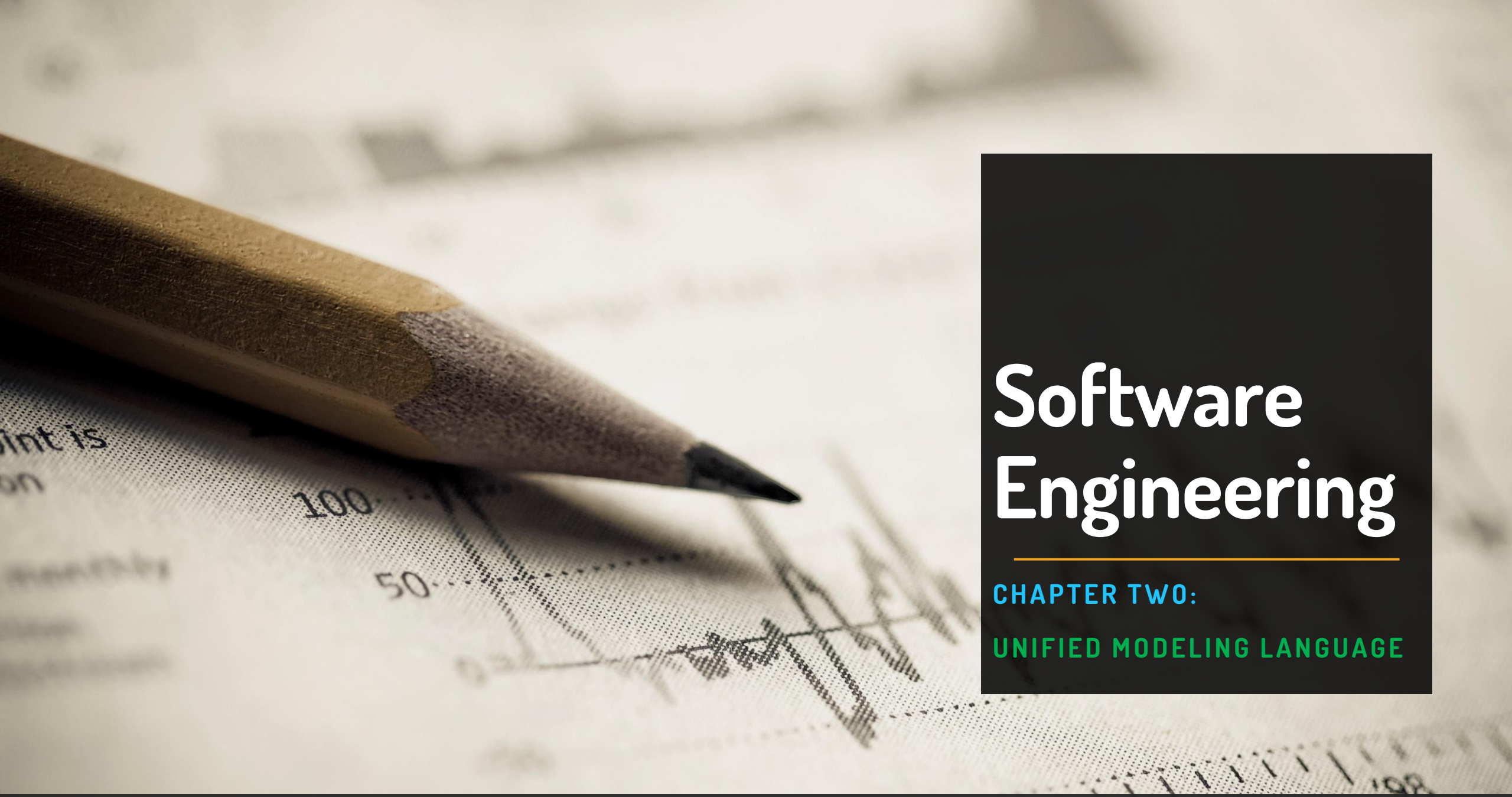


Software Engineering

CHAPTER TWO:

UNIFIED MODELING LANGUAGE

Outline

- UML Introduction
- Use Case Diagram
- Class Diagram
- Object Diagram
- Activity Diagram
- Interaction Diagrams
  - Sequence Diagram
  - Collaboration Diagram
- State Chart Diagram
- Component Diagram
- Deployment Diagram

Intro: Unified Modeling Language (UML)

- **Notations** enable us to ***articulate*** and ***communicate*** complex ideas ***succinctly*** and ***precisely***.

- For a notation to make accurate communication, it must come with a ***well-defined semantics***, it must be ***well suited*** for representing a ***given aspect of a system***, and it must be ***well understood among project participants***.

- With ***standards*** and ***conventions***: When a ***notation*** is used by a large number of participants, there is little room for ***misinterpretation*** and ***ambiguity***.

Intro: Unified Modeling Language (UML)

- **UML** is ***Convergence*** of different ***notations*** used in object-oriented methods, mainly
  - **Object-Modeling Technique (OMT)** (James Rumbaugh and collegues) ***and***, **OOSE** (Ivar Jacobson), Booch (Grady Booch)
- They also developed the **Rational Unified Process**, which became the Unified Process in 1999

25 year at GE Research, where he developed OMT, joined (IBM) Rational in 1994, CASE tool OMTool

At Ericsson until 1994, developed use cases and the CASE tool Objectory, at IBM Rational since 1995, http://www.ivarjacobson.com

Developed the Booch method ("clouds"), ACM Fellow 1995, and IBM Fellow 2003 http://www.booch.com/

Intro: Unified Modeling Language (UML)

- UML provides a wide variety of notations (***standard diagrammatic tool***) for ***modeling*** many aspects of software systems
- It is a graphical language for ***specifying***, ***visualizing***, ***constructing*** and ***documenting*** the ***artefacts of software systems.***
- Current Version: UML 2.5: http://www.uml.org/
- Commercial tools:
  - Rational (IBM),Together (Borland), Visual Architect (Visual Paradigm), Enterprise Architect (Sparx Systems)
- Open Source tools http://www.sourceforge.net/
  - ArgoUML, StarUML, Umbrello (for KDE), PoseidonUML
- Example of research tools: Unicase, Sysiphus
  - Based on a unified project model for modeling, collaboration and project organization: http://unicase.org and http://sysiphus.in.tum.de/

Intro: Unified Modeling Language (UML)

- In software engineering, ***modeling*** allows all individuals involved in a project to maintain a ***common understanding of the product being developed***.
- **Modeling** is at the ***heart of analysis and design*** allowing us to "***describe the structure and behavior both of real-world phenomena and of the abstractions to be built in a software product***" ***(Lee 2013).***
- A **model** can be broken down into ***three parts:*** ***(D'Sourza and Wills 1999):***

1. ***Static Part:*** Describes an ***object's state at any given moment***.
2. ***Dynamic Part:*** Represents the ***state changes that will happen as events occur.***
3. ***Interactive part:*** Used to denote ***how objects interact with one another.***

Intro: Unified Modeling Language (UML)

Traffic Light System: The following example shows how a model represents an object's state, how it changes over time, and how it interacts with other objects in a system.

1.Static Part:

1. The traffic light has **three possible states**: **Red** **Yellow** **Green**
2. At any given moment, the light is in one of these states.

