Handout: Java Klassen, Packages und Package Naming

1. Was ist eine Klasse in Java?

Eine **Klasse** in Java ist ein Bauplan für Objekte. Sie enthält Eigenschaften (Variablen) und Verhalten (Methoden), die Objekte dieser Klasse besitzen können.

Beispiel einer Auto-Klasse

```
public class Auto {
    private String marke;

    public Auto(String marke) {
        this.marke = marke;
    }

    public void start() {
        System.out.println(marke + " startet.");
    }
}
```

---

2. Was ist ein Package in Java?

Ein **Package** ist eine Sammlung von Klassen und Interfaces, die zusammengehören. Es wird verwendet, um Code zu organisieren und Namenskonflikte zu vermeiden.

Warum Packages verwenden?

- **Bessere Organisation:** Gruppiert verwandte Klassen.
- **Namenskonflikte vermeiden:** Klassen in verschiedenen Packages können denselben Namen haben.

Beispiel eines Packages mit einer Auto-Klasse

Dateistruktur:

Datei: `com/vehicles/Auto.java`

```
package com.vehicles;

public class Auto {
    public void start() {
        System.out.println("Das Auto startet.");
    }
}
```

Datei: `com/vehicles/Main.java`

```
package com.vehicles;

public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();
        auto.start();
    }
}
```

Import von Packages

Wenn Klassen aus einem anderen Package verwendet werden sollen, müssen sie importiert werden.

```
import com.vehicles.Auto;

public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();
        auto.start();
    }
}
```

Package Naming

Konvention zur Benennung von Packages

Die Verwendung von `com.package` (z. B. `com.vehicles`) als Teil von Java-Packagenamen folgt einer bewährten Konvention für die **Organisation und Strukturierung** von Projekten. Diese Konvention wird als **Reverse-Domain-Naming-Konvention** bezeichnet.

Gründe für die Verwendung von `com.package`

1. Eindeutigkeit der Packagenamen

Durch die Verwendung von Domains (z. B. `com`, `org`, `net`) wird sichergestellt, dass der Paketname eindeutig ist.

Beispiel:

- Zwei Entwickler erstellen eine Klasse namens `Auto`:
  - Paket 1: `com.example.vehicles.Auto`
  - Paket 2: `org.myproject.vehicles.Auto`
- Es gibt **keinen Konflikt**, da die Packagenamen eindeutig sind.

---

2. Strukturierung von Projekten

- **Hierarchische Organisation:**
  - Der Domain-Teil (`com`, `org`) steht oben in der Hierarchie.
  - Danach folgt der spezifische Projekt- oder Modulname (z. B. `example`, `vehicles`).

Beispiel: Projektstruktur für ein Autoprojekt

```
src/
├── com/
│   └── example/
│       ├── vehicles/
│       │   ├── Auto.java
│       │   ├── Werkstatt.java
│       ├── utils/
│       │   ├── Helper.java
```

Diese Struktur:

- Organisiert Code nach Funktionalität oder Verantwortlichkeiten.
- Erleichtert die Navigation in größeren Projekten.

---

3. Domain-Referenz

- **Konvention:** Der Domainname des Unternehmens oder Projekts wird verwendet, aber in umgekehrter Reihenfolge.
- **Beispiel:**
  - Firma mit Domain `example.com` → Package-Präfix: `com.example`.

Dies stellt eine Verbindung zwischen dem Projekt und seiner Organisation her und macht den Ursprung des Codes deutlich.

---

4. Vermeidung von Namenskonflikten

- **Ohne Domain-Namen:**
  - Wenn zwei Entwickler denselben Paketnamen verwenden, z. B. `vehicles`, könnten Konflikte auftreten.
- **Mit Domain-Namen:**
  - Die Pakete `com.example.vehicles` und `org.myproject.vehicles` sind **eindeutig** und können unabhängig existieren.

---

5. Best Practice und Standard in Java

- **Java Naming Conventions:**
  - Java-Package-Namen folgen der **Reverse-Domain-Naming-Konvention**.
- **Framework-Unterstützung:**

- Frameworks wie **Spring** und **Maven** setzen diese Konvention voraus.

Wie wählt man einen Packagenamen aus?

1. Eigene Domain nutzen

- Wenn Sie ein Projekt für Ihre Firma erstellen, verwenden Sie den Firmen-Domainnamen.

Beispiel:

- Firma mit Domain `example.com` → Package-Präfix: `com.example`.

2. Kein Domainname?

- Nutzen Sie folgende Präfixe:
  - `org` für Organisationen.
  - `net` für Netzwerke.
  - `app` oder `my` für persönliche Projekte.

Beispiele

Firma mit Domain: `vehicles.com`

- **Packagenamen:**
  - `com.vehicles.cars` (für Fahrzeugklassen)
  - `com.vehicles.utils` (für Hilfsklassen)

Persönliches Projekt

- **Packagenamen:**
  - `app.myproject.main`
  - `app.myproject.utils`

Fazit

Die Verwendung von `com.package`:

1. **Erhöht die Eindeutigkeit** und verhindert Konflikte.
2. **Organisiert den Code** in einer hierarchischen Struktur.
3. **Ermöglicht Zusammenarbeit**, da verschiedene Teams oder Entwickler ihre Pakete klar abgrenzen können.

Auch wenn es nicht zwingend erforderlich ist, ist diese Konvention eine bewährte Praxis, die Professionalität und Ordnung in Java-Projekten fördert.

Praktisches Beispiel

Dateistruktur für ein Beispielprojekt:

```
src/
├── com/
│   └── vehicles/
│       ├── Auto.java
│       ├── Main.java
│   └── utils/
│       ├── Helper.java
│
```

Code für `Auto.java`:

```java
package com.vehicles;

public class Auto {
    public void start() {
        System.out.println("Das Auto startet.");
    }
}
```

Code für `Main.java`:

```java
package com.vehicles;

public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();
        auto.start();
    }
}
```