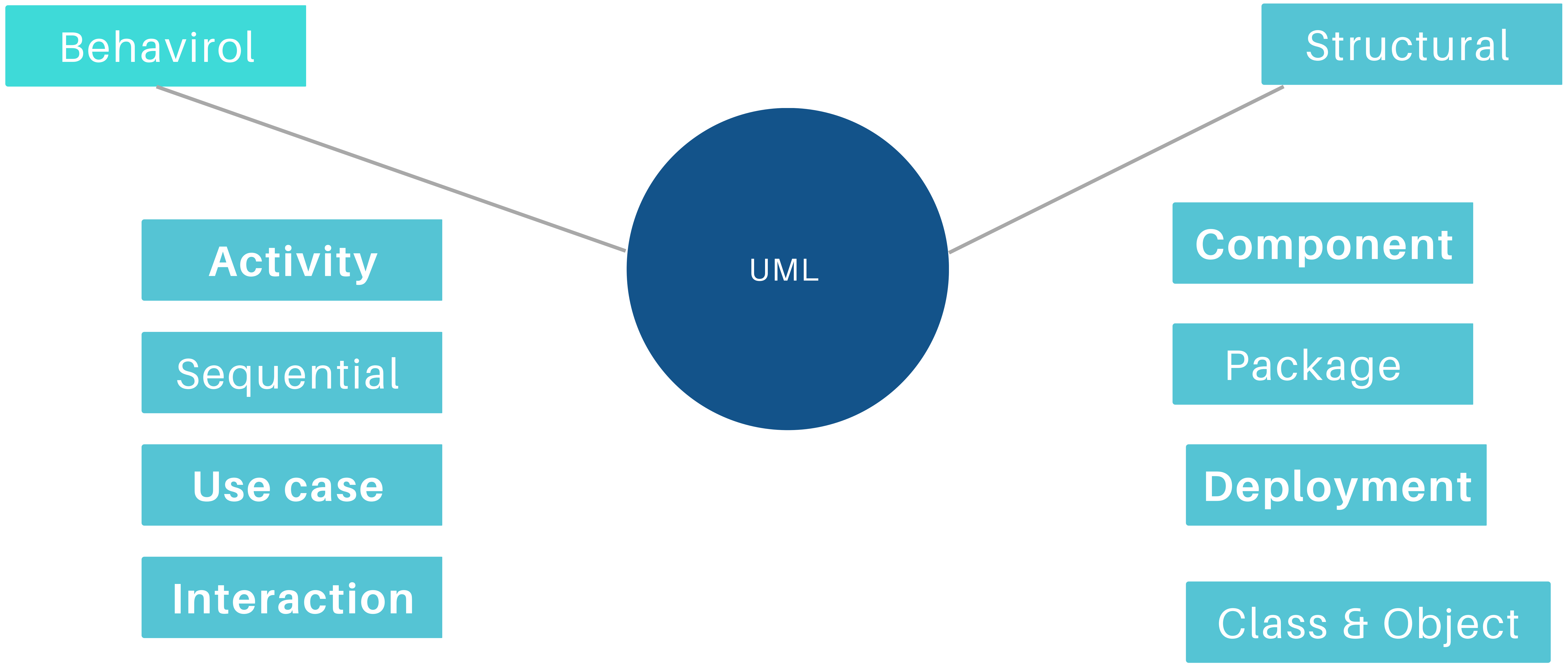
UML

- UML → "Unified Modeling Language"
- Language: express idea, not a methodology
- Modeling: Describing a software system at a high level of abstraction
- industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- uses mostly graphical notations to express the OO analysis and design of software projects.
- Simplifies the complex process of software design

Why we use UML?

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

TYPES OF UML

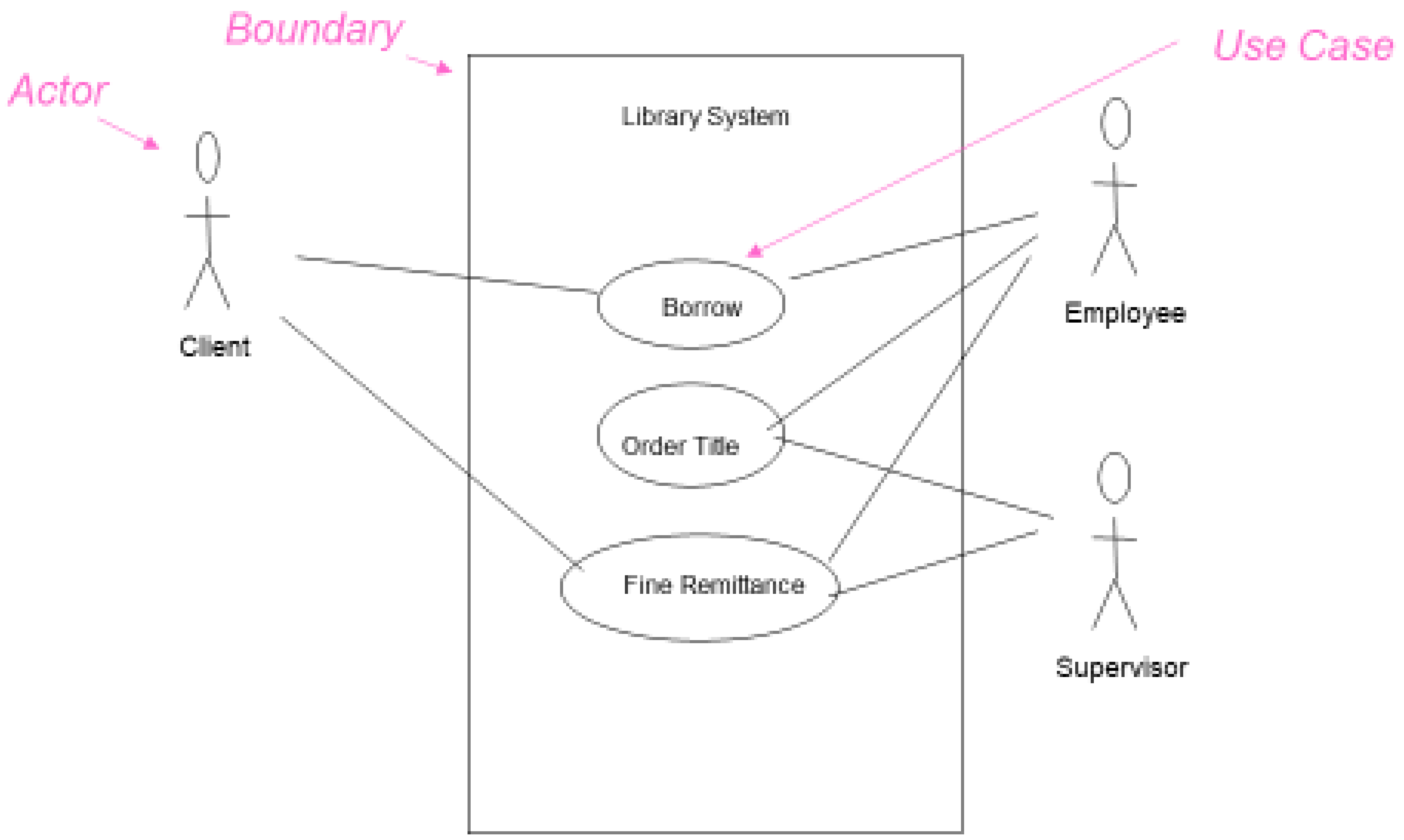

Behavirol
Structural
UML
Activity
Sequential
Use case
Interaction
Component
Package
Deployment
Class & Object

Use-Case Diagrams

- A use-case diagram is a set of use cases
- A use case is a model of the interaction between
  - External users of a software product (actors) and
  - The software product itself
  - More precisely, an actor is a user playing a specific role

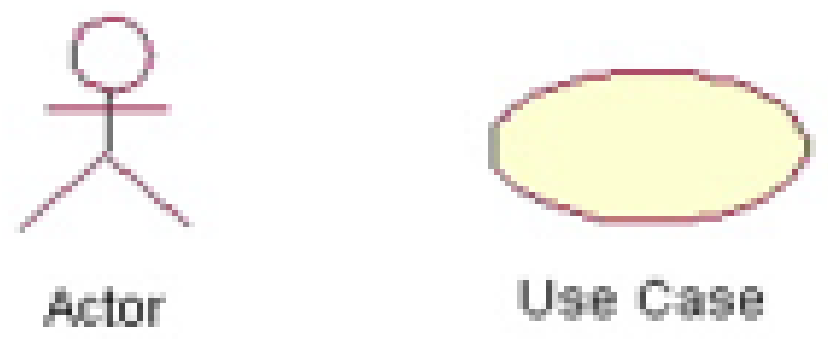
- describing a set of user **scenarios**
- capturing user requirements
- **contract** between end user and software developers

Use-Case Diagrams



Use-Case Diagrams

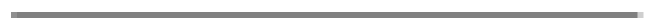
- **Actors:** A role that a user plays with respect to the system, including human users and other systems. e.g., inanimate physical objects (e.g. robot); an external system that needs some information from the current system.
- **Use case:** A set of scenarios that describing an interaction between a user and a system, including alternatives.
- **System boundary**: rectangle diagram representing the boundary between the actors and the system.



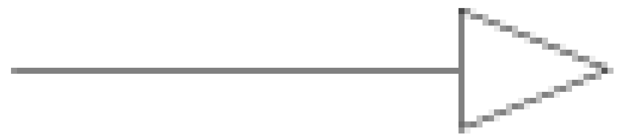
Use-Case Diagrams

- <u>Association:</u>

communication between an actor and a use case; Represented by a solid line.

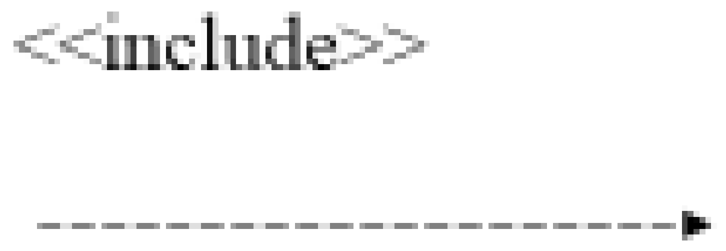


- <u>Generalization</u>: relationship between one general use case and a special use case (used for defining special alternatives) Represented by a line with a triangular arrow head toward the parent use case.

