



# **CSE 301**

# **Software Design and Architecture**

---

## **6. Introduction to design patterns**

---

**Dr. Salisu Garba**  
**[Salisu.garba@slu.edu.ng](mailto:Salisu.garba@slu.edu.ng)**

# Outline

---

- Introduction to design patterns
- Design pattern classifications
- Examples of Design Patterns



# Motivation

---

- Developing software is hard
- Designing reusable software is more challenging
  - finding good objects and abstractions
  - flexibility, modularity, elegance, reuse
  - takes time for them to emerge, trial and error
- Successful designs do exist
  - exhibit recurring class and object structures
- Other patterns:
  - Novels (tragic, romantic, crime),
  - Movies genres (drama, comedy, documentary)
  - Football (442, 433, 343, 4231)
  - Clothes (Suit, Bridesmaid, Gown)



# Design Patterns

---

- Design patterns were derived from ideas put forward by Christopher Alexander who suggested that
  - there were certain common patterns of building design that were inherently pleasing and effective.
- The pattern is a **description of the problem** and the **essence of its solution**, so that the solution may be **reused** in different settings.
  - The pattern is not a detailed specification.
  - “Patterns == problem/solution pairs in a context”
- Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience.
  - Patterns facilitate reuse of successful software architectures and designs.



# Mastering the patterns

## Becoming a Chess Master

- *First learn rules*
  - e.g., names of pieces, legal movements, chess board geometry and orientation, etc.
- *Then learn principles*
  - e.g, relative value of certain pieces, strategic value of center squares, power of a threat, etc.
- *To become a Master of chess, one must study the games of other masters*
  - These games contain patterns that must be understood, memorized, and applied repeatedly.
- There are hundreds of these patterns

## Becoming a Software Design Master

- *First learn rules*
  - e.g., algorithms, data structures, and languages of software.
- *Then learn principles*
  - e.g., structured programming, modular programming, object-oriented programming, etc.
- *To become a Master of SW design, one must study the designs of other masters*
  - These designs contain patterns that must be understood, memorized, and applied repeatedly.
- There are hundreds of these patterns



# Design Patterns: Applications

---

- Wide variety of application domains:
  - drawing editors, banking, CAD, CAE, cellular network management, telecomm switches, program visualization
- Wide variety of technical areas:
  - user interface, communications, persistent objects, O/S kernels, distributed systems
- Scope specifies whether the pattern applies primarily to classes or to objects.
  - Class patterns deal with relationships between classes and their subclasses.
  - These relationships are established through inheritance, so they are static—fixed at compile-time.
  - Object patterns deal with object relationships, which can be changed at run-time and are more dynamic.



# Design Patterns

---

- To use patterns in your design, you need to recognize that any design problem you are facing may have an associated pattern that can be applied.
- Examples of such problems, documented in the 'Gang of Four's original patterns book, include:
  - Tell several objects that the state of some other object has changed (Observer pattern).
  - Tidy up the interfaces to a number of related objects that have often been developed incrementally (Façade pattern).
  - Provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented (Iterator pattern).
  - Allow for the possibility of extending the functionality of an existing class at run-time (Decorator pattern).





# Why Study Patterns?

---

- Reuse tried, proven solutions
  - Provides a head start
  - Avoids gotchas later (unanticipated things)
  - No need to reinvent the wheel
- Establish common terminology
  - Design patterns provide a common point of reference
  - Easier to say, “We could use Strategy here.”
- Provide a higher level prospective
  - Frees us from dealing with the details too early
- Most design patterns make software more modifiable, less brittle
  - we are using time tested solutions
- Using design patterns makes software systems easier to change—more maintainable
- Helps increase the understanding of basic object-oriented design principles
  - encapsulation, inheritance, interfaces, polymorphism





# 4 Essential Elements of Pattern

- The four essential elements of design patterns were defined by the 'Gang of Four' or "GoF" (Gamma, Helm, Johnson, Vlissides) compile a catalog of design patterns
  1. **Pattern name:** A name that is a meaningful reference to the pattern.
  2. **Problem:** A description of the problem area that explains when the pattern may be applied.
  3. **Solution:** A solution description of the parts of the design solution, their relationships, and their responsibilities.
    - This is not a concrete design description. It is a template for a design solution that can be instantiated in different ways. This is often expressed graphically and shows the relationships between the objects and object classes in the solution.
  4. **Consequences:** A statement of the consequences is the results and trade-offs—of applying the pattern.
    - This can help designers understand whether or not a pattern can be used in a particular situation.



# Gof Design Patterns Classification

---

- Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, 1994
- Written by this "gang of four"
  - Dr. Erich Gamma, then Software Engineer, Taligent, Inc.
  - Dr. Richard Helm, then Senior Technology Consultant, DMR Group
  - Dr. Ralph Johnson, then and now at University of Illinois, Computer Science Department
  - Dr. John Vlissides, then a researcher at IBM
- This book defined 23 patterns in three categories
  - ***Creational patterns*** deal with the process of object creation
  - ***Structural patterns*** deal primarily with the static composition and structure of classes and objects
  - ***Behavioral patterns*** which deal primarily with dynamic interaction among classes and objects



# GoF Patterns

---

- *Creational Patterns*

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

- *Structural Patterns*

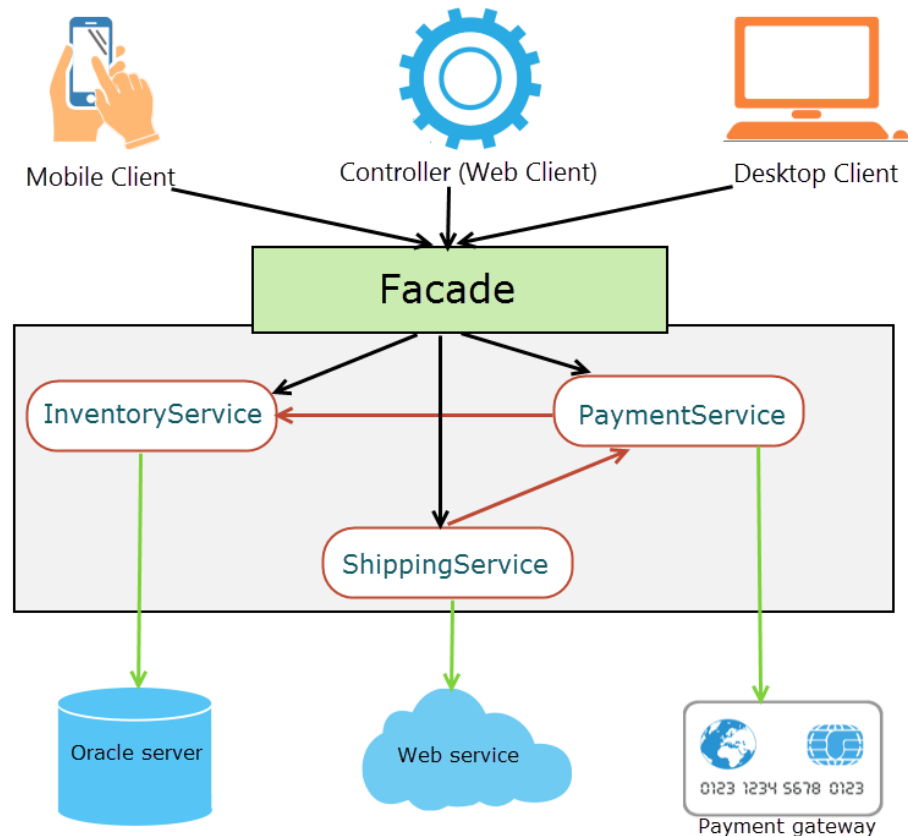
- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

- *Behavioral Patterns*

- Chain of Responsibility
- Command
- Interpreter
- **Iterator**
- Mediator
- Memento
- Observer
- State
- **Strategy**
- Template Method
- Visitor

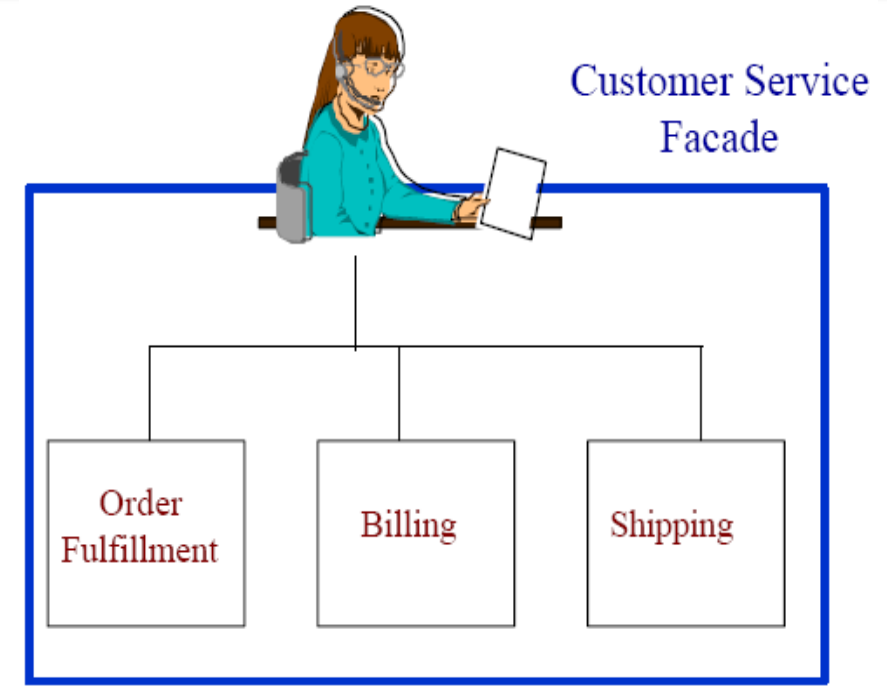
# Facade

- Provide a unified interface to a set of interfaces in a subsystem.



