A. Creational

a. Simple Factory
    i. Simple Factories refer to one large conditional to decide which kind of object to make and when. It acts as an intermediary that can choose to call a constructor given a set of conditions are met. This provides increased separation between the code where the parameters are held and the code where the objects created are used.

b. Factory Method
    i. A Factory is a pattern that creates objects by referring to a small number of interfaces instead of only using a constructor to do so. Used to "increase the separation between the producers and the consumers of the objects constructed from certain designated classes." Factory Methods use method calls which call constructors to create new objects.

c. Abstract Factory
    i. An Abstract Factory is a factory that uses abstract classes in order to call constructors to create new objects. These patterns can produce families of closely related objects.

d. Prototype
    i. Prototypes are used to create objects based on an existing object, used in the same way a "prototype" would be used in real life. There is less room for error when creating objects this way instead of from the ground up because if the consumer doesn't specify certain parameters, etc. the object being used as the prototype will be able to fill in those blanks.

e. Builder
    i. A builder pattern is used to determine whether an object should be built or not, and if it should, whether to build it in its entirety or piece by piece. These are normally used to create complex objects. The builder uses conditionals to determine if certain object creation constraints are met, and will construct some or all of the object based on the specified conditions.

f. Singleton
    i. Singletons are used to limit the production of certain objects to make sure that too many aren't created. Generally this pattern is used to ensure that there is only one instance of a certain object in use at a given time, and that if further objects are created, it will return the already-created object.

B. Structural

a. Object Pool
    i. Object pools are used to keep a set of initialized objects on deck, ready to use, instead of destroying the objects after they are created. The main reason to use this pattern would be for object types that have a very high cost to make, ones that don't get used a lot, or both.

b. Adapter
    i. Adapters are used to "adapt" code from one method/object and use it for another without rewriting the original method/constructor. It allows objects with incompatible interfaces to interact.

c. Bridge
    i. Bridges are in charge of connecting the entities that specify concepts to the entities that implement these concepts. Bridges ensure that the implementation code hierarchy is not permanently bound to that of the concept abstraction code.

d. Composite
    i. Composite patterns are used as a "container" for 2 or more objects. It can be used to contain 2 or more subclasses that are children of a superclass, while itself inheriting all of the features and methods of the superclass, creating an easy and modular way of dealing with object types. (See "seat" example from book.)

e. Decorator
    i. Decorators are used to provide additional embellishment code to supplement the basic functionality. These are used to give the user the freedom to customize code and mix and match options for the code without hurting the underlying integrity of the original code.

f. Facade
    i. Facade patterns are used "to create different usage views of a system of classes for different categories of users." An example of a facade would be to have different modes on a program that correspond with different levels of experience.

g. Flyweight

    Flyweights are used to store "core forms" of an object and to take one of those forms, clone it, and add any user-specified embellishment on top of the copied core form. A reason to use flyweights is that it reduces the likelihood of an error occurring from making a new object from scratch, as the facade just keeps a copy of all the core forms of the objects that are needed. This is especially important in cases where the core object has a lot

of parameters/attributes that would have to be transferred from one object to another.