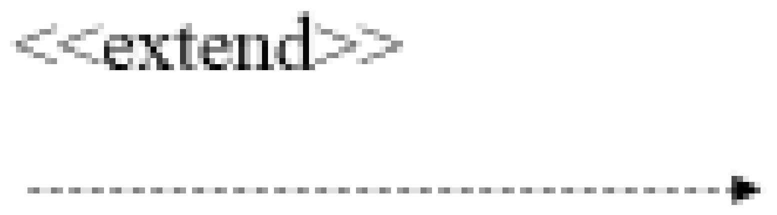


Use-Case Diagrams

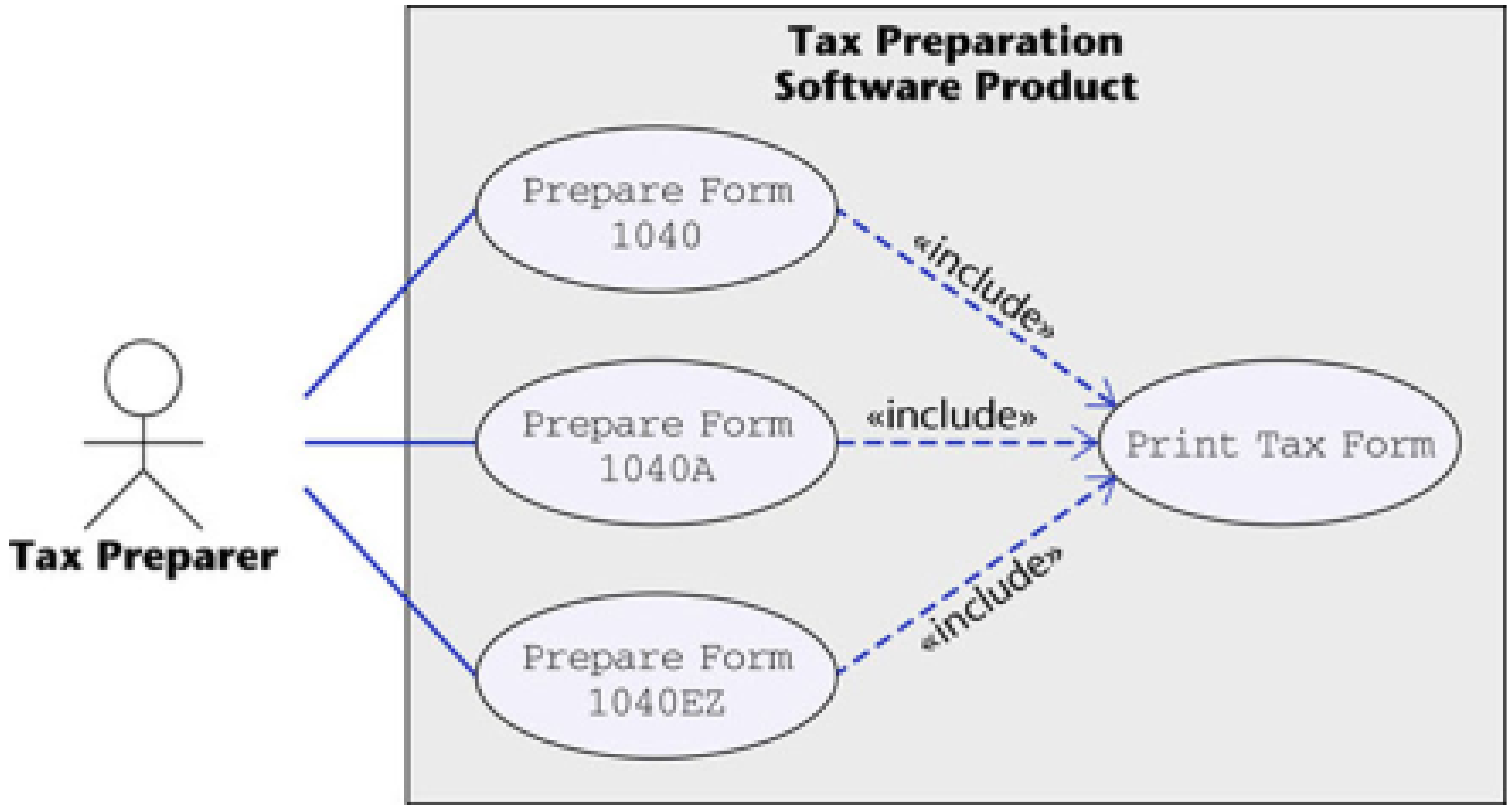
Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case. The include relationship occurs when a chunk of behavior is similar across more than one use case. Use "include" in stead of copying the description of that behavior.

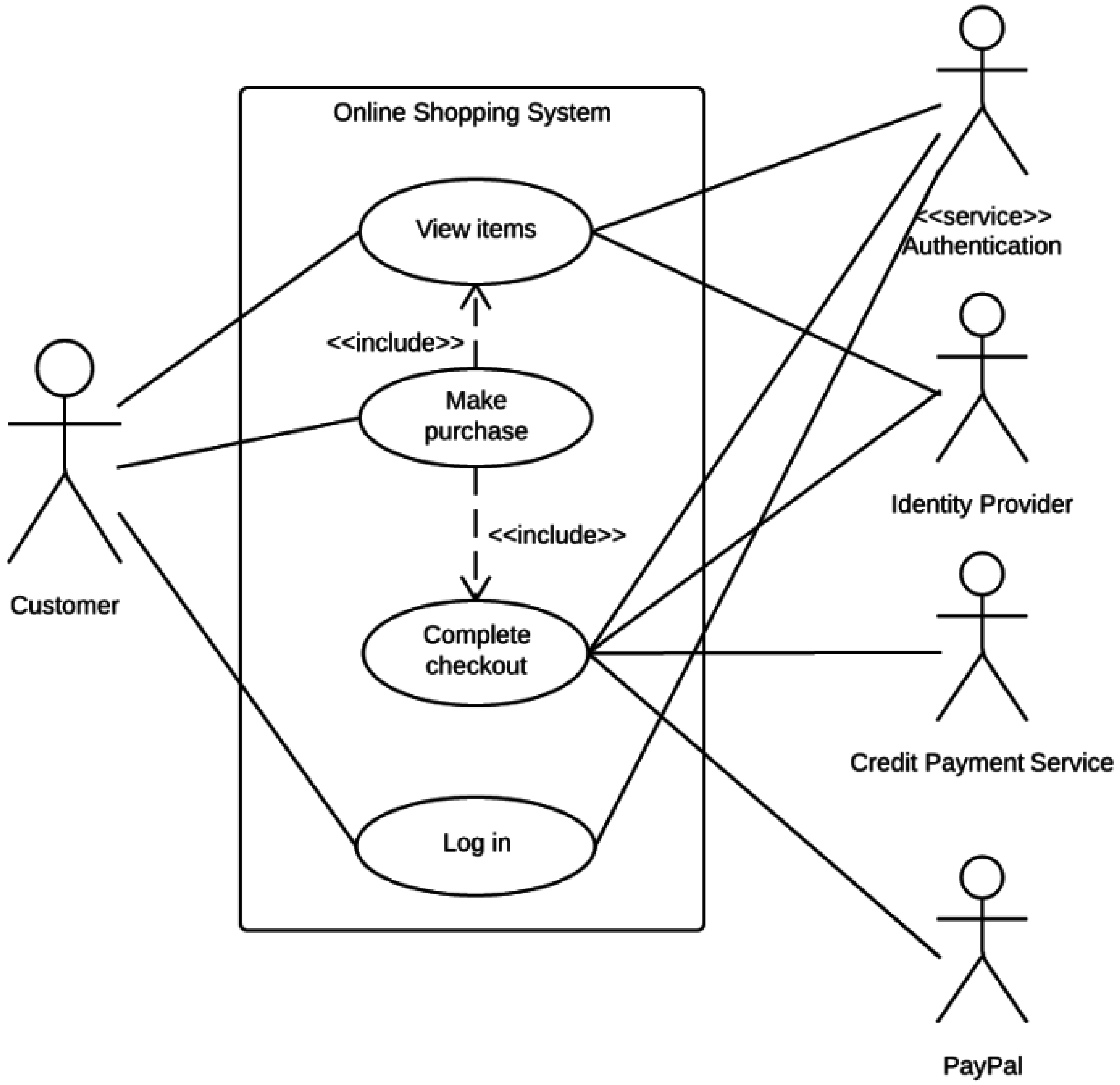


Extend: a dotted line labeled <<extend>> with an arrow toward the base case. The extending use case may add behavior to the base use case. The base class declares "extension points".



Use-Case Diagrams




Online Shopping System
View items
<<include>>
Make purchase
<<include>>
Complete checkout
Log in
Customer
<<service>> Authentication
Identity Provider
Credit Payment Service
PayPal

To represent complex relationships between different use cases, we can use the extend and include relationships.

- **Extend relationship:** The use case is optional and comes after the base use case. It is represented by a dashed arrow in the direction of the base use case with the notation <<extend>>.
- **Include relationship:** The use case is mandatory and part of the base use case. It is represented by a dashed arrow in the direction of the included use case with the notation <<include>>.

the use case “Read book” includes the use case “Open book”. If a reader reads the book, she must open it too, as it is mandatory for the base use case (read book). The use case “read book” extends to the use case “turn page”, which means that turning the page while reading the book is optional. The base use case in this scenario (read book) is complete without the extended use case.

Class diagram

- A class diagram depicts classes and their interrelationships
- Used for describing structure and behavior in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

- To model the objects that make up the system
- To display the relationships between the objects
- To describe what those objects do and the services that they provide.
- Class diagrams are useful in many stages of system design:

1. In the analysis stage, a class diagram can help you to understand the requirements of your problem domain and to identify its components.
2. Later, you can refine your earlier analysis and conceptual models into class diagrams that show the specific parts of your system, user interfaces, logical implementations, and so on.
3. Your class diagrams then become a snapshot that describes exactly how your system works, the relationships between system components at many levels, and how you plan to implement those components.

