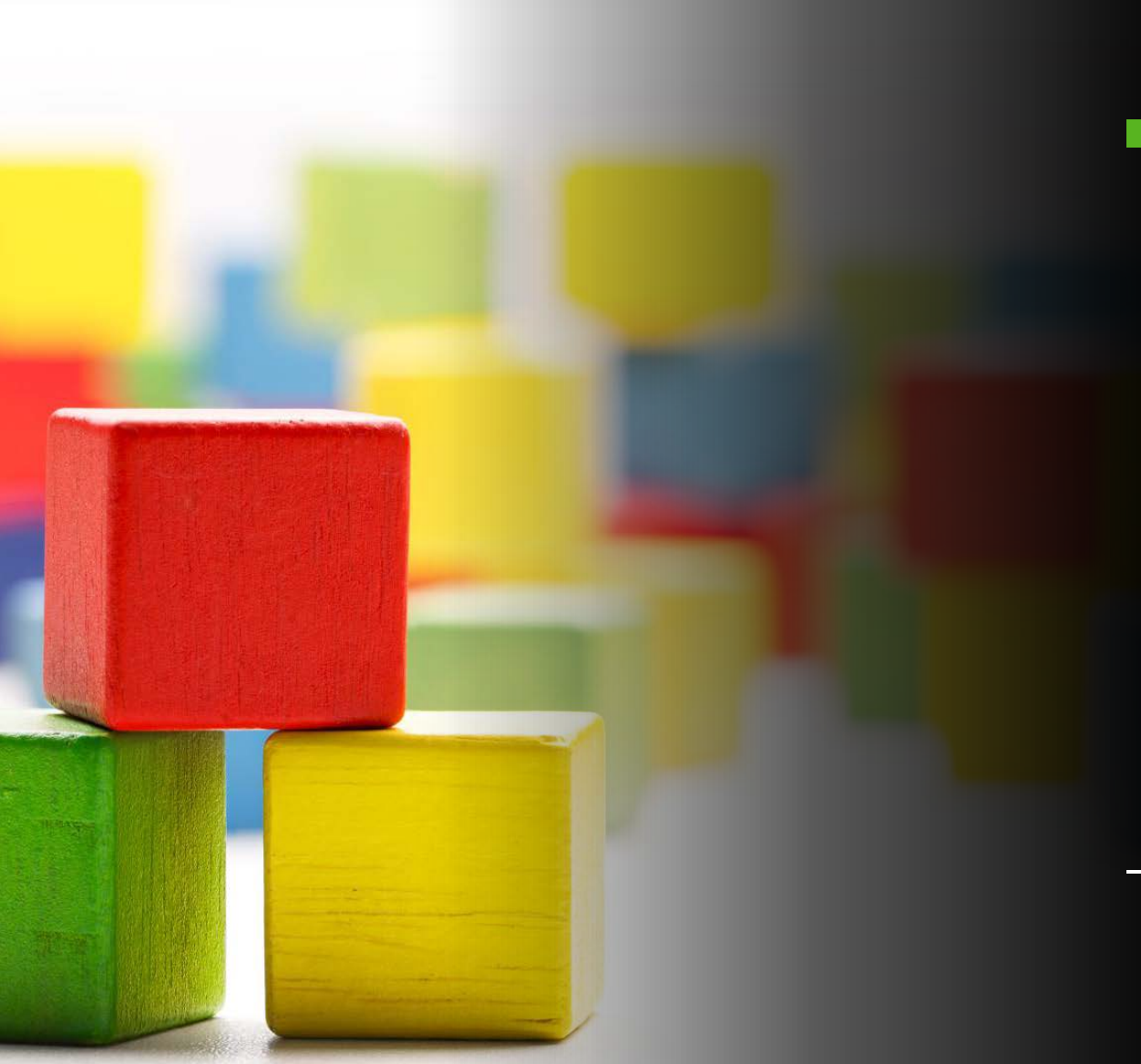


Software Engineering

Dr. Arijit Karati

Contents of the slides are prepared based on the materials from web and textbooks. It is stated that this material will be used to make the students aware of the topics and practiced for non-profit purposes.

(slide can be found in this secure domain)

Object-Oriented Software Design

Contents...

- Object-oriented concepts
- Object modelling using Unified Modelling Language (UML)
- Object-oriented software development and patterns
- CASE tools
- Summary

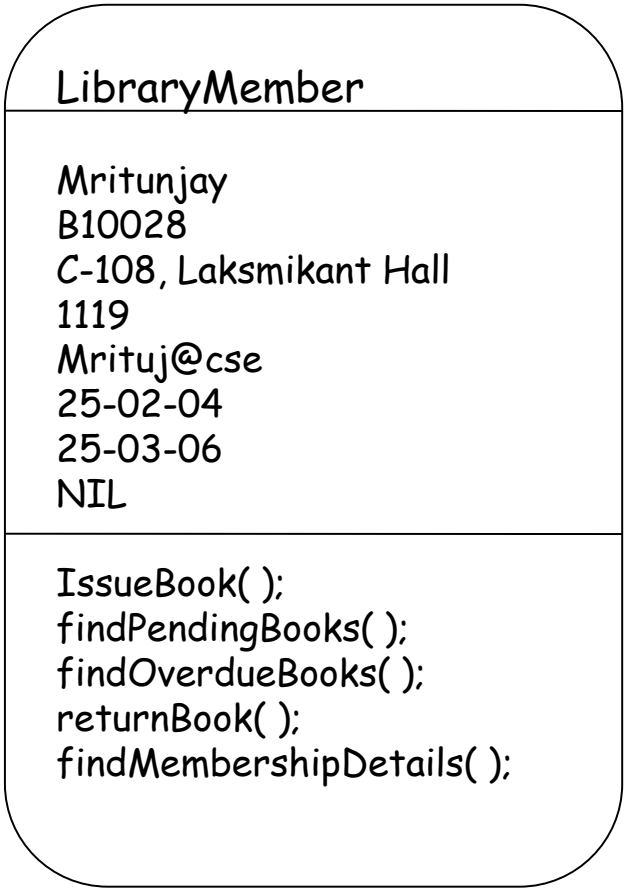
Class Diagram

- Describes static structure of a system
- Main constituents are classes and their relationships:
  - ✓ Generalization
  - ✓ Aggregation
  - ✓ Association
  - ✓ Various kinds of dependencies

Class Diagram

- Entities with common features, i.e. attributes and operations
- Classes are represented as solid outline rectangle with compartments
- Compartments for name, attributes, and operations.
- Attribute and operation compartments are optional depending on the purpose of a diagram.

Object Diagram



Different representations of the LibraryMember object

Interaction Diagram

- Models how groups of objects collaborate to realize some behaviour
- Typically each interaction diagram realizes behaviour of a single use case
- Two kinds: Sequence and Collaboration diagrams.
- Two diagrams are equivalent
  - ✓ Portray different perspectives
- These diagrams play a very important role in the design process.

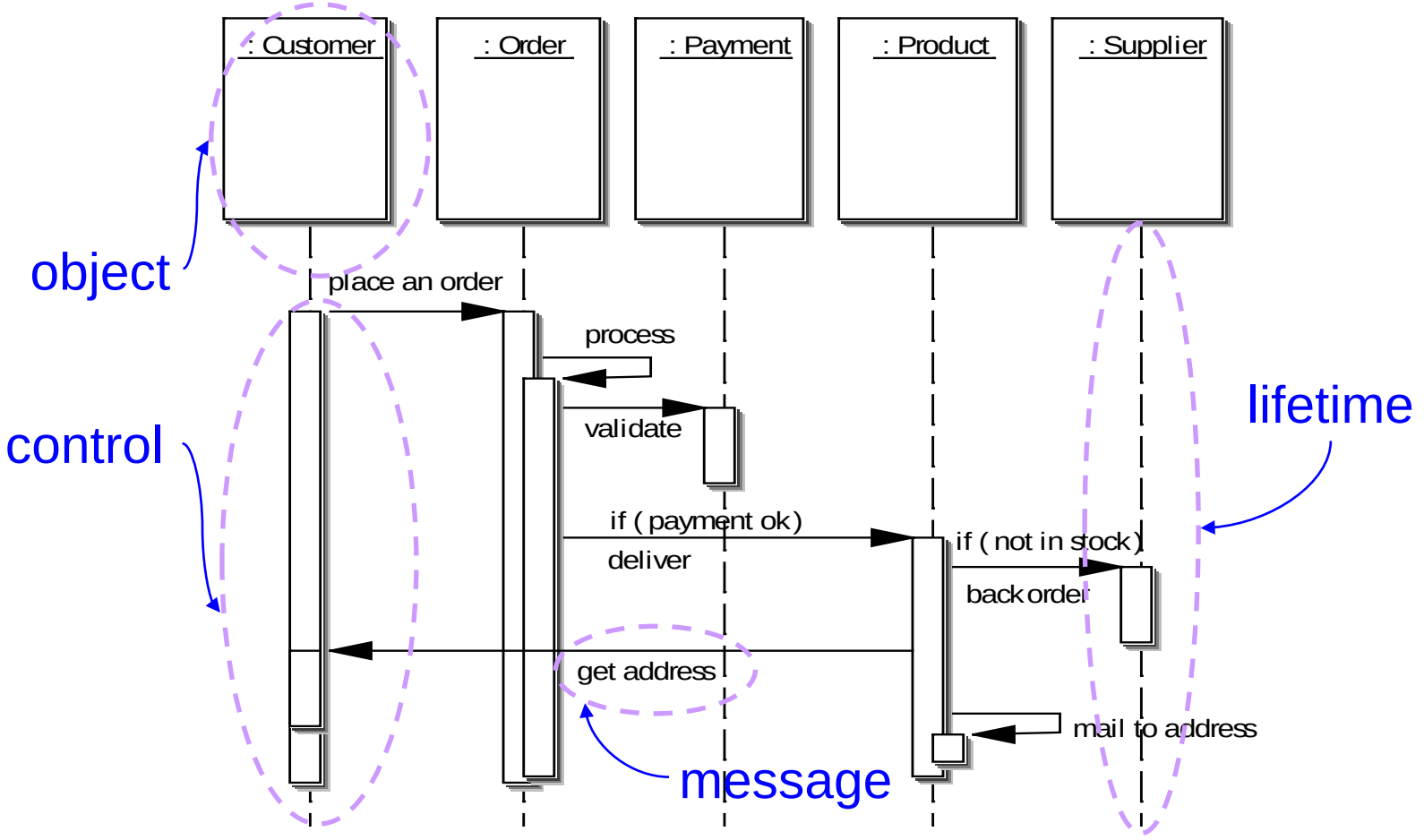
Sequence Diagram

- Shows interaction among objects as a two-dimensional chart
- Objects are shown as boxes at top
- If object created during execution then shown at appropriate place
- Objects existence are shown as dashed lines (lifeline)
- Objects activeness, shown as a rectangle on lifeline

