
VIRTUAL INTERIOR DESIGN:
THE NEW WAY TO DECORATE

TABLE OF CONTENTS

“VIRTUAL INTERIOR DESIGN”

ABSTRACT:

"Virtual Interior Design is an innovative approach that utilises advanced technologies to create immersive and interactive virtual environments, enabling individuals to visualise and experience interior spaces before they are physically realised. By leveraging virtual reality (VR) and augmented reality (AR) technologies, Virtual Interior Design offers a realistic and dynamic representation of architectural designs, allowing users to explore and interact with various elements such as furniture, textures, lighting, and spatial arrangements. This cutting-edge methodology revolutionises the traditional interior design process by providing clients, designers, and stakeholders with a comprehensive understanding of the proposed design, facilitating effective decision-making, customization, and seamless communication. Through Virtual Interior Design, individuals can embark on a virtual journey to witness the fusion of aesthetics, functionality, and ambiance, ultimately leading to the creation of captivating and personalised interior spaces."

INTRODUCTION:

Virtual interior design encompasses a variety of design patterns that can transform a space into a visually captivating and functional environment. These design patterns serve as the foundation for creating immersive virtual experiences that bring interior designs to life. From the simplicity and serenity of minimalist design to the vibrant and eclectic nature of electic design, each pattern offers its own unique aesthetic and ambiance. By carefully selecting and incorporating these design patterns, virtual interior design projects can evoke specific moods, cater to different preferences, and ultimately provide an engaging and personalised experience for users.

The patterns described in the book (GANG OF FOUR) are divided into three categories:

- Creational patterns
- Structural patterns
- **B**ehavioral patterns

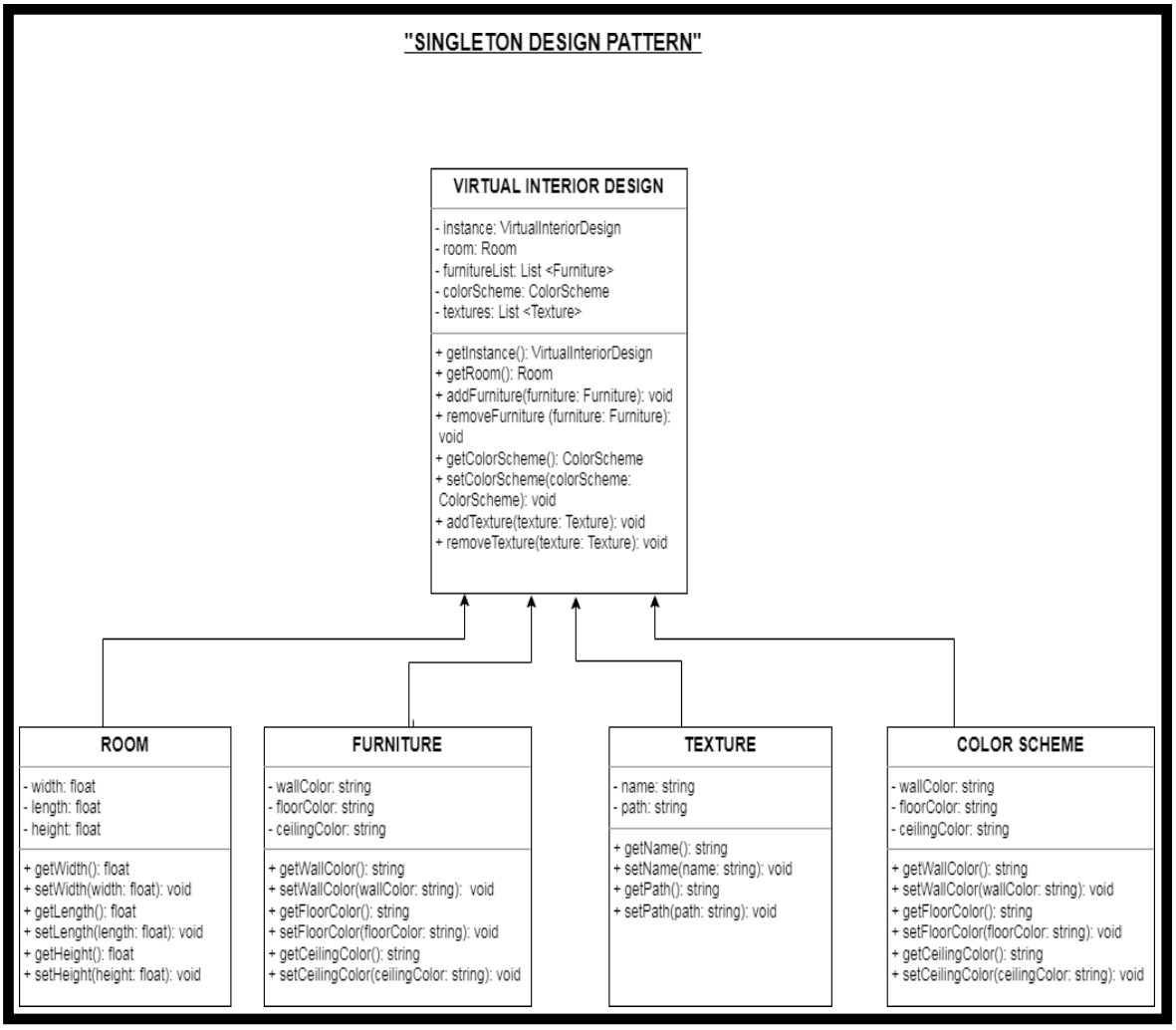
1. CREATIONAL PATTERNS :

Creational patterns in virtual interior design refer to design patterns that focus on the creation and initialization of objects and components within a virtual environment. These patterns help in creating and managing virtual entities in a flexible and efficient manner. There are several kinds of Creational patterns but we consider only three types for virtual interior design:

- Singleton pattern
- Builder pattern
- Factory pattern

1.1. SINGLETON PATTERN:

The Singleton pattern is a design pattern that allows a class to have only one instance (object) created and provides global access to that instance.



This is the main class that represents **virtual interior design**. It follows the Singleton pattern, ensuring that only one instance of **VirtualInteriorDesign** exists. It contains instance variables such as room, which represents the dimensions of the room, furnitureList to store a list of furniture objects, colorScheme to store the color scheme of the room, and textures to store a list of texture objects. It provides methods to access and modify these attributes, such as getRoom(), addFurniture(), getColorScheme(), addTexture(), etc.

Room: This class represents the dimensions of the room in the virtual interior design. It has instance variables for width, length, and height, and provides getter and setter methods for each of these attributes, such as getWidth(), setHeight(), etc.

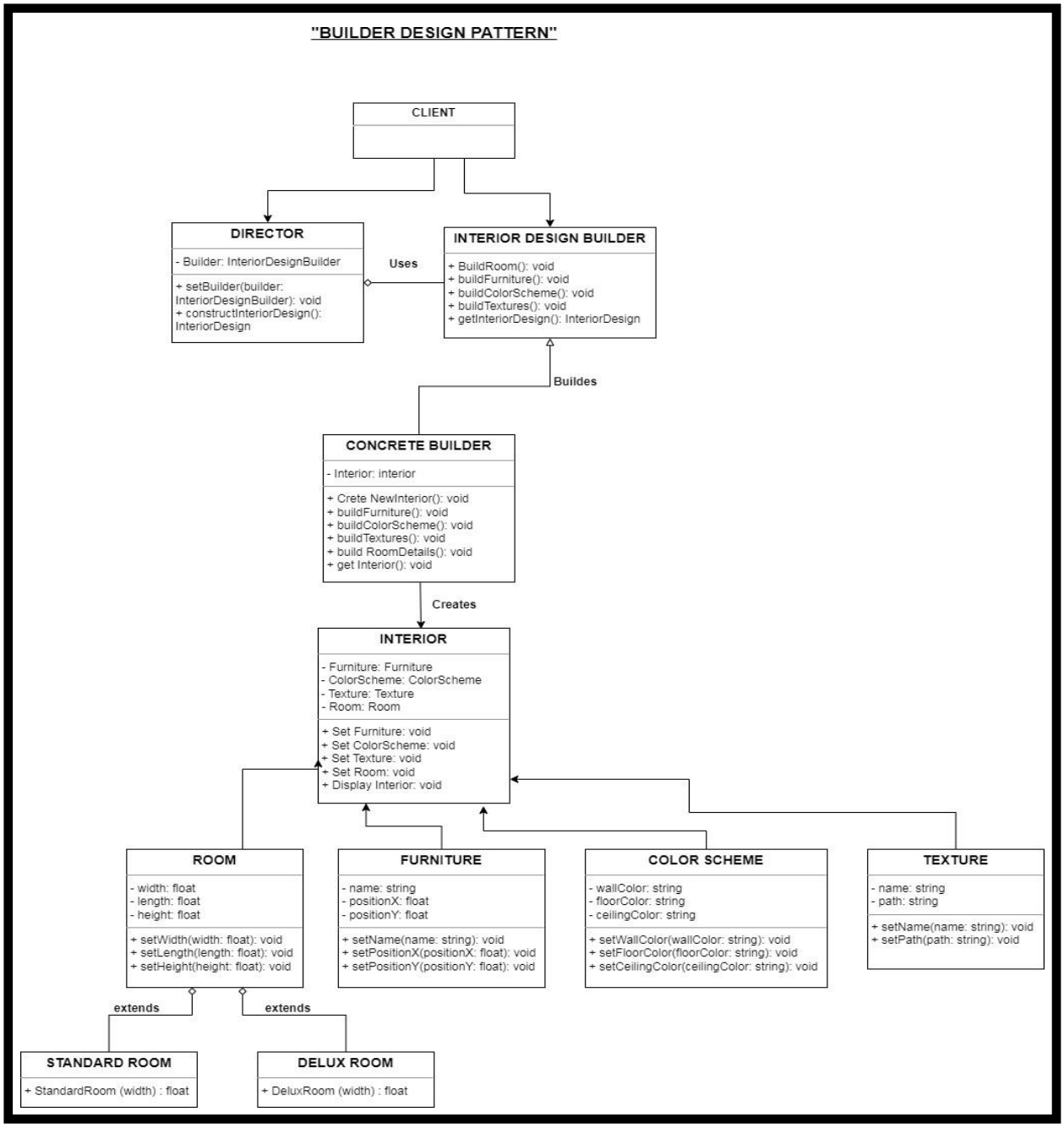
Furniture: This class represents a piece of furniture in the virtual interior design. It has instance variables for name, positionX, and positionY, which indicate the name of the furniture and its position in the room. It provides getter and setter methods for these attributes, such as getName(), setPositionX(), etc.