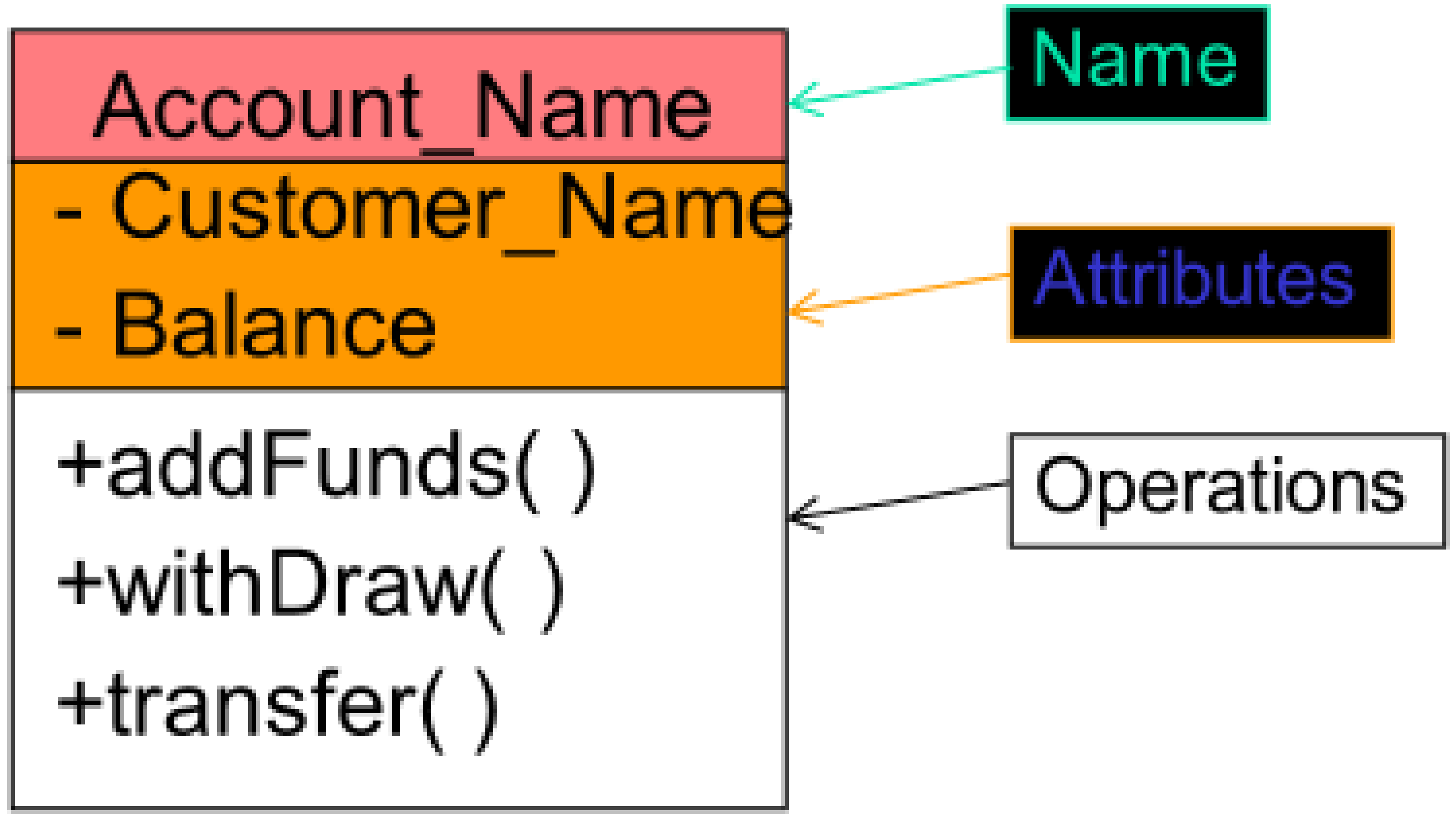
Class diagram

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations

- Modifiers are used to indicate visibility of attributes and operations.
  - '+' is used to denote *Public* visibility (everyone)
  - '#' is used to denote *Protected* visibility (friends and derived)
  - '-' is used to denote *Private* visibility (no one)

- By default, attributes are hidden and operations are visible.

Class diagram

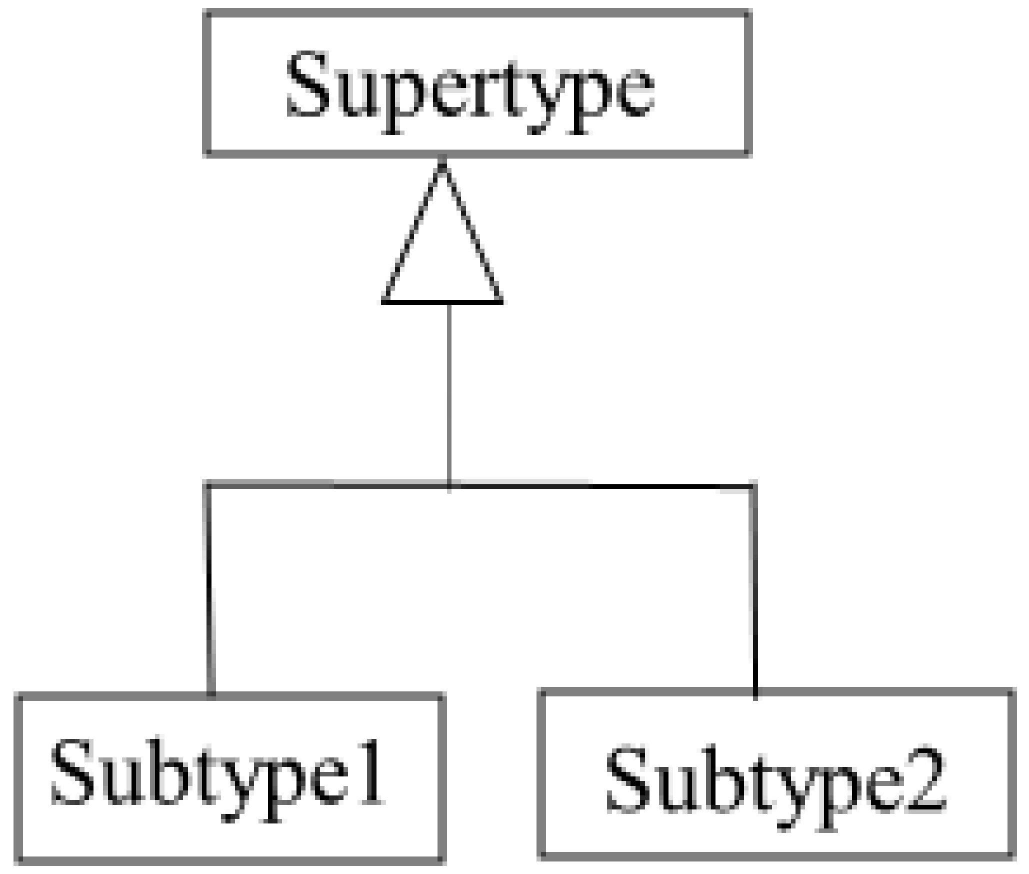


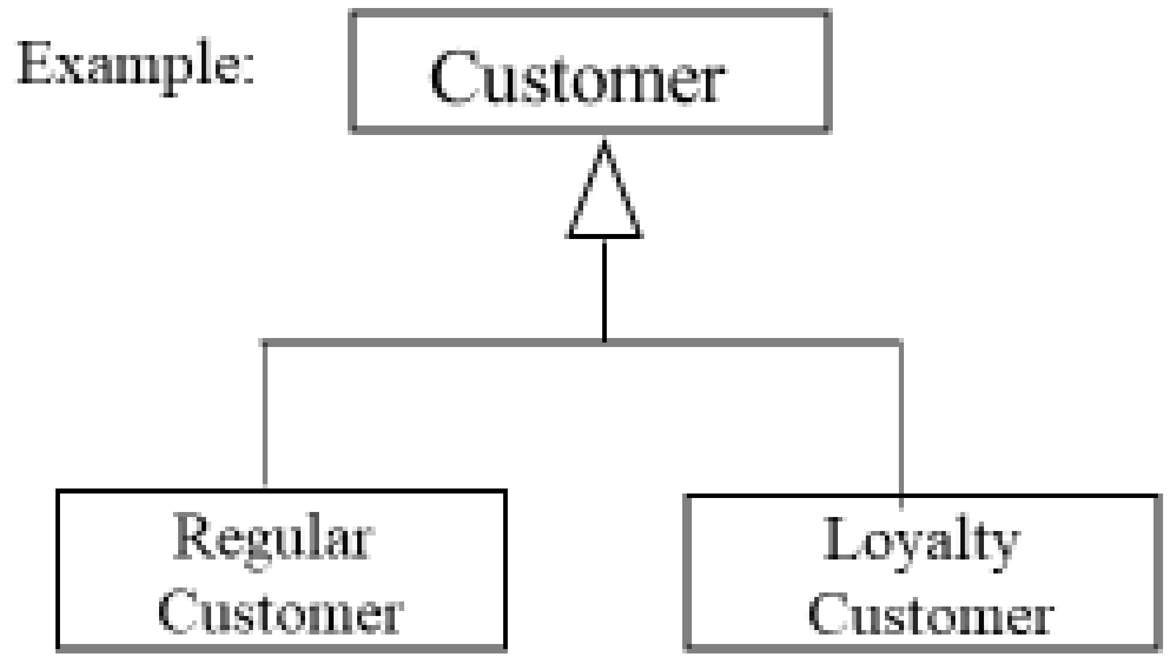
OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)

- Associations can be further classified as
  - Aggregation
  - Composition

OO Relationships: **Generalization**





-Inheritance is a required feature of object orientation

-Generalization expresses a parent/child relationship among related classes.

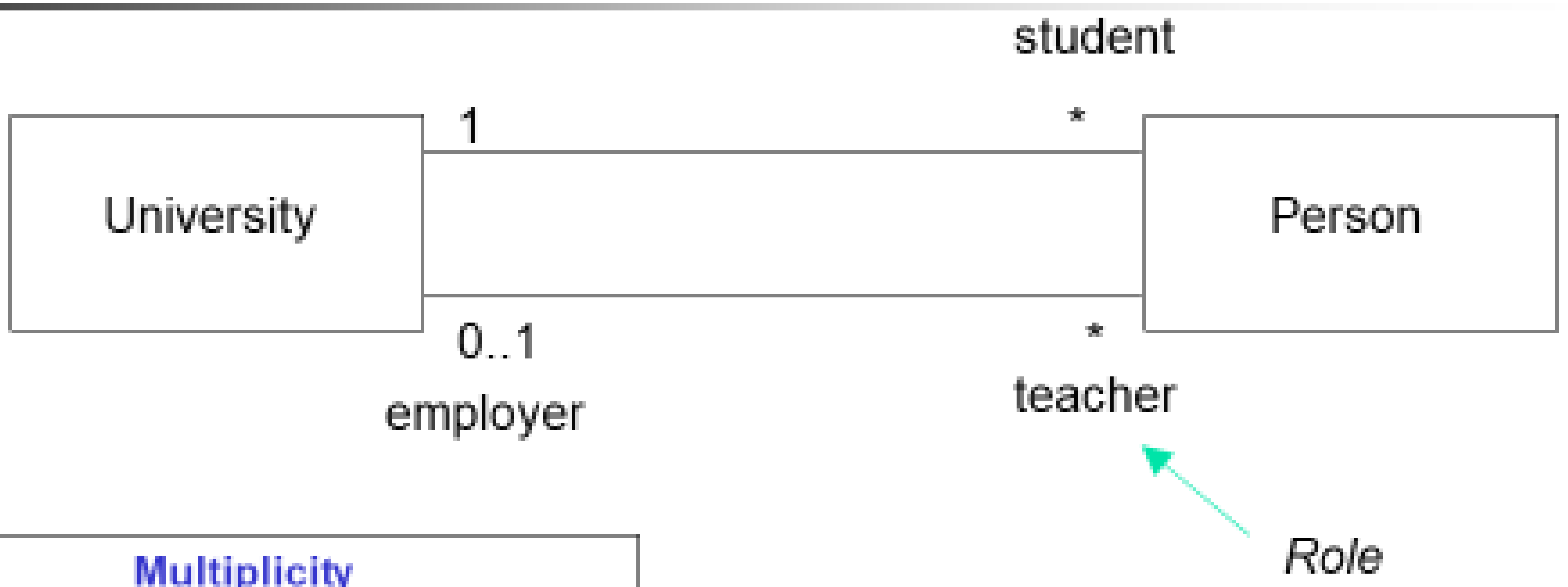
-Used for abstracting details in several layers

OO Relationships: **Association**

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.

- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles

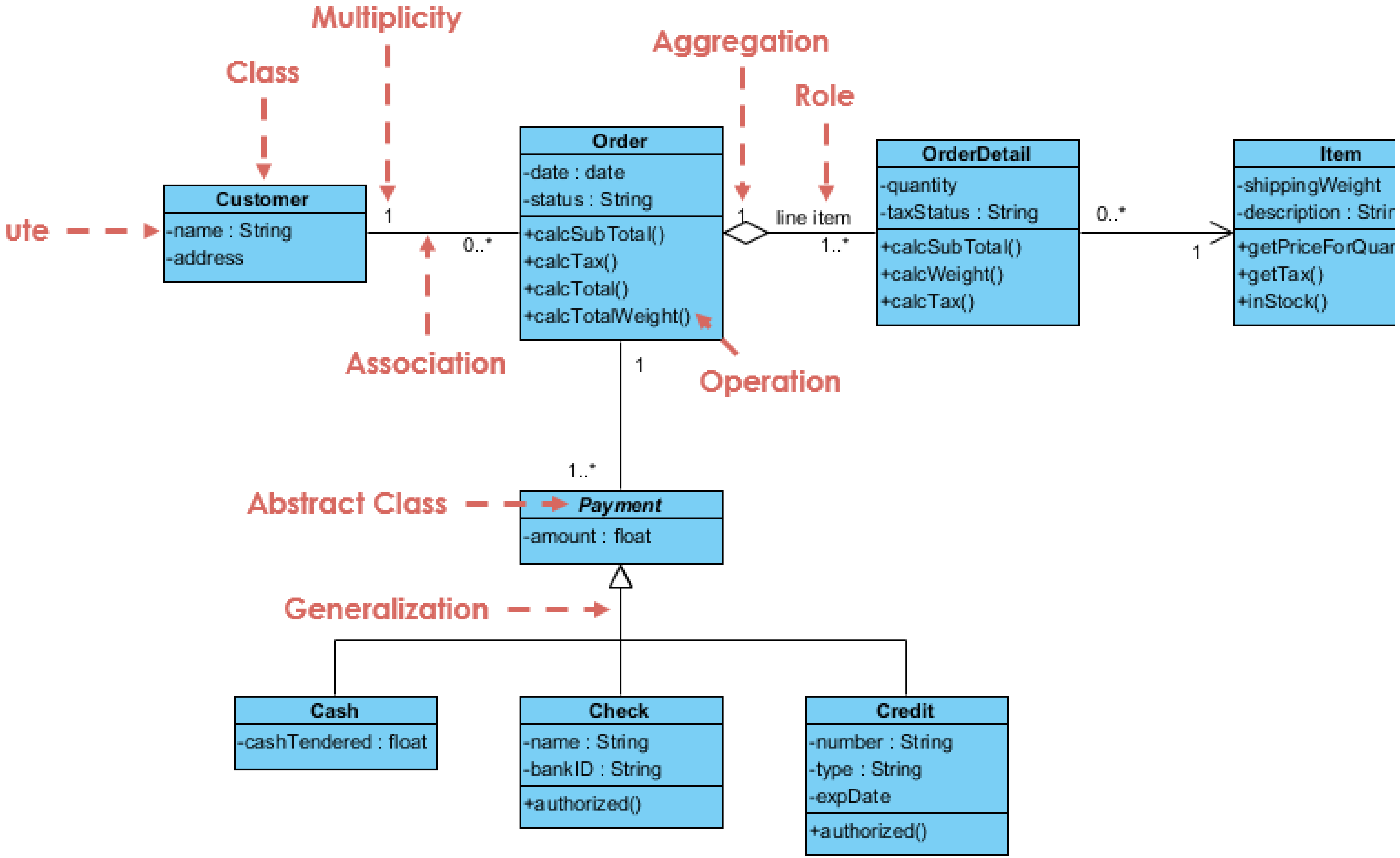


Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

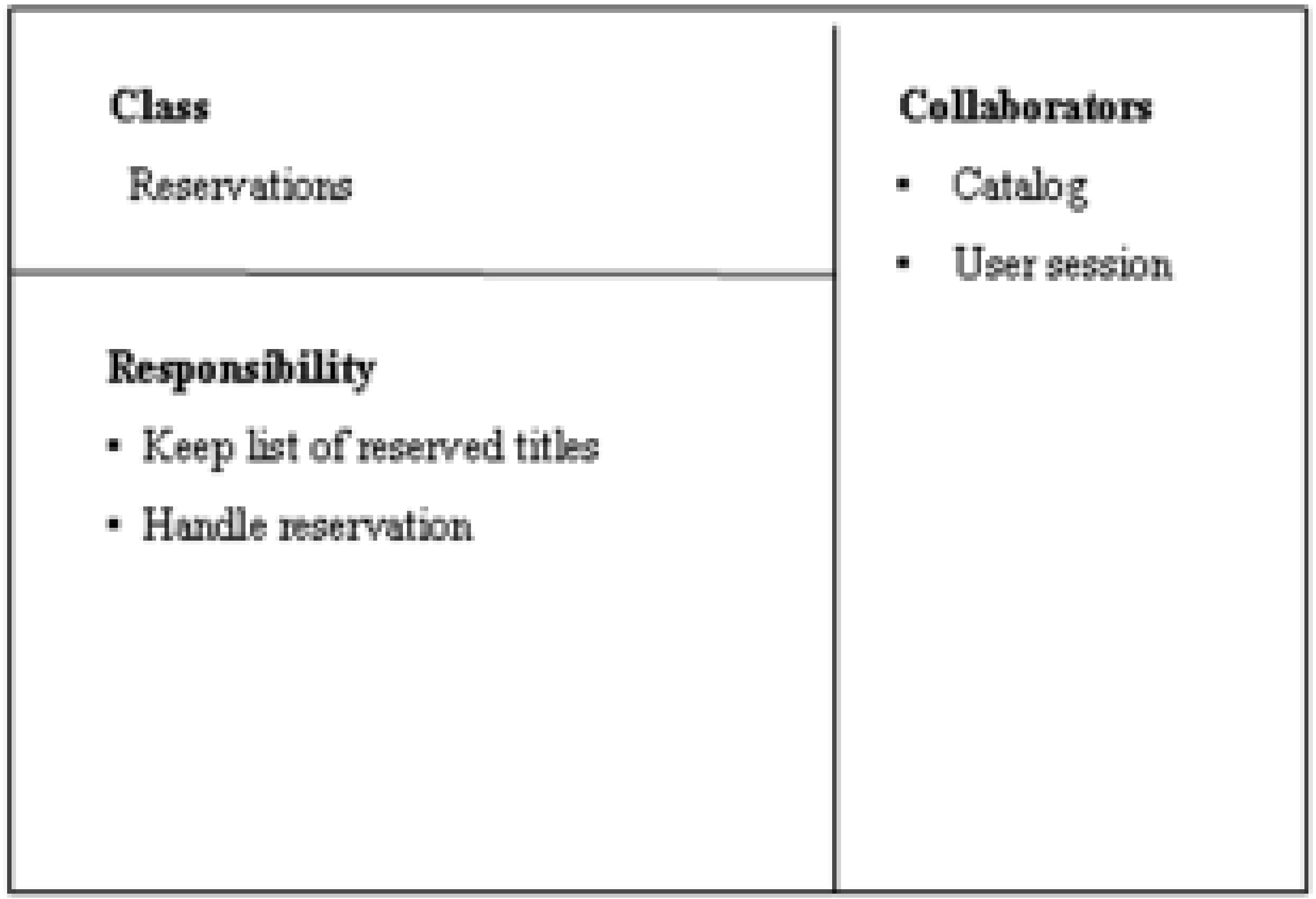
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."


Multiplicity
Class
Aggregation
Role
ute
Customer
-name : String
-address
1
0..*
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
line item
1..*
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description : Strir
+getPriceForQuar
+getTax()
+inStock()
Association
Operation
1
1..*
Abstract Class
Payment
-amount : float
Generalization
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()

Good Practice: CRC Card

Class Responsibility Collaborator

- easy to describe how classes work by moving cards around; allows to quickly consider alternatives.



