hhu.

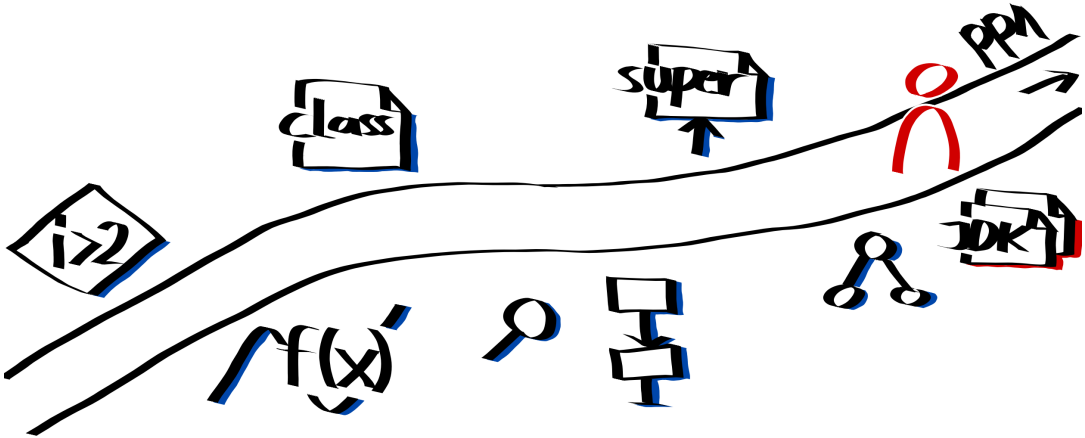
# Kapitel 8: Packages, Frameworks & Co.

## Packages, Classpath & Java-Archive

Wintersemester 2023/24

Stand 3. August 2023, 15:37 Uhr

# Wo stehen wir gerade?



# Vorlesungsfeedback

# Motivation

- Viele Klassen in einem Ordner $\rightarrow$ unübersichtlich
- Mehrdeutigkeiten von Klassennamen möglich
  - Punkt in einem Bild vs. Punkt im Raum

```java
public class Point {

    private int red;
    private int green;
    private int blue;
```

```java
public class Point {

    private double x;
    private double y;
    private double z;
```

# Mehrdeutigkeiten beheben: Möglichkeit 1

Heinrich Heine Universität Düsseldorf

Eindeutige Namen, Beispiele aus nativer Windows-API:

- IoCreateFile
- ZwCreateFile
- NtQueryInformationFile
- ZwQueryInformationFile

```java
public class ImagePoint {
    private int red;
    private int green;
    private int blue;
```

```java
public class GeometryPoint {
    private double x;
    private double y;
    private double z;
```

Gute Lösung?

# Mehrdeutigkeiten beheben: Möglichkeit 1

Heinrich Heine Universität Düsseldorf

Eindeutige Namen, Beispiele aus nativer Windows-API:

- IoCreateFile
- ZwCreateFile
- NtQueryInformationFile
- ZwQueryInformationFile

```java
public class ImagePoint {
    private int red;
    private int green;
    private int blue;
```

```java
public class GeometryPoint {
    private double x;
    private double y;
    private double z;
```

Gute Lösung?

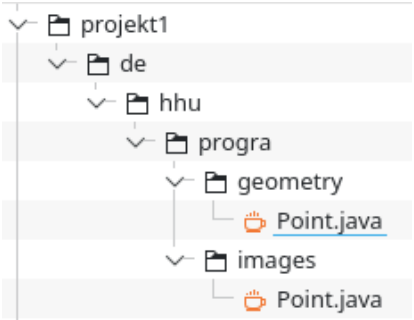
- Fehleranfällig
- Unübersichtlich

# Java-Lösung: Packages – Namensräume für Klassen

Heinrich Heine Universität Düsseldorf

- Hierarchischer Aufbau, entspricht Ordnerstruktur
- `package`-Statement gibt an, zu welchem Package Klasse gehört
  - muss erstes Statement in Datei sein
- Konventionen:
  - Packagenamen kleingeschrieben
  - Umgekehrte URL-Notation: weltweite Eindeutigkeit, sofern alle nur eigene Domains nutzen
- Analogie: Klasse = Vorname, Package = Familienname

