Design Patterns

Part I

https://sourcemaking.com/design_patterns

Software Developer

- *Knowledge of programming language*
  - ✓ Necessary, but not sufficient
- *Design notations (UML)*
  - ✓ Specification and Documentation

Software Developer

- *Design experience*
  - ✓ Abstraction
  - ✓ Flexibility
  - ✓ Reuse
  - ✓ Quality
  - ✓ Modularity

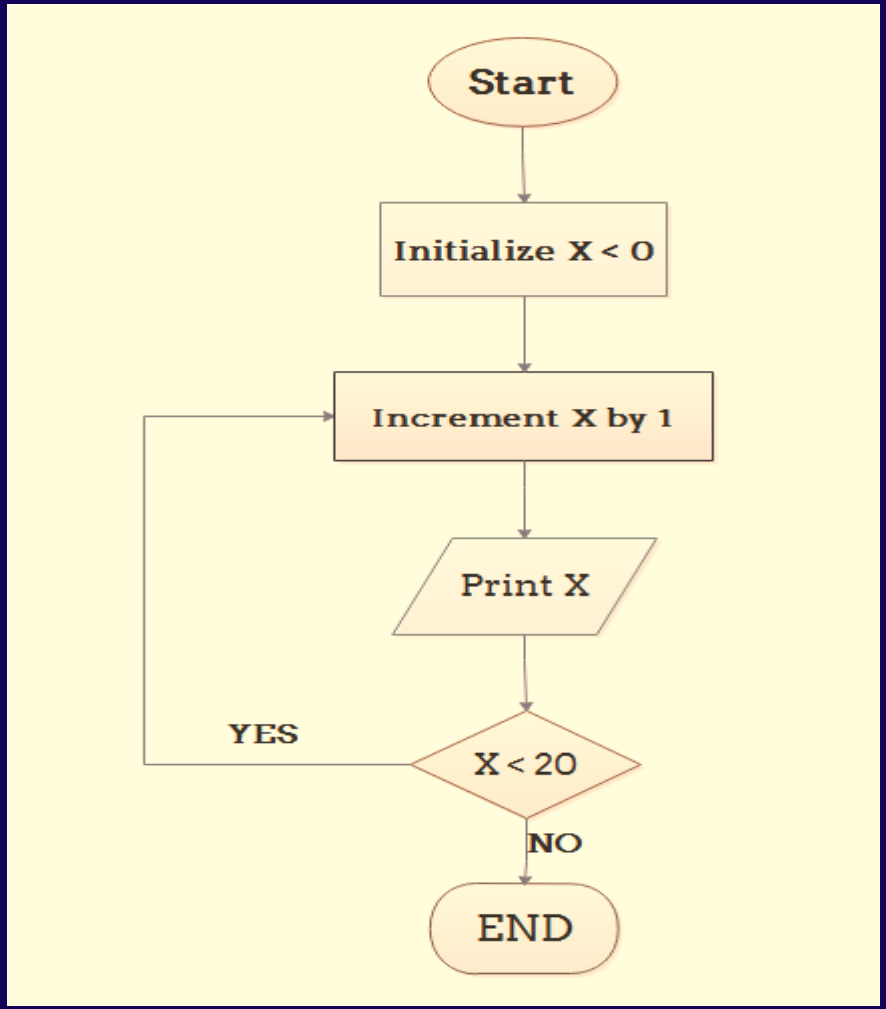
UML

	50s & 60s Prehistory	70–85 Structured Programming	85–01 OO Programming
Algorithmic structuring	Flowcharts	+Compositional constructs	+Object Interaction
Storage Structuring	Arrays	+Records, unions, pointers	+Object relationships
System structuring	Subroutines	+Modules (packages)	+Templates, Frameworks
Dominant Languages	ASM, Fortran, COBOL	PL/1, Pascal, C, Fortran 77, Ada	Ada, C++, Java
Important Languages	Algol 60, Algol 68, LISP	Simula, Smalltalk, APL, Prolog, Euclid	Haskell, SML

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

UML

- 1950s & 60s
  - ✓ unstructured use of conditional and unconditional branches
- Flowcharts evolved to help S/W engineers visualize the complexity



UML

- It was observed that all algorithms could be expressed using only 3 patterns of composition
- Moreover, each part has a meaning of its own (a function, or more generally a relation)
- Eventually flow-charts were replaced by pseudo-code, which is less expressive, but expresses the patterns well

System Structuring

- 50s & 60s:
  - ✓ unstructured collections of subroutines operating on global data structure
- 70 – 85:
  - ✓ subroutines operating on same data collected in a “module” together with that data
  - ✓ Some subroutines comprise the “public interface” to the module
  - ✓ The rest & the data are private
  - ✓ Black-box view

Module Relationships

- Modules use (depend on) each other.
  - ✓ X calls a subroutine of Y
  - ✓ X uses a data type defined in Y
  - ✓ X uses a constant defined in Y
- Often dependence is in layers.
  - ✓ Modules depend on modules below them as bricks depend on bricks below them
- A compiler as a layered system

Changing Views on Programs

- Old view
  - ✓ program is an algorithm that operates on variables
- New view
  - ✓ program is a collection of mutually dependant classes
  - ✓ program in execution is an evolving community of relating objects

Changing Views on Programs

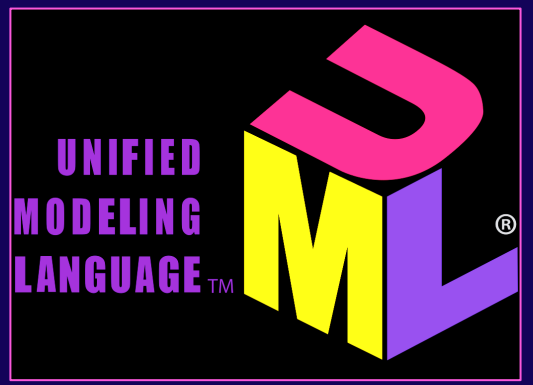
The main problem of software engineering is

Mastering complexity

UML

Unified Modeling Language (UML)

- General purpose modelling language
  - ✓ provide a standard way to visualize the design of a system



UML

- Language
  - ✓ express idea, not a methodology
- Modeling
  - ✓ Describing a software system at a high level of abstraction
- Unified:
  - ✓ UML is a world standard
  - ✓ Object Management Group (OMG): www.omg.org

UML

- *An industry-standard graphical language*
  - ✓ specifying
  - ✓ visualizing
  - ✓ constructing
  - ✓ documenting
- Uses mostly graphical notations
- Simplifies the complex process of software design

UML

Why use UML?

