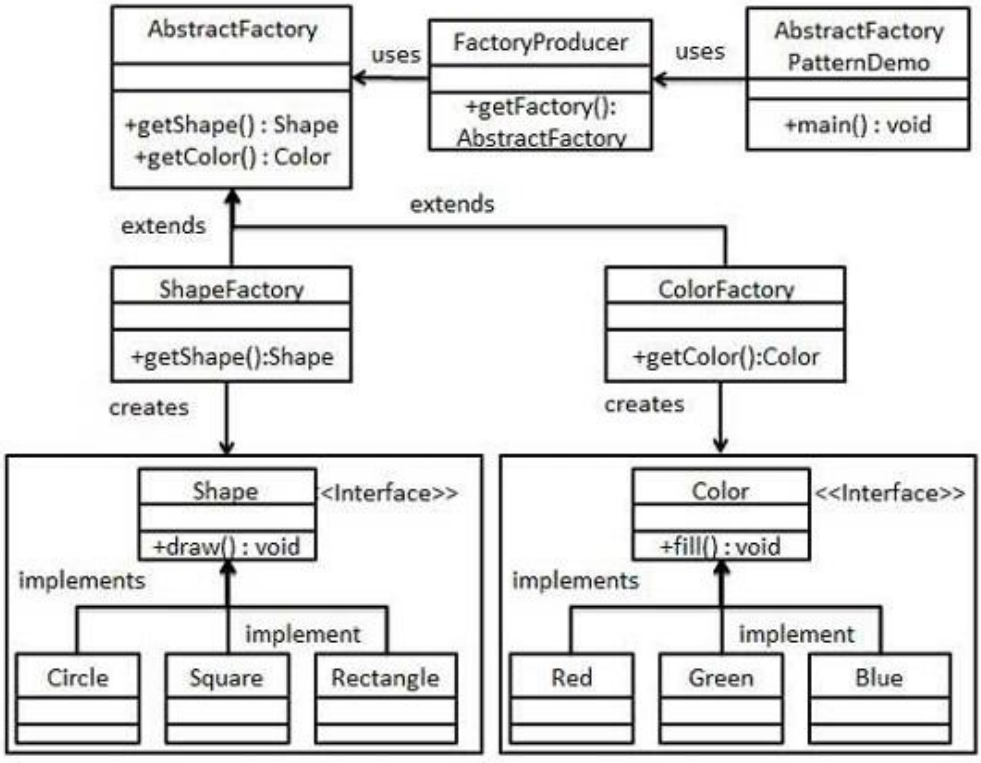
Problem No: 02
Problem Name: Implementation and Design of Abstract Factory Pattern

Objectives:
The objective of this lab report is to implement and demonstrate the Abstract Factory Pattern in C++ for a vehicle manufacturing system. The goal is to show how the pattern facilitates the creation of related product families, such as urban and off-road vehicles, while promoting flexibility, scalability, and loose coupling between the client and concrete product classes. This implementation aims to highlight the advantages of using the Abstract Factory Pattern in managing related objects in object-oriented design.

Theory:

The Abstract Factory design pattern falls under the creational design patterns category. It provides an interface for creating families of related or dependent objects without specifying their concrete classes. This pattern involves a factory interface with multiple factory methods, each responsible for creating a different type of object. By using Abstract Factory, the client code can create objects without knowing their concrete classes, thus promoting loose coupling and enhancing flexibility.

UML class design



Implementation in C++:

```
#include <iostream>
 using namespace std;
 // Abstract Products
 class Chair {
 public:
    virtual void sitOn() = 0;
    virtual ~Chair() {}
 };
class Table {
 public:
    virtual void use() = 0;
    virtual ~Table() {}
 };
 // Concrete Products - Modern Style
 class ModernChair : public Chair {
 public:
    void sitOn() override {
        cout << "Sitting on a modern chair." << endl;
    }
 };
 class ModernTable : public Table {
 public:
    void use() override {
        cout << "Using a modern table." << endl;
    }
 };
 // Concrete Products - Victorian Style
 class VictorianChair : public Chair {
 public:
    void sitOn() override {
        cout << "Sitting on a Victorian chair." << endl;
    }
 };
 class VictorianTable : public Table {
 public:
    void use() override {
        cout << "Using a Victorian table." << endl;
    }
 };
// Abstract Factory
 class FurnitureFactory {
 public:
```

```
    virtual Chair* createChair() = 0;
    virtual Table* createTable() = 0;
    virtual ~FurnitureFactory() {}
};
// Concrete Factory - Modern Furniture
class ModernFurnitureFactory : public FurnitureFactory {
public:
    Chair* createChair() override {
        return new ModernChair();
    }
Table* createTable() override {
        return new ModernTable();
    }
};
// Concrete Factory - Victorian Furniture
class VictorianFurnitureFactory : public FurnitureFactory {
public:
    Chair* createChair() override {
        return new VictorianChair();
    }
Table* createTable() override {
        return new VictorianTable();
    }
};
// Client Code
int main() {
    // Creating Modern Furniture
    FurnitureFactory* modernFactory = new ModernFurnitureFactory();
    Chair* modernChair = modernFactory->createChair();
Table* modernTable = modernFactory->createTable();
    modernChair->sitOn();  // Output: Sitting on a modern chair.
    modernTable->use();    // Output: Using a modern table.
    delete modernChair;
    delete modernTable;
    delete modernFactory;
    // Creating Victorian Furniture
    FurnitureFactory* victorianFactory = new VictorianFurnitureFactory();
    Chair* victorianChair = victorianFactory->createChair();
Table* victorianTable = victorianFactory->createTable();
    victorianChair->sitOn();  // Output: Sitting on a Victorian chair.
    victorianTable->use();    // Output: Using a Victorian table.
    delete victorianChair;
    delete victorianTable;
    delete victorianFactory;
```

```
    return 0;
}
```

Result and Discussion:

The Abstract Factory Pattern was successfully implemented in C++ to create a vehicle manufacturing system. The pattern allowed the creation of families of related vehicles, such as urban and off-road types, without specifying their exact classes in the client code. By using abstract factories, the system became highly flexible and scalable, allowing new product families to be added without modifying existing client code.

The key benefits of the pattern include encapsulation, where the product creation logic is hidden from the client; loose coupling, which keeps the client independent of concrete product classes; and scalability, which makes it easy to add new vehicle families without altering the code.

Overall, the implementation demonstrated how the Abstract Factory Pattern simplifies handling related products and improves maintainability in object-oriented design.