2.Dynamic Part:

1. The traffic light **changes** from **Red** → **Green** → **Yellow** → **Red** in a cycle.
2. These **changes** occur based on a **timer** or **sensor inputs.**

3.Interactive Part:

1. The traffic light **interacts** with ***vehicles*** **and** ***pedestrians***.
2. When the light is **Red**, **vehicles stop**, and **pedestrians can cross.**
3. When the light is **Green**, **vehicles move**, and **pedestrians must wait.**

Intro: Unified Modeling Language (UML)

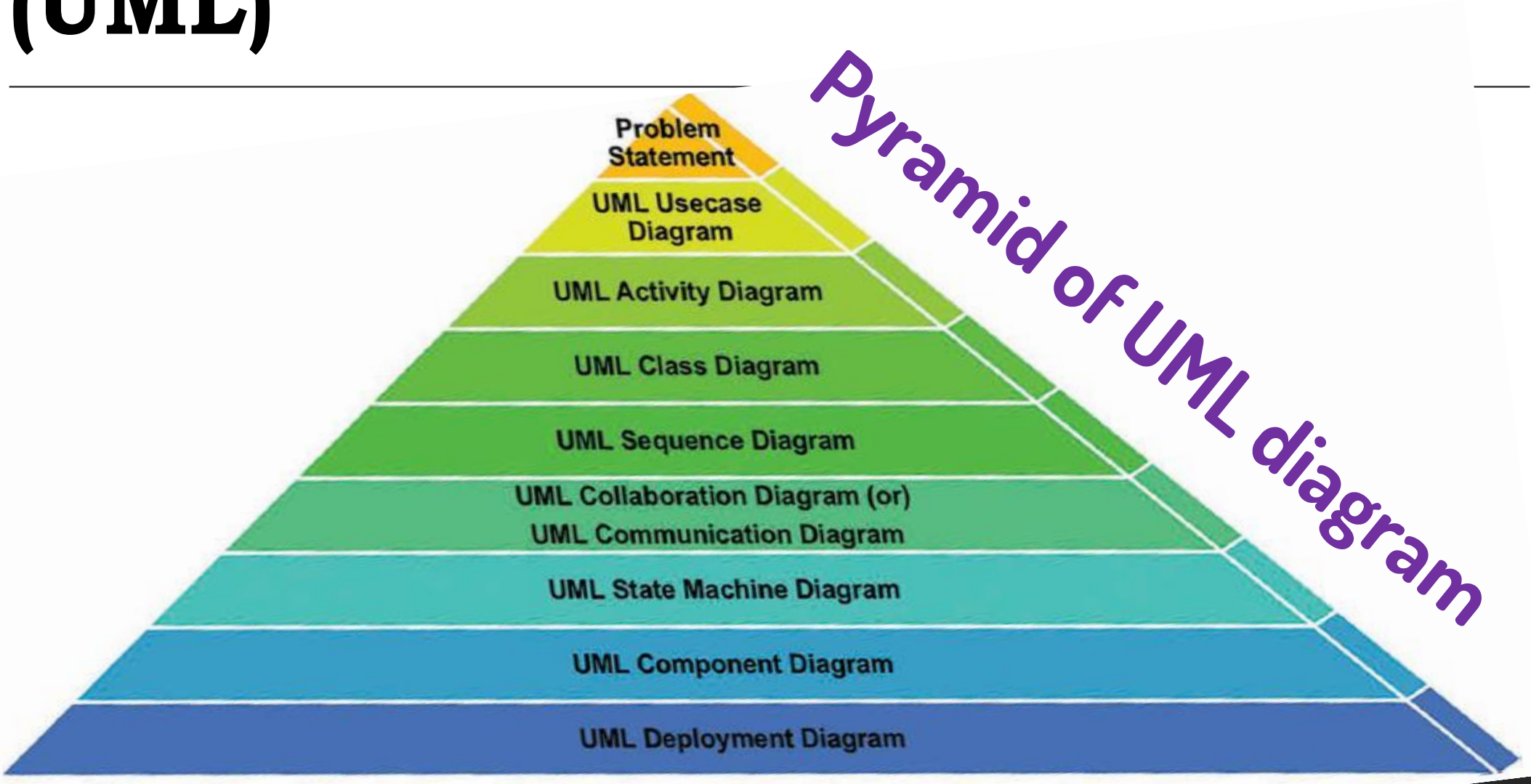
- System development focuses on ***three different models*** of the system:

1. The **functional model**, represented in UML with use **case diagrams**, describes the ***functionality*** of the system from the ***user's point of view.***
2. The **object model**, represented in UML with **class diagrams**, describes the ***structure of a system*** in terms of ***objects, attributes, associations***, and ***operations***.
3. The **dynamic model**, represented in UML with **sequence diagrams, state chart diagrams**, and **activity diagrams**, describes the ***internal behavior of the system.***

Intro: Unified Modeling Language (UML): *Example: Online Shopping System*

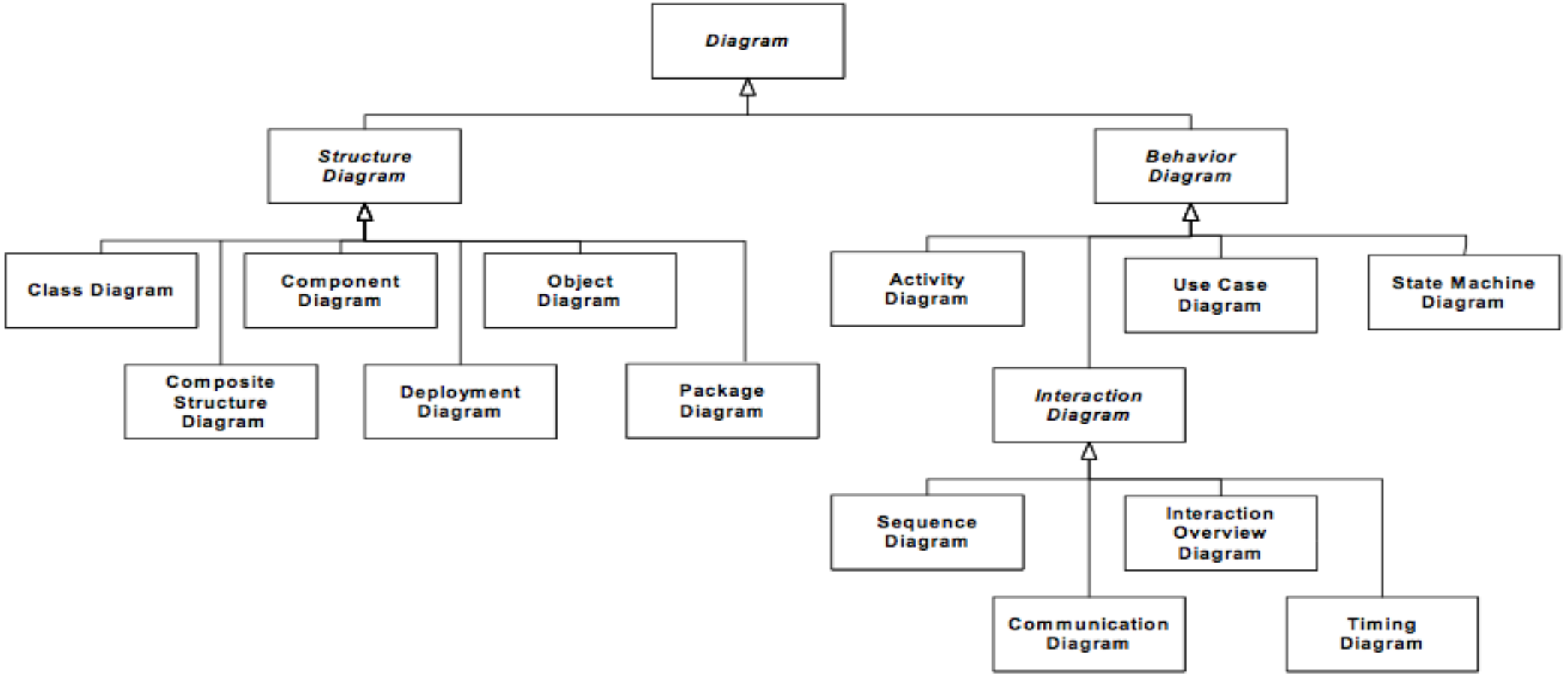
1. **Functional Model**:
   1. Defines **key processes** such as **User Registration, Product Search, Adding Items to Cart, and Checkout**.
   2. Shows **how data moves between these functions** (e.g., ***user details flow into the registration process***).
2. **Object Model:**
   1. Defines **key objects** such as **User, Product, Cart, and Order.**
   2. Shows **relationships**:
      - A **User** can place multiple **Orders**.
      - An **Order** consists of multiple **Products**.
3. **Dynamic Model:**
   1. Describes **how objects interact:**
      - When a **User** ***adds a product*** to the cart, the **Cart** ***state updates.***
      - When the **User** ***makes a payment***, the **Order** ***status changes*** from ***"Pending"*** to ***"Confirmed".***
      - The **System** ***sends an order confirmation email***

Intro: Unified Modeling Language (UML)



Intro: Unified Modeling Language (UML)

Another view of UML diagrams



Unified Modeling Language (UML):

UML Use Case Diagram

Use cases are used during ***requirements elicitation*** and ***analysis*** to represent the ***functionality of the system***.