Class

Reservations

Responsibility

- Keep list of reserved titles
- Handle reservation

Collaborators

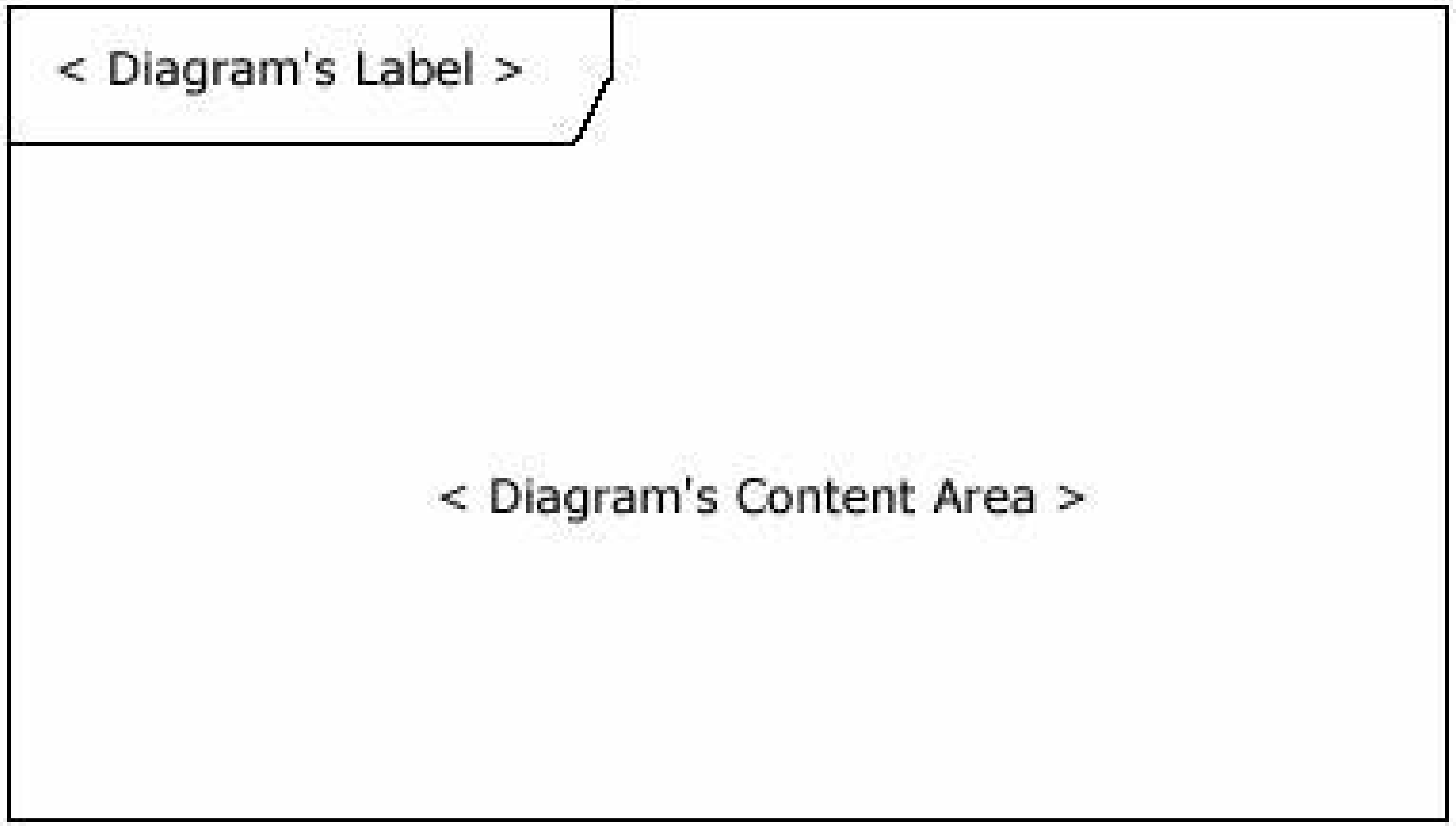
- Catalog
- User session

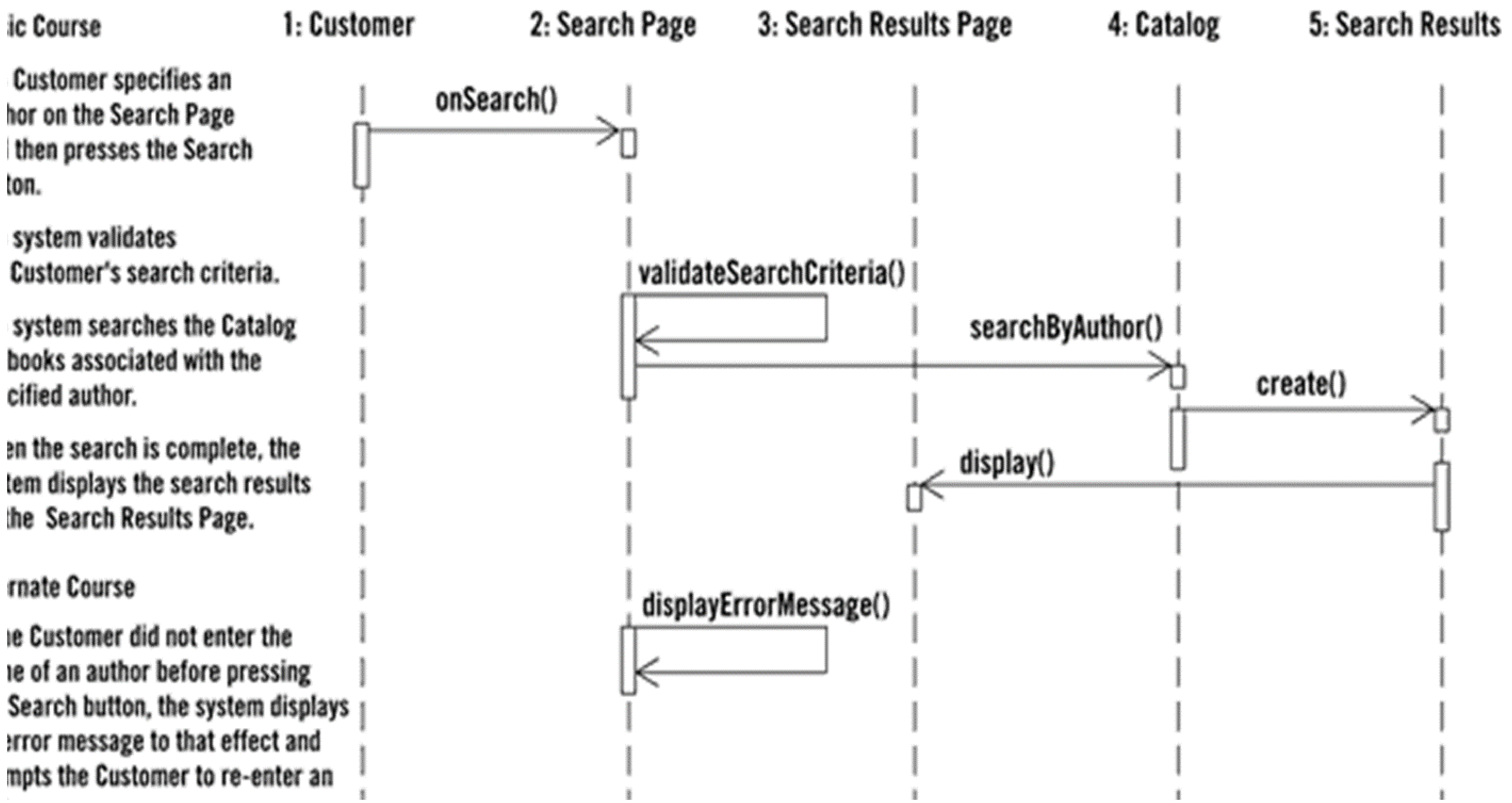
Sequence Diagram

- Purpose :

To show the interactions between objects in the sequential order that those interactions occur.

- useful in documenting how a future system should behave.
- An "interaction diagram" that models a single scenario executing in the system
- Transition from requirements expressed as use cases to the next and more formal level of refinement. Use cases are often refined into one or more sequence diagrams. In addition to their use in designing new systems, sequence diagrams can be used to document how objects in an existing system currently interact
- CRC cards -> class diagram
- use cases -> sequence diagrams


< Diagram's Label >
< Diagram's Content Area >


ic Course

Customer specifies an
hor on the Search Page
l then presses the Search
ton.

system validates
Customer's search criteria.

system searches the Catalog
books associated with the
cified author.

en the search is complete, the
tem displays the search results
the Search Results Page.

rnate Course

ie Customer did not enter the
ne of an author before pressing
Search button, the system displays
error message to that effect and
mpts the Customer to re-enter an

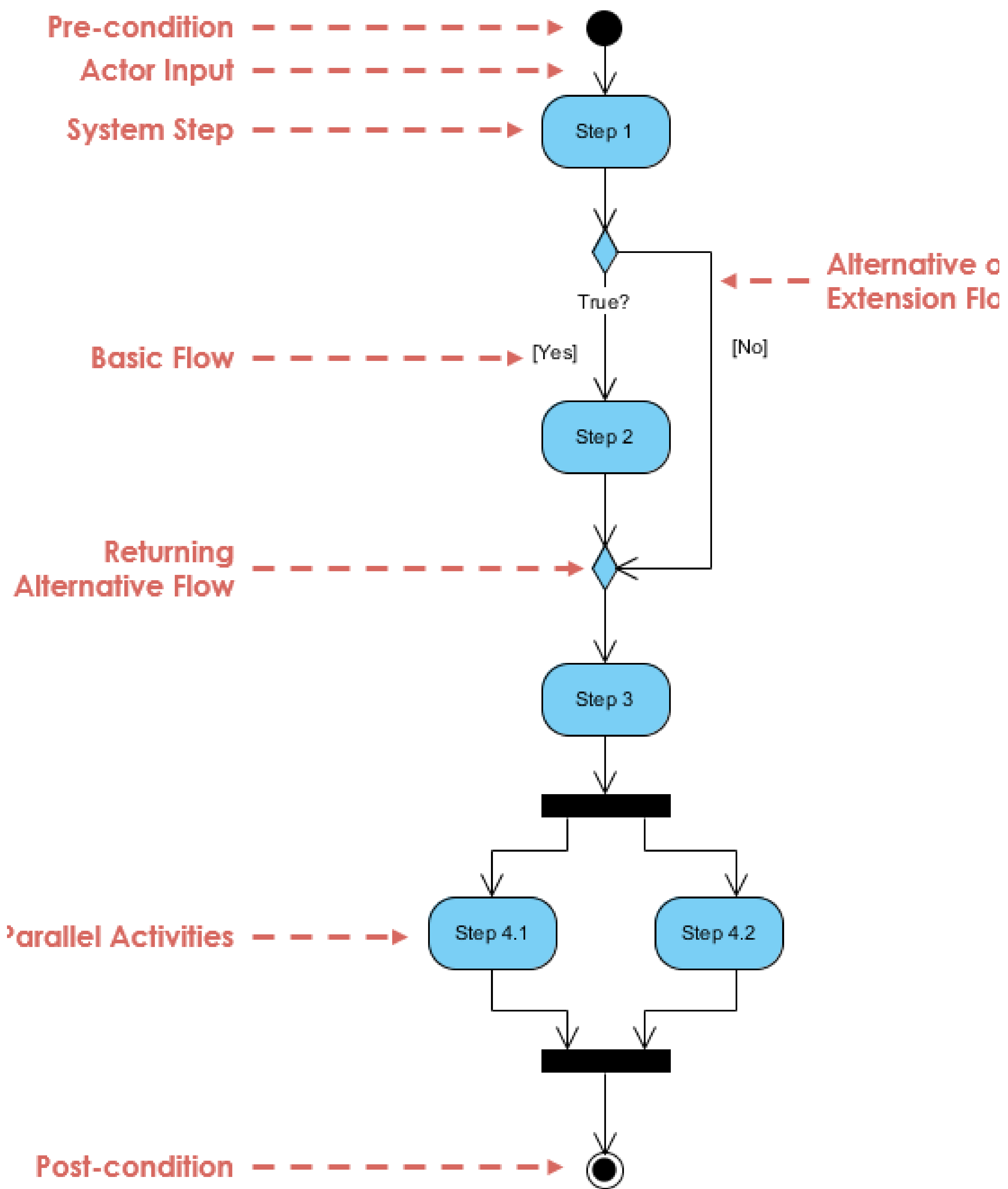
Why not just code it?

