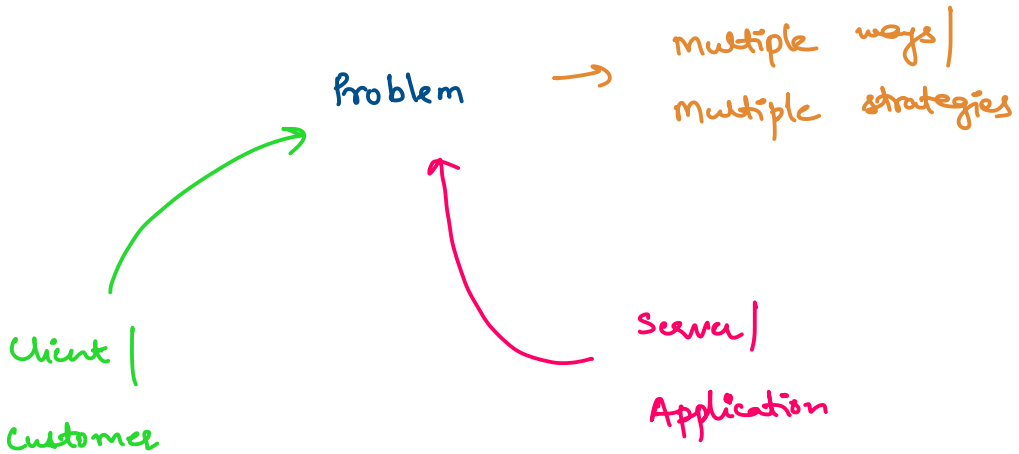
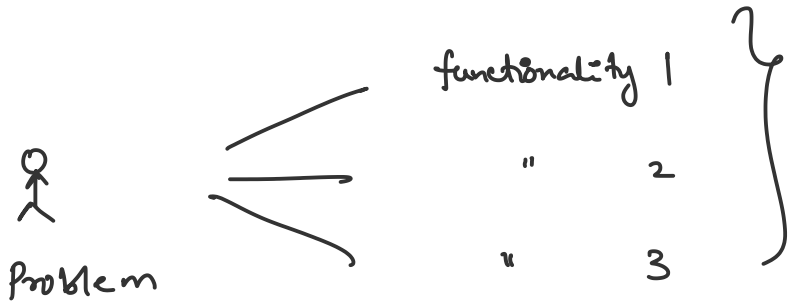
19__11-09-2022

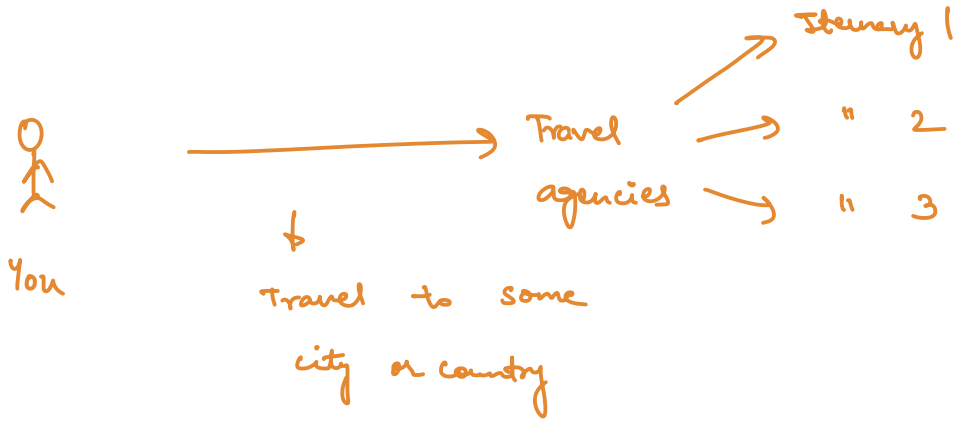
Sunday, 11 September 2022 8:03 PM

Strategy design pattern





A strategy design pattern says that - in this we will have a family of functionalities, encapsulates each one and make them interchangeable.

