Technische Hochschule Rosenheim
Technical University of Applied Sciences

Programming Basics – WiSe21/22

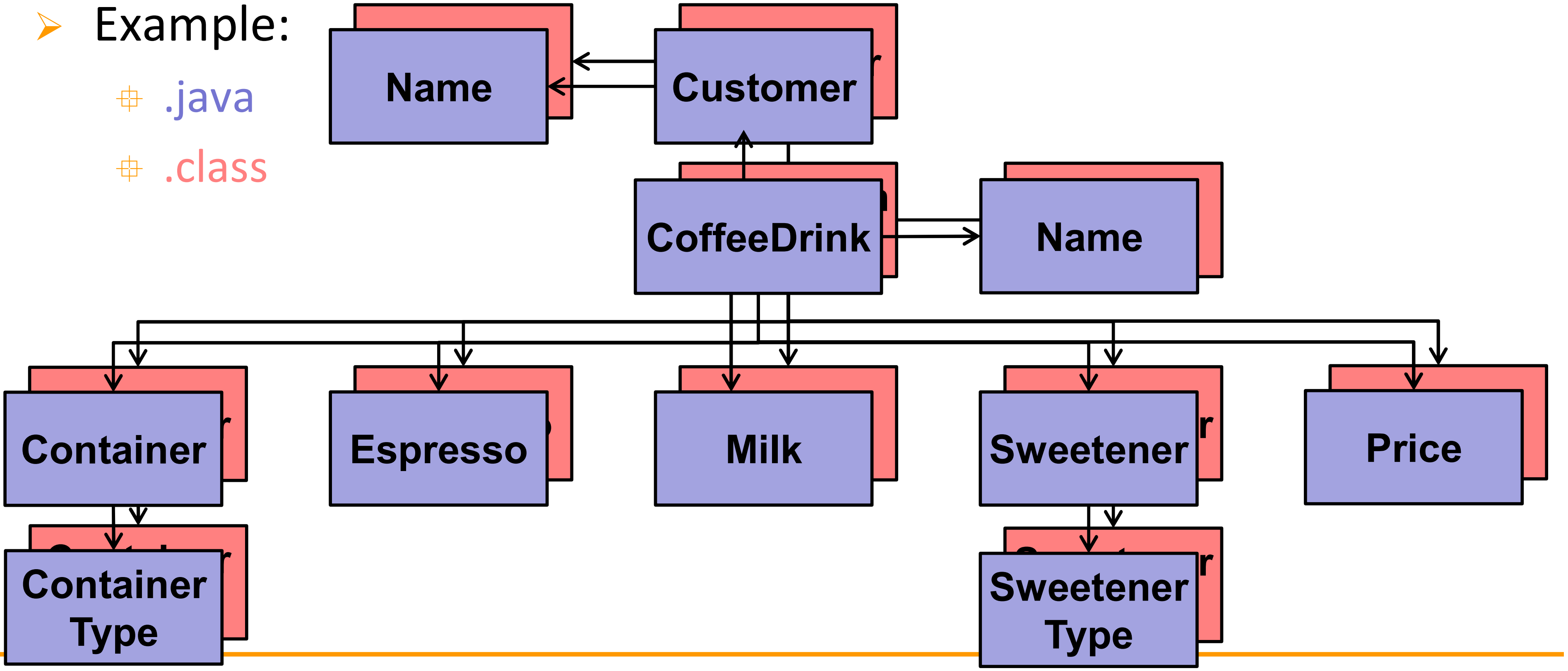
Packages

Prof. Dr. Silke Lechner-Greite

Table of contents – planned topics

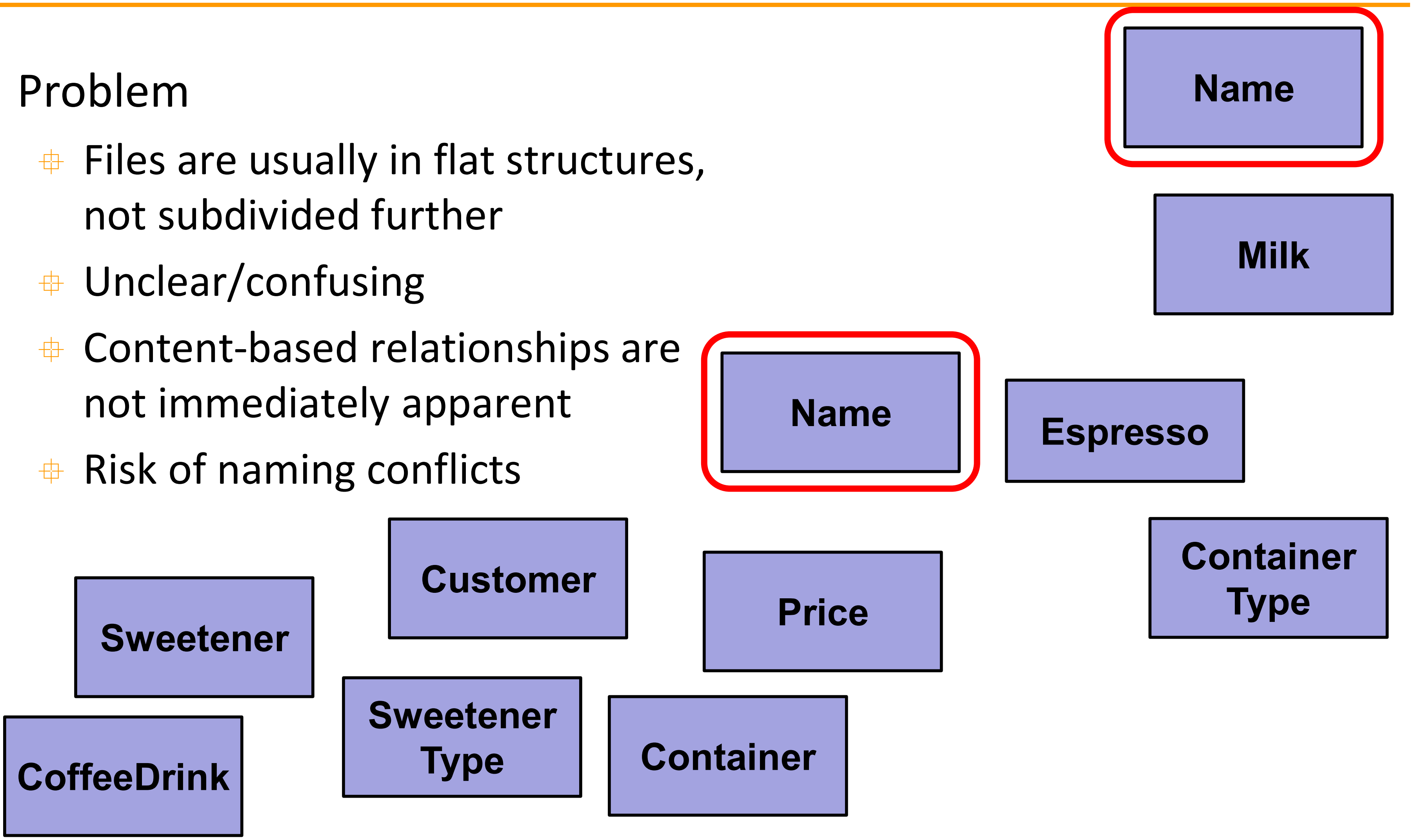
Motivation - initial situation

- Initial situation
  - Complex systems often involve a large number of classes