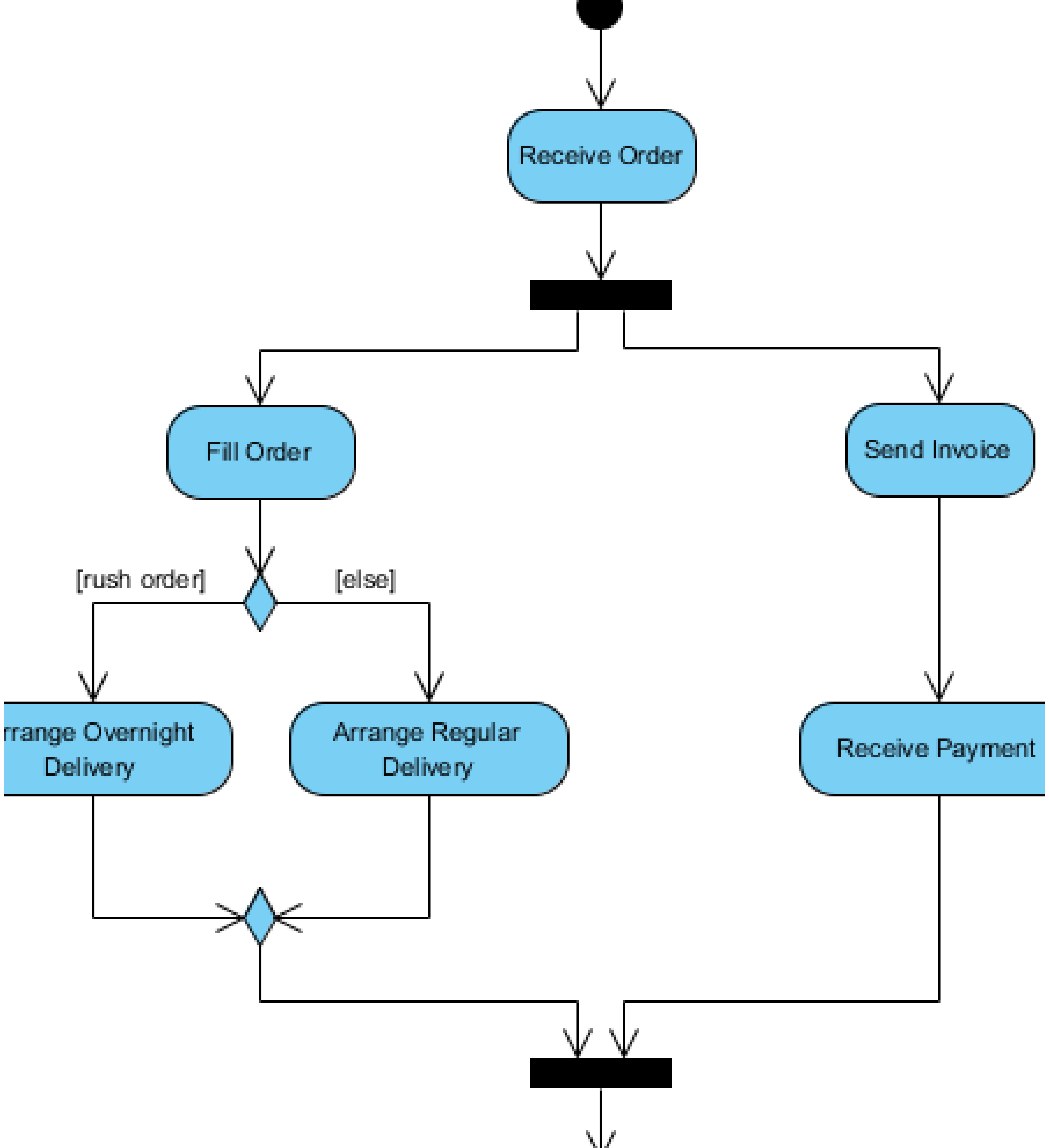
Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram

- A good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
- sequence diagrams are language-agnostic (can be implemented in many different languages
- non-coders can do sequence diagrams
- easier to do sequence diagrams as a team can see many objects/classes at a time on same page (visual bandwidth)

Activity Diagram

- Supplements the use-case by providing a diagrammatic representation of procedural flow
- a flowchart to represent the flow from one activity to another activity.
- The activity can be described as an operation of the system.
- The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.


Pre-condition
Actor Input
System Step
Step 1
True?
Alternative o
Extension Flo
[No]
Basic Flow
[Yes]
Step 2
Returning
Alternative Flow
Step 3
arallel Activities
Step 4.1
Step 4.2
Post-condition


Receive Order
Fill Order
Send Invoice
[rush order]
[else]
rrange Overnight Delivery
Arrange Regular Delivery
Receive Payment

Activity Diagram Notation Summary

- **Activity :**Is used to represent a set of actions
- Action : A task to be performed
- **Control Flow :**Shows the sequence of execution
- **Initial Node :** Portrays the beginning of a set of actions or activities
- **Activity Final Node :** Stop all control flows and object flows in an activity (or action)
- **Decision Node"** Represent a test condition to ensure that the control flow or object flow only goes down one path
- **Fork Node :** Split behavior into a set of parallel or concurrent flows of activities (or actions)
- **Join Node :**Bring back together a set of parallel or concurrent flows of activities (or actions)

COMPONENT DIAGRAM

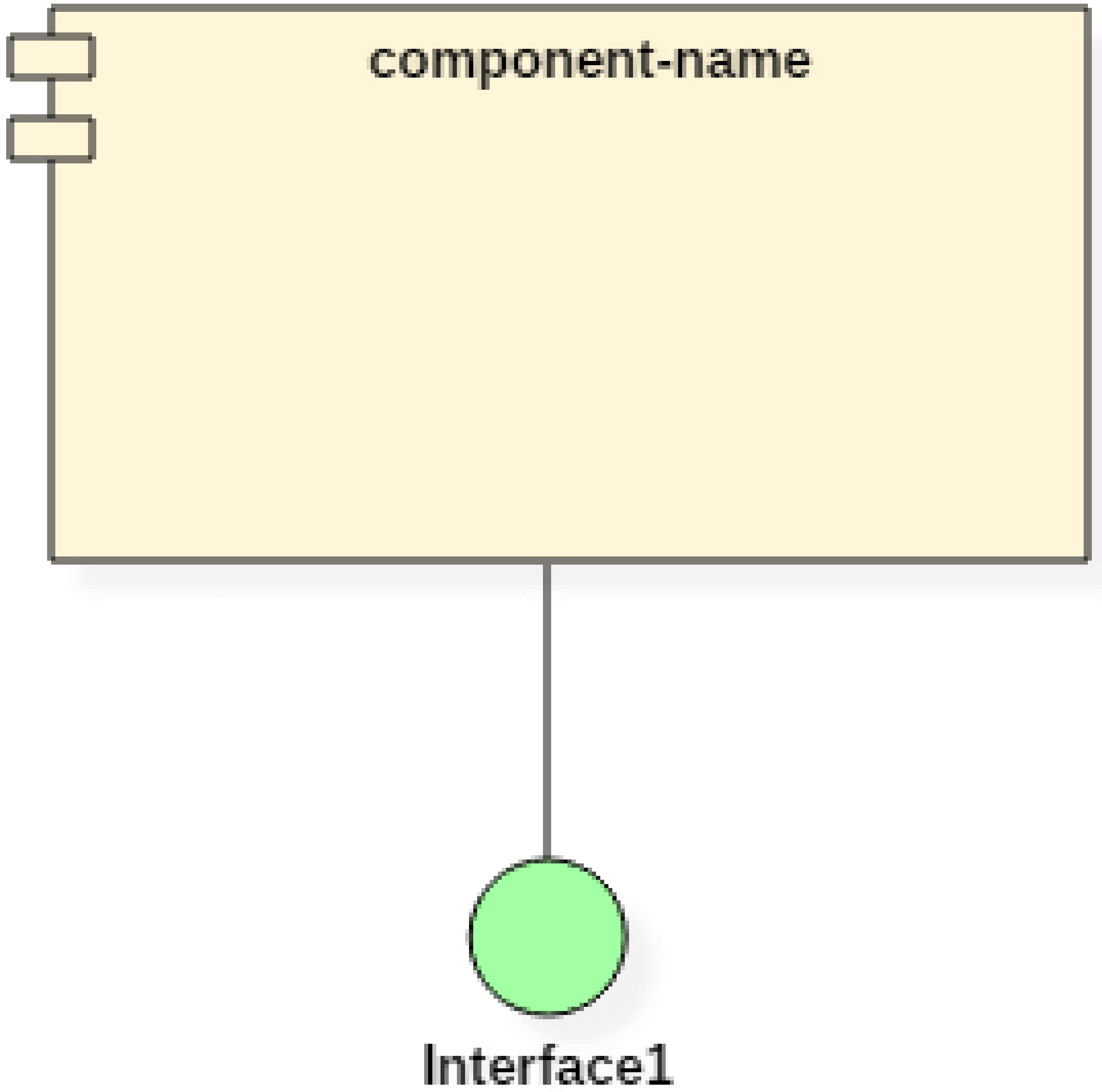
- Component diagrams are used to model physical aspects of a system
- Physical aspects are the elements like executables, libraries, packages, files, documents etc which resides in a node.
- A component diagram – Shows the various components in a system and their dependencies.

Purpose

1. Visualize the components of a system.
2. Construct executables by using forward and reverse engineering.
3. Describe the organization and relationships of the components.
4. This diagrams are used during the implementation phase of an application.
5. Model the components of a system.
6. Model database schema.
7. Model executables of an application.
8. Model system's source code.

