Design Patterns

An Introduction

Design Patterns

- A description of communicating objects and classes that are customized to solve a general design problem in a particular context
- Applies to a specific problem we face
- Most design patterns help us to adhere to the well-known design principles

Design Patterns

A solution to an abstract software problem within a particular context

- *Abstract* - Not specific to a language
- *Context* - Common situations in code design
- *Problem* - Goals vs. constraints in some context
- *Solution* - Design for cooperating classes/object which resolve the goals and constraints

Discovering Design Patterns

1. Solve a software problem
2. Abstract and generalize the solution into patterns
3. Record the patterns for future use

Applying Design Patterns

1. Search for patterns applicable to the problem at hand
2. Understand the patterns and how to use them to solve the problem
3. Adapt and implement the patterns to solve the problem

Components of Design Patterns

Four main components of a design pattern:

1. **Pattern name**
   - Brief descriptive name that summarizes the pattern

2. **Problem**
   - Describes when to apply the pattern
   - Often includes conditions that should be met before applying the pattern

Components of Design Patterns

3. Solution

- The classes/objects that make up the designs, along with their relationships, responsibilities, and collaborations
- Doesn't describe a particular concrete implementation → a pattern is a template
- Provides an abstract description of a design problem and how a particular arrangement of classes and objects solves it

Components of Design Patterns

4. Consequences

- Results and trade-offs of applying the pattern
- Help us to evaluate the costs and benefits of applying the pattern
- Often concern time/space trade-offs
- May address language/implementation issues
- The term *consequences* often has a negative connotation – this is not the case here

Design Patterns in Detail

- Pattern name and classification
- Intent (purpose, rationale, what it does, etc.)
- Also known as (alias name, if any)
- Motivation (scenario illustrating the problem solved by the pattern)
- Applicability (situations where it is useful, etc.)
- Structure (graphical representation)
- Participants (elements, both classes and objects, involved, etc.)

Design Patterns in Detail

- Collaborations (how the participants collaborate to do the required job)
- Consequences (tradeoffs and results of using the pattern)
- Implementation (pitfalls, hints, or techniques useful for implementation)
- Sample code (code fragments to aid in implementation)
- Known uses (examples in real systems)
- Related patterns (other patterns related to this, similarities/differences with other patterns, other patterns used in conjunction with this one)

Types of Design Patterns

- Three main types:
  - Creational: primarily deal with object creation
  - Behavioural: primarily deal with the ways in which classes or objects interact and distribute responsibility
  - Structural: deal with composition of classes or objects