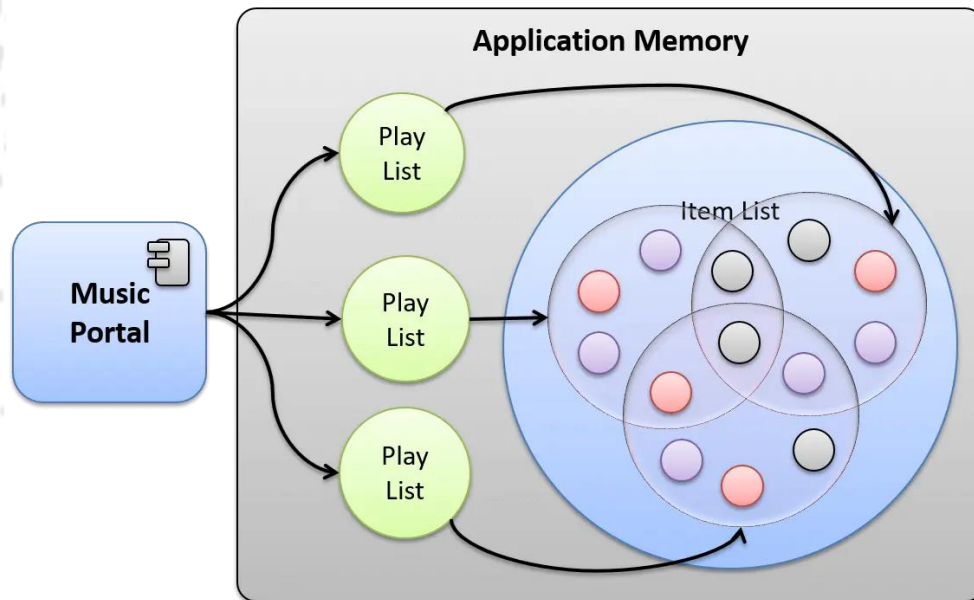
*Services implemented  
by subsystem classes*

*External Resources*



# Flyweight

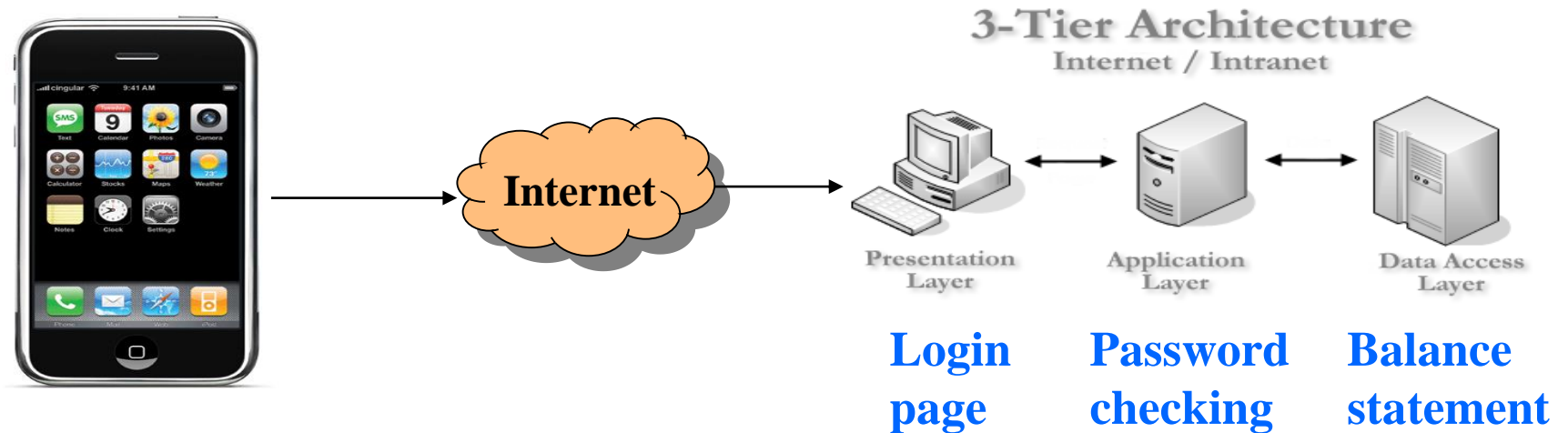
- Use sharing to support large numbers of fine-grained objects efficiently
- Help to reduce the memory usage when creating numerous objects by sharing their common states



By implementing the Flyweight design pattern, Some songs must be shared between various playlists, and some songs must have shared sections between them in order to save memory space.

# Chain of Responsibility

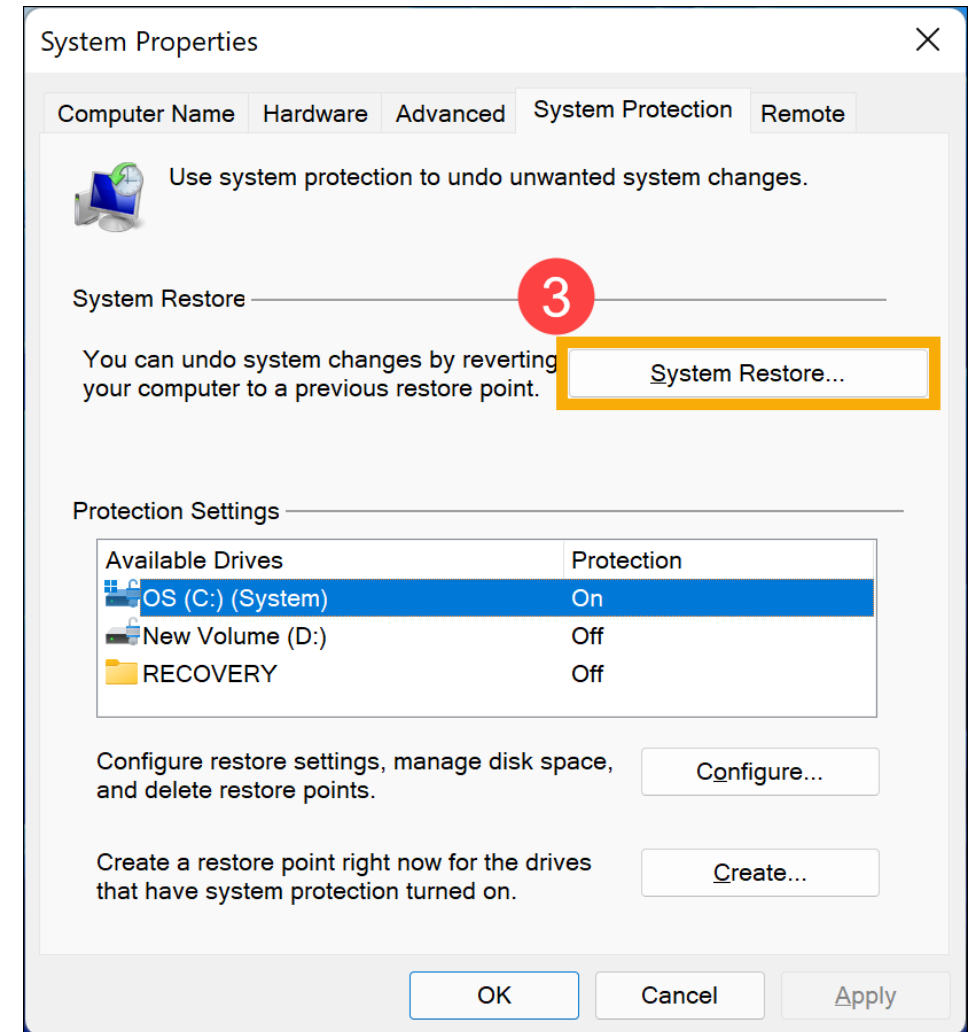
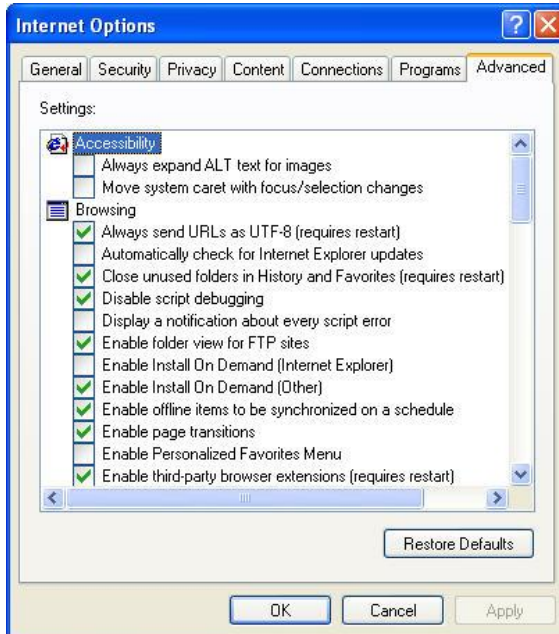
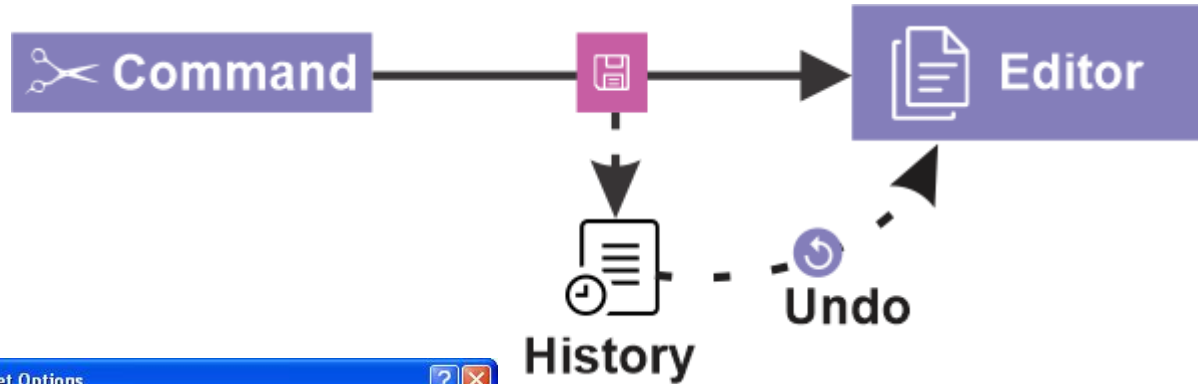
- Chain the receiving objects and pass the request along the chain until an object handles it.





