# Software Design Patterns

CS-370: Software Design and Development

Robert Bruce

# What are design patterns in programming?

- **Design patterns:** Strategically defines the application programmer interface to your classes/methods (in an object-oriented programming language) or a modular series of interoperable functions (for non-object oriented programming).
- *Rob note: design patterns are a very powerful technique. With them, you can strategically create wrapper functions – our textbook classifies wrapper functions as structural patterns – to insulate external dependency libraries from your internal calls.*
- *Rob note: Utilizing wrapper libraries effectively abstracts your code away from external libraries. Strategically, wrappers allow you to avoid having to refactor your code extensively when external libraries change their application programmer interface. You only need to update your wrapper library. This minimizes refactoring hell.*

Source: page 290, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Design patterns categories

- **Creational patterns:** “These patterns deal with ways to create objects.”
- **Structural patterns:** “Simplify the relationships among objects.”
- **Behavioral patterns:** “Deal with communication among objects.”

Source: pages 291-293, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Creational patterns

- **Factory method:** “Creates a new instance of a class.”
- **Lazy initialization:** “A technique for delaying object creation until the object is needed, usually to prevent slow startup.”
- **Object pool:** “Creates a pool of objects that can be recycled and reused.”
- **Prototype:** “Uses a prototype instance of an object with default properties filled in.”
- **Singleton:** “Ensures that there is only one instance of an ojvect and provdes a method for accessing that object.”

Source: page 291, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Structural patterns

- **Adapter, Wrapper, or Translater:** “Provides a simplified or otherwise transformed interface to a class. An adapter allows incompatible classes to communicate.”
- **Composite:** “Composes objects into tree- or network-like structure so you can represent objects and groups of objects uniformly.”
- **Decorator:** “Attaches additional features to an object.”
- **Facade:** “Provides an interface to a (usually complicated subsystem.”
- **Flyweight:** “Used to allow objects to share common data so they can remain relatively small.”
- **Module:** “Groups related classes, methods, or other objects into a single entity.”

Source: page 291, *Beginning Software Engineering* (2nd edition) by Rod Stephens

# Bahavioral patterns

- **Blackboard:** “Combines data from various data sources, possibly translating into a common format.”
- **Chain of responsibility:** “A chain of objects that handles a request by passing the request along until an object handles it.”
- **Command:** “Encapsulates a command [for] an object to take a command as a parameter and then execute it.”
- **Iterator:** “Provides a way to access the elements inside some sort of collection class without exposing the collection’s underlying details.”
- **Mediator:** “An object through which other objects can interact.”
- **Momento:** “Captures an object’s internal state so that object (or another object) can be placed in that state later.”
- **Model-view-controller (MVC):** “This pattern defines three kinds of classes that work together, usually to provide a user interface.”
- **Model-view-presenter (MVP):** “This is a refinement of the MVC pattern.”
- **Model-view-view/model (MVVM):** “ The MVVM pattern is somewhat similar to MVP.”
- **Null object:** “A default object that can be used in place of a null reference.”
- **Observer or publish/subscribe:** “An object that should receive notification when another object’s state has changed in some particular way.”
- **Helper or servant:** “A package of helper methods that can be used by multiple classes.”
- **State:** “An internal representation of an object’s state so it can act differently when its state changes.”
- **Strategy:** “An encapsulated algorithm so you can use different algorithms interchangeably.”
- **Template:** “An outline of an algorithm that allows subclasses to redefine selected steps.”
- **Visitor:** “Represents an operation on a data structure so you can define new behaviors without modifying the data structure.”

Source: pages 292-293, *Beginning Software Engineering* (2nd edition) by Rod Stephens