PACKAGES

CSC 207 SOFTWARE DESIGN

Computer Science
UNIVERSITY OF TORONTO

LEARNING OUTCOMES

- Understand what packages are in Java
- Understand various ways you can organize your code in packages

PACKAGES IN JAVA

https://docs.oracle.com/javase/tutorial/java/concepts/package.html

Package

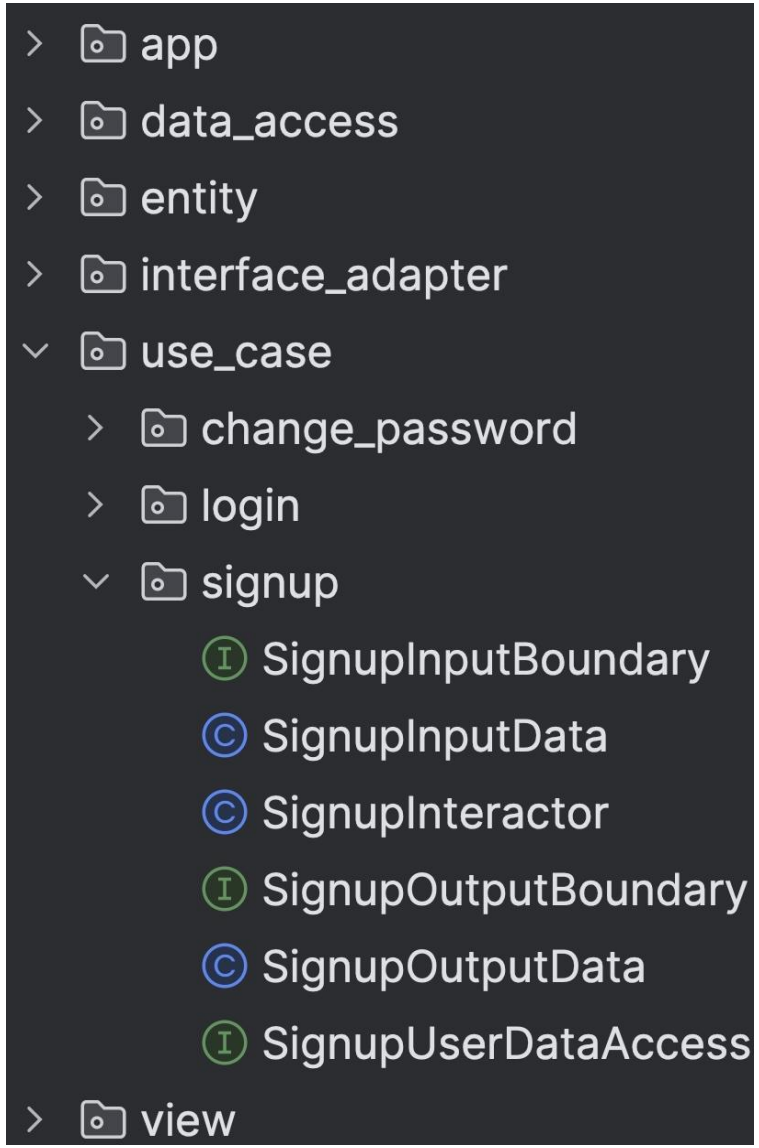
- Contains related classes and interfaces
- The full name of a class includes the package name: `java.util.ArrayList`
- The path from your sources root to a given class — delimited by .

When you get to a dozen or more classes, package organization is essential!

Examples: `use_case.signup`

`use_case.login`

Our homework 5 package structure



ACCESS MODIFIERS REVISITED

- Classes can be **public** or **package-private** (no explicit modifier)
- Members of a class (methods, variables) can also have **private** (within class only) or **protected** (within package and in subclass outside of package) modifiers.
- Note that there is no concept of providing access specifically to "subpackages".
  - **Other than for organizational purposes, the packages `use_case.signup` and `use_case` have NO connection.**
    - For example, if we defined a class in the `use_case` package, we have no way of saying that classes in `use_case.signup` can access it, but not classes in the `app` package.

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

EXAMPLE: ONLINE BOOK STORE

- When building an online book store, one of the use cases we've been asked to implement is about customers being able to view the status of their orders.

