Overview

Table of Contents

1. Overview
2. Interface-based Programming
3. Tools
4. Project Structures

SOFTWARE SYSTEM

Notes
sub–system: A part of the entire system that provide a well–defined functionality.
module: A development unit that has a well–defined purpose. Modules are identified by their name.
component: A module used by another module.
artifact: A specific version of a module. A module with a version number.

These terms are slightly different. During the course, we will stick to these definitions. Do not mix them.

Overview

Modules, Components
- Tests
  - Unit
  - Component
  - Integration
- Dependencies
- Build process
- Deployment

Version number
- major.minor.build.revision
- alpha, beta, release
candidate, commercial
distribution
- Never use multiple version of the same module!

Component
- Standalone Development Unit
- Specific Functionality
  - Abstractness
  - Granularity
  - Communicates via Interface
- Specific Technologies
  - JDBC, JPA, myBatis
  - J2EE, Spring
  - Jackson, JAXB
- Other Components
  - Integrate
  - Depend
- Encapsulate Functionalities
- Simplify Development
  - Standardization
  - Categorize Services
  - Lock Up Technologies
- Facilitates Testing
  - Component Tests
  - Integration Tests
- Difficult to Design
  - Experience Required
  - Costly Decisions
- Obedience to Standards
  - Code Review
### 3rd Party Components

**Pros**
- Boxed Solutions
- General Tasks
- Faster Development
- Reusable Components

**Cons**
- Learning
- Depends on Providers
- Versions
- Bugs!!!
- Support???

- Logging
  - log4j, log4j2, slf4j

- Data Access
  - JDBC, myBatis
  - JPA, Hibernate
  - Spring Data

- Data Conversion, Marshalling
  - JAXB,
  - Jackson, gson

- Testing
  - JUnit
  - EasyMock, Mockito

---

### Interface-based Programming

#### Table of Contents

1. Overview
2. Interface-based Programming
3. Tools
4. Project Structures

---

### Component Design

- Divide and Conquer!
  - Complex tasks can be broken down
  - Increase re-usability
  - Simplify tasks
  - Separate different
    - programming language
    - tools
    - technologies
# Interface-based Programming

- Component Communication
- Separation of Definition and Implementation
- Decouple Components
  - Loose Coupling
  - Exchangeable Components
- Facilitates
  - Design
  - Development
  - Maintenance

## Interface

```java
/**
 * Interface Description
 */
interface MoneyExchangeService {
    /**
     * Method details
     * @param amount ...
     * @param currency ...
     * @return ...
     * @throws ...
     */
    void exchange(
        Double amount,
        Currency currency)
        throws ExchangingException;
}
```

## Abstract Class vs Interface

### Similarities
- Design Elements
- Abstract Types
- Define Behavior

### Differences
- Fields
- Concrete Methods
- Multiple Inheritance

### Decision Support
- Abstract class if:
  - Fields are Needed.
  - Constructor is Needed.
  - Concrete Method is Defined
    - Template Method
- Otherwise Interface
Testing Dependencies

Component Tests
- Tested Separately
- Mocking External Dependencies

Integration Tests
- Testing with External Dependencies
- No Mocking
- Testing in “Real” Environment
- Assume Everything is Available

Does the component work properly, if the external dependencies work expectedly?

Does the component works properly in the System?

Tools

Table of Contents

1. Overview
2. Interface-based Programming
3. Tools
4. Project Structures

Tools – Maven

Application
  - mvn <goal>
  - Eclipse plugin

Packaging
  - pom
  - jar
  - war

Properties
  - Inheritance

Command Line Tool
  - Scripts
  - Integration

Project Structure
  - src
    - main
    - test
  - target
  - pom.xml
    - groupId, artifactId, version
**Tools**

**Project Object Model - pom.xml**

- **Artifact Identification**
  - `groupId` Company or Project Name
  - `artifactId` Component Name
  - `version` Version Number
- **Parent Project**
- **Packaging**
- **Properties**

```xml
<project>
  <groupId>Company or Project Name</groupId>
  <artifactId>Component Name</artifactId>
  <version>Version Number</version>
  <parent></parent>
  <properties>
    <junit.version>4.12</junit.version>
    <project.basedir>${project.basedir}</project.basedir>
    <project.version>${project.version}</project.version>
  </properties>
</project>
```

