GOF Design Patterns

WHAT IS A PATTERN?

- A pattern is a solution to a reoccurring problem in a context
  - Context - Situation in which the pattern applies
  - Problem - Goal in the given context referencing the constraints imposed by the context
  - Solution - Describes a general way to achieve the goal and the set of constraints

BENEFITS OF DESIGN PATTERNS

- Reuse solutions, not code
  - Benefit by learning from the experience of others.
  - You do not have to reinvent solutions for commonly recurring problems
- Establish Common Terminology
  - Communication and teamwork require a common base of vocabulary and a common viewpoint of the problem.
  - Design patterns provide a common point of reference during the analysis and design phase of a project.
- Higher Level perspective
  - Focus on the problem and on the process of design and object orientation. This frees you from the tyranny of dealing with the details too early.

OBJECT-ORIENTED DESIGN PATTERNS PRINCIPLES

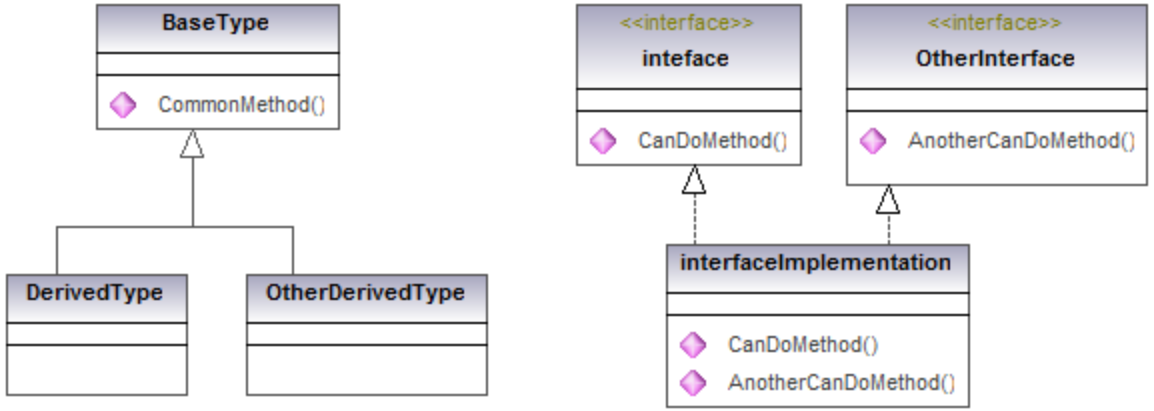
- Design to interfaces
  - Write interface classes and then produce implementation classes
  - Abstract classes satisfy this requirement
- Encapsulate what varies
  - Separate code that stays the same to code that varies
- Types should be closed for modification but open for extension
- Maximize re-use
  - Designing code with change in mind can lead to quicker releases and fewer bugs

INHERITANCE VS COMPOSITION

- Inheritance was seen as a powerful tool in early days of OO
- Over time it has been found to be far less useful, systems built using large inheritance hierarchies have proven to be fragile.
- A single change in a common base class can have far reaching effects.
- Composition can deliver many of the benefits of inheritance such as code re-use.

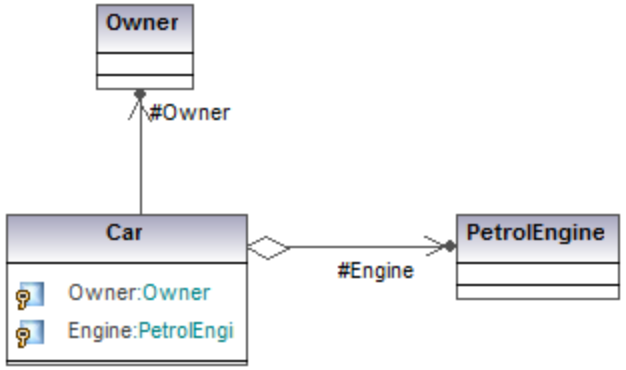
CLASS DIAGRAMS, RELATIONSHIPS

Inheritance: Is-kind-of
Interface: Can Do



Association: one object holds a reference to another

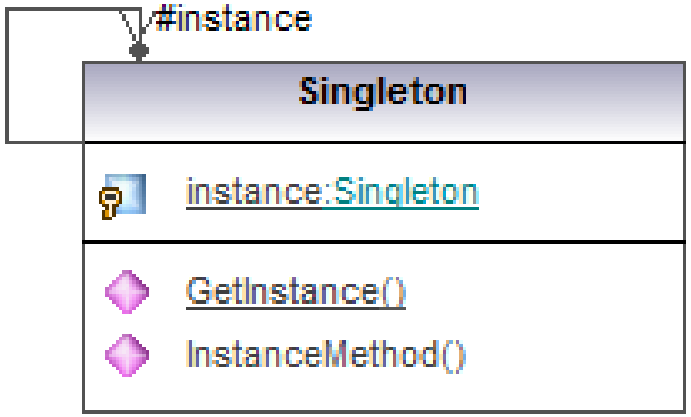
Composition: the item being reference is actually residing in the parent object