- ✓ Use graphical notation: more clearly than natural language (imprecise) and code (too detailed)
- ✓ Help acquire an overall view of a system
- ✓ Not dependent on any one language or technology
- ✓ moves from fragmentation to standardization

https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

UML

- Complex applications need collaboration and planning from multiple teams
  - ✓ require a clear and concise way to communicate
- Businessmen do not understand code
  - ✓ communicate essential requirements, functionalities and processes of the system
- *Visualize processes, user interactions and static structure of the system*

UML

- Linked with object oriented design and analysis
- Makes the use of elements and forms associations between them to form diagrams
- Diagrams
  - ✓ Structural Diagrams
  - ✓ Behavior Diagrams

UML

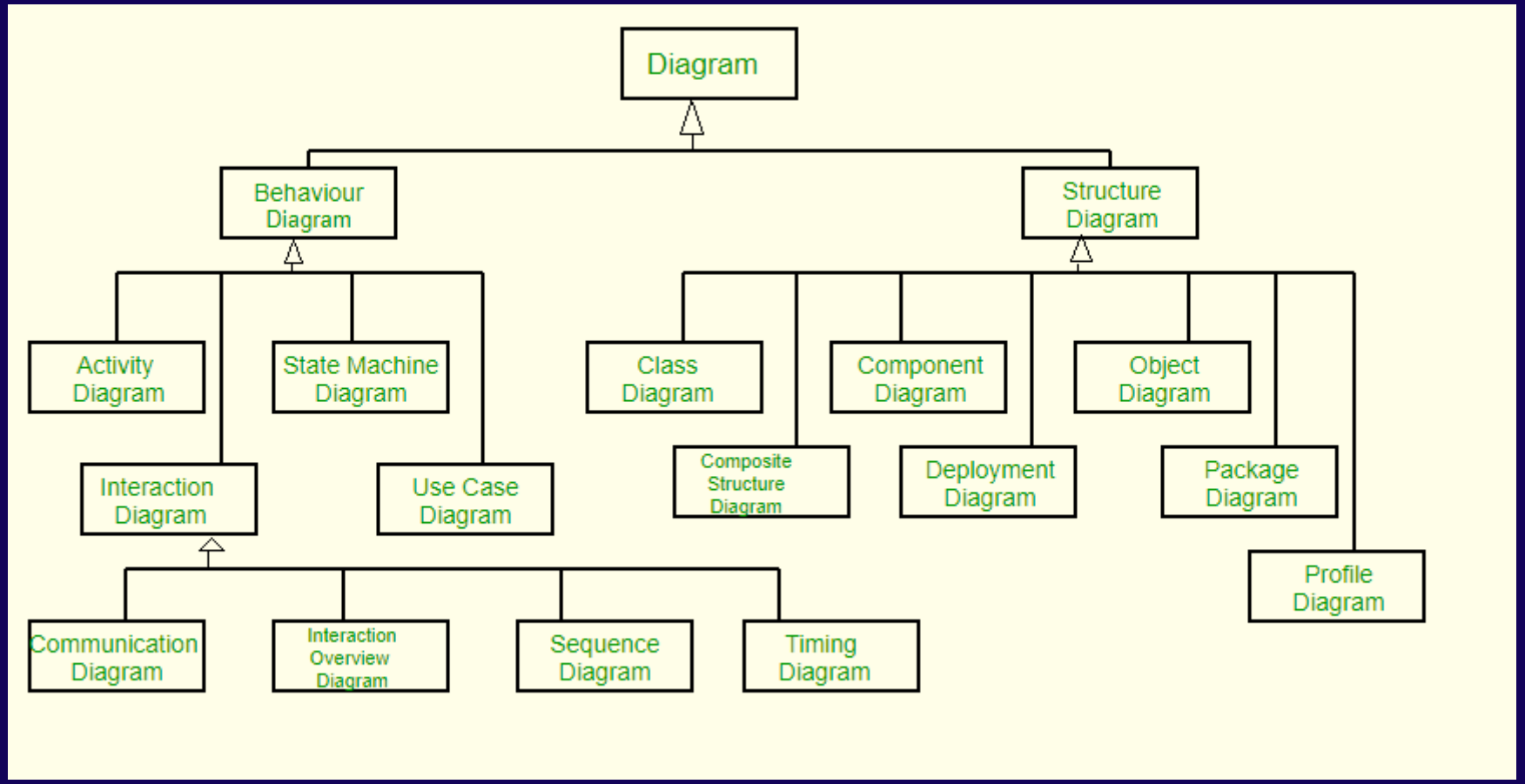
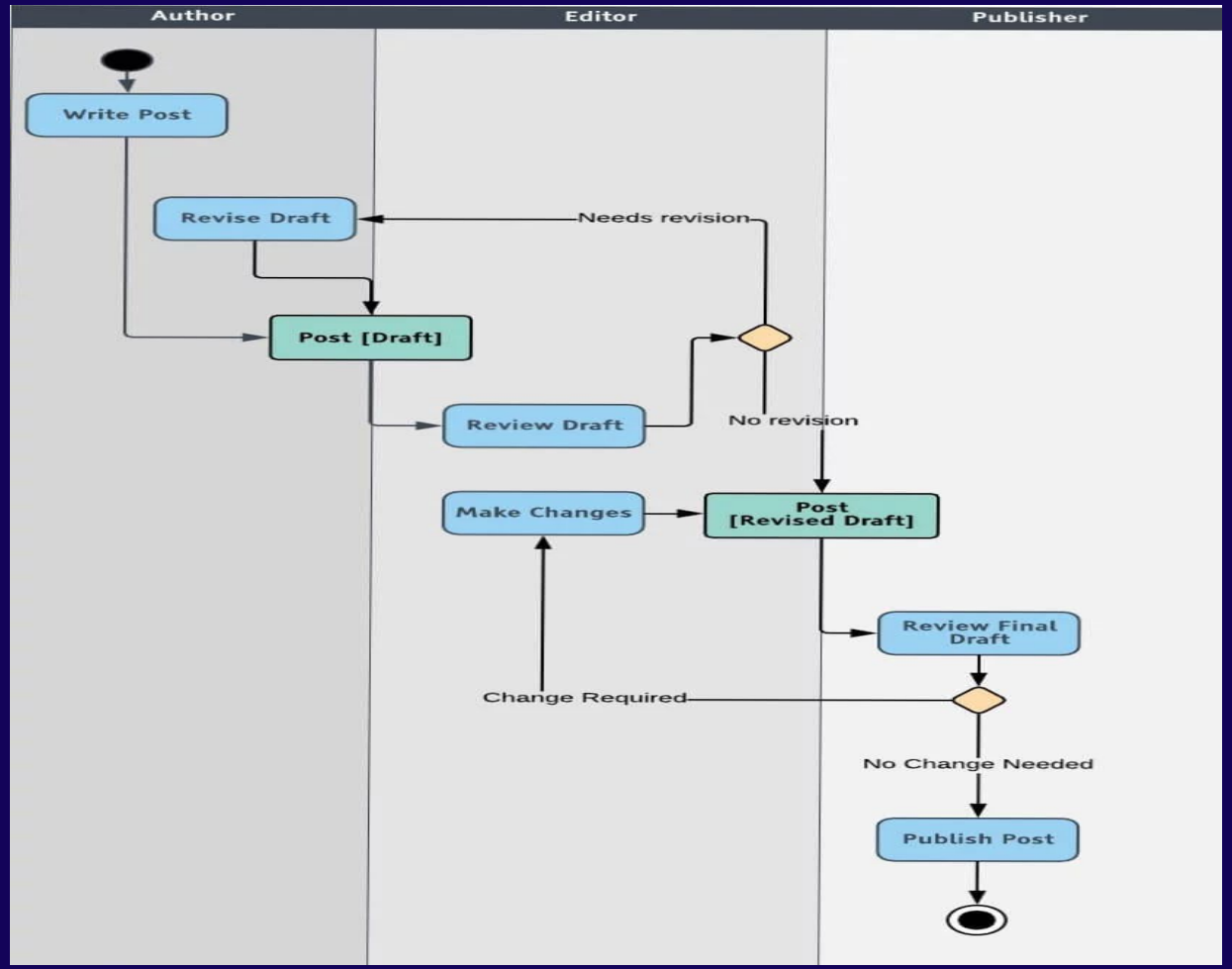
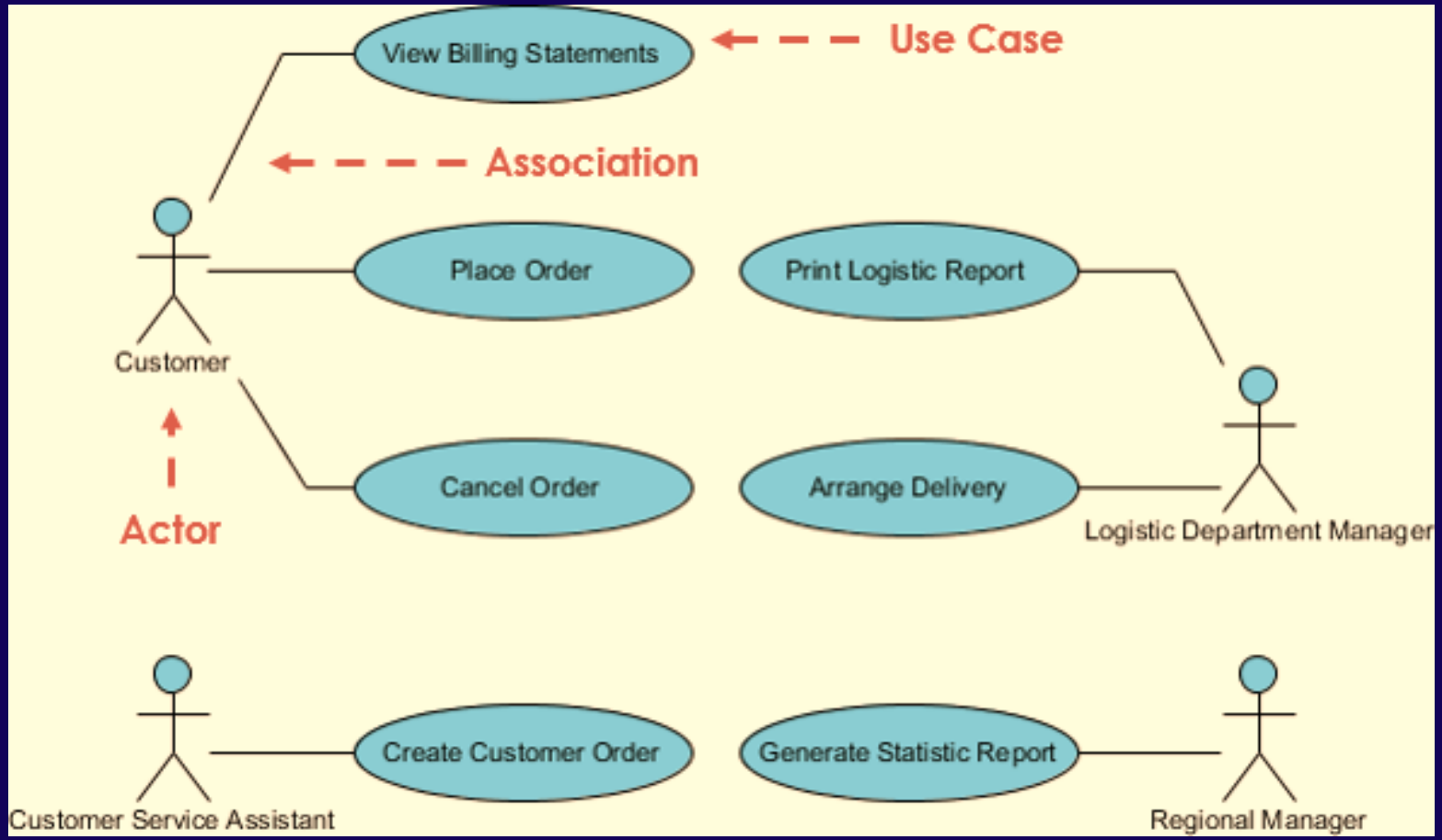