Use cases focus on the ***behavior of the system from an external point of view.*** **Use case** is a ***visual representation of a distinct business functionality provided by a system*** that yields a visible ***result of value for one or more actors.***

- It ensures that the business process is discrete in nature
- List the **discrete business functions** in given **problem statement**.
- They are **verbs describing an action**

Unified Modeling Language (UML):

UML Use Case Diagram

Two kinds of flows of events are associated with **use cases**. These are as follows:

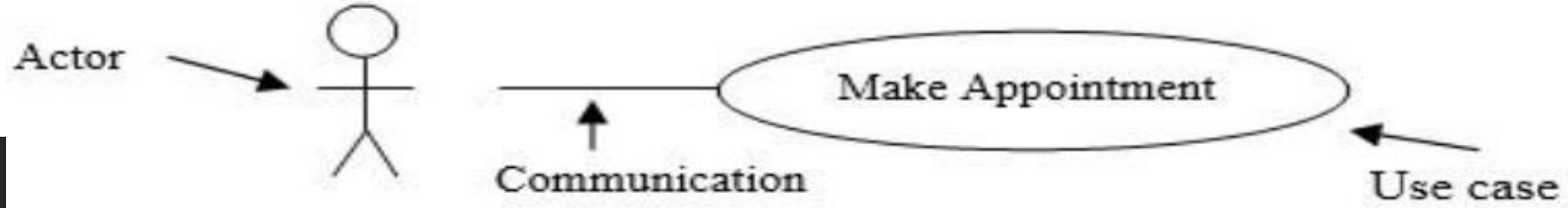
1. **The main flow of events** (***basic course of action***) is the **sunny-day scenario**-> "***the main start-to-finish path that the actor and the system follow under normal circumstances***." The ***assumption is that the primary actor doesn't make any mistakes***, and ***the system generates no errors***. **A use case *always has a main flow of events.***
2. **An exceptional flow of events** (***alternate course of action***) is a ***path through a use case that represents an error condition*** or a path that the actor and the system take less frequently. **A use case often has at least one exceptional flow of events.**

Unified Modeling Language (UML):

UML Use Case Diagram

Actor describes any ***external entity that interacts with the system*** (e.g., a ***user, another system, the system's physical environment***). Actors have ***unique names*** to properly ***identify*** them within the system. An ***actor interacts*** with a ***use case/s***

- The following Figure shows ***relationship*** that an ***actor*** has with a ***system***. For example, the **oval** ***'Make Appointment'*** represents a ***use case*** that happens in ***response*** to an ***externally initialized event.*** The ***event in question*** is the ***line connecting*** the ***actor***, such as a ***patient, to the use case.*** *The Figure illustrates a patient using the system to make an appointment.*



Unified Modeling Language (UML):

UML Use Case Diagram

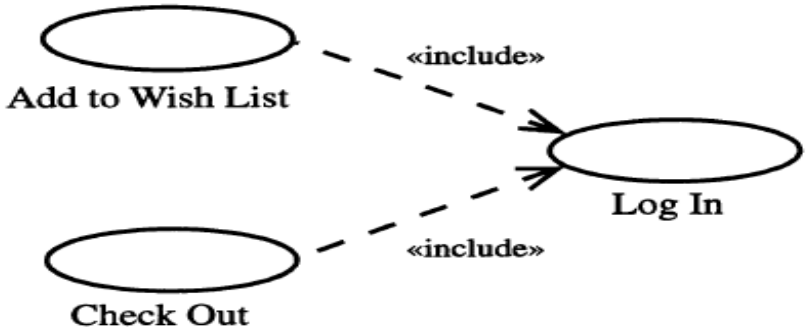
- The ***identification of actors and use cases*** results in the definition of the **boundary of the system,** that is, in ***differentiating the tasks accomplished by the system*** and ***the tasks accomplished by its environment.***
- It represents the scope of the system. It encapsulates the complete set of functionalities of the system.
- The **actors** are ***outside the boundary of the system***, whereas the **use cases** are ***inside the boundary of the system.***

Unified Modeling Language (UML):
UML Use Case Diagram:
Relationships

- **Uses/Include** *and* **Extends** are **two types** of relationships used in **use case diagrams.**
- **Uses** specifies that the "***semantics of the source element are dependent upon the semantics of the public portion of the target element***" (Lee 2013).
- **Extends** specifies that "the ***target use case adopts the behavior*** of the ***source use case***" (Lee 2013).
- Both involve ***selecting a common behavior from several use cases*** and ***using them in a single-use case.*** This is then ***used or extended*** by other use cases.

Unified Modeling Language (UML):
UML Use Case Diagram:
Relationships

1. **Uses/Include:** one use case ***EXPLICITLY*** **includes** the ***behavior of another use case*** at a specified point within a course of action. The ***included use case doesn't stand alone***, it has to be connected with one or more base use cases. The ***included use case is a part of the main use case*** and ***is essential for its execution.*** In the following, the ***Add to Wish List*** and ***Check Out use cases*** **include** the ***behavior*** captured within the ***Log In use case,*** because a ***Customer must be logged in before he or she can add a book to a wish list or make a purchase. It means Log In is a mandatory part of Adding to Wish List or Checking Out.***



Unified Modeling Language (UML):
UML Use Case Diagram:
Relationships

2. **Extends:** base use case ***IMPLICITLY*** **includes** the ***behavior of another use case*** at one or more specified points. These points are called ***extension points.***

   ▪**Extends** is used to show that ***one use case (the extending use case) adds additional functionality to another use case (the base use case) under specific conditions. The use case to be extended from is optional*** and This relationship is represented by a **dashed arrow** with the label ***<<extend>> pointing from the extending use case to the base use case.***

   ▪Let's consider an **Online Shopping System** where a customer can place an order. The base use case is "**Place Order**", and there are ***optional behaviors*** that may extend this use case, such as "**Apply Discount**" and "**Validate Payment**".

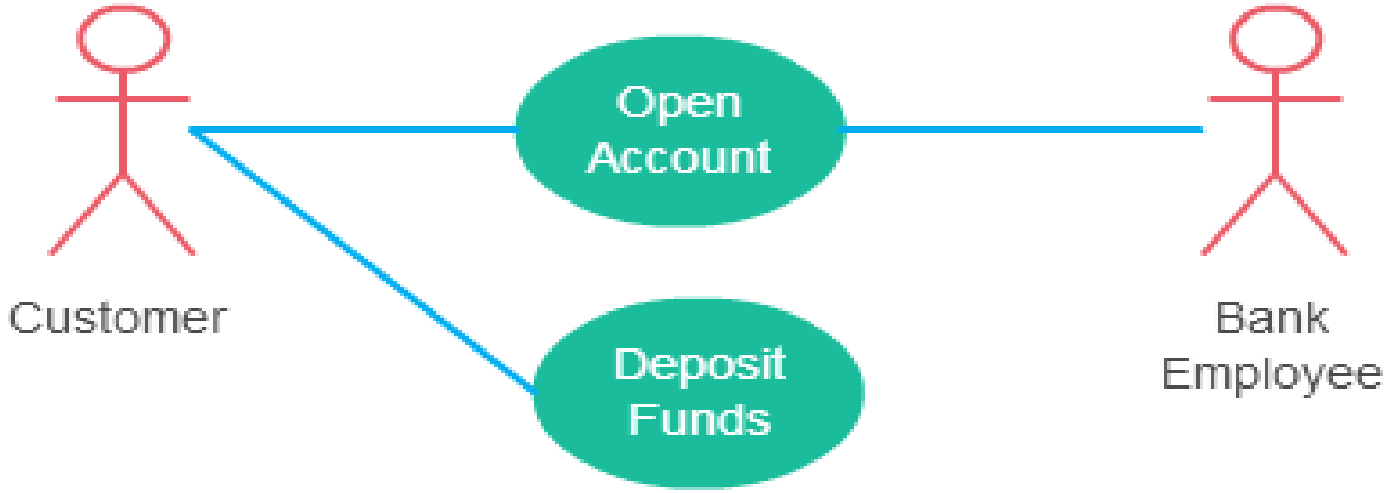
Unified Modeling Language (UML):
UML Use Case Diagram:
Relationships

1. **Base Use Case: Place Order**
    1. This is the primary use case where the ***customer places an order.***
    2. It includes steps like ***selecting items, entering shipping details,*** *and* ***confirming the order.***
2. **Extending Use Case: Apply Discount**
    1. This use case extends **Place Order** when the customer applies a discount code.
    2. It adds the behavior of validating the discount code and applying the discount to the total order amount.
3. **Extending Use Case: Validate Payment**
    1. This use case extends **Place Order** when the payment validation fails.
    2. It adds the behavior of re-validating the payment details or notifying the customer of the failure.