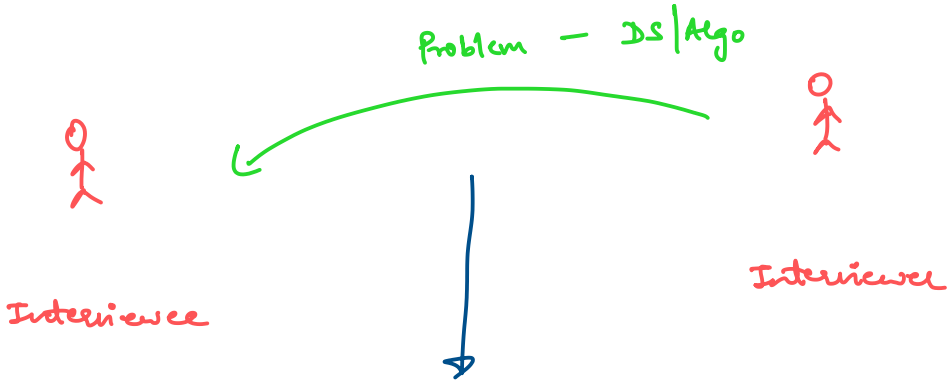


In strategy pattern, a class behavior or its algorithm can be changed at run time.

In strategy design pattern, we create objects which represent various strategies and a content object whose behaviour varies as per its strategy object. This strategy object changes the executing algorithm of

the content object.

The strategy design pattern is also known as policy.

