Chapter 5:
OBJECT-ORIENTED DESIGN PATTERN

By:
Ts. Azaliza binti Zainal

OBJECT-ORIENTED DESIGN PATTERNS

- Design patterns provide a means for capturing knowledge about problems and successful solutions in systems design.
- A design pattern codifies design decisions and best practices for solving a particular design problem according to design principles.
- Design patterns are not the same as software libraries; they are not packaged solutions that can be used as is. Rather, they are templates for a solution that must be modified and adapted for each particular use.

OBJECT-ORIENTED DESIGN PATTERNS

- Design pattern is a general repeatable solution to a commonly-occurring problem in software design.
- Design patterns are recurring solutions to design problems.
- Studying design patterns is a way of studying how the "experts" do design.
- Design patterns constitute a set of rules describing how to accomplish certain tasks in the realm of software development.

OBJECT-ORIENTED DESIGN PATTERNS

- Patterns capture the static and dynamic structure and collaboration among key participants in software designs
- Especially good for describing how and why to resolve nonfunctional issues
- Patterns facilitate reuse of successful software architectures and designs.

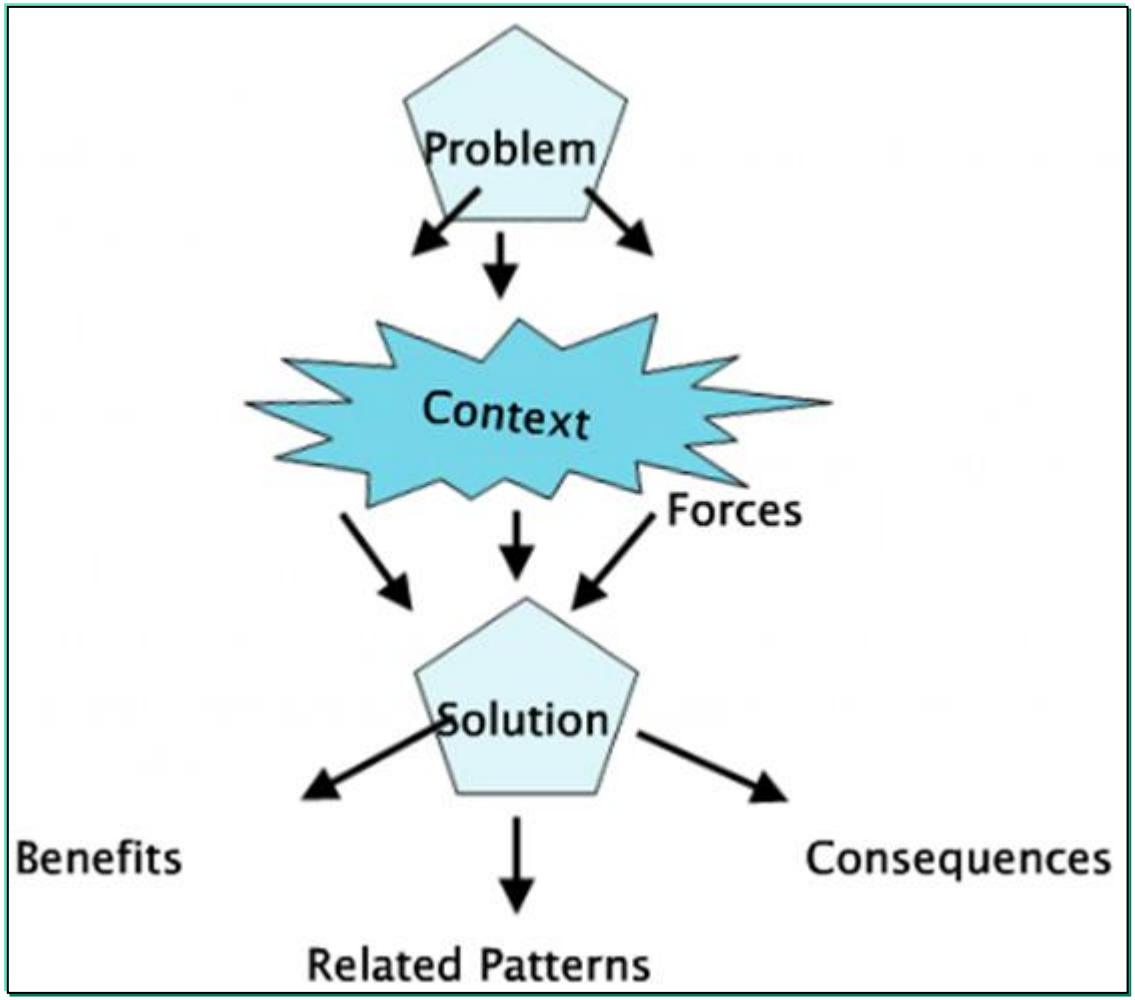
ORIGIN OF DESIGN PATTERNS

"Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it in the same way twice"

- **Christopher Alexander, A Pattern Language, 1977**

  Context: City Planning and Building architectures

HOW PATTERN ARIES



ARCHITECTURAL PATTERN CLASSIFICATION

- The choice of applying architectural patterns for designing some architectural element depends on the particular system type, requirements, and desired quality attributes.
- These characteristics help guide the choice of selecting one particular pattern over another.

TYPE OF SOFTWARE SYSTEMS FOR CLASSIFYING ARCHITECTURAL PATTERNS

Type of Software Systems for Classifying Architectural Patterns

Type	Description
Data-centered	Systems that serve as a centralized repository for data, while allowing clients to access and perform work on the data
Data flow	Systems oriented around the transport and transformation of a stream of data
Distributed	Systems that primarily involve interaction between several independent processing units connected via a network
Interactive	Systems that serve users or user-centric systems
