Builder Pattern Example with Computer Database

Overview

This project demonstrates the implementation of the Builder Pattern in Java, focusing on creating Computer objects with various configurations. It also includes a simple in-memory database to store and retrieve these Computer objects. The project showcases how the Builder Pattern can be used to construct complex objects step by step and how it can be integrated with a basic data storage mechanism.

Classes

The project consists of three main classes:

1. Computer

The `Computer` class represents a computer with various attributes:

- CPU

- RAM

- Storage

- GPU

- Operating System

Key features:

- Uses the Builder pattern for object creation

- Private constructor to ensure objects are only created through the Builder

- Getter methods for all attributes

- A `toString()` method for string representation

Inner Builder Class

The `Computer` class contains an inner `Builder` class that:

- Provides methods to set each attribute of the Computer

- Implements a fluent interface for method chaining

- Includes a `build()` method to create the final Computer object

2. ComputerDatabase

The `ComputerDatabase` class serves as a simple in-memory database for storing Computer objects.

Key features:

- Maintains a list of Computer objects

- Provides methods to add computers and retrieve all stored computers

3. BuilderPatternExample

This class contains the `main` method and serves as the entry point of the application. It demonstrates:

- Creation of Computer objects using the Builder pattern

- Addition of computers to the database

- Retrieval and display of all computers from the database

Implementation Process

Link[: Click here for the code.](https://github.com/Akashmondal55/Akash_5016855/tree/main/Week-1/Design%20patern%20and%20princple/Exercise-2)

1. Define the Computer class:

- Create the Computer class with all necessary attributes

- Implement a private constructor that takes a Builder object

- Create getter methods for all attributes

- Implement the `toString()` method for easy printing of Computer objects

2. Implement the Builder:

- Create an inner Builder class within the Computer class

- Add methods in the Builder to set each attribute

- Implement the `build()` method to create and return a new Computer object

3. Create the ComputerDatabase:

- Implement methods to add computers and retrieve all computers

- Use a List to store Computer objects

4. Develop the main application:

- In the BuilderPatternExample class, create Computer objects using the Builder

- Add the created computers to the database

- Retrieve and display all computers from the database

Usage Example

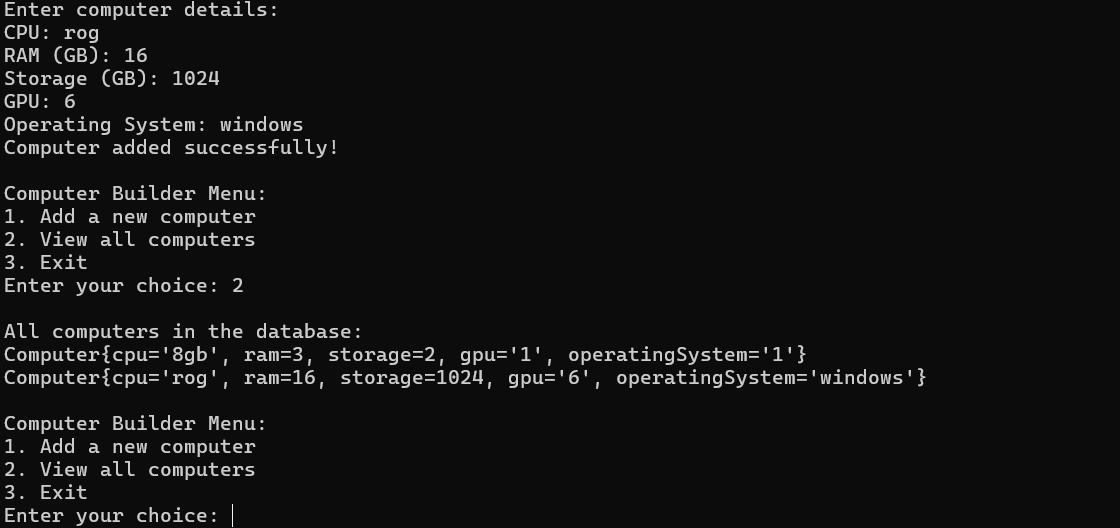
The main method in BuilderPatternExample demonstrates how to:

1. Create Computer objects with different configurations using the Builder pattern

2. Add these computers to the ComputerDatabase

3. Retrieve and display all stored computers

Output



Conclusion

This project effectively demonstrates the power and flexibility of the Builder Pattern in creating complex objects. By using the Builder Pattern, we achieve:

1. Improved readability and maintainability of object creation code

2. The ability to create different representations of an object using the same construction process

3. More control over the construction process, allowing for step-by-step creation of objects

The integration with a simple database also shows how the Builder Pattern can be used in a practical scenario where objects need to be created and stored. This implementation provides a solid foundation for further expansion, such as adding more complex database operations or extending the Computer class with additional features.

Overall, this project serves as an excellent example of how design patterns like the Builder can be used to solve common software design problems and improve code quality.