Diagram
Behaviour Diagram
Structure Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Use Case Diagram
Communication Diagram
Interaction Overview Diagram
Sequence Diagram
Timing Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram

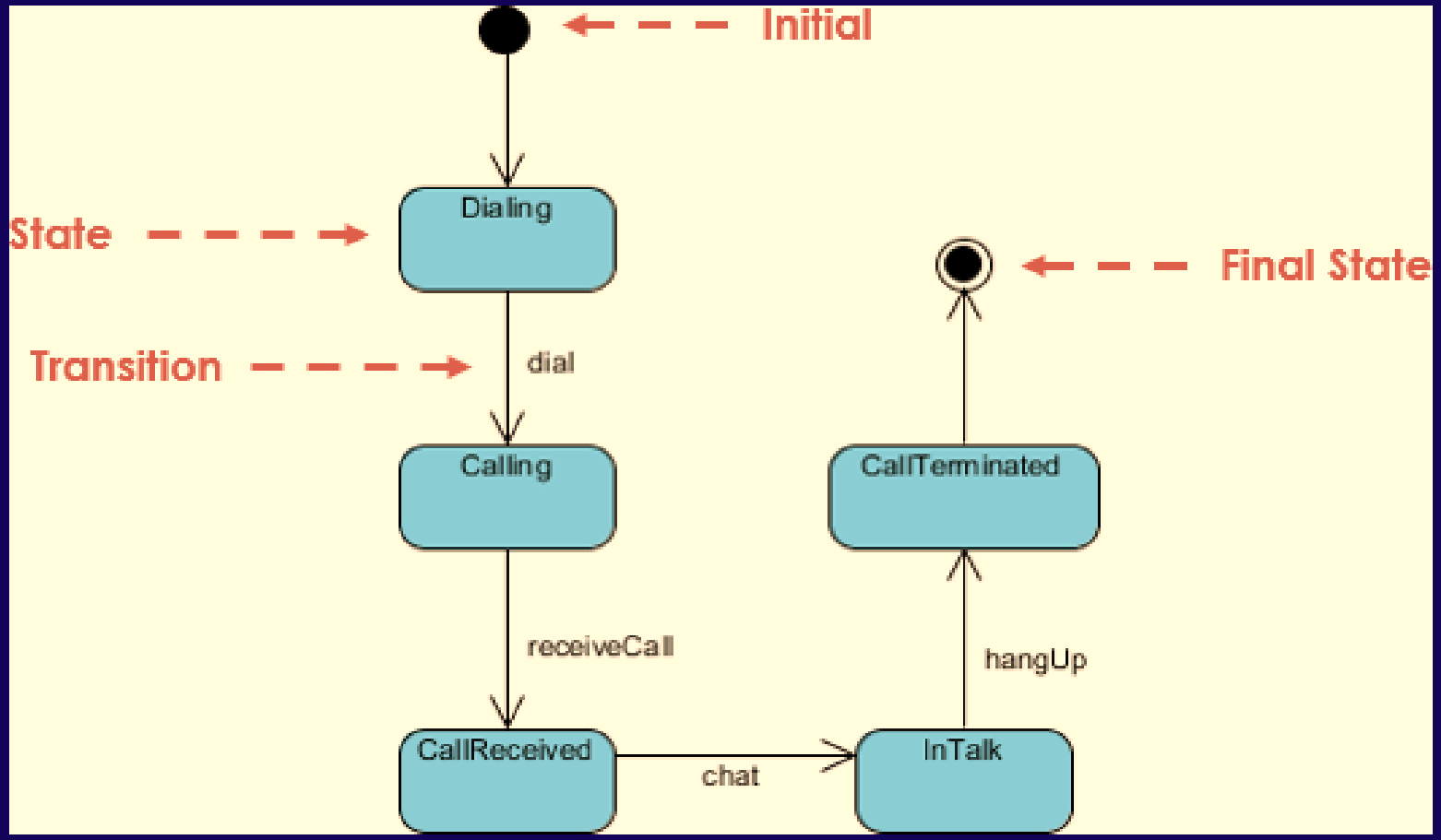
UML: Activity Diagram



UML: Case Diagram

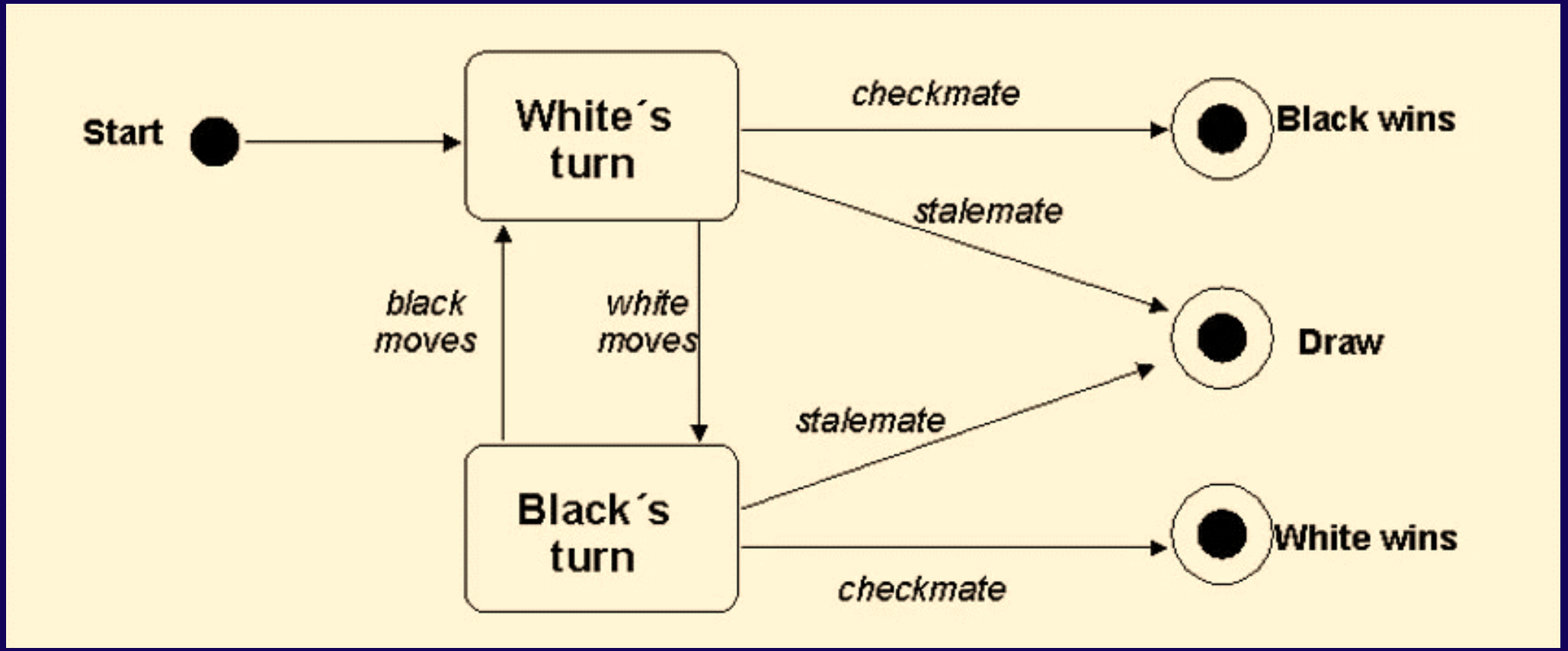


UML: State Machine Diagram



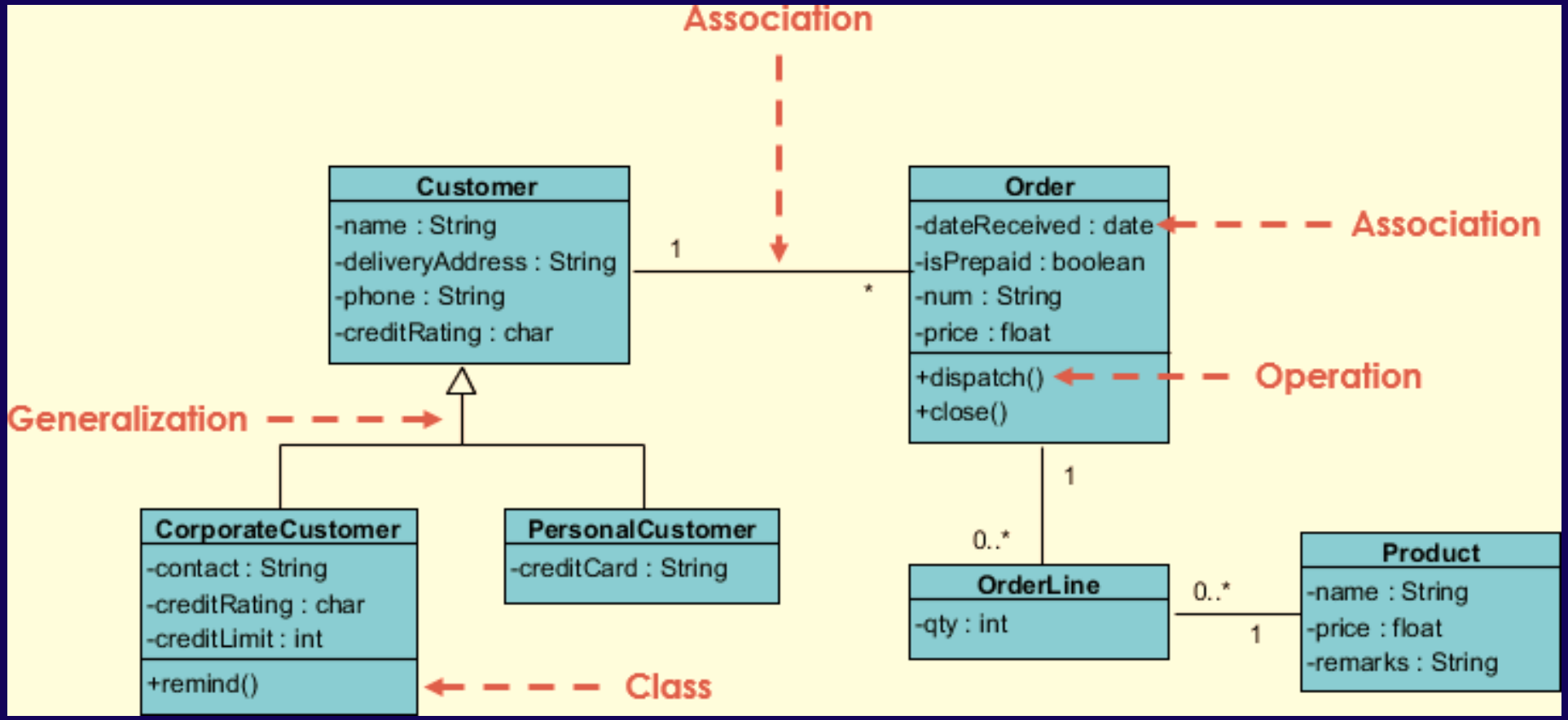
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

