JAVA BOOTCAMP

DESIGN PATTERNS

DESIGN PATTERNS

TOPICS

- What are design patterns?
- Why we use design patterns
- Three main design patterns

WHAT ARE DESIGN PATTERNS?

WHAT ARE DESIGN PATTERNS

WHAT IS A DESIGN PATTERN?

- Design patterns are predefined solutions to general software development problems.
- Design pattern solutions represent best practices
- Design patterns are tried and tested solutions developed by experienced object-oriented developers

WHY USE DESIGN PATTERNS?

STANDARDIZE SOLUTIONS

A standard solution for a particular scenario allows developers to build an application around a commonly know pattern

WHY USE DESIGN PATTERNS?

BEST PRACTICES

Design patterns have been developed and tested by experienced developers over a long period of time and increases the speed and ease of software development of less experienced developers

THREE DESIGN PATTERN TYPES

CREATIONAL PATTERNS

Provides a way to create Java objects without directly using the new operator

STRUCTURAL PATTERNS

Defines new ways to compose objects to obtain new functionality

THREE DESIGN PATTERN TYPES

BEHAVIORAL PATTERNS

Define ways to communicate between objects

J2EE PATTERNS

Design patterns for the presentation tier in J2EE applications

THREE MAIN DESIGN PATTERNS

DESIGN PATTERNS

There are many design patterns but the main three you need to know are:

- Factory
- Singleton
- MVC

THREE MAIN DESIGN PATTERNS

DESIGN PATTERNS

Additional Design Patterns not covered that you may encounter

- Adapter Design Pattern
- Composite Pattern
- Template Method Pattern

FACTORY DESIGN PATTERN (CREATIONARY)

DEFINITION

“The Factory pattern provides a way to use an instance as a object factory.The factory can return an instance of one of several possible classes (in asubclass hierarchy), depending on the data provided to it.”

FACTORY DESIGN PATTERN IMPLEMENTATION

IMPLEMENTATION

1. Create an interface
2. Create concrete classes implementing the interface
3. Create a factory to generate an object based on a given parameter
4. Use a factory to get an object of the class by passing in the requested parameter type

FACTORY DESIGN PATTERN EXAMPLES I

CREATE INTERFACE

Java.util.Calendar

NumberFormat getInstance()

FACTORY DESIGN PATTERN EXAMPLES I

CREATE INTERFACE

```
public interface Shape {void draw();}
```

FACTORY DESIGN PATTERN EXAMPLES I

CREATE CONCRETE CLASSES

```
public class Rectangle implements Shape {
  @Override
  public void draw() {
    System.out.println("Inside Rectangle::draw() method.");
  }
}
```

FACTORY DESIGN PATTERNS EXAMPLES II

CREATE FACTORY

```
public class ShapeFactory {
  //use getShape method to get object of type shape
  public Shape getShape(String shapeType){
    //Code to create different object types
  }
}
```

FACTORY DESIGN PATTERNS EXAMPLES II

USE FACTORY

```
//get an object of Circle and call its draw method.
Shape shape1 = shapeFactory.getShape("CIRCLE");

//call draw method of Circle
shape1.draw();

//get an object of Rectangle and call its draw method.
Shape shape2 = shapeFactory.getShape("RECTANGLE");

//call draw method of Rectangle
shape2.draw();
```

SINGLETON DESIGN PATTERN(CREATIONAL)

DEFINITION

“The Singleton pattern provides the possibility to control the number of instances (mostly one) that are allowed to be made. We also receive a globalpoint of access to it (them).”

SINGLETON IMPLEMENTATION

IMPLEMENTATION

1. Create an Singleton class
2. Create Singleton static object declaration
3. Make a private constructor
4. Create a static get instance method

SINGLETON EXAMPLES

Create an Singleton class

public class SingleObject {}

Create static Singleton static object declaration

private static SingleObject instance = new SingleObject();