Unified Modeling Language (UML):

UML Use Case Diagram

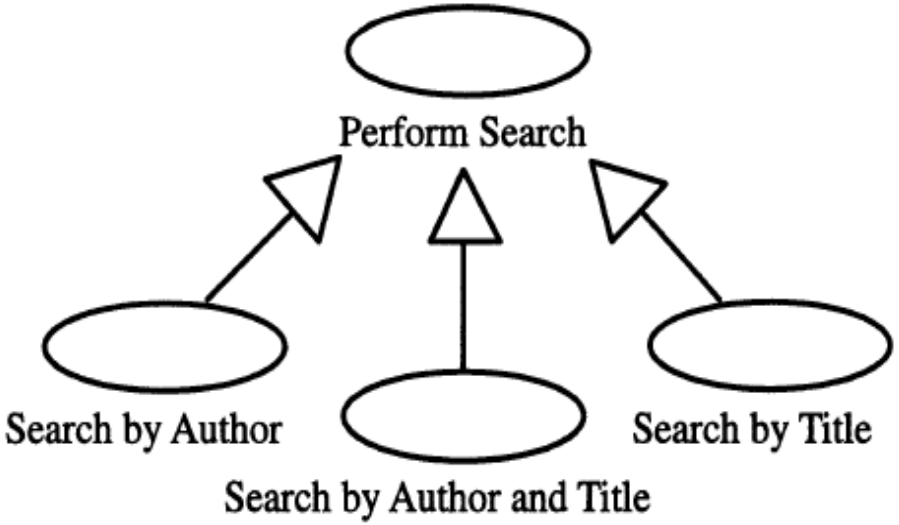
- **Association Between Actor and Use Case:** An **actor** must be ***associated with at least one use case***. An **actor** can be ***associated with multiple use cases.*** ***Multiple actors*** can be ***associated with a single use case.***



Unified Modeling Language (UML):

UML Use Case Diagram

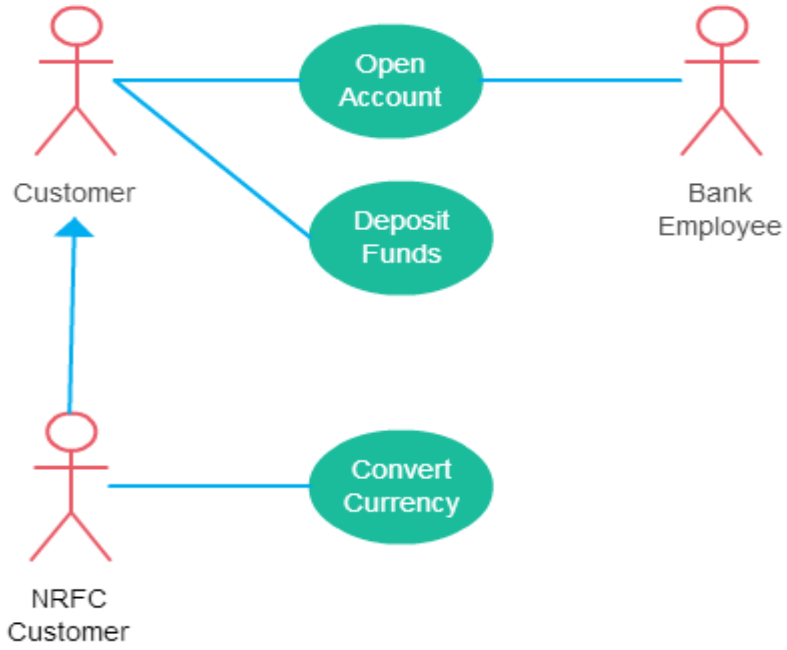
- **Use Case Generalization:** is the ***relationship*** where a ***parent use case*** defines ***behavior*** that its ***children use case*** can **inherit** and the children can ***add to*** or ***override that behavior.*** This is used when there is ***common behavior between two use cases*** and also ***specialized behavior specific to each use case.***



Unified Modeling Language (UML):

UML Use Case Diagram

- **Actor Generalization:** means that one **actor** can ***inherit*** the ***role*** of the **other actor.** The ***descendant inherits all the use cases of the ancestor***. The descendant has one or more use cases that are specific to that role.



Unified Modeling Language (UML):

UML Use Case Diagram:

Use Case Template

Use case Template: Every use case is explained in detail through a standard template called as **use case template.**

Use case ID	represents a unique ID for a use case.
Use Case Name	represents a meaningful name of the use case
Actors Involved	actors interacting with the use case
Description	the brief explanation of the functionality of the use case.

Unified Modeling Language (UML):

UML Use Case Diagram:

Use Case Template

Entry/Pre-Conditions	state of the system before executing this functionality of the use case OR conditions that need to be satisfied before the use case is initiated
Main Flow	description of how the functionality is implemented
Exit/Post Conditions	state of the system after executing this functionality of the use case OR the conditions that are satisfied after the completion of the use case
Alternate Flow	other possibilities available

Unified Modeling Language (UML):

UML Use Case Diagram:

Use Case Guidelines

Use Case Guidelines: Actors

- **Give meaningful business relevant names for actors**
- **Primary actors should be to the left side of the diagram** – This enables you to quickly highlight the important roles in the system.
- **Actors model roles (not positions)** – In a hotel both the front office executive and shift manager can make reservations. So, something like "**Reservation Agent**" should be used for actor name to highlight the role.
- **External systems are actors** – If your use case is send-email and if interacts with the email management software then the software is an actor to that particular use case.
- **Actors don't interact with other actors** – In case actors interact within a system you need to create a new use case diagram with the system in the previous diagram represented as an actor.
- **Place inheriting actors below the parent actor** – This is to make it more readable and to quickly highlight the use cases specific for that actor.