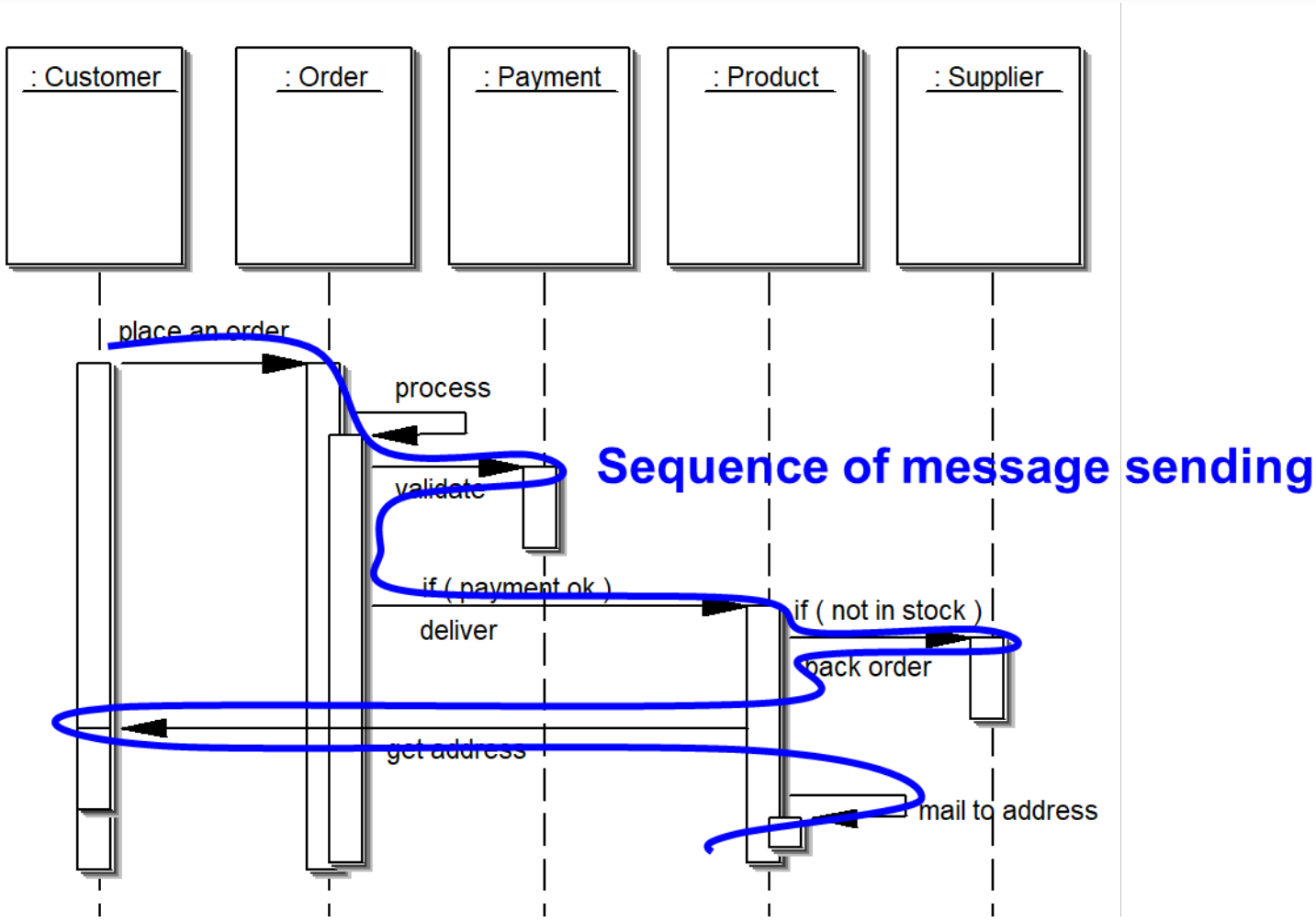
Sequence Diagram

- Messages are shown as arrows
- Each message labelled with corresponding message name
- Each message can be labelled with some control information
- Two types of control information
  - ✓ condition ([])
  - ✓ iteration (*)

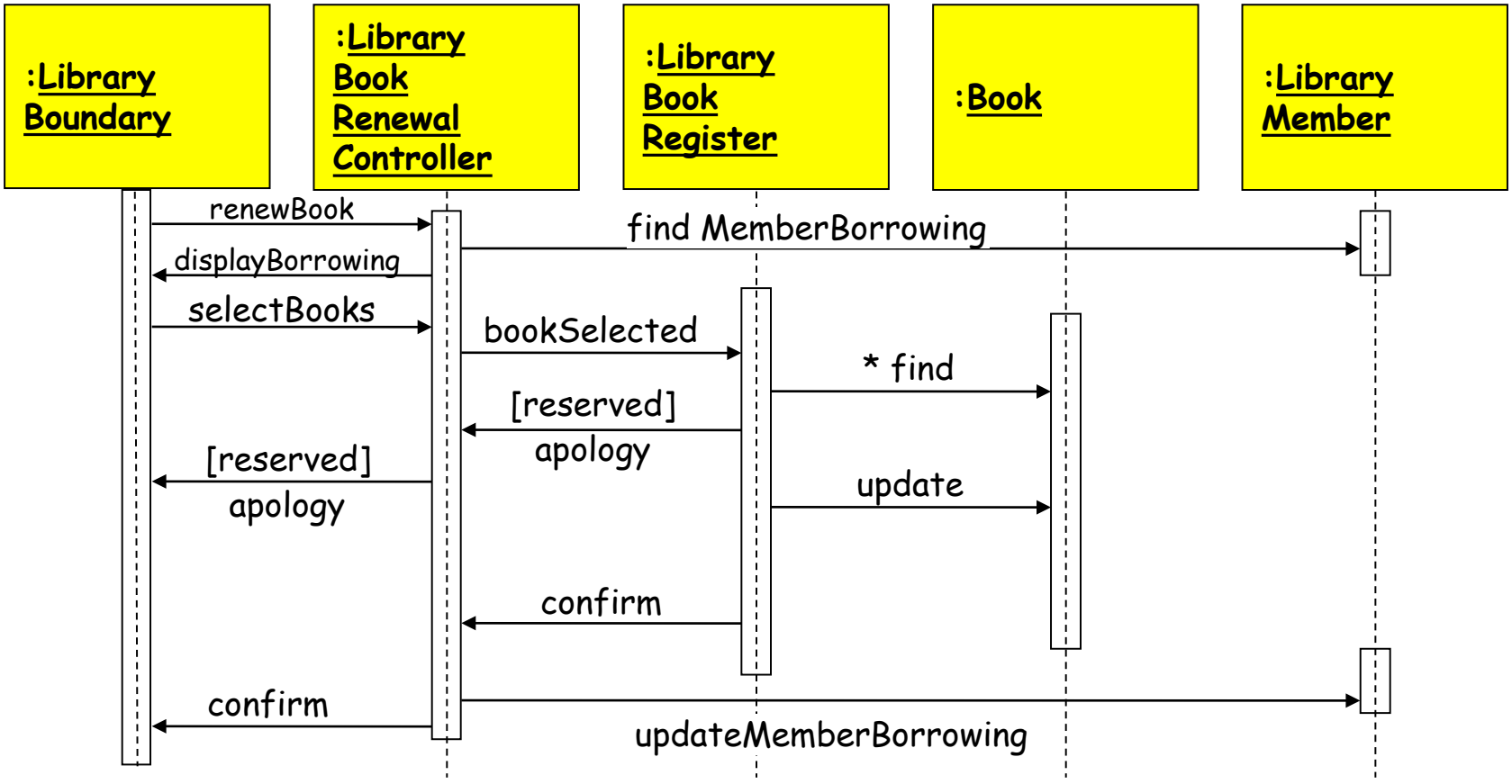
Elements of a Sequence Diagram



Example


: Customer
: Order
: Payment
: Product
: Supplier
place an order
process
validate
Sequence of message sending
if (payment ok)
deliver
if (not in stock)
back order
get address
mail to address

An Example of A Sequence Diagram

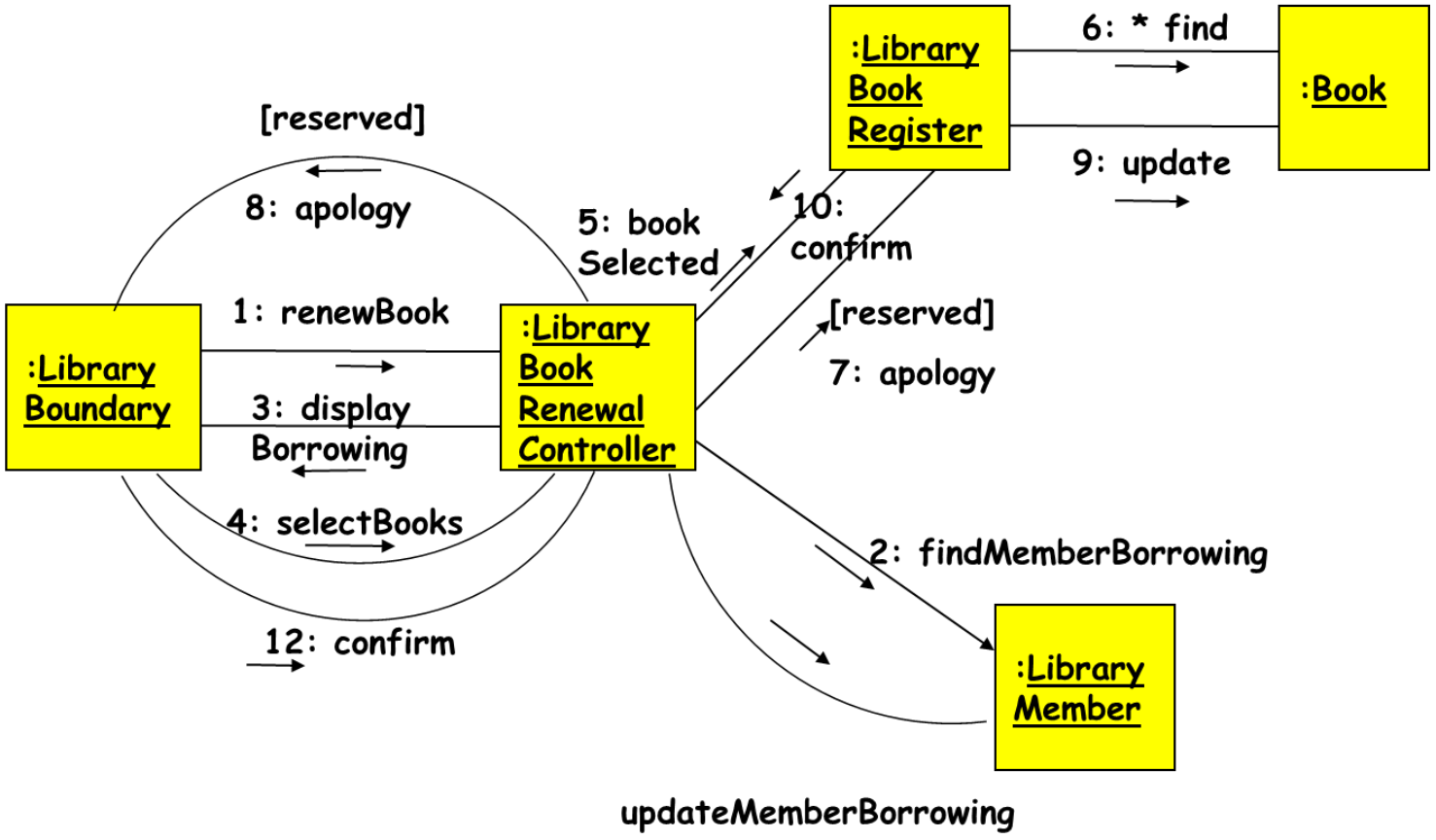


Sequence Diagram for the renew book use case

Collaboration Diagram

- Shows both structural and behavioural aspects
- Objects are collaborator, shown as boxes
- Messages between objects shown as a solid line
- A message is shown as a labelled arrow placed near the link
- Messages are prefixed with sequence numbers to show relative sequencing

An Example of A Collaboration Diagram



Collaboration Diagram for the renew book use case

Activity Diagram

- Not present in earlier modelling techniques:
  - ✓ Possibly based on event diagram of Odell [1992]
- Represents processing activity, may not correspond to methods
- Activity is a state with an internal action and one/many outgoing transitions
- Somewhat related to flowcharts

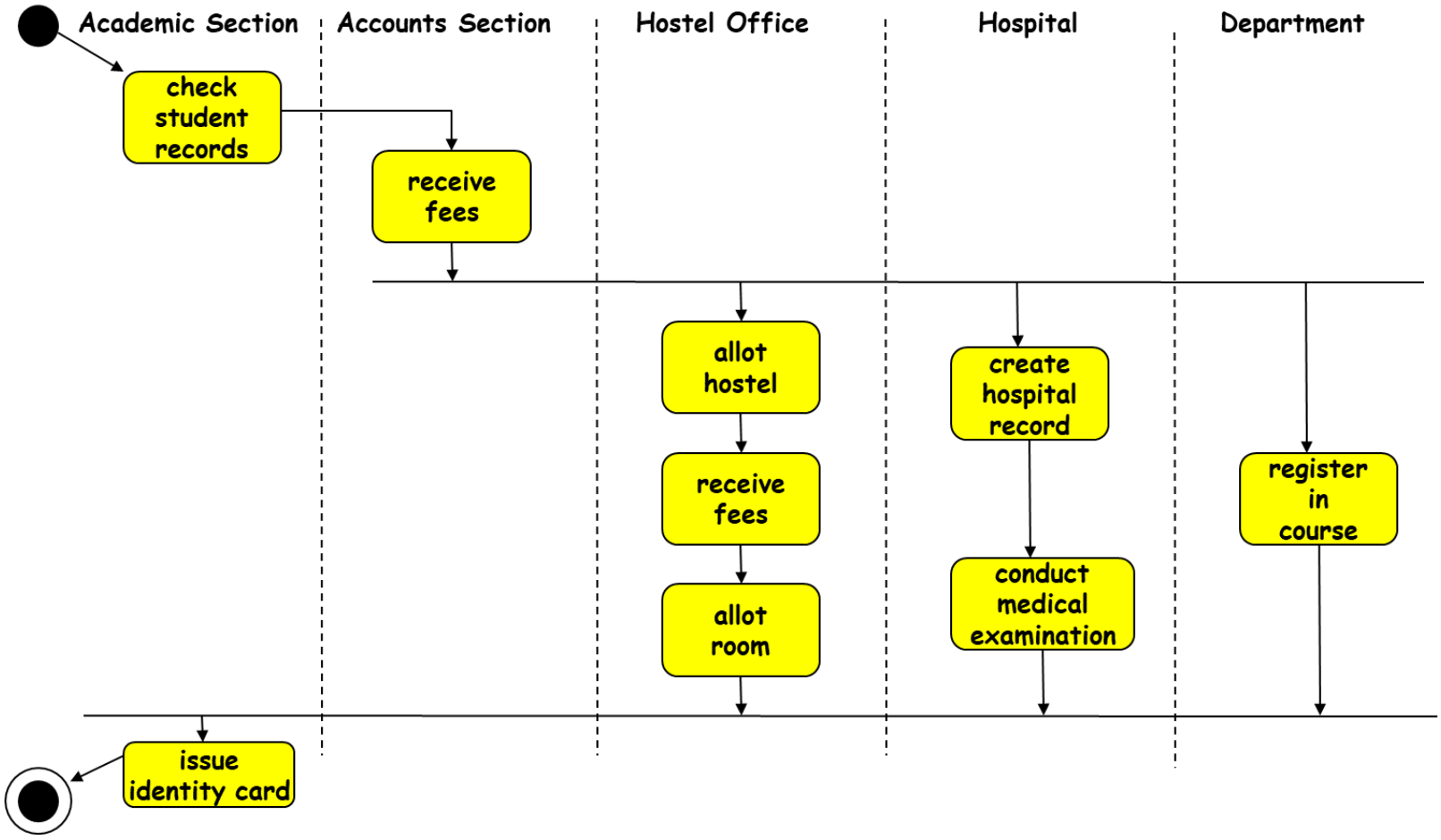
Activity Diagram vs Flow Chart

- Can represent parallel activity and synchronization aspects
- Swim lanes can be used to group activities based on who is performing them
- Example: academic department vs. hostel