# Memento (Non software example)

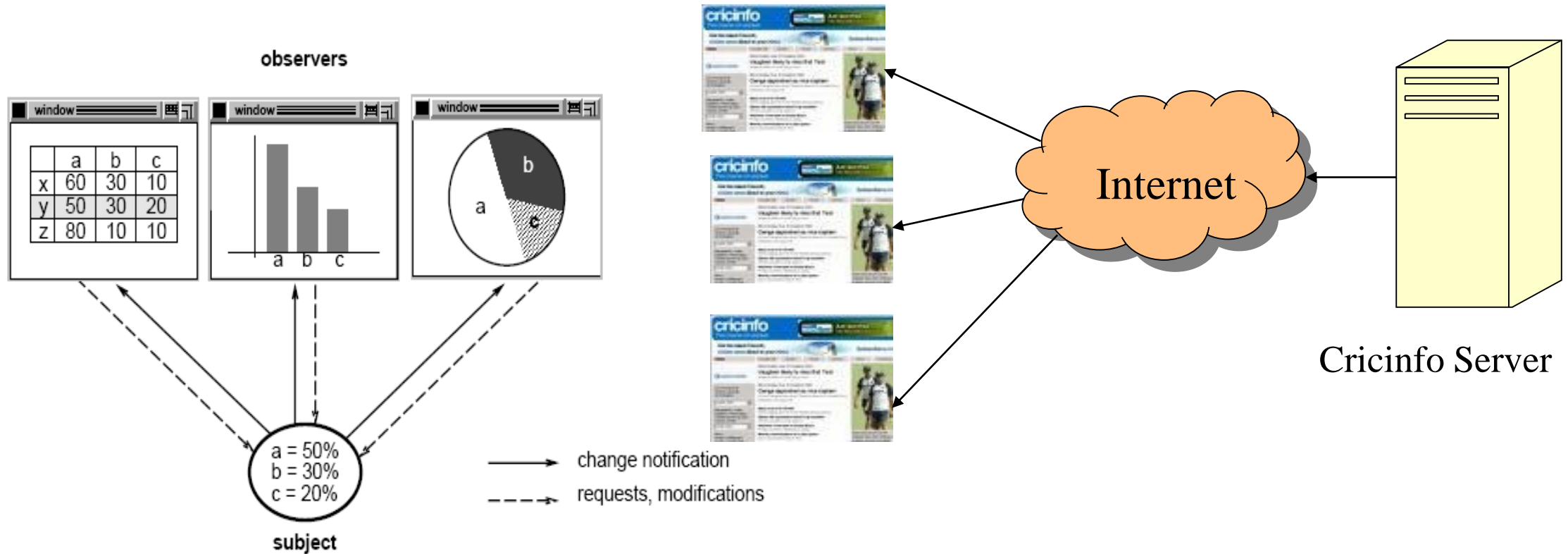
- Externalize object's state so that object can be restored to this state later.





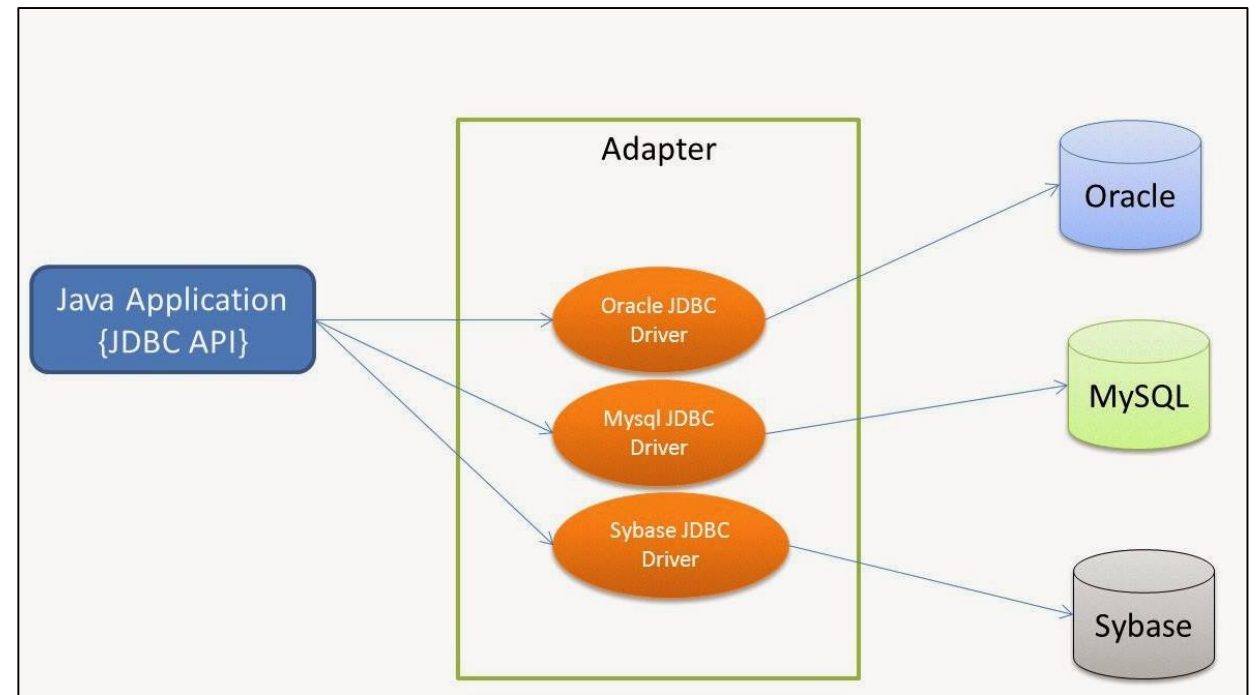
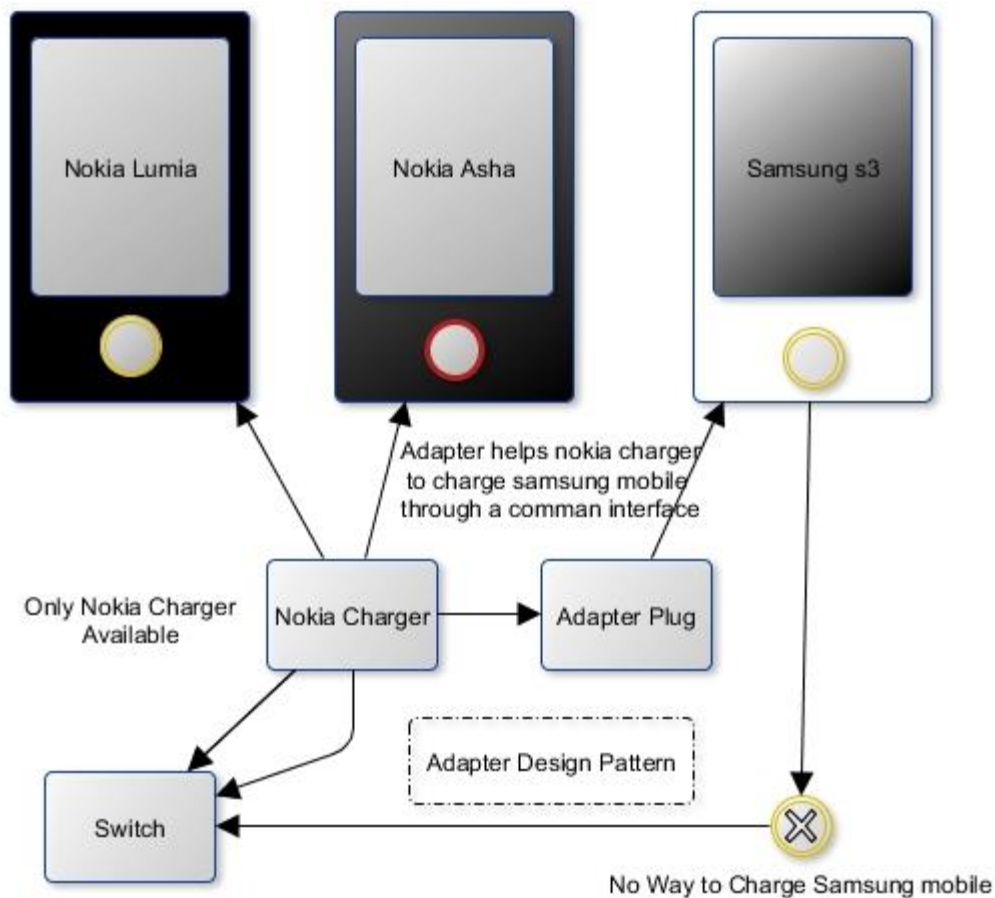
# Observer (Non software example)

When an object changes its state, all its dependents are notified.



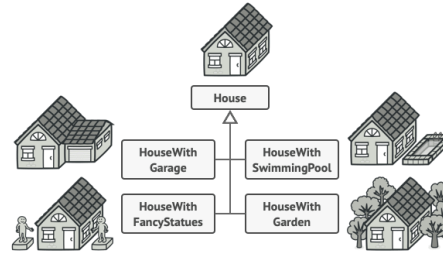
# Adapter

Adapter pattern works as a bridge between two incompatible interfaces. Convert the interface of a class into another Interface clients expect.

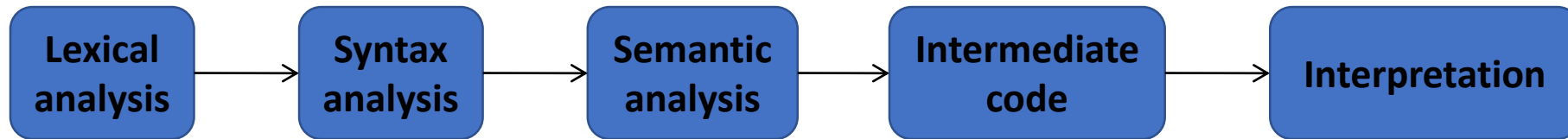


# Builder

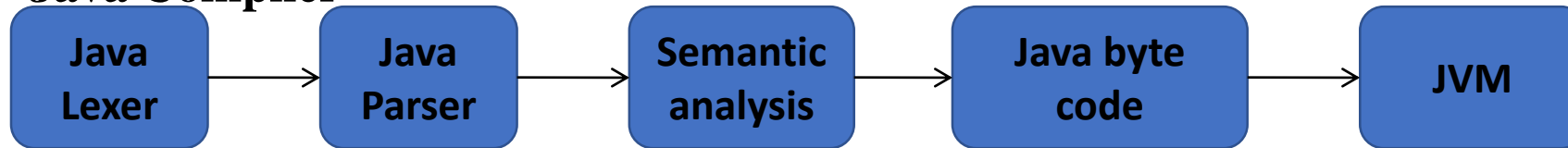
- Separate the construction process of a complex object from its representation so that the same construction Process can create different representations.



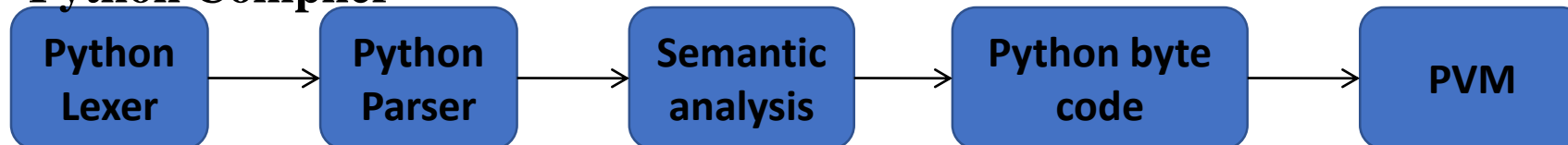
## Compiler process



## Java Compiler

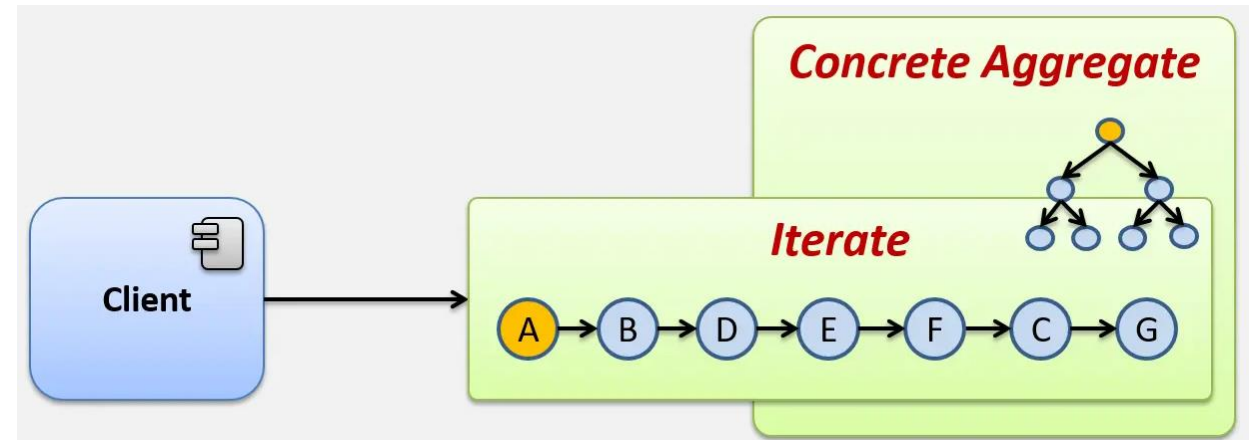
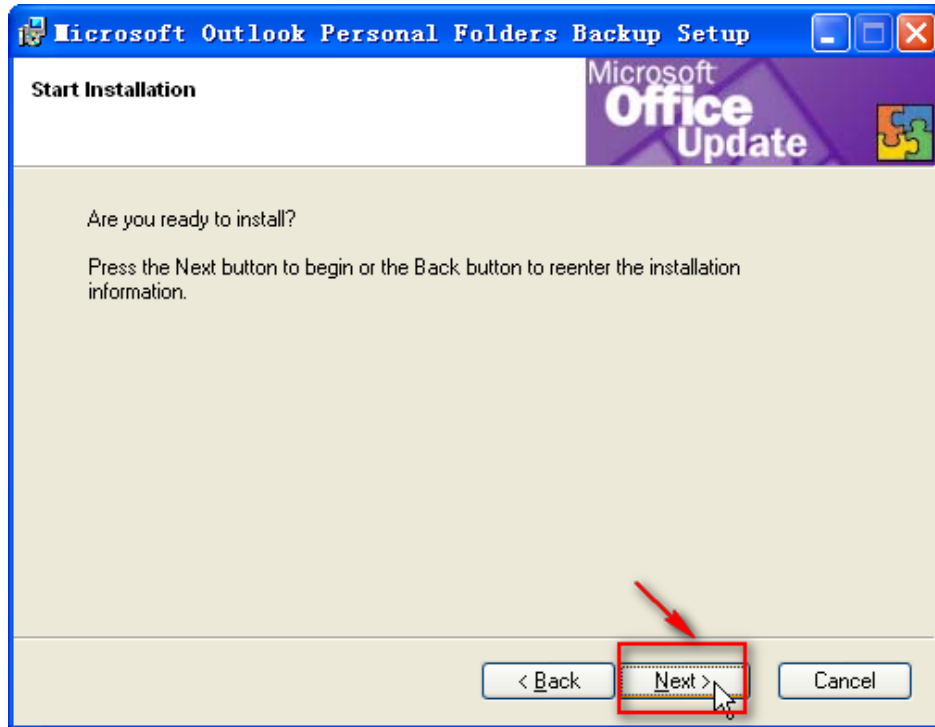


## Python Compiler



# Iterator

Provide a way to access the elements of a set sequentially.



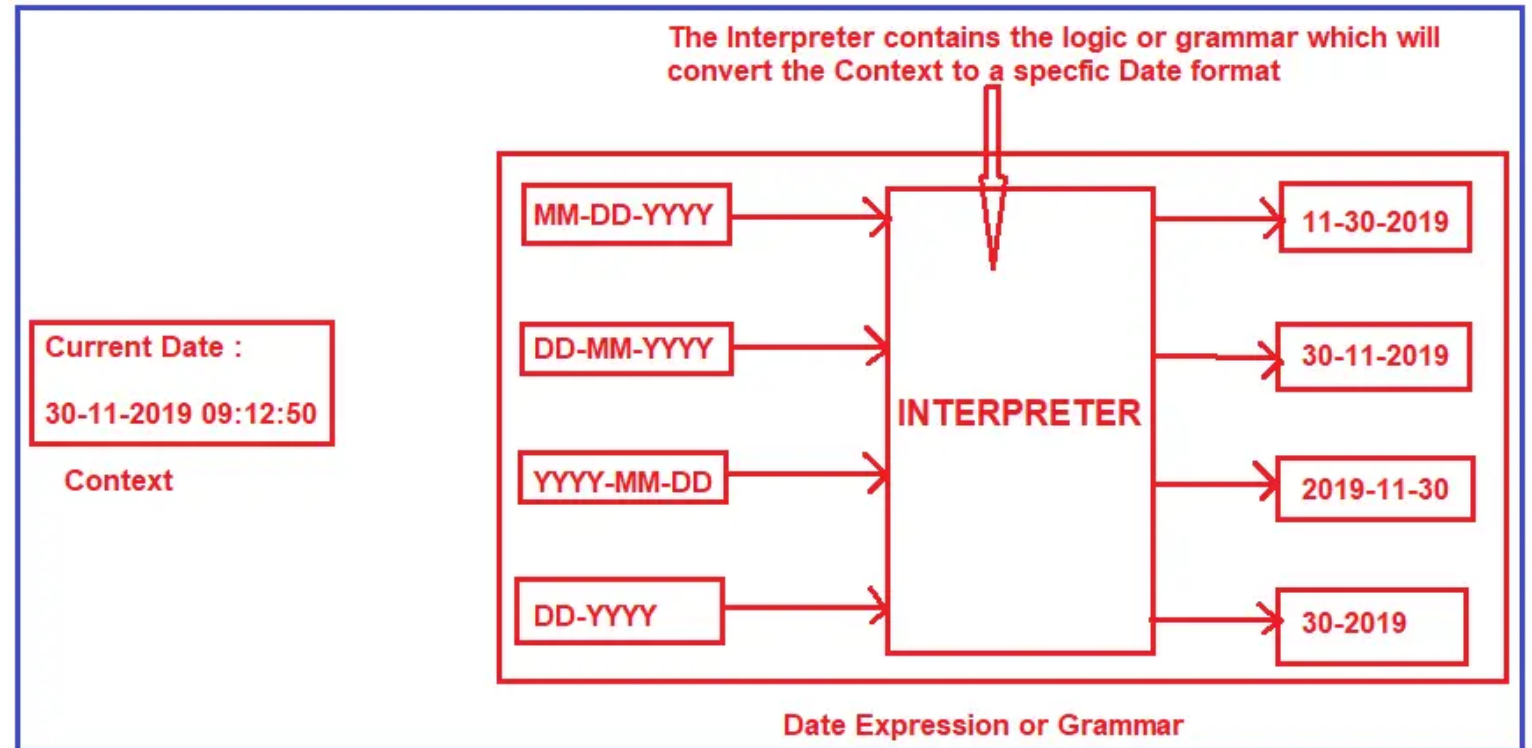
# Interpreter

Interpreter interprets the sentences in a language based on its grammar.

