Spring Framework – Fundamentals

Web Application Development

Zsolt Tóth

Coeus Consulting

2021

Introduction

Principles

- emphasize the key concepts.
- can answer the question "Why ...?"
- always have to be born in mind.
- affect on everything.

Architectures

- allow the modeling of software systems.
- facilitate the design of big systems.

Architectural Patterns

- give a general solution for frequent problems.

Outline

Basics of Object Oriented Programming

- Class
- Object
- Inheritance
- Polymorphism
- Encapsulation
- Information Hiding
- Message
- Static Type vs Dynamic Type
- Late Binding
- Abstract class vs Interface
- Specialization vs Encapsulation
- Scopes
- Instances

Static vs Dynamic Type – Late Binding

```
interface Shape { double area(); }
class Rectangle implements Shape{
        public double area() { return this.a *
           this.b;  } ... }
class Circle implements Shape{
        public double area(){ return
           Math.pow(this.r,2) * Math.pi;
        } }
// App.java
Collection<Shape> shapes = List.of( new
   Rectangle(4.0,5.0), new Circle(3.0));
shapes.stream().foreach(shape ->
   System.out.println(shape.area()));
```

Dependency Injection

```
class InvoiceChecker{
        private Collection<Rule> rules; // mandatory
        private Optional<NotificationService>
           notifier; //optional
        // Constructor Injection
        public InvoiceChecker(Collection<Rule>
           rules){
                this.rules = rules;
                this.notifier = Optional.empty();
        }
        //Method Injection
        public void setNotifier(NotificationService
           notifier){
                this.notifier = notifier;
        }
}
```

SOLID

- Basic Principles of Object Oriented Design
- Applied Together
- Affect on Design Patterns
- Language Independent
- Robert C. Martin
- Early 2000

Single Responsibility Principle
Open-Close Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle

Single Responsibility Principle

"A class should have only one reason to change."

Consequences

- + Simple classes and methods
- + Easy-to-understand
- + Improve Code Quality
- + Facilitate testing
- - More classes
- - Difficult to stick to it.

While a Button has state, text, icon and callback function,

- it is rendered by the current GUI Engine.
- the event is forwarded to its callback function, when the Button is clicked.

Open-Close Principle

"Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."

Consequences

- Do not override!
- Polymorphism

+ Abstraction
+ Stable behavior
+ Better interface design
- No overriding

```
abstract class Template{

abstract void method1();
abstract void method2();

final void
    templateMethod(){
        method1();
        method2();
        }
}
```

Liskov Substitution Principle

"A variable v should be substituted by any object whose dynamic type is a subtype of the static type of v without changing its desired properties."

- Static types are
  - as abstract as possible.
  - as concrete as necessary.

+ Re-usability

+ Abstractness

- Well-designed class hierarchy required

Collections

- List
- Set
- Map
- Queue
- Stack

Differences?

Interface Segregation Principle

"Provide as small and specific interfaces to the client as they are needed."

+ Abstractness
+ Loose coupling
+ Smaller interfaces

- More interfaces

- Service definition
- Component based design
- Facade

Dependency Inversion Principle

"Abstractions should not depend on details. Details should depend on abstractions."

- Use interfaces
- Ask dependencies, do not instantiate them.

+ Facilitate testing
+ Decoupling
+ Re-usability

- More interfaces to maintain
- Inverts the thinking

* Constructor definition
  * public
  * protected
* Mocking
* Interface and implementation should be separated.
* Database Connectivity

KISS – Keep It Simple, Stupid!

- U. S. Navy
- 1960s
- Simplicity
- Occam’s Razor

In Software Engineering

- Limit of class / method size
- Design specific tools
- Unix commands
- Combine them

Divide and Conquer

Whole is greater than the sum of its parts.