Activity Diagram

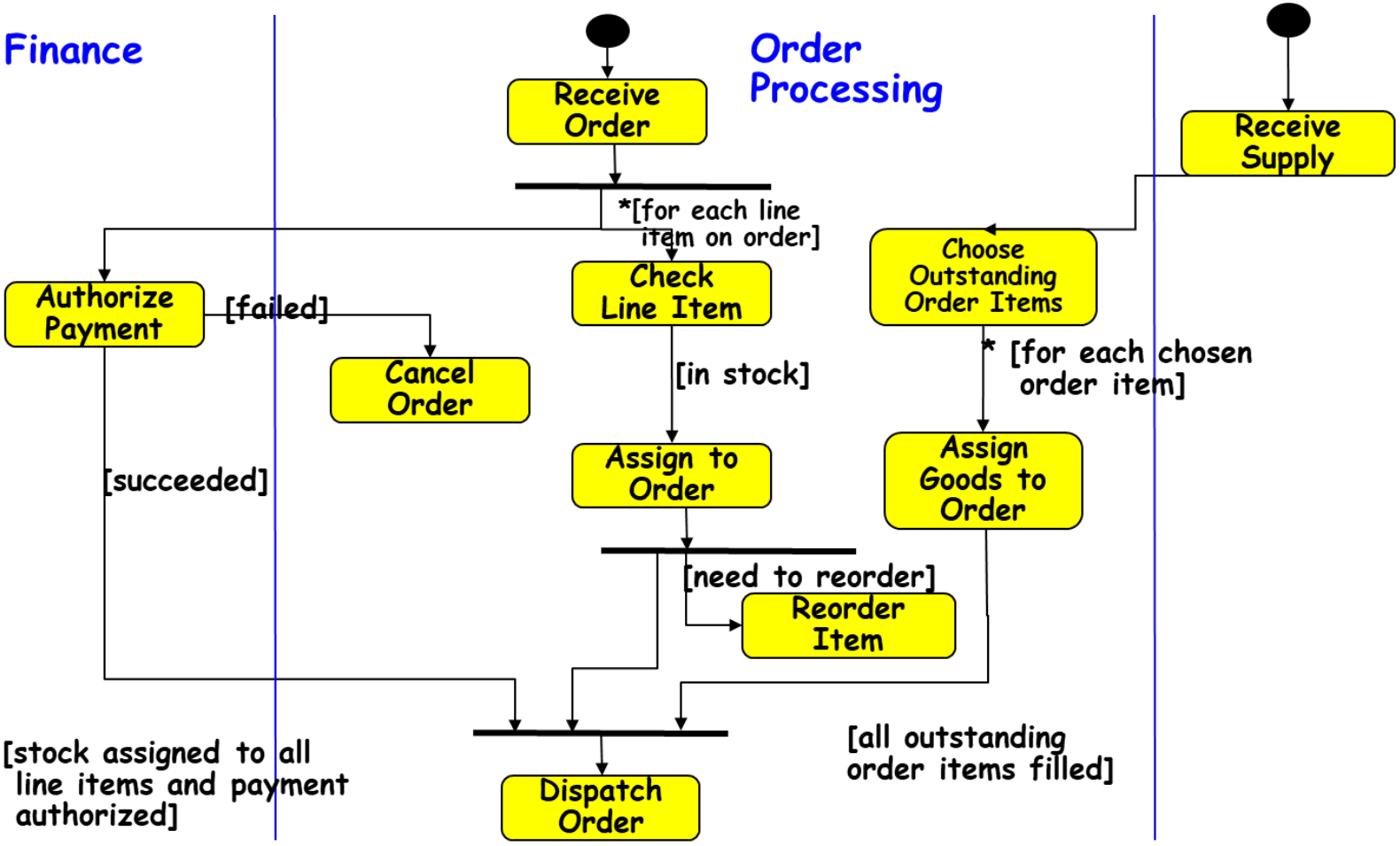
- Normally employed in business process modelling.
- Carried out during requirements analysis and specification stage.
- Can be used to develop interaction diagrams.

An Example of An Activity Diagram



Activity diagram for student admission procedure at IIT

Activity Diagram: Example 2



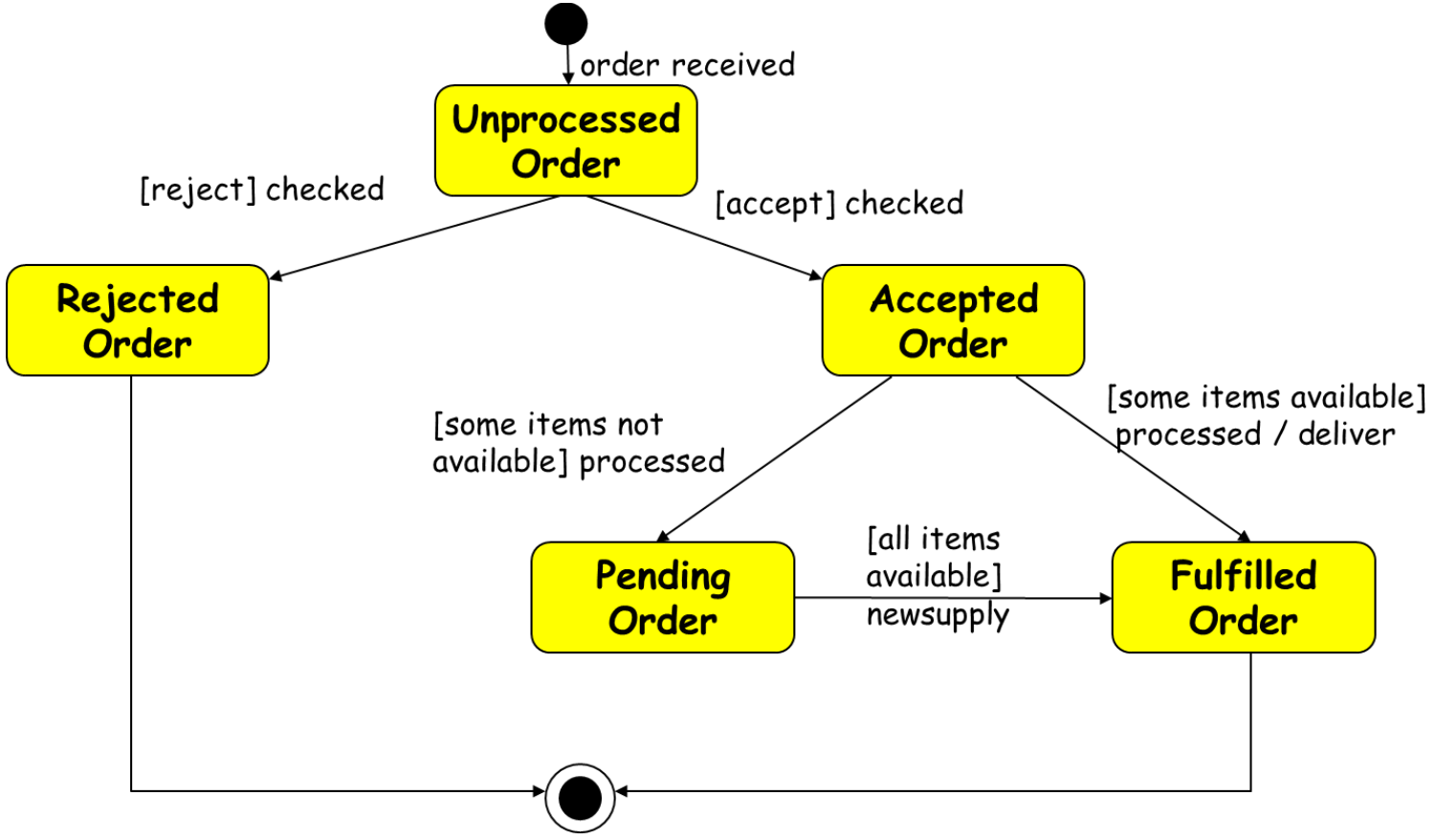
State Chart Diagram

- Based on the work of David Harel [1990]
- Model how the state of an object changes in its lifetime
- Based on finite state machine (FSM) formalism
- State chart avoids the problem of state explosion of FSM.
- Hierarchical model of a system:
  - ✓ Represents composite nested states

State Chart Diagram

- Elements of state chart diagram
  - ✓ Initial State: A filled circle
  - ✓ Final State: A filled circle inside a larger circle
  - ✓ State: Rectangle with rounded corners
  - ✓ Transitions: Arrow between states, also boolean logic condition (guard)

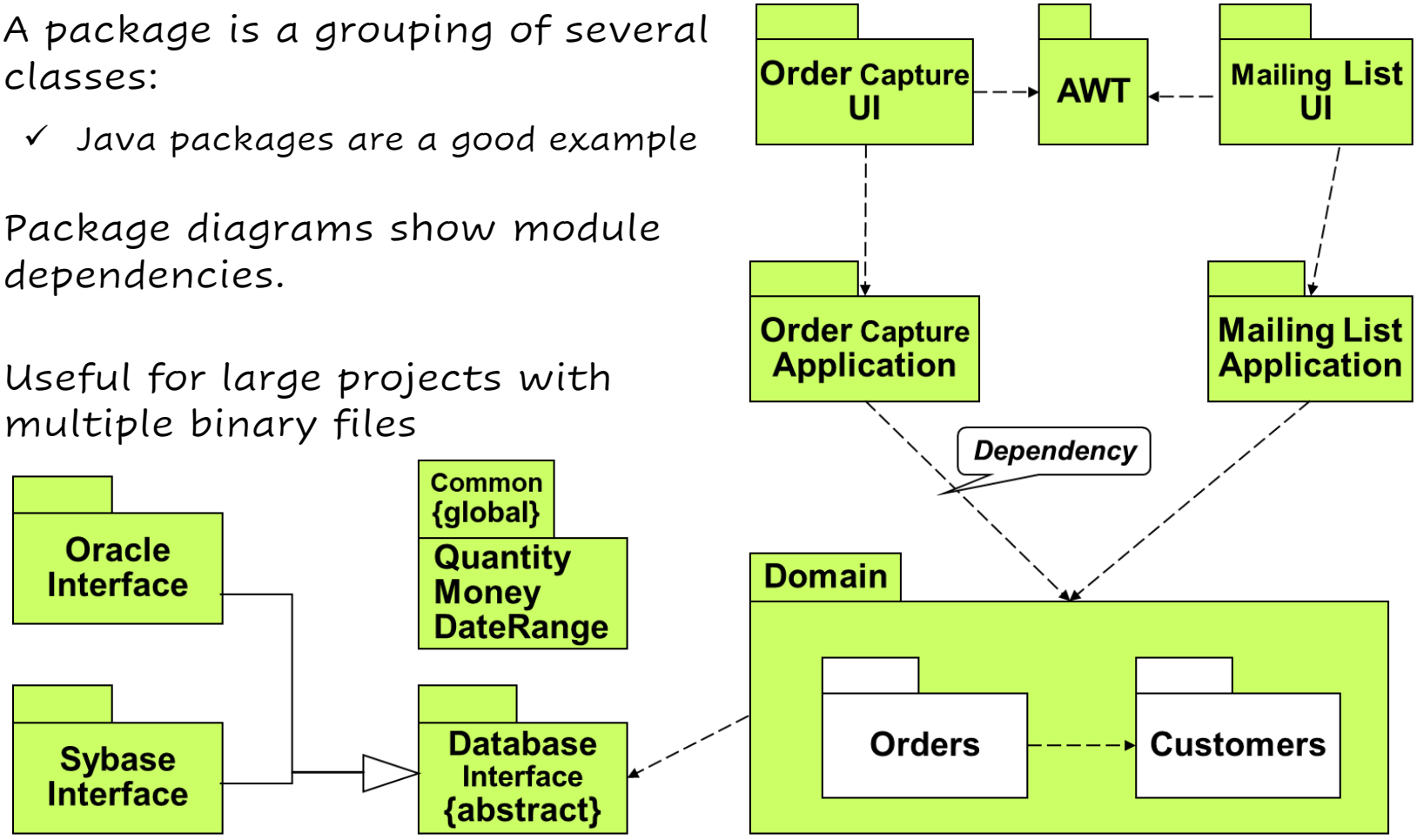
An Example of A State Chart Diagram



Example: State chart diagram for an order object

Package Diagrams

- A package is a grouping of several classes:
  - ✓ Java packages are a good example
- Package diagrams show module dependencies.
- Useful for large projects with multiple binary files

