Architectural patterns

Alexandru Olteanu

Universitatea Politehnica Bucuresti
Facultatea de Automatică si Calculatoare, Departamentul Calculatoare
alexandru.olteanu@upb.ro

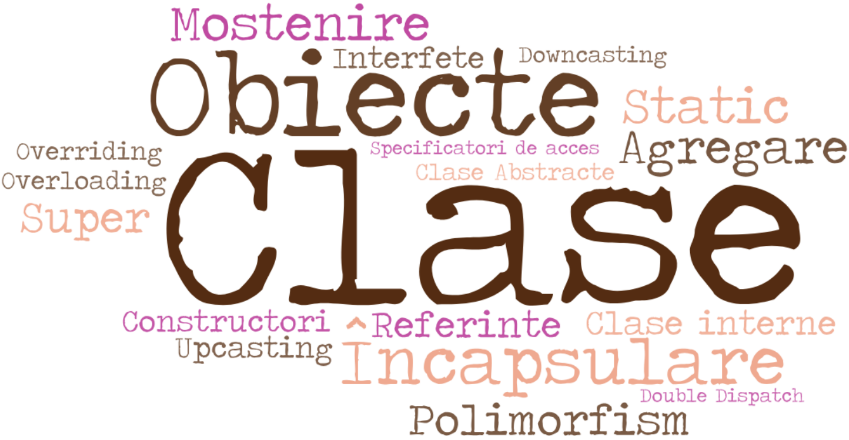
OOP, 2020

Computer Science & Engineering Department

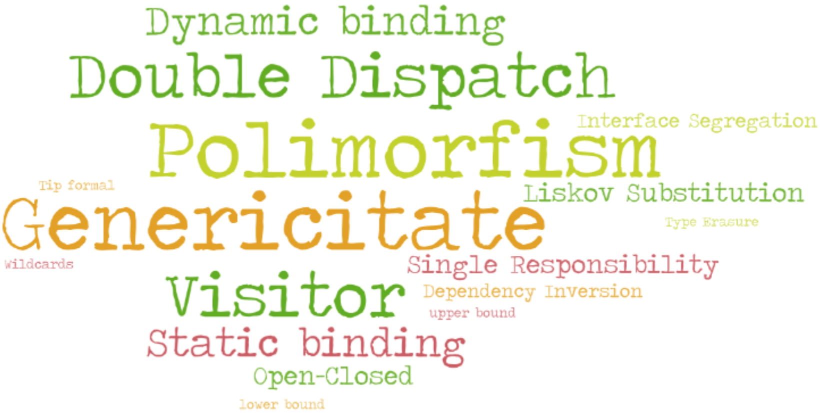
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

