# Overview:

* These design patterns are specifically concerned with communication between objects:
  + Characterize complex control flow that is difficult to follow at run-time.
  + Shift the focus away from flow of control to let you concentrate just on the way objects are interconnected.
* These patterns increase flexibility in carrying out this communication.
* Provide solutions on how to segregate objects to be both dependent and independent.
* Concerned with algorithms and the assignment of responsibilities between objects.

# Class Patterns vs Object Patterns (sub-category):

* Behavioral class patterns use inheritance to describe algorithms and flow of control.
  + The template method is an abstract definition of an algorithm.
    - Defines an algorithm step by step.
    - A subclass fleshes out the algorithm by defining the abstract operations.
* Behavioral object patterns describe how a group of objects cooperates to perform a task that no single object can carry out alone.
  + Uses object composition rather than inheritance.
  + The mediator pattern uses a mediator object for peer object communication.
    - Mediator provides the indirection needed for loose coupling.

# Summary:

# Encapsulation Variation:

* Encapsulating variation is a theme of many behavioral patterns.
* When an aspect of a program changes frequently, these patterns define an object that encapsulates that aspect:
  + Then other parts of the program can collaborate with the object whenever they depend on that aspect.
* This theme runs through other kinds of patterns.
* Abstract Factory, Builder, and Prototype all encapsulate knowledge about how objects are created.
* Decorator encapsulates responsibility that can be added to an object.
* Bridge separates an abstraction from its implementation, letting them vary independently.
* A Strategy object encapsulates an algorithm.
* A State object encapsulates a state-dependent behavior.
* A Mediator object encapsulates the protocol between objects.
* An Iterator object encapsulates the way you access and traverse the components of an aggregate object.
* Not all object behavioral patterns work like this:
  + Chain of Responsibility deals with an arbitrary number of objects, all of which may already exist in the system.

# Objects as Arguments:

* A several patterns introduce an object that is always used as an argument.
* A visitor object is the argument to a polymorphic accept operation on the objects it visits
  + Visitor is never considered a part of those objects.
* Other patterns define objects that act as magic tokens to be passed around and invoked at a later time
  + In the command pattern, the token represents a request
  + In the Memento, it represents the internal state of an object at a particular time.
  + In both cases, the token can have a complex internal representation, but the client is never aware of it.
* Polymorphism is important in the Command pattern:
  + Executing the Command object is a polymorphic operation.
* The Memento interface is so narrow that a memento can only be passed as a value.
  + It is likely to present no polymorphic operations at all to its clients.

# Mediator vs Observer:

* Mediator and Observer are competing patterns:
  + Observer distributes communication by introducing Observer and Subject objects.
  + A Mediator object encapsulates the communication between other objects.
* In the Observer pattern, there is no single object that encapsulates a constraint
  + The Observer and the Subject must cooperate to maintain the constraint
  + A single subject usually as many observers
    - Sometimes the observer of one subject is a subject of another observer.
* The Mediator pattern centralizes rather than distributes:
  + It places the responsibility for maintaining a constraint squarely in the mediator.
* Easier to make reusable Observers and Subjects than to make reusable Mediators.
* On the other hand, it is easier to understand the flow of communication in Mediator that in Observer.
  + Observers and subjects are usually connected shortly after they have been created.
    - Hard to see how they are connected later in the program.
    - The indirection that Observer introduces will still make a system harder to understand.

# Decoupling Senders and Receivers:

* When collaborating objects refer to each other directly, they become dependent on each other.
  + Can have an adverse impact on the layering and reusability of a system.
* Command, Observer, Mediator, and Chain of Responsibility address how you can decouple senders and receivers, but with different trade-offs.
* For example, the Command pattern supports decoupling by using a Command object to define the binding between a sender and receiver.

# Behavioral Patterns working together:

* Behavioral design patterns complement and reinforce each other
* A class in Chain of Responsibility will probably include at least one application of Template Method.
  + The Template method can use primitive operations to determine whether the object should handle the request and to choose the object to forward to.
* A class in the Chain of Responsibility can use the Command pattern to represent requests as objects.
* An interpreter can use the State pattern to define parsing contexts.
* An Iterator can traverse an aggregate, and a visitor can apply an operation to each element in the aggregate.
* A system that uses the Composite pattern might use a visitor to perform operations on components of the composition.
* A Composite could use the Chain of Responsibility to let components access global properties through their parent
  + A Composite could also use Decorator to override these properties on parts of the composition.
* A Composite could use the Observer pattern to tie one object structure to another and the State pattern to let a component change its behavior as its state changes
* The composition itself might be created using the approach in Builder.
* Well-designed object-oriented systems have multiple patterns embedded in them.