ColorScheme: This class represents the color scheme of the virtual interior design. It has instance variables for wallColor, floorColor, and ceilingColor, indicating the colors of the walls, floor, and ceiling respectively. It provides getter and setter methods for these attributes, such as getWallColor(), setCeilingColor(), etc.

Texture: This class represents a texture that can be applied to surfaces in the virtual interior design. It has instance variables for name and path, which store the name of the texture and its file path. It provides getter and setter methods for these attributes, such as getName(), setPath(), etc.

1.2. BUILDER PATTERN:

The Builder pattern is a software design pattern that separates the construction of an object from its representation, allowing complex objects to be created step by step with different configurations, while promoting code readability and maintainability.



Director: The Director class is responsible for managing the construction process and interacts with the builder to build the interior.

InteriorBuilder: The abstract Builder class defines the common interface for building an interior and holds a reference to the Interior object being constructed.

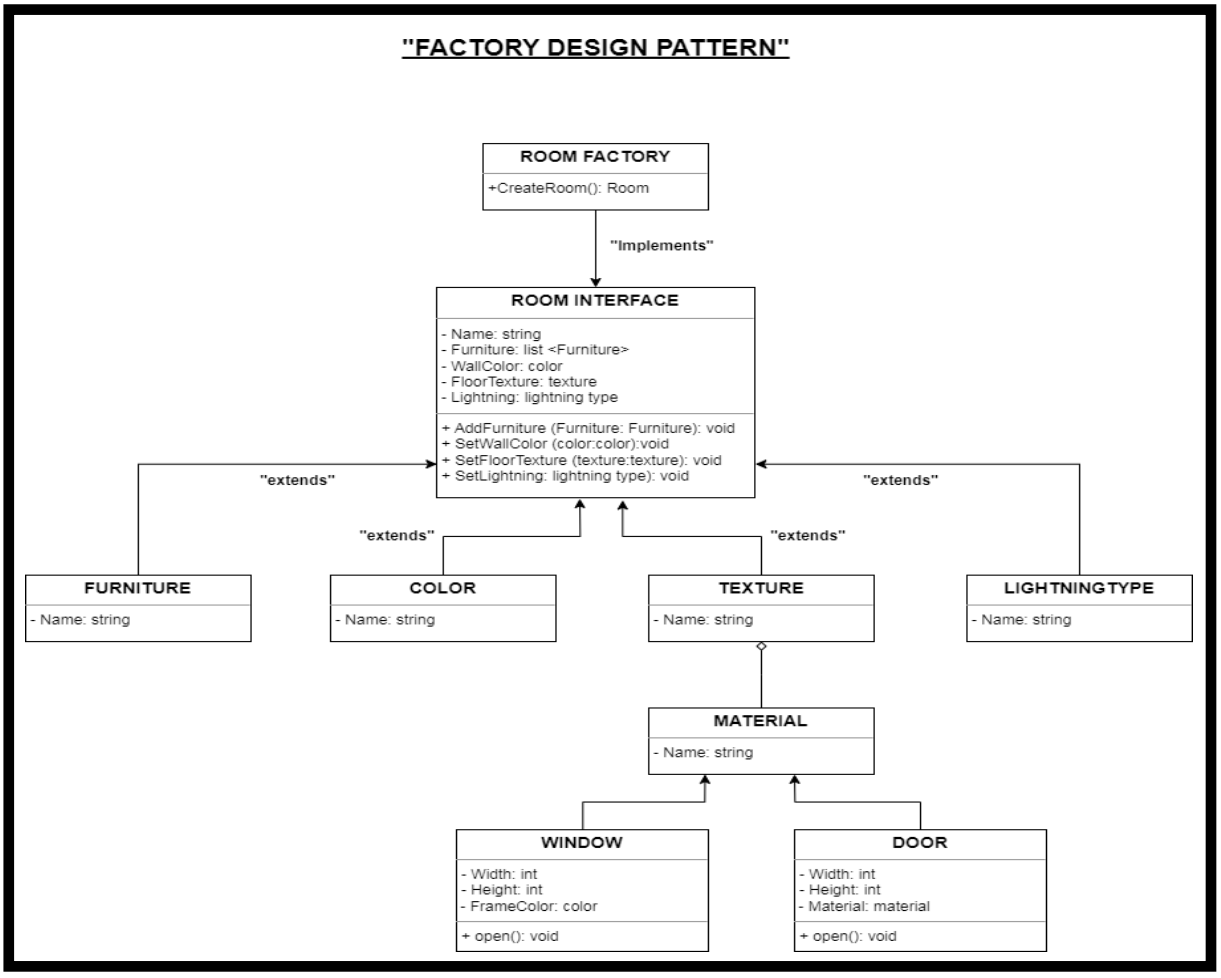
ConcreteBuilder: The ConcreteBuilder class extends the InteriorBuilder and provides the implementation for building the interior components such as furniture, color scheme, texture, and room details.

