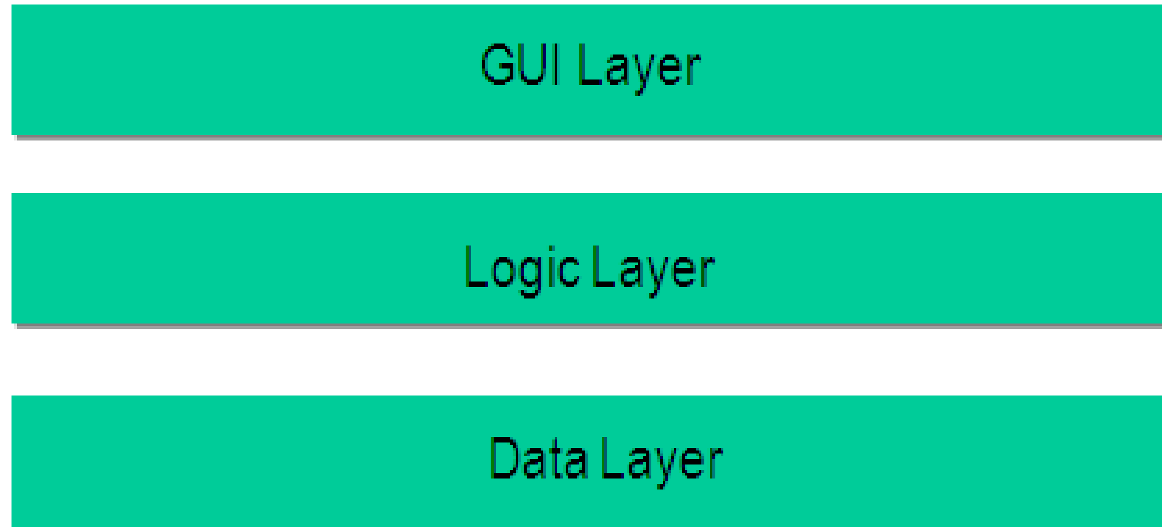
XIV. Design Patterns

Program Architecture – 3 tier Architecture



Design Patterns - Motivation & Concept

Christopher Alexander

A Pattern Language: Towns, Buildings, Construction

Motivation & Concept

OO methods emphasize design notations

Fine for specification, documentation

But OO is more than just drawing diagrams

Good draftsmen ▪ good designers

Good OO designers rely on lots of experience

At least as important as syntax

Most powerful reuse is *design* reuse

Match problem to design experience

Motivation & Concept (cont'd)

Recurring Design Structures

OO systems exhibit recurring structures that promote

- abstraction
- flexibility
- modularity
- elegance

Therein lies valuable design knowledge

Problem:

capturing, communicating, & applying this knowledge

Motivation & Concept (cont'd)

A Design Pattern…

- abstracts a recurring design structure
- comprises class and/or object
  - dependencies
  - structures
  - interactions
  - conventions
- names & specifies the design structure explicitly
- distills design experience

Motivation & Concept (cont'd)

Four Basic Parts

1. **Name**
2. **Problem (including "forces")**
3. **Solution**
4. **Consequences & trade-offs of application**

Language- & implementation-independent
A "micro-architecture"
Adjunct to existing methodologies

Part I: Motivation & Concept (cont'd)

Goals

Codify good design
- distill & generalize experience
- aid to novices & experts alike

Give design structures explicit names
- common vocabulary
- reduced complexity
- greater expressiveness

Capture & preserve design information
- articulate design decisions succinctly
- improve documentation

Facilitate restructuring/refactoring
- patterns are interrelated
- additional flexibility

Motivation & Concept (cont'd)

Design Space for GoF Patterns

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

***Scope**: domain over which a pattern applies*
***Purpose**: reflects what a pattern does*

Motivation & Concept (cont'd)

Design Pattern Template (1st half)

NAME scope purpose

Intent

short description of the pattern & its purpose

Also Known As

Any aliases this pattern is known by

Motivation

motivating scenario demonstrating pattern's use

Applicability

circumstances in which pattern applies

Structure

graphical representation of the pattern using modified UML notation

Participants

participating classes and/or objects & their responsibilities

Motivation & Concept (cont'd)
Design Pattern Template (2nd half)

...

Collaborations
how participants cooperate to carry out their responsibilities

Consequences
the results of application, benefits, liabilities

Implementation
pitfalls, hints, techniques, plus language-dependent issues

Sample Code
sample implementations in C++, Java, C#, Smalltalk, C, etc.