- Politics, Warfare, ...
- Engineering
    - Interior Design
        - Heating system
        - IKEA Furniture
    - Mechanical Parts
        - Cogs, Pulley
    - Circuits
        - AD/DA Converter
        - Amplifiers
    - Software Components
        - Subsystems
        - DBMSs
    - Algorithms
        - Parallelism

- More functions
    - Categorize - Partitioning
    - Subsystems
- More complex
    - Analyze - Define Steps
    - More simple, easier
    - Still could be complex
- Recursion
- Replaceable parts
    - Maintenance
- Assembly
    - Additional Cost
    - Administration
    - Quality Assurance

Pareto Principle

For many events, roughly 80% of the effects come from 20% of the causes.

- 80/20 rule
- Law of the vital few
- Economy & Business
- Software Engineering
    - Bug fixing
    - Load testing
    - Optimization

Software Projects

- Easy to start
    - Let's rewrite!
- Hard to finish
    - Testing, Documentation, ...
- Usually unfinished
- "It satisfies industrial needs."
    - Full of bugs.
    - Unprofessional work.
    - Poor design.
    - **But someone pays for it.**

Boy Scout Rule

Always leave the camp ground cleaner than you found it.

- Continuous Refactoring
- Take care of
  - your code
  - its context
- Small Steps
  - rename a variable
  - outsource a function
  - eliminate a magic number
  - etc.

Technical Debt

- Easy, limited solutions
  - time constraints
  - uncertain requirements
  - knowledge, experience
- Effects
  - slower development
  - difficult testing
- Clean Up Sprint

F.I.R.S.T.

- Fast
- Isolated
- Repeatable
- Self–validating
- Through

- Testing
  - Unit,
  - Component,
  - Integration
- Implementation Pattern
  - given–when–then
  - arange–act–assert

Brook's Law

Adding manpower to a late software project makes it later"

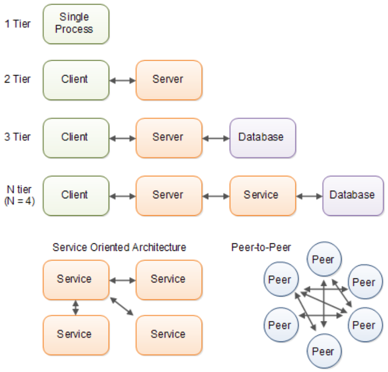
- Project Management
- Takes time to become productive
- Communication overhead $O(n^2)$
- The mythical man-month

- Agile
  - Test Driven Development
  - Scrum
- Sprint 2-4 weeks
- Team ≈8 member

Outline

Software Architectures

- Abstract structure of Software Systems
- Independent of
  - Platform
  - Programming Language
- Depend on
  - Project Goal
    - Purpose of the System
    - Scalability
  - Target Users
    - 1, 100, 10.000, ...
    - Expected usage?
    - Expected load?



Monolithic Architecture

- Only one process
- Installed on a single computer
  - Mainframes
  - Desktop Applications
- Unix commands
  - `ls, ln, ps, mkdir, grep, chmod, chown`
- Computer Games, Word processors
- Off-line work
- Installation
- Complex computations
- Our first programs

+ Simple, easy-to-understand the Architecture

+ Independent from other applications

+ Self-contained

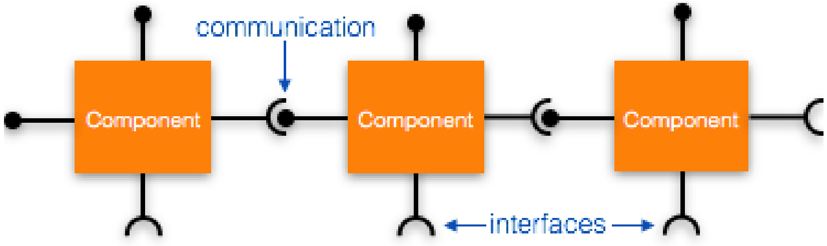
- Unmaintainable Application

- No Modularity

Component Based Design

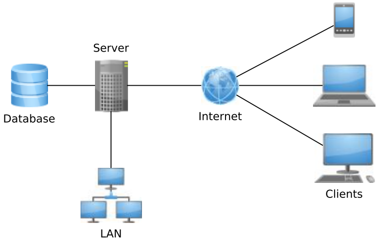
- Independent development unit
- A part of the system
- Provide service via its interface
- Use other components

