# 

# 

# 

# **Documentation: Homework 4 (Advanced Structural Patterns – Singleton & Adapter)**

By Tolstova Kristina

# 

## **Table of contents:**

[Overview](#_1gwnwytgc3)

[Part 1: Global Configuration Manager (Singleton Pattern)](#_ovtz71an1k6p)

[Objective](#_7387zaw9hb2l)

[Implementation Details](#_irecqetvlrhf)

[ConfigurationManager Class](#_23md6khgzvj)

[ConfigManagerDemo Class](#_ojr2mmn4d6yx)

[Code Implementation](#_hdnqunameax0)

[ConfigurationManager.java](#_xytoj3x95ugy)

[ConfigManagerDemo.java](#_2ezmw0ipp3u6)

[Expected Output](#_q6vse9bc4u67)

[Part 2: Chat Service Adapter (Adapter Pattern)](#_kn9dmq6eipok)

[Objective](#_y8ve92736udc)

[Implementation Details](#_4ioxu7uxgtgf)

[LegacyChatService Class](#_xb06n83bpazv)

[ChatService Interface](#_ggrtjerqmuzm)

[ChatServiceAdapter Class](#_gte96pm2dbi7)

[ChatAdapterDemo Class](#_kdtmea499lyr)

[Code Implementation](#_70dhoz36adz9)

[LegacyChatService.java](#_zi76nlxj9nmf)

[ChatService.java](#_lcop4t8fwn16)

[ChatServiceAdapter.java](#_ctkh8zdu2ci7)

[ChatAdapterDemo.java](#_yxrmumx3vfss)

[Expected Output](#_agg3jo7yrzv5)

[Conclusion](#_34e24hsn4bas)

## **Overview**

This document provides a detailed explanation of the implementation of two structural design patterns: Singleton and Adapter. The assignment includes two parts:

1. **Global Configuration Manager (Singleton Pattern)**
2. **Chat Service Adapter (Adapter Pattern)**

The implementation ensures that a single instance of the configuration manager exists throughout the application and adapts a legacy chat system to a new interface.

## **Part 1: Global Configuration Manager (Singleton Pattern)**

### **Objective**

The goal of this implementation is to ensure that only one instance of a configuration manager exists, providing a globally accessible resource.

### **Implementation Details**

#### **ConfigurationManager Class**

* Uses **lazy initialization** to ensure that the instance is created only when needed.
* Stores configuration settings in a HashMap.
* Provides a method to retrieve values by key.
* Includes a method to print all configuration settings.

#### **ConfigManagerDemo Class**

* Demonstrates the use of the singleton configuration manager.
* Retrieves and prints configuration values.

### **Code Implementation**

#### **ConfigurationManager.java**

package Part1;

import java.util.HashMap;

import java.util.Map;

class ConfigurationManager {

private static ConfigurationManager *instance*;

private Map<String, String> config;

private ConfigurationManager() {

config = new HashMap<>();

config.put("maxPlayeArs", "100");

config.put("defaultLanguage", "en");

config.put("gameDifficulty", "medium");

}

public static ConfigurationManager getInstance() {

if (*instance* == null) {

*instance* = new ConfigurationManager();

}

return *instance*;

}

public String getConfig(String key) {

return config.getOrDefault(key, "Not Found");

}

public void printAllConfigs() {

config.forEach((key, value) -> System.*out*.println(key + " -> " + value));

}

}

#### **ConfigManagerDemo.java**

package Part1;

class ConfigManagerDemo {

public static void main(String[] args) {

ConfigurationManager configManager = ConfigurationManager.*getInstance*();

System.*out*.println("Max Players: " + configManager.getConfig("maxPlayers"));

System.*out*.println("Game Difficulty: " + configManager.getConfig("gameDifficulty"));

System.*out*.println("All Configurations:");

configManager.printAllConfigs();

}

}

### **Expected Output**

Max Players: 100

Game Difficulty: medium

All Configurations:

maxPlayers -> 100

defaultLanguage -> en

gameDifficulty -> medium

## **Part 2: Chat Service Adapter (Adapter Pattern)**

### **Objective**

The goal of this implementation is to integrate a legacy chat system by adapting its interface to match a new ChatService interface.

### **Implementation Details**

#### **LegacyChatService Class**

* Simulates a legacy system by providing a method legacySend(String message), which prefixes messages with "Legacy Chat:".

#### **ChatService Interface**

* Defines the standard method sendMessage(String message) for the modern chat system.

#### **ChatServiceAdapter Class**

* Implements ChatService and translates calls from sendMessage(String message) into the legacy system’s legacySend() method.

#### **ChatAdapterDemo Class**

* Demonstrates sending a message using the adapter, ensuring correct message conversion.

### **Code Implementation**

#### **LegacyChatService.java**

package Part2;

class LegacyChatService {

public void legacySend(String message) {

System.*out*.println("Legacy Chat: " + message);

}

}

#### **ChatService.java**

package Part2;

interface ChatService {

void sendMessage(String message);

}

#### **ChatServiceAdapter.java**

package Part2;

class ChatServiceAdapter implements ChatService {

private LegacyChatService legacyChatService;

public ChatServiceAdapter(LegacyChatService legacyChatService) {

this.legacyChatService = legacyChatService;

}

@Override

public void sendMessage(String message) {

legacyChatService.legacySend(message);

}

}

#### **ChatAdapterDemo.java**

package Part2;

class ChatAdapterDemo {

public static void main(String[] args) {

LegacyChatService legacyChat = new LegacyChatService();

ChatService chatService = new ChatServiceAdapter(legacyChat);

chatService.sendMessage("Hello world!");

}

}

### **Expected Output**

Legacy Chat: Hello world!

## **Conclusion**

This assignment successfully demonstrates the **Singleton** and **Adapter** design patterns, showcasing their application in a configuration manager and a chat service integration. The implementation ensures efficient resource management and smooth adaptation of legacy components into a modern system.