دانشگاه شهید بهشتی ۱۳۳۸

Design Pattern

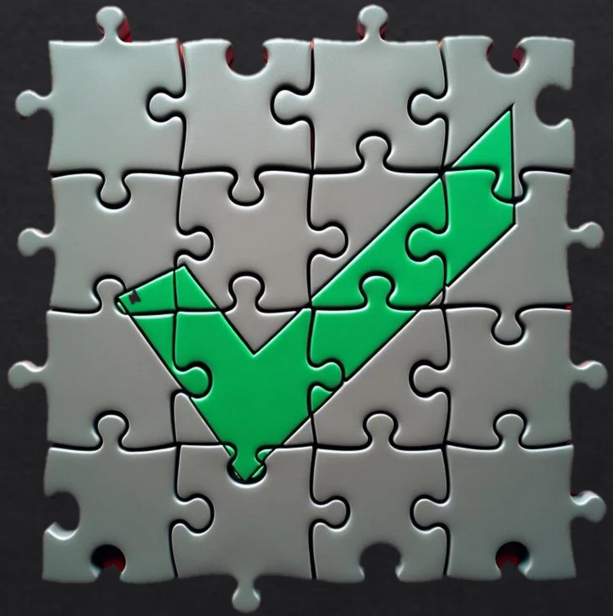
MVC
SINGLETON

Presented by Amir Hossein Ashrafian

AP Spring 1404 - Dr. Mojtaba Vahidi Asl

What are Design Patterns

- Design patterns are standardized solutions to common problems in software design.
- They act as templates that can be applied in various situations to solve design issues.
- Design patterns provide reusable and efficient approaches.
- Instead of specific code, they offer a general concept adaptable to different programming contexts.



Advantages of Design Patterns

- ✓ **Reusability**: Design patterns provide **proven solutions** that can be reused across different projects, preventing the need to reinvent the wheel.

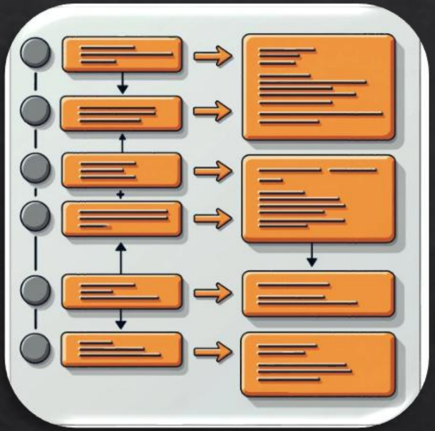
- ✓ **Best Practices**: They encapsulate best practices and common solutions identified by experienced developers, making it easier to solve recurring design problems.

- ✓ **language-independent**: Design patterns can be implemented in any programming language.

Design patterns fall into 3 main categories

Creational

1



Structural

2

Behavioral

3

Creational

- These patterns focus on the process of **object creation.**

- They emphasize methods that make object creation more **flexible** and **efficient**.

- Instead of directly creating objects through code, these patterns offer **alternative approaches**.

- They provide **more control** over the object creation process.

Singleton

Ensures that a class has **only one instance.**

Provides a **global point of access** to that instance.

Commonly used to **manage resources** like databases.

Structural

- Focus on the **structure** of classes and objects.
- Facilitate **efficient** and **scalable** composition of classes and objects.
- Aim to **simplify design** by creating larger, more complex structures from simpler components.

Adapter

Acts as a **bridge** between two incompatible interfaces.

Allows incompatible classes to work together by wrapping an existing class with a new interface.

Behavioral

- Focus on the **interaction** and **responsibility** between objects.
- Improve communication between objects while making the system more **flexible** and easier to **maintain**.
- Help in defining how objects interact and **collaborate** to perform tasks.

Chain of Responsibility

Allows multiple objects to handle a request in a chain, with each object having the opportunity to process the request.

Passes the request along the chain until an object handles it or the chain ends.

Are you ready to learn two common Design Patterns?

NO YES

MVC Design pattern divides an application into 3 interconnected components

MODEL

The **Model** represents the data and the business logic of the application. It directly manages the data, logic, and rules of the application.