SINGLETON PATTERN

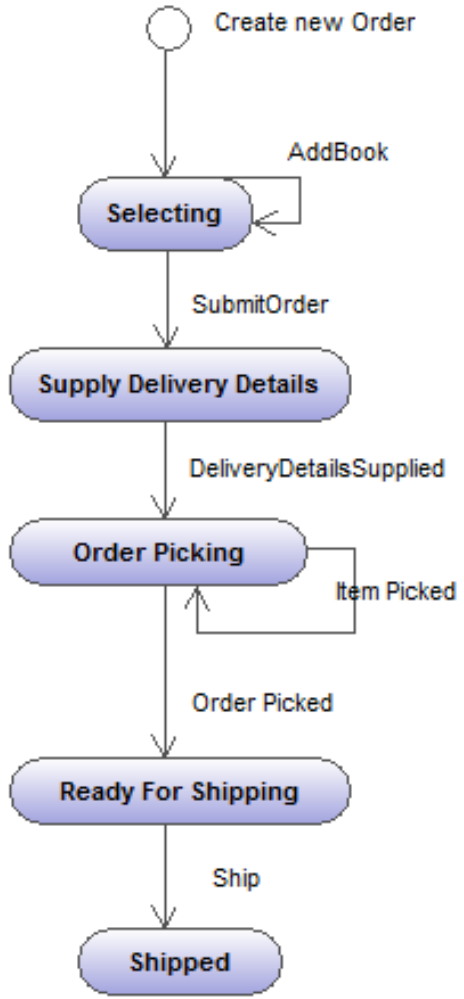
Objects that have a single instance.

- lobal point of access to the object
- Lazy initialization
- Thread safe



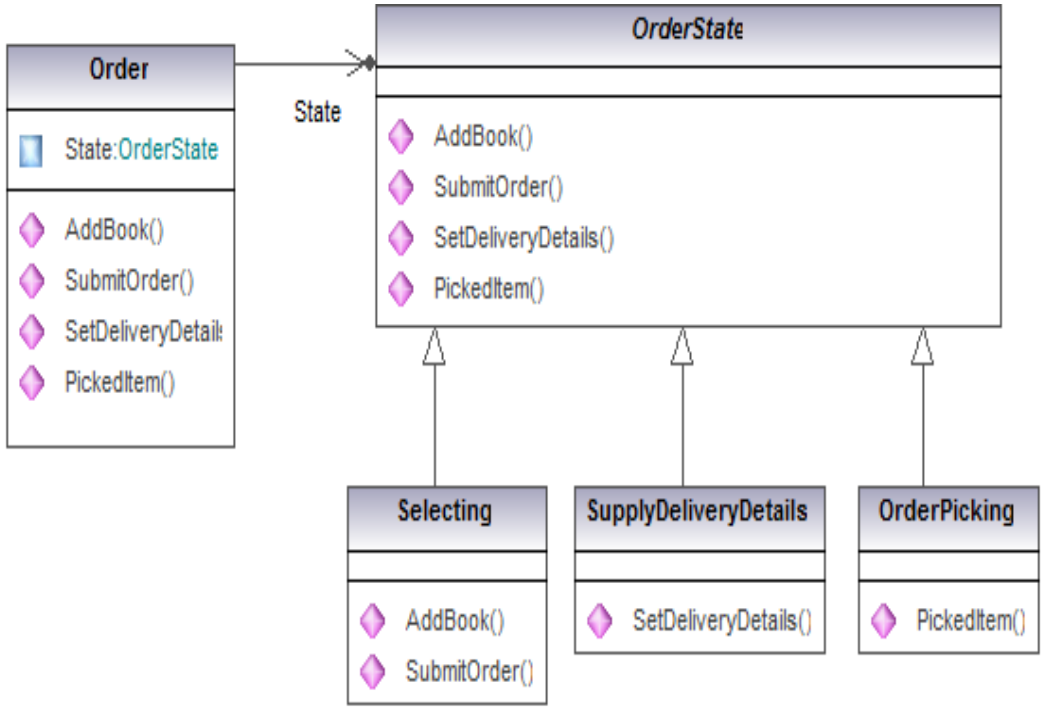
STATE MACHINE

- Objects can behave differently over time. The State Machine pattern satisfies the need for state awareness.
- An Order type is created to represent an order as it passes through the business process. The ordering process goes through a series of steps triggered by events.
  - Select items
  - Set delivery details
  - Items are selected
  - Order is shipped
- The Order type is responsible for ensuring the business process is followed.