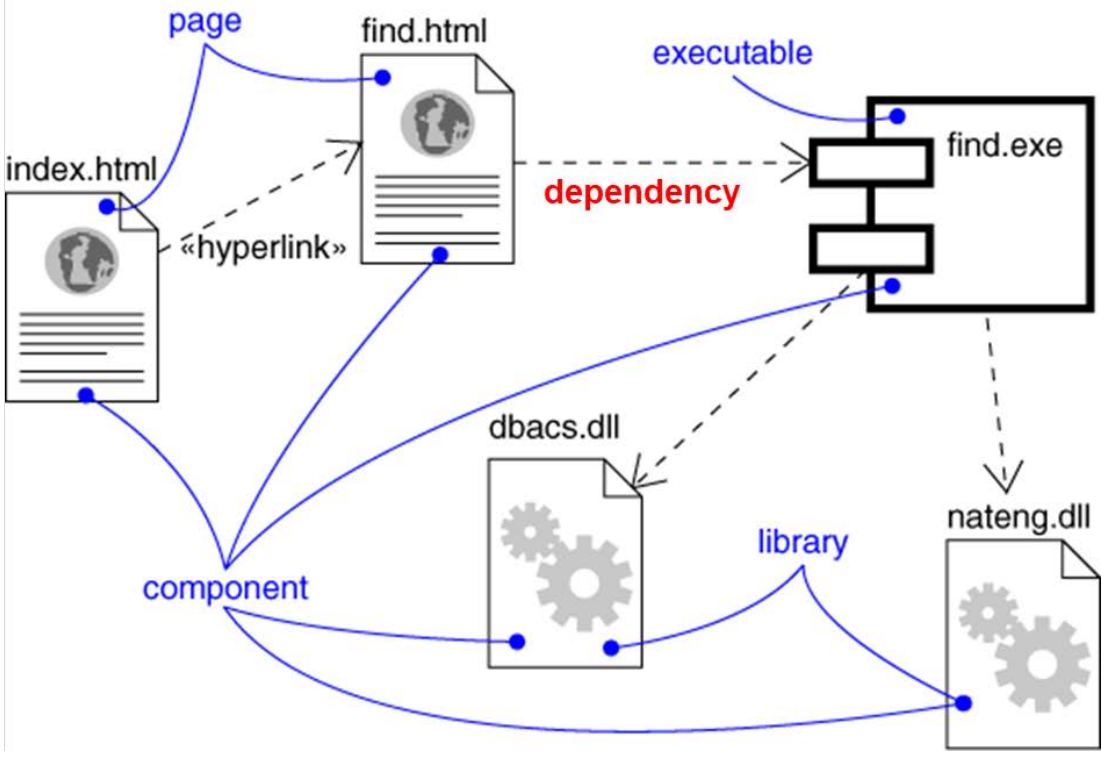


Component Diagram

- Captures the physical structure of the implementation (code components)

Components:

- **Executables**
- **Library**
- **Table**
- **File**
- **Document**

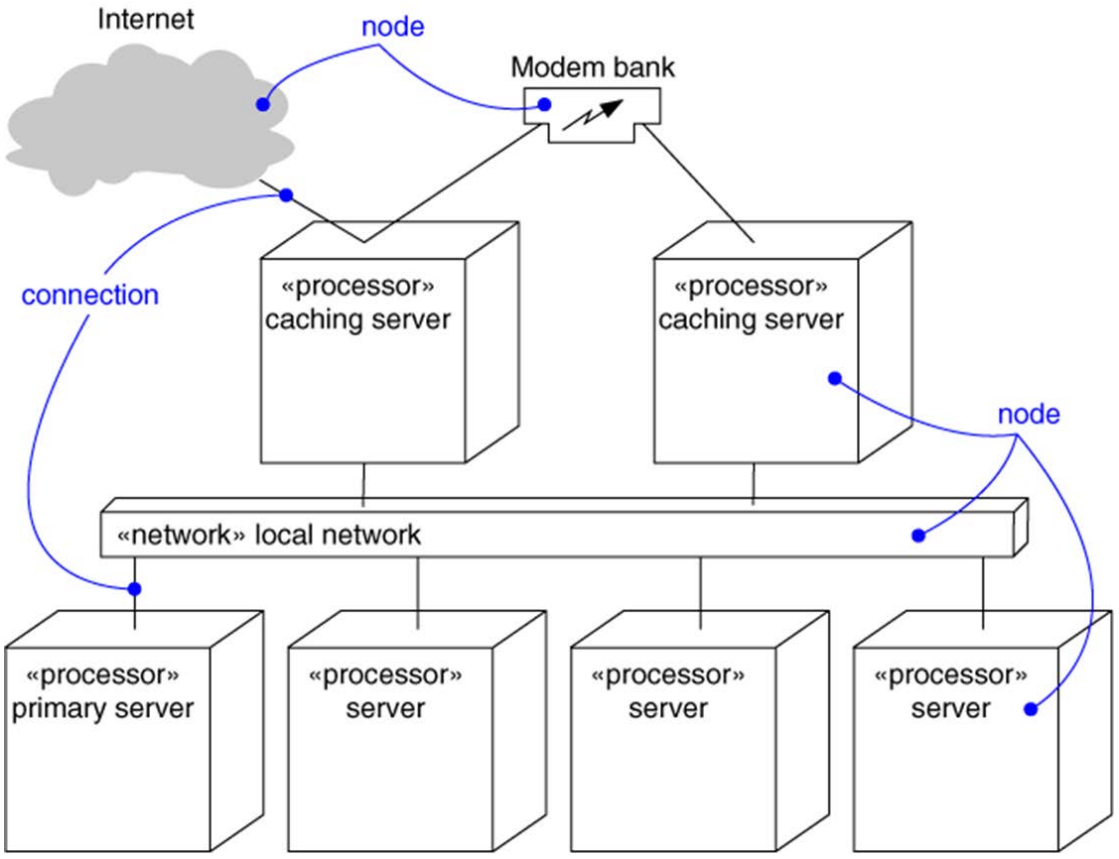


Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - ✓ Organize source code
  - ✓ Construct an executable release
  - ✓ Specify a physical database
- Developed by architects and programmers

Deployment Diagram

- Captures the topology of a system's hardware

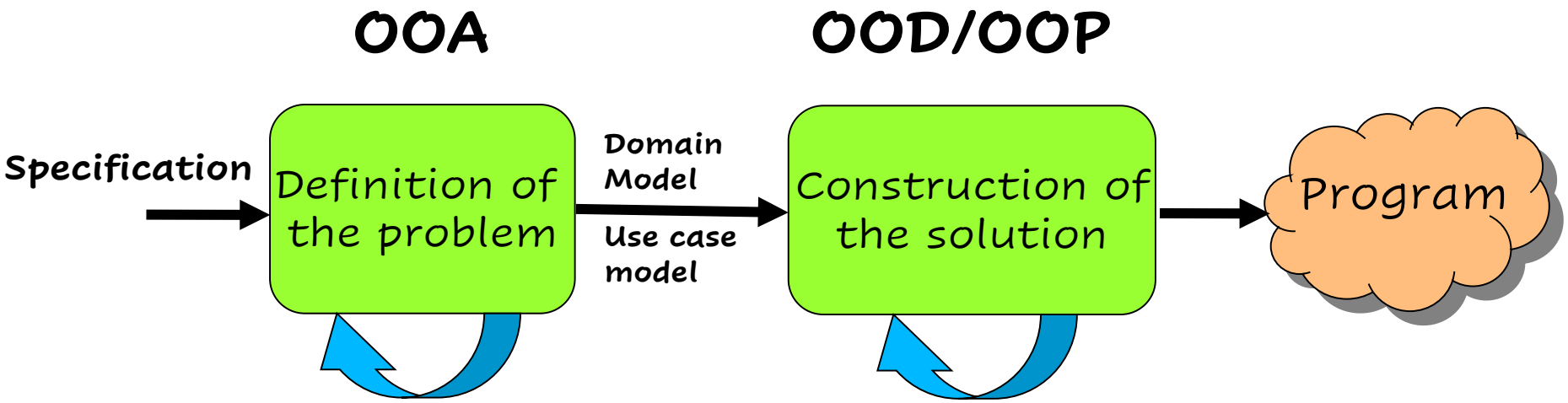


A Design Process

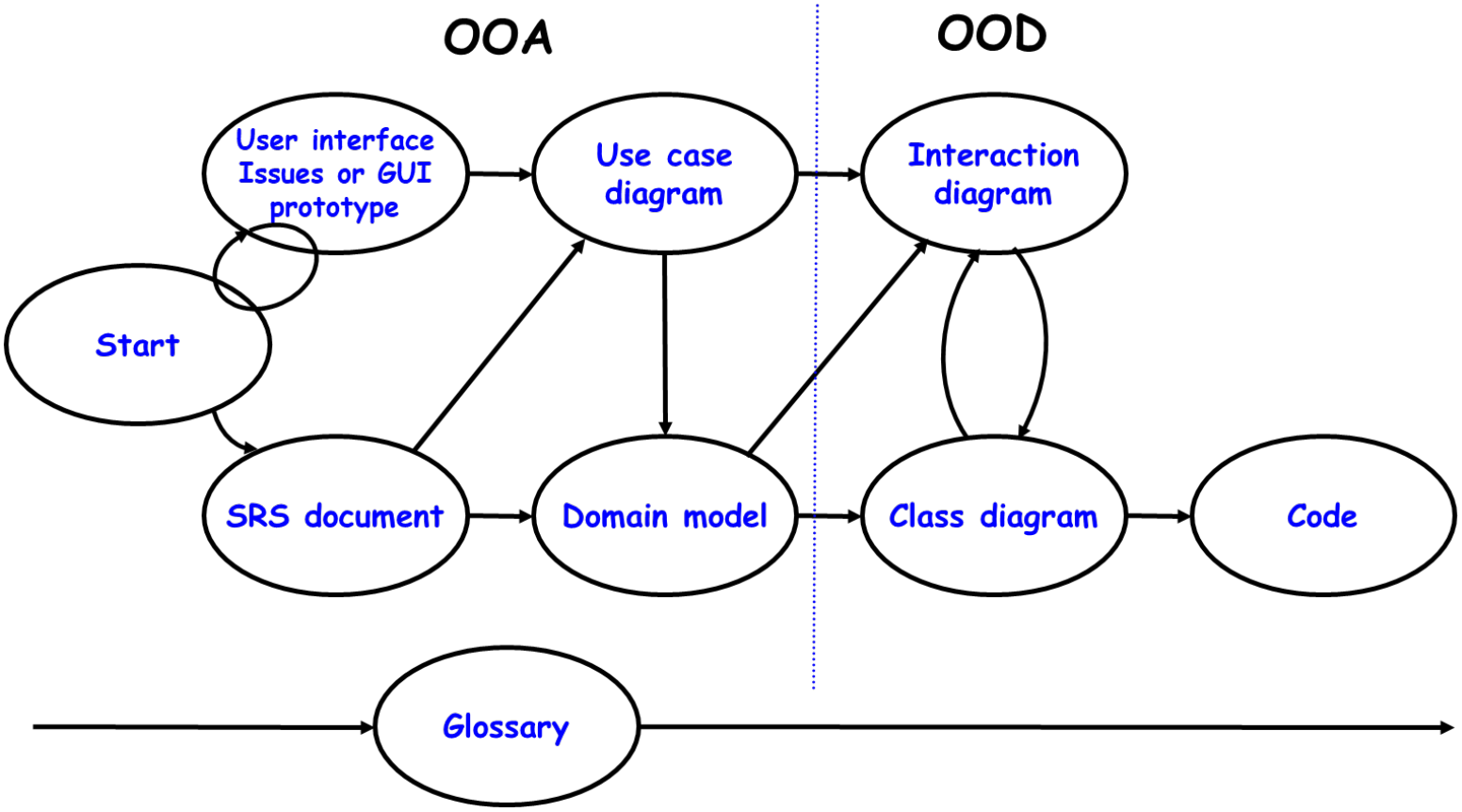
- Developed from various methodologies.
  - ✓ However, UML has been designed to be usable with any design methodology.
- From requirements specification, initial model is developed (OOA)
  - ✓ Analysis model is iteratively refined into a design model
- Design model is implemented using OO concepts

OOAD

Iterative and Incremental



Unified Development Process cont...