In Gtalk/Yahoo/WhatsApp messengers

:-) is interpreted as 😊

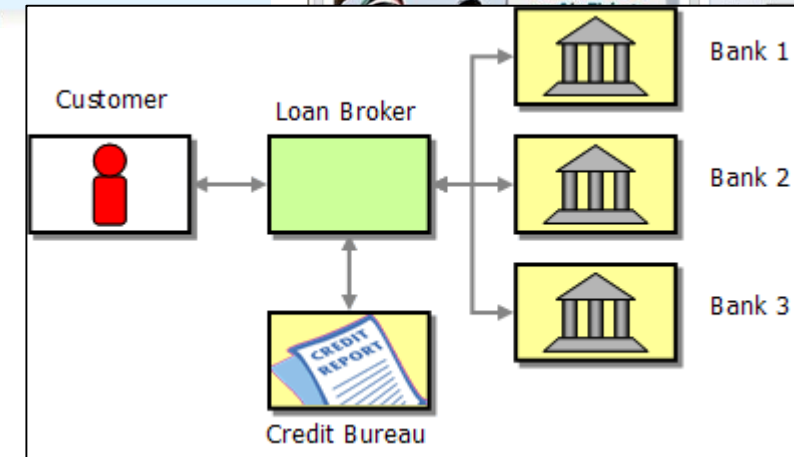
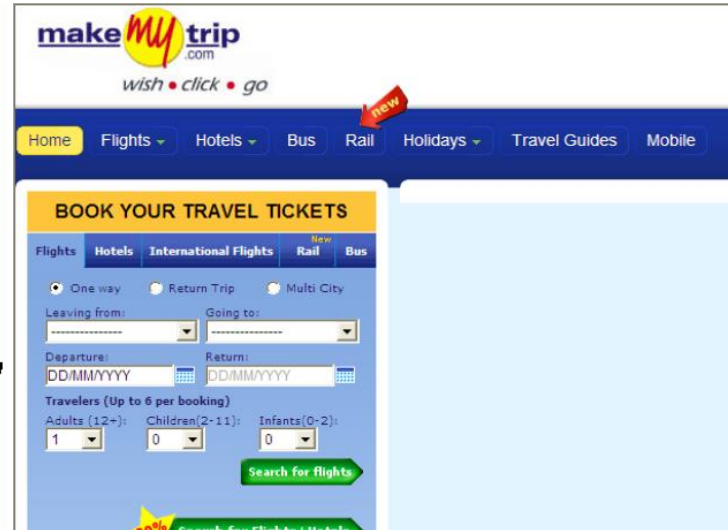
:-( is interpreted as ☹️





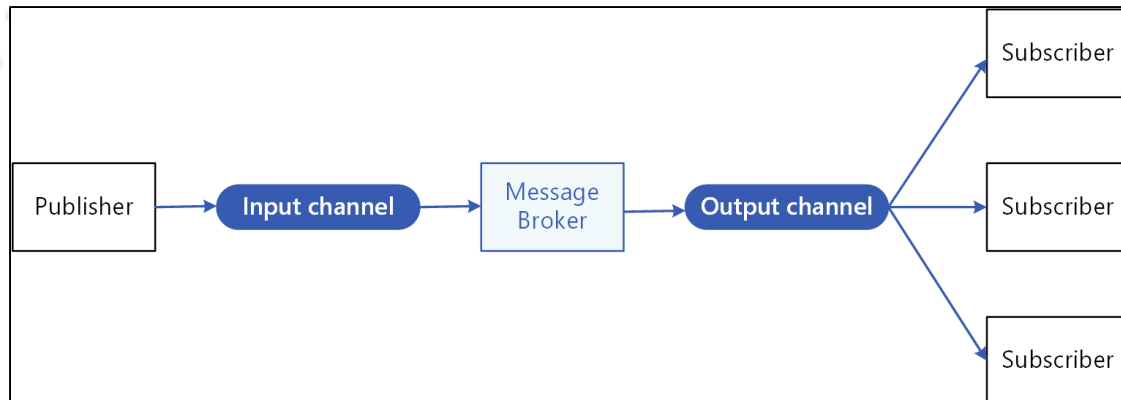
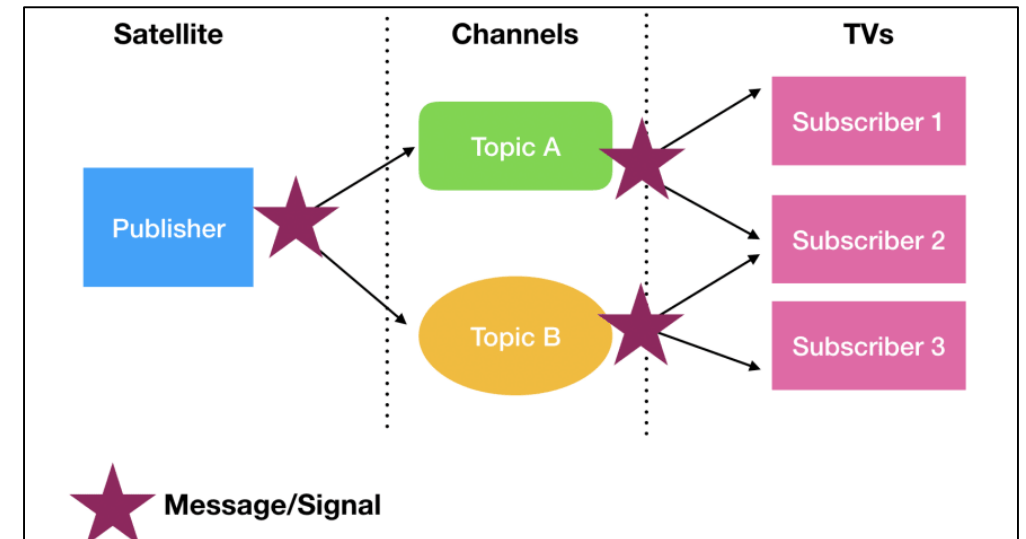
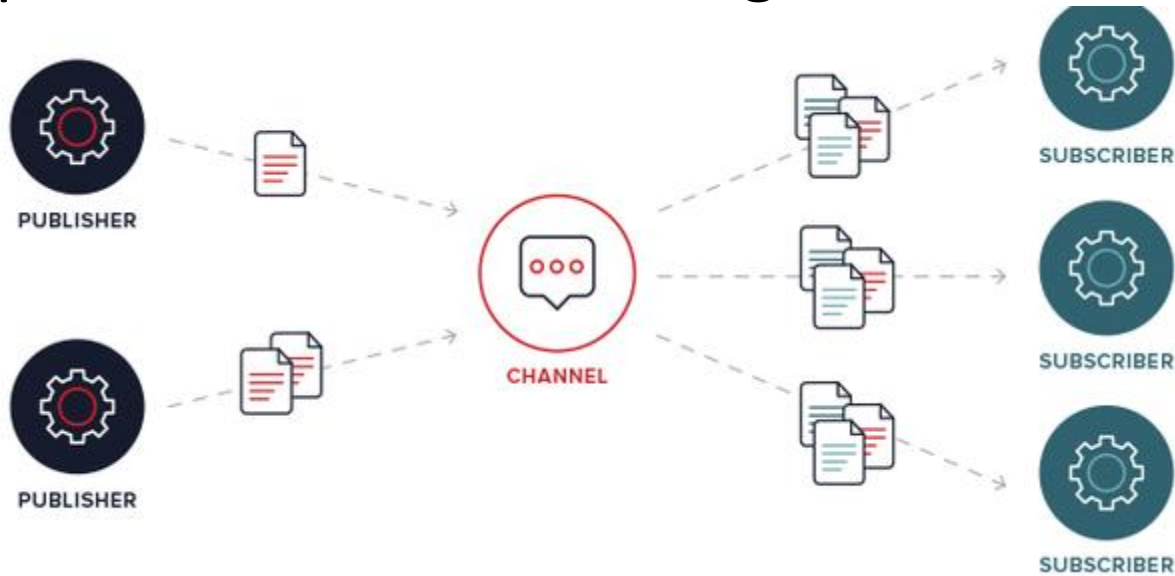
# Broker

- Broker component is responsible for coordinating communication between clients and remote servers.



