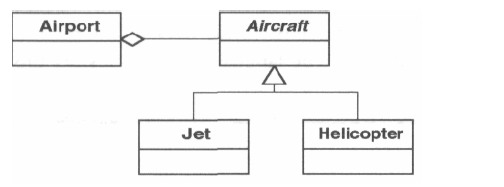
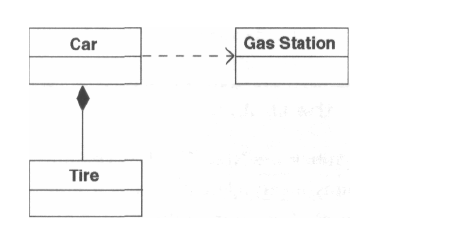
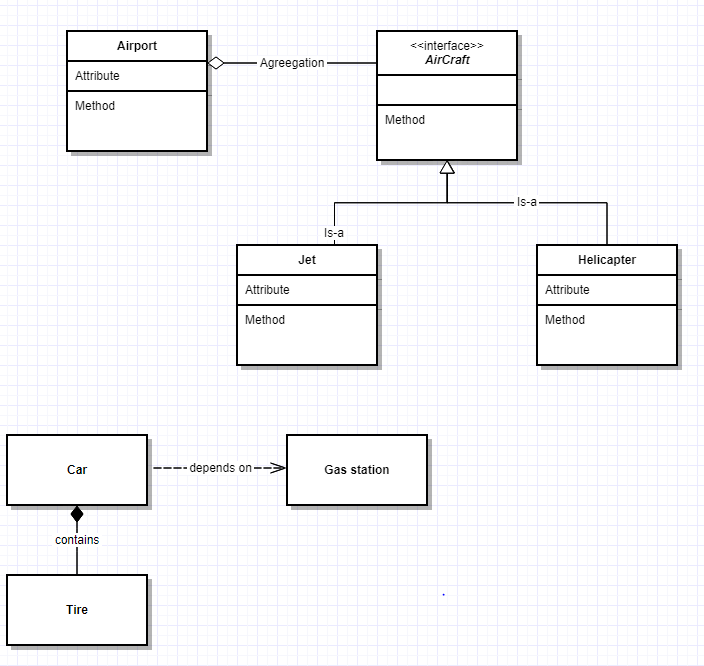
1. **Identify Relationships in the following diagrams**