Universitatea Politehnica Bucureşti

Partea 1: Bazele teoretice ale OOP

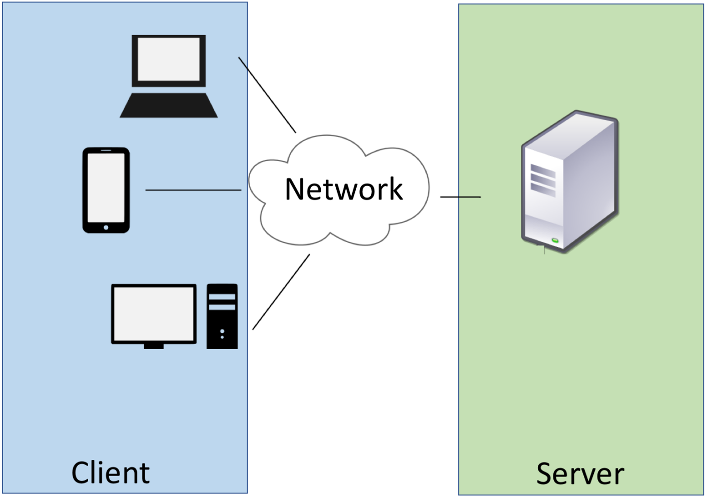


Partea a 2-a: Notiuni avansate de OOP



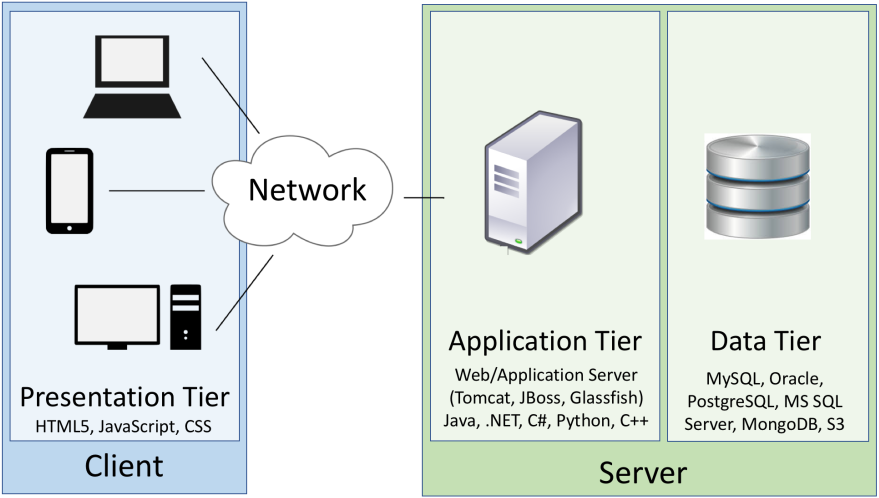
Multitier Architecture

Client-server model

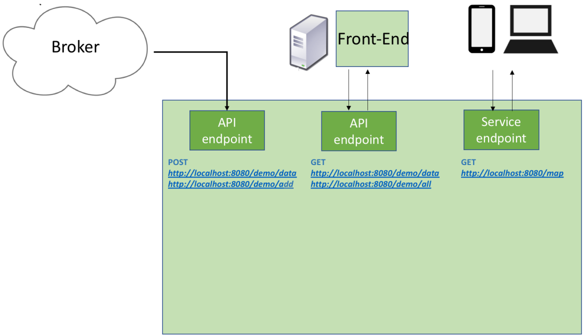


as opposed to peer-to-peer model

Three-Tier Architecture



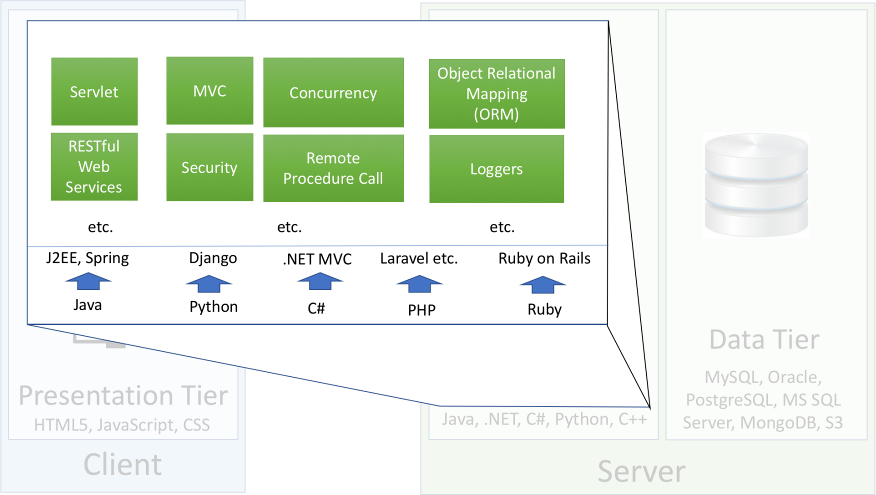
Three-Tier Architecture


Broker
Front-End
API endpoint
POST
http://localhost:8080/demo/data
http://localhost:8080/demo/add
API endpoint
GET
http://localhost:8080/demo/data
http://localhost:8080/demo/all
Service endpoint
GET
http://localhost:8080/map