Known Uses
examples drawn from existing systems

Related Patterns
discussion of other patterns that relate to this one

Command Pattern: Motivation

- **Say you have a remote control with three buttons**
  - You would like to be able to walk around and press the buttons to turn on/off different devices
  - However, each device you want to control has a different interface for the power command
    - Ceiling Fan: OnOff();
    - Garage Door: OpenClose();
    - Television: TogglePower();

Command Pattern Motivation

- **Approach that works but very static:**

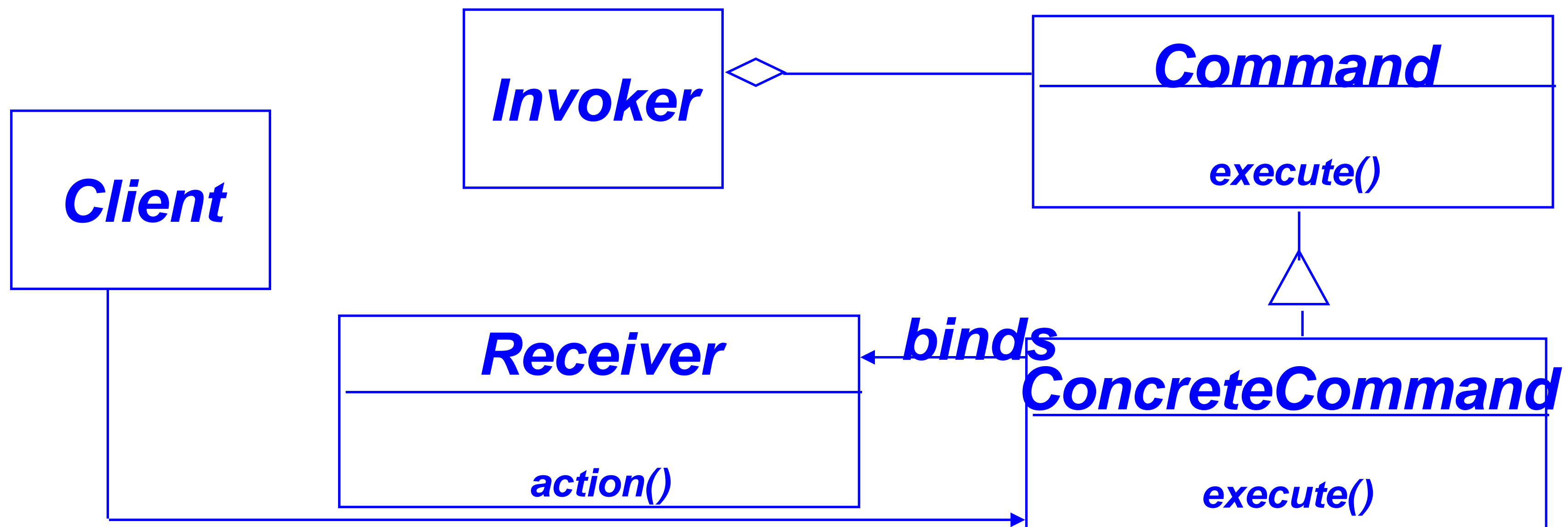
```
if (buttonPress == 0)
        TogglePower();   // TV
else if (buttonPress == 1)
        OpenClose();     // Garage
else if (buttonPress == 2)
        OnOff();         // Fan

Etc.
```

*More flexible and easier to use: Create an object, the **command object**, that encapsulates the desired request, and have the user invoke the request from the command object. In this case we may have 3 command objects in an array:*

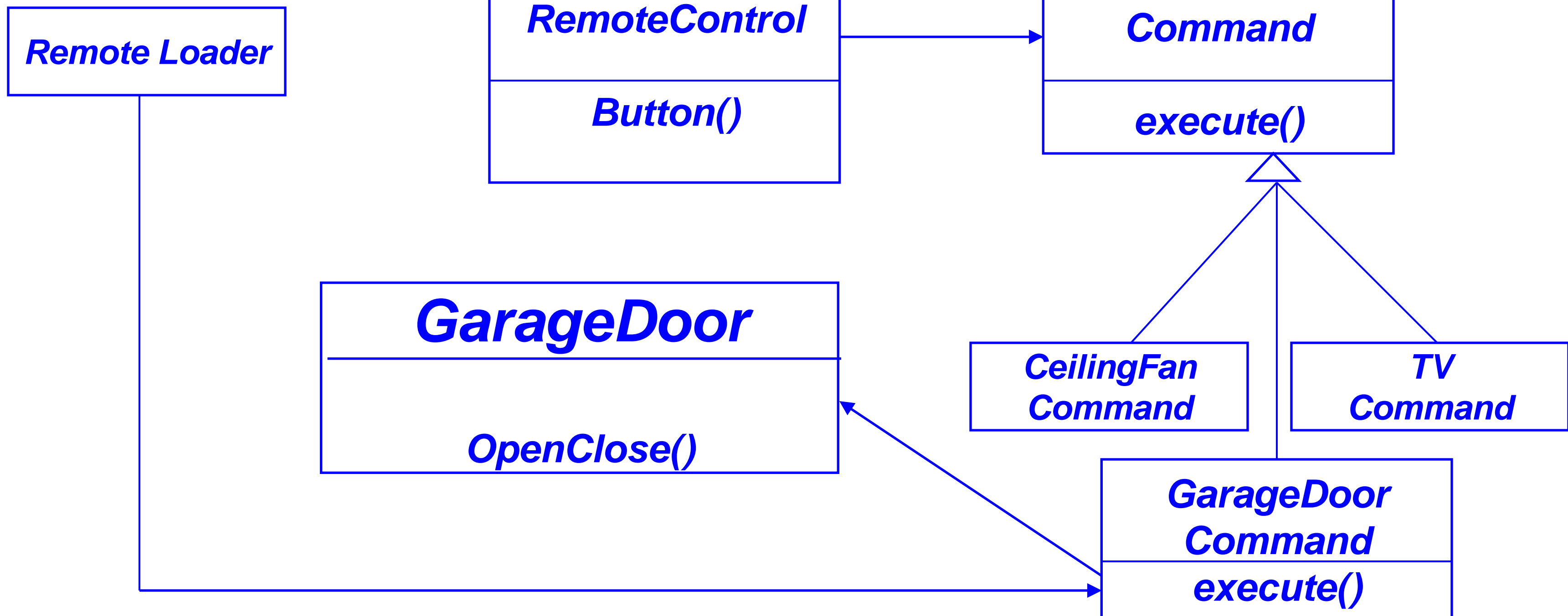
Button[buttonPress].execute();