- We will spend some time exploring each type shortly

Gang of Four Design Patterns

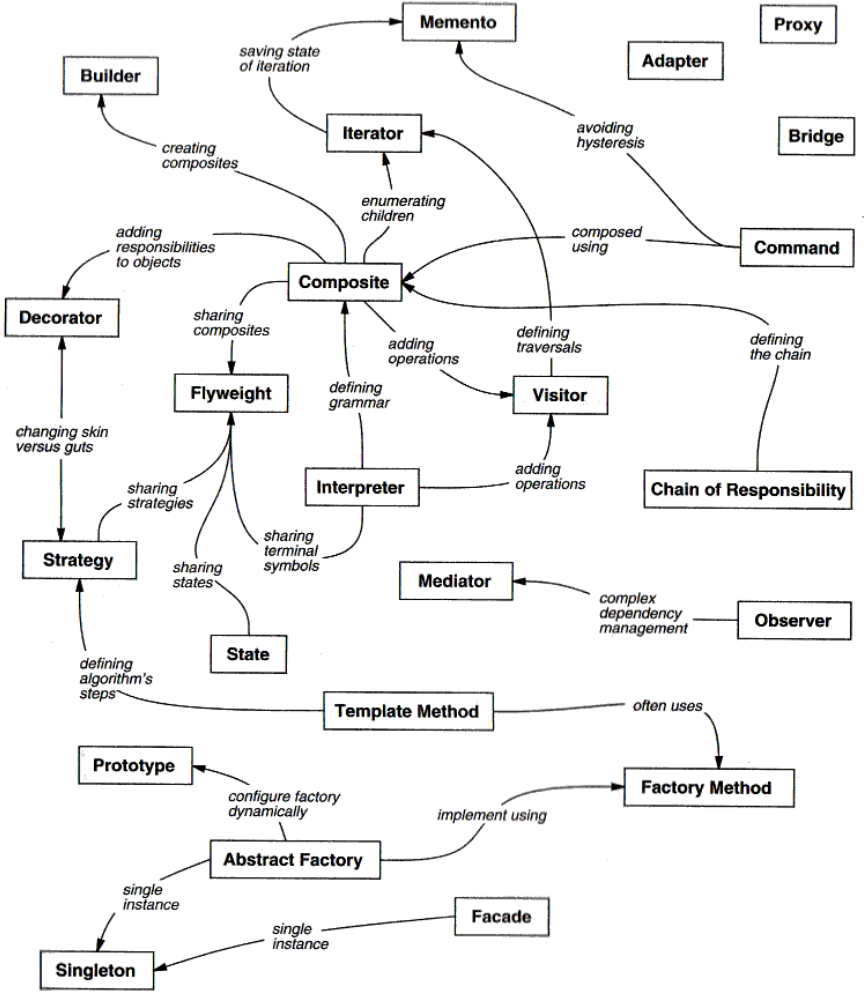
- The "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) classified a collection of design patterns into creational, behavioural, and structural categories

- Their book on the subject, simply titled "Design Patterns", is the canonical reference for design patterns

Gang of Four Design Patterns

Creational	Structural	Behavioral
Factory Method Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Flyweight Façade Proxy	Interpreter Template Method Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Gang of Four
Design Pattern Relationships



Benefits of Design Patterns

- Patterns encapsulate a particular design to solve a specific problem
  - Helps create a knowledge base of patterns
- Reuse previously demonstrated solution without reinventing the wheel (possibly more costly and error-prone)
  - Faster development, recognized quality, lower cost, more reliable time/cost estimates, and so on
- Focused/improved communication amongst the pattern stakeholders
  - Common terminology
  - Common point of reference during design and analysis

Benefits of Design Patterns

- Allows one to think in “design” terms and not have to deal with code early on
- Increased use of patterns may move a system towards a more modular design in general
- Such a modular design may aid ease of change (maintenance) and evolvability of systems
- Encourages shared learning among developers and developing a mindset for abstract solutions to common problems

Drawbacks of Design Patterns

- Perhaps creates a tendency to use a “standard” solution instead of creating an innovative solution to the problem at hand
- Risk of forcing patterns into situations they don’t fit
- Risk of adjusting problem to make patterns work, instead of solving the actual problem
- Relying heavily on patterns may cause novices to think abstractly at the expense of not knowing the detailed workings

A Sample Pattern: Model-View-Controller

- A behavioural design pattern (although some consider this to be an architectural pattern … more on this later)
- Encapsulate components of interactive applications

Model	The data classes
View	The user interface components
Controller	Application logic / functionality

Model

- Class(es) which capture data, which can be:
  - Persistent - stored between program executions
  - Dynamic - data required during execution