# Beispiel: Package de.hhu.progra.geometry




```java
package de.hhu.progra.geometry;

public class Point {

    private double x;
    private double y;
    private double z;
```

```
~/projekt1 % javac de/hhu/progra/geometry/Point.java
~/projekt1 % java de.hhu.progra.geometry.Point
```

# Klassen aus anderen Packages benutzen

Heinrich Heine Universität Düsseldorf

2 Möglichkeiten:

1. vollqualifizierter Name: `java.util.Scanner`
   - unüblich
   - notwendig, falls zwei Klassen aus unterschiedlichen Packages mit demselben Namen benötigt

```
1 public class ScannerQualified {
2     public static void main(String[] args) {
3         java.util.Scanner stdin = new java.util.Scanner(System.in);
```

2. Klasse importieren: `import java.util.Scanner;`

```
1 import java.util.Scanner;
2
3 public class ScannerImport {
4     public static void main(String[] args) {
5         Scanner stdin = new Scanner(System.in);
```

# Muss man Klassen immer importieren?

# Muss man Klassen immer importieren?

Heinrich Heine
Universität Düsseldorf

Kein Import notwendig bei

- Klassen im selben Package
- Klassen aus `java.lang`, z. B.
  - `java.lang.Integer`
  - `java.lang.Math`
  - `java.lang.Object`
  - `java.lang.System`
- Verwendung von vollqualifizierten Namen

# Sind alle Klassen in einem Package?

# Sind alle Klassen in einem Package?

Heinrich Heine
Universität Düsseldorf

- Klassen ohne `package`-Statement in Default-Package[1]

! Klassen in Default-Package in keinem anderen Package verwendbar

⇒ Default-Package jenseits von Mini-Projekten vermeiden

[1] unnamed package

# Alle Klassen eines Packages importieren

Heinrich Heine Universität Düsseldorf

- `import java.util.*` importiert alle Klassen aus Package java.util

! Subpackages nicht importiert ⇒ Klassen aus java.util.stream nicht importiert

! Praxis: *-Imports vermeiden
  - schnell unübersichtlich (woher kommt welcher Datentyp?)
  - Namenskollisionen möglich

# Ist es nicht total lästig, andauernd zu importieren?

# Ist es nicht total lästig, andauernd zu importieren?

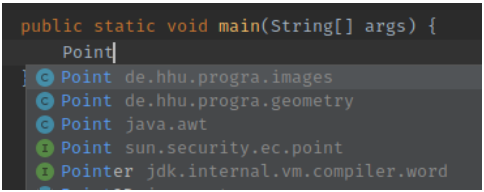
Heinrich Heine Universität Düsseldorf

Ja!

Lösung: Hilfe durch integrierte Entwicklungsumgebungen[2]

- (halb-)automatischer Import
- Wahl des korrektern Compilerverzeichnisses (sofern nicht falsch eingestellt)
- Classpath richtig gesetzt (sofern nicht falsch eingestellt)

! hätten aber auch wegen Klassen in Default-Package gemeckert




[2] Integrated Development Environments, IDEs: extrem aufgemotzte Texteditoren →Propra

# Sichtbarkeit von Klassen und Variablen

| Sichtbar in | `private` | – (package-private) | `protected` | `public` |
|---|---|---|---|---|
| Klasse selbst | ✓ | ✓ | ✓ | ✓ |
| Klassen im selben Package[3] | ✗ | ✓ | ✓ | ✓ |
| Unterklassen | ✗ | ✗ | ✓ | ✓ |
| überall sonst | ✗ | ✗ | ✗ | ✓ |

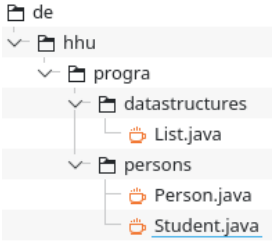
[3] gilt nicht für Unterpackages

# Für wen ist `studentNumber` sichtbar?

```java
package de.hhu.progra.persons;

public class Student extends Person {

    final int studentNumber;
```

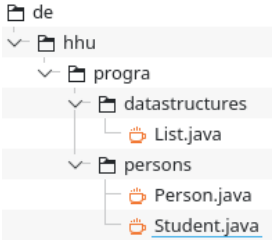


# Für wen ist `studentNumber` sichtbar?

```java
package de.hhu.progra.persons;

public class Student extends Person {

    final int studentNumber;
```




- Klasse Student selbst
- alle anderen Klassen im Package: Person
- **nicht** für Unterklassen und Klassen außerhalb des Packages

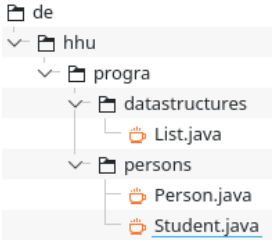
War das Absicht?

# Für wen ist `studentNumber` sichtbar?

```java
package de.hhu.progra.persons;

public class Student extends Person {

    final int studentNumber;
```




- Klasse Student selbst
- alle anderen Klassen im Package: Person
- **nicht** für Unterklassen und Klassen außerhalb des Packages

War das Absicht?

- wahrscheinlich `private` vergessen

# Sichtbarkeit und Vererbung



- Sichtbarkeit darf beim Überschreiben nicht kleiner werden:

```java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    String toString() {
        return "Ich heiße " + name + ".";
    }
}
```

```
Person.java:8: error: toString() in Person cannot override toString() in Object
    String toString() {
           ^
  attempting to assign weaker access privileges; was public
1 error
```

# Classpath

- JVM sucht Klassen/Packages in Classpath
- Standard-Classpath: `.` (Verzeichnis, in dem `java` ausgeführt wird)
- Überschreibbar:
  - Setzen der Umgebungsvariable `CLASSPATH`
  - Parameter `-cp` bei Aufruf von `javac`/`java`
- Mehrere Verzeichnisse mit Doppelpunkt[4] angebbar

```
javac -cp /mnt/java/project1:. ListTest.java
```

[4] Windows: Semikolon

# jar-Dateien

- Motivation: Weitergabe von Klassen an andere Personen
- Lösung: class-Datein in Java-Archiv (jar) verpacken
  - = zip-Archiv + Zusatzinfos $\rightarrow$Propra
- Java-Bibliotheken typischerweise als jar-Datei erhältlich
  - Bibliothek: Sammlung von Klassen + (hoffentlich) Dokumentation öffentlicher Klassen/Methoden
- Auch Java-Anwendungen mit main-Methode als jar-Datei verpackbar

# Erstellen und Verwenden einer jar-Datei

Heinrich Heine
Universität Düsseldorf

```
% javac de/hhu/progra/datastructures/List.java
% jar cf List.jar de/hhu/progra/datastructures/List*.class
% javac -cp List.jar:. ListTest.java
% java -cp List.jar:. ListTest
Erstes Listenelement: Hi
```

# Wer noch mehr wissen möchte . . .

Heinrich Heine Universität Düsseldorf

- Import statischer Methoden:
  `import static org.assertj.core.api.Assertions.assertThat;`
- Modulsystem (seit Java 9):
  `https://www.oracle.com/corporate/features/understanding-java-9-modules.html`

# Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- Klassen in Packages **anlegen**.
- Klassen aus anderen Packages **benutzen**.
- **erklären**, wo Java nach Klassen sucht.
- **angeben**, wie Klassen typischerweise an andere Personen weitergegeben werden.

# Wichtige Begriffe

Heinrich Heine
Universität Düsseldorf

package
Default-Package
Bibliothek

vollqualifizierter Name
Classpath

import
jar