  - Per class (usually) one `.java` file and always one `.class` file
- Example:
  - .java
  - .class



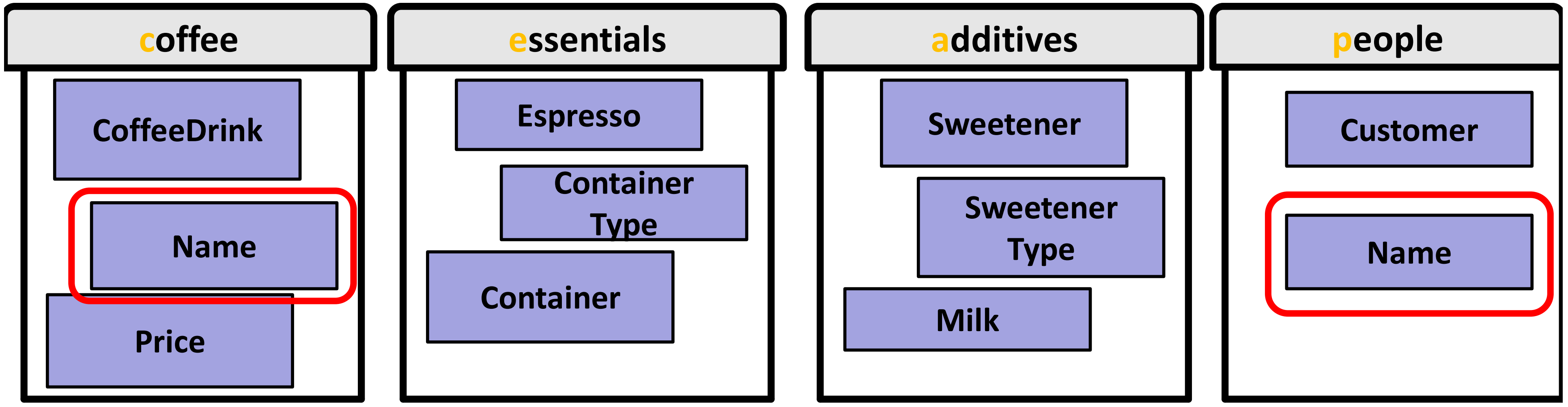
Motivation - problem

- Problem
  - Files are usually in flat structures, not subdivided further
  - Unclear/confusing
  - Content-based relationships are not immediately apparent
  - Risk of naming conflicts



