**Questions**

1. What type of a design pattern did you employ (creational, structural, behavioural, custom)? Why?
2. What is inversion of control?

Inversion of control is a design pattern that allows classes to be flexibly combined and therefore easier to test and maintain. Moreover, transferring the control of objects from one main program to a one framework such as express in JavaScript. Ioc enables a program to focus on its main work rather than worrying about other work.

Some examples namely are dependency injection, service locator pattern and event-based programs.

Referred from: https://www.educative.io/answers/what-is-inversion-of-control

1. The open/close principle states that any class, component or entity should be open for extension but closed for modification. How could one extend a class without modifying the code?

One of the ways we have learnt in class is:

**Inheritance and Subclassing**: You can extend a class by creating a subclass that inherits from the original class. The subclass can add new methods or override existing ones to provide extended or specialized behavior. This is known as "subclassing for extension."

class Shape {

// Common shape properties and methods

}

class Circle extends Shape {

// Additional properties and methods specific to circles

}

**Interfaces and Implementations**: Define interfaces that specify the contract that classes must adhere to, and then provide new implementations of those interfaces to extend functionality. This allows you to swap implementations without modifying existing code.

We have used a similar method of interfaces in our assignment 1 in order to create an interface between our toolbox and grocery versus customer, shopping cart and retail store factory class.

interface PaymentProvider {

    void processPayment();

}

class CreditCardPaymentProvider implements PaymentProvider {

    // Implement processPayment for credit cards

}

class PayPalPaymentProvider implements PaymentProvider {

    // Implement processPayment for PayPal

}

1. How are design patterns different from algorithms?

The main differences between design patterns and algorithms are their abstraction level, purpose, applicability, and the types of problems they address. Design patterns are concerned with high-level design and architectural issues, offering reusable solutions to design challenges. Algorithms, on the other hand, are specific procedures for solving computational problems efficiently and are focused on implementation details. While they serve different roles, both design patterns and algorithms are important aspects of software development. Design patterns help create maintainable and flexible software structures, while algorithms enable efficient problem-solving within those structures.