Unified Modeling Language (UML):

UML Use Case Diagram:

Use Case Guidelines

Use Case Guidelines: Use Cases

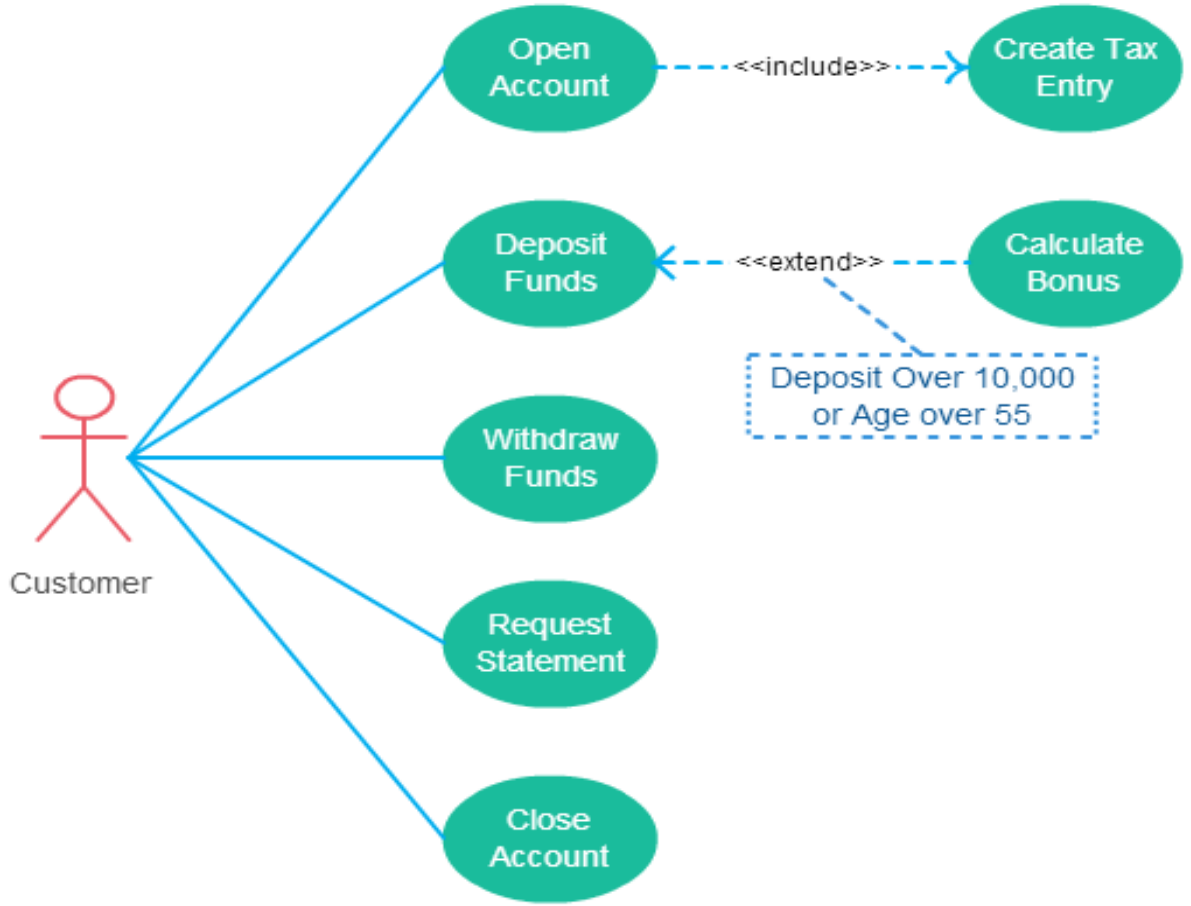
- **Names begin with a verb** – A use case models an **action** so the name should begin with a verb.
- **Make the name descriptive** – This is to give more information for others who are looking at the diagram. For example, "***Print Invoice***" is better than "***Print***".
- **Highlight the logical order** – For example, if you're analyzing a bank customer typical use cases include ***open account***, ***deposit*** and ***withdraw.*** Showing them in the logical order makes more sense.
- **Place included use cases to the right of the invoking use case** – This is done to improve readability and add clarity.
- **Place inheriting use case below parent use case** – Again this is done to improve the readability of the diagram.

Unified Modeling Language (UML):

UML Use Case Diagram:

Use Case Guidelines

Use Case Guidelines: Use Cases



Unified Modeling Language (UML):
UML Use Case Diagram:
Use Case Guidelines

Use Case Guidelines: Relationships

- Arrow points to the base use case when using <<extend>>
- <<extend>> can have optional extension conditions
- Arrow points to the included use case when using <<include>>
- Both <<extend>> and <<include>> are shown as dashed arrows.
- Actor and use case relationship don't show arrows.

Unified Modeling Language (UML):

UML Use Case Diagram:

Example: "Online Shopping System"

- Use Case ID:
  - *OSP00001*
- Use Case Description:
  - *Place Order in Online Shopping System*
- Use Case Name:
  - *Place Order*
- Actors:
  - *Customer, Online Shopping System, Payment Gateway*
- Preconditions:
  - *The customer is logged in.*
  - *The shopping cart has at least one item.*
- Postconditions:
  - *The order is successfully placed.*
  - *The customer receives an order confirmation.*

Unified Modeling Language (UML):

UML Use Case Diagram:

Example: "Online Shopping System"

- **Main Flow (Basic Flow of Events):**

1. *The customer selects products and adds them to the cart.*
2. *The customer proceeds to checkout.*
3. *The system displays the order summary, including product details, quantity, and total price.*
4. *The customer enters or selects the shipping address.*
5. *The system calculates shipping costs and updates the total amount.*
6. *The customer selects a payment method (credit card, PayPal, Telebirr, etc.).*
7. *The system redirects the customer to the payment gateway.*
8. *The customer enters payment details and confirms the payment.*
9. *The payment gateway processes the payment and sends a confirmation to the system.*
10. *The system confirms the order and updates the order status.*
11. *The system sends an order confirmation email to the customer.*

Unified Modeling Language (UML):

UML Use Case Diagram:

Example: "Online Shopping System"

- Alternate Flow (Alternative Scenarios):
  - *A1: Customer Cancels the Order*
    - *At step 2, the customer decides to cancel the checkout process.*
    - *The system discards the order, and the cart remains unchanged.*
  - *A2: Payment Fails*
    - *At step 9, if the payment fails, the system displays an error message.*
    - *The customer can retry payment or choose a different payment method.*

Unified Modeling Language (UML):

UML Use Case Diagram:

Example: "Online Shopping System"

- Extensions (Optional Behaviors):
  - *E1: Apply Discount (Extends "Place Order")*
    - *If the customer has a valid discount code, they can enter it at checkout.*
    - *The system verifies the code and applies the discount.*
    - *The total price updates accordingly.*
  - *E2: Validate Payment (Extends "Place Order")*
    - *Before processing, the system validates the payment details.*
    - *If incorrect details are entered, the system prompts the customer to correct them.*

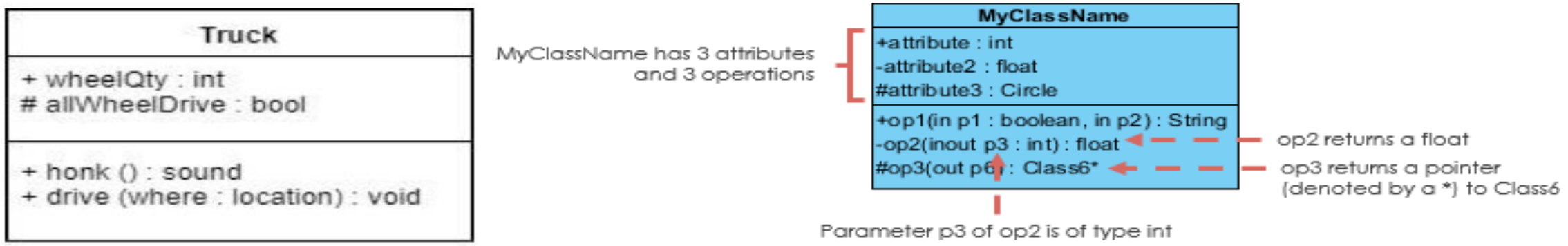
Unified Modeling Language (UML):

UML Class Diagram

- **Class diagram:** is a type of ***static structure diagram*** that describes the ***structure of a system*** by showing the system's:
  - ***classes-*** start with an ***uppercase letter*** and must be ***noun*** or ***noun phrase,***
  - ***their attributes-*** are ***data describing the class. Attributes map onto member variables (data members) in code.*** The ***data type*** of attribute is shown ***after the colon.***
  - ***operations (methods)-*** They are ***services*** the class provides. The ***return type*** of a method is shown ***after the colon.*** The ***return type of method parameters*** are shown ***after the colon following the parameter name. Operations map onto class methods in code*** and ***onto the relationships among objects.***

Unified Modeling Language (UML):

