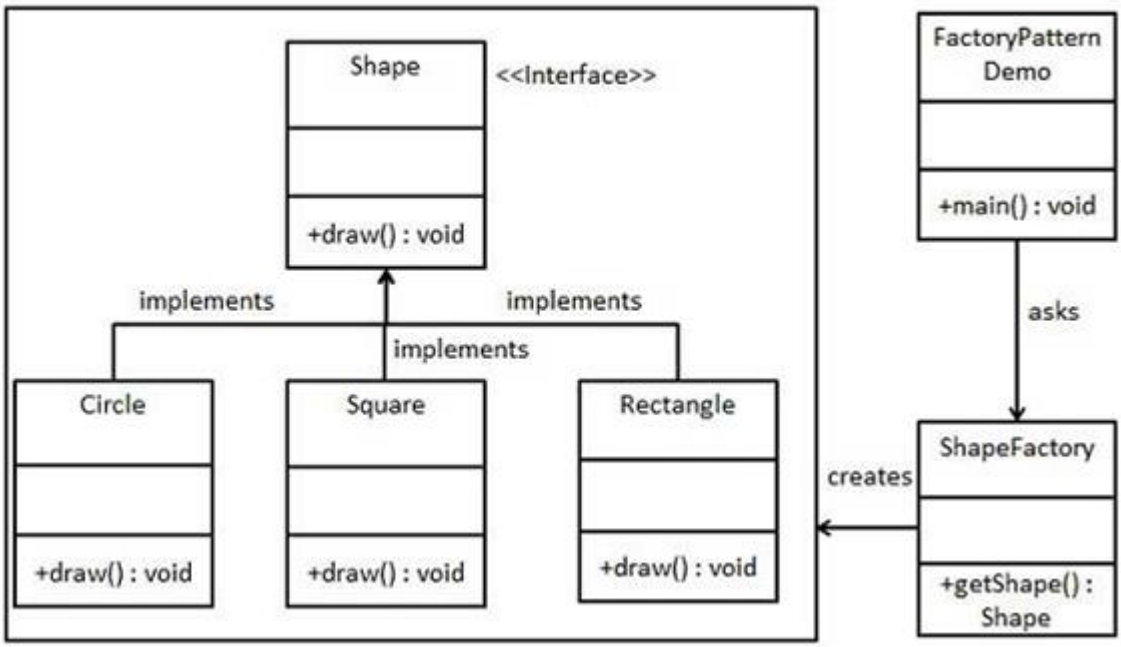
**Problem No:** 01
**Problem Name:** Implementation and Design of Factory Method Pattern

**Objectives**

- To understand the Factory Method design pattern.
- To implement the Factory Method pattern in C++/Java.
- To design a UML diagram for the Factory Method pattern.
- To demonstrate the use of the Factory Method pattern with a real-life example.

**Real-life example:** When I am building a ride-booking system where users can choose different types of rides like Economy, Luxury, or SUV**.** Instead of creating ride objects manually, you can use the Factory Method Pattern to dynamically instantiate the appropriate ride class.

**UML/User-defined Class Design:**




**Implementation in C++:**

```
#include <iostream>
using namespace std;
// Product Interface
class Transport {
public:
   virtual void deliver() = 0;
   virtual ~Transport() {}
};
```

```
// Concrete Products
class Truck : public Transport {
public:
  void deliver() override {
    cout << "Delivering goods by truck." << endl;
  }
};
class Ship : public Transport {
public:
  void deliver() override {
    cout << "Delivering goods by ship." << endl;
  }
};
// Creator Interface
class Logistics {
public:
  virtual Transport* createTransport() = 0;
  void planDelivery() {
Transport* transport = createTransport();
    transport->deliver();
    delete transport;
  }
  virtual ~Logistics() {}
};
// Concrete Creators
class RoadLogistics : public Logistics {
public:
Transport* createTransport() override {
    return new Truck();
  }
};
class SeaLogistics : public Logistics {
public:
Transport* createTransport() override {
    return new Ship();
  }
};
// Client Code
int main() {
  // Creating RoadLogistics and calling planDelivery
  Logistics* roadLogistics = new RoadLogistics();
  roadLogistics->planDelivery(); // Output: Delivering goods by truck.
```

```
    // Creating SeaLogistics and calling planDelivery
    Logistics* seaLogistics = new SeaLogistics();
    seaLogistics->planDelivery(); // Output: Delivering goods by ship.
    // Clean up memory
    delete roadLogistics;
    delete seaLogistics;
    return 0;
}
```

**Result and Discussion**

The Factory Method Pattern was successfully implemented in C++, encapsulating object creation within a factory class. This design reduces the client's dependency on specific concrete classes and offers flexibility. It hides the object creation details, ensuring that the client doesn't need to know the exact class being used. Additionally, the pattern allows for easy scalability, as new product types can be added without modifying existing client code. Centralizing object creation also helps reduce code redundancy, promoting reusability. Overall, the experiment showcased the significant advantages of using the Factory Method Pattern in object-oriented programming.