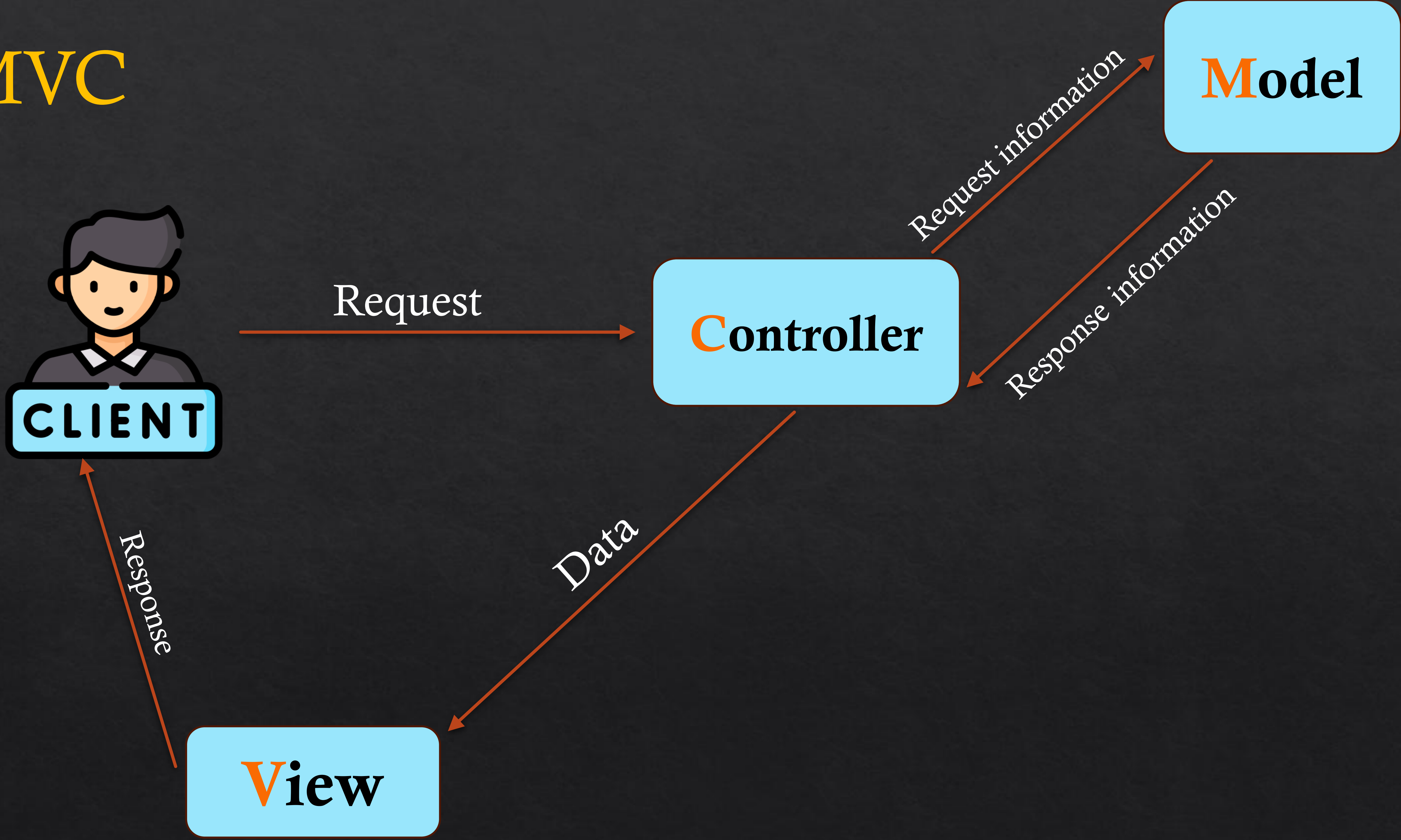
VIEW

The **View** is the component responsible for displaying the data to the user. It represents the UI of the application.

