1. Was ist eine Klasse?

- Klassen sind Baupläne/Schablonen für Objekte (Instanzen)
- Sie bieten Konstruktoren zur Objekterzeugung an
- Von einer Klasse können mehrere Objekte erzeugt werden
- Sie spezifizieren ihr Verhalten über Methoden
- Bieten öffentliche Schnittstellen um den Zustand eines Objektes von außen zu können

2. Was ist ein Objekt im Sinne von OOP?

- Objekte (Instanzen) sind konkrete Realisierungsregeln von Klassen
- Haben einen konkreten Zustand (Instanzvariablen der Klasse)
- Jedes Objekt hat zur Laufzeit eine Repräsentation im Speicher
- Manipulation des Objektzustandes $\rightarrow$ über die Methoden der Klasse

3. Was ist die Object-Klasse?

- Alle Klassen erben von Object
- Alle selbst erstellte Klassen erben implizit auch von Object
- Object Methoden können mit eigener Implementierung überschrieben werden

4. Was ist der Konstruktor?

- Dient zur Objekterzeugung von Klassen
- werden bei der Erzeugung von Objekten einer Klasse aufgerufen
- Initialisieren den Objekt-Zustand
- NAme des Konstruktors $\hat{=}$ Klassennamen
- wird kein Konstruktor explizit definiert $\Rightarrow$ erstellt Compiler einen Default-Konstruktor leere ** Tabelle **

- Mehrere Konstruktoren mit unterschiedlicher Signatur möglich
- Verwendung von set-Methoden vermeiden
- Construktor-Chaining = in einem Konstruktor wird ein anderer Konstruktor aufgerufen:
    - einer der eigenen Klassen
    - einer der direkte Oberklasse

  → das muss an erster Stelle passieren

```
public class Vehicle{
    private Motor engine = null;
    private String name = null;
    public Vehicle(){
        this.name = new String(" ");
    }
    public Vehicle (Motor engine){
        this.Vehicle(); //an erster Stelle
        this.engine=engine;
    }
    public Vehicle (Motor engine, String name){
        this.name=name; //oder anderer Konstruktor möglich
        this.engine = engine;
    }
}
```

5. Objekterzeugung - Ablauf + Zusammenhang mit Vererbung (Erzeugung)

```
Vehicle car = new Vehicle ("BMW", null);
```

1. Referenzvariable car wird am Stack angelegt und mit null belegt
2. new-Operator von Vehicle wird aufgerufen. Auf dem JAva-Heap wird der Platz für ein Objekt der Klasse Vehicle reserviert ⇒ Objekt wird instanziiert.
3. Datenfleder des Objekts werden inititialisiert
4. Methodenblock des Konstruktors wird ausgeführt
5. new-Operator gibt die Refefinalize-renz auf das erzeugte Ibjekt zurück
6. Referenz wird in Varriable car gespeichert.

- In Variable wird Referenz auf das Objekt gespeichert

- Konstruktor-Chaining
- Dynamisches Binden ⟶ Bsp. mit Super & Subklasse

6. Garbagecollector

- Wenn auf dem Heap Speicher von Objekten liegen auf die nicht mehr referenziert wird ⇒ werden die automatisch freigegeben
- erkennt auch zyklisch regerenzierte Objekte die nicht mehr verwendet werden
- Memory Leaks weiterhin möglich
- finalize-Methode wird vor dem Zerstören des Objektes aufgerufen

7. Methodenaufruf

Aufruf einer Instanzmethode:

