1. **SOLID – S – Single Responsability**

class Engine:

    def \_\_init\_\_(self):

        pass

    def getRPM(self):

        return 3000

class Vehicle:

    def \_\_init\_\_(self, name, speed):

        self.\_name = name

        self.\_speed = speed

        self.\_engine = Engine()

    def getName(self):

        return self.\_name

    def getEngineRPM(self):

        return self.\_engine.getRPM()

    def getMaxSpeed(self):

        return self.\_speed

    def printInfo(self):

        print(

            "Vehicle: {}, Max Speed: {}, RMP: {}".format(

                self.\_name, self.\_speed, self.\_engine.getRPM()

            )

        )

if \_\_name\_\_ == "\_\_main\_\_":

    vehicle = Vehicle("Car", 200)

    vehicle.printInfo()

class Engine:

    def getRPM(self):

        return 3000  # valor por defecto del motor

class Vhicle:

    def \_\_init\_\_(self, name, speed, engine):

        self.\_name = name

        self.\_speed = speed

        self.\_engine = engine

    def getName(self):

        return self.\_name

    def getEngimeRPM(self):

        return self.\_engine.getRPM()

    def getMaxSpeed(self):

        return self.\_speed

class VehiclePrinter:

    def \_\_init\_\_(self, vehicle):

        self.\_vehicle = vehicle

    def printInfo(self):

        print(

            "Vehicle: {}, Max Speed: {}, RPM: {}".format(

                self.\_vehicle.getName(),

                self.\_vehicle.getMaxSpeed(),

                self.\_vehicle.getEngimeRPM(),))

class VehiclePersistance:

    def \_\_init\_\_(self, vehicle, db):

        self.\_vehicle = vehicle

        self.\_persistance = db

        print("Hey, storing data! in", self.\_persistance)

if \_\_name\_\_ == "\_\_main\_\_":

    engine = Engine()

vehicle = Vehicle(name="Car", engine=engine, speed=200)

    persistance = VehiclePersistance(vehicle=vehicle, db="SQL")

    printer = VehiclePrinter(vehicle=vehicle)

    printer.print

class Order:

    def \_\_init\_\_(self):

        self.items = []

        self.quantities = []

        self.prices = []

        self.status = "open"

    def add\_item(self, name: str, quantity: int, price: float) -> None:

        self.items.append(name)

        self.quantities.append(quantity)

        self.prices.append(price)

    def total\_price(self):

        total = 0

        for quantity, price in zip(self.quantities, self.prices):

            total += quantity \* price

        return total

    def pay(self, payment\_type: str, security\_code):

        if payment\_type == "debit":

            print("Processing debit payment type")

            print(f"Verifying security code: {security\_code}")

            self.status = "paid"

        elif payment\_type == "credit":

            print("Processing credit payment type")

            print(f"Verifying security code: {security\_code}")

            self.status = "paid"

        else:

            raise Exception(f"Unknown payment type: {payment\_type}")

order = Order()

order.add\_item("Keyboard", 1, 50)

order.add\_item("SSD", 1, 150)

order.add\_item("USB cable", 2, 5)

print(order.total\_price())

order.pay("debit", "0372846")

class Order:

    def \_\_init\_\_(self):

        self.items = []

        self.quatities = []

        self.prices = []

        self.status = "open"

    def add\_item(self, name: str, quantity: int, price: float) -> None:

        self.items.append(name)

        self.quatities.append(quantity)

        self.prices.append(price)

class PaymentProcesor:

    def pay(self, order: Order, security\_code: str, payment\_type: str):

        if payment\_type == "debit":

            print("Processing debit payment type")

            print(f"Verifying security code: {security\_code}")

            order.status = "paid"

        elif payment\_type == "credit":

            print("Processing credit payment type")

            print(f"Verifying security code: {security\_code}")

            order.status = "paid"

        else:

            raise Exception(f"Unknown payment type: {payment\_type}")

class CalculateProcesor:

    def total\_price(self, order=Order):

        total = 0

        for quantity, price in zip(order.quatities, order.prices):

            total += quantity \* price

        return total

order = Order()

print(order.status)

order.add\_item("Laptop", 3, 150)

processor = PaymentProcesor()

processor.pay(order, "12345", "debit")

print(order.status)

total = CalculateProcesor()

print(total.total\_price(orden))