UML: State Machine Diagram



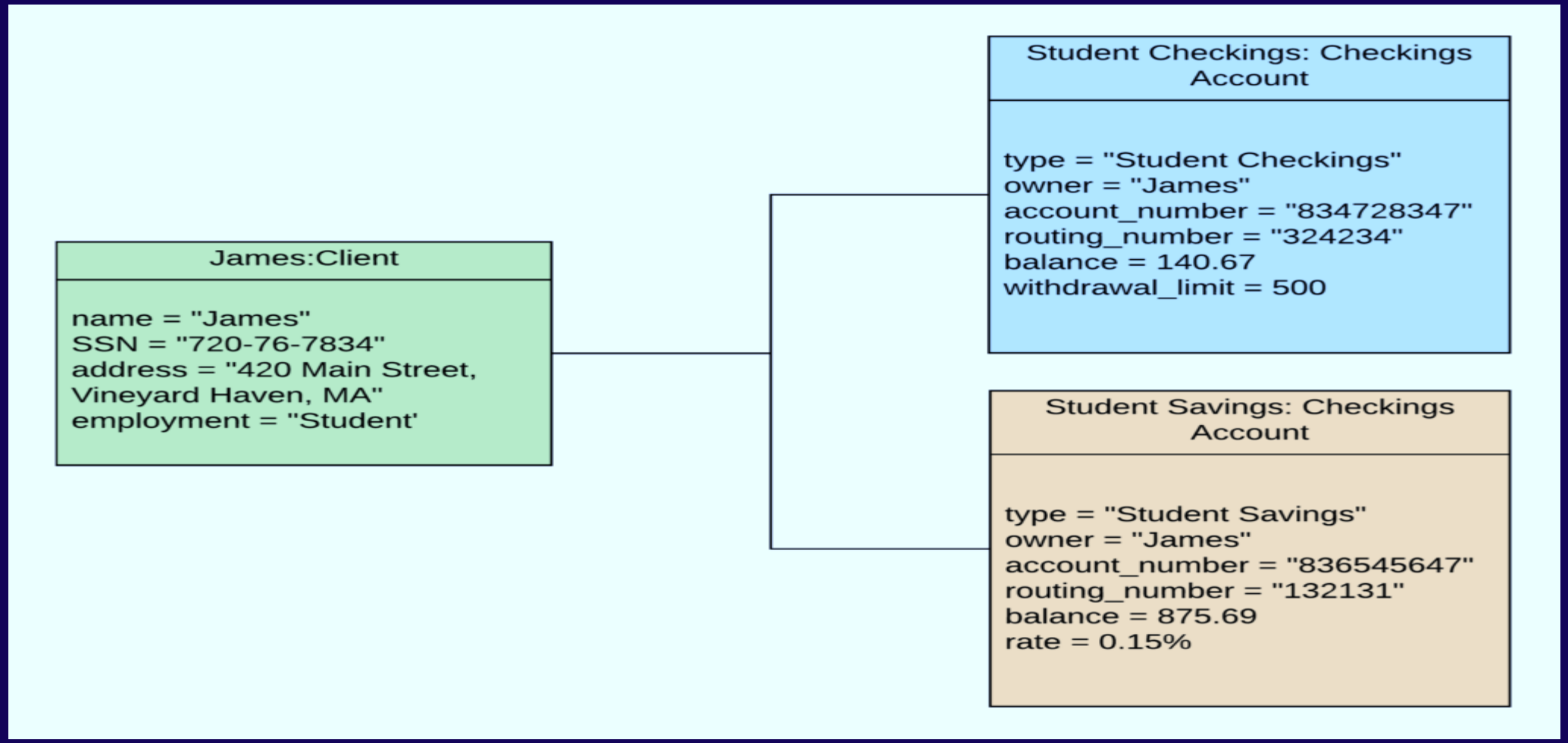
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

UML: Class Diagram



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

UML: Object Diagram



Part II

https://sourcemaking.com/design_patterns

What are Design Patterns?

DESIGN PATTERNS

are optimized, reusable solutions to the programming problems commonly occurring in software design

https://code.tutsplus.com/articles/a-beginners-guide-to-design-patterns--net-12752

What are Design Patterns?

- It is a *template* that has to be implemented in the correct situation
  - Not code reuse
    - ✓ solution/strategy reuse
    - ✓ interface reuse
- It's not *language-specific*

https://code.tutsplus.com/articles/a-beginners-guide-to-design-patterns--net-12752

Types

- Mapping design problems to proven solution
- Identifying and naming recurring structures and behaviors
- Conveying architectural knowledge
- Codifying design expertise
- Enabling systematic reuse

Types

- *Design patterns* reside in the domain of modules and interconnections
- Architectural patterns describe an overall pattern followed by an entire system
- *Design patterns* describes a solution to a common problem arising within a context (software)
  - Mobile devices, aerospace, electronic trading, etc.

Types

- *Algorithm Strategy patterns*
  - ✓ Addressing concerns related to high-level strategies describing how to exploit application characteristics on a computing platform

https://en.wikipedia.org/wiki/Software_design_pattern

Types

- *Computational design patterns*
  - Addressing concerns related to key computation identification

https://en.wikipedia.org/wiki/Software_design_pattern

Types

- *Execution patterns*
  - ✓ Which address issues related to lower-level support of application execution, including strategies for executing streams of tasks and for the definition of building blocks to support task synchronization.