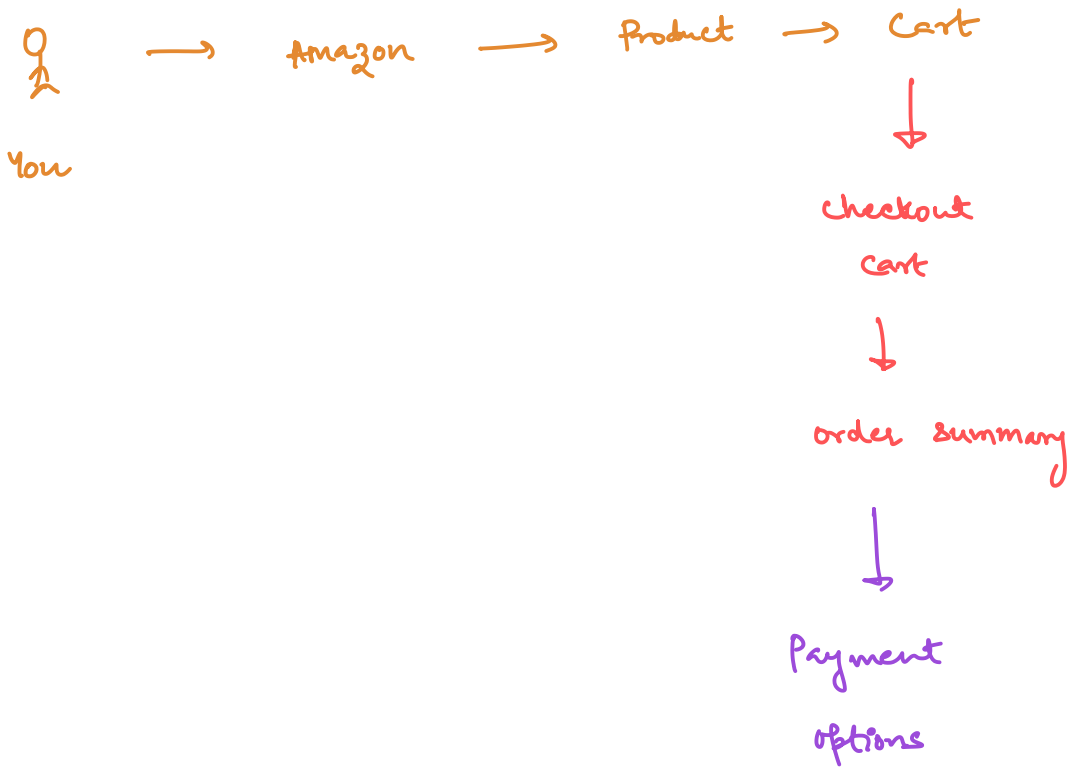


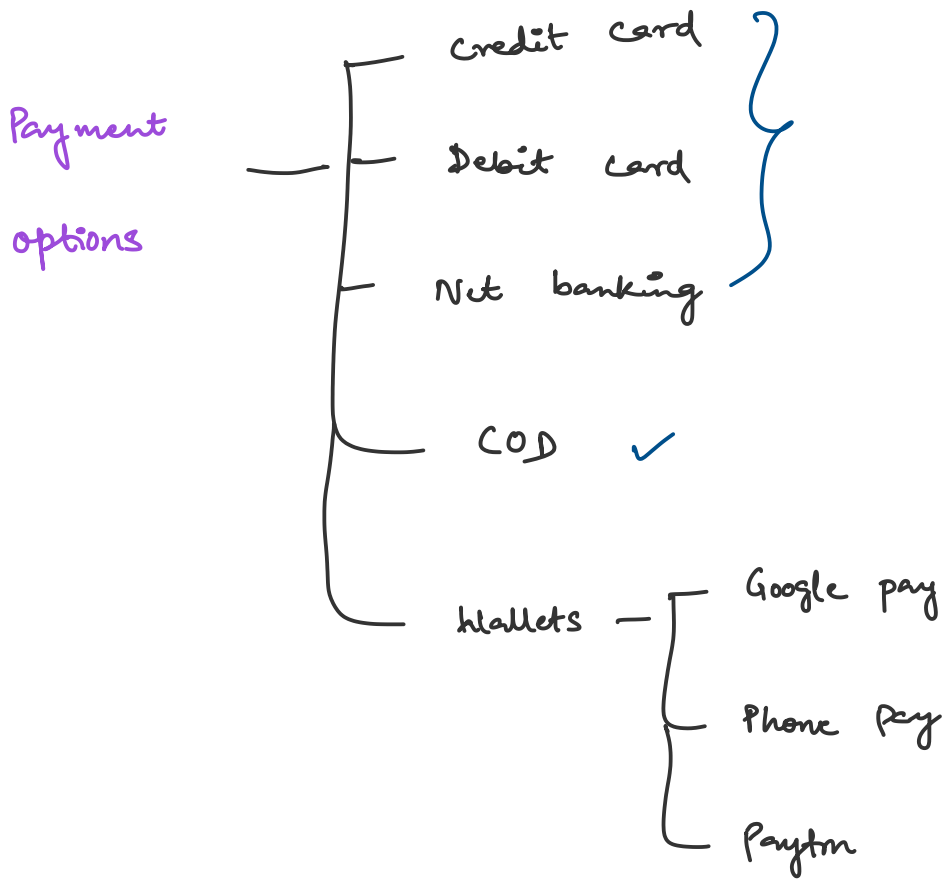
various strategies to solve the given problem–

1. Brute force
2. Optimize with extra space
3. optimize with no extra space.
4. Optimize with time complexity.

various strategies

Another eg -


You
Amazon
Product
Cart
checkout cart
order summary
Payment options


Payment options
Credit card
Debit card
Net banking
COD
Wallets
Google pay
Phone pay
Paytm

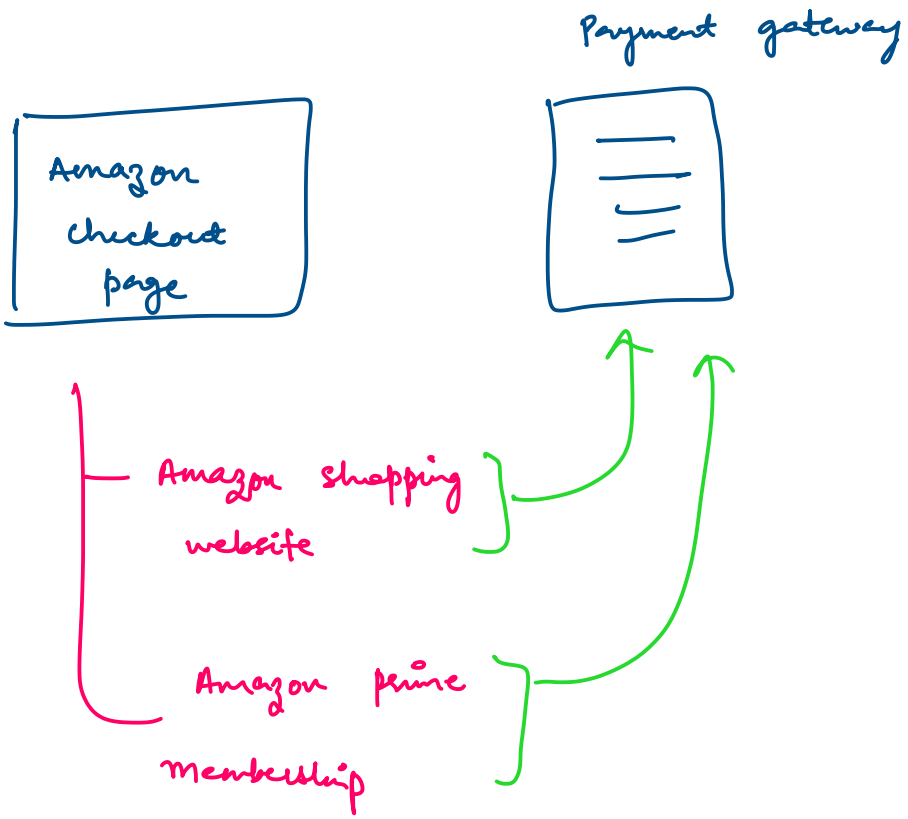
Amazon → Payment gateways → [CC, DC, NB]