Hierarchical	Systems where components can be structured as a hierarchy (vertically and horizontally) to reflect different levels of abstraction and responsibility

DESIGN PATTERN vs FRAMEWORK

Design Pattern	Framework
A design pattern is a specification.	A framework is a product.
Design pattern is part of the system design.	A framework is part of the system.
Design pattern is a proven way to solve a problem programmatically.	Framework is a set of well designed components with the help of which applications can be built upon.
Pattern is a subset of framework	Framework is/can be collection of patterns with implementation.

SOME MORE DIFFERENCES

- **A framework embodies a complete design of an application, while a pattern is an outline of a solution to a class of problems.**
- **A framework can be viewed as the implementation of a system of design patterns.**
- **However patterns:**
  - –are more abstract and general than frameworks
  - –are smaller architectural elements than frameworks
  - –are less specialized than frameworks

BENEFITS

- **Design Pattern:**
  - They provide users with a way to solve issues related to software development using a proven solution.
  - Its solutions facilitates the development of highly cohesive modules with minimal coupling.
  - Design patterns helps to improve developer communication.

- **Framework:**
  - Easy access to a list of pre-qualified suppliers for non-permanent workers that meet a predefined standard of service.
  - Quality candidates that can be provided at short notice and deliver high performance.

TYPES OF DESIGN PATTERN

- Creational Patterns
  - **Deal with initializing and configuring classes and objects.**
- Structural Patterns
  - **Deal with decoupling interface and implementation of classes and objects.**
  - **Composition of classes or objects.**
- Behavioral Patterns
  - **Deal with dynamic interactions among societies of classes and objects**
  - **How they distribute responsibility**

WHEN TO USE PATTERNS?

- **Solutions to problems that recur with variations**
  - No need for reuse if problem only arises in one context
- **Solutions that require several steps:**
  - Not all problems need all steps
  - Patterns can be overkill if solution is a simple linear set of instructions
- **Solutions where the solver is more interested in the existence of the solution than its complete derivation**
  - Patterns leave out too much to be useful to someone who really wants to understand
  - They can be a temporary bridge

WHAT MAKES IT A PATTERN?