CONTROLLER

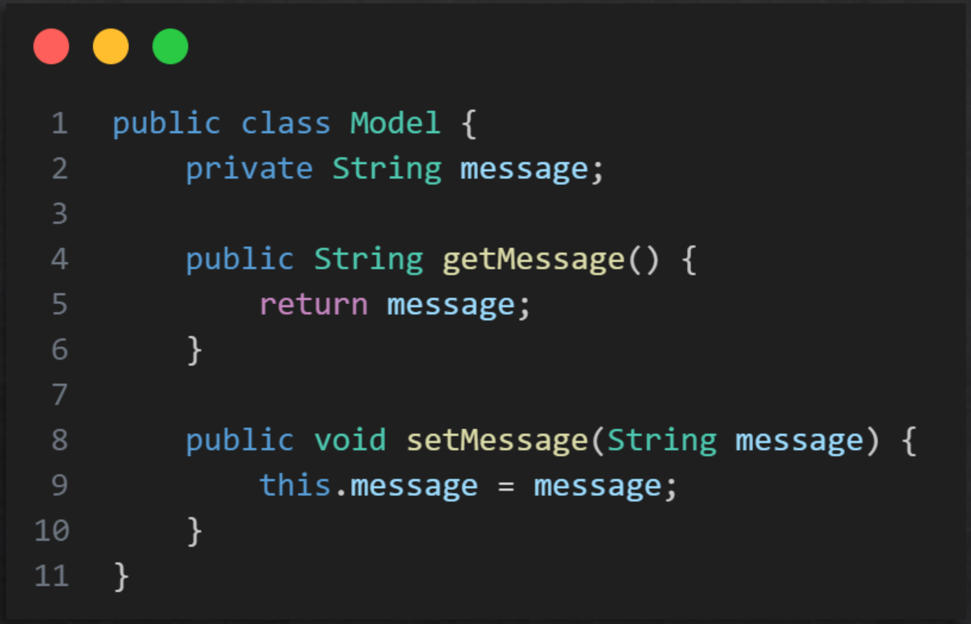
The **Controller** acts as an intermediary between the Model and the View. It handles user input, processes it (often modifying the Model as needed), and returns the output display to the View.

MVC


Model
Request information
Response information
Request
Controller
CLIENT
Data
Response
View