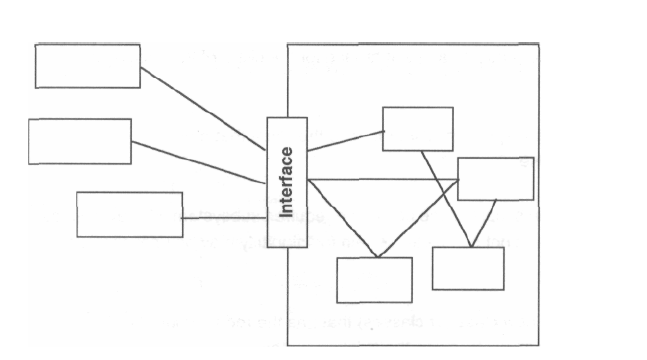


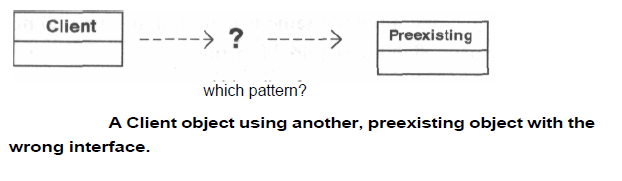


**Answer:**



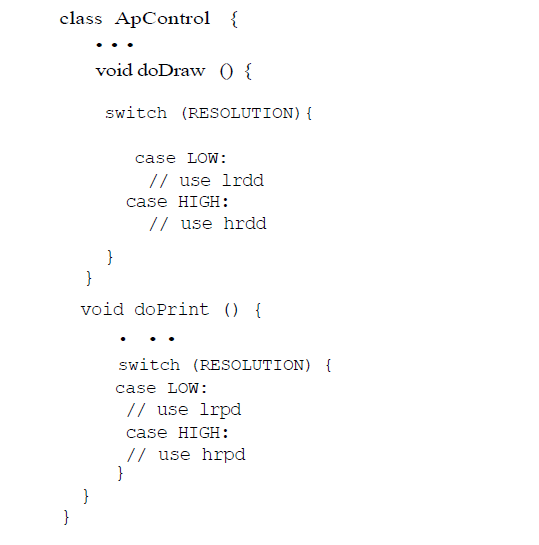
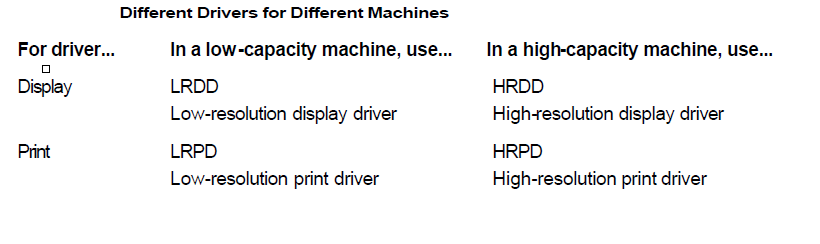
1. **Identify the Pattern**