A Pattern must:

- Solve a problem and be useful
- Have a context and can describe where the solution can be used
- Recur in relevant situations
- Provide sufficient understanding to tailor the solution
- Have a name and be referenced consistently

BENEFITS OF DESIGN PATTERNS

- Design patterns enable large-scale reuse of software architectures and also help document systems
- Patterns explicitly capture expert knowledge and design tradeoffs and make it more widely available
- Patterns help improve developer communication
- Pattern names form a common vocabulary
- Patterns help ease the transition to OO technology

DRAWBACKS TO DESIGN PATTERNS

- Patterns do not lead to direct code reuse
- Patterns are deceptively simple
- Teams may suffer from pattern overload
- Patterns are validated by experience and discussion rather than by automated testing
- Integrating patterns into a software development process is a human-intensive activity.

SUGGESTIONS FOR EFFECTIVE USE

- Do not recast everything as a pattern
- Instead, develop strategic domain patterns and reuse existing tactical patterns
- Institutionalize rewards for developing patterns
- Directly involve pattern authors with application developers and domain experts
- Clearly document when patterns apply and do not apply
- Manage expectations carefully.

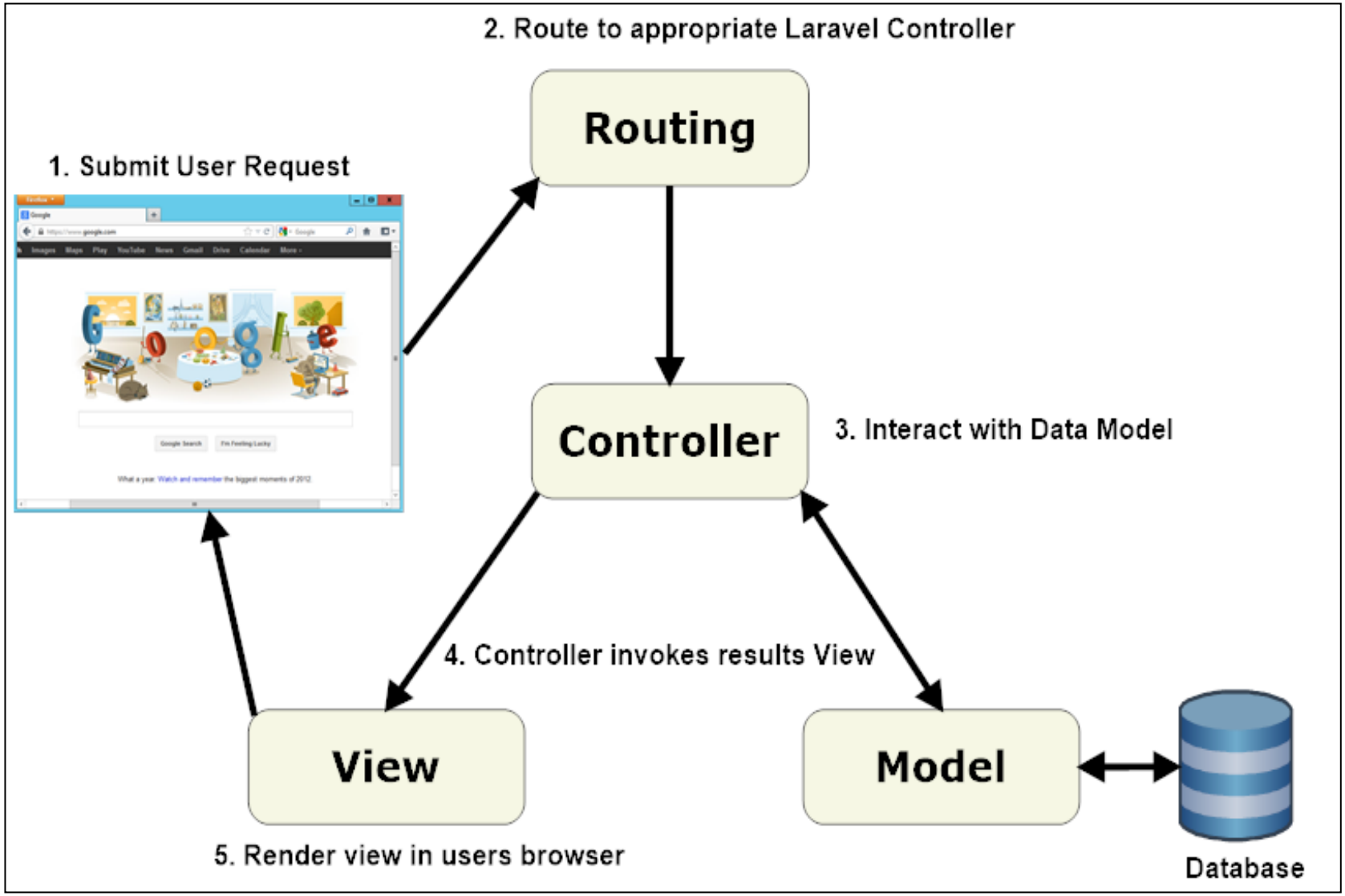
MVC (MODEL – VIEW – CONTROLLER)