Idea

- Remedy
  - Bundling related classes into a package
  - Things with related contents go into the same package



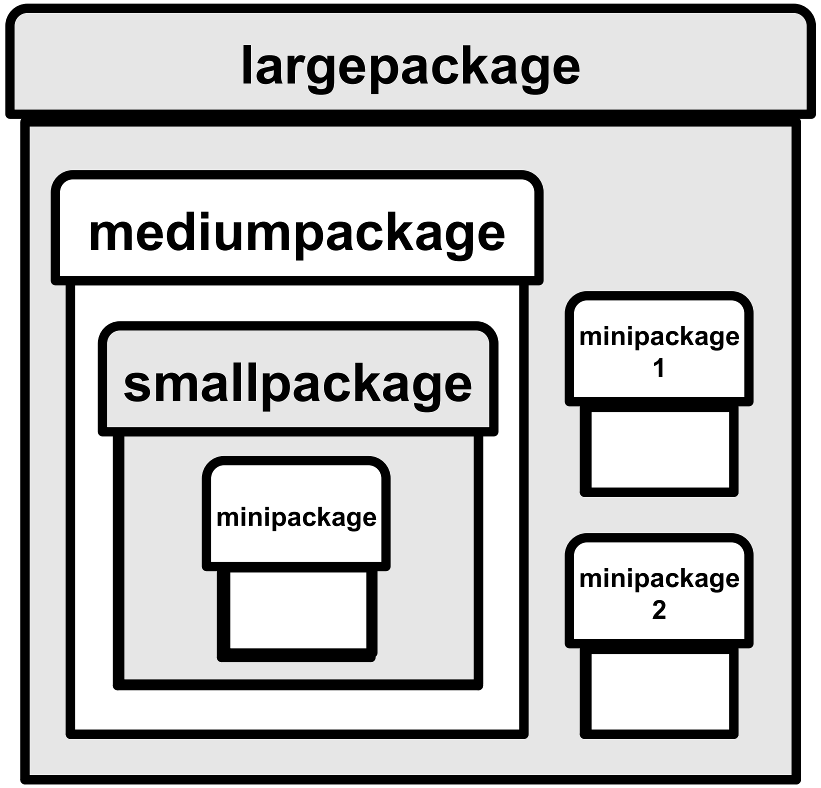
  - Each package contains unique identifiers
  - Class names in different packages are thus independent
  - Class name must only be unique within a package!

Packages

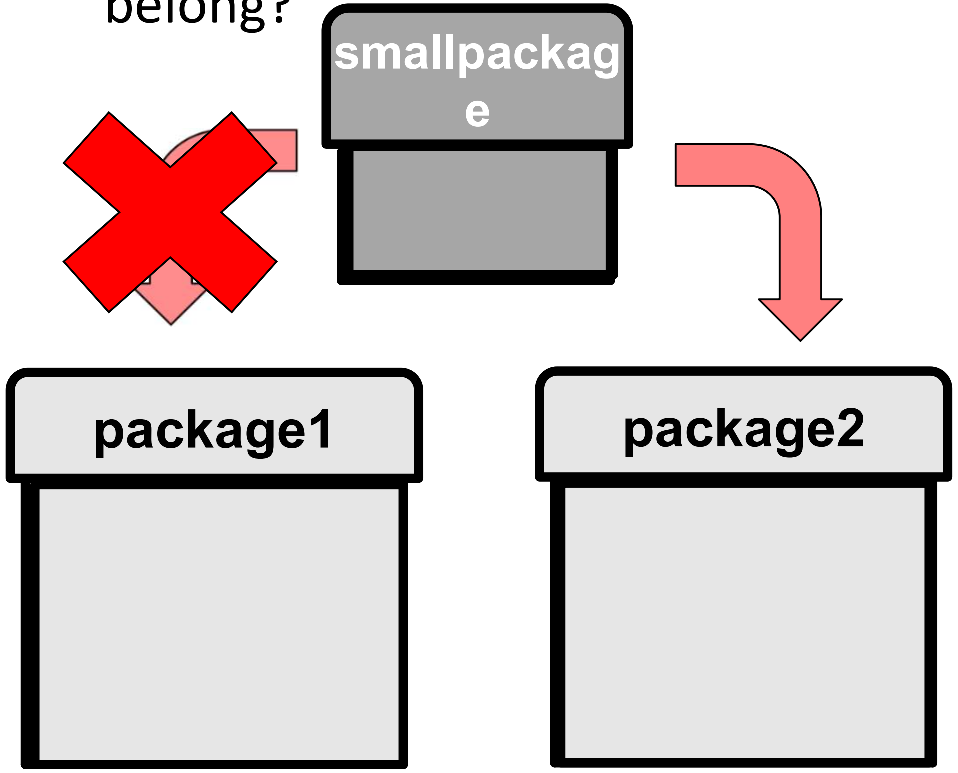
- Structuring mechanism
- Summarising components into a larger unit
- Package can itself contain other packages: hierarchy of available components
- Every class belongs to a maximum of one package