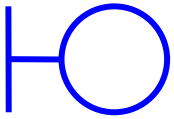
Domain Modelling

- Represents concepts or objects appearing in the problem domain.
- Also captures relationships among objects.
- Three types of objects are identified
  - ✓ Boundary objects
  - ✓ Entity objects
  - ✓ Controller objects

Class Stereotypes

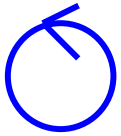
- Three different stereotypes on classes are used: <<boundary>>, <<control>>, <<entity>>.

Boundary



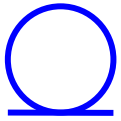
Cashier Interface

Control



Withdrawal

Entity



Account

Boundary Objects

- Interact with actors:
  - ✓ User interface objects
- Include screens, menus, forms, dialogs etc.
- Do not perform processing but validates, formats etc.

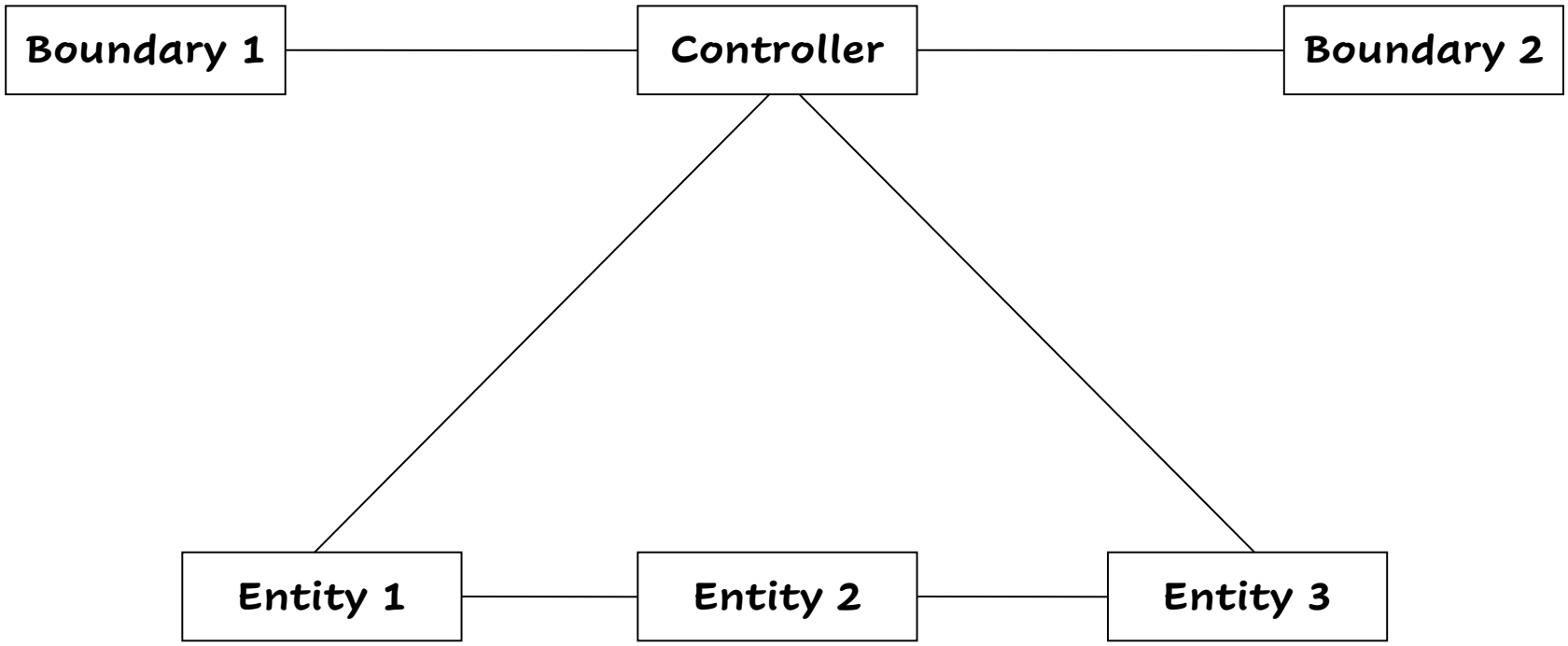
Entity Objects

- Hold information:
  - ✓ Such as data tables & files, e.g. Book, BookRegister
- Normally are dumb servers
- Responsible for storing data, fetching data etc.
- Elementary operations on data such as searching, sorting, etc.
- Entity Objects are identified by examining nouns in problem description

Controller Objects

- Coordinate the activities of a set of entity objects
- Interface with the boundary objects
- Realizes use case behaviour
- Embody most of the logic involved with the use case realization
- There can be more than one controller to realize a single use case

Use Case Realization



Realization of use case through the collaboration of Boundary, controller and entity objects

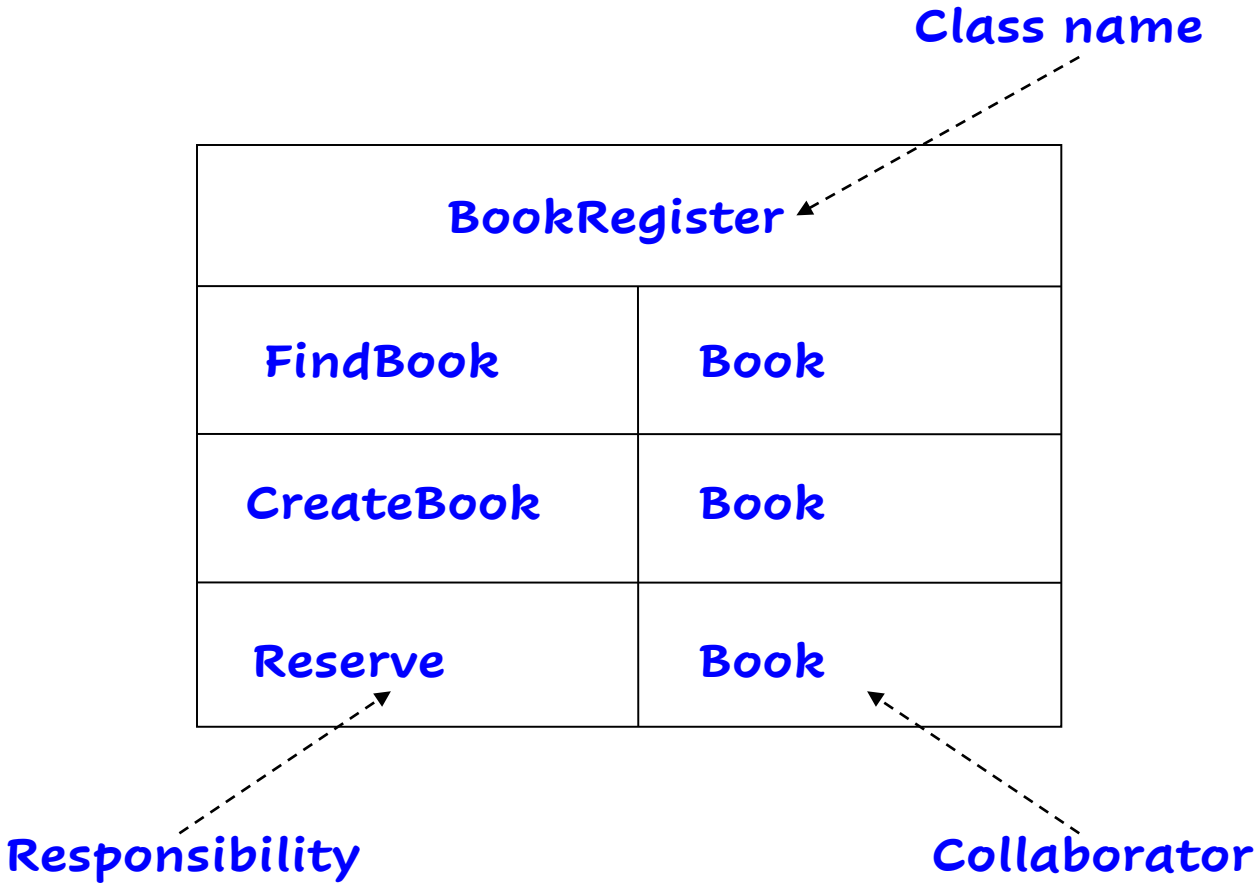
Class-Responsibility-Collaborator(CRC) Cards

- Pioneered by Ward Cunningham and Kent Beck
- Index cards prepared one each per class
- Class responsibility is written on these cards
- Collaborating object is also written

Class-Responsibility-Collaborator(CRC) Cards

- Required for developing interaction diagram of complex use cases
- Team members participate to determine:
  - ✓ The responsibility of classes involved in the use case realization

An Example of A CRC Card



CRC card for the BookRegister class

Patterns versus Idioms

- A pattern:
  - ✓ Describes a recurring problem
  - ✓ Describes the core of a solution
  - ✓ Is capable of generating many distinct designs

- An Idiom is more restricted
  - ✓ Still describes a recurring problem
  - ✓ Provides a more specific solution, with fewer variations
  - ✓ Applies only to a narrow context
  - ✓ e.g., the C++ language

Patterns

- The essential idea:
  - ✓ If you can master a few important patterns, you can easily spot them in application development and use the pattern solutions.

Idioms

- In English:
  - ✓ A group of words that has meaning different from a simple juxtaposition of the meanings of the individual words.
  - ✓ "Raining cats and dogs"
- A C idiom:
  - ✓ for(i=0;i<1000;i++){
  - ✓ }

Antipattern

- If a pattern represents a best practice:
  - ✓ Antipattern represents lessons learned from a bad design.
- Antipatterns help to recognise deceptive solutions:
  - ✓ That appear attractive at first, but turn out to be a liability later.

Design Patterns

- Standard solutions to commonly recurring problems
- Provides good solution based on common sense
- Pattern has four important parts
  - ✓ The problem
  - ✓ The context
  - ✓ The solution
  - ✓ The context in which it works or does not work

Example Pattern: Expert

- Problem: Which class should be responsible for doing certain things
- Solution: Assign responsibility to the class that has the information necessary to fulfil the required responsibility

Example Pattern: Expert cont...

Class Diagram

Collaboration Diagram

Example Pattern: Creator

- Problem: Which class should be responsible for creating a new instance of some class?
- Solution: Assign a class C1 the responsibility to create class C2 if
  - ✓ C1 is an aggregation of objects of type C2
  - ✓ C1 contains object of type C2

Example Pattern: Controller

- Problem: Who should be responsible for handling the actor requests?
- Solution: Separate controller object for each use case.

Example Pattern: Facade

- Problem: How should the services be requested from a service package?
- Context (problem): A package (cohesive set of classes), example: RDBMS interface package
- Solution: A class (DBfacade) can be created which provides a common interface to the services of the package

Example Pattern: MVC

- Model-View-Controller
- How should the user interface (Boundary) objects interact with the other objects?
- Solution 1: Pull from Above
  - ✓ Boundary object invokes other objects.
  - ✓ Does not work when data needs to be asynchronously displayed, simulation experiment, stock market alert, network monitor, etc.

Example Pattern: MVC

- Solution 2: Publish-Subscribe
  - ✓ The boundary objects register themselves with an event manager object.
  - ✓ Other objects, notify the event manager object as and when an event of interest occurs.
  - ✓ The event manager notifies those boundary objects that have registered with it by using a call back.

