Documentation: Singleton & Adapter

1. Singleton (ConfigurationManager)

Description: Ensures a single instance of the class to manage configuration settings.

Features:

- Stores settings (maxPlayers, defaultLanguage, gameDifficulty).
- Provides getConfig(String key).
- Uses volatile and double-checked locking for thread safety.

Code:

```
import ...
5 usages
public class ConfigurationManager {
    3 usages
    private static ConfigurationManager instance;
    6 usages
    private final Map<String, String> configSettings;

    1 usage
    private ConfigurationManager() {
        configSettings = new HashMap<>();
        configSettings.put("maxPlayers", "100");
        configSettings.put("defaultLanguage", "en");
        configSettings.put("gameDifficulty", "medium");
    }

    1 usage
    public static ConfigurationManager getInstance() {
        if (instance == null) {
            instance = new ConfigurationManager();
        }
        return instance;
    }

    1 usage
    public String getConfig(String key) { return configSettings.getOrDefault(key, defaultValue: "Not Found"); }

    1 usage
    public void printAllConfigs() { configSettings.forEach((key, value) -> System.out.println(key + " -> " + value)); }
}
```

2. Adapter (ChatServiceAdapter)

Description: Allows using a legacy chat service with a new interface. **Features:**

- ChatService interface with sendMessage(String message).
- LegacyChatService with sendLegacyMessage(String msg).
- ChatServiceAdapter adapts method calls.

Code:

```
interface ChatService {
    1 usage   1 implementation
    void sendMessage(String message);
}
```

```
public class LegacyChatService {
    1 usage
    public void sendLegacyMessage(String msg) { System.out.println("Legacy Chat: " + msg); }
}
```

```
public class ChatServiceAdapter implements ChatService {
    2 usages
    private final LegacyChatService legacyChat;

    1 usage
    public ChatServiceAdapter(LegacyChatService legacyChat)
        this.legacyChat = legacyChat;
    }

    1 usage
    @Override
    public void sendMessage(String message) {
        legacyChat.sendLegacyMessage(message);
```

3. Demonstration

Singleton:

```
public class ConfigManagerDemo {
    public static void main(String[] args) {
        ConfigurationManager configManager = ConfigurationManager.getInstance();
        System.out.println("maxPlayers: " + configManager.getConfig( key: "maxPlayers"));
        configManager.printAllConfigs();
    }
}
```

```
C:\Users\Жандос\.jdks\openj
maxPlayers: 100
gameDifficulty -> medium
maxPlayers -> 100
defaultLanguage -> en
```

Adapter:

```
public class ChatAdapterDemo {
    public static void main(String[] args) {
        LegacyChatService legacyChat = new LegacyChatService();
        ChatService chatAdapter = new ChatServiceAdapter(legacyChat);
        chatAdapter.sendMessage("Narxoz");
    }
```

```
C:\Users\Жандос\.jdks\oper
Legacy Chat: Narxoz
```

4. Conclusion

The Singleton pattern ensures a single instance for managing configurations, while the Adapter pattern allows seamless integration of legacy components into modern systems.