Relationships between packages (1)

Package can contain any number of subpackages



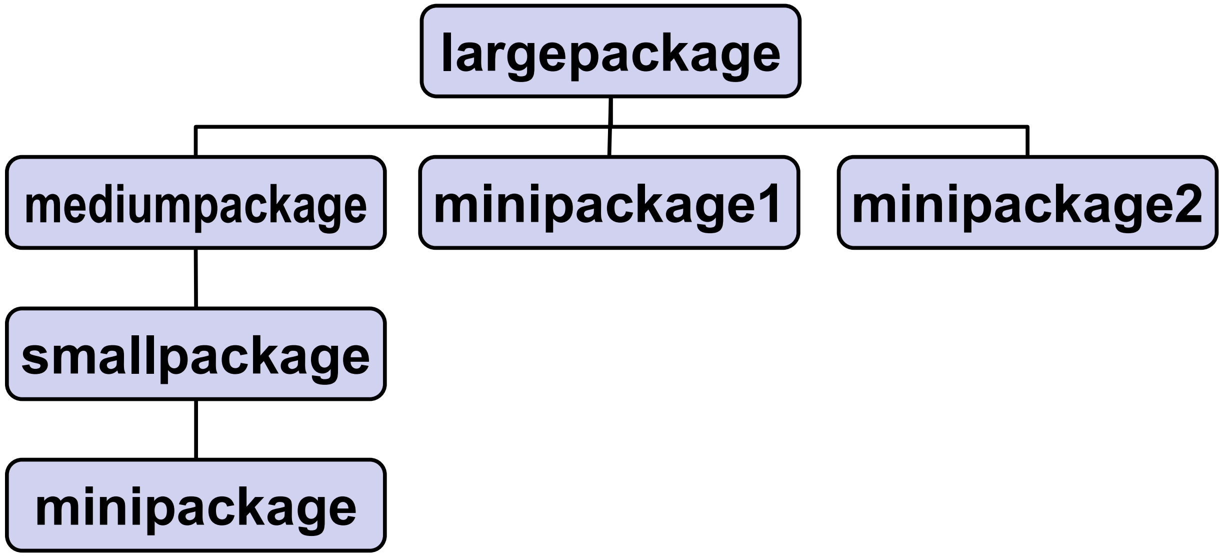
Where does the small package belong?



Package is in a maximum of one directly superordinate package!

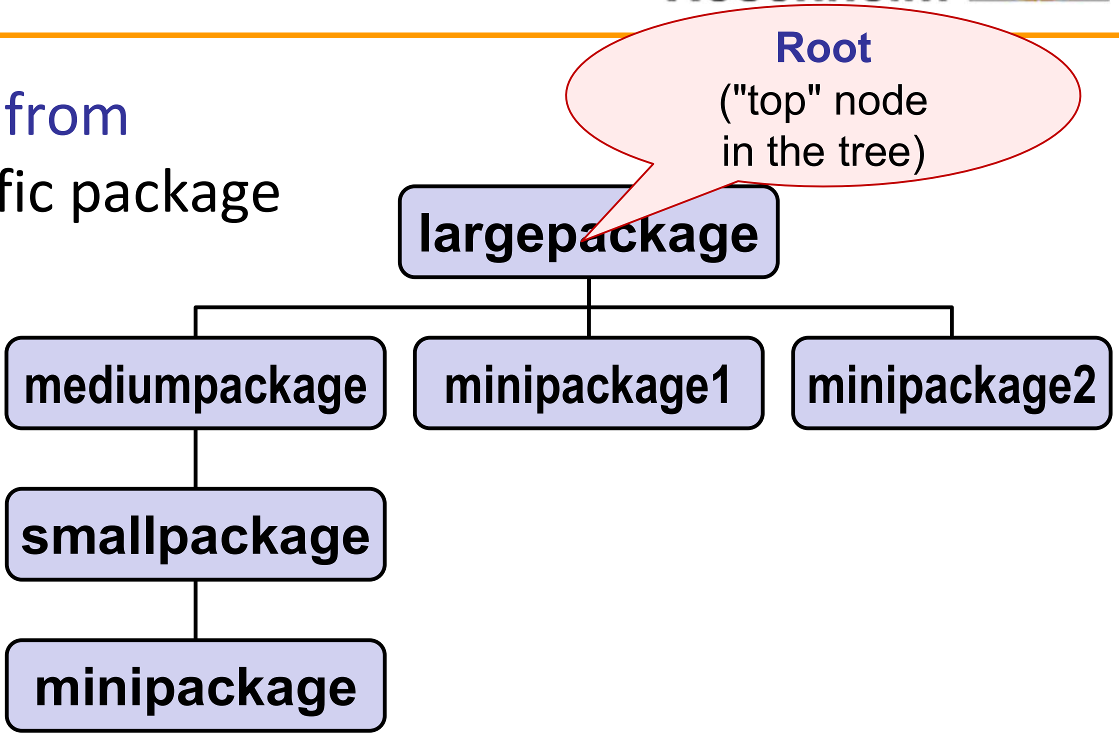
Relationships between packages (2)