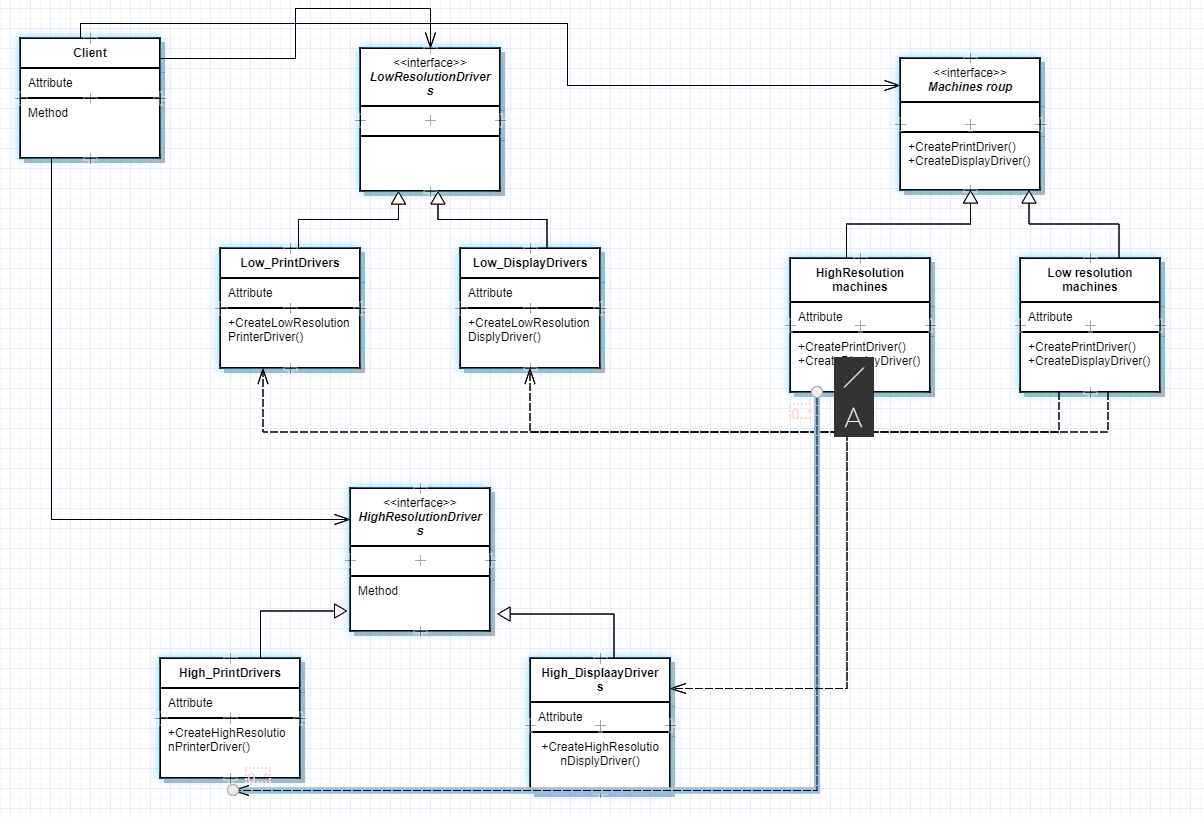




**Answer: Observer pattern**

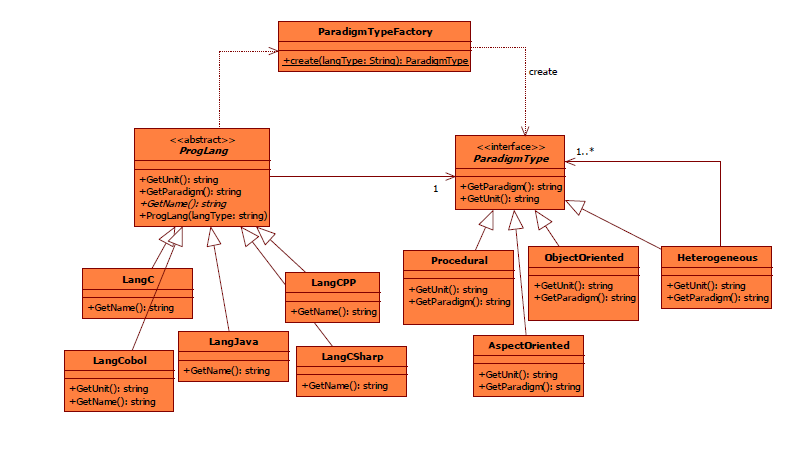
1. **Refactor the Design and Name the pattern**



**Answer:** ****

Abstract Factory pattern

1. **Identify the patterns in the following design:**



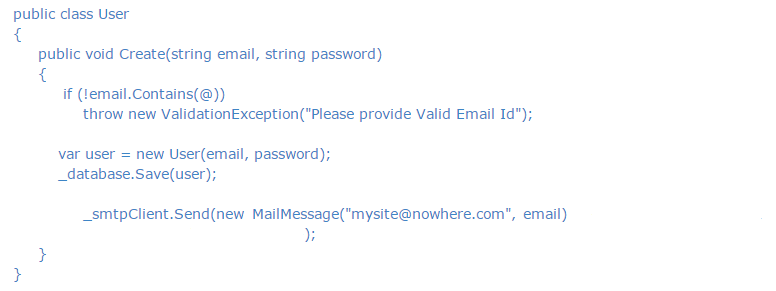
**Answer: Patterns used:**

**1. Strategy**

**2. Composite**

**3. Abstract factory**

1. **Which of the SOLID principle violated in the following code and Refactor the Design**



**Answer: Single responsibility principle**

1. **Which best defines the use of the Composite Design Pattern?**
   1. The Composite design pattern allows the clients to build a complex object of smaller different ones
   2. The Composite design pattern allows adding and removing functionality dynamically
   3. the Composite design pattern allows the clients to treat individual objects and compositions uniformly

**Answer: C**

1. **\_\_\_\_\_\_\_\_\_\_\_The relationship used to represent a class invoking static methods of another class is**
   1. association
   2. contains
   3. comoposition
   4. inheritance

**Answer: a. Association**

1. **Which of the following pattern works as a bridge between two incompatible interfaces?**
   1. Builder Pattern
   2. Adapter Pattern
   3. Prototype Pattern
   4. Bridge Pattern

**Answer: b. Adopter Pattern**

1. **Which of the following pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance?**
   1. Composite Pattern
   2. Facade Pattern
   3. Flyweight Pattern
   4. Decorator Pattern