Types

- *Implementation strategy patterns*
  - ✓ Addressing concerns related to implementing source code to support program organization, and the common data structures specific to parallel programming

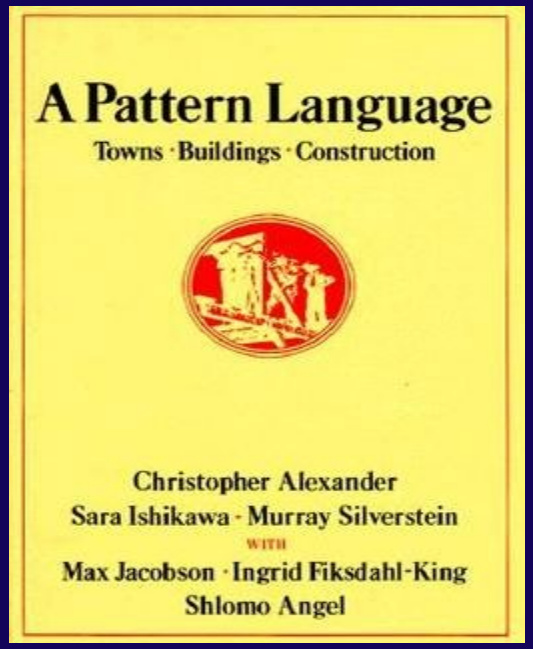
https://en.wikipedia.org/wiki/Software_design_pattern

Types

- *Structural design patterns*
  - ✓ Addressing concerns related to global structures of applications being developed

History of patterns

The concept of a "pattern" was first expressed in Christopher Alexander's work *A Pattern Language* in 1977 (2543 patterns)



https://code.tutsplus.com/articles/a-beginners-guide-to-design-patterns--net-12752

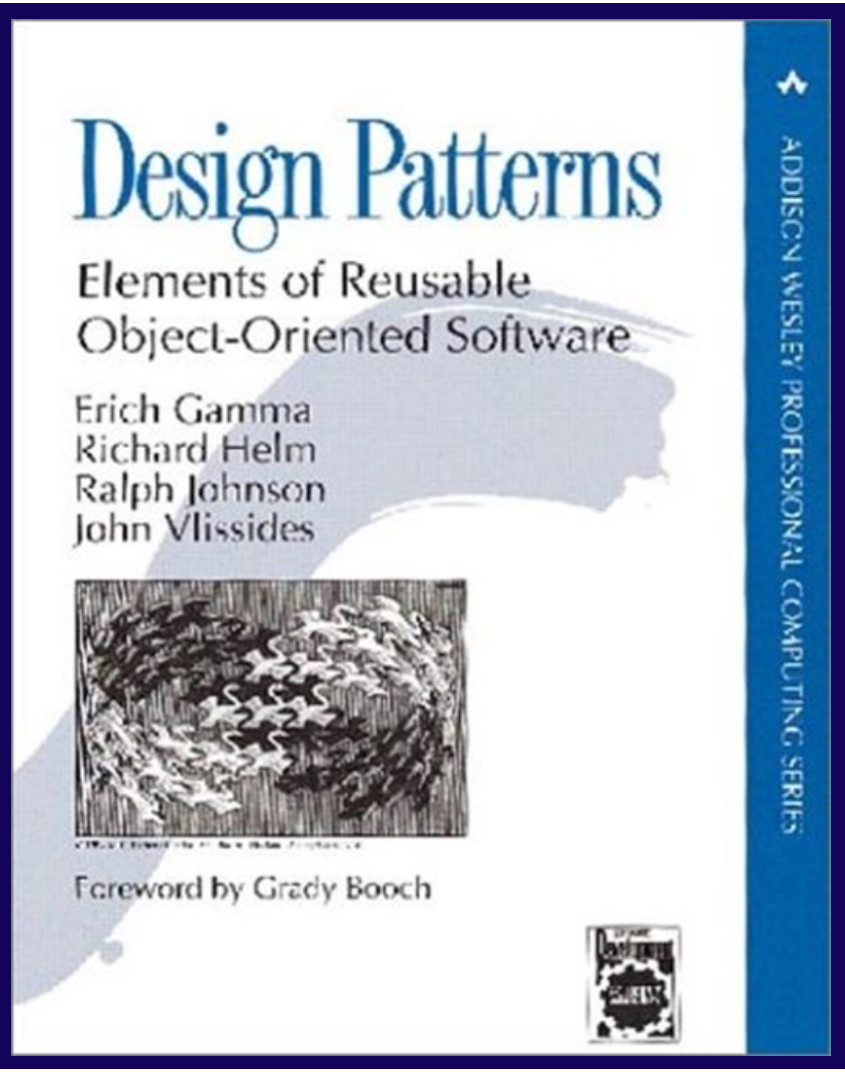
History of patterns

- In 1987, Kent Beck and Ward Cunningham began experimenting with the idea of applying patterns to programming
- In 1990 a group called the Gang of Four or "GoF“ (*Gamma*, *Helm*, *Johnson*, *Vlissides*) compile a catalog of design patterns

https://code.tutsplus.com/articles/a-beginners-guide-to-design-patterns--net-12752

Gang of Four

- The book that started it all
- Community refers to authors as the “Gang of Four”



Gang of Four

Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides (Addison-Wesley, 1995)

- Design Patterns book catalogs 23 different patterns as solutions to different classes of problems, in C++ & Smalltalk
- The problems and solutions are broadly applicable, used by many people over many years

Benefits of using patterns

A common design vocabulary

- allows engineers to abstract a problem and talk about that abstraction in isolation from its implementation
- embodies a culture; domain-specific patterns increase design speed

Marty Stepp, 2007.