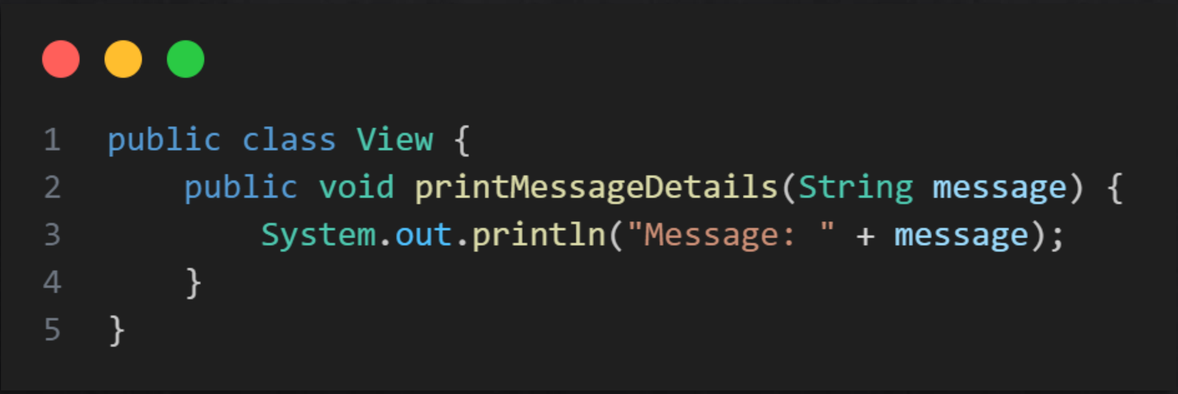
Implementation

Model



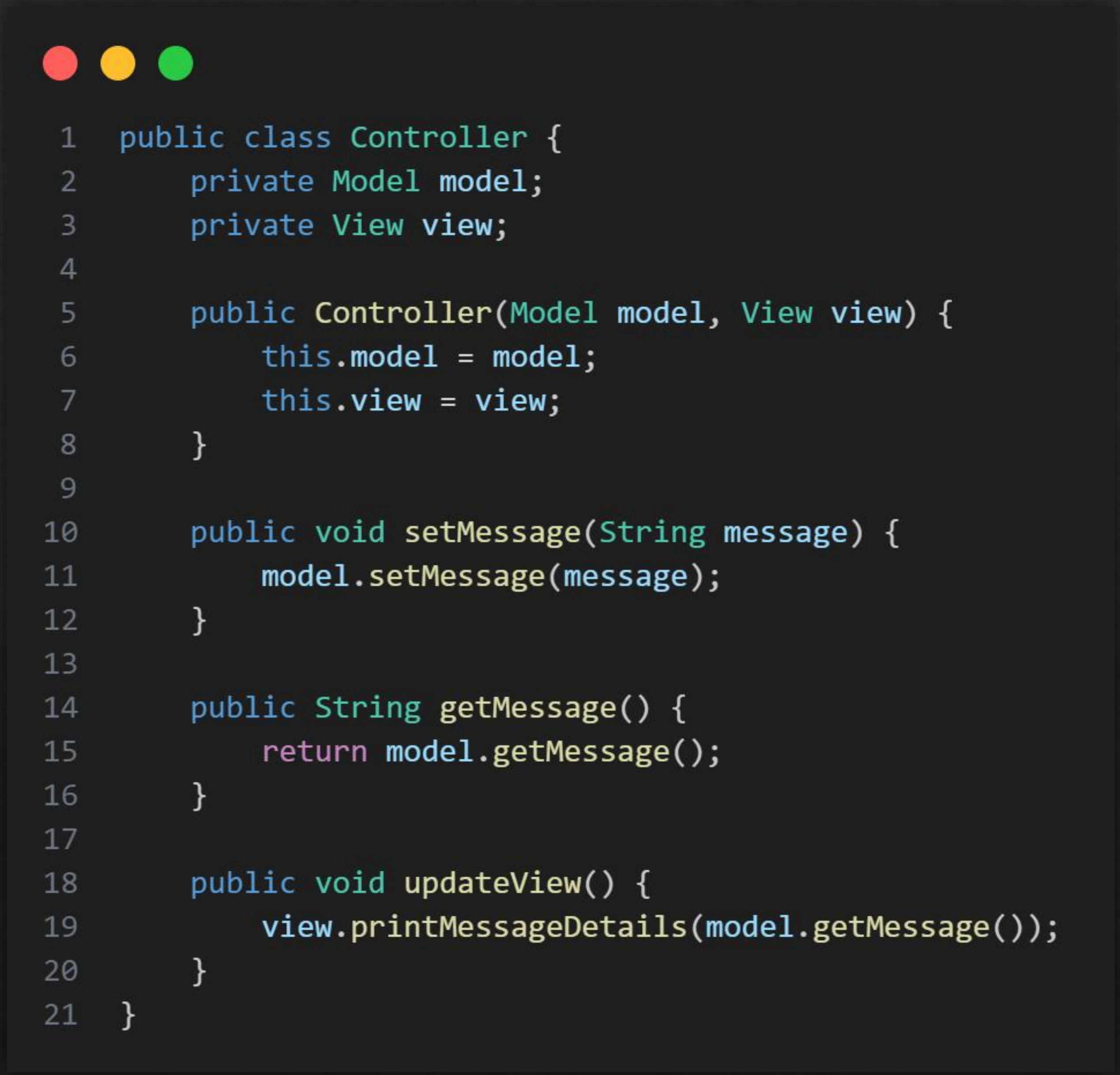
```java
public class Model {
    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

View



```java
public class View {
    public void printMessageDetails(String message) {
        System.out.println("Message: " + message);
    }
}
```

Controller



```java
public class Controller {
    private Model model;
    private View view;

    public Controller(Model model, View view) {
        this.model = model;
        this.view = view;
    }

    public void setMessage(String message) {
        model.setMessage(message);
    }