- Reliability and Availability
- Scalability
- Extensibility and Maintanibility

▸ More info

Frameworks


Servlet
MVC
Concurrency
Object Relational Mapping (ORM)
RESTful Web Services
Security
Remote Procedure Call
Loggers
etc.
etc.
etc.
J2EE, Spring
Django
.NET MVC
Laravel etc.
Ruby on Rails
Java
Python
C#
PHP
Ruby
Presentation Tier
HTML5, JavaScript, CSS
Client
Java, .NET, C#, Python, C++
Data Tier
MySQL, Oracle, PostgreSQL, MS SQL Server, MongoDB, S3
Server

Frameworks, Inversion of Control, Dependency Injection

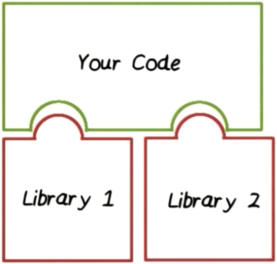
Frameworks

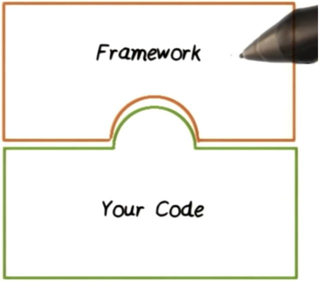
A software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software

Inversion of Control

Definition

Inversion of Control is a design principle in which custom code you write receives flow of control from a generic framework (as opposed to a 'traditional' architecture where custom code you write is the one that calls into reusable libraries)

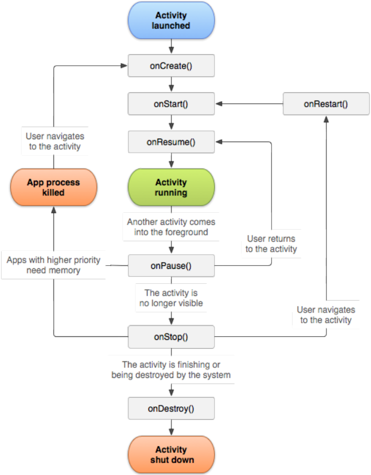




▸ Inversion of Control Youtube video

Inversion of Control in action

Android Studio > New Project > Basic Activity


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity Lifecycle Callbacks

Inversion of Control: implementation techniques

Many implementation techniques, relying heavily on design patterns:

- Service Locator pattern
- Dependency Injection
- Contextualized lookup
- Template Method pattern
- Strategy pattern

IoC in Android: Dependency Injection and Service Locator

Dependency Injection

Definition

Dependency Injection is a design pattern that:

- requires custom classes to link to Dependencies through setters or constructors (instead of instantiating with new)
- allows frameworks to inject proper implementations to those Dependencies (aka autowiring in some frameworks)

Dependency Injection vs Dependency Inversion?