# Publisher-Subscriber

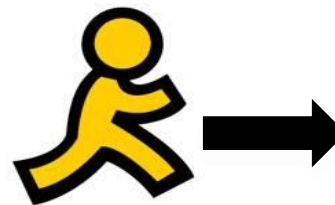
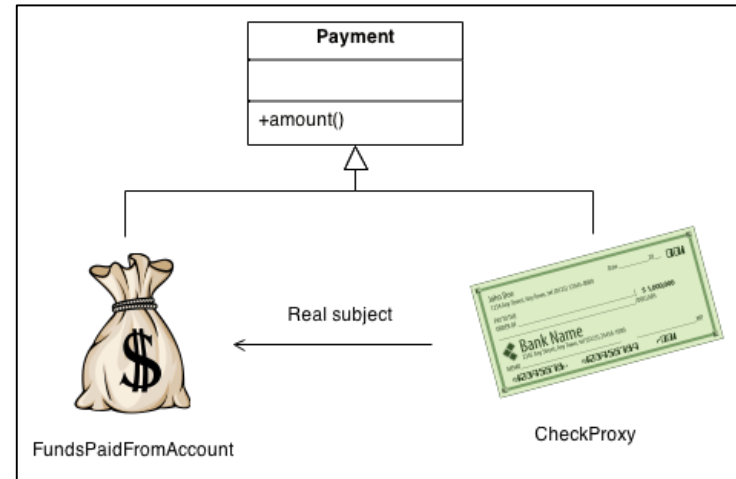
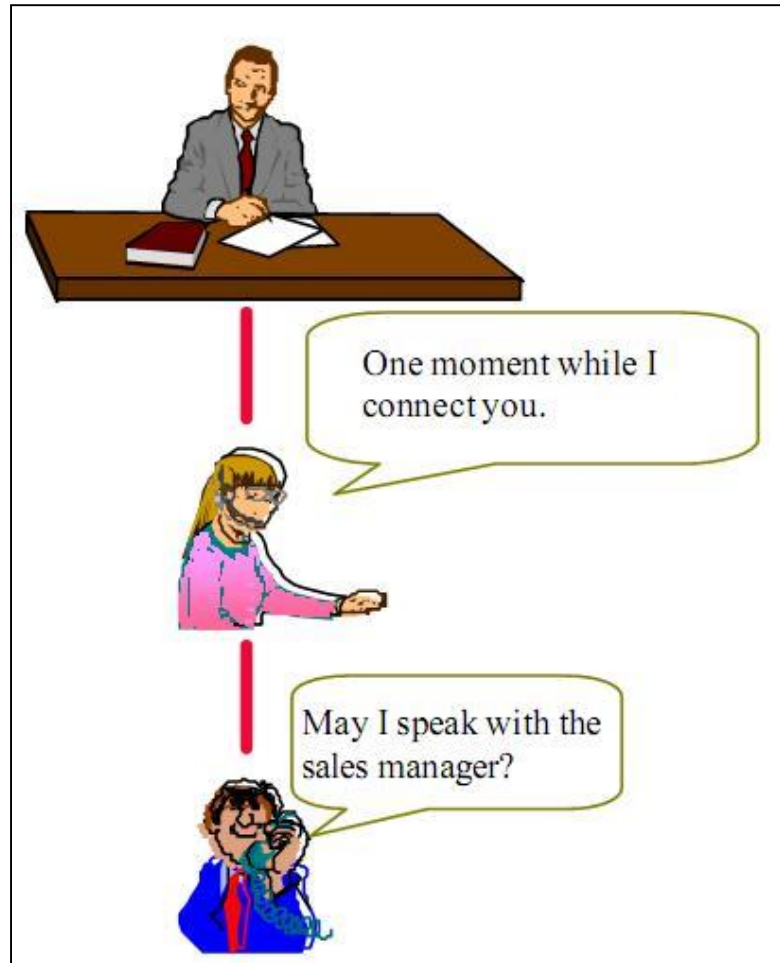
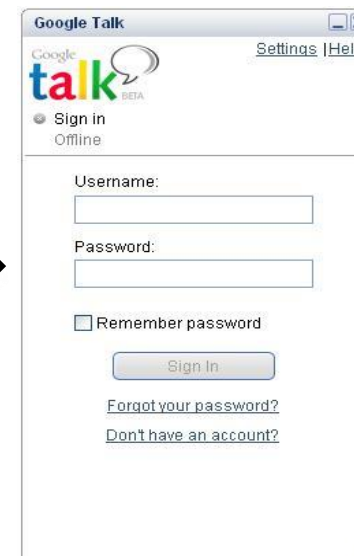
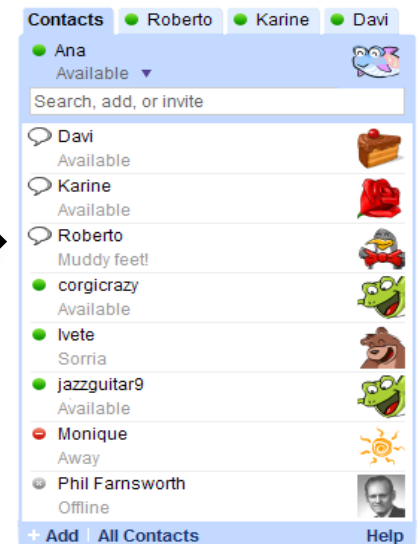
- Publishers register themselves to a broker and subscribers discover publisher from broker. E.g. Netflix, YouTube etc





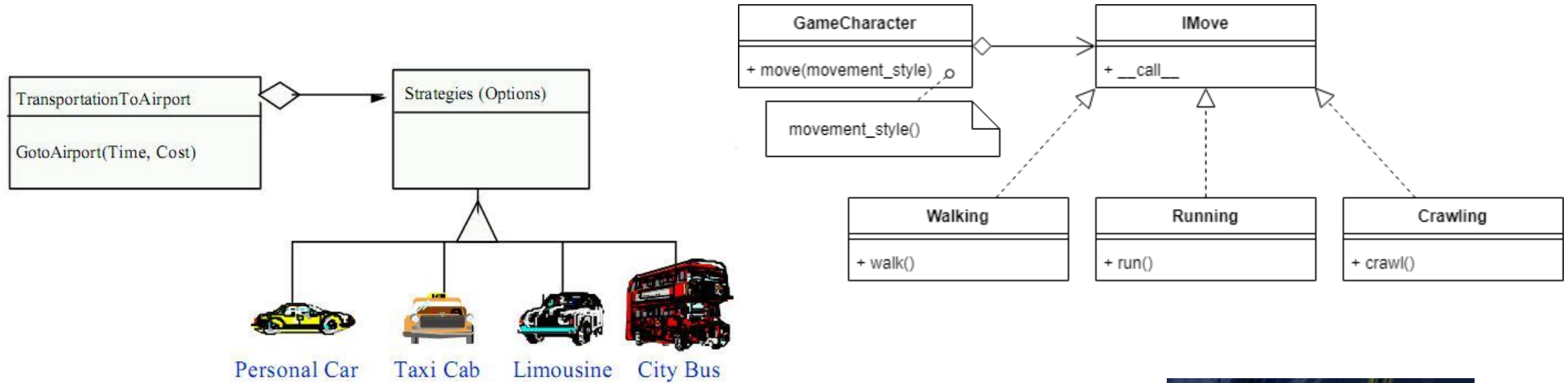
# Proxy

- Provide a surrogate or placeholder for another object to control access to it.

# Strategy

A Strategy defines a set of algorithms that can be used interchangeably.

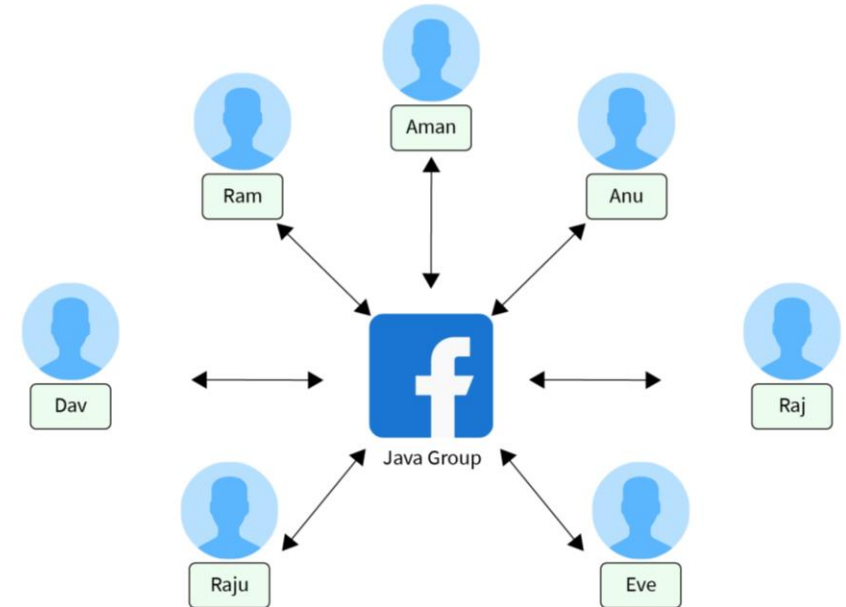
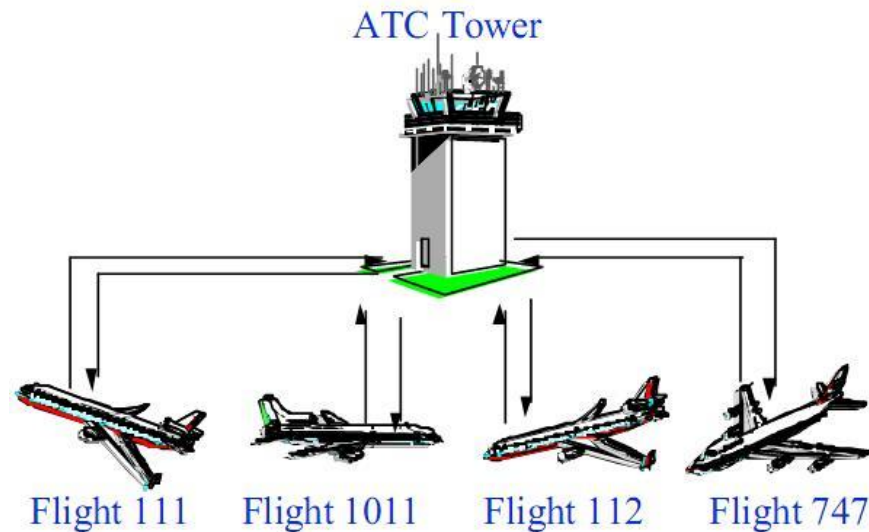


- Multiple interchangeable weapons available to attack an enemy in a video game.



