Activity 18-1: Design Patterns Review

A. What are design pattens?

Discuss the following statements and come to an agreement about whether each statement is true or false.

1. A software design pattern is a general, reusable solution to a problem that commonly occurrs in a given context when designing software.
2. Design patterns typically show relationships and interactions between classes or objects.
3. In a software development project, which design patterns to use is decided early on in the development process.
4. Design patterns formalize best practices for solving common programming problems.
5. Design patterns typically make code less complex.
6. Design patterns typically make code less flexible.
7. Code that solves a problem using a design pattern will be easy to plug in and reuse in other software projects to solve similar problems.

B. Design pattern roundup

The leftmost column of the table below has descriptions of different design patterns, and the rightmost column describes example contexts when design patterns might be used. Match the descriptions on the left and the right to the patterns. Discuss any disagreements until you come to a consensus.

Pattern Description	Pattern Name	Example Use
Provides a simpler, unified interface that hides the complexity of working with a group of objects.	Command	Creating a base class that implements a list sorting algorithm with subclasses that contain the specifics for sorting different types of objects (e.g., ints, doubles, Persons).
Provides a generic way of creating different types of related objects by using an interface to abstract away the concrete class implementations.	Template Method	Creating an empty_node class to represent empty subtrees in a tree ADT.
A subclass that implements a "nothing" version of a base class.	Factory	Allowing a program for playing a game of scrabble to have only one game board object.

Pattern Description	Pattern Name	Example Use
Uses a base to define a general algorithm with subclases being used to specify variants of the general algorithm.	Facade	Treating files and collections of files (i.e., directories) the same way in a system for managing a file system.
Encapsulates the communication between two or more objects so that the objects no longer communicate directly with each other.	Mediator	Abstracting away details about the specific methods invoked by toolbar buttons or menu items so that their content can be easily changed.
Allows for multiple objects with the same interface to be treated as if they were a single object.	Null Object	Event handling systems.
Objects sign up with a special object tasked with watch for something to change; when the change happens, the object that is watching notifies other objects of the change.	Singleton	Writing a library of functions to wrap-around and simplify the use of a complex graphics library.
Encapsulates information to perform an action so that the system can invoke the action without knowing anything specifics about the it.	Observer	Defining an interface for the generic creation of different types of monsters for a dungeon-crawl game.
Ensures that a class never has more than one instance at any given time.	Composite	Creating one or more classes to manage the user connections and interactions in a chat room.