UML Class Diagram



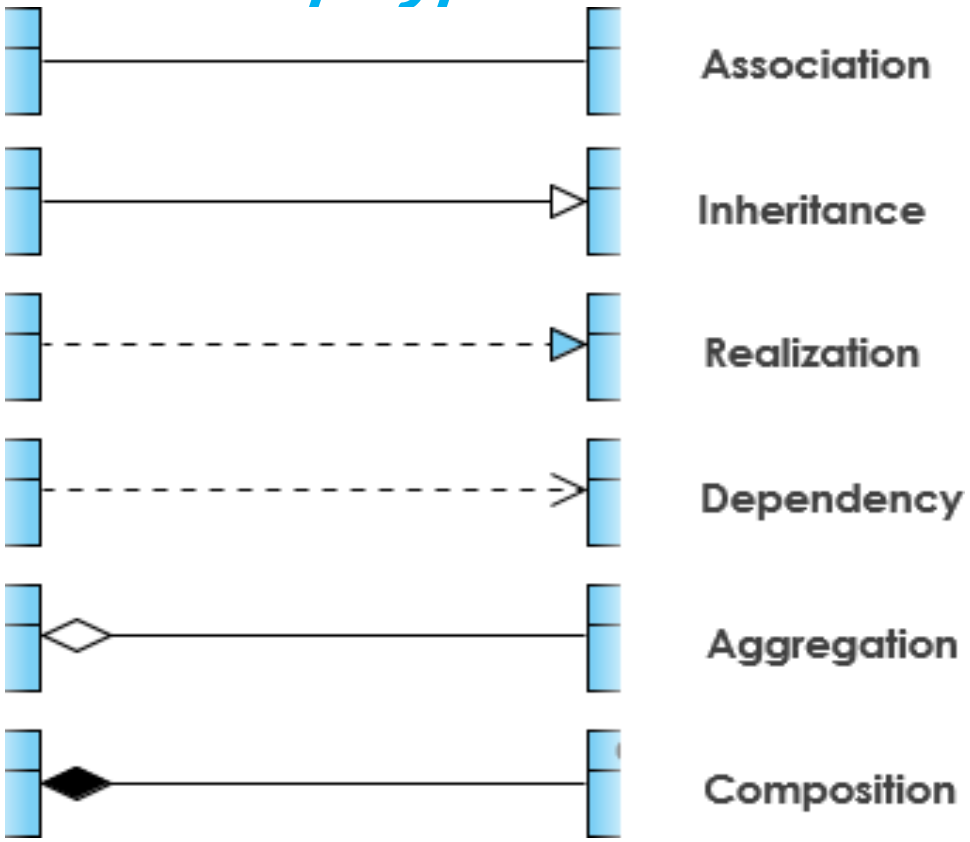
- **Class diagram:** The class diagram representing Truck lists important information about the class.
  - **Top box** holds the ***class name***,
  - **Middle box** holds the ***class attributes***, and
  - **Bottom box** holds the ***class methods.***
- There are several symbols used to represent critical information about the attributes and methods.
  - '**+**' symbol denotes a ***public attribute/method.***
  - '**#**' symbol denotes a ***protected attribute/method.***
  - '**-**' symbol denotes a ***private attribute/method.***

Unified Modeling Language (UML):
UML Class Diagram:
Relationships

- **UML is not just about pretty pictures.**

If used correctly, UML precisely conveys ***how code should be implemented from diagrams***. If precisely interpreted, ***the implemented code will correctly reflect the intent of the designer.***

Relationship Types:

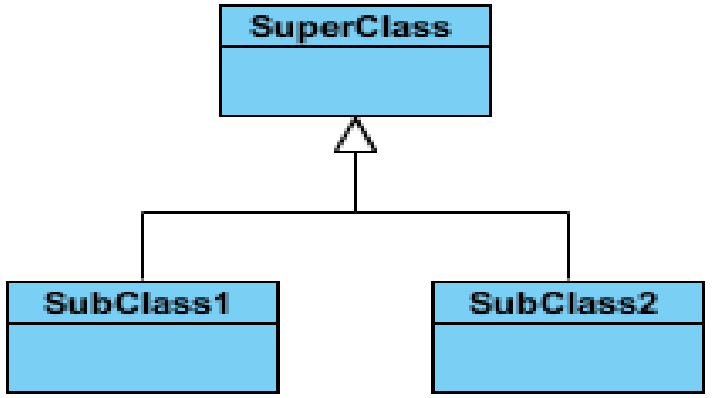


Unified Modeling Language (UML):
UML Class Diagram:
Relationships

1. **Inheritance (or Generalization):** is a relationship between a *more general classifier(super or parent class)* and a *more specific classifier(subclass or child)*. *Each instance of the specific classifier is also an indirect instance of the general classifier.* Thus, the *specific classifier inherits the features* of the more *general classifier. Generalization* enables us to *describe all the attributes and operations that are common to a set of classes.*

- Represents an *"is-a"* relationship.

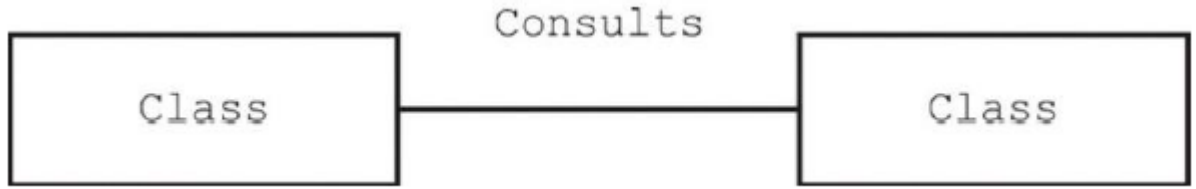
The relationship is displayed as a solid line with a **hollow arrowhead** *that points from the child element to the parent element.*



Unified Modeling Language (UML):
UML Class Diagram:
Relationships

2. **Associations:** are represented by a ***solid line between classes.*** They are typically ***named using a verb or verb phrase*** which ***reflects the real-world problem domain.***

   - Instances of classes involved in an association will most likely be ***communicating with each other at program execution time,*** but all we're concerned with here is the fact that these ***instances have some attachment to each other***. It is ***structural connection between classes called link.***
   - **Simple/binary Association:** A structural link between two ***peer classes***.



Unified Modeling Language (UML):
UML Class Diagram:
Relationships

2. **Associations: … Cardinality/Multiplicity:** defined by the ***number of possible associations that a class can have*** expressed in terms of ***one to one***, ***one to many*** and ***many to many.***
   - **Multiplicity** defines the "***number of instances of an individual class that may be associated with a single instance of another class***".
   - ***Multiplicity is denoted by a single digit***, signifying the exact ***number of relationships that an entity will have, or it will use the form "x:y".***
   - ***x and y,*** in this case, represent the number of relationships that fall between the range of "x" to "y".

Unified Modeling Language (UML):
UML Class Diagram:
Relationships

2. Associations: … Cardinality/Multiplicity…

Multiplicity	Meaning
0..1	0 or 1 instance.
0..*	0 to infinite instance. (There is no limit of the number of instances.)
1	Exactly 1 instance.
1..*	At least 1 instance. (There must be a minimum of 1 instance.)