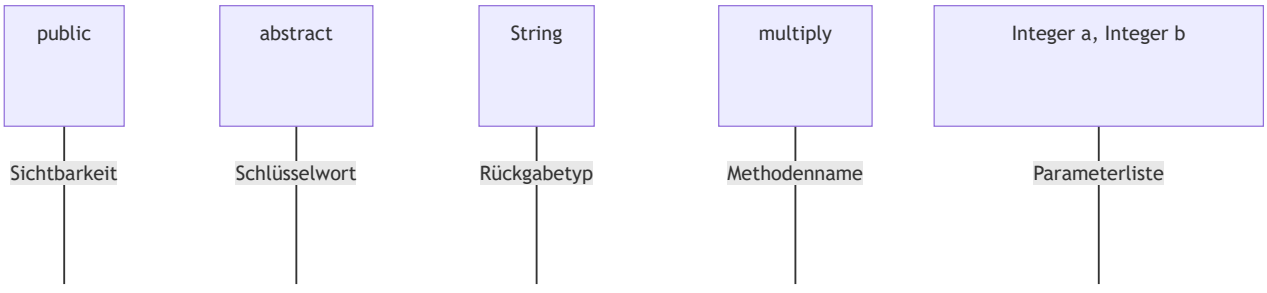
```
myObject.methodName(par1,par2,...);
```

Aufruf einer statischen Methode:

```
ClassName.methodName(par1,par2,...);
```

8. Methodensignatur

- definiert die formale Schnittstelle einer MEthode
- enthält Name der Funktion, Anzahl und Reihenfolge der kompartiblen Parameterdatentypen (Bei Java:nicht der Rückgabetyp)



In Java gehört der Rückgabetyp nicht zur Signatur

9. Parameterübergabe

- primitive Datentypen ⇒ als Kopie übergeben
- bei Objekten ⇒ Kopie der Referenz auf das Objekt übergeben (Copy by Value-Reference)

10. Overloading

- 2 oder meht Methoden haben exakt die gleichen Methodennamen aber unterschiedliche Parameterlisten
- Rückgabetyp wird icht berücksichtigt
- Mechanismus zeigt sich zur Compile-Time

```
public class Overloading{
    public Integer add(Integer a, Integer b){...}
    public Integer add(Integer a, Integer b, Integer c){...}
    public Float add(Integer a, Integer b, Float c){...}
}
```

11. Statische Methode

- Ist nicht mit einem Objekt verbunden
- Keine Instanz notwendig um Methoden aufzurufen ⇒ über Klassenname

```
public class StaticExample{
    public static Integer add(Integer a, Integer b){...}
    // int result = StaticExample.add(12,44);
}
```

12. Abstrakte Methoden

- legen nur Methodensignatur fest
- es gibt keine Implementierung (kein Methodenblock)
- erbt eine nicht abstrakte Klasse von einer abstrakten, müssen alle abstrakte Methoden implementiert werden

13. Abstrakte Klassen

- definieren auch einem Typ
- von abstrakten Klassen känen keine Instanzen erstellt werden
- können abstrakte Methoden beinhalten
- von einer abstrakte Klasse kann geerbt werden
- erbt nicht abstrakt von abstrakte Klasse $\Rightarrow$ müssen alle abstrakte Methoden implementiert werden
- Enthält eine Klasse eine abstrakte Methode $\Rightarrow$ muss die Klasse abstrakt sein

14. Primitive Datentypen

- sind keine Klasse
- für jeden primitiven Datentyp git es eine eigene Wrapperklasse
- automatische Konversion zwischen den beiden über Auto/Unboxing

Basistyp	Wrapperklasse
int	Integer
long	Long
float	Float
boolean	Boolean
byte	
short	
double	
char	

15. Vererbung

- Dient zur Erweiterung/Einschränkung einer bestehenden Klasse
- Expliziter Zugriff auf Methoden oder Attribute der direkten Oberklasse

- Zugriffsmöglickeit aber abhängig von Sichtbarkeit
- Explizieter Zugriff mit dem Schlüsselwort super
- Vererbung bei Klassen u interfaces möglich
- Vererbung erzwingt keine Subtypenbildung im Sinne von OOP
- nutzt Ähnlichkeiten im Source Code aus
- Untertypen haben ersetzbares Verhalten
- Instanzvariablen werden vererbt & können überdeckt werden
- Instanzmethoden werden vererbt & private Instanzmethoden können in Subklasse überdeckt werden
- Neue Varriablen können angelegt werden
- Neue Methoden können angelegt werden
- Keine Vererbung von Konstruktoren

```
class [SUBCLASS] extends [SUPERCLASS] {}
```