- **Before drawing a component diagram the following artifacts are to be identified clearly:**

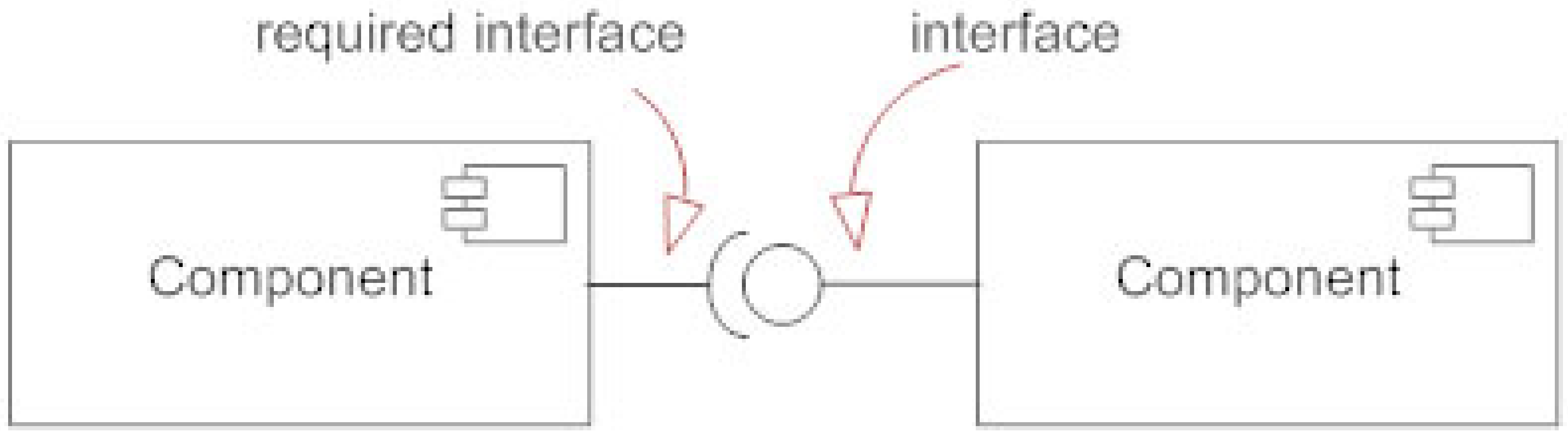
1. Files used in the system.
2. Libraries and other objects relevant to the application.
3. Relationships among the objects.
4. After identifying the objects the following points needs to be followed:
5. Use meaningful name to identify the component to which diagram is drawn.
6. Prepare a mental layout before producing using tools. 3. Use notes for clarifying important points.

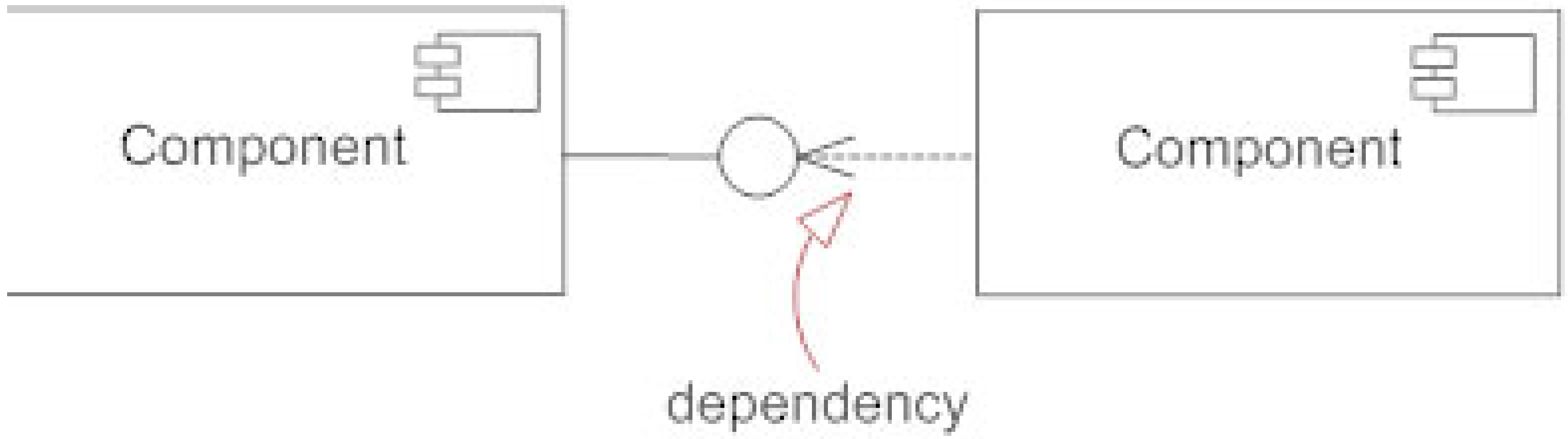

component-name
Interface1

There are two types of Interfaces in Component Diagram:

1. **Provided interfaces**
2. **Required interfaces**

An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.

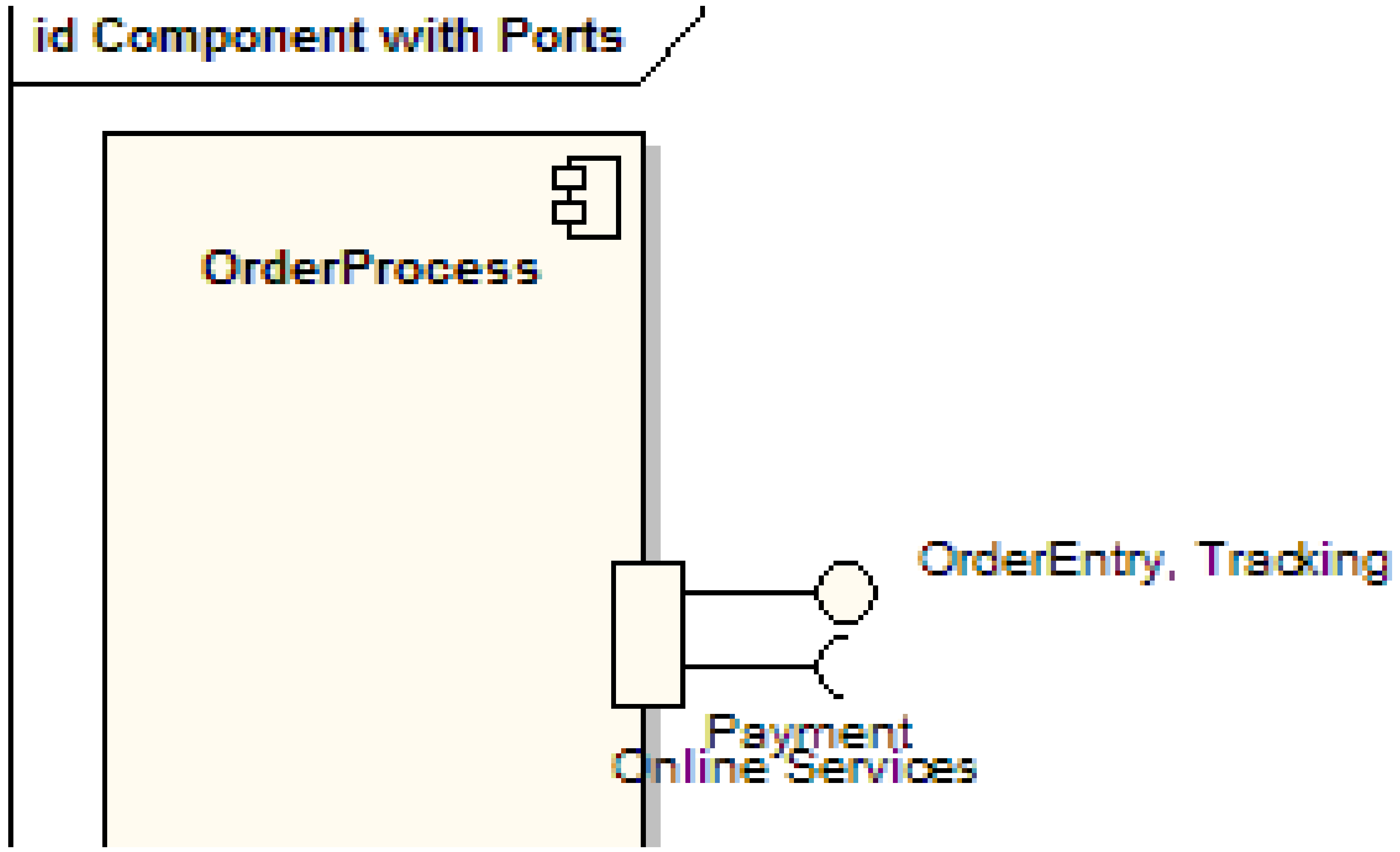




Dependencies : Draw dependencies among components using dashed arrows.

Components with Ports

- Using Ports with component diagrams allows for a service or behavior to be specified to its environment as well as a service or behavior that a component requires.
- Ports may specify inputs and outputs as they can operate bi-directionally. The following diagram details a component with a port for online services along with two provided interfaces order entry and tracking as well as a required interface payment.