- Package structure is hierarchical
- Nested packages as a tree



Package path (1)

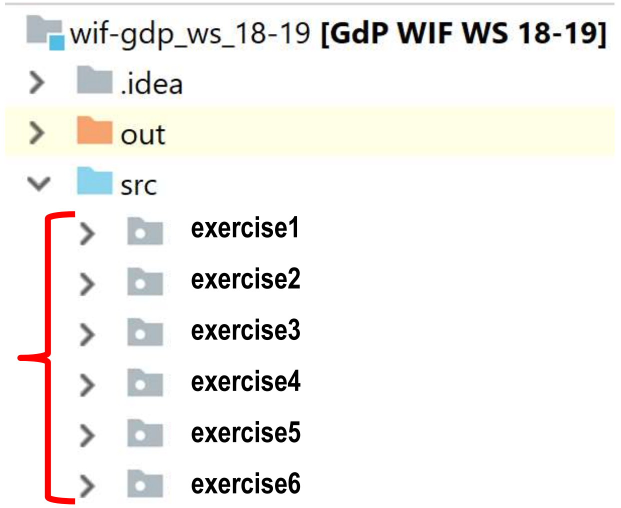
- Clearly defines the path from the root node to a specific package
- Described by a list of nested package names, separated by a dot



- Examples:
  - `largepackage`
  - `largepackage.mediumpackage`
  - `largepackage.mediumpackage.smallpackage`
  - `largepackage.mediumpackage.smallpackage.minipackage`
  - `largepackage.minipackage1`

Package path (2)

- View of the packages in the IntelliJ project tree

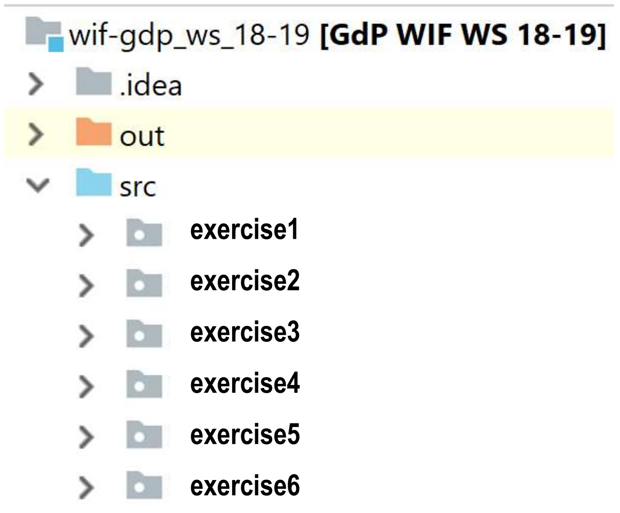


Note:

- Every class is part of exactly one package (if not explicitly assigned, then the `default package`)
- Within a package: classes & sub-packages are unique
- Package structure only externally apparent; for Java, all packages have equal rank

Package naming (identifier) conventions (1)

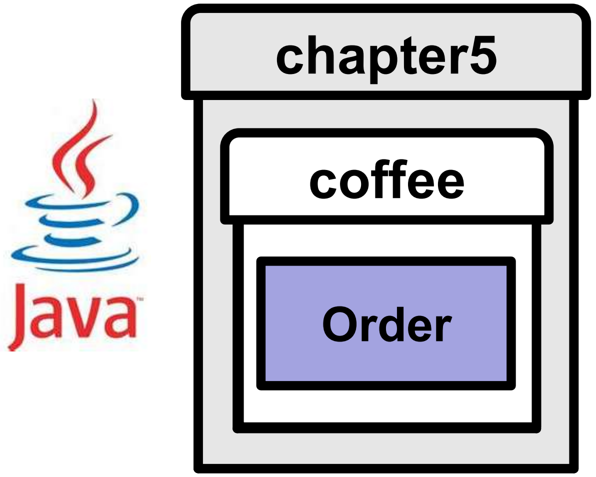
- Convention for notation:
  - English names in lower case letters and numbers
  - No upper case letters, special characters, etc.!
  - Examples: `largepackage, package1`
- Why?
  - Package structure is mapped to directories in the file system, conflicts due to case insensitivity or special characters
  - Nested packages correspond to nested directories

