Josh Enes

creational

- Simple factory: A pattern that has a large conditional whose purpose is to determine whether to create an object.

- Factory method: A method that is used to call a constructor and create an object. Unlike the abstract factory, the factory method doesn't use inheritance to accomplish this.

- Abstract Factory: A pattern that has a superclass which has subclasses that are able to create a family of new objects. The superclass containing information that is used for all the objects in the family, and the variable parts of each object in the family is stored in subclasses.

Prototype: Prototypes use an object as a "base" to create other similar objects or clones of the "base" object. This is similar to a prototype in real life because they are used as tools that represent all or most of the final product. Prototypes help reduce error because instead of building a new object from scratch, it uses a preexisting one as a reference.

Builder: This Pattern is used for creating complex objects/structures. Instead of having a big object with a huge constructor where everything is stored in one class, the builder "builds" this big/complex object using smaller objects, kind of like the building of a house.

Singleton: Singletons are classes with a private constructor that make sure that only one instance of a certain object is created at a time, while providing global access to this object.

Structural

Adapter: This Pattern is used to allow 2 different objects to communicate with eachother. This is like an adapter in real life (like a USB to USBC cable), which helps 2 devices with different connectors communicate.

Bridge: A bridge is a link between the functional Part of the code and the implementation of the code. It provides seperation between the code and its implementation and allows the code to be changed more easily without affecting the

user as much.

Composite: Composite patterns allow objects to be stored in a tree hierarchy and to be combined and stored in an object that possesses the attributes of both objects that make it up. This can be described by the chair/seat example from the book by Avinash Kak.

Decorator: This pattern adds extra layers of functionality to an object. One can think of this like a Russian nesting doll because the extra functionality is added on top of the base object.

Facade: Facades are used to hide extra unwanted functionality from the user. This can be thought of as how a window in a shop allows you to see some of what's inside, but not every thing.

Flyweight: This pattern allows a bunch of objects to receive traits from a single place, instead of having each object individually store that information. This is like how a bullet in an FPS stores a single texture for all bullets, but each one stores its vector and other bullet-specific info.

Behavioral

State: This Pattern allows an object to change functionality without changing the object itself, based on criteria specified. An example of this is the function of buttons on a payment page changing functionality as you progress through the steps.

Template method: This pattern allows a superclass to store the skeleton of an algorithm, and subclasses to store specialized steps for each case. This is used a lot in processing different types of files or reading from a collection because there is a different use case for each object type

Observer: observers monitor code to check if a certain action was committed. This can be compared to an event handler because they both "observe" to see if actions have happened.

Iterator: Iterators loop through a collection of objects without exposing information about their implementation.

Chain of responsibility: Allows objects to be passed up a chain of objects until it is handled. An example of this is an event handler, and a real life example is tech support

problem escalation

Mediator: controls the order and time that certain actions are performed in a complex software system.

visitor: provides "accept" hooks that allow external code to break encapsulation and add functionality to a class.

Proxy: proxies are objects that are interfaces for interacting with a real object. Often used for security reasons. An ATM is an example of a proxy because you are going through the ATM to access your bank account.

memento: An object that saves states of your code. used a lot in the code for undo/redo buttons.

Interpreter: An object that reads input and converts it to output that your code can understand. This is used in compilers to translate from a language you code in to machine.

Command: An object that stores 1 or more command inside of it in the form of a request.

Strategy: An object that has subclasses for different cases that contain instructions for handling each case. Unlike the State Pattern, Strategy patterns do not transition from one to another, instead running all the way through and then stopping.

Null object: A Placeholder for nothing. Useful for implementing the "do nothing" functionality in your code. An example of this is the exception we made in Homework 2 (LISP List) called UnsupportedOperationException.

Model-View-Controller: This composite pattern breaks the code up into 3 parts: a model(s), a view(s), and a controller(s).

a real life example