Unser erstes und letztes Programmierbeispiel

openHPI-Java-Team

Hasso-Plattner-Institut

Unser erstes Programmierbeispiel

```
1  class HelloPaco {
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Ausgabe:

```
Hallo Paco
```


```
1  class HelloPaco {
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Klassen

- Schlüsselwort `class`
- Baupläne für (mehr oder weniger abstrakte) Konstrukte aus der realen Welt
- Verfügen über Zustand (Attribute) und Verhalten (Methoden)
- Klassenbezeichner kann frei gewählt werden und wird groß (CamelCase) geschrieben
- Verwandte Konzepte: Vererbung, Abstrakte Klassen, Interfaces


```
1  class HelloPaco {
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Sichtbarkeiten

- Regeln Zugriff auf Elemente
- “Sichtbarkeit” - bezieht sich auf den Kontext (wo ist was sichtbar)
- `public` = uneingeschränkter Zugriff von überall
- main Methode muss immer `public` sein
- Alternativen für alle anderen Methoden: `protected`, `private`, default
- Attribute werden in der Regel als `private` deklariert
- Regelung der Lese- und Schreibrechte über Getter/Setter


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Statische Methoden und Attribute

- Element der Klasse nicht des Objekts
  - Keine Objekt-Instantiierung mit `new`
  - Aufruf direkt auf der Klasse: `Double.parseDouble("22.8")`
  - Attribute: gemeinsamer Wert für alle Objekte der Klasse

Schritt für Schritt (4/12)

```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Rückgabetypen:

- Datentyp des Werts mit dem die Methode antwortet
- `void` = kein Rückgabewert
- `main` immer `void`
- Konstruktoren geben immer ein Objekt ihrer Klasse zurück
- Andere Methoden können `void` oder beliebige primitive oder Objekt-Datentypen zurückgeben


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Methodenbezeichner:

- Individueller Name für Methode
- `main` ist reserviert und darf nur einmal pro Programm vorkommen
- `main`-Methode startet Programmfluss
- Bezeichner für andere Methoden können nahezu frei gewählt werden
  - Darf nicht mit Zahl beginnen, keine Sonderzeichen
  - Kleingeschrieben (camelCase)
  - Ausnahme: Konstruktor (wie Klassenname → CamelCase)


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Parameter:

- Variablen für Argumente, die von der Methode beim Aufruf erwartet werden
- Mittels des Bezeichners kann innerhalb der Methode auf den beim Methodenaufruf übergebenen Wert zugegriffen werden

Schritt für Schritt (7/12)

```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Array:

- Container fester Größe für Objekte (oder primitive) gleichen Datentyps
- Gekennzeichnet durch <Datentyp>[]
- `String[] args` = User-Eingabe beim Programmstart
  - z.B. Kommandozeilenargumente


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Ausgaben auf der Konsole

- `System.out.println(...)`
  - Überladen für alle primitiven Datentypen und String
  - Objektdatentypen werden mittels `toString()` als String ausgegeben
  - Gibt einzelne Zeile aus (mit Zeilenumbruch am Ende)
- `System.out.print(...)`
  - Gibt einzelne Zeichenkette aus (ohne Zeilenumbruch am Ende)


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Argumente:

- Übergeben einen Wert an den Parameter der aufgerufenen Methode
- Als Argument übergeben werden können:
  - Werte: `add(3, 4);` `println("Hallo Paco");`
  - Variablen: `add(x, y);`
  - Methodenaufrufe: `println(getName());`
    `println(paco.getName());`


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Anweisung

- Befehl an den Computer etwas auszuführen


```
1  class HelloPaco{
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Semikolon

- Beendet Anweisung
- Häufige Fehlerquelle: Semikolon vergessen


```
1  class HelloPaco {
2     public static void main(String[] args){
3        System.out.println("Hallo Paco");
4     }
5  }
```

Code Blöcke

- Begrenzt durch { }
- Definieren Scopes (Gültigkeitsbereiche)
  - Klassen
  - Methoden
  - Schleifen / Bedingungen
  - Einfach nur Blöcke (selten genutzt)