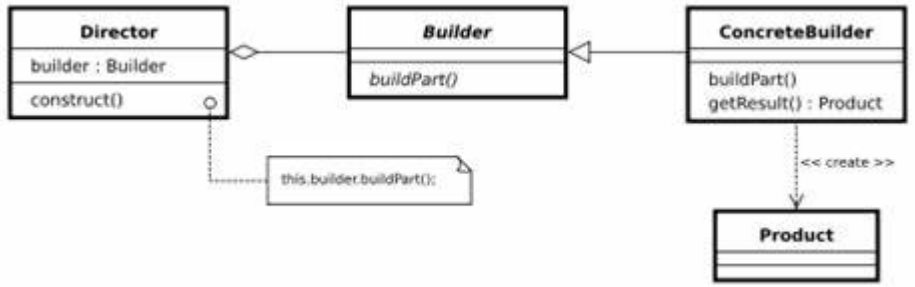
STATE PATTERN

- Localize the behavior of each state so that changes to one state don't effect another.
- Implement each state as its own nested class.
- Have the Order object delegate behavior to the current state object.



BUILDER PATTERN

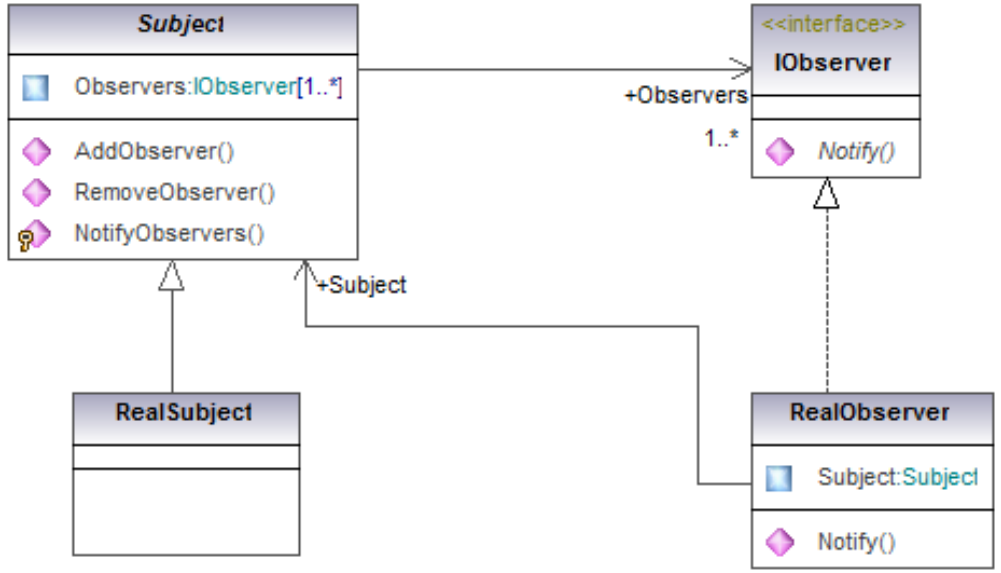
- Builders allow the construction of a complex object over a series of steps
  - Builder hides the complexity of the construction
  - Complex object representation is independent of method of construction
- Construction methods typically return self to allow natural method chaining
- Build method returns complex object



OBSERVER PATTERN

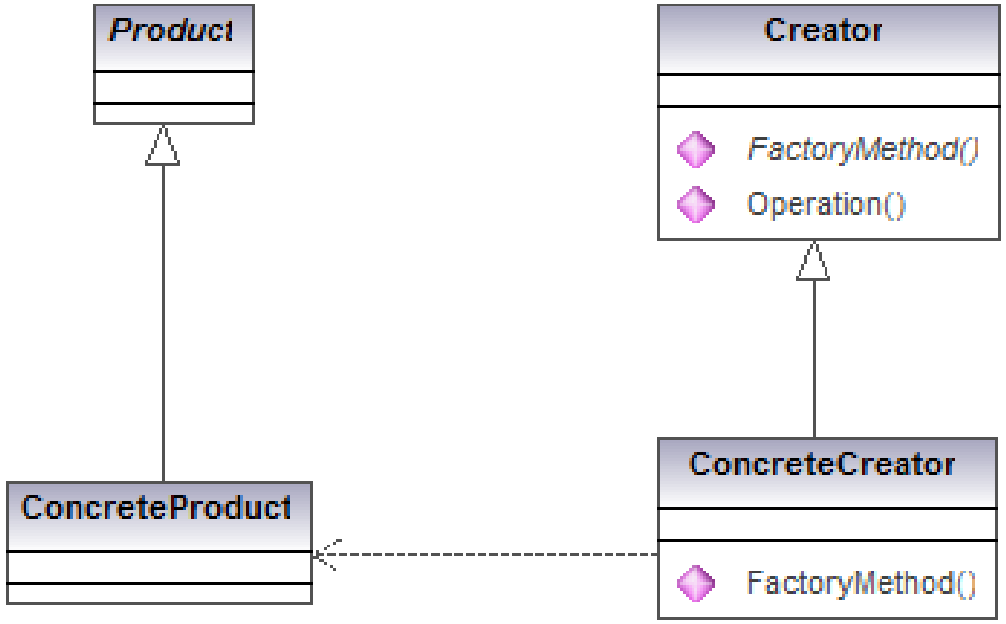
Patterns for creating events, event notification, and event handling.