**Answer: a. Composite Pattern**

1. **It is also known as Wrapper, it is used when sub classing is not possible or practical to add functionality and it is used to add functionality at runtime. This pattern is :**
   1. Composite
   2. Adapter
   3. Decorator
   4. Proxy

**Answer: Proxy Pattern**

1. A typical product database consists of two types of product components — product categories and product items. A product category is generally composite in nature. It can contain product items and also other product categories as its subcategories. Example Product Categories: a. Computers b. Desktops c. Laptops d. Peripherals e. Printers f. Cables the Computers product category contains both the Desktops and the Laptops product categories as its subcategories. The Desktop category can contain a product item such as Compaq Presario 5050. Product items are usually individual, in the sense that they do not contain any product component within. Design and implement an application to list the dollar value of a product component. Use the appropriate pattern to allow the client application to refer to both the product categories and the product items in a uniform manner

**Answer: Observer pattern with composite pattern can be used.**

1. Suggest the Correct Pattern

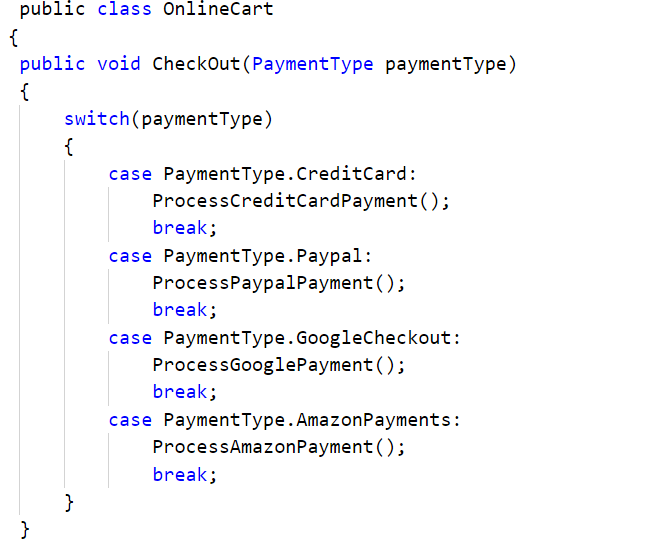
Let us build a sales reporting application for the management of a store with multiple departments. The features of the application include:

* 1. Users should be able to select a specific department they are interested in.
  2. Upon selecting a department, two types of reports are to be displayed:
     1. Monthly report — A list of all transactions for the current month for the selected department.
     2. YTD sales chart — A chart showing the year-to-date sales for the selected department by month.

Whenever a different department is selected, both of the reports should be refreshed with the data for the currently selected department.

**Answer: Strategy Pattern**

1. Refactor the Design using Appropriate Pattern



**Answer: We can use Strategy Pattern**

class IPaymentModel

{

public:

virtual void ProcessPayment() = 0;

};

class CreditCardPayment : public IPaymentModel

{

public:

void ProcessPayment()

{

cout << "Credit card payment" << endl;

}

};

class PeyPalPayment : public IPaymentModel

{

public:

void ProcessPayment()

{

cout << "Paypal payment" << endl;

}

};

class GoogleCheckOutPayment : public IPaymentModel

{

public:

void ProcessPayment()

{

cout << "Google checkout payment" << endl;

}

};

class AmazonPaymet : public IPaymentModel

{

public:

void ProcessPayment()

{

cout << "Amazon payment " << endl;

}

};

int main()

{

IPaymentModel \*amazonPayment = new AmazonPaymet();

amazonPayment->ProcessPayment();

IPaymentModel \*creditCardPayment = new AmazonPaymet();

creditCardPayment->ProcessPayment();

IPaymentModel \*PayPalPayment = new AmazonPaymet();

PayPalPayment->ProcessPayment();

IPaymentModel \*GoogleCheckOutPayment = new AmazonPaymet();

GoogleCheckOutPayment->ProcessPayment();

getchar();

}