↓

[Add wallet support]

↓

open | close principle

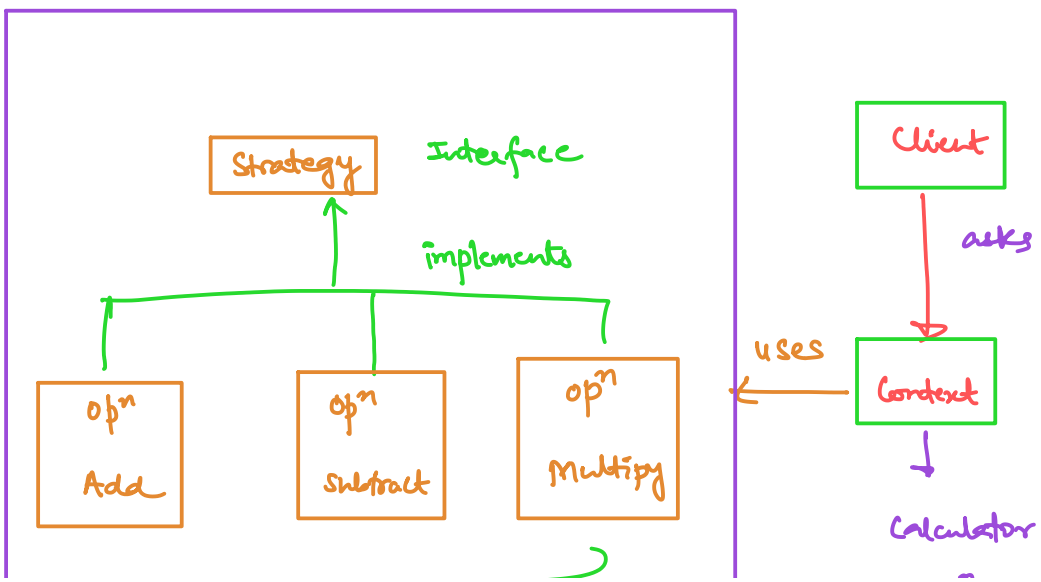


Benefits / Advantages –

1. It makes it easier to extend and incorporate new behavior without changing the application.

2. It defines each behavior within its own class, eliminating the need for conditional statements.

3. It provides a substitute for sub-classing.



Usages of strategy design pattern –

1. When we need the different variations of an algorithm.

2. When the multiple classes differ only in their behaviors.

Problem statement

↳ Design payment gateway with the payment options as –

wallet – Google pay
Phone pay

Steps –

1. Create an interface

IWallet Strategy
↳ pay()

2. Create concrete classes implementing the same interface.

concrete Classes –
- Google Pay
- Phone Pay

3. Create context class.

Product –
- product name
- product price

4. Create client class.

↳ main method.

```
public interface IWalletStrategy {
   public void pay(int amount);
}
```

```
public class GooglePay implements IWalletStrategy {
   @Override
   public void pay(int amount) {
      System.out.println("Paying by google pay, amount : " + amount);
   }
}
```

```
public class PhonePay implements IWalletStrategy {
   @Override
   public void pay(int amount) {
      System.out.println("Paying by phone pay, amount : " + amount);
   }
}
```

```
public class Product {
   private String productName;
   private int productPrice;

   public Product(String productName, int productPrice) {
      this.productName = productName;
      this.productPrice = productPrice;
   }

   public String getProductName() {
      return productName;
```

```
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public int getProductPrice() {
        return productPrice;
    }

    public void setProductPrice(int productPrice) {
        this.productPrice = productPrice;
    }
}
```

```
public class AmazonCart {
    private Product product;
    private IWalletStrategy walletStrategy;

    public AmazonCart(Product product, IWalletStrategy walletStrategy) {
        this.product = product;
        this.walletStrategy = walletStrategy;
    }

    public void pay()
    {
        this.walletStrategy.pay(product.getProductPrice());
    }
}
```

```
public class Program {
    public static void main(String[] args) {
        //this will consist of phone specifications
        Product product = new Product("Phone", 10000);

        //Provide strategy to the client for making payment
        AmazonCart amazonCart = new AmazonCart(product, new GooglePay());
        amazonCart.pay();
    }
}
```

Output:
Paying by google pay, amount : 10000