Interior: The Interior class represents the final product that is constructed by the builder. It contains attributes for furniture, color scheme, texture, and room details. It also provides methods to set and get these attributes and display the interior.

Furniture, ColorScheme, Texture, RoomDetails: These are the individual components of the Interior. They are separate classes representing furniture details, color scheme details, texture details, and room details, respectively. Each of these classes provides methods.

1.3. FACTORY PATTERN:

The Factory pattern is a design pattern that encapsulates the object creation process by providing a centralized factory class. This class is responsible for creating and returning objects based on specific conditions or inputs. By utilizing the Factory pattern, client code can create objects without directly instantiating their concrete classes, promoting loose coupling and flexibility in the system.



The **RoomFactory** class represents the factory responsible for creating Room objects. It has a method createRoom() that returns an instance of the Room class.

The **Room** class represents a room in virtual interior design. It has attributes such as name, furniture, wallColor, floorTexture, and lighting. It also provides methods such as addFurniture(), setWallColor(), setFloorTexture(), and setLighting() to customize the room's properties.

The **Furniture, Color, Texture**, and **LightingType** classes represent different entities used in virtual interior design, such as furniture pieces, colors, textures, and lighting types. They have attributes specific to their respective types and may provide additional methods as needed.

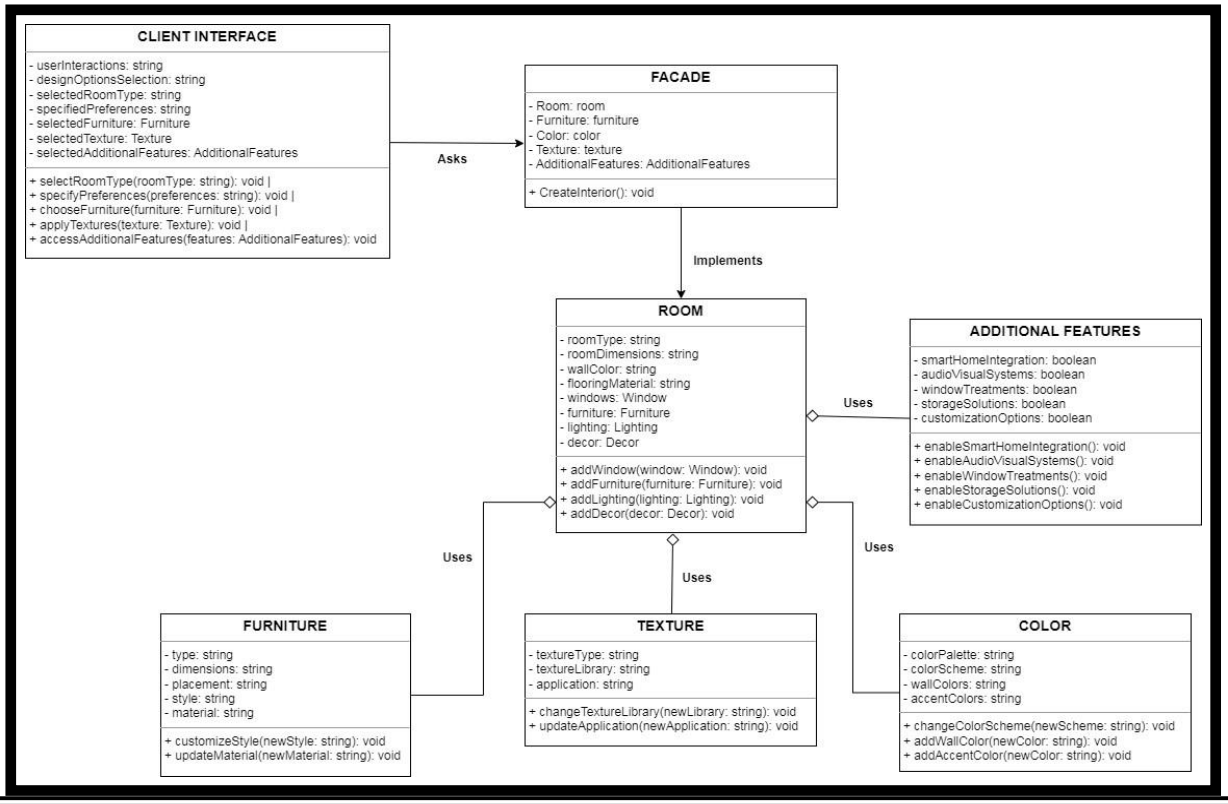
The factory design pattern allows the RoomFactory to encapsulate the creation logic and provide a consistent interface for creating Room objects. By using the factory method createRoom(), you can obtain instances of Room without directly specifying their concrete classes. This promotes loose coupling and makes it easier to extend and manage the creation of different types of rooms in the virtual interior design system.