SINGLETON EXAMPLES

Make a private no argument constructor

private SingleObject(){}

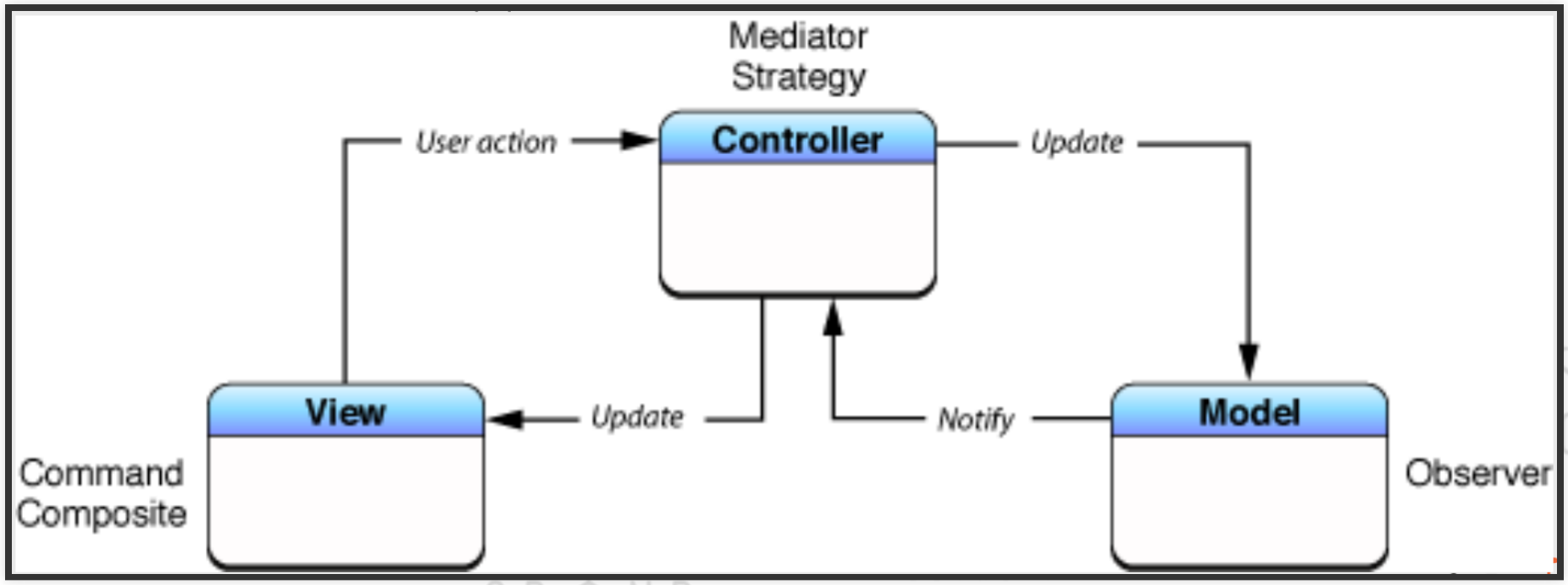
Create a static get instance method

public static SingleObject getInstance(){return instance;}

MVC DESIGN PATTERN

DEFINITION

The Model-View-Controller pattern is used to separate the layers of an interactive (Web) Application



EXAMPLE

View - HTML/JSP

Data from a database - Model

Business logic - Controller

MVC DESIGN PATTERN

MODEL

Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

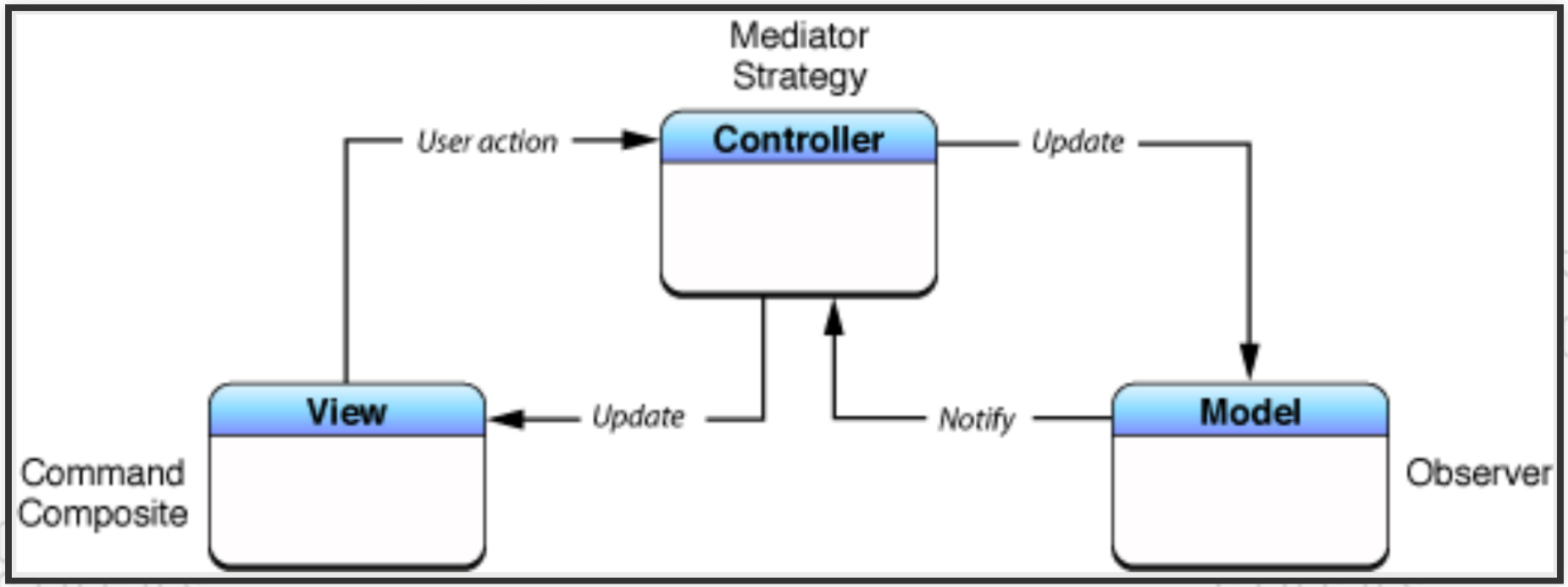
VIEW

View represents the visualization of the data that model contains

MVC DESIGN PATTERN

CONTROLLER

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.



EXAMPLE

View - HTML/JSP

Data from a database - Model

Business logic - Controller

MVC IMPLEMENTATION

IMPLEMENTATION

1. Create a model
2. Create a view
3. Create controller
4. Use the controller to demonstrate MVC usage

MVC DESIGN PATTERN EXAMPLE

I

CREATE MODEL

```
public class Student {
  private String rollNo;
  private String name;
  //code getters and setters
}
```

MVC DESIGN PATTERN EXAMPLE

I

CREATE VIEW

```
public class StudentView {
  public void printStudentDetails(String studentName, String studentRollNo){
    //view code here
  }
}
```

MVC DESIGN PATTERN EXAMPLE

I

CREATE CONTROLLER

```
StudentController controller = new StudentController(model, view);
```

USE CONTROLLER

```
//update model data
controller.setStudentName("John");
controller.updateView();
```

RECAP

RECAP

WHAT YOU SHOULD KNOW

- What is a design pattern.
- Know when to use a design pattern.
- Know three main design patterns.
- How how to implement the three main design patterns.