This is based on Chapter 34: The Missing Chapter from the CA textbook

HOW COULD YOU PACKAGE THIS CODE?

- **`OrdersController`**: A web controller, something that handles requests from the web
- **`OrdersService`**: An interface that defines the “business logic” related to orders.
- **`OrdersServiceImpl`**: The implementation of the orders service.
- **`OrdersRepository`**: An interface that defines how we get access to persistent order information.
- **`JdbcOrdersRepository`**: An implementation of the repository interface that saves information in a database.

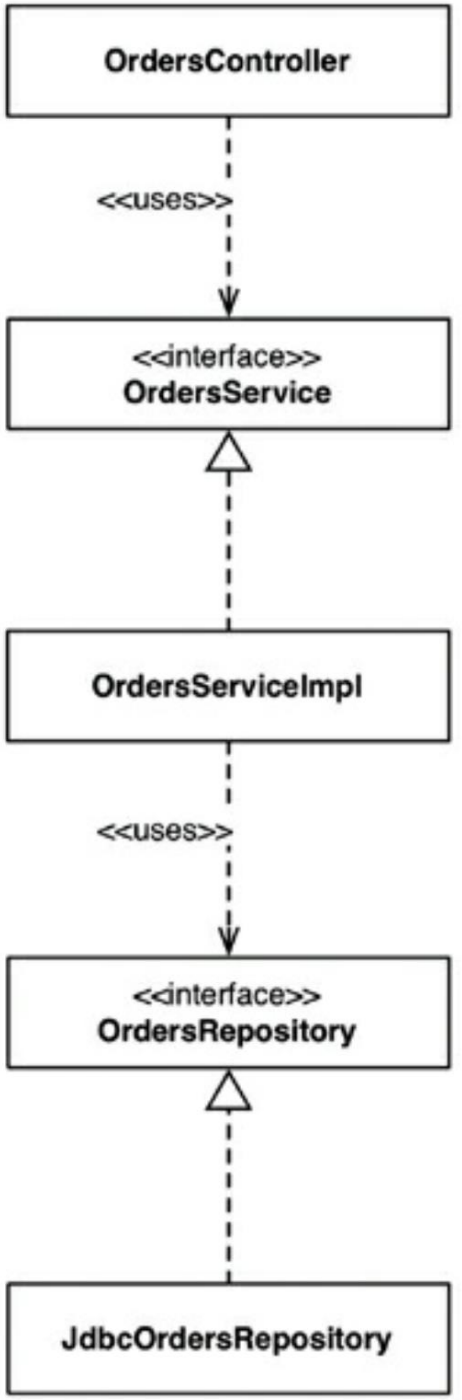
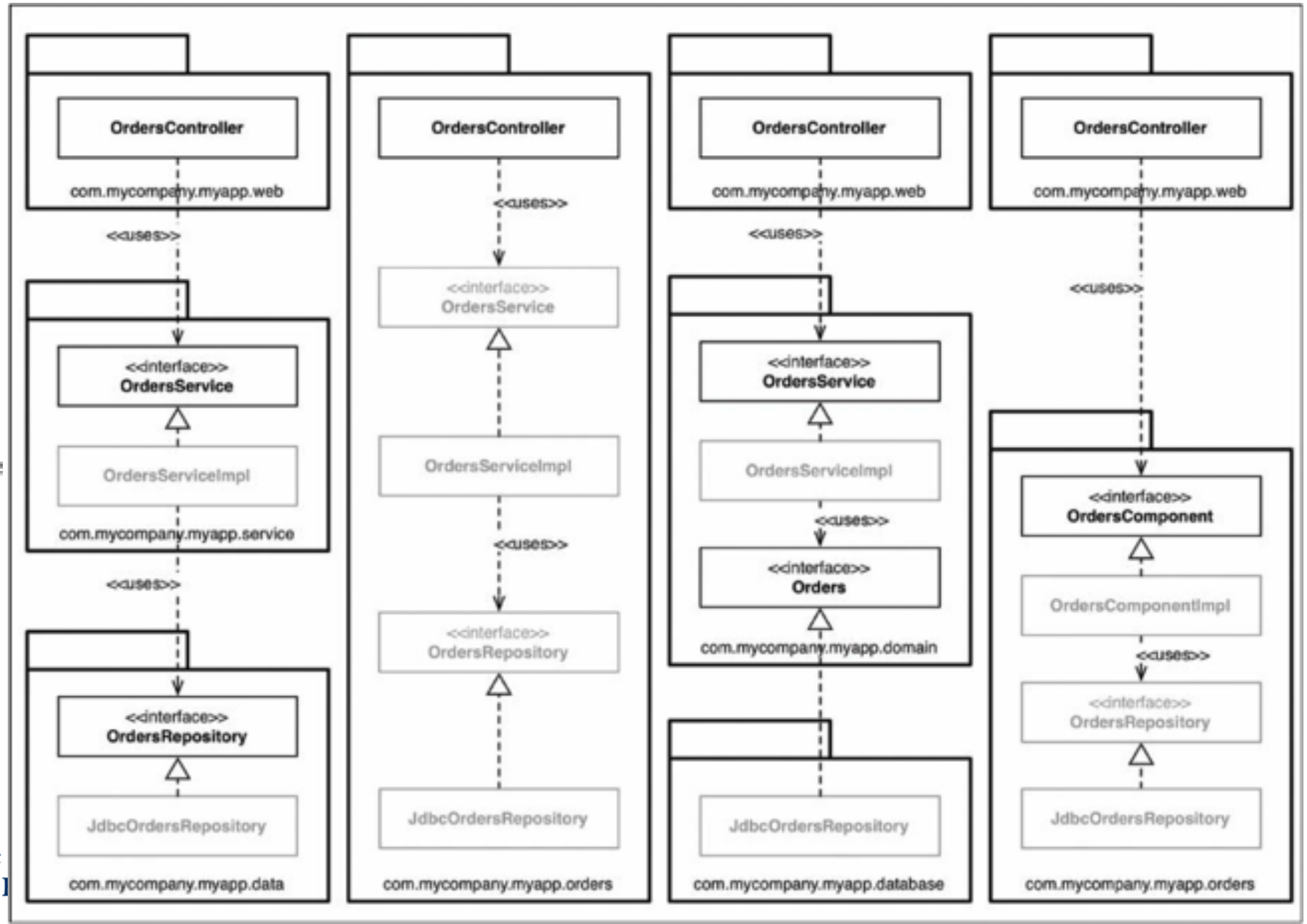