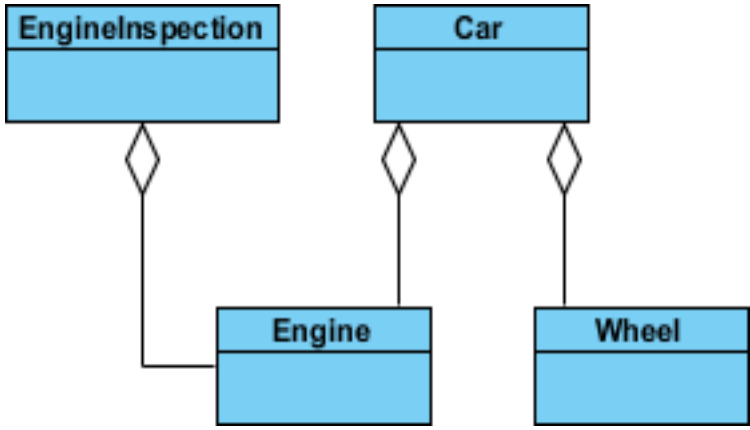
Unified Modeling Language (UML):
UML Class Diagram:
Relationships

3. **Aggregation:** It represents a "**part of**" relationship. ***Class2*** is part of ***Class1***.
   - ***Many instances (denoted by the *)*** of ***Class2*** can be associated with ***Class1*.**
   - Objects of ***Class1*** and ***Class2*** have ***separate lifetimes.***
   - The relationship is displayed as a ***solid line with unfilled diamond*** at the association end, which is ***connected to the class that represents the aggregate.***
   - ***It is a relationship of*** where the ***child class can exist independently of the parent class.***

Unified Modeling Language (UML):
UML Class Diagram:
Relationships

3. **Aggregation:...** *This example shows that the car always needs a wheel and an engine to operate/function correctly, whereas the engine and wheel do not always need a car. They can be used with a bus, a bike, machines, lawnmower, and much more.*

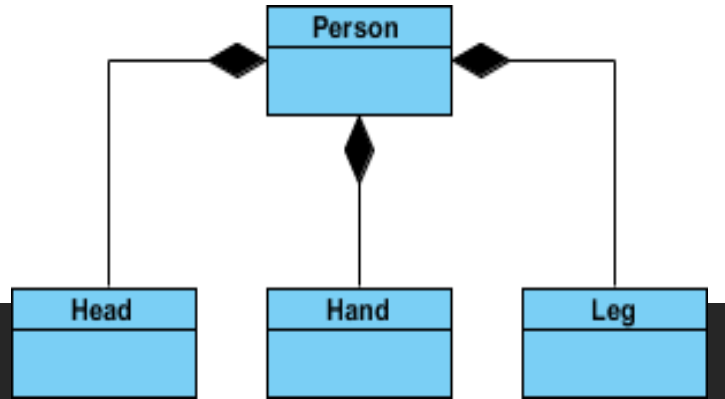
   Similarly, the engine is necessary for an engine inspection, but the engine can exist as an independent entity and be associated with other vehicles as well.



Unified Modeling Language (UML):
UML Class Diagram:
Relationships

4. **Composition:** A special ***type of aggregation*** where ***parts are destroyed*** when the ***whole is destroyed.*** **Objects of Class2 live and die with Class1. Class2 cannot stand by itself.** It represents a '**whole**' relationship.

   - The relationship is displayed as a ***solid line with a filled diamond*** at the association end, which is connected to the class that represents the whole or composite.



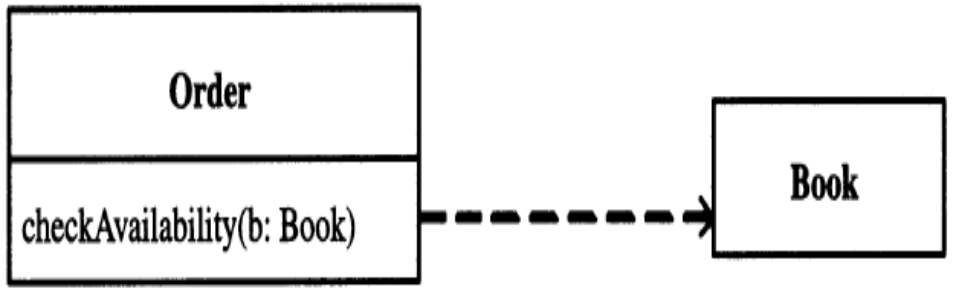
Unified Modeling Language (UML):
UML Class Diagram:
Relationships

5. **Dependency:** ***is special type of association*** where ***one entity depends on or requires the resources of another.*** ***For example, A flashlight needs a battery to function.***

   - It is ***when the existence of one class is dependent on the existence of another class. Changes to the definition of one may cause changes to the other (but not the other way around).***

   - The relationship is displayed as a ***dashed line with an open arrow.*** The arrow originates from the dependent entity and points towards the independent entity.

If the definition of the Book class changes, the way that the checkAvailability function works may have to change as well.



Unified Modeling Language (UML):
UML Class Diagram:
Relationships

6. **Interface and Role of Realization**: Interfaces can be of two types namely ***required and provider interfaces.***

   - ***Provider interface describes the functionality offered by a class. Provided interface*** is the interface implemented by a class, ***i.e a class implements an interface.***

   - ***Required interfaces describe the functionality needed by another class.*** The ***required interface*** would be any use of an interface by a component, ***i.e if a class defines a method that has the interface as a parameter.***

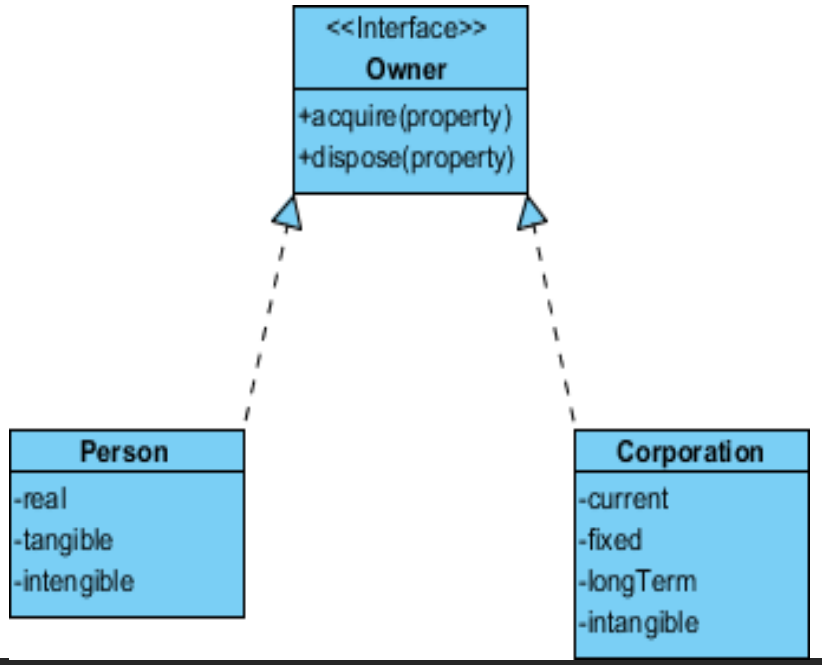
- **Realization** is a relationship between the ***blueprint class and the object containing its respective implementation level details.***

Unified Modeling Language (UML):
UML Class Diagram:
Relationships

6. **Realization:** ...This object is said to realize the blueprint class. **OR** , it is ***relationship between the interface and the implementing class.***

For example:, the Owner interface might specify methods for acquiring property and disposing of property.

The Person and Corporation classes need to implement these methods, possibly in very different ways.



Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

Step 1: Identify Key Classes: the main classes could be:

1. **Customer** *(stores customer details)*
2. **Order** *(manages orders)*
3. **ShoppingCart** *(handles items before purchase)*
4. **Product** *(represents items available for sale)*
5. **Payment** *(handles payment transactions)*
6. **ShippingAddress** *(to where the product should be delivered)*

Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

Step 2: Define Attributes & Methods for Each Class

1. **Customer:**
    1. *Attributes: customerId, name, email, address*
    2. *Methods: register(), login(), placeOrder()*
2. **Product:**
    1. *Attributes: productId, name, price, description, stockQuantity*
    2. *Methods: updateStock()*

Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

Step 2: Define Attributes & Methods for Each Class

3. **Order**:
    1. *Attributes: orderId, orderDate, totalAmount, status*
    2. *Methods: calculateTotal(), updateStatus()*
4. **ShoppingCart**:
    1. *Attributes: cartId*
    2. *Methods: addItem(), removeItem(), calculateTotal()*

Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

Step 2: Define Attributes & Methods for Each Class

5. **Payment**:
    1. *Attributes: paymentId, paymentDate, amount, paymentMethod*
    2. *Methods: processPayment()*
6. **ShippingAddress**:
    1. *Attributes: addressId, street, city, state, zipCode*
    2. *Methods: validateAddress()*

Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

Step 3: Identify Relationships Between Classes:

1. Customer *places an* Order
   - A Customer can place multiple Orders, but each Order belongs to only one Customer *(1-to-many relationship).*
2. Order *contains multiple* Product *items*
   - An Order can contain multiple Products, and a Product can be part of multiple Orders *(many-to-many).*
3. ShoppingCart *is associated with a* Customer
   - Each Customer has one ShoppingCart, and a ShoppingCart belongs to one Customer (1-to-1). *(1-to-1 relationship).*

Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

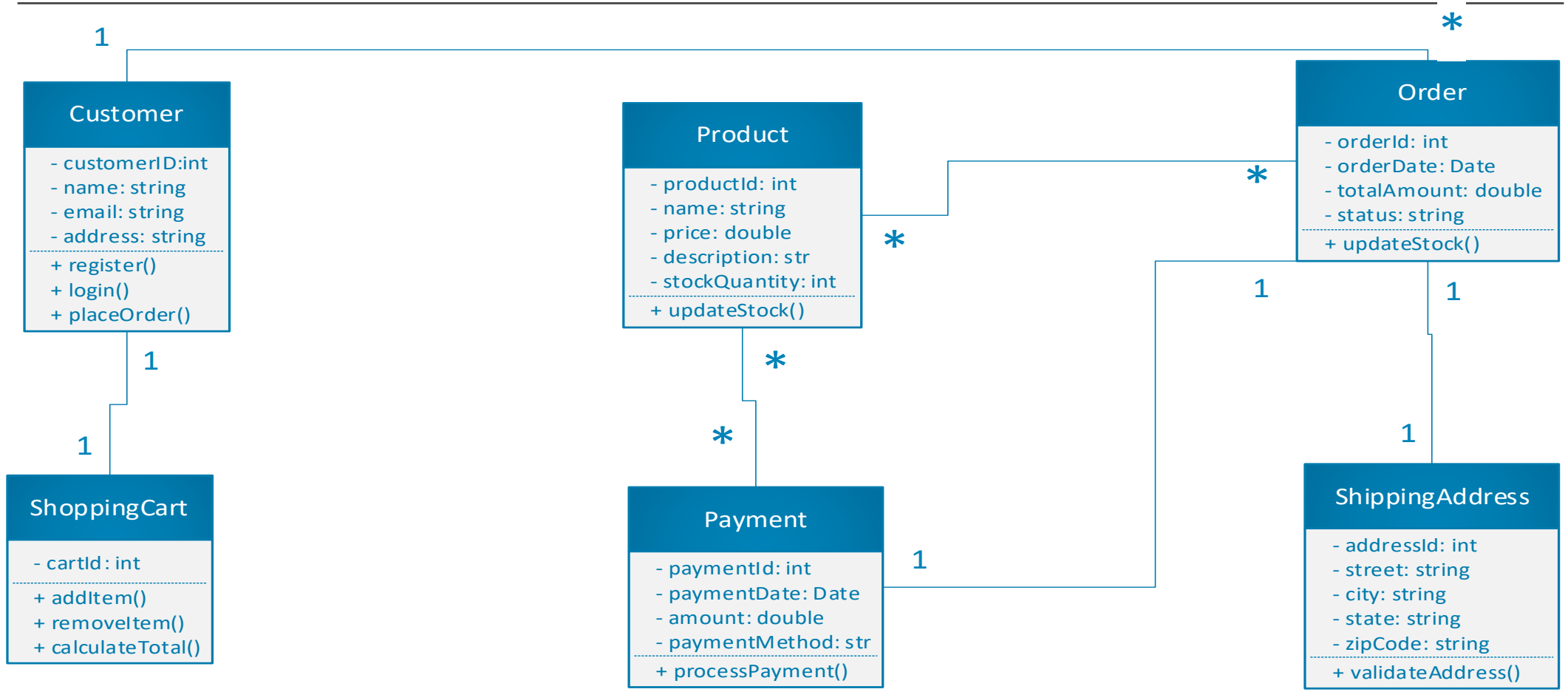
Step 3: Identify Relationships Between Classes:

4. ShoppingCart *contains multiple* Product *items*
   - A ShoppingCart can contain multiple Products, and a Product can be in multiple ShoppingCarts *(many-to-many).*
5. Order *is associated with a* Payment
   - Each Order has one Payment, and a Payment belongs to one Order *(1-to-1).*
6. Order *is associated with a* ShippingAddress
   - Each Order has one ShippingAddress, and a ShippingAddress belongs to one Order *(1-to-1).*

Unified Modeling Language (UML):

UML Class Diagram:

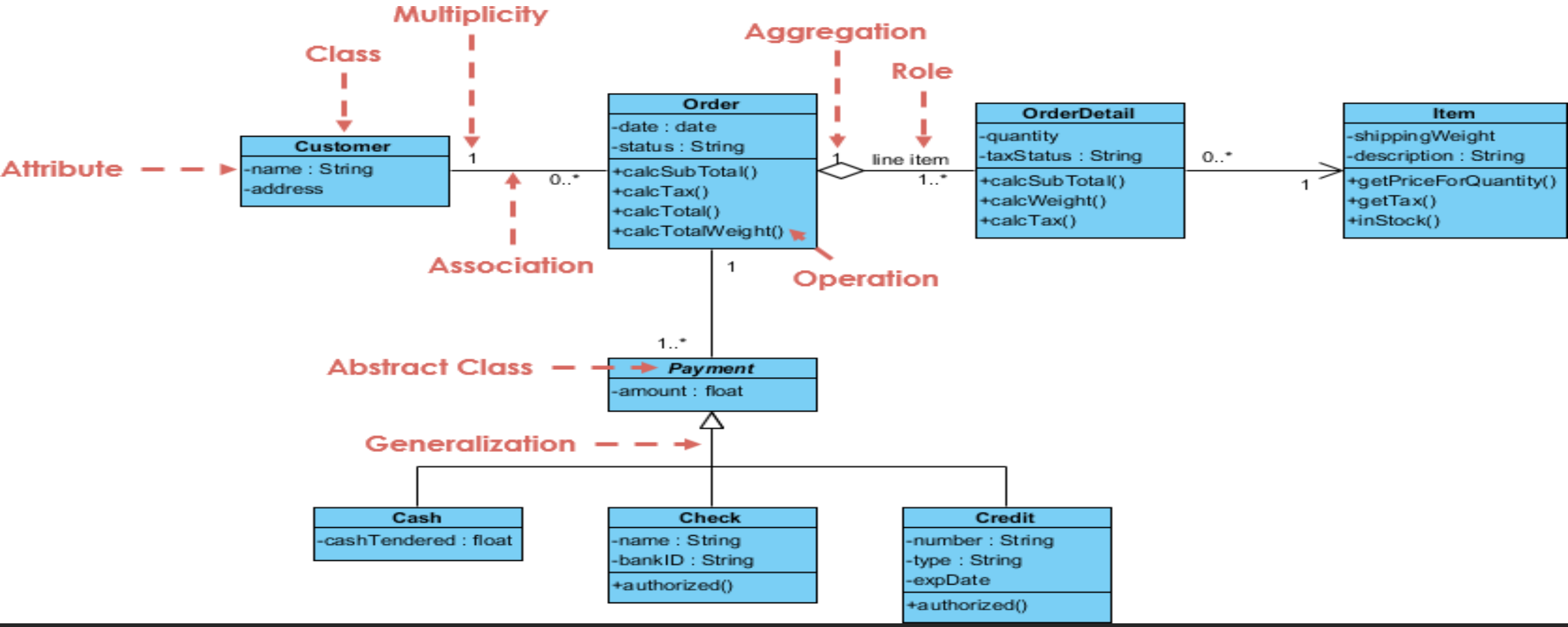
Example: "Online Shopping System"



Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"



Unified Modeling Language (UML):

UML Class Diagram:

Example: "Online Shopping System"

- **The Diagram in the previous slide:** depicts a ***customer order system*** centered around the **Order class**, which interacts with both the **Customer** and **Payment Class**.
- The **Order class** contains the ***OrderDetail*** class, which is **associated** with an **Item class** ***describing the actual item being ordered***, which contains information about a ***specific individual's order.***
- The **payment class** has three derived classes being ***Credit, Cash, and Check***