Command pattern



- **Client creates a ConcreteCommand and binds it with a Receiver.**
- **Client hands the ConcreteCommand over to the Invoker which stores it.**
- **The Invoker has the responsibility to do the command (“execute” or “undo”).**

Command Pattern for Remote



execute() for each concrete command would use delegation to the corresponding device, e.g.
garagedoor.OpenClose()
or tv.TogglePower()

Command pattern Applicability

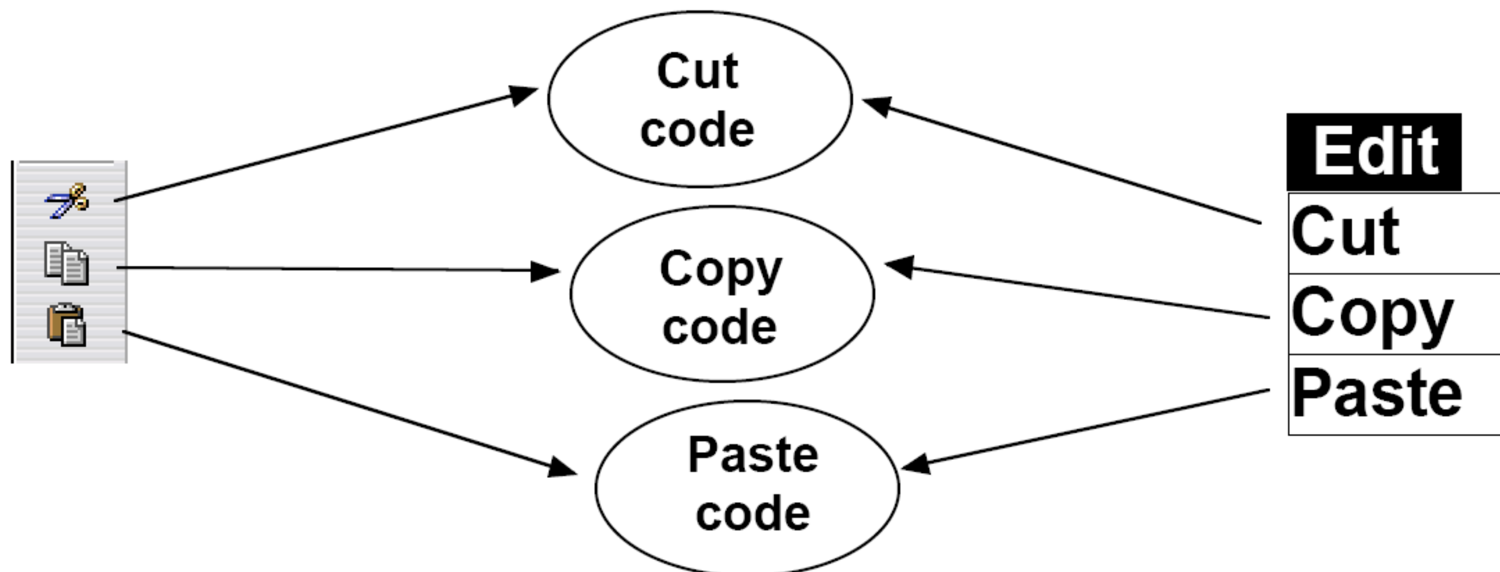
“Encapsulate a request as an object, thereby letting you

- parameterize clients with different requests,
- queue or log requests, and
- support undoable operations.”

- **Uses:**
  - Undo queues, can add now since each command is sent through a command object and we can create a history of commands within this object
  - Database transaction buffering
  - Structure the application around commands

Common UI commands

- **it is common in a GUI to have several ways to activate the same behavior**
  - example: toolbar "Cut" button and "Edit / Cut" menu
  - this is *good* ; it makes the program flexible for the user
  - we'd like to make sure the code implementing these common commands is not duplicated



Command pattern facilities understanding of applications code