2. STRUCTURAL PATTERN:

The structural pattern in virtual interior design refers to the organization and arrangement of elements within a virtual space. It includes aspects such as spatial layout, scale, proportion, circulation, lighting, and material choices. This pattern forms the foundation for the overall design concept and allows designers and clients to visualize and assess the virtual interior before physical implementation. It ensures a well-designed and functional space that meets the client's requirements and design principles.

2.1. FACADE PATTERN:

The facade pattern is a software design pattern that provides a simple and unified interface to a complex subsystem or set of classes. It acts as a "facade" or front-facing interface that encapsulates the underlying complexity and presents a simplified view to the client.



In the context of virtual interior design, the facade pattern can be applied to the client interface, which interacts with various classes representing rooms, furniture, textures, colors, and additional features. The client interface serves as a simplified entry point for the client or user to interact with the virtual interior design system.

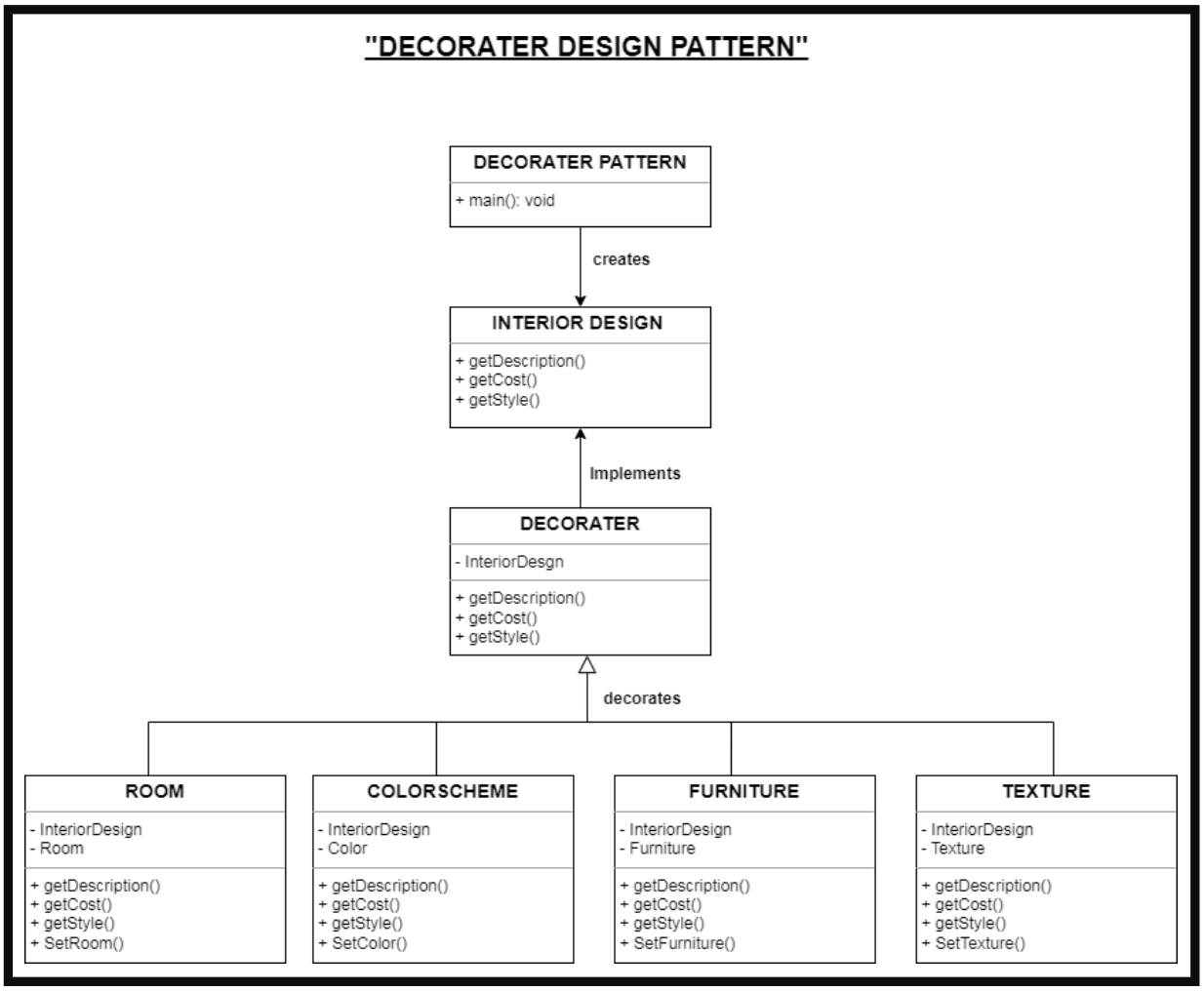
The **client** interface may have methods like createRoom(), addFurniture(), applyTexture(), changeColor(), and addAdditionalFeature(). These methods delegate the appropriate actions to the corresponding classes responsible for rooms, furniture, textures, colors, and additional features. The facade pattern shields the client from having to know and manage the intricacies of each individual class, making the overall system easier to use and understand.