Figure 34.8
Chapter 34
Clean Architecture



SCREAMING ARCHITECTURE

https://blog.cleancoder.com/uncle-bob/2011/09/30/Screaming-Architecture.html

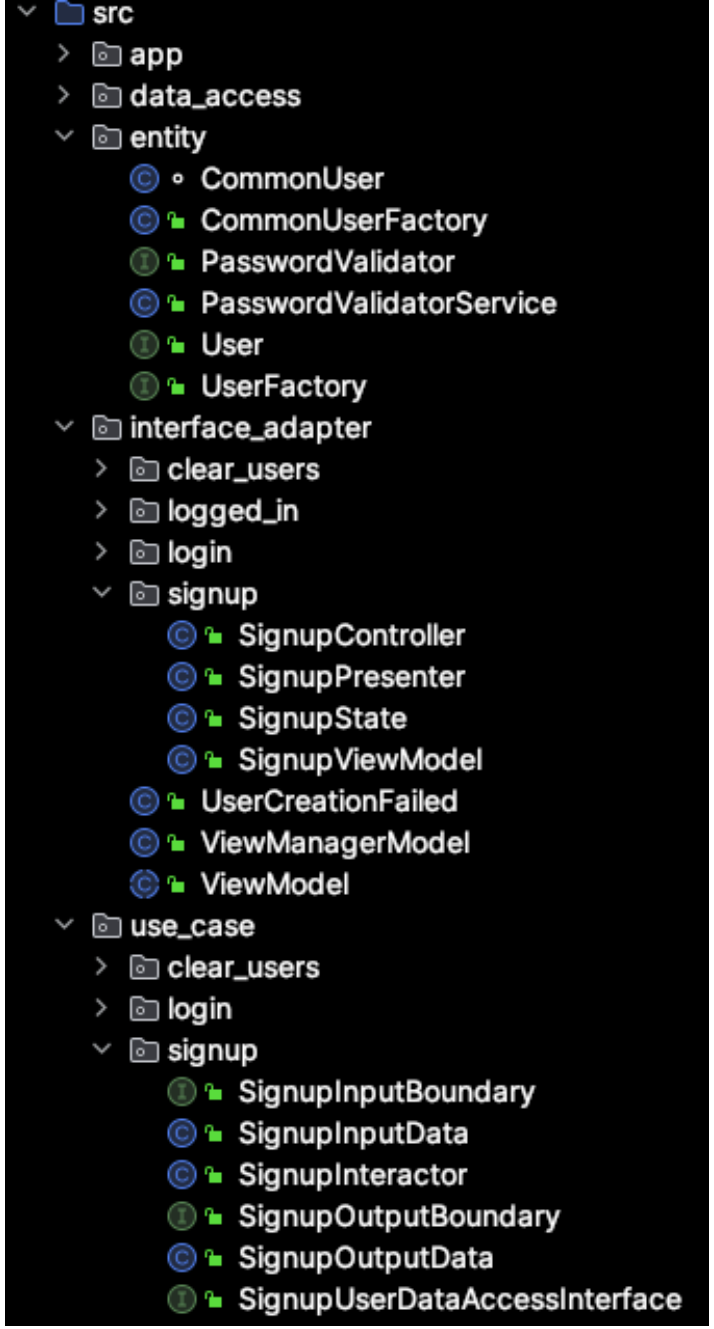
The structure of the code should primarily reflect ***what*** the program is doing (what the domain/application is) rather than ***how*** it is doing it (what the frameworks and details of the implementation involve)

```
~/Code/screaming tree ./screaming
./screaming
├── backoffice
│   ├── contracts
│   │   ├── create
│   │   │   ├── controller.py
│   │   │   └── serializer.py
│   │   └── models.py
│   └── offices
│       ├── list
│       │   ├── controller.py
│       │   └── serializer.py
