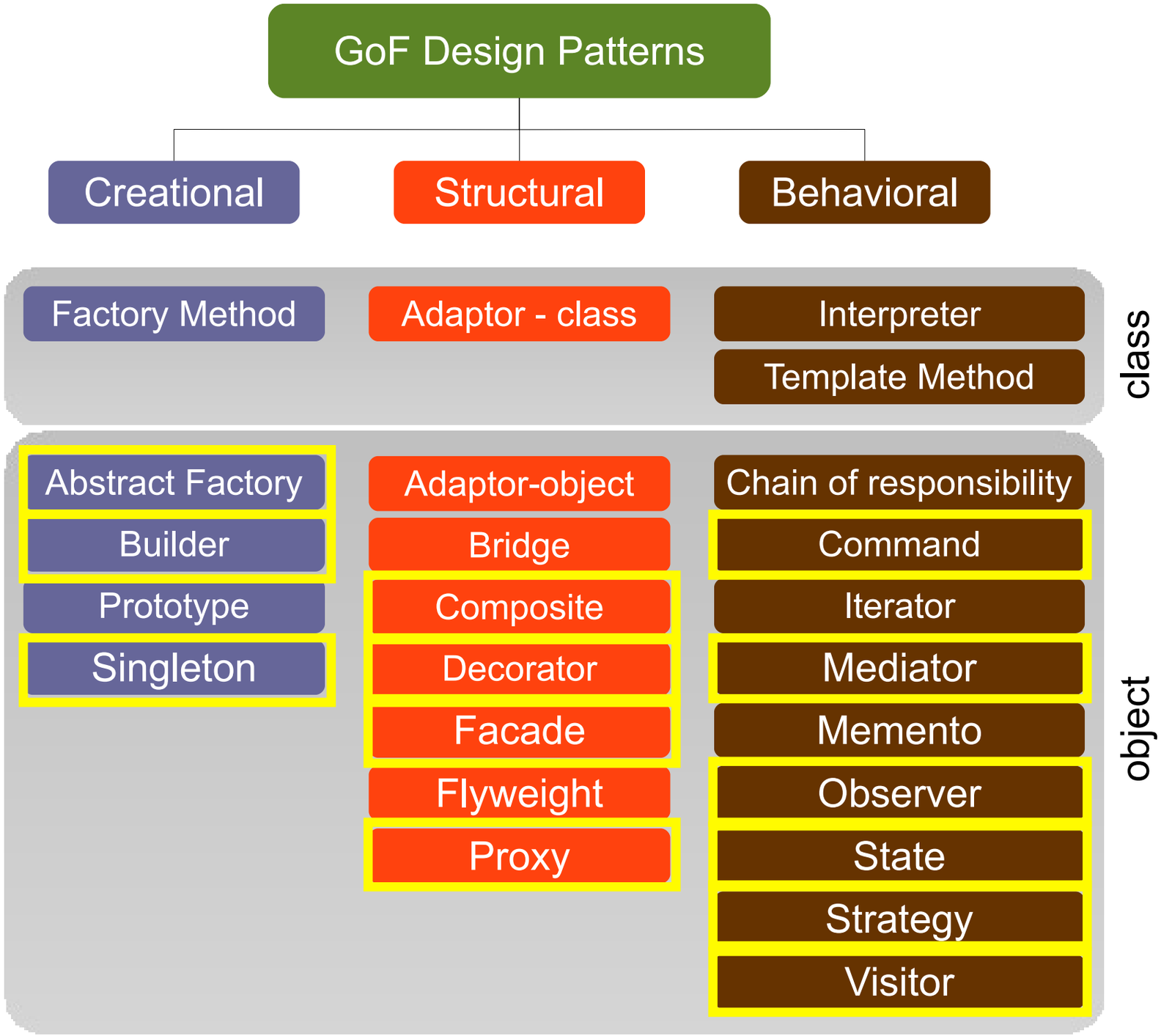
Material and some slide content from:
- GoF Design Patterns Book

Design Patterns #2

Reid Holmes

Lecture 12 - Thursday October 21 2010.

GoF design patterns



Composite

- Intent: “Compose objects into whole-part hierarchies, allowing ?
- Motivation: For situations where complex artifacts can be composed from multiple elements.
- Applicability:
  - For representing whole-part relationships.
  - For situations when you want your clients to ?

Composite

- Structure:

- Participants:
  - Component (lowest common denominator)
  - Leaf
  - Composite
  - Client

Composite

- Collaborations
  - Clients interact with component interface; composite objects forward requests to leaves.
- Consequences:
  - Allows applications to ?
  - Easy to add new components.
  - Negative: can make your app overly general. Hard to restrict how leaves can be composed.

Composite

- Implementation:
  1. Explicit parent references can be helpful
  2. Maximize the component interface
  3. Child ordering may matter
  4. Caching for performance

- Related to:
  - Often used with Decorator and Visitor patterns.

Decorator

- Intent: “Dynamically add additional responsibilities to structures.”
- Motivation: Sometimes we want to add new responsibilities to individual objects, not the whole class. Can enclose existing objects with another object.
- Applicability:
  - Add responsibilities dynamically and transparently.
  - Remove responsibilities dynamically.
  - ?

Decorator

- Structure

- Participants:
  - Component / concrete component
  - Decorator / concrete decorator

Decorator (code ex)

Decorator

- Collaborations
  - Decorators forward requests to component object.
- Consequences:
  - More flexible.
  - Avoids ?
  - Warn: Decorator & component are not identical.
    - ?
  - Negative: Many of little objects.

Decorator

- Implementation:
  - 1) Interface conformance.
  - 2) Abstract decorator not needed if only one decoration is required.
  - 2) Keep component classes lightweight.
  - 3) Changing a skin instead of changing the guts.
- Related to: Decorators are a kind of single-node Composite. Decorators can change the skin, Strategy pattern can change the guts.

Visitor

- Intent: “Represent operations to be performed on classes of elements.”
- Motivation: ?
- Applicability:
  - When you have a large object structure you want to traverse.
  - ?
  - The objects rarely change but the operations may.

Visitor

- Structure

- Participants:
  - Visitor / ConcreteVisitor
  - Element / ConcreteElement
  - ObjectStructure

Visitor

- Collaborations:
  - Client creates the ConcreteVisitor that traverses the object structure.
  - The visited object calls its corresponding visitor method on the ConcreteVisitor.
- Consequences:
  - Adding new operations is easy.
  - Visitor gathers related operations
  - Adding ConcreteElement classes is hard.
  - Accumulating state.
  - Negative: ?

Visitor

- Implementation:
  - 1) Double dispatch.
  - 2) Who traverses structure?
- Related to: Good ad visiting Composite structures.

Command

- Intent: “Encapsulate requests enabling clients to log / undo them as required.”
- Motivation: In situations where you need to be able to make requests to objects without knowing anything about the request itself or the receiver of the request, the command pattern enables you to pass requests as objects.
- Applicability:
  - ?
  - ?
  - Support undo.
  - Model high-level operations on primitive operations.

Command

- Structure

- Participants:
  - Command / ConcreteCommand
  - Client
  - Invoker
  - Receiver

Command

- Collaborations:
  - Client creates ConcreteCommand and specifies receiver.
  - Invoker stores ConcreteCommand object.
  - Invoker requests execute on Command; stores state for undoing prior to execute (if undoable).
  - Concrete invokes operations on its receiver to perform request.
- Consequences:
  - ?
  - Commands are first-class objects.
  - Commands can be assembled into composite.
  - ?

Command

- Implementation:
  - 1) How smart should a command be?
  - 2) Support undo/redo.
  - 3) Avoiding error accumulation in the undo process.
- Related to: Composite commands can be created; the Memento pattern can store undo state. Commands often use Prototype when they need to be stored for undo/redo.

iRoadTrip demo

- Functionality being demonstrated:
  - Creating a trip.
  - Joining a trip.
  - Adding new users to a trip dynamically.
  - Updating positions dynamically.
- Not implemented:
  - Statistics / querying.
  - Interaction.
  - Leaving a trip / managing trips.