The **Façade** class is the central component that encapsulates the subsystem classes. It contains references to specialised facade classes: **Room**, **Furniture**, **Texture**, **Color** and **AdditionalFeatures**. These specialised facade classes handle the interaction with their respective subsystems.

2.2. DECORATOR PATTERN:

The Decorator pattern dynamically adds new behavior to an object by wrapping it with a decorator class.It allows for flexible extension of an object's functionality without modifying its original implementation.

The pattern promotes code reusability and maintainability by enabling objects to be decorated with new features at runtime.



Decorater Pattern is the interface or abstract class that defines the common operations for the components.

Interior Design is a concrete class that implements the Component interface. It represents the base component or room in the virtual interior design.

Decorator is the abstract class that extends the **Decorater Pattern** interface. It contains a reference to the **Decorater Pattern** and overrides the getDescription method.

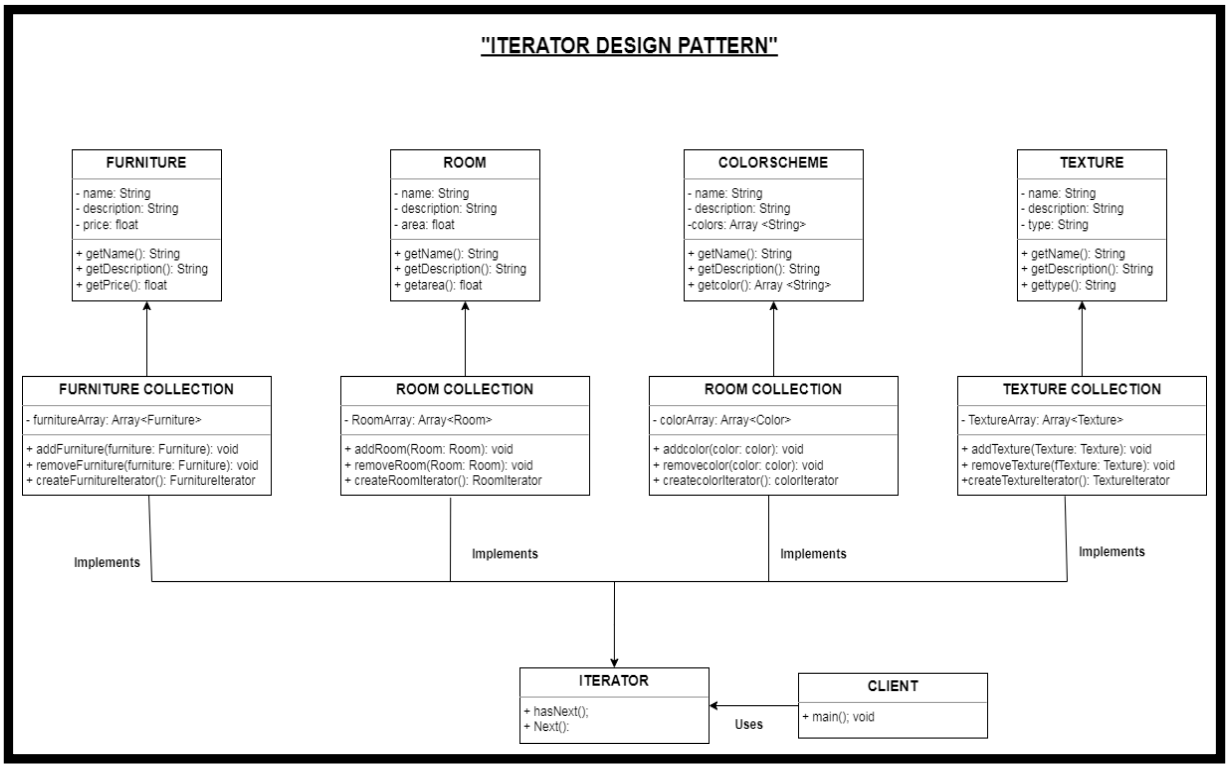
Room, **Furniture**, **ColorScheme**, **Texture** are concrete decorator classes that extend the Decorator class. Each decorator adds specific functionalities related to room, furniture, or texture, respectively.