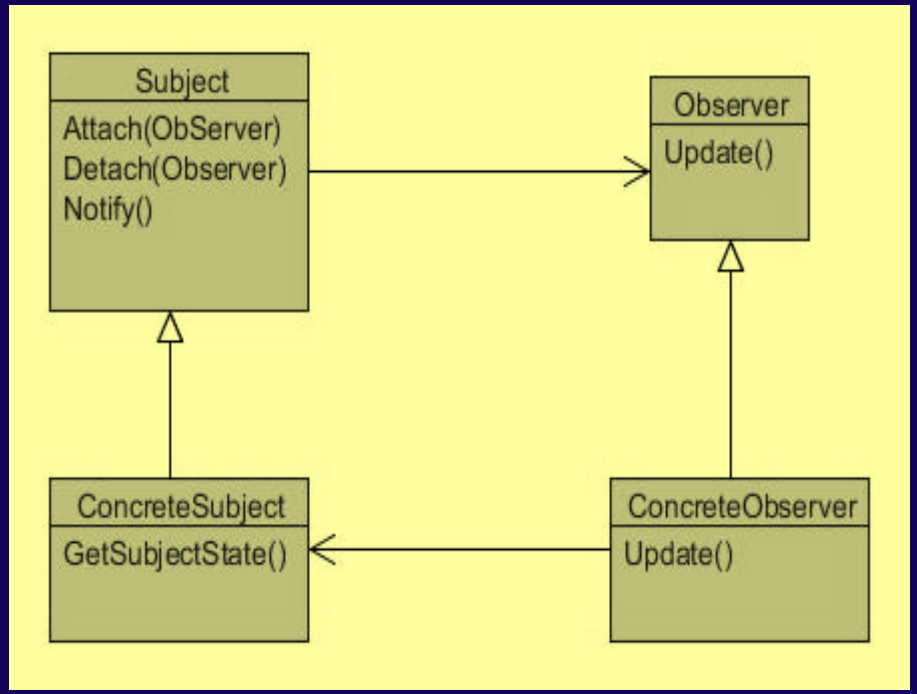
Benefits of using patterns

- Patterns capture design expertise and allow that expertise to be communicated
  - promotes design reuse and avoid mistakes
- Improve documentation (less is needed) and understandability (patterns are described well once)

Marty Stepp, 2007.

Benefits of using patterns

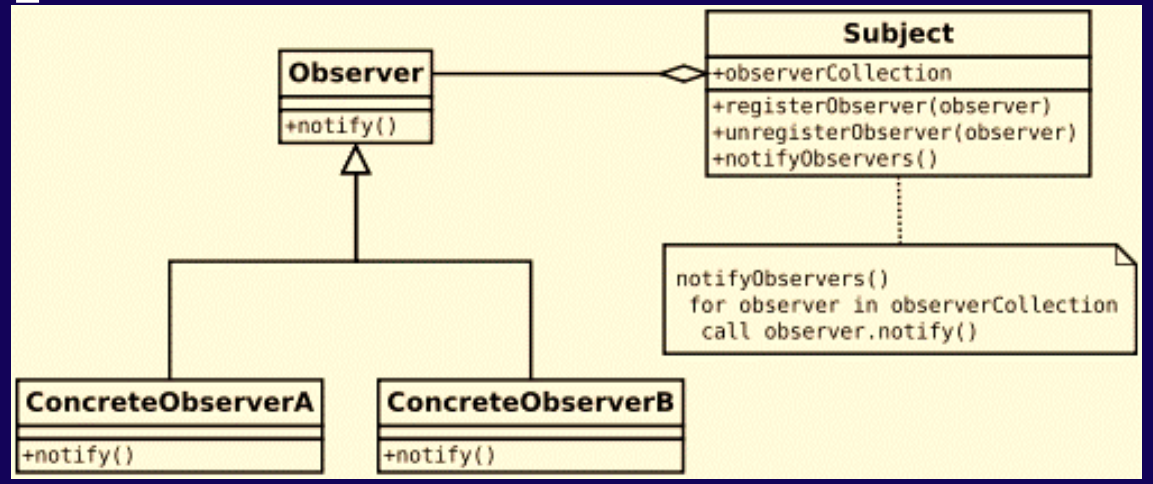
- Naming a recurring design structure
- Specifying design structure explicitly by identifying key class/object
  - ✓ Roles and relationships
  - ✓ Dependencies
  - ✓ Interactions
  - ✓ Coventions



Marty Stepp, 2007.

Benefits of using patterns

- Naming a recurring design structure
- Specifying design structure explicitly by identifying key class/object
  - ✓ Roles and relationships
  - ✓ Dependencies
  - ✓ Interactions
  - ✓ Coventions



Marty Stepp, 2007.

Benefits of using patterns

- Abstracting from concrete design elements
  - ✓ Problem domain
  - ✓ Form factor
  - ✓ Vendor (
- Distilling and coding knowledge

Marty Stepp, 2007.

Gang of Four patterns

There are three basic kinds of design patterns:

- Creational
- Structural
- Behavioral

Creational Patterns

- Creational design patterns separate the object creation logic from the rest of the system
- Deal with initializing and configuring classes and objects
- Instead of you creating objects, creational patterns create them for you

Creational Patterns

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

Structural Patterns

- Sometimes you need to build larger structures by using an existing set of classes
- Structural class patterns use inheritance to build a new structure
- Structural object patterns use composition / aggregation to obtain a new functionality

Structural Patterns

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral Patterns

- Behavioral patterns govern how objects communicate with each other
- Deal with dynamic interactions among societies of classes and objects
- How they distribute responsibility

Behavioral Patterns

- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor

Elements of Design Patterns

Design patterns have 4 essential elements

- *Pattern name*
  - ✓ increases vocabulary of designers
- *Problem*
  - ✓ intent, context, when to apply
- *Solution*
  - ✓ UML-like structure, abstract code
- *Consequences* (results and tradeoffs)