Spring Boot

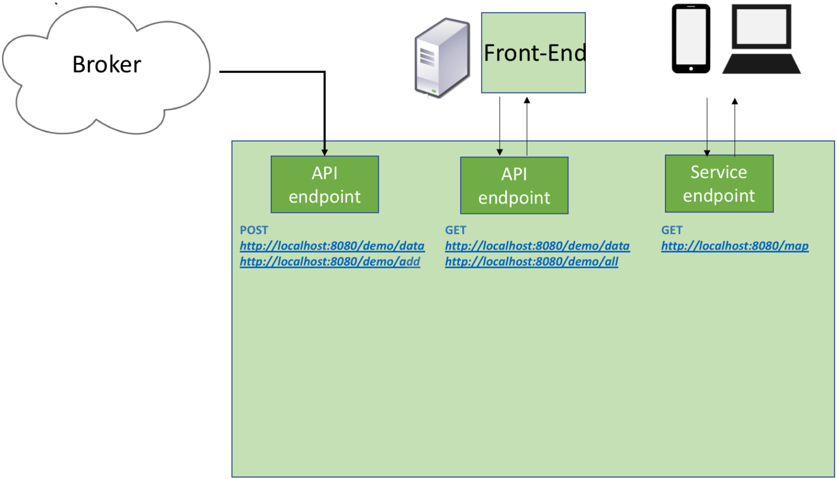
Definition

A Spring bean is basically an object managed by Spring. More specifically, it is an object that is instantiated, configured and otherwise managed by a Spring Framework container. Spring beans are defined in Spring configuration files (or, more recently, with annotations), instantiated by Spring containers, and then injected into applications.

Note that Spring beans need not always be JavaBeans. Spring beans might not implement the java.io.Serializable interface, can have arguments in their constructors, etc.

Application architecture: MVC, FrontController

Application architecture: endpoints


Broker
Front-End
API endpoint
API endpoint
Service endpoint
POST
http://localhost:8080/demo/data
http://localhost:8080/demo/add
GET
http://localhost:8080/demo/data
http://localhost:8080/demo/all
GET
http://localhost:8080/map

MVC

Model-View-Controller (MVC) is an architectural design pattern that deals with separation of concerns, splitting the code in three components:

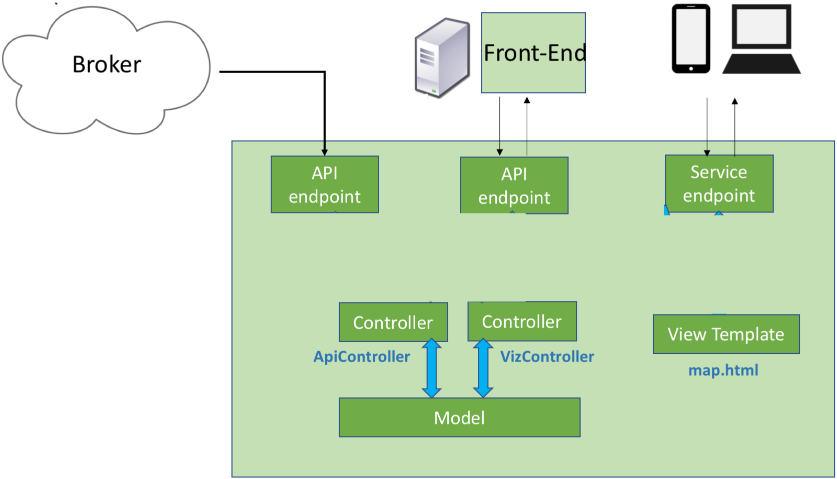
- Model: data, state, business logic
- View: representation of data (usually UI, but nowadays JSON/XML may be interpreted as views)
- Controller: the logic for reacting to user interaction and model changes

Tweaked over the years to accommodate various technologies, improve testability etc.

Pentru aprofundarea subiectului

▸ MVC vs. MVP vs. MVVM, Niraj Bhatt ▸ MVC vs. MVP vs. MVVM on Android, Eric Maxwell