3. BEHAVIORAL PATTERN:

Behavioural patterns in virtual interior design are influenced by user preferences, emotional responses, spatial perception, cultural influences, and customization options. Users' personal tastes, cultural backgrounds, and lifestyle choices shape their design preferences. Design elements such as colors, lighting, and textures can evoke different emotions and create desired atmospheres. Understanding spatial perception helps optimize usability and comfort, while cultural and social factors influence design aesthetics. Personalization and customization allow users to express their individuality and create unique virtual spaces.

3.1. ITERATIVE PATTERN:

The Iterator pattern provides a way to sequentially access elements of a collection without exposing its underlying structure. It encapsulates the iteration logic within an iterator object, which allows clients to access elements of the collection one by one using a common interface. This pattern promotes decoupling of the client code from the collection implementation, enhancing code flexibility and reusability.



The **Furniture**, **Room**, **ColorScheme**, and **Texture** classes represent the different elements of the interior design system, each having relevant attributes and getter methods.

The **FurnitureCollection**, **RoomCollection**, **ColorSchemeCollection**, and **TextureCollection** classes manage collections of their respective objects.

Each collection class includes a createIterator() method that returns an Iterator object specifically designed for iterating over that collection.

The Iterator class provides a standardized interface with hasNext() to check for the availability of the next element and next() to retrieve the next element.

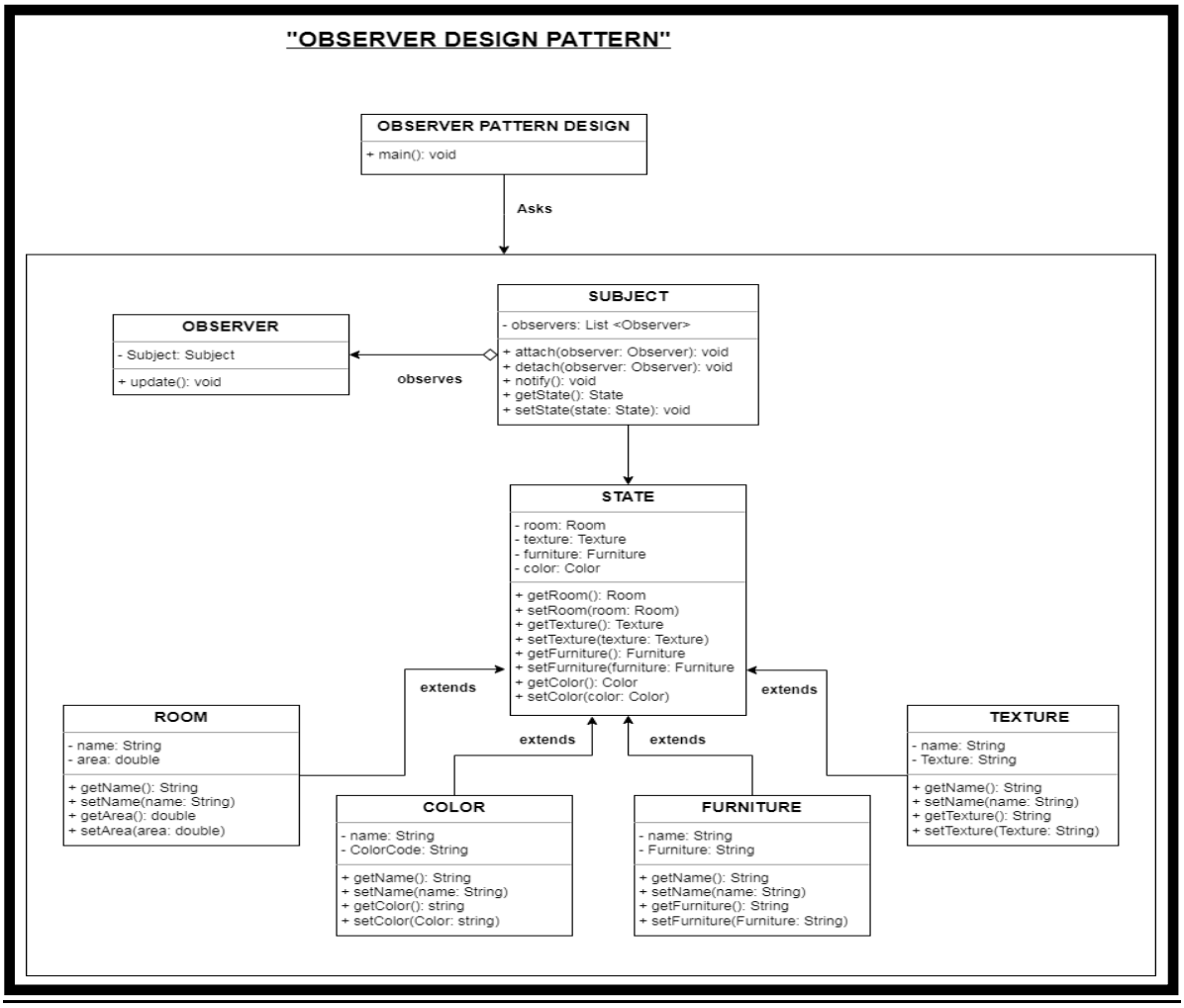
The FurnitureCollection, RoomCollection, ColorSchemeCollection, and TextureCollection classes maintain an internal array of objects.