- Observer is the event client, which implements a handler.
- Subject is the object or target. This is the event server responsible for notifying the client objects. In addition, there could be no observers.



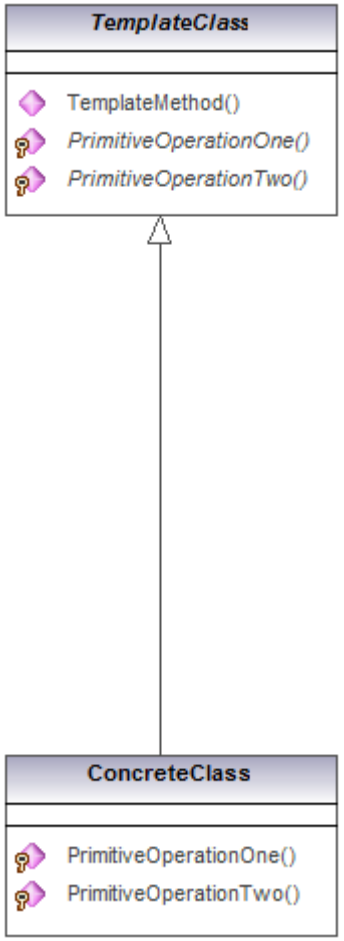
FACTORY METHOD PATTERN

- Define an interface for creating an object, but let subclasses decide which class to instantiate.
- Factory Method lets a class defer instantiation to subclasses



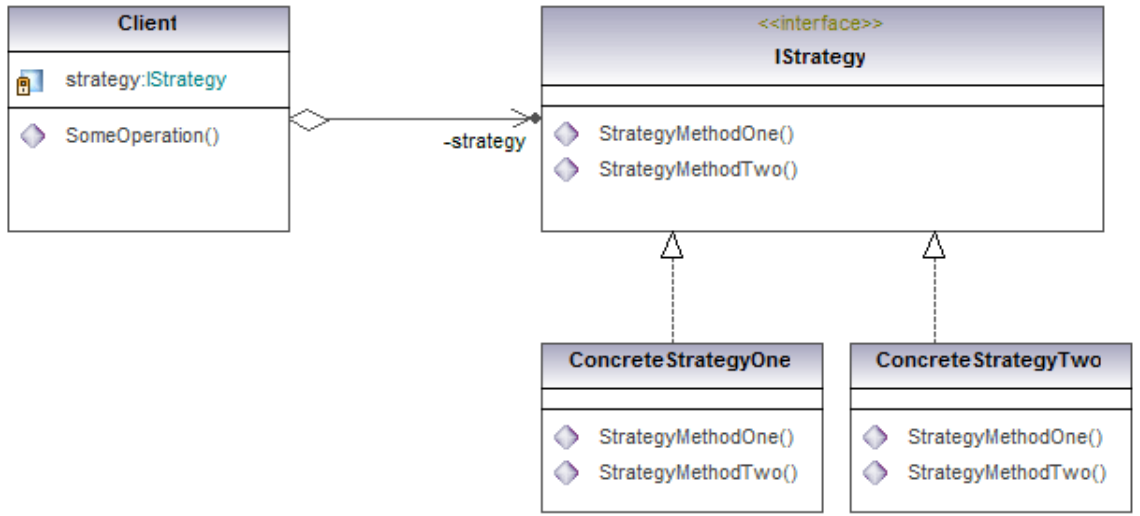
TEMPLATE METHOD PATTERN

- Template method defines the skeleton of an algorithm in an operation, deferring some steps to subclasses.
- It lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



STRATEGY PATTERN

- Template method used abstract methods and inheritance to call different behavior
- Alternatively supply implementation behavior at run time using composition



OTHER USEFUL PATTERNS

- There are many other widely used GOF patterns
  - Command - a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time.
  - Façade - hides the complexities of the system and provides an interface to the client using which the client can access the system.
  - Adapter - a structural design pattern that allows objects with incompatible interfaces to collaborate.
  - Decorator - a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.
  - Proxy - a structural design pattern that lets you provide a substitute or placeholder for another object.
  - Many more…
    - Design-Patterns-GoF