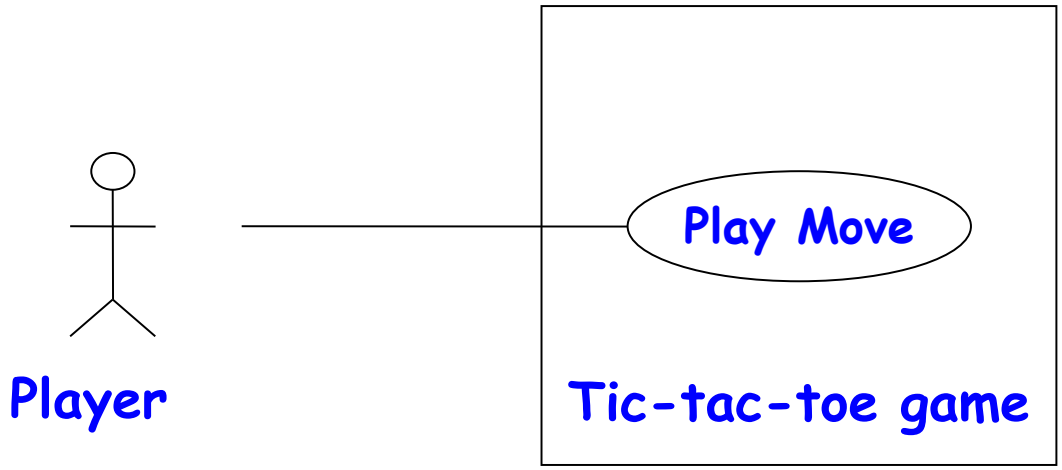
Example 1: Tic-Tac-Toe Computer Game

- A human player and the computer make alternate moves on a 3 3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.

Example 1: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
  - ✓ A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
  - ✓ And all the squares on the board are filled up,
  - ✓ Then the game is drawn.
- The computer always tries to win a game.

Example 1: Use Case Model

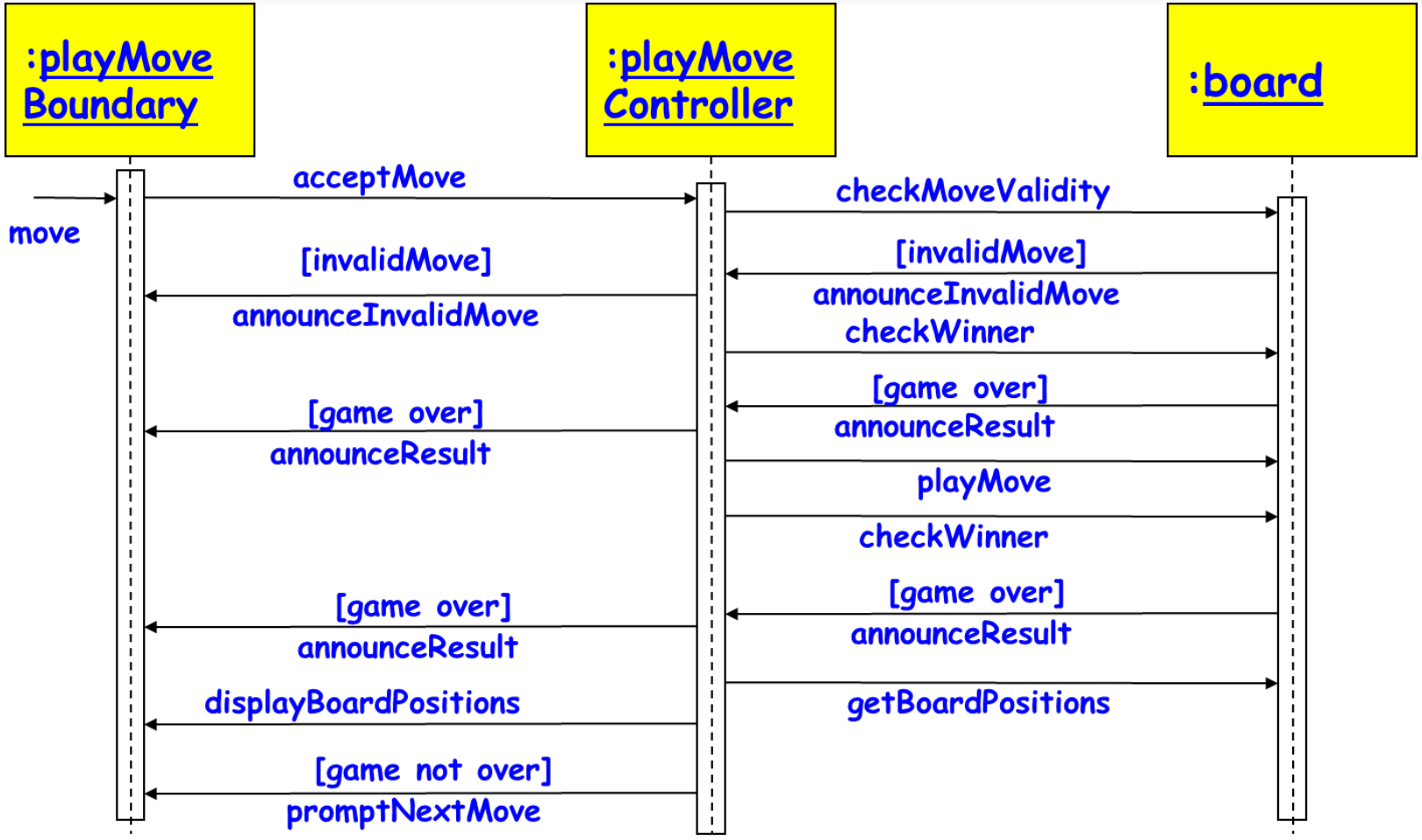


Example 1: Initial and Refined Domain Model

Initial domain model

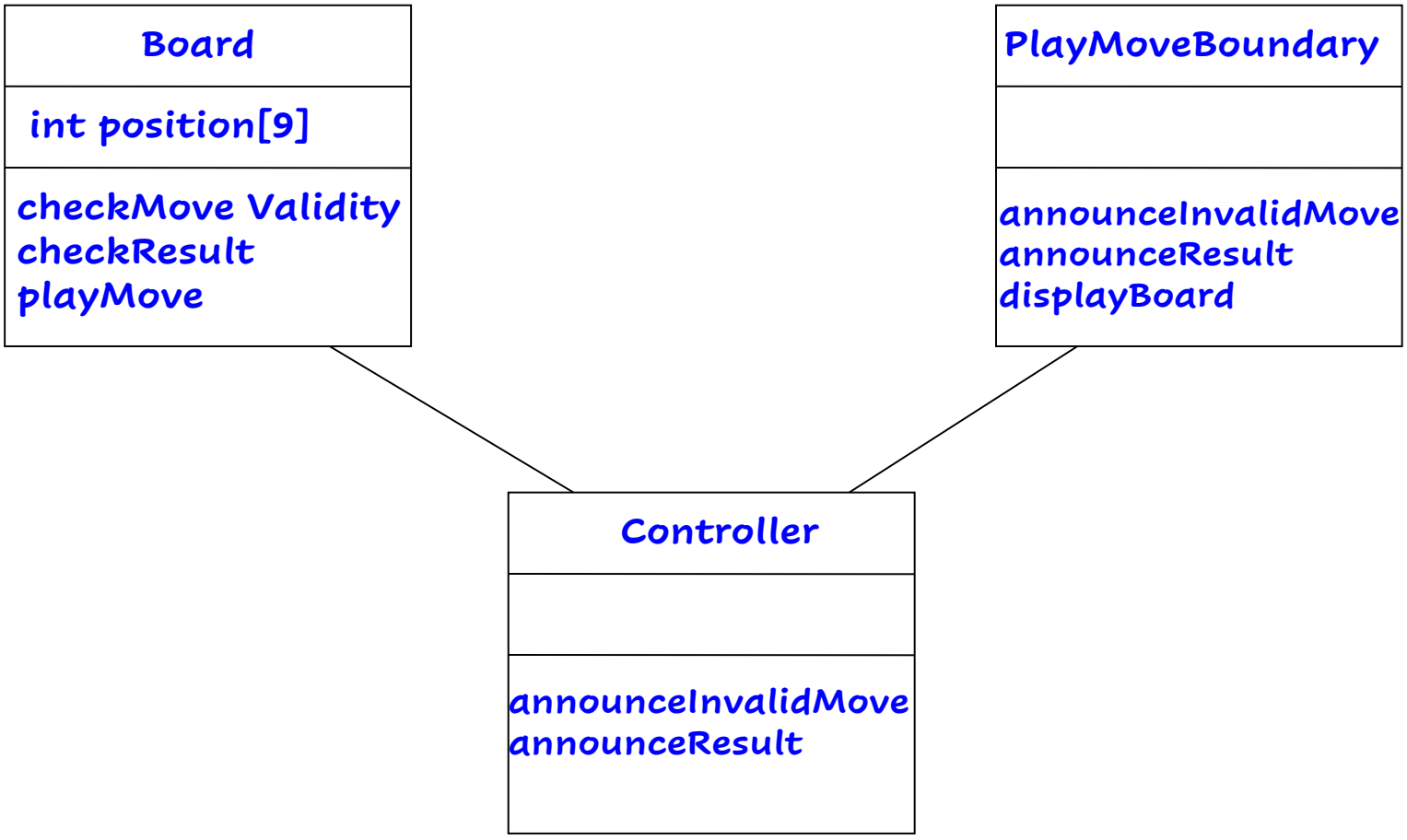
Refined domain model

Example 1: Sequence Diagram



Sequence Diagram for the play move use case

Example 1: Class Diagram



Example 2: Supermarket Prize Scheme

- Supermarket needs to develop software to encourage regular customers.
- Customer needs to supply his:
  - ✓ Residence address, telephone number, and the driving licence number.
- Each customer who registers is:
  - ✓ Assigned a unique customer number (CN) by the computer.

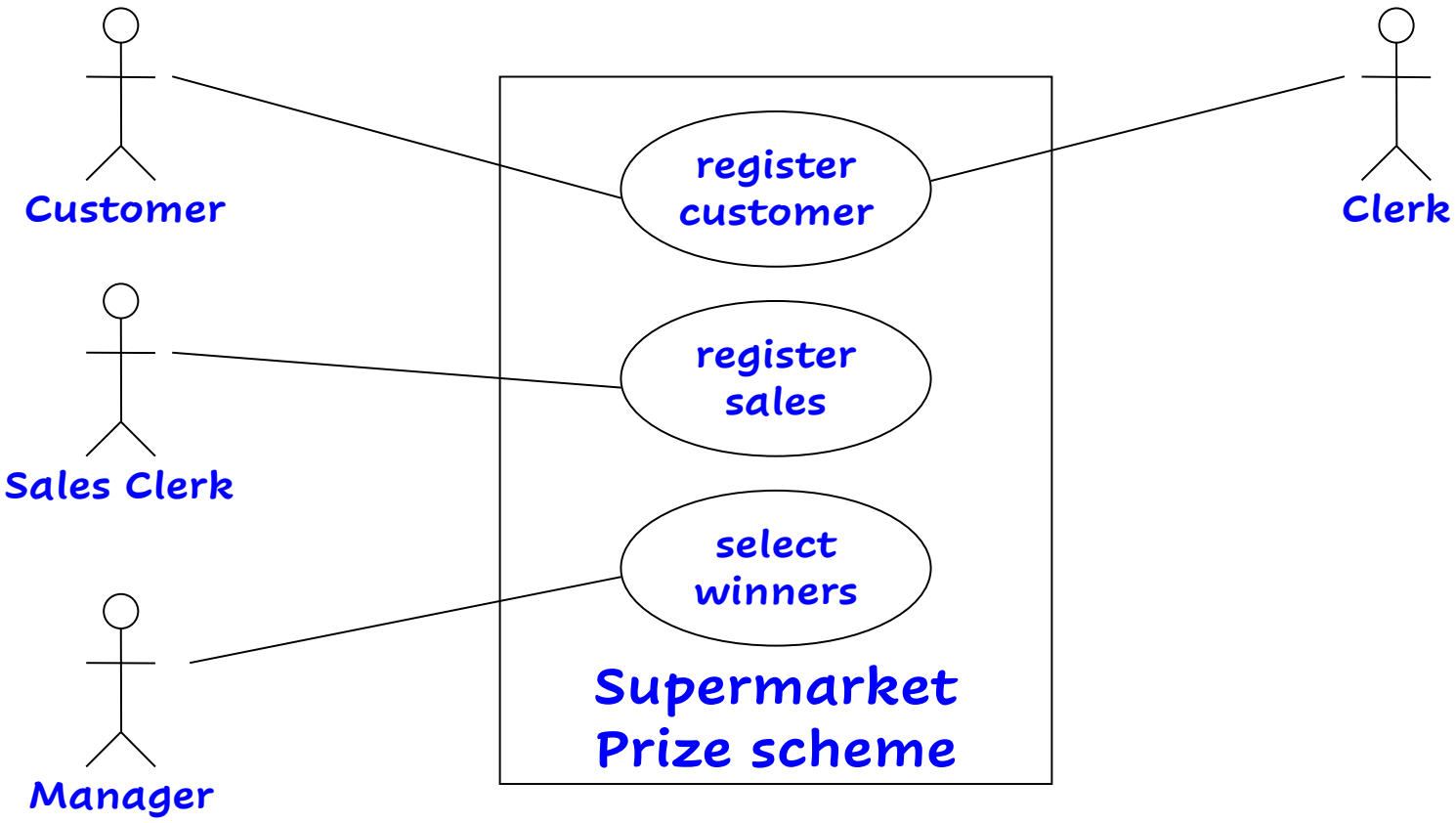
Example 2: Supermarket Prize Scheme

- A customer can present his CN to the staff when he makes any purchase.
- The value of his purchase is credited against his CN.
- At the end of each year:
  - ✓ The supermarket awards surprise gifts to ten customers who make highest purchase.

