Build Your Own Payment System

This activity helps students understand:

- Interfaces
- Polymorphism
- Dynamic Method Dispatch
- Loose Coupling

📝 Activity Plan

Title: Build Your Own Payment System

Objective:

Students will create different types of payment processors using classes and explore how polymorphism allows flexibility and modular design.

Step-by-Step Tasks:

1. **Starter Code:**
    - Provide students the basic `Payments` class necessarily having a processPayments method that accepts a double amount.
    - Ask them to implement 2-3 different payment methods (via classes that extend Payments Class) and overrides the processPayment Method:
        - `UPIPayments`
        - `CreditCardPayments`
        - `PayPalPayments`
2. **Portal Design:**
    - Implement `PaymentPortal` class

      In this class make a method OpenPortal which takes a `Payments` object and a double amount as parameters and processes payments.
3. **Test Class:**
    - Write a `main` method to:
        - Instantiate different payment types.
        - Pass them to the portal.
        - Observe runtime polymorphism in action.

Code -

Payments Class

```
public class Payments {

    public void processPayments(double amt){

        //nothing to write here

    }

}
```

UPIPayments

```
class UPIPayments extends Payments{

    @Override

    public void processPayments(double amt){

        System.out.println("A payment of "+amt+" rs is processed using UPI.");

    }

}
```

CreditCardPayments

```
class CreditCardPayments extends Payments {

    @Override

    public void processPayments(double amt){

        System.out.println("A payment of "+amt+" rs is processed using Credit
Card.");

    }

}
```

PayPalPayments

```
class PayPalPayments extends Payments {

    @Override
```

```
    public void processPayments(double amt){

        System.out.println("A payment of "+amt+" rs is processed using Pay
Pal.");

    }

}
```

PaymentPortal Class

```
public class PaymentPortal {
    public void OpenPortal(Payments p, double amt){
        System.out.println("Portal Open");
        p.processPayments(amt);
    }
}
```

Test Class

```
class testPay{
    public static void main(String[] args) {
        Payments p1 = new Payments();
        PaymentPortal Mypay = new PaymentPortal();
        UPIPayments p2 = new UPIPayments();
        Mypay.OpenPortal(p2, 1000.12);
        CreditCardPayments p3 = new CreditCardPayments();
        Mypay.OpenPortal(p3, 10000);
        Mypay.OpenPortal(new PayPalPayments(), 120);
    }
}
```

Observations :

1. **Using the Payments Class as a Common Type**

```
Payments p1 = new UPIPayments();
```

This means:

- p1 is declared as a Payments type (the interface).
- But it actually holds a UPIPayments object.

Java allows this because UPIPayments **implements** the Payments interface.

2. **How Java Knows Which Method to Run**

```
p1.processPayments(100);
```

Java looks at the **object inside** p1 (which is UPIPayments), **not** the reference type. Call is resolved based on the actual type of the object.

So it runs the method from the UPIPayments class

3. **How is this useful ?**

```
public void OpenPortal(Payments p, double amt){
```

In PaymentPortal class the method OpenPortal accepts a Payments object, so now it is not tightly coupled to any Payments like but can accept any type of Payments Object here.

Here, the PaymentPortal class does **not care** whether it's dealing with:

- UPI
- Credit Card
- PayPal
- Or any future payment method

As long as the class extends the Payments class, it works.

Further Discussion :

1. The Payments class does not represent any real life object but represents the idea or the abstract of the Payment Methods

2. The processPayment Method in the Payment class does not represent a concrete method as we have no logical idea how the Payment class will process the payment.

3. In simple words, the processPayment method we can only define **What to do?** But we don't want to define **How to do it?**

4. It makes logical sense that we make that method ***an abstract method***
   We should define it as :
   ```
   public abstract void processPayment(double amount);
   ```
   And thus also making the Payments class as an abstract class :
   ```
   public abstract class Payments{
       public abstract void processPayment(double amount);
   }
   ```