    public String getMessage() {
        return model.getMessage();
    }

    public void updateView() {
        view.printMessageDetails(model.getMessage());
    }
}
```

Singleton design pattern

Key Characteristics of the Singleton Pattern

1.**Single Instance**: The class restricts the instantiation of itself so that only one instance is created throughout the application's lifecycle.

2. **Global Access**: The instance is accessible globally, typically through a static method that returns the instance.

3. **Lazy Initialization (Optional)** : The instance is created only when it is needed, which can improve performance if the instance is not required immediately.

Implementation

Step 1

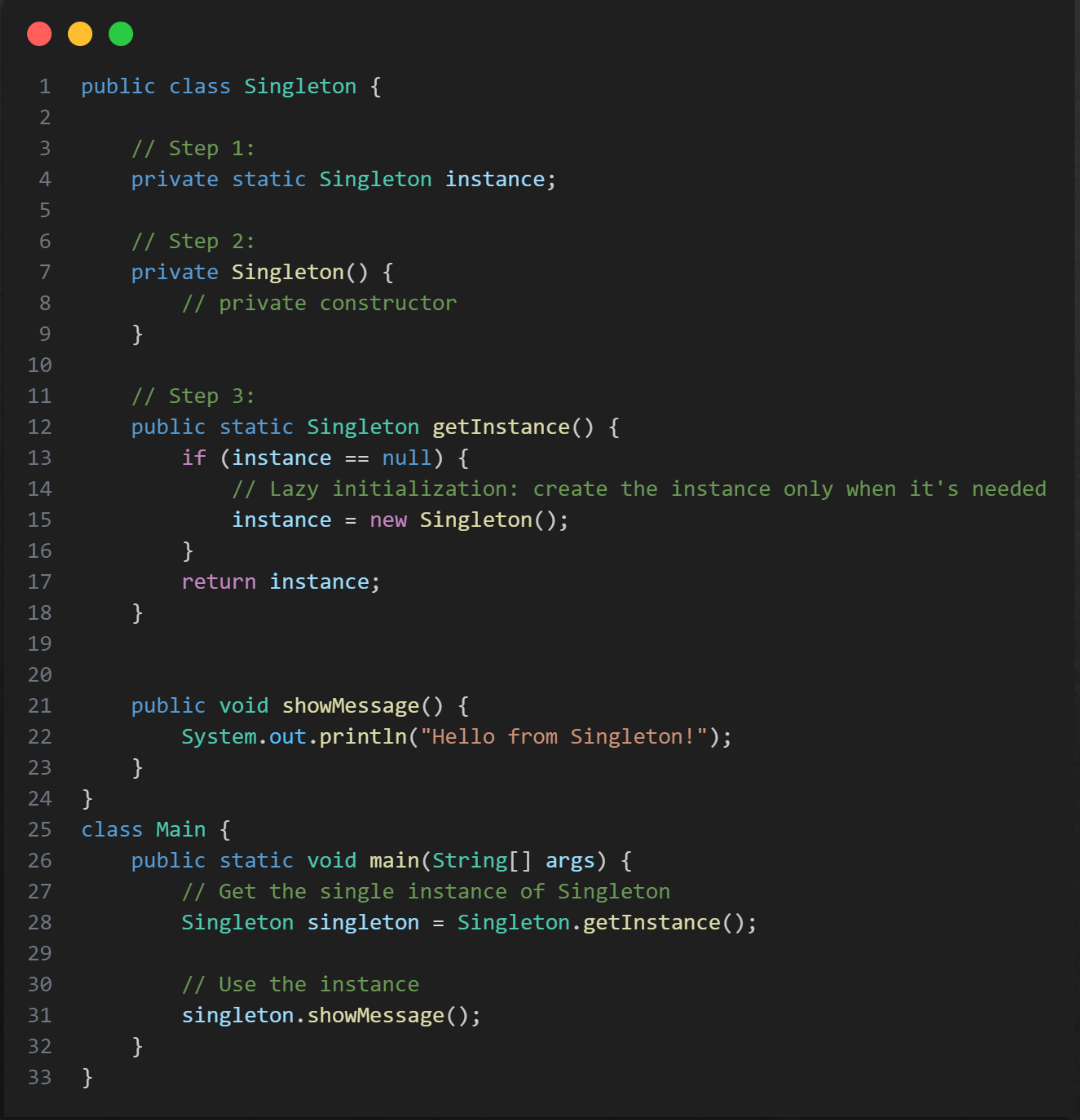
Create a private static variable of the class itself

Step 2

Make the constructor private to prevent instantiation from other classes

Step 3

Provide a public static method to get the instance



```java
public class Singleton {

    // Step 1:
    private static Singleton instance;

    // Step 2:
    private Singleton() {
        // private constructor
    }

    // Step 3:
    public static Singleton getInstance() {
        if (instance == null) {
            // Lazy initialization: create the instance only when it's needed
            instance = new Singleton();
        }
        return instance;
    }


    public void showMessage() {
        System.out.println("Hello from Singleton!");
    }
}
class Main {
    public static void main(String[] args) {
        // Get the single instance of Singleton
        Singleton singleton = Singleton.getInstance();

        // Use the instance
        singleton.showMessage();
    }
}
```

Resources

https://refactoring.guru/design-patterns

https://www.geeksforgeeks.org/java-design-patterns

https://www.tutorialspoint.com/design_pattern/index.htm are there any questions ?

NO YES

Thanks for your Attention.