Application architecture: model, view, controller

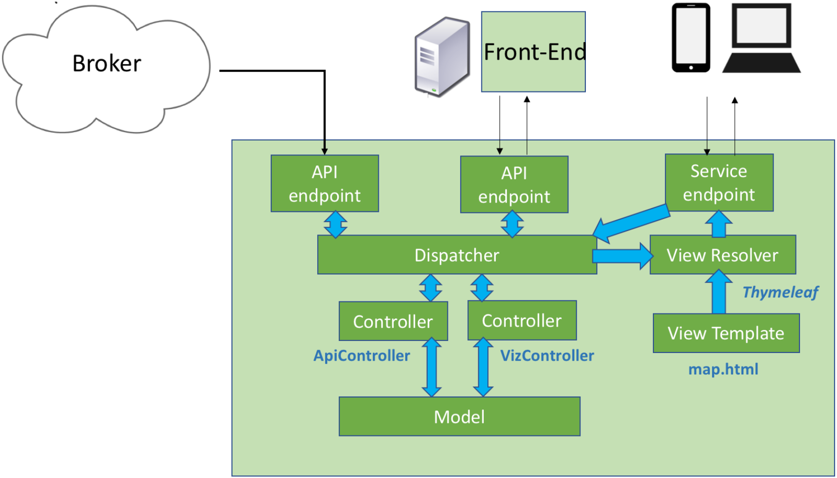


MVC vs Front Controller

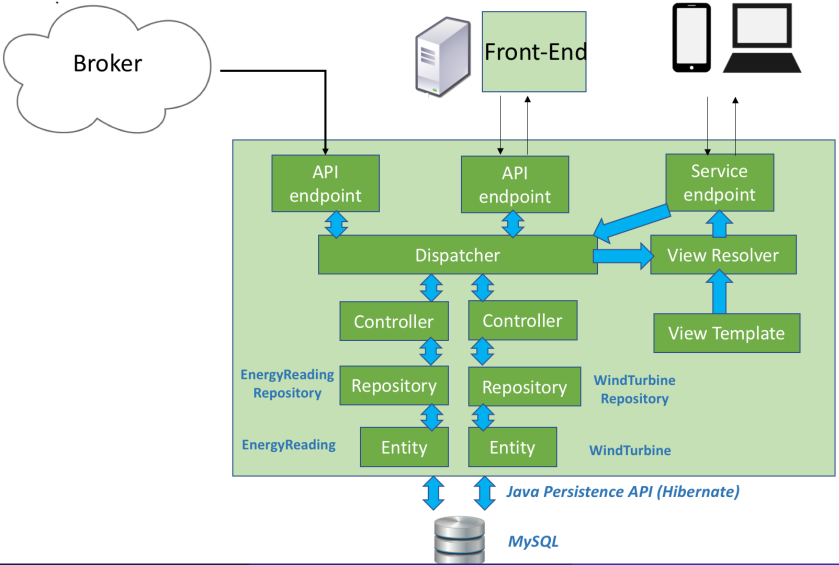
Front Controller is a twist on a typical MVC designed for large web applications: there is a main controller that dispatches actions on various controllers:

▸ Quick Guide to Spring Controllers

Application architecture: connection from endpoints - dispatcher resolver



Application architecture: connection to persistence - entity, repository



Entities

Definition

Java Persistence API (JPA) Entities are classes specifically designated by the programmer whose nontransient fields should be persisted to a relational database using the services of an entity manager obtained from a JPA persistence provider.

Entities instances are POJOs.

DTOs

Definition

Data Transfer Objects (DTO) is a very simple object meant to carry data between processes, without any behavior (except for serialization, storage and retrieval).

Somewhat similar to struct in C.

Repository pattern

The Repository design pattern provides an abstraction of data, so that your application can work with a simple abstraction that has an interface approximating that of a collection.

Adding, removing, updating, and selecting items from this collection is done through a series of straightforward methods, without the need to deal with database concerns like connections, commands, cursors, or readers.

▸ Repository Pattern - A data persistence abstraction

Repository implementation

In the Repository Per Entity implementation: create a new Repository implementation for each business object you need to store to or retrieve from your persistence layer.

- Advantage: YAGNI - not implementing methods that are not needed
- Disadvantage: class explosion

Cunostinte avansate despre Repository:

- Common Mistakes with the Repository Pattern
- Why shouldn't I use the Repository Pattern with Entity Framework

De citit

- Difference between dependency injection and dependency inversion
- MVC vs. MVP vs. MVVM, Niraj Bhatt
- Repository Pattern - A data persistence abstraction
- Why shouldn't I use the Repository Pattern with Entity Framework