loop control variable
} while (k <= 5); // Loop condition
// Enhanced for loop (for-each loop) example
System.out.println("\nEnhanced For Loop:");
int[] numbers = {1, 2, 3, 4, 5};
for (int num : numbers) {
    // Iterate over elements of the array
    System.out.println("Number: " + num);
```

Stream_Predikate

`filter(Predicate)`
- Rauspicken gemäss Predicate-Funktionsobjekt/Lambda

`map(Function)`
- Projizieren gemäss Funktionsobjekt/Lambda

`mapToInt/mapToLong/mapToDouble(Function)`
- Projizieren auf int, long, double (primitiver Datentyp)

`sorted()`
- Sortieren, mit oder ohne Comparator

`distinct()`
- Unterschiedliche Elemente gemäss equals()

`limit(long n)`
- Erste n Elemente liefern, danach ignorieren

`skip(long n)`
- Erste n Elemente ignorieren, danach weiterliefern

Exception_Handling

```java
// NullPointerException example
try {
    String str = null;
    int length = str.length();
    System.out.println(length);
} catch (NullPointerException e) {
    System.out.println(e.getMessage());}
// ArithmeticException example
try {
    int result = 10 / 0;
    System.out.println(result);
} catch (ArithmeticException e) {
    System.out.println(e.getMessage());}
// IndexOutOfBoundsException example
try {
    int[] arr = {1, 2, 3};
    int element = arr[3];
    System.out.println(element);
} catch (IndexOutOfBoundsException e) {
    System.out.println(e.getMessage());}

class CustomException extends Exception {
    1 usage
    public CustomException(String message) {
        super(message);}}
public static void main(String[] args) {
    try {
        throwCustomException();
    } catch (CustomException e) {
        System.out.println(e.getMessage());}}
1 usage
public static void throwCustomException()
        throws CustomException {
    throw new CustomException("Exception");}}
```

Streams_and_Lambdas

```java
// Lambda: (Parameter) ->
// {Expression}
// Using streams to filter
// and print even numbers
numbers.stream()
  .filter(n -> n % 2 == 0)
  .forEach(System.out::println);
// Using streams to filter
// and print numbers > 10
numbers.stream()
  .filter(n -> n > 10)
  .forEach(System.out::println);
//Angenommen numbers ist Collection
//andere Zwischenoperationen
```

Vereinfachungsbeispiel

1. Einsatz anonymer Klassen

```
List<String> animals = new ArrayList<String>();
animals.add("Hamster");
animals.add("Guinea Pig");
animals.add("Root Vole");
animals.add("Rat");
animals.add("Groundhog");
```

```
animals.replaceAll(new UnaryOperator<String>() {
    @Override
    public String apply(String string) {
        return string.toUpperCase();
    }
});

for (String animal : animals) {
    System.out.println(animal);
}
```

2. Einsatz von Lambda-Ausdrücken

```
List<String> animals = new ArrayList<String>();
animals.add("Hamster");
animals.add("Guinea Pig");
animals.add("Root Vole");
animals.add("Rat");
animals.add("Groundhog");
```

```
animals.replaceAll(x -> x.toUpperCase());

animals.forEach(x -> System.out.println(x));
```

3. Einsatz von Methodenreferenzen

```
List<String> animals = new ArrayList<String>();
animals.add("Hamster");
animals.add("Guinea Pig");
animals.add("Root Vole");
animals.add("Rat");
animals.add("Groundhog");
```

```
animals.replaceAll(String::toUpperCase);

animals.forEach(System.out::println);
```