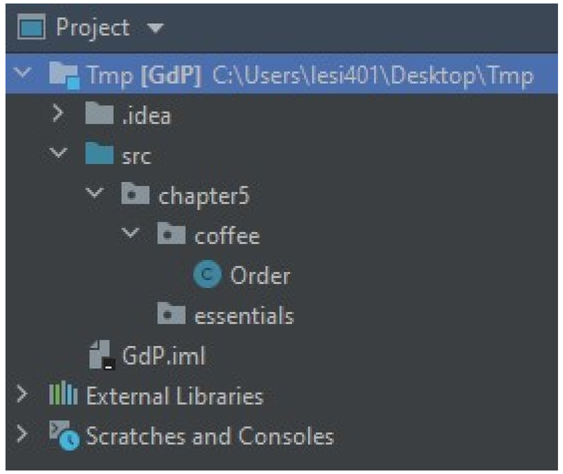


Package naming (identifier) conventions (2)

- Example: `Order.java`
  Package `coffee.Order`

Directory `chapter5`
Subdirectory `coffee`





- Package names checked by Java
- Directory names checked by the operating system
- Possible difficulties with upper/lower case and special characters
- => defensive naming rule: only lower case letters and numbers

Organisation schema for package identifiers

- Aim:
  - Smooth exchange of bytecode between developers
  - Regardless of source
- Package naming convention:
  - Package path analogous to domain names on the Internet
  - The most abstract (most high-ranking) domain delivers the highest package
  - Subdomains identify subpackages
  - Further package organisation according to conventions of the institutions; for example, include team and project names
- Example:
  - Classes under the package path `de.ro.inf`

Predefined packages

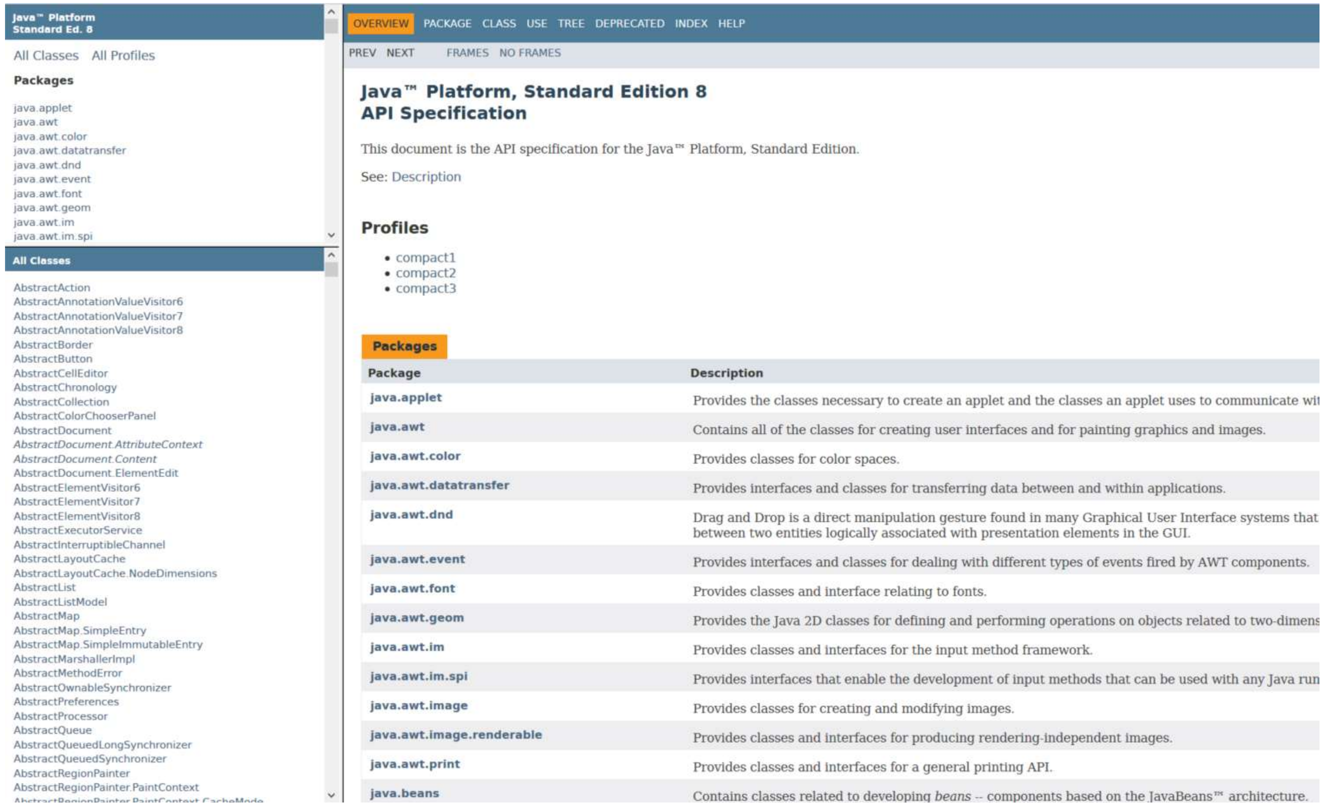
- Java includes a variety of packages that come with the Java Development Kit (JDK)
  - Standard classes in the `java` package
  - Subpackage `java.lang`
    - Stands for *Java language*
    - Contains the most important standard classes, e.g. `String`, `Array`, …
    - Automatically imported; no explicit import necessary
  - Candidates for future standard classes in the `javax` package
    - *Java extensions*
    - May perhaps be moved to the `java` package in future Java versions
  - Documentation in *Java API Specification*

⚠ Do not subordinate your own packages to `java` or `javax` !

