Structural Patterns Lab – Theory

a) Describe briefly the fundamental characteristics of Structural design patterns.

One fundamental characteristic is that structural patterns use inheritance to compose interfaces or implementations. One example is using multiple inheritance mixes two or more classes into one. Which makes it useful for independently developed class libraries work together. One key characteristic of Structural pattern is that rather than composing interfaces or implementations, structural object patterns describe the ways to compose objects to realize new functionality.

b) What is the difference between an Abstract and Concrete class?

The abstract class mainly serves as a blueprint meaning that it holds the bare functions for the program but also it cannot be instantiated. Or in other words Abstract classes often serve as abase for other classes since function signatures are being extended to the subclass. On the other hand Concrete can be instantiated directly and essentially it represents a fully realized object ready to use.

C) Distinguish clearly between class inheritance and interface inheritance

Class inheritance mainly involves a subclass inheriting the attributes and methods from a parent class. It promotes code reuse and creates a easily maintainable codebase. For example Car inherits from Vehicle then a car is a vehicle.

Interface Inheritance on the other hand is about defining a set of methods without giving any implementations details.

So the clear difference between the both of them is that class inheritance it emphasizes on extending existing behavior and interface inheritance just enforces a particular set of behaviors over some classes

d) What is the intent of the Adapter design pattern?

The intent of the adapter design pattern is to convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

e) What is the intent of the Decorator design pattern

The intent of the decorator pattern is to attach responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.