- Model–View–Controller (usually known as MVC) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts.
- This is done to separate internal representations of information from the ways information is presented to and accepted from the user.
- The MVC design pattern decouples these major components allowing for code reuse and parallel development.

MVC (MODEL – VIEW – CONTROLLER)

- Traditionally used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications.
- Popular programming languages like JavaScript, Python, Ruby, PHP, Java, and C# have MVC frameworks that are used in web application development straight out of the box.

MVC (MODEL – VIEW – CONTROLLER)



MVC (MODEL – VIEW – CONTROLLER)

- **Organizing the code in the website by using MVC design pattern.**
- **The goal is to divide the project into three big parts:**
  - Model: interacts with the database. It receives, stores and retrieves data for the user.
  - View: displays information to the user and integrates data from the controller.
  - Controller: sends and receives data from the model and passes to the view.

MVC (MODEL – VIEW – CONTROLLER)

- Separate business logics and presentation layer (user interface).
- code cleaner and extensible.
- **Models** : data structures, database.
- **Views** : page templates and output.
- **Controllers** : page requests, bind everything

MVC (PHP example)

AbcProfilesTable.php - cakep368 - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

EXPLORER

OPEN EDITORS
- app.php config
- AbcProfilesTable.php src\Model\Table

CAKEP368
- bin
- config
  - schema
  - .env.default
  - app.default.php
  - app.php
  - bootstrap_cli.php
  - bootstrap.php
  - paths.php
  - requirements.php
  - routes.php
- logs
- plugins
- src
  - Console
  - Controller
  - Model
    - Behavior
    - Entity
    - Table
      - AbcDivisionsTable.php
      - AbcProfilesTable.php
  - Shell
  - Template
  - View
  - Application.php
- tests
- tmp
- vendor
- webroot
- .editorconfig
- .htaccess
- .travis.yml
- composer.json
- composer.lock
- index.php
- phpunit.xml.dist
- README.md

app.php | AbcProfilesTable.php

```php
     * Initialize method
     *
     * @param array $config The configuration for the Table.
     * @return void
     */
    public function initialize(array $config)
    {
        parent::initialize($config);

        $this->setTable('abc_profiles');
        $this->setDisplayField('name');
        $this->setPrimaryKey('id');

        $this->addBehavior('Timestamp');

        $this->belongsTo('AbcDivisions', [
            'foreignKey' => 'abc_division_id',
            'joinType' => 'LEFT'//'INNER'
        ]);
    }

    /**
     * Default validation rules.
     *
     * @param \Cake\Validation\Validator $validator Validator instance.
     * @return \Cake\Validation\Validator
     */
    public function validationDefault(Validator $validator)
    {
        $validator
            ->integer('id')
            ->allowEmpty('id', 'create');

        $validator
            ->scalar('name')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Looking for git in: C:\Program Files\Git\cmd\git.exe
Looking for git in: C:\Program Files (x86)\Git\cmd\git.exe
Looking for git in: C:\Program Files\Git\cmd\git.exe
Looking for git in: C:\Users\User\AppData\Local\Programs\Git\cmd\git.exe
Git installation not found.
```

Q & A

THANK YOU FOR YOUR ATTENTION

☺