│       └── models.py
└── car-rental
    ├── cars
    │   ├── models.py
    │   └── rent
    │       └── controllers.py
    └── customers
        ├── create
        │   ├── controller.py
        │   └── serializer.py
        └── models.py

10 directories, 11 files
~/Code/screaming
```

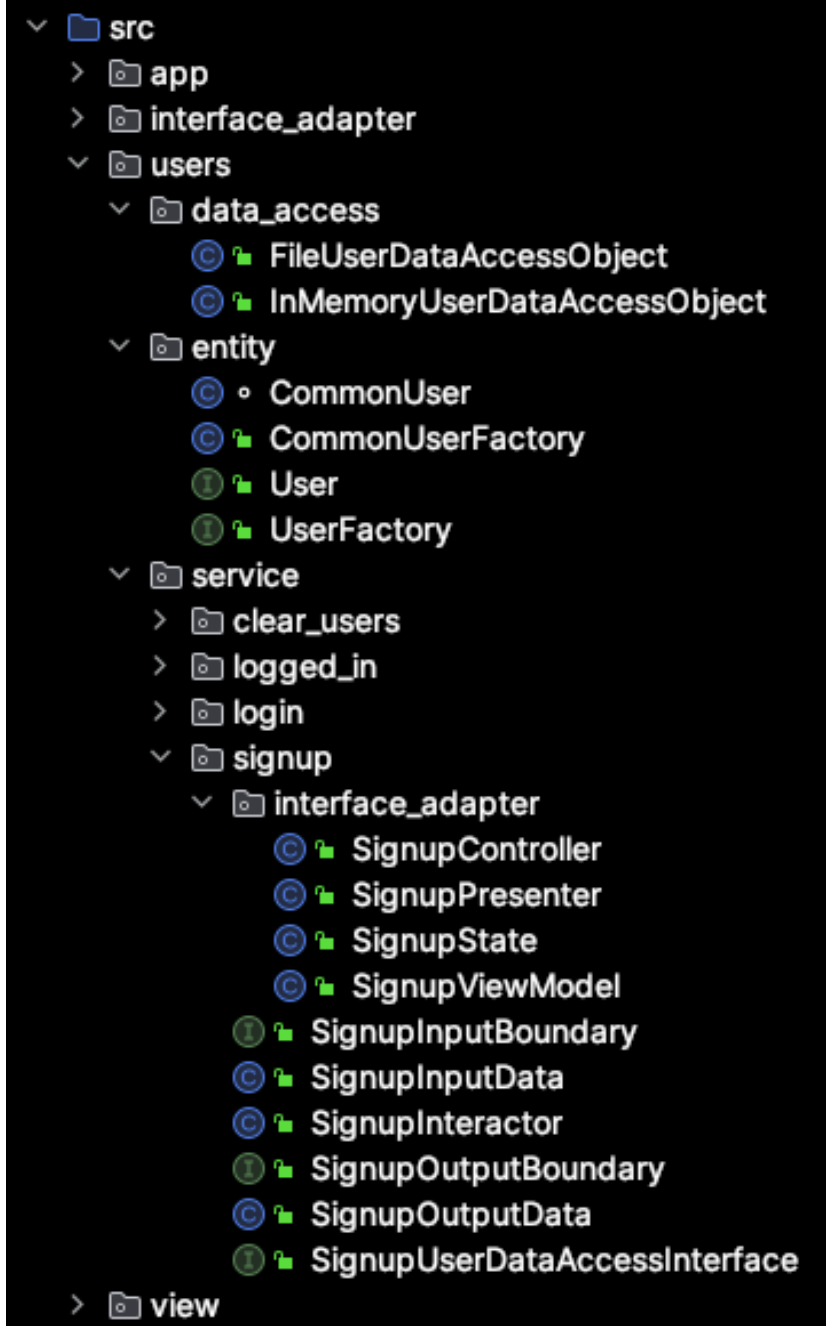
```
~/Code/screaming tree ./non-screaming
./non-screaming
└── django
    ├── business
    │   ├── rules.py
    │   └── utils.py
    ├── database
    │   └── models.py
    └── web_api
        ├── controllers
        │   ├── backoffice.py
        │   └── car_rental.py
        └── serializers
            └── serializers.py

6 directories, 6 files
~/Code/screaming
```

