Instructions

1. ให้ใช้ gen ai 2 ตัว สร้างโค้ดที่เป็น adaptor pattern

2. อ่าน วิเคราะห์ และตรวจสอบโค้ดที่ได้มาจาก gen ai แต่ละตัวว่า เป็น adaptor pattern จริงหรือไม่ โดยชี้จุดที่แสดงถึงคุณลักษณะสำคัญที่ทำให้เป็น adaptor pattern และอธิบาย

Gemini 2.5 Pro

```java
public interface Customer {
    String getFullName();
    String getCustomerID();
}
```

```java
public class EmployeeAdapter implements Customer {

    private Employee employee;

    public EmployeeAdapter(Employee employee) {
        this.employee = employee;
    }

    @Override
    public String getFullName() {
        return employee.getFirstName() + " " + employee.getLastName();
    }
    @Override
    public String getCustomerID() {
        return "EMP_" + employee.getEmployeeId();
    }
}
```

```java
public class Employee {
    private String firstName;
    private String lastName;
    private String employeeId;

    public Employee(String firstName, String lastName, String employeeId) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.employeeId = employeeId;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String getEmployeeId() {
        return employeeId;
    }
}
```

```java
public class Main {

    public static void main(String[] args) {
        Employee employee = new Employee("Jane", "Smith", "98765");

        Customer employeeAsCustomer = new EmployeeAdapter(employee);

        System.out.println("--- Preparing to process transaction ---");
        System.out.println("Original Employee First Name: " + employee.getFirstName());
        System.out.println("Original Employee ID: " + employee.getEmployeeId());

        System.out.println("\n--- Using the Adapter ---");

        processTransaction(employeeAsCustomer);
    }

    public static void processTransaction(Customer customer) {
        System.out.println("Processing transaction for Customer:");
        System.out.println("  Full Name: " + customer.getFullName());
        System.out.println("  Customer ID: " + customer.getCustomerID());
        System.out.println("Applying employee discount... Transaction complete.");
    }
}
```

เป็น Adapter Pattern เพราะหลักการคือ มี class ที่มีอยู่แล้วคือ Employee แต่ Employee อาจมาเป็น Customer และต้องการรู้ชื่อลูกค้าเต็มๆ เลยแต่ Employee ไม่ provide มาให้จึงต้องมี Adapter interface ของ Customer เพื่อให้สามารถทำได้ แต่ EmployeeAdapter class อาจไม่ make Sense เพราะรหัสลูกค้าถูกนำมาจาก employeeId โดยหลักการที่มีคลาสที่มี interface ที่ต้องการมาทำ adapter กับ คลาสที่มีอยู่แล้วจึงถือว่าเป็น adapter pattern

ChatGPT 4.1

```java
public class AmericanPlug {
    public void connectWithTwoFlatPins() {
        System.out.println("Connected with two flat pins (American plug).");
    }
}
```

```java
public interface EuropeanSocket {
    void connectWithTwoRoundPins();
}
```

```java
public class AmericanToEuropeanAdapter implements EuropeanSocket {
    private AmericanPlug americanPlug;

    public AmericanToEuropeanAdapter(AmericanPlug americanPlug) {
        this.americanPlug = americanPlug;
    }

    @Override
    public void connectWithTwoRoundPins() {
        System.out.println("Adapter converts round pins to flat pins.");
        americanPlug.connectWithTwoFlatPins();
    }
}

```

```java
public class Main {
    public static void main(String[] args) {
        AmericanPlug americanPlug = new AmericanPlug();
        EuropeanSocket socket = new AmericanToEuropeanAdapter(americanPlug);

        socket.connectWithTwoRoundPins();
    }
}

```

เป็น Adapter Pattern เพราะจากคลาสมี Interface ที่ต้องการคือ EuropeanSocket แต่มีคลาสที่มีอยู่แล้วเป็น AmericanPlug ซึ่งต้องการใช้งานแบบ EuropeanSocket จึงมีการ implements คลาส AmericanToEuopeanSocket ทำให้สามารถใช้งานคลาส AmericanPlug ได้ ตรงหลักการใช้งาน Interface ที่ต้องการกับคลาสที่มีอยู่แล้วจึงเป็น Adapter Pattern