Predefined packages - Java API Specification

- API = Application Programming Interface
- Description of the programming interface of the respective Java version

Source: https://docs.oracle.com/javase/8/docs/api/



Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractElementVisitor8
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer
AbstractPreferences
AbstractProcessor
AbstractQueue
AbstractQueuedLongSynchronizer
AbstractQueuedSynchronizer
AbstractRegionPainter
AbstractRegionPainter.PaintContext
AbstractRegionPainter.PaintContext.CacheMode

Java™ Platform, Standard Edition 8
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate wit
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimens
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java run
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.

Using packages

- In order to use a class, the package in which it is located must be specified
- 2 types:
  1. Address the class with full names

     ```
     java.util.Random aCoincidence = new java.util.Random();
     ```

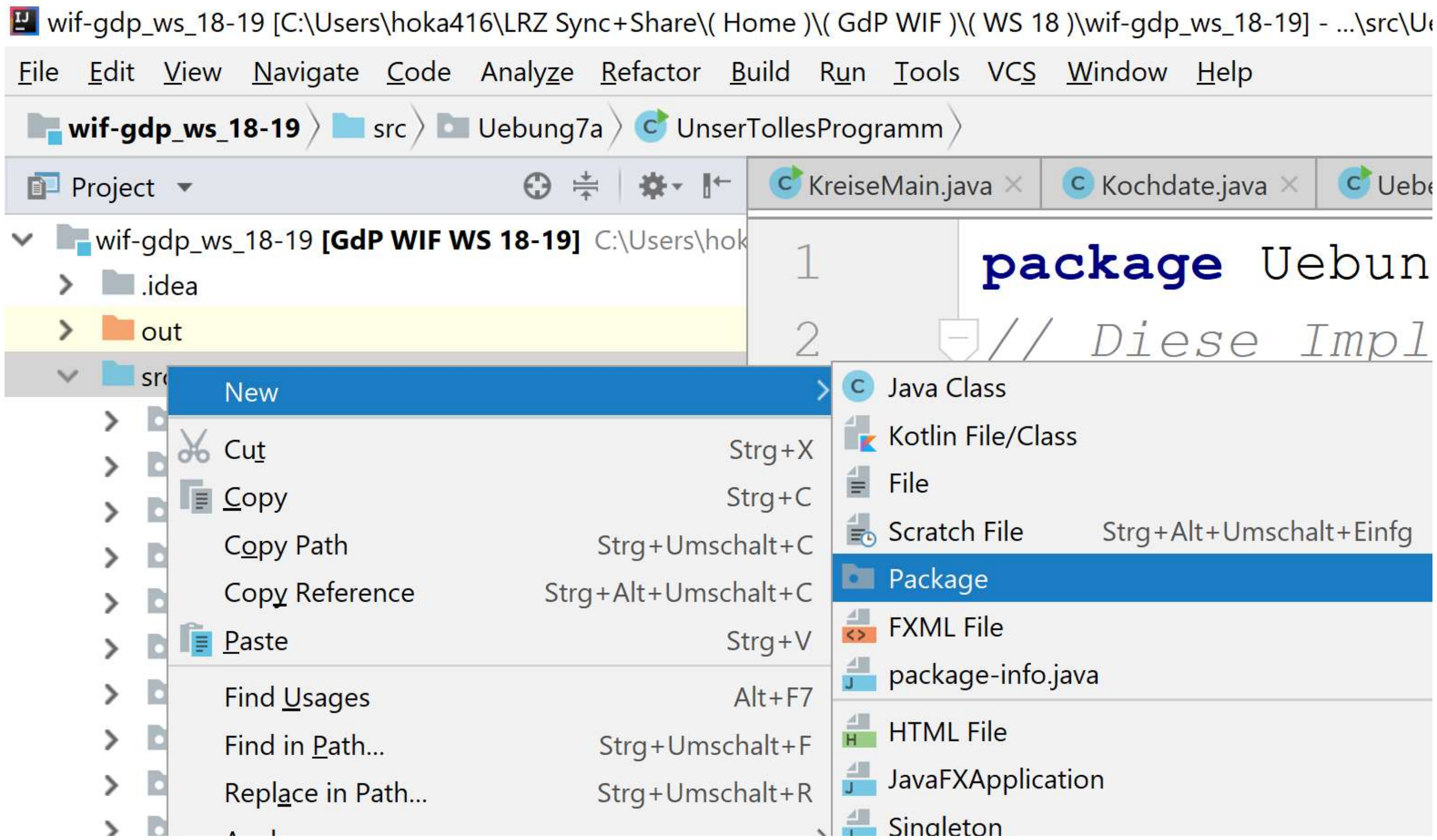
  2. Include with `import` statement

     ```
     import java.util.Random;
     Random aCoincidence = new Random();
     ```

- Can include all classes of a package:

```
import package.*;
```

Own packages - creation in IntelliJ


wif-gdp_ws_18-19 [C:\Users\hoka416\LRZ Sync+Share\(Home)\(GdP WIF)\(WS 18)\wif-gdp_ws_18-19] - ...\src\Ue

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

wif-gdp_ws_18-19 〉 src 〉 Uebung7a 〉 UnserTollesProgramm 〉

Project ▾ | KreiseMain.java × | Kochdate.java × | Uebe

wif-gdp_ws_18-19 **[GdP WIF WS 18-19]** C:\Users\hok
.idea
out
src

```
package Uebun
// Diese Impl
```

New	›
Cut	Strg+X
Copy	Strg+C
Copy Path	Strg+Umschalt+C
Copy Reference	Strg+Alt+Umschalt+C
Paste	Strg+V
Find Usages	Alt+F7
Find in Path...	Strg+Umschalt+F
Replace in Path...	Strg+Umschalt+R

Java Class	
Kotlin File/Class	
File	
Scratch File	Strg+Alt+Umschalt+Einfg
Package	
FXML File	
package-info.java	
HTML File	
JavaFXApplication	
Singleton	

Own packages – assigning class

```java
package de.ro.inf.p1.packages;
```

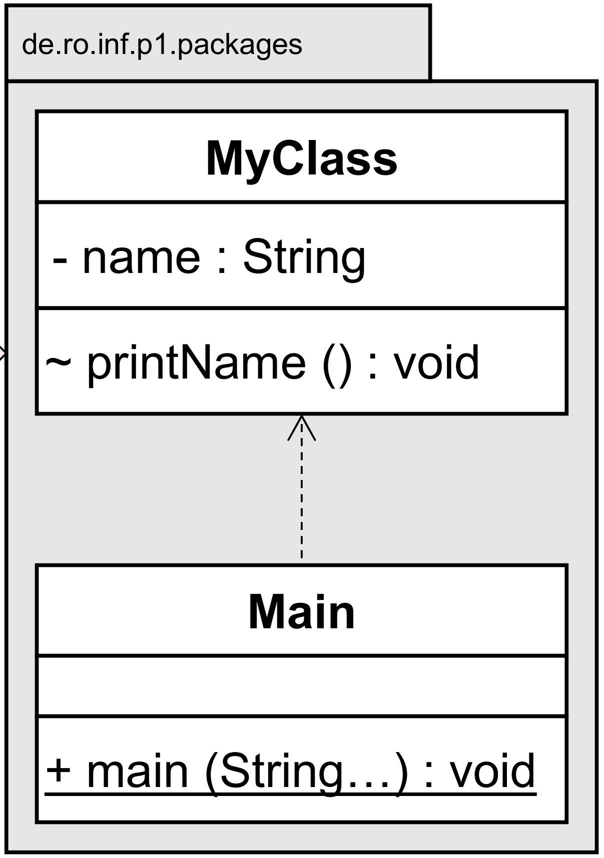
```java
class MyClass {
   /** Attribute */
  private String name = "Max";
  void printName() {
    System.out.println(name);
  }
}
```

Package

Standard visibility, i.e. package-wide

```java
package de.ro.inf.p1.packages;
```

```java
public class Main {
  public static void main(String[] args) {
   MyClass mc = new MyClass();
   mc.printName();
   }
}
```



Specifying package membership (1)

Technische Hochschule Rosenheim

- Meaning:
  - `package` clause specifies package membership
  - Counterpart: `import` clauses regulate access to other packages
  - Syntax: `package` *`packagepath`*`;`
- Example:

```
package coffeeshop.people;
class Name {...}
```

- Guidelines for use:
  - `package` clause first in the source text, before `import` clauses
  - `package` clause and path in the file system must match!

Specifying package membership (2)

- Standard package
  - Without specifying a `package` clause:
    Class declaration is in the standard package (default package)
  - Nameless package
  - Therefore, content cannot be imported into other classes

Own packages – access rights and visibility

- Packages introduce additional access rights and visibility rules
- Four different access categories:
  - `public`: allows "global" access (UML: +)
  - `private`: visible only within the own class (UML: -)
  - `protected`: applies in connection with inheritance (UML: #)
  - not specified: only visible within the package in which the class is declared; no access from outside the package (UML: ~)
- Can be assigned individually for each class, each attribute and each method

! As restrictive as possible! (information hiding!)