https://levelup.gitconnected.com/what-is-screaming-architecture-f7c327af9bb2

Package by layer of Clean Architecture



An attempt at Package by component (screaming)



CONCLUSIONS FROM CHAPTER 34 FROM CLEAN ARCHITECTURE TEXTBOOK

Packages can potentially provide both organization and encapsulation (using access modifiers)

For the project, your team needs to choose a package structure … and that takes practice!

- IntelliJ will automatically support you as you move files between packages!
- Choose your access modifiers to reflect your design, exposing only the necessary methods.

ASIDE: PROJECT STRUCTURES

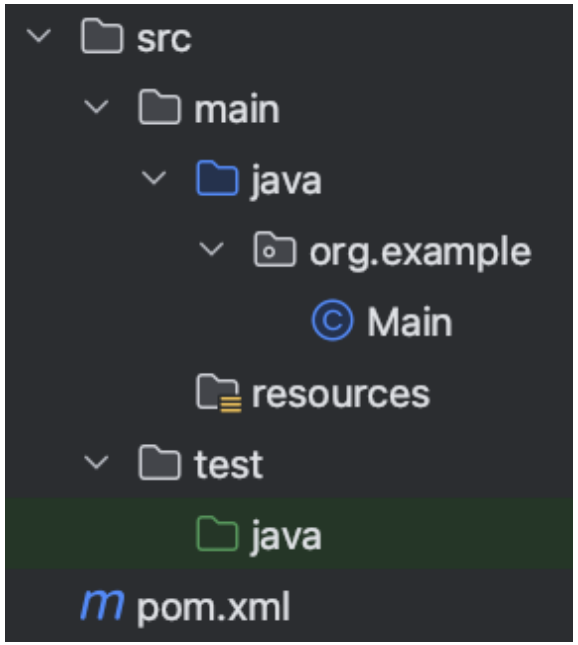
- In IntelliJ, you can create a project which will use one of three build systems: IntelliJ (default), gradle, or maven.
- Each of these define their own structure for how they organize your files.
- In all cases, you'll have a Sources Root and a Test Sources Root marked.
- Within these, you will have your usual Java package structure.
  - Parallel directory structures in your Sources Root and Test Sources Root directories will correspond to the same Java packages!

ASIDE: PROJECT STRUCTURES

Gradle (https://docs.gradle.org/current/userguide/organizing_gradle_projects.html)

Maven (https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html)

Maven



Gradle