# ENTERPRISE PATTERNS [EP]

Problems of Enterprise System:

* Complexity of system
* Continuously growing systems
* Monolitic
* Geterogennost - Too many frameworks
* Integrate one system to anther can be also challengable

# DECOMPOSITION

**Decomposition** – means split something into small pieces

# PATTERNS

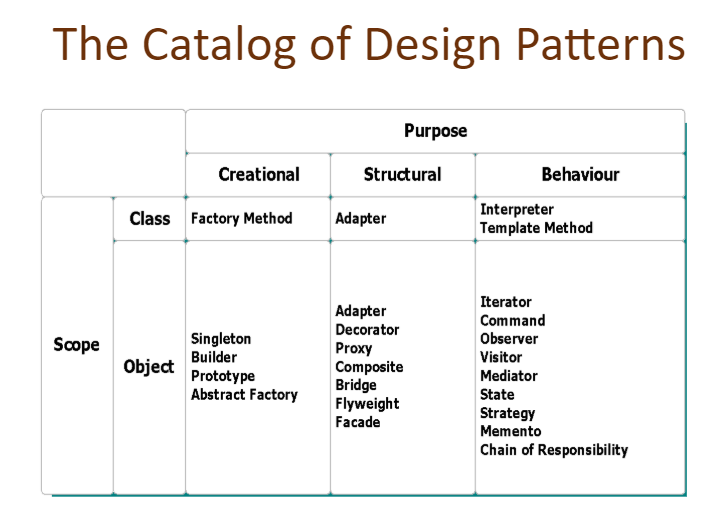
# OVERVIEW

Christopher Alexander was architecture (building)). He published a book about architecture pattens in building. It contained list of approaches to solve common problems

Four people from software took the idea of Christopher Alexander to use it in programming language [SmallTalk].

There are patterns

* GoF (gang of Four) = 24 patterns
* GRASP – actually it’s not pattern. It’s a list of principles



# ENTERPRISE SYSTEM

How build system, steps

1. Decompose model in UML
2. Layer of Business logic
3. Layer of DB
4. UI

There are 3 styles to build business logic based on:

* **Transact script**
* **Table model**
* **Domain model**

# TRNSACT SCRIPT MODEL APPROACH

**Transact script** – is a description of business rules ib **procedure** style ([if.. then ..else] )

# TABLE MODEL APPROACH

Table model – is approach to build system, based on table style. It requires ,first of ll, define structure of data. In contrast, Java developers works in OOP style.

It does not assume that it will have service layer. Business logic stores in DB. For example, stored procedures. The most popular is in banking, financial companies

* Presentation layer
* DAO layer
* DB

Note: UI directly access DAO

Cons: UI directly access DAO. It means difficult make changes

# DOMAIN MODEL APPROACH

Domain model – is used in those cases when business logic is the most difficult. The most suitable for enterprise

**Simple domain model** – model without behavior (when use framework). It has only state

**Rich domain model** – it has all behavior, it has all methods that describe those behaviour

Note: *Object of Domain model might not match with table in DB* .

# SERVICE LAYER

If we use [simple rich model] we don’t have a place where to keep [business logic]. Therefore we add new layer [**service layer**]

# LAYERS

History of layers:

* [< 90 years] - There were not layers. [Transaction script] (procedure style) covered all tasks
* [90s years] - client server
* [2000s years] – blooming of OOP. 3 layer application

-user interface

-domain layer

-data source

# 3 MAIN LAYERS

3 main layers is model of Martin Fowler

3 main layers:

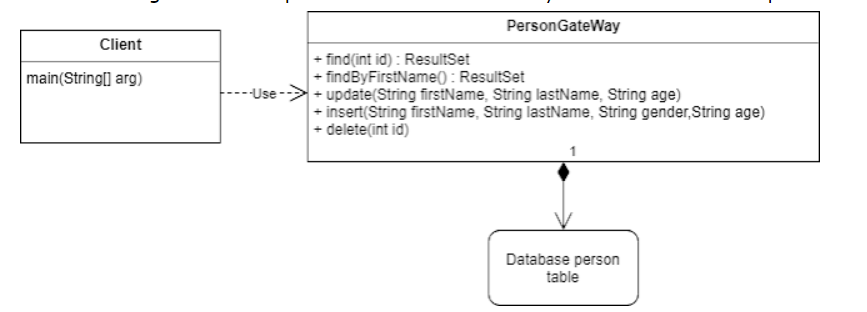
* **Presentation** –
* **Domain Logic** –
* **Data Source** –

# DAO LAYER

**DAO patterns:**

**Data Accessor** - is primitive patterns. It consists of SQL, connections. For example, method [getAllStudents() – is supposed to be in this layer]

**Table Data Gateway** – design pattern. Object acts as [gateway] for table.is just simple interface to implement CRUD. It’s build on top of table



# RESOURCE PATTERNS

RESOURCE PATTERNS:

* **Resource decorator** – it is enterprise implementation of GoF pattern [decorator]
* **Resource pool** – can be as [connection pool]
* **Resource timer** – it closes resource in the case of timeout
* **Resource descriptor** -
* **Retryer** - it repeats operations until it works ( re-try)

# INPUT AND OUTPUT PATTERNS

**SELECTION PATTERN**

search by example( user clicked check boxes on a form). The problem is

there are many filter search in your app

findByDateStudentTeacher

findByMonthLesson

.....

And you ca not provide queries for all possible combinations. You need to use this pattern

idea

you have object that extends you all filters and you apply them (down to DAO layer)

**UPDATE FACTORY**

Similar to [selection factory]

# ORM (OBJECT RELATIONAL MAPPING ) PATTERNS

ORM – when you map objects of DAO layer to Domain layer. For example, a few table with joins can be mapped to one object in Domain



**BEHAVIORAL PATTERNS**

**Unit of Work** – it’s implemented inside of Hibernate. It works like “transaction”. You create new object or change or delete – it saves changes in DB

Identity Map –

Lazy Load -