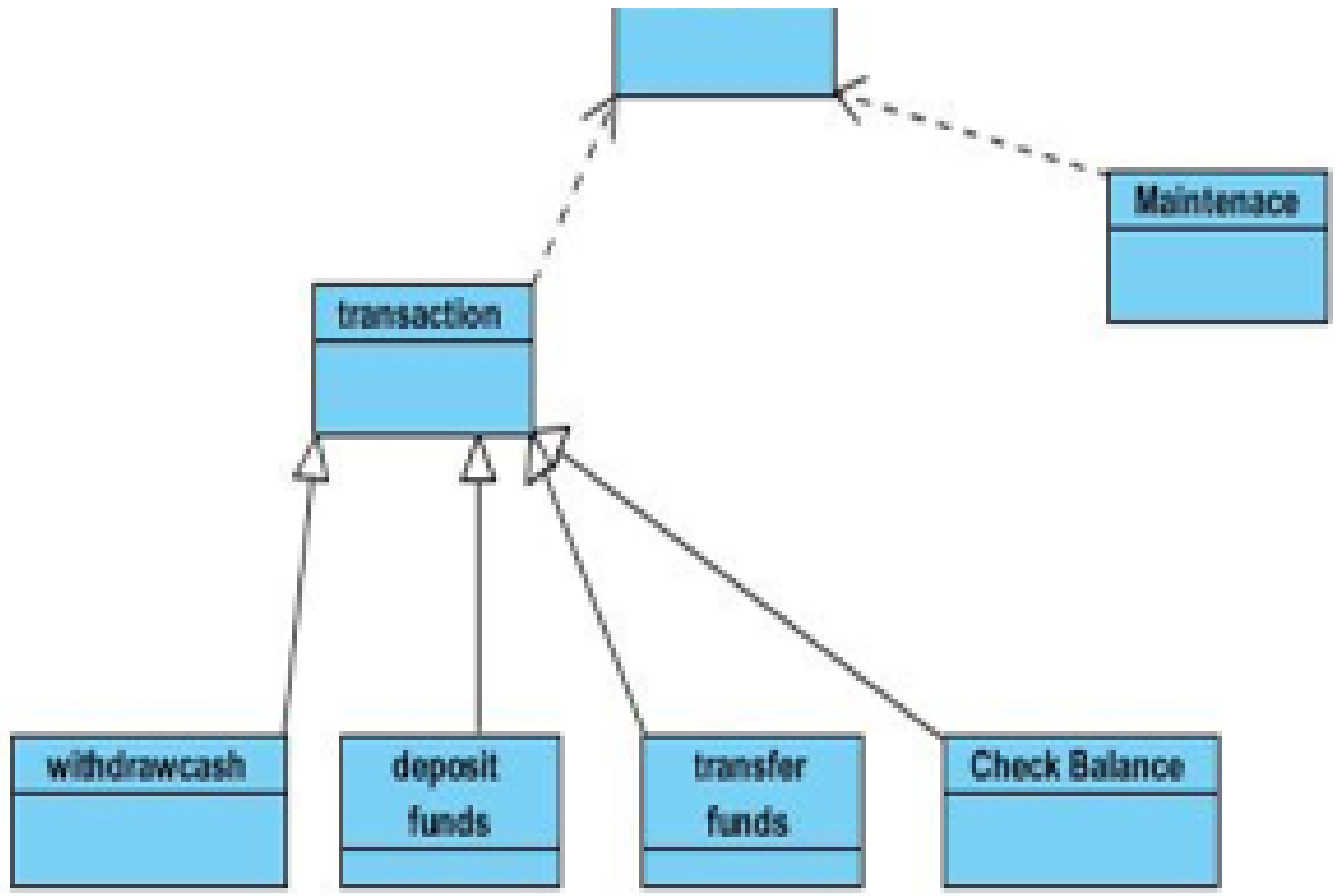


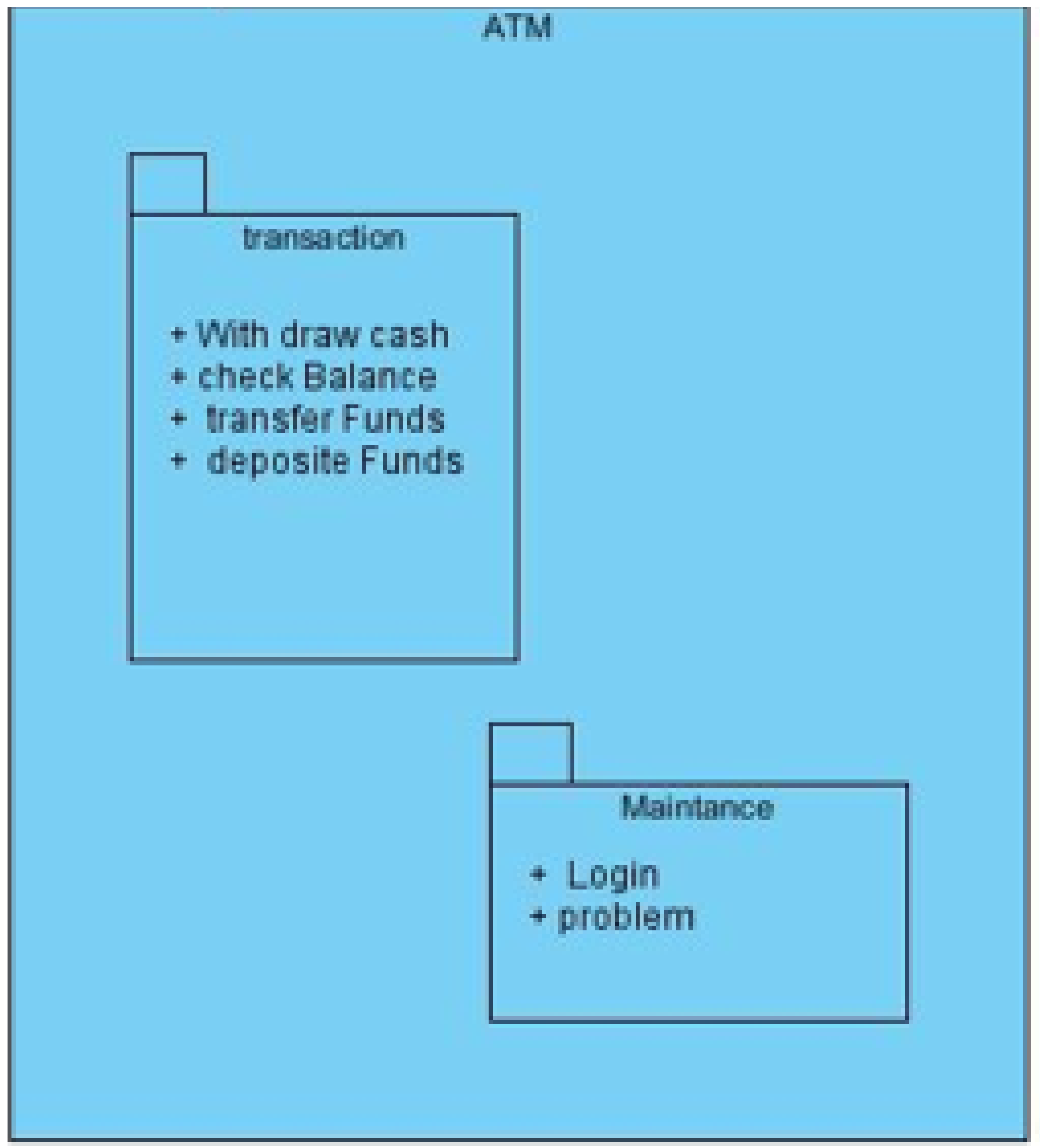
Package diagram

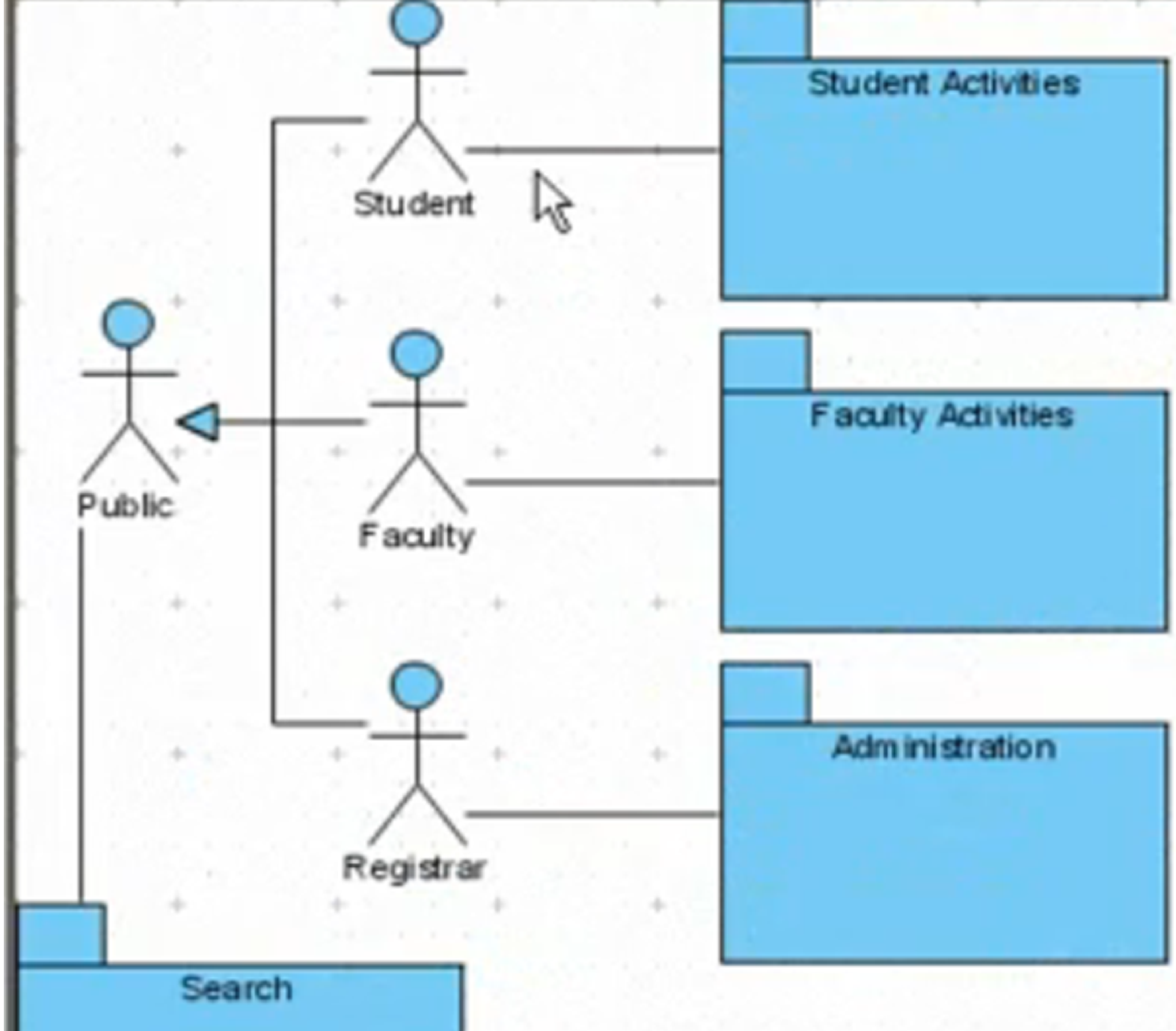
- Package diagrams are structural diagrams used to show the organization and arrangement of various model elements in the form of packages.
- A package is a grouping of related UML elements, such as diagrams, documents, classes, or even other packages.
- **The difference between package diagrams and component diagrams** is that Component Diagrams offer a more semantically rich grouping mechanism. With component diagrams all of the model elements are private, whereas package diagrams only display public items.
- Component diagram typically employed to illustrate interfaces. So a component might be a class or it might be a collection of classes.
- A package diagram is a mechanism for you to group together related UML items, think of it as acting like (file system) directory/folder for UML.

When to use Package Diagram ???

- A large complex project can hundred of classes. Without some way to organization those classes. It becomes impossible to make sense of tem all.
- Packages create a structure for you classes or other Uml element by grouping related elements.


Maintenace
transaction
withdrawcash
deposit funds
transfer funds
Check Balance


ATM
transaction
+ With draw cash
+ check Balance
+ transfer Funds
+ deposite Funds
Maintance
+ Login
+ problem


Student Activities
Student
Public
Faculty
Faculty Activities
Registrar
Administration
Search