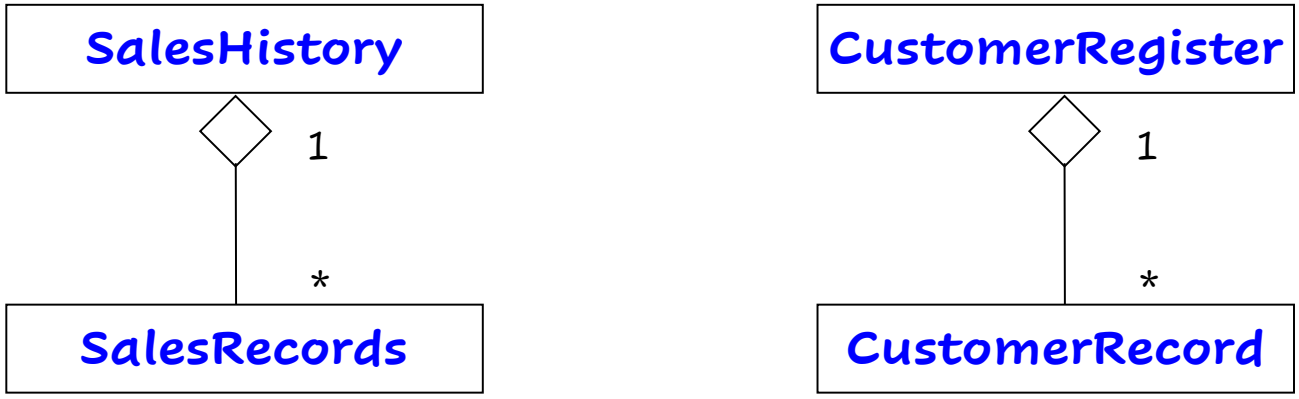
Example 2: Supermarket Prize Scheme

- Also, it awards a 22 carat gold coin to every customer:
  - ✓ Whose purchases exceed Rs. 10,000.
- The entries against the CN are reset:
  - ✓ On the last day of every year after the prize winner's lists are generated.