- Persistent data ↔ Dynamic data
- Model classes handle computation on the data

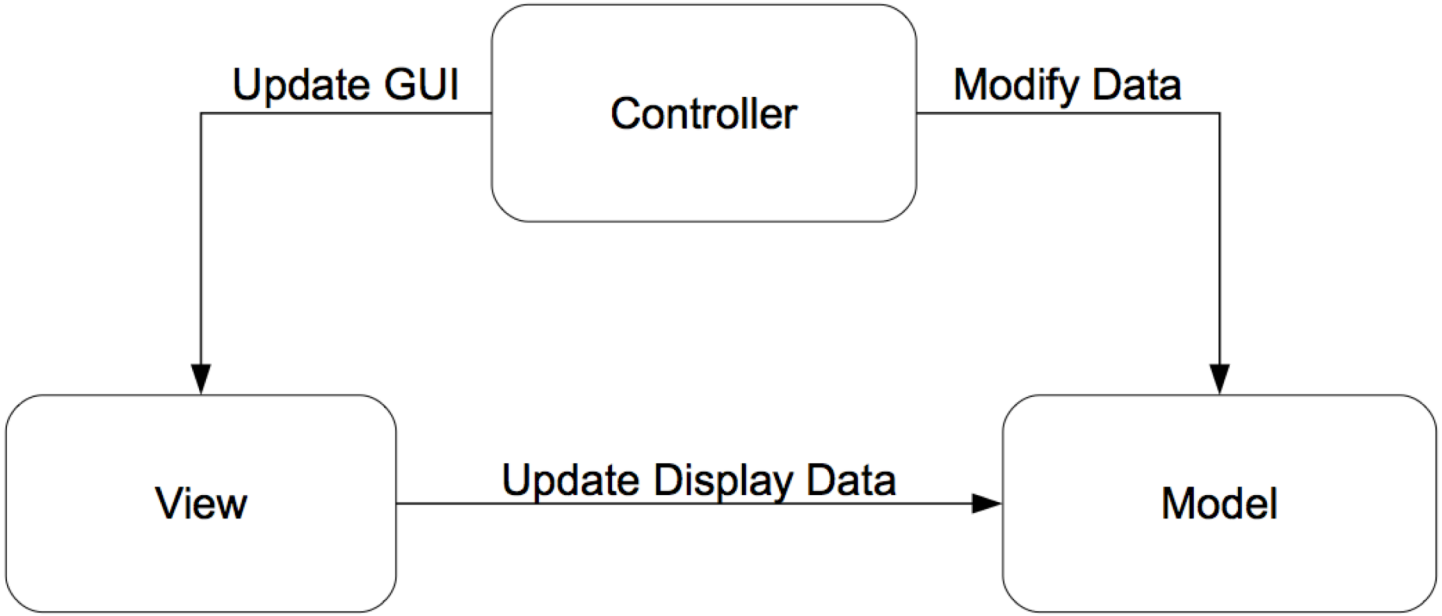
View

- Class(es) which present the interface to the user
- Typically GUI elements

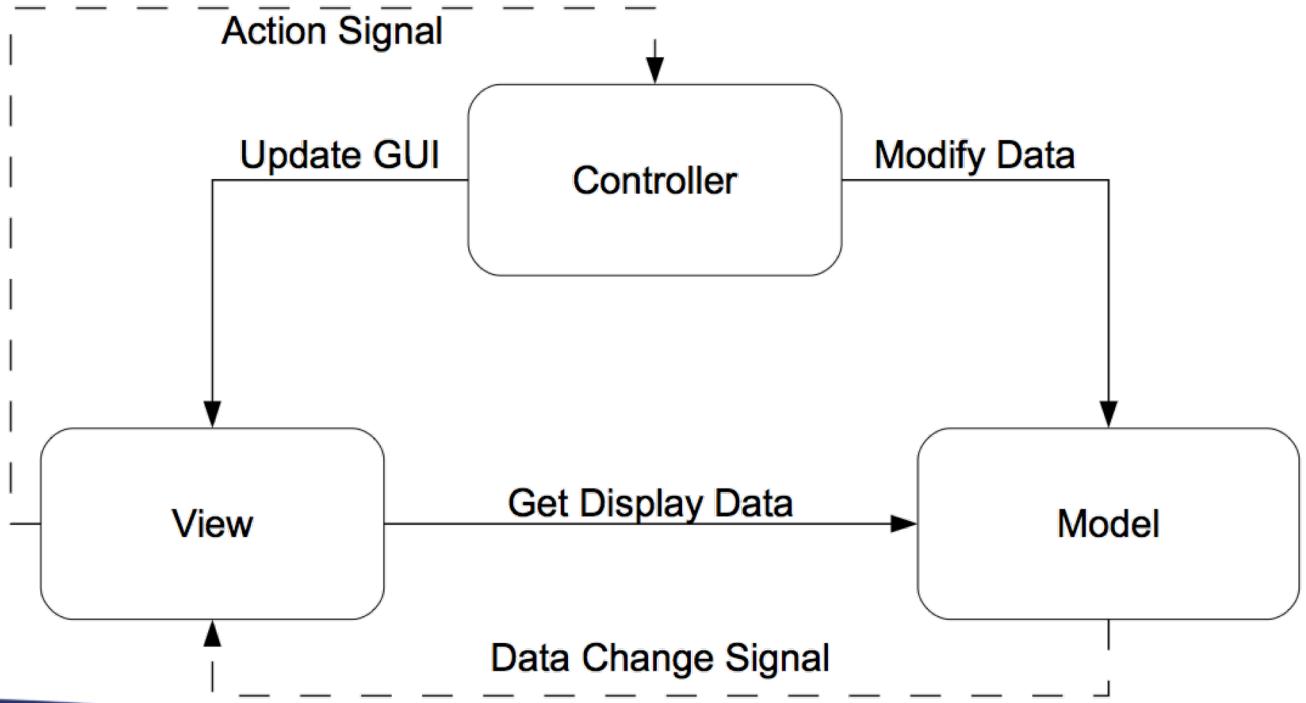
Controller

- Coordinates the Model and View classes
- May change the view
- Can modify and retrieve data from the model

MVC

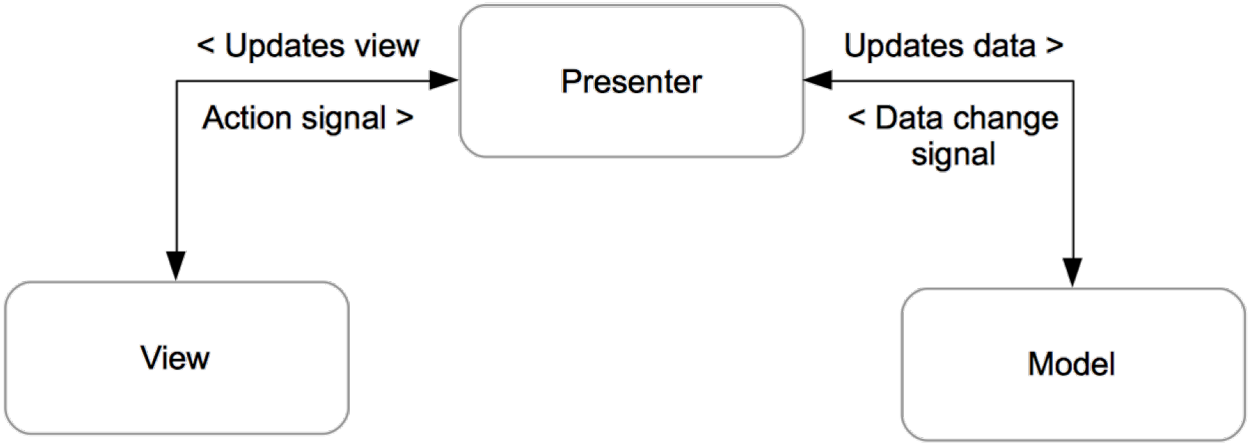

Update GUI
Controller
Modify Data
View
Update Display Data
Model

MVC


Action Signal
Update GUI
Controller
Modify Data
View
Get Display Data
Model
Data Change Signal

Model View Presenter

- Related pattern
- Further decouples view and model
- Often called MVP



Model View Controller

- Reduces coupling
- High cohesion within each element
- Widespread approach to GUI software design