**Build Configuration**
- **Project Information**
- **Development Environment**
  - Source Code Management
  - Issue Tracker
  - Mailing Lists
  - Developers

---

**Maven Build**

- **Source code → Software**
- **Common Task & Fix Steps**
  - Compile
    - Classes
    - Components
  - Testing
  - Linking
  - Distribution
  - Make
    - maven, gradle, ant
    - Jenkins CI
  - **Validation & Fix Steps**
    - Validate
    - Compile
    - Test
    - Package
    - Integration-Test
    - Verify
    - Install
    - Deploy

---

**Maven Life-Cycle**

- **Build Steps → Goals**
- **Previous Steps are Required**
- **Step Failure → Build Failure**
- **Configuration via Plugins**

---
Maven Life-Cycle

- **clean**
  - Remove target directory
- **validate**
  - Check pom.xml
- **compile**
  - src/**.java → **.class
- **test**
  - JUnit (test/**IT.java)
  - Surefire
- **package**
  - Zip to jar or war
- **integration-test**
  - JUnit (test/**IT.java)
- **verify**
  - Check Quality Criteria
- **install**
  - Copy to Local Repository
- **deploy**
  - distributionManagement
  - Publishing, Sharing

Maven Dependency Management

- **Other Component**
  - 3rd Party Library
  - Other Part of the System
- **Deployed Artifact**
- **Stored in Repository**
- **Automatic**
  - Search
  - Download
  - Adding to ClassPath

Repositories

- **Company Network**
- **Private Repository**
- **Proxy**
- **Internet**
- **Central Repository**
- **Private Repository**
- **Proxy**
- **Internet**
- **Central Repository**
- **Proxy**
- **Internet**
- **Central Repository**

**<dependency>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j</artifactId>
.VERSION>2.8.2</version>
</dependency>
## Maven Central & Company’s Private Repositories

### Maven Central Repository
- **Known Location**
  - www.maven.org

### Private Repository
- **Proxy**
- **Our Precious Products**
  - Kept in Secret
  - LAN
  - VPN

## Local Repository $HOME/.m2/

### Maven Configurations
- Used Dependencies
- Known Location
  - repository directory
  - settings.xml
  - security-settings.xml
- Stored Locally
- Downloaded Once
- Shared Among Projects

### settings.xml

```xml
<settings xmlns="...">
  <localRepository>+-org/apache/logging
  |   +-log4j/log4j-core
  |     +-2.2
  |       |\log4j-core-2.2.jar
  |       |\log4j-core-2.2.pom
  |     +-2.5
  |       |\log4j-core-2.5.jar
  |       |\log4j-core-2.5.pom
  |     +-2.6.2
  |       |...
  ...
</settings>
```
Project Structures

Table of Contents

1 Overview
2 Interface-based Programming
3 Tools
4 Project Structures

Zsolt Tóth (University of Miskolc)
Component-based Development
2017 25 / 30

No Silver Bullet

- Depends on
  - Company
  - Architect
  - Project Requirements
  - Competence
  - Laziness
  - etc.
- Defined by
  - Software Architects
  - Senior Developers

Should be Considered:

- Functionalities based on
  - Users / Roles
  - Commercial Units
  - Reusability
- Technologies
  - Programming Techniques
  - Programming Languages
- Build and Testings

Zsolt Tóth (University of Miskolc)
Component-based Development
2017 26 / 30

Example Project Structure #1

Notes

Zsolt Tóth (University of Miskolc)
Component-based Development
2017 27 / 30
Example Project Structure #1

util
- Utility Functions
- Logging Configuration
- Do not Fit Elsewhere

model
- Domain Model
- Low Level Validation

persist
- Data Access Object
- Interfaces
- DAO Implementation
- Depends on Technology

service
- Service Definition
- Interfaces

service-impl
- Service Implementation

controller
- Entry Point of the Component
- Validate & Sanitize

Discussion
Pros and Cons?
Why?

Example Project Structure #2

bundle

Core → Service → Web
DAO

Discussion
Pros and Cons?
Why?