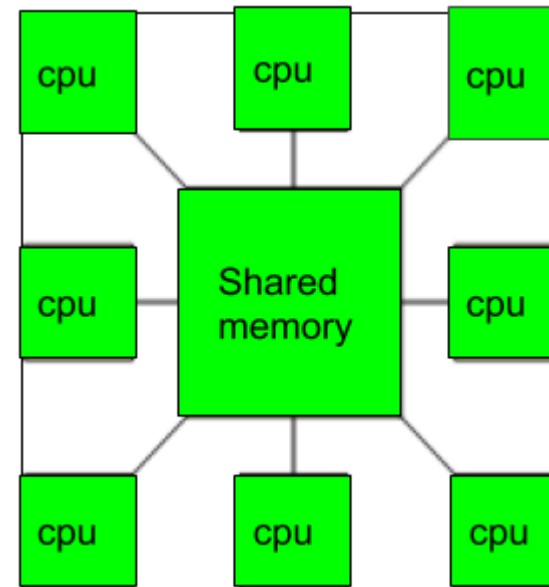
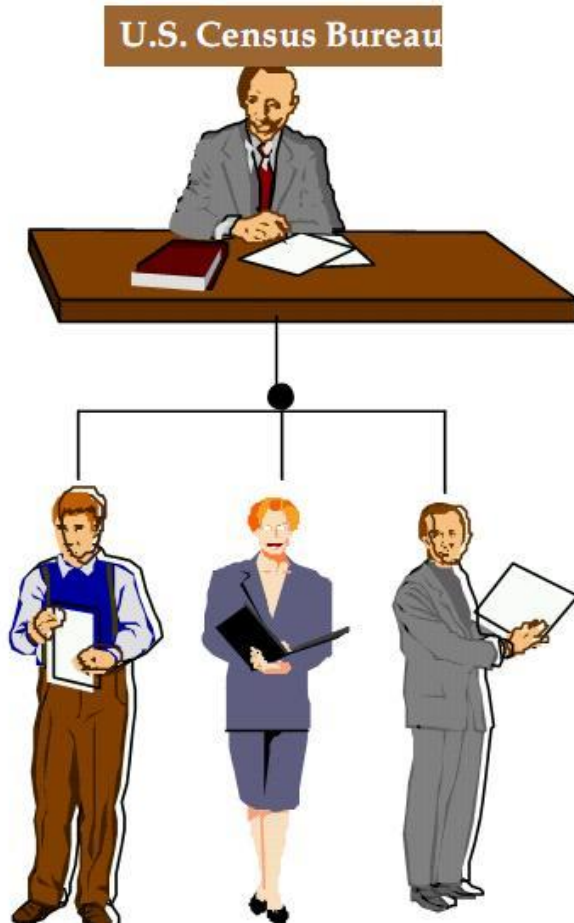
# Mediator

- Loose coupling between colleague objects is achieved by having colleagues communicate with the Mediator, rather than one another.



# Master-Slave

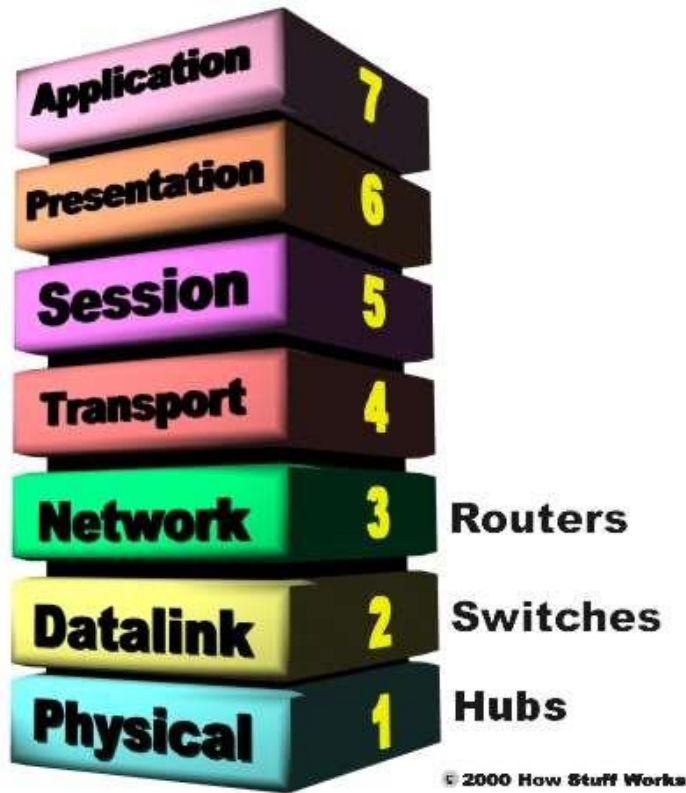
Master component distributes work to identical slave components and computes a final result from the results when the slaves return.



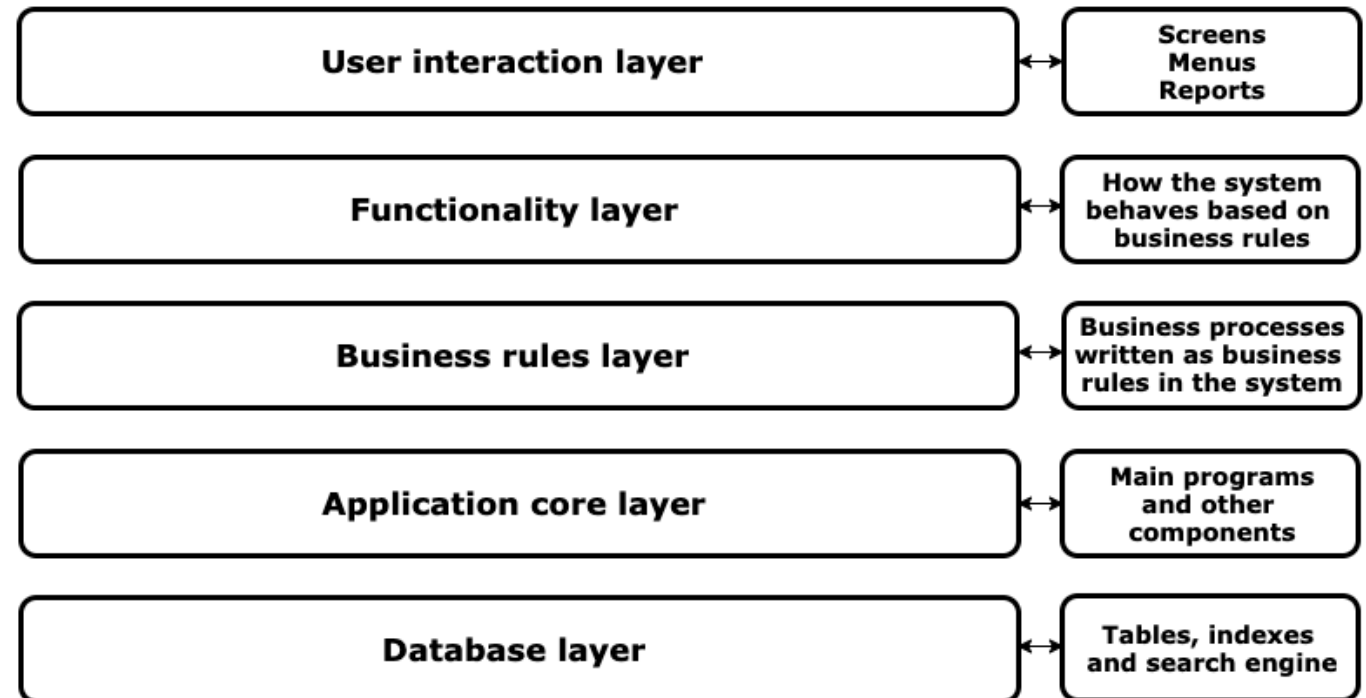


# Layers

- Layers structure applications whose dominant characteristic is a mix of low- and high- level issues.



## Layered Architecture High Level Diagram



# Reflection

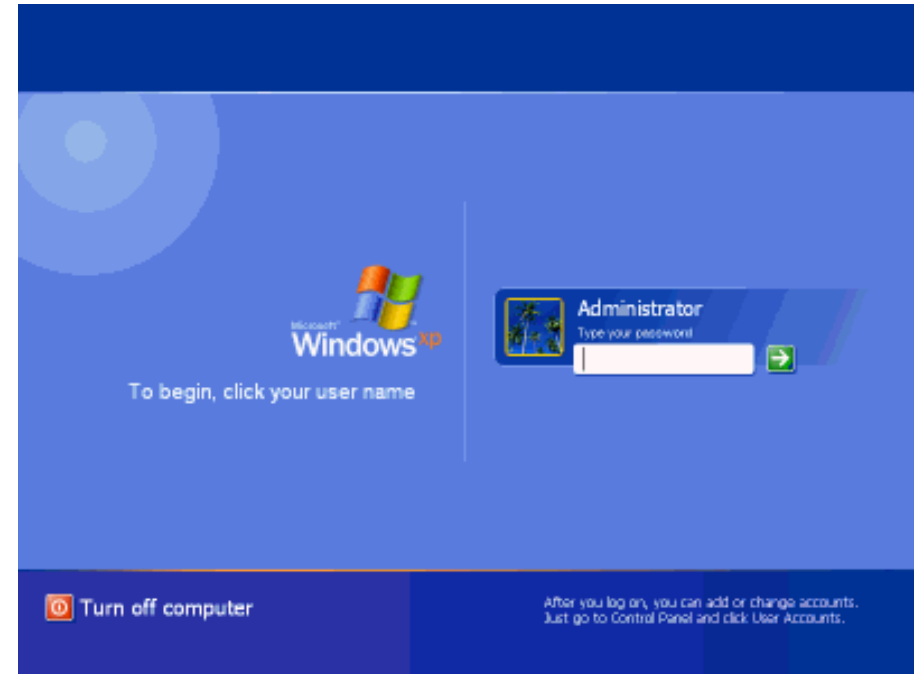
- Reflection pattern provides a mechanism for changing structure and behavior of a system dynamically.



