Warm-up Activity 3

✅ SOLID Principles Warm-Up Activity (All-in-One)

Each section demonstrates a common violation.

1. SRP Violation – Single Responsibility Principle

```
class ReportGenerator:
    def __init__(self, employee_name, hours_worked):
        self.employee_name = employee_name
        self.hours_worked = hours_worked

    def calculate_salary(self):
        return self.hours_worked * 100

    def print_report(self):
        print(f"{self.employee_name} worked {self.hours_worked} hours and earned
${self.calculate_salary()}")
```

👉 What's wrong? Suggest how to split this class.

2. OCP Violation – Open/Closed Principle

```
class DiscountCalculator:
    def calculate_discount(self, customer_type):
        if customer_type == "regular":
            return 5
        elif customer_type == "vip":
            return 20
        else:
```

```
        return 0
```

👉 What's the issue if more types are added? How can you apply polymorphism?

3. LSP Violation – Liskov Substitution Principle

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def set_width(self, width):
        self.width = width

    def set_height(self, height):
        self.height = height

    def get_area(self):
        return self.width * self.height

class Square(Rectangle):
    def set_width(self, width):
        self.width = width
        self.height = width

    def set_height(self, height):
        self.height = height
        self.width = height
```

👉 What happens when you use Square in place of Rectangle?

4. ISP Violation – Interface Segregation Principle

```
class MultiFunctionPrinter:
    def print_document(self):
        pass

    def scan_document(self):
        pass

    def fax_document(self):
        pass

class BasicPrinter(MultiFunctionPrinter):
    def print_document(self):
        print("Printing...")

    def scan_document(self):
        raise NotImplementedError("This printer can't scan")

    def fax_document(self):
        raise NotImplementedError("This printer can't fax")
```

👉 What principle is broken? How should you split interfaces?

5. DIP Violation – Dependency Inversion Principle

```
class EmailSender:
    def send_email(self, message):
        print(f"Email sent: {message}")
```

```
class NotificationManager:
    def __init__(self):
        self.sender = EmailSender()  # Tight coupling

    def notify(self, message):
        self.sender.send_email(message)
```

👉 How would you inject an abstract interface instead?