- Tests
  - Unit
  - Component
  - Integration
- Dependencies
- Build process
- Deployment



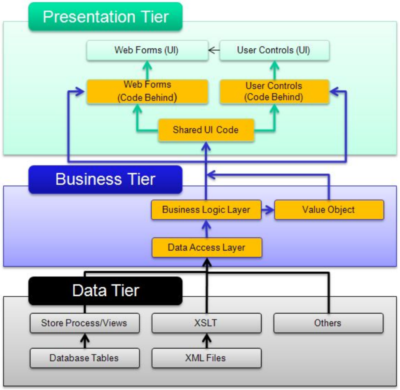
Client - Server Model

- Simple Model
  - Clients request services
  - Server waits for and serves requests
- Widely used
  - FTP, SSH
  - WWW, SMTP
- Web Applications
  - Information Systems
  - Search Engines
  - Social Media
  - Web Shops
  - e–Government
  - e–Banking
  - Monitoring Systems



n-Tier Architecture

- Detailed than Client–Server Model
- Tiers are not Layers
- Tiers have specific functions, purpose
- Typical tiers
  - Presentation
    - Client–side
    - Web sites
    - Mobile / Desktop applications
  - Business
    - Server–sidHeves Megyei Természetbarát Szövetsége
    - Business logic
  - Database



Service Oriented Architecture

- Collection of Services
- Service - Service communication
  - Simple data passing
  - Coordinating some activity
- Solutions
  - CORBA
  - DCOM
  - Web Services
    - WSDL, SOAP
    - REST

A service

- represents an activity.
- is self-contained.
- is black box.
- may use other services.

Micro services

- Micro service is a process.
- Implementation of SOA.
- Popular since 2014.
- Characteristics
    - Fine-grained
    - Cloud applications
    - Continuous delivery
- Reusability of services.

EMailService

- Email notifications are required in many business activities.
- Each email has the same structure.
    - sender, receiver addresses
    - subject
    - content (generated by other services)

Outline

Model View Controller

- Separation of value and appearance
- Desktop applications
- Web applications
- Variants
  - Model View Presenter
  - Model View ViewModel

Model

- Domain objects
- State

View

- Graphical appearance
- Visualization

Controller

- Connection between model and view
- Refresh the View
- User interactions
- Event handlers

Data Access Object

- Abstract interface for persistence storage
- Separation of persistence and business logic
- Storage Independent
- Facilitates usage of different DBMSs
- Defines expected behavior
- Multiple specific implementations

Typical methods

- Create
- Read
- Update
- Delete

Data Transfer Object

- Messaging between systems.
- Separates
  - internal domain objects
  - external environment
- Hides sensitive data
- Reduce network traffic
- Request - Response Objects
- Marshalling

```
{
"coordinate": {
"x": 0, "y": 0, "z": 0
},
"zone": {
"id":
   "00000000-0000-0000",
"name": "Unknown"
},
"uuid":
   "2ee1f927-50af-45c9"
}
```

See Also

- Design Patterns
  - Creation
    - Singleton
    - Factory Method
    - Prototype
  - Structural
    - Adapter
    - Decorator
    - Composite
    - Proxy
    - Bridge
  - Behavioral
    - Strategy
    - Template Method
    - State
    - Chain of Responsibility
- HTTP – CRUD Mapping
- System Integration
  - Synchronous
  - Asynchronous
    - postback
  - Message Driven
    Publish – Subscribe
    - message queue
    - Kafka, Redis, RabbitMQ