Unknown Object

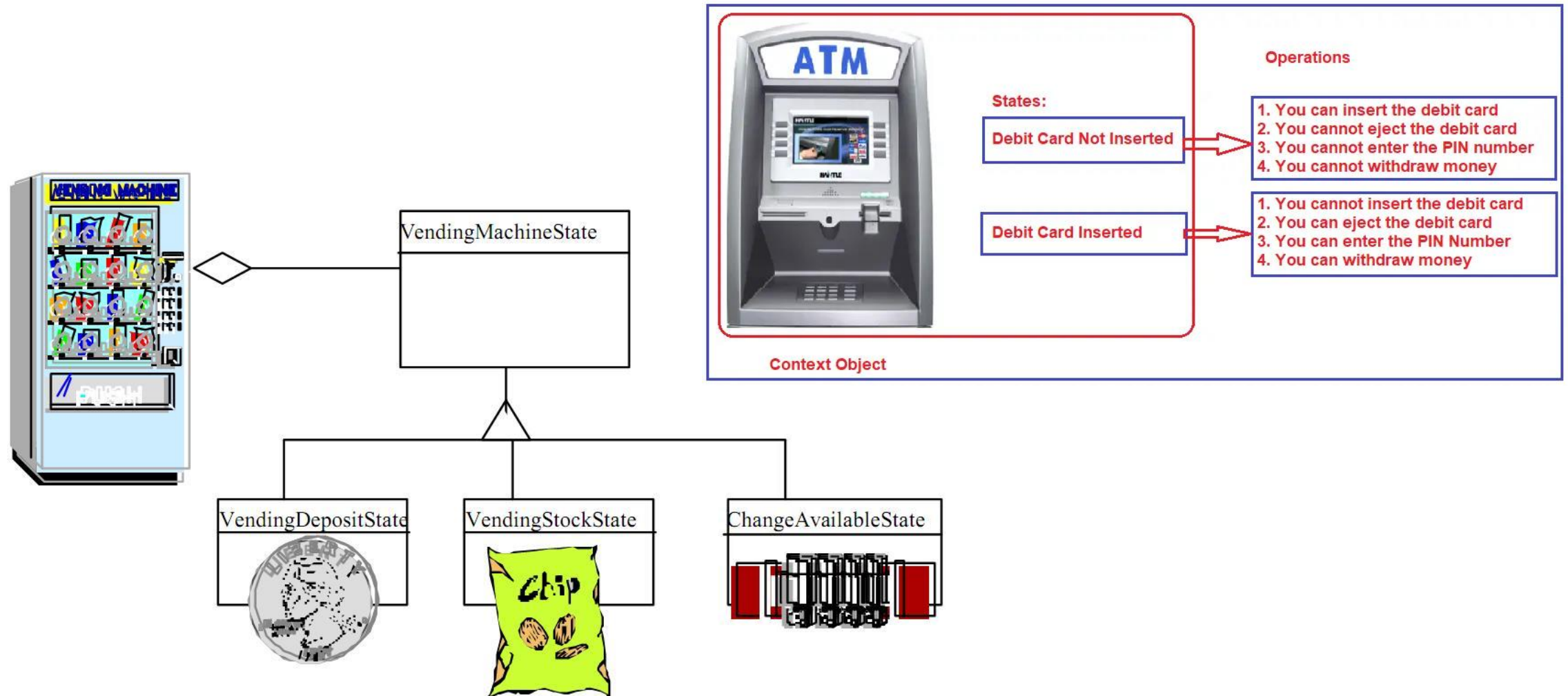
Reflection API

Modify behaviour of methods, classes,  
interfaces at runtime



# State (Software example)

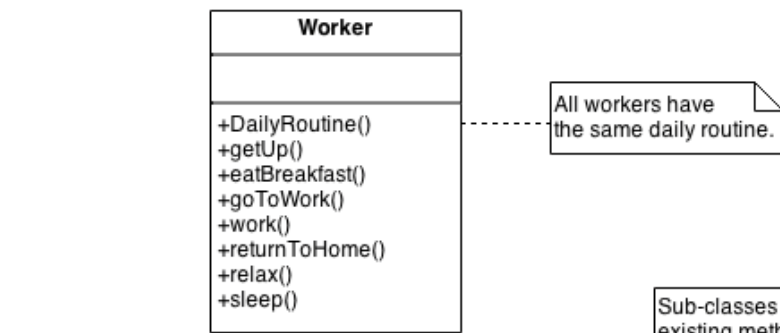
An object alters its behavior when its internal state changes.





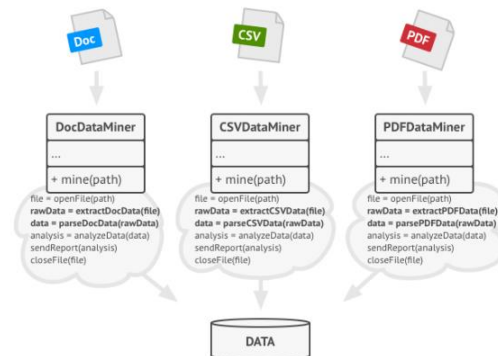
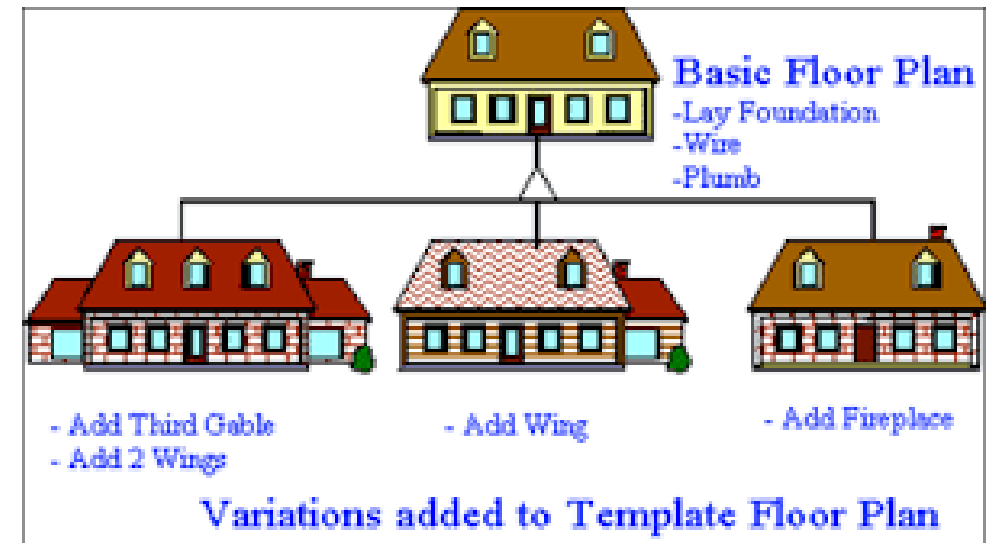
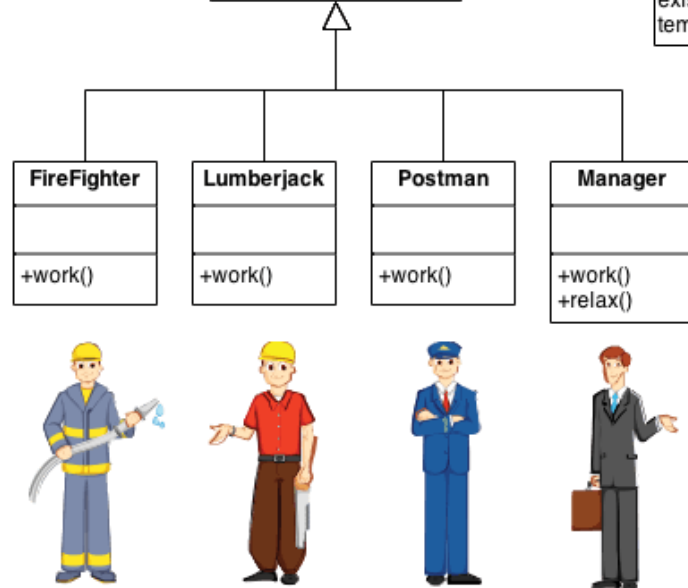
# Template Method (Software example)

- Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.



All workers have the same daily routine.

Sub-classes override existing methods of the template class.





# Summary

---

- Introduction to design patterns
- Design pattern classifications
- Examples of Design Patterns



# CSE301-SDA Assignment

---

- Use **any tools available** E.g. Lucidchart, Cacao, EdrawMax, Microsoft Visio Pro, Astah, StarUML to **design the Software based on** 4 +1 View Model
- Use **any tools available** E.g. FLUID, AppInventor (Android), LucidChart, Wavemaker, Visual Studio to **design a GUI** for your project
- Document/Compile/Zip and Submit to [salisu.garba@slu.edu.ng](mailto:salisu.garba@slu.edu.ng)
- **Due Date:** 20-05-2024

# Q & A

---



***Any Question?***