The FurnitureCollection creates a FurnitureIterator, the RoomCollection creates a RoomIterator, the ColorSchemeCollection creates a ColorSchemeIterator, and the TextureCollection creates a ColorSchemeIterator, and the TextureCollection creates a TextureIterator.

With the Iterator pattern, the client code can iterate over the collections of Furniture, room, texture and color.

3.2. OBSERVER PATTERN:

Observer pattern establishes a relationship between objects where multiple observers are notified automatically when a subject's state changes.



The **Subject** class represents the subject being observed, which the virtual interior design application is. It maintains a list of observers and provides methods to attach, detach, and notify observers of any changes in the state.
The **Observer** interface defines the common interface for all observers. It has an `update()` method that gets called by the subject when there is a state change.
The **State** class represents the state of the virtual interior design application. It holds information such as the room, texture, furniture, and color details.
The observer pattern allows the observers) to receive updates from the subject (the virtual interior design application) when there are changes in the state (such as changes in the room, texture, furniture, or color details). This pattern enables loose coupling between the subject and the observers, allowing for easy addition or removal of observers without affecting the subject's code.
The **Furniture** class represents a piece of furniture in the virtual interior design application. It has attributes like name and material, and corresponding getters and setters.
The **Texture** class represents the texture of the interior design. It has attributes like name and material, and corresponding getters and setters.
The **Room** class represents a room in the virtual interior design. It has attributes like name and area, and corresponding getters and setters.
The **Color** class represents a color used in the interior design. It has attributes like name and hexCode, and corresponding getters and setters.
These classes are part of the State class, which maintain the overall state of the virtual interior design application, including the room, texture, furniture, and color details.

4. ARCHITECTURE PATTERN:

Virtual interior design utilizes various architectural patterns to create visually appealing and functional virtual spaces. These patterns, such as open floor plans, zoning, circulation, lighting and ventilation, material selection, and focal points, play a crucial role in organizing the layout, defining the structure, and enhancing the overall design of the virtual environment. By strategically incorporating these patterns, designers can create immersive virtual spaces that accurately represent their design concepts and provide clients with a realistic visualization of the final result.

4.1. MODEL VIEW CONTROLLER (MVC) PATTERN:

The MVC (Model-View-Controller) pattern is a software design pattern that separates an application into three main components: the model, the view, and the controller. The model manages the data and logic, the view handles the presentation of the data, and the controller handles user interactions and updates the model and view accordingly. This separation of concerns enhances code organization, modularity, and maintainability in applications.
The MVC (Model-View-Controller) pattern in virtual interior design involves dividing the design process into three main components:
Model: The Model represents the underlying data and logic of the virtual interior design. It includes the details of the room, furniture pieces, textures, and color options. The Model encapsulates the properties and characteristics of each element, such as dimensions, positions, and other relevant information.

View: The View is responsible for the visual representation of the virtual interior design. It displays the room layout, furniture arrangements, textures, and colors to the user. The View provides an interactive interface for users to view and modify the design elements, making informed decisions based on the visual information presented.

Controller: The Controller acts as the bridge between the Model and the View. It manages user interactions and updates the Model and View accordingly. The Controller handles user actions, such as selecting furniture pieces, applying textures and colors, and modifying the room layout. It ensures that changes made by the user are reflected in the Model and that the updated design is rendered in the View.