Example 2: Use Case Model

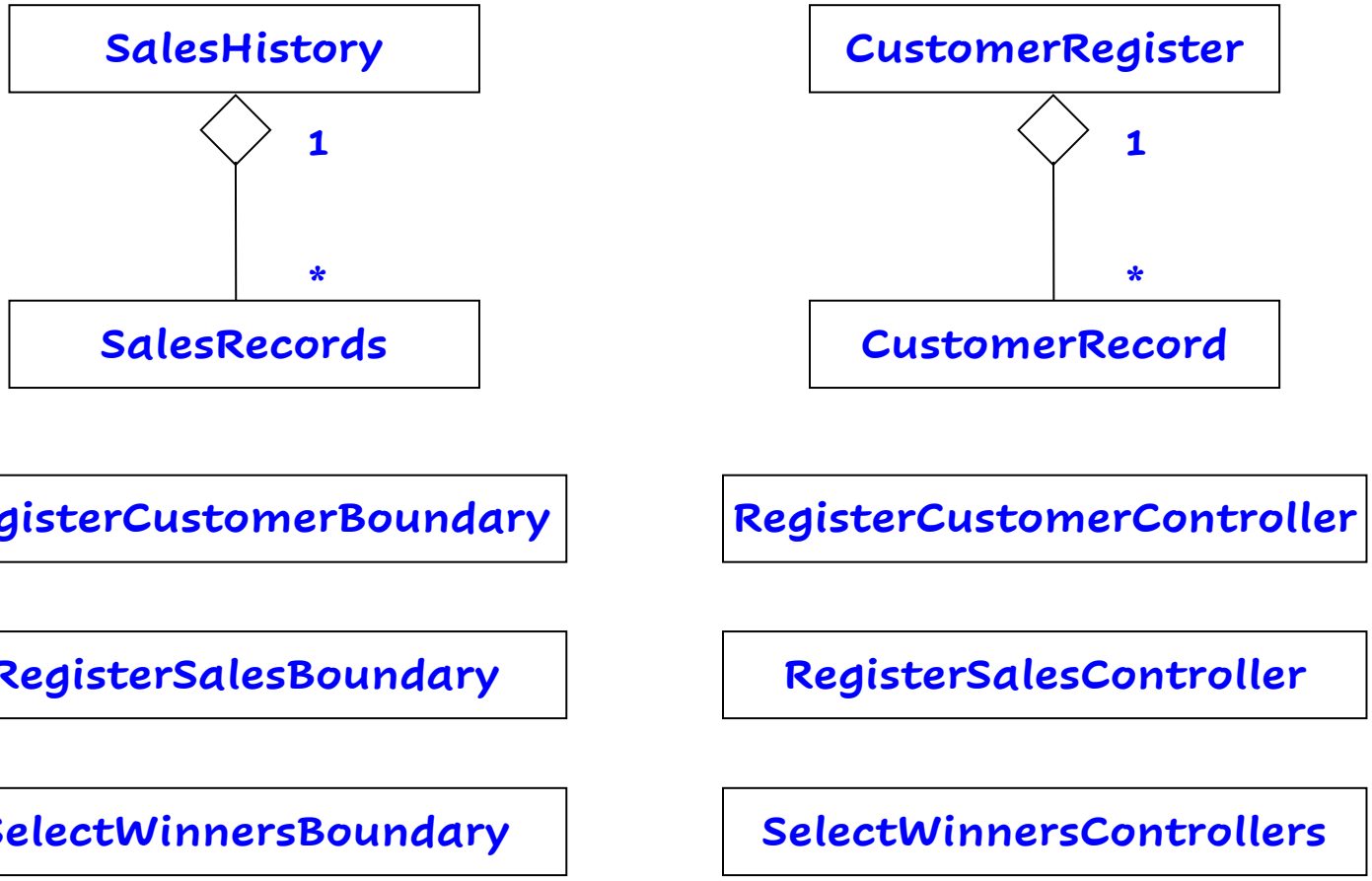


Example 2: Initial Domain Model



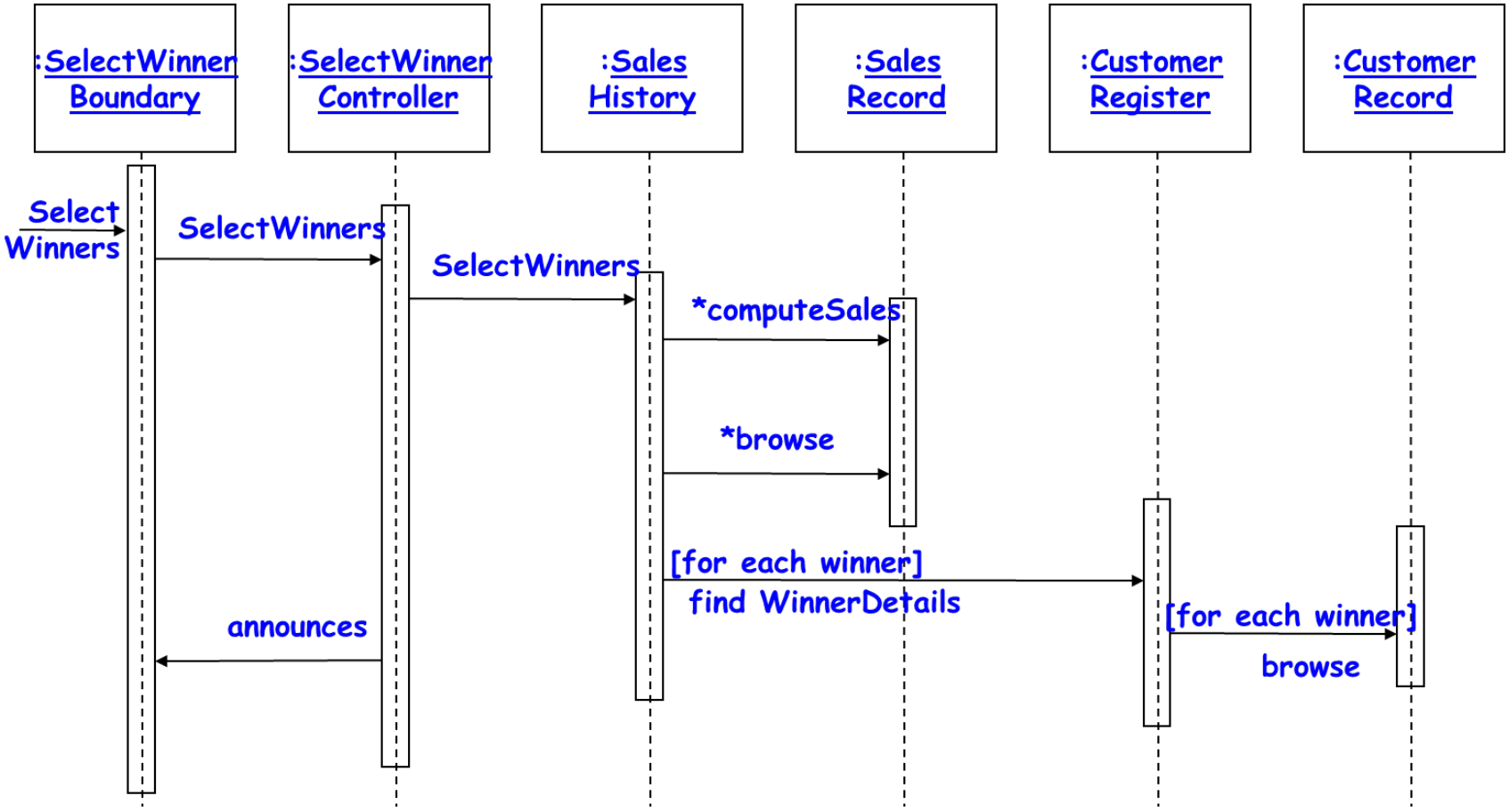
Initial domain model

Example 2: Refined Domain Model



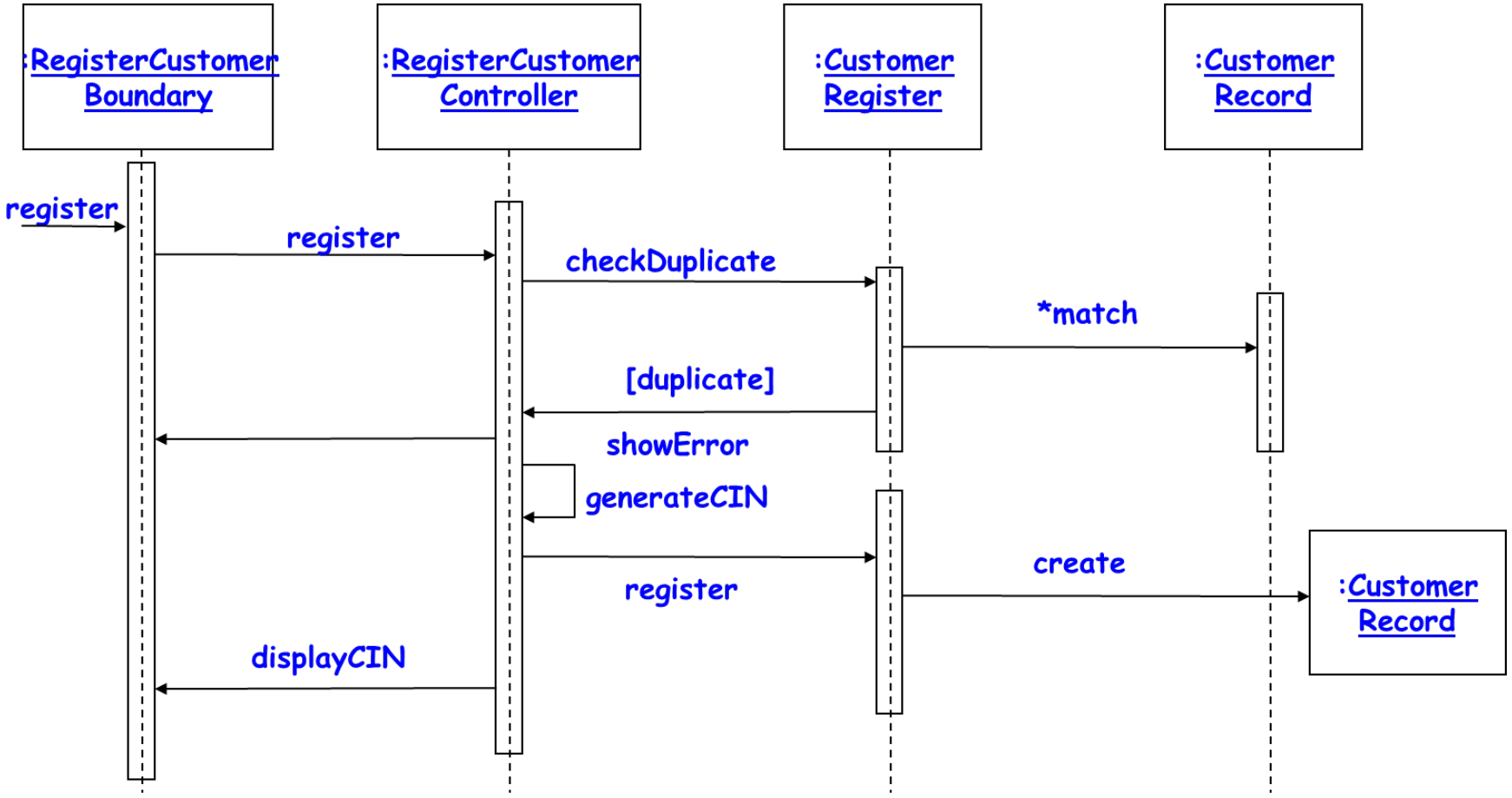
Refined domain model

Example 2: Sequence Diagram for the Select Winners Use Case



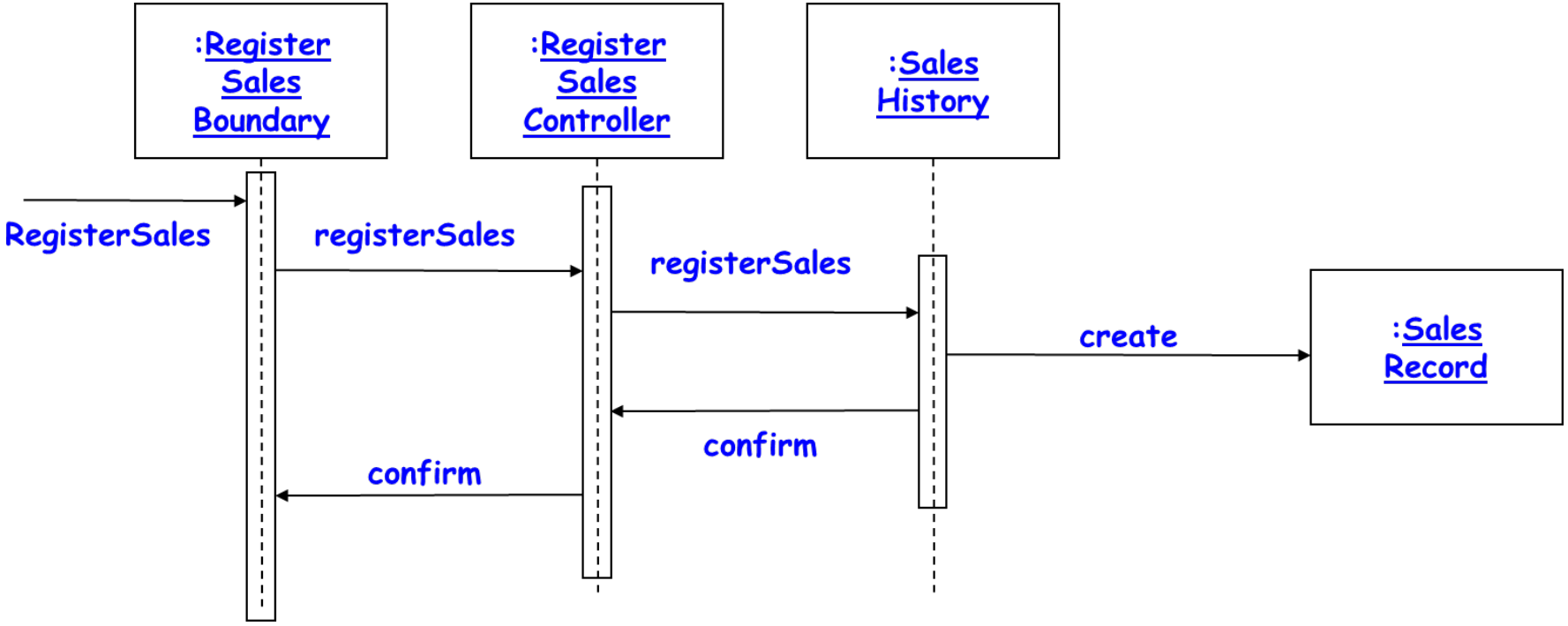
Sequence Diagram for the select winners use case

Example 2: Sequence Diagram for the Register Customer Use Case



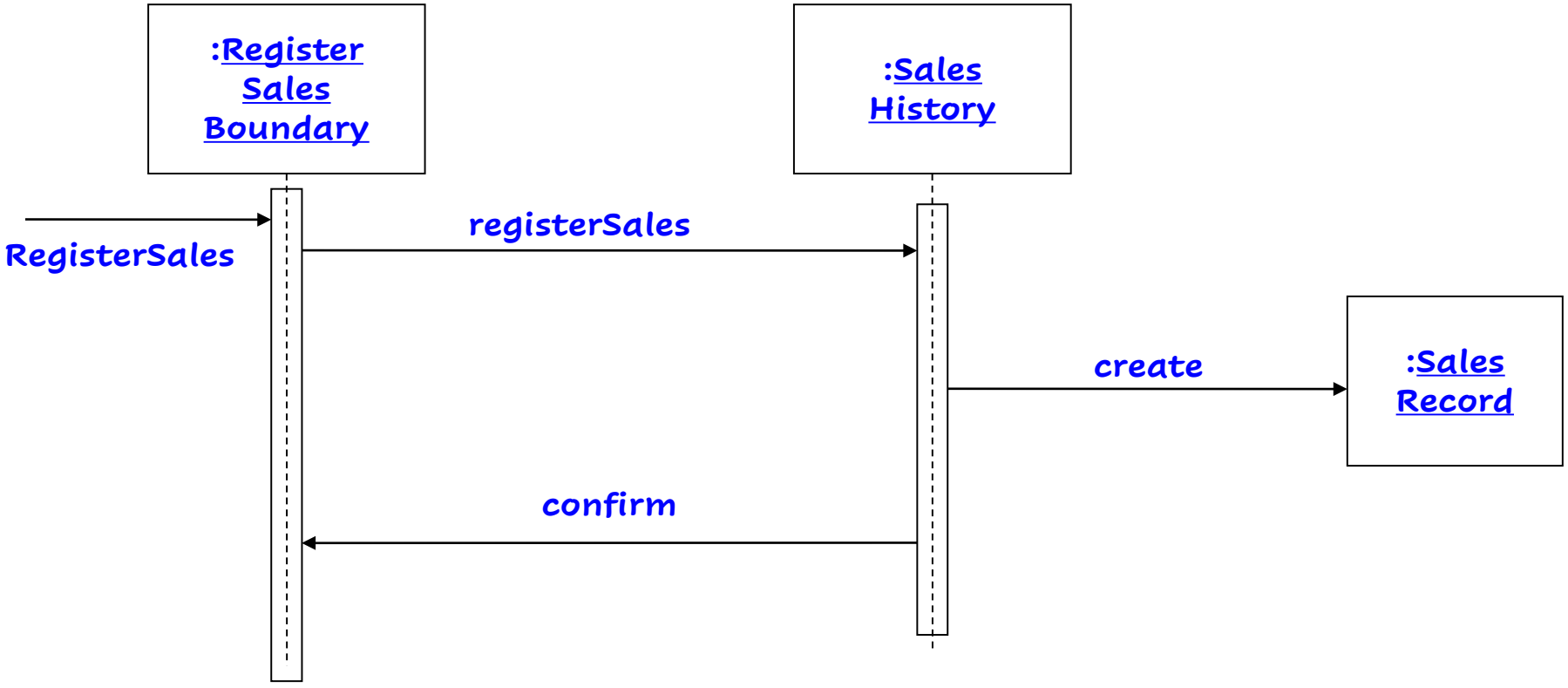
Sequence Diagram for the register customer use case

Example 2: Sequence Diagram for the Register Sales Use Case



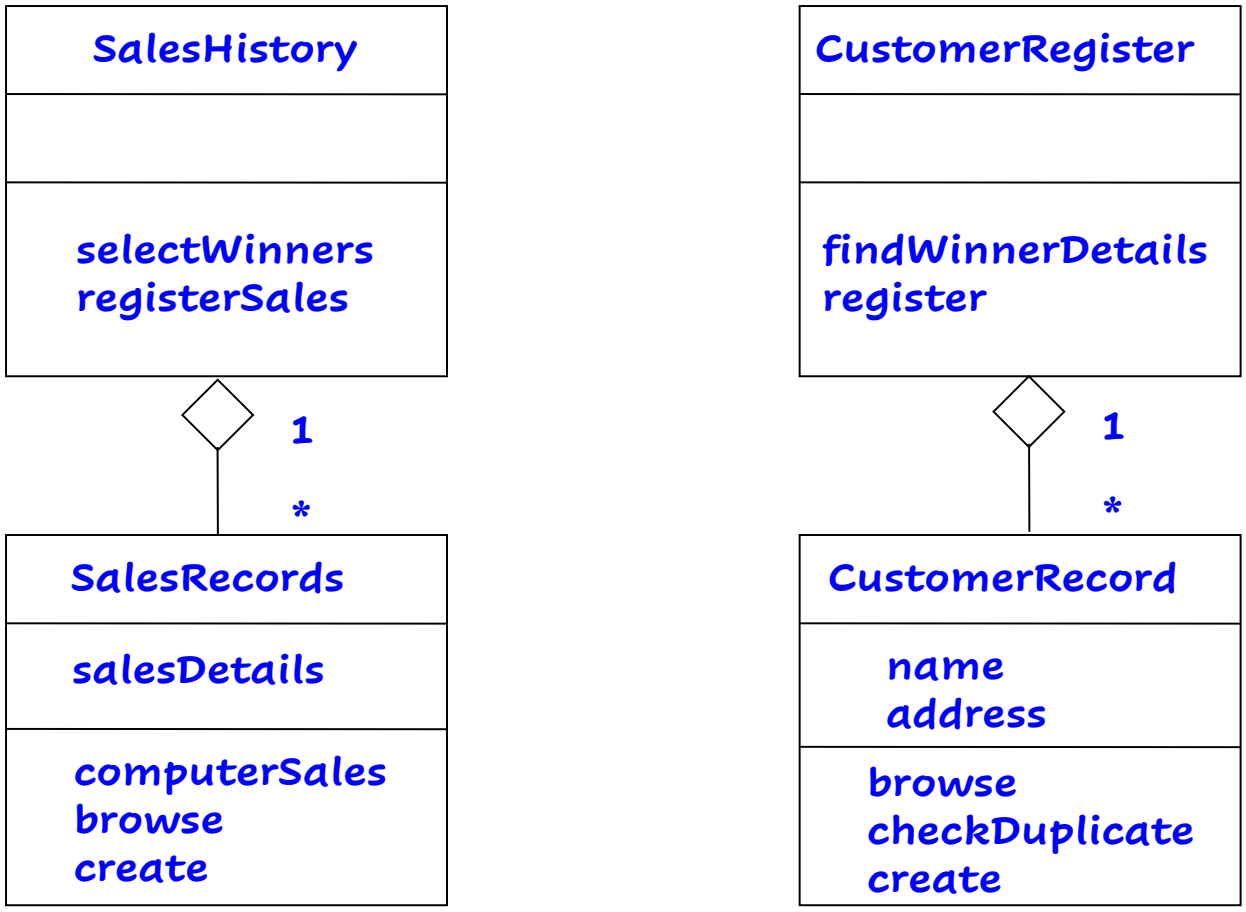
Sequence Diagram for the register sales use case

Example 2: Sequence Diagram for the Register Sales Use Case



Refined Sequence Diagram for the register sales use case

Example 2: Class Diagram



Summary

- We discussed object-oriented concepts
  - ✓ Basic mechanisms: Such as objects, class, methods, inheritance etc.
  - ✓ Key concepts: Such as abstraction, encapsulation, polymorphism, composite objects etc.
- We discussed an important OO language UML:
  - ✓ Its origin, as a standard, as a model
  - ✓ Use case representation, its factorisation such as generalization, includes and extends
  - ✓ Different diagrams for UML representation
  - ✓ In class diagram we discussed some relationships association, aggregation, composition and inheritance

Summary

- Other UML diagrams:
  - ✓ Interaction diagrams (sequence and collaboration),
  - ✓ Activity diagrams,
  - ✓ State chart diagrams.
- We discussed OO software development process:
  - ✓ Use of patterns lead to increased productivity and good solutions.