In [20]:
```python
from abc import ABC, abstractmethod
#to make class as abstarct by ABC module
#to make method as abstract by using @abstractmethod decorator
class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass
    @abstractmethod
    def stop(self):
        pass
class Car(Vehicle):
    def start(self):
        print("Car started with key")

    def stop(self):
        print("Car stopped")
# ✅ Usage
c = Car()
c.start()
c.stop()


#v = Vehicle()  # Error! Can't instantiate abstract class
```

```
Car started with key
Car stopped
```

In [22]:
```python
from abc import ABC, abstractmethod
class Account(ABC):
    @abstractmethod
    def deposit(self, amount):
        pass
    @abstractmethod
    def withdraw(self, amount):
        pass
class SavingsAccount(Account):
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited ₹{amount}. New balance: ₹{self.balance}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ₹{amount}. Remaining balance: ₹{self.balance}")
        else:
            print("Insufficient balance")
# ✅ Usage
acc = SavingsAccount(1000)
acc.deposit(500)
acc.withdraw(800)
```

```
Deposited ₹500. New balance: ₹1500
Withdrew ₹800. Remaining balance: ₹700
```

In [14]:

```python
from abc import ABC, abstractmethod

# Abstract class for payment method
class PaymentMethod(ABC):

    @abstractmethod
    def pay(self, amount):
        pass

# Concrete class for UPI payment
class UPIPayment(PaymentMethod):
    def pay(self, amount):
        print(f"Paid ₹{amount} via UPI.")

# Concrete class for Credit Card payment
class CreditCardPayment(PaymentMethod):
    def pay(self, amount):
        print(f"Paid ₹{amount} using Credit Card.")

# Concrete class for Wallet payment
class WalletPayment(PaymentMethod):
    def pay(self, amount):
        print(f"Paid ₹{amount} from Wallet.")

# Order class that takes a payment method (abstraction in action)
class Order:
    def __init__(self, payment_method: PaymentMethod):
        self.payment_method = payment_method

    def checkout(self, amount):
        print("Processing payment...")
        self.payment_method.pay(amount)
        print("✅ Payment successful!\n")

# ✅ Usage
upi = UPIPayment()
card = CreditCardPayment()
wallet = WalletPayment()

order1 = Order(upi)
order2 = Order(card)
order3 = Order(wallet)

order1.checkout(1200)
order2.checkout(2500)
order3.checkout(500)
```

```
Processing payment...
Paid ₹1200 via UPI.
✅ Payment successful!

Processing payment...
Paid ₹2500 using Credit Card.
✅ Payment successful!

Processing payment...
Paid ₹500 from Wallet.
✅ Payment successful!
```

In [ ]: