Advanced Programming (OOP)

Unified Modeling Language (UML)

SDB

Spring 2025

Unified Modeling Language (UML): Topics

Software Requirements Specification (SRS) I

Definition: A Software Requirements Specification (SRS) is a detailed document that describes the functional and non-functional requirements of a software system.

Importance of SRS:

- Serves as a contract between stakeholders and developers.
- Ensures clarity of requirements and avoids scope creep.
- Acts as a reference for validation and verification of the final product.
- Helps in planning, design, development, and maintenance.

Key Components:

- **Introduction:** Purpose, scope, and overview of the system.
- **Overall Description:** System environment, user characteristics, and constraints.
- **Functional Requirements:** Specific functionalities the system must support.

Software Requirements Specification (SRS) II

- **Non-Functional Requirements:** Performance, security, usability, and reliability aspects.
- **System Architecture:** High-level design and module interactions.
- **External Interfaces:** Interaction with hardware, software, and users.
- **Assumptions and Dependencies:** Any external factors influencing system behavior.

Use Case Example: Consider an online banking system where the SRS outlines functionalities like account management, fund transfers, and security constraints such as encryption and authentication protocols.

How UML Supports SRS

Relationship Between SRS and UML:

- UML diagrams help visualize and refine requirements defined in the SRS.
- Ensure that all system functionalities described in the SRS have a clear design representation.
- Improve communication between stakeholders by providing a structured, graphical representation of the system.

UML Diagrams Mapped to SRS Sections:

- **Use Case Diagram** - Represents system functionalities and user interactions.
- **Class Diagram** - Defines system structure and relationships.
- **Sequence Diagram** - Visualizes interactions between components.
- **Activity Diagram** - Maps workflows and process logic.
- **State Machine Diagram** - Models state transitions and behaviors.

Example: If an SRS describes a "User Authentication" feature, UML Use Case and Sequence Diagrams illustrate login interactions and authentication flows.

Introduction to UML

Definition: UML (Unified Modeling Language) is a standardized visual language for modeling software systems.

Key Features:

- Provides a blueprint for system architecture.
- Supports object-oriented design and software documentation.
- Helps in visualizing relationships between components.
- Enhances communication between stakeholders.
- Allows modeling of both static (structural) and dynamic (behavioral) aspects of a system.

Types of UML Diagrams:

- **Structural Diagrams:** Class, Object, Component, Deployment, Package, Composite Structure, Profile.
- **Behavioral Diagrams:** Use Case, Activity, State Machine, Sequence, Communication, Interaction Overview, Timing.

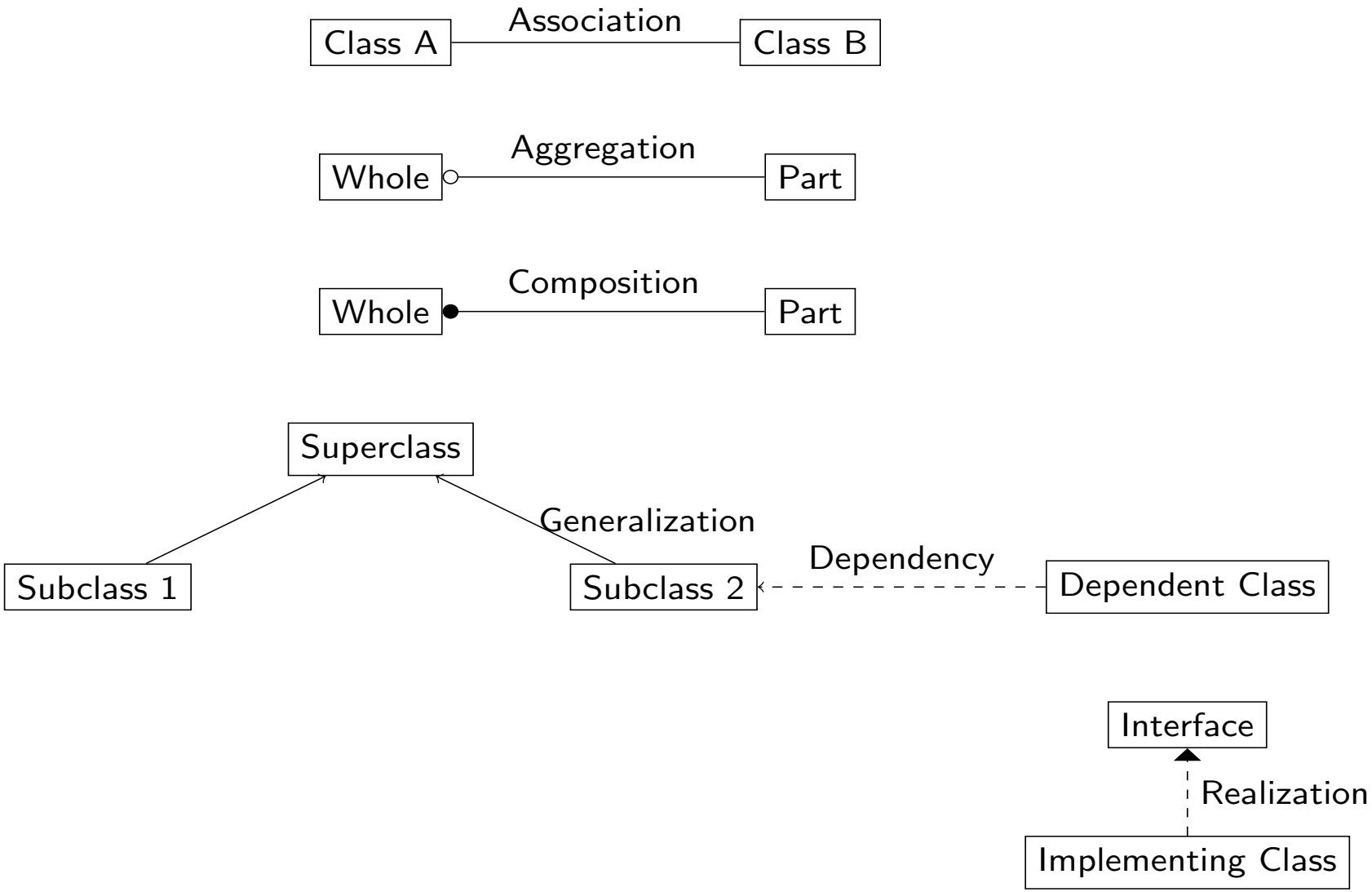
UML Relationships and Associations

Definition: UML relationships define how different elements in a model interact.

Types of Relationships:

- **Association** - A generic relationship between two classes.
- **Aggregation** - A "whole-part" relationship where parts can exist independently.
- **Composition** - A strong "whole-part" relationship where parts cannot exist independently.
- **Generalization** - Represents inheritance between a superclass and a subclass.
- **Dependency** - One class depends on another for some functionality.
- **Realization** - Indicates that a class implements an interface.

UML Relationships



Real-World Examples of UML Relationships

Understanding UML Relationships Through Real-World Examples

Association: A teacher and a student have an association in a school system. A teacher can teach multiple students, and a student can have multiple teachers.

Aggregation: A university and its departments. Departments belong to a university, but they can also exist independently if the university closes.

Composition: A house and its rooms. If the house is demolished, the rooms no longer exist, making them completely dependent on the house.

Generalization: A car and a truck are both types of vehicles. They inherit common properties like having wheels and an engine from the general "Vehicle" category.

Dependency: A smartphone application depends on an external API for weather updates. If the API changes, the app functionality may be affected.

Realization: A payment system that implements a "PaymentProcessor" interface with different payment methods like Credit Card and PayPal.

UML Stereotypes and Constraints

Definition: Stereotypes in UML provide a way to extend the language by defining new model elements based on existing ones. Constraints impose conditions on model elements to ensure consistency.

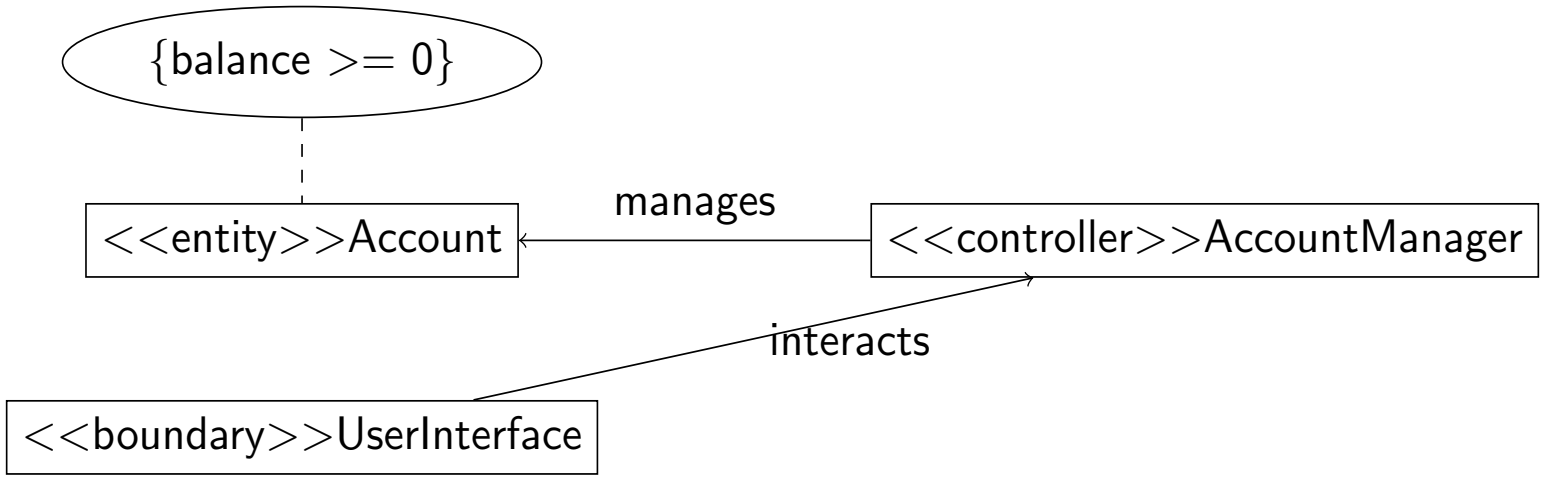
Common Stereotypes:

- <<**interface**>> - Represents an interface.
- <<**entity**>> - Represents a data-centric component.
- <<**controller**>> - Represents a processing component in an MVC architecture.
- <<**boundary**>> - Represents interaction points between users and a system.

Constraints:

- Constraints define rules that must be followed by UML elements.
- Expressed using Object Constraint Language (OCL) or informal descriptions.
- Example: $\{\text{balance} >= 0\}$ ensures an account balance cannot be negative.

UML Stereotypes and Constraints: Standard Representation



UML Modeling Best Practices

When to Use Different Diagrams:

- Use **Class Diagrams** for structural design.
- Use **Sequence Diagrams** to model interaction flows.
- Use **Activity Diagrams** for workflow modeling.
- Use **Use Case Diagrams** to capture user interactions.

Common Mistakes in UML Modeling:

- Overcomplicating diagrams with too many elements.
- Using incorrect relationships between components.
- Not maintaining consistency across different UML diagrams.

Online Shopping System I

Example SRS

The following Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for an online shopping system.

1. **Introduction**
   - Purpose: Define the features and scope of the online shopping system.
   - Scope: The system enables users to browse, purchase, and manage orders securely.
   - Stakeholders: Customers, Administrators, System Developers, Payment Providers.

2. **Overall Description**
   - System Environment: Web and mobile-based application.
   - User Characteristics: Customers with varying levels of technical proficiency.
   - Constraints: Compliance with PCI-DSS for secure transactions.

Online Shopping System II

Example SRS

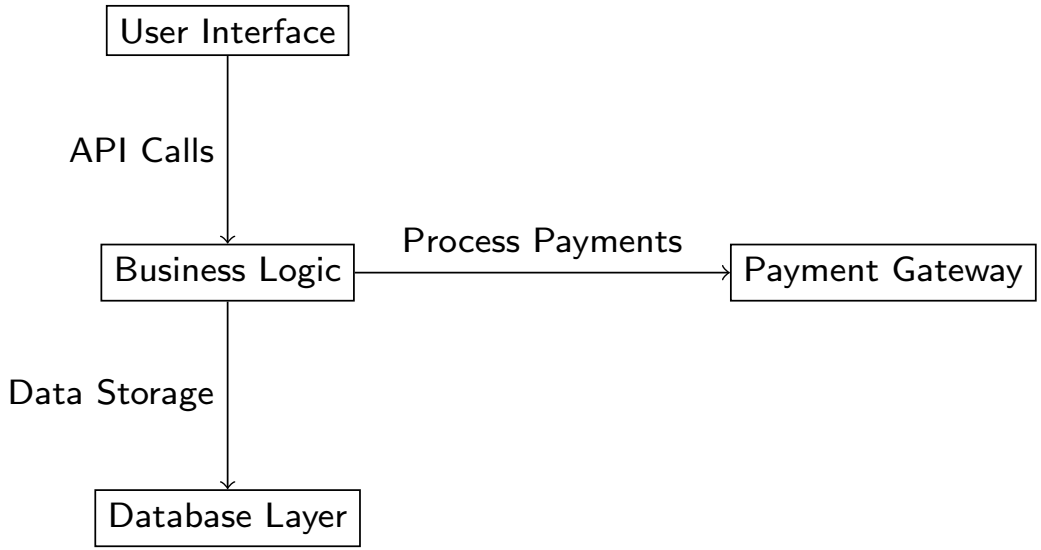
3. **Functional Requirements**
   - Users should be able to create accounts and log in securely.
   - The system should allow product browsing and filtering.
   - Users can add, update, and remove items from their cart.
   - The checkout process should include multiple payment options.
   - Order tracking should be available in real-time.
   - Administrators should be able to manage inventory and generate reports.
4. **Non-Functional Requirements**
   - **Performance:** The system should handle 1000+ concurrent users.
   - **Security:** Must encrypt sensitive user data using AES-256.
   - **Scalability:** Should be able to integrate with third-party APIs.
   - **Usability:** Should have an intuitive UI with accessibility support.
   - **Availability:** 99.9

Online Shopping System III

Example SRS

5. System Architecture Overview
Components:

- User Interface (Web and Mobile Frontend)
- Business Logic Layer (Order Processing, Cart Management)
- Database Layer (User, Product, Order Information)
- Payment Gateway Integration (Stripe, PayPal)



6. Use Case Scenarios

- **Customer Browsing:** A user logs in, searches for products, and adds them to the cart.
- **Checkout Process:** User proceeds to checkout, selects a payment method, and completes the purchase.
- **Order Tracking:** The user checks order status and receives updates.
- **Inventory Management:** Admins update stock levels and view product sales.

Online Shopping System IV

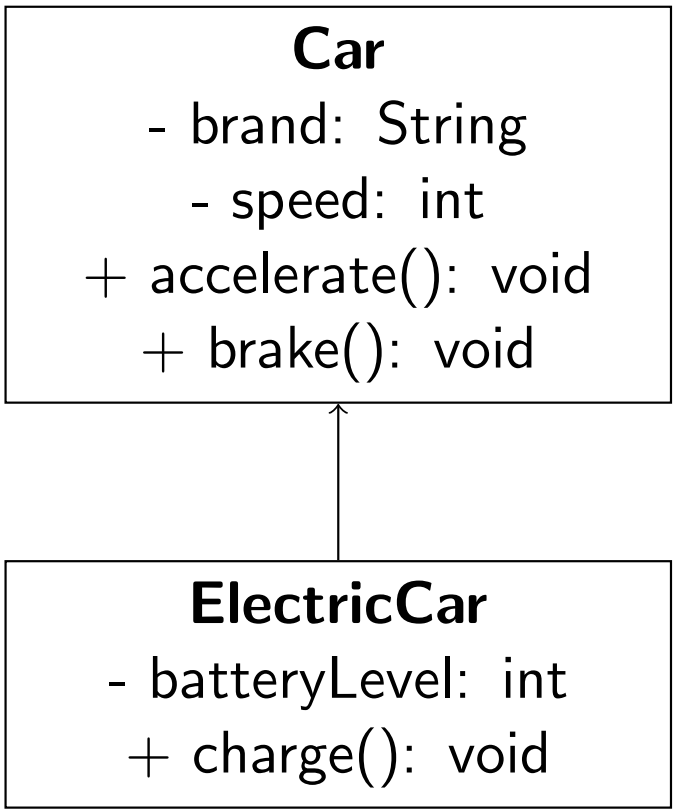
Example SRS

7. **External Interfaces**
   - Payment Gateway (Stripe, PayPal) for secure transactions.
   - Email/SMS Notification Services for order updates.
   - Third-party API Integration (Shipping, Product Catalogs).
8. **Assumptions and Dependencies**
   - Users must have stable internet connectivity for real-time updates.
   - The system depends on third-party services for payment and delivery tracking.
   - Compliance with regional data protection laws (GDPR, CCPA) is required.

UML Class Diagram

Class diagrams represent the structure of a system by showing its classes, attributes, methods, and relationships.

Example Class Diagram:



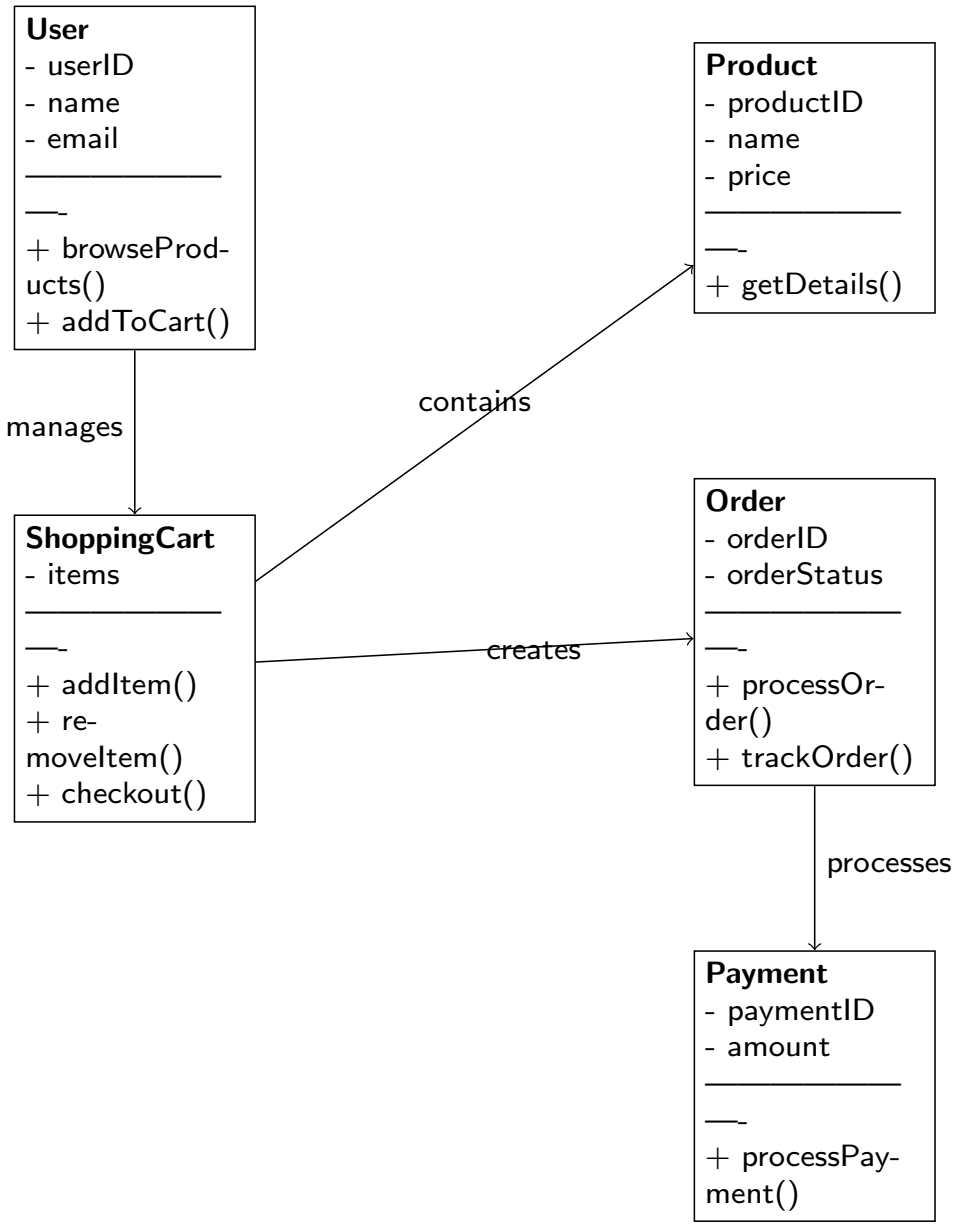
Key Relationships:

- **Inheritance** (ElectricCar extends Car)
- **Composition** (Car has Engine)
- **Association** (Car interacts with Driver)

Class Diagram: Online Shopping System

Explanation:

- The **User** class interacts with the system to browse and purchase products.
- The **Product** class contains details about each product.
- The **ShoppingCart** class holds selected items and facilitates checkout.
- The **Order** class processes orders and tracks their status.
- The **Payment** class manages transactions and interactions with payment gateways.

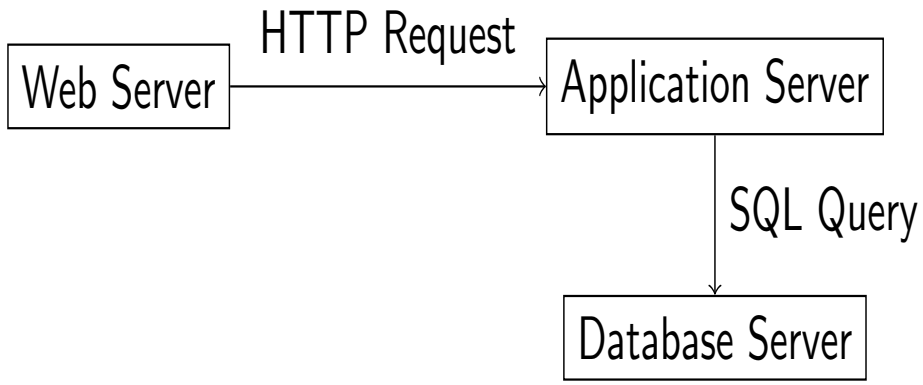


UML Component Diagram

A Component Diagram represents the high-level structure of a system, showing its components and dependencies.

Key Elements:

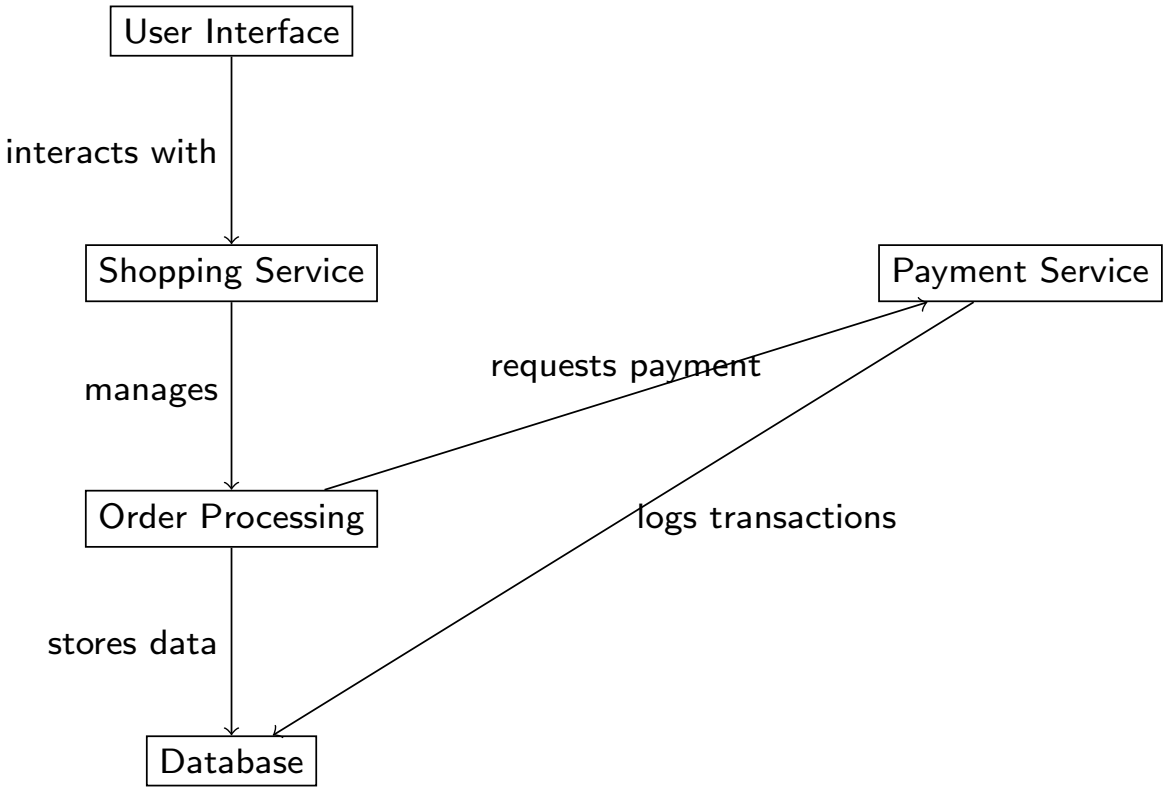
- Components (Modules of a system)
- Interfaces (Provided/required connections between components)
- Dependencies (Relationships between components)



Component Diagram: Online Shopping System

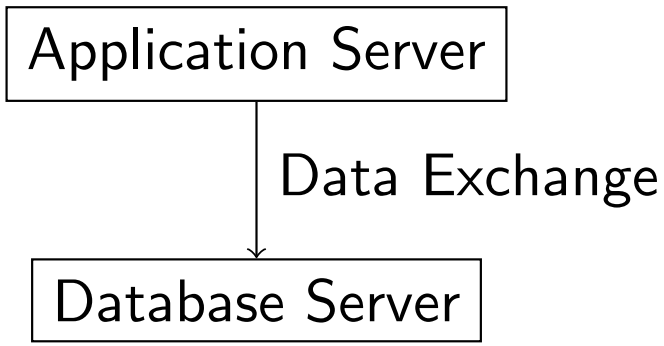
Explanation:

- The system is divided into different components, each responsible for specific functionality.
- The **User Interface** component handles customer interactions.
- The **Shopping Service** component manages shopping cart operations and order processing.
- The **Payment Service** handles transactions and external payment gateway interactions.
- The **Database** component stores all product, user, and order information.



Deployment Diagram

Deployment diagrams model the physical deployment of software artifacts onto hardware nodes.



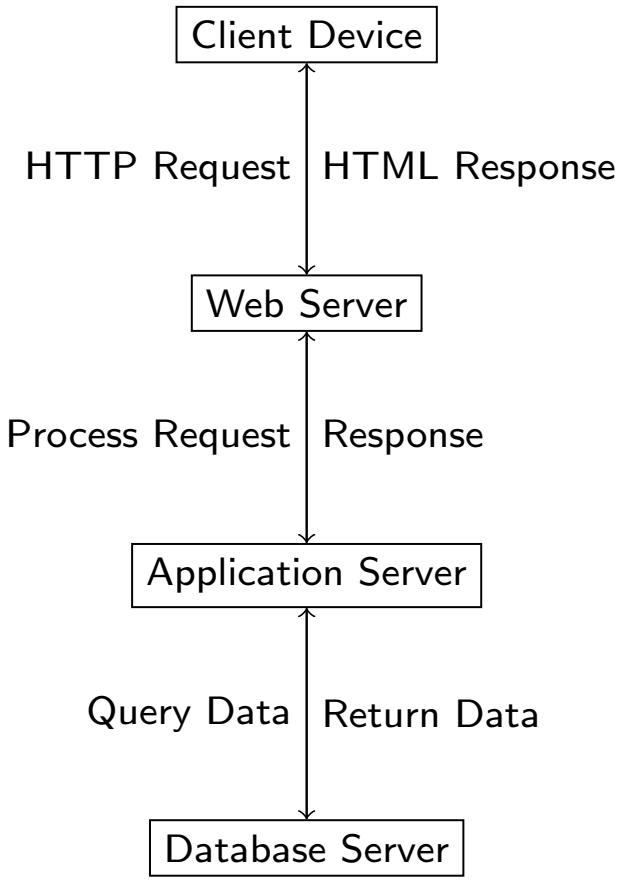
Key Elements:

- Nodes (Physical devices/servers)
- Artifacts (Deployed software components)
- Connections (Communication links between nodes)

Deployment Diagram: Online Shopping System

Explanation:

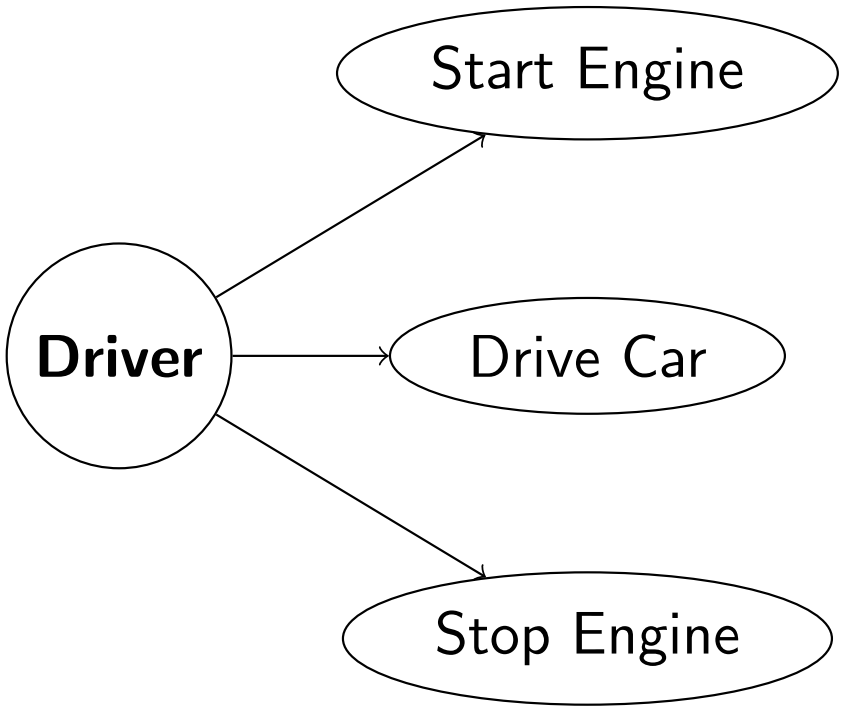
- The system is deployed across multiple nodes.
- The **Client Device** hosts the user interface for customer interactions.
- The **Web Server** manages business logic and serves requests.
- The **Application Server** handles order processing and payment transactions.
- The **Database Server** stores user, product, and order data.
- Secure communication occurs between components.



UML Use Case Diagram

Use case diagrams show the interactions between users (actors) and the system.

Example Use Case Diagram:

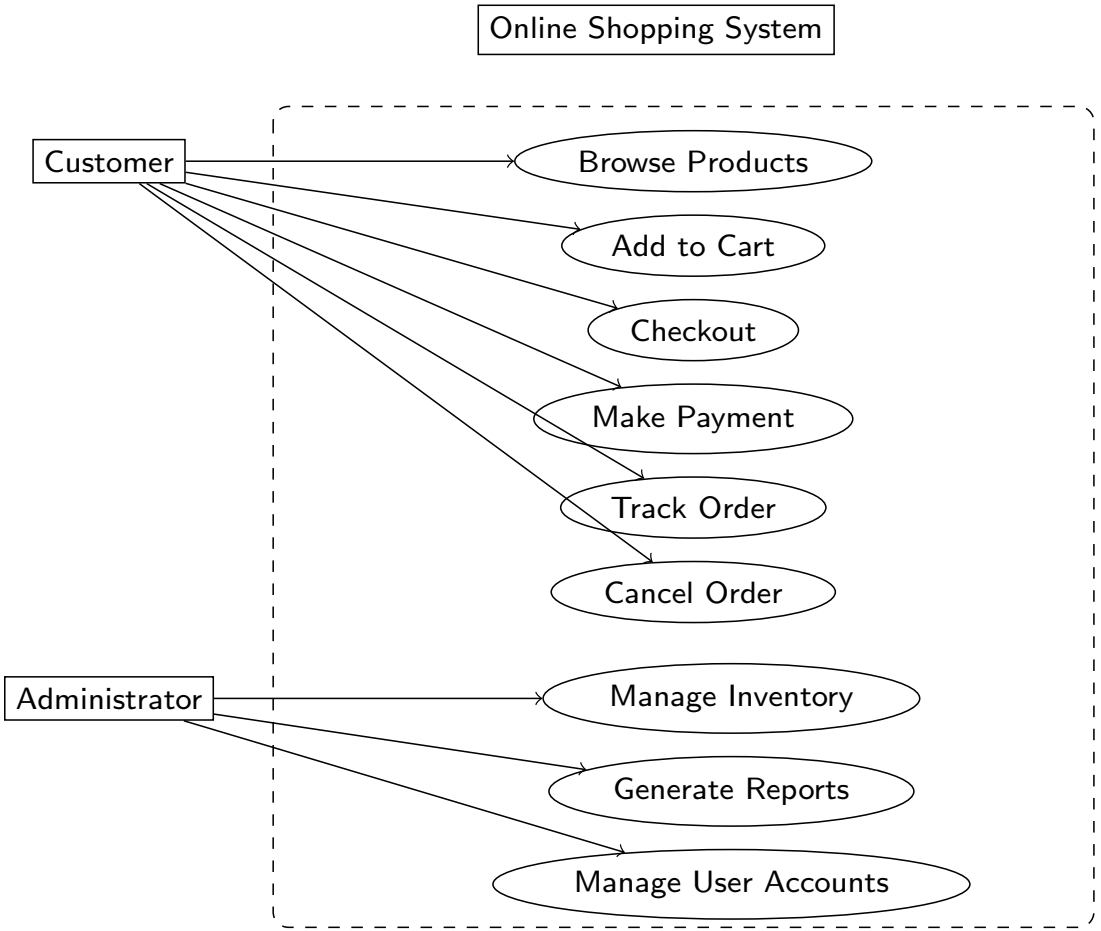


Key Elements:

- **Actors** (Users interacting with the system)
- **Use Cases** (Functionalities of the system)
- **Relationships** (Associations between actors and use cases)

Use Case Diagram: Online Shopping System

Scenario: Online Shopping System



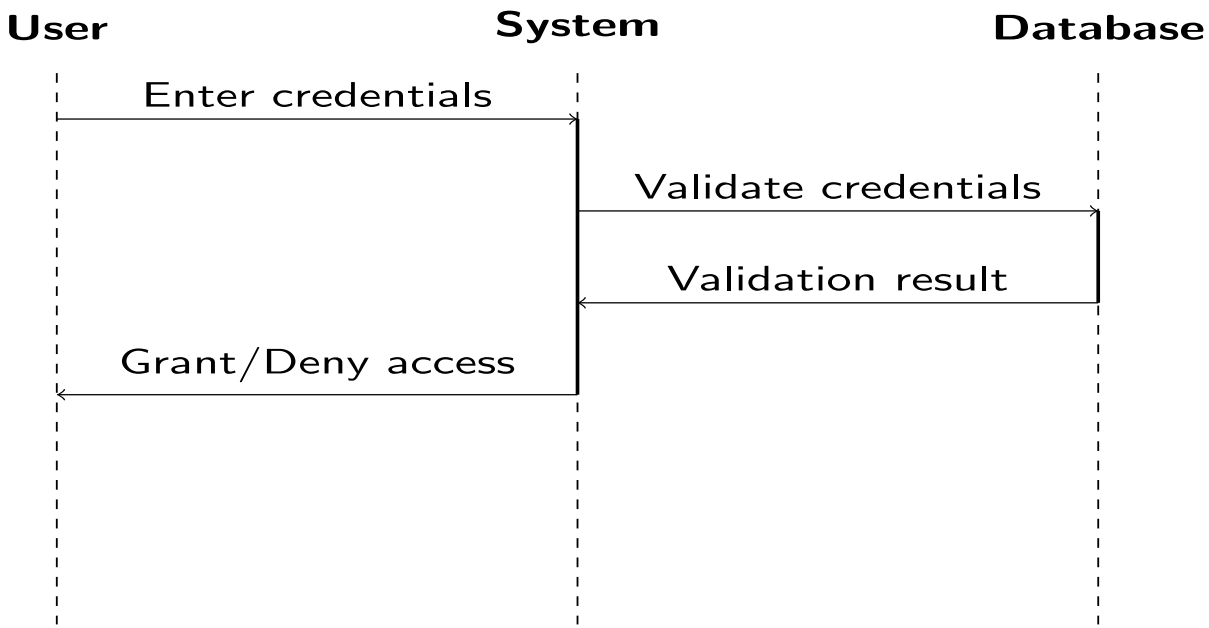
Explanation:

- The **Customer** interacts with the system to browse, purchase, and track orders.
- The **Administrator** manages inventory, generates reports, and handles user accounts.
- The system boundary groups all system functionalities.
- The layout has been optimized to avoid overlapping, ensuring clarity.

UML Sequence Diagram

Sequence diagrams depict object interactions over time.

Example Sequence Diagram:



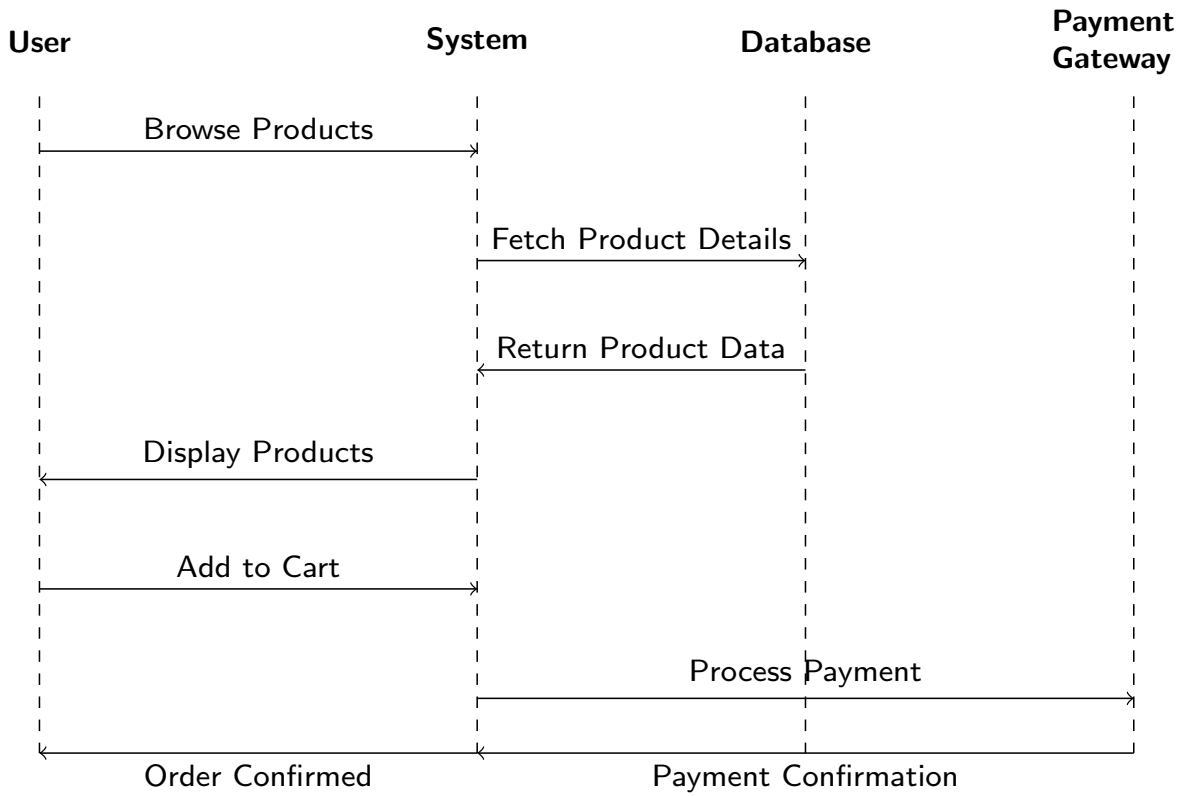
Key Elements:

- **Actors and Objects** (Users, system components)
- **Lifelines** (Dashed vertical lines showing object lifespan)
- **Messages** (Interactions between objects)

Sequence Diagram: Online Shopping Process

Explanation:

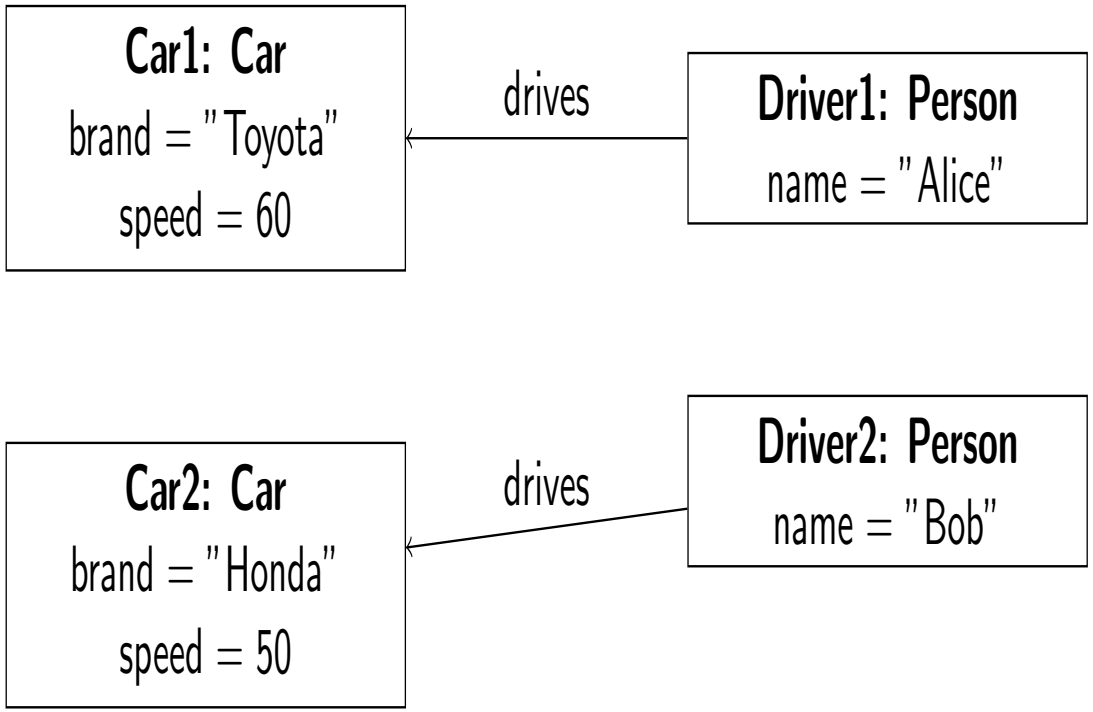
- The user initiates the process by browsing products.
- The system retrieves product details and displays them.
- The user adds items to the cart and proceeds to checkout.
- Payment details are processed, and the system confirms the order.
- A confirmation message is sent to the user, and the order is forwarded to fulfillment.



UML Object Diagram

Object diagrams provide a snapshot of instances of classes at a particular moment in time, illustrating the relationships between objects.

- Represents real-world examples of class structures.
- Showcases object relationships and attribute values.
- Useful for debugging and understanding class behavior.

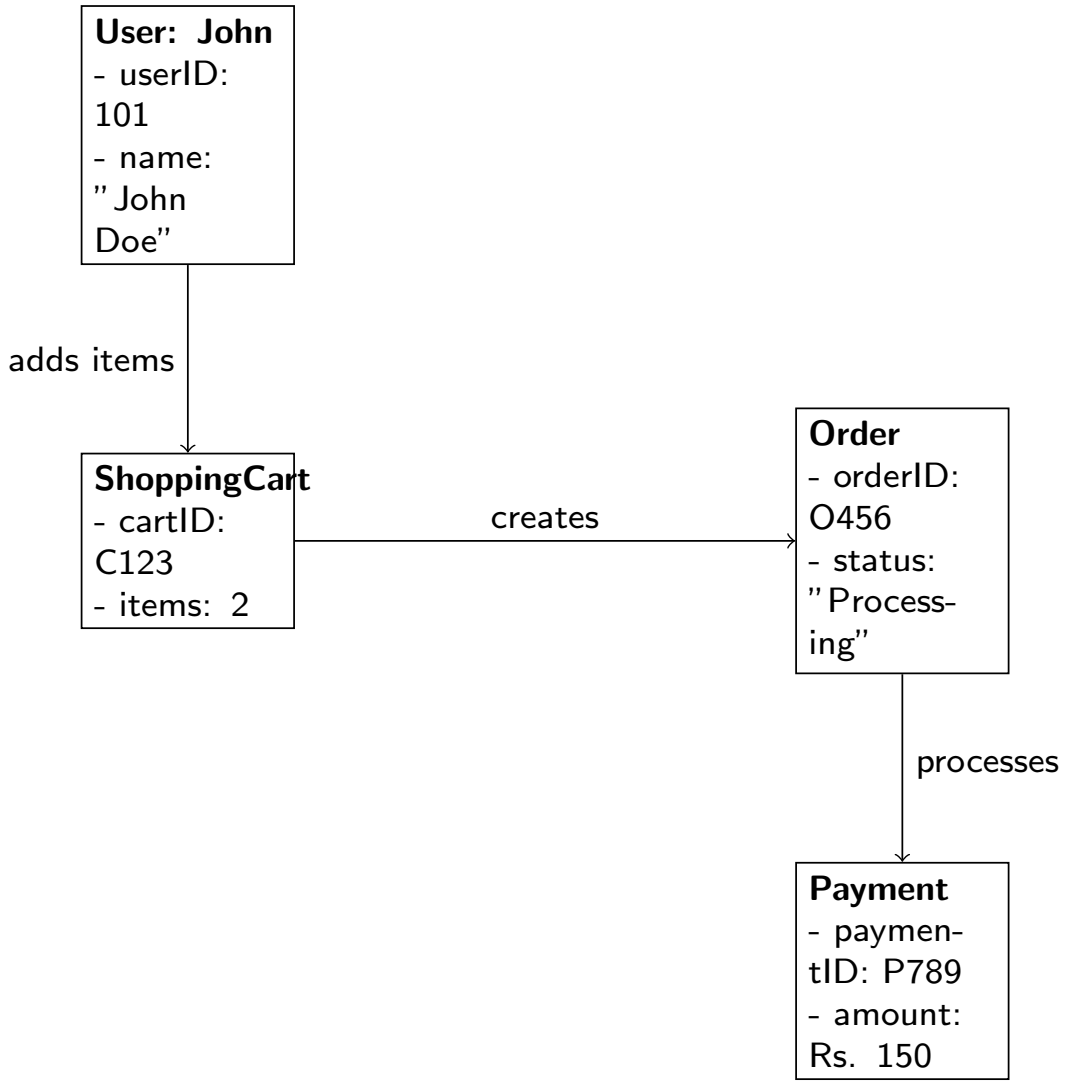


Use Case: Object diagrams help in analyzing real-world object interactions, debugging runtime behavior, and validating class diagrams.

Object Diagram: Online Shopping System

Explanation:

- Illustrates real-time examples of objects and their connections.
- The **User** instance interacts with the system to browse and purchase products.
- The **Cart** object holds selected items.
- The **Order** instance represents a purchase request.
- The **Payment** object handles the transaction.

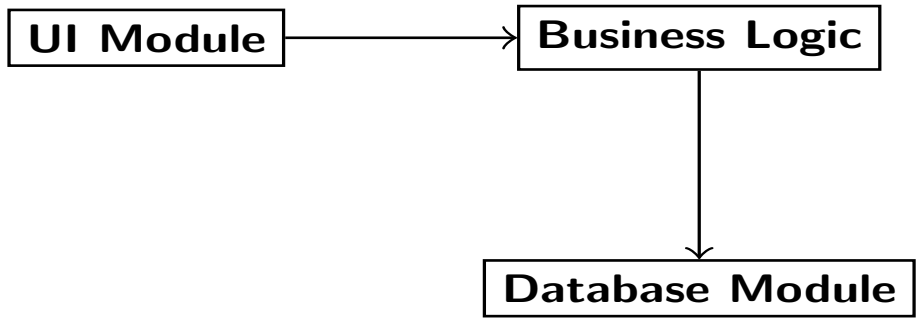


UML Package Diagram

Package diagrams organize elements into meaningful groups to manage complexity.

Key Elements:

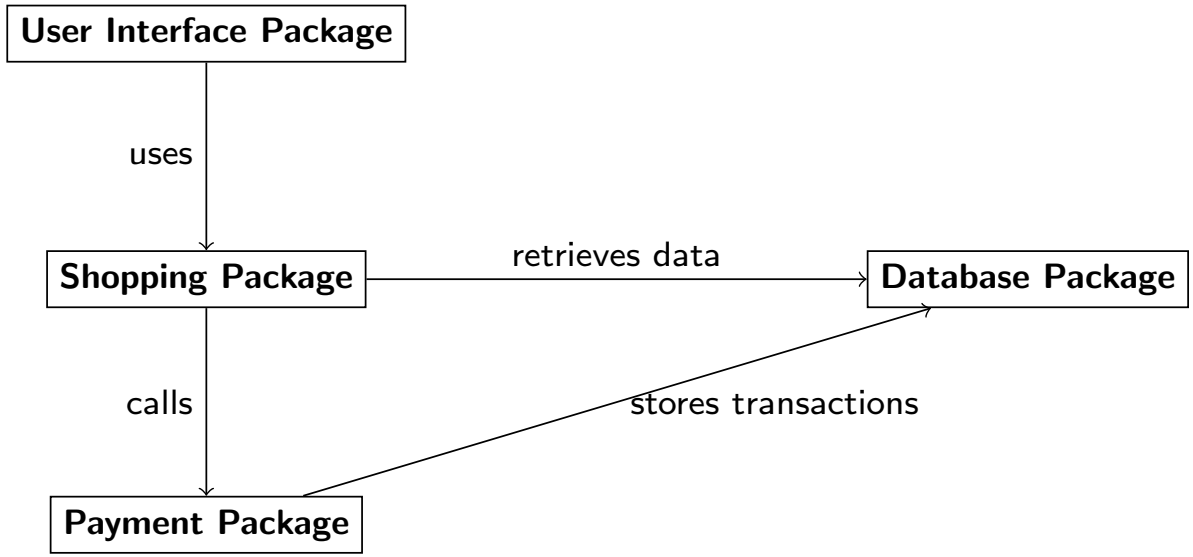
- Packages (Logical grouping of related classes and interfaces)
- Dependencies (Relationships between different packages)



Package Diagram: Online Shopping System

Explanation:

- Packages group related classes together to improve modularity.
- The **User Interface Package** handles user interactions.
- The **Shopping Package** manages cart operations and order placement.
- The **Payment Package** handles transactions.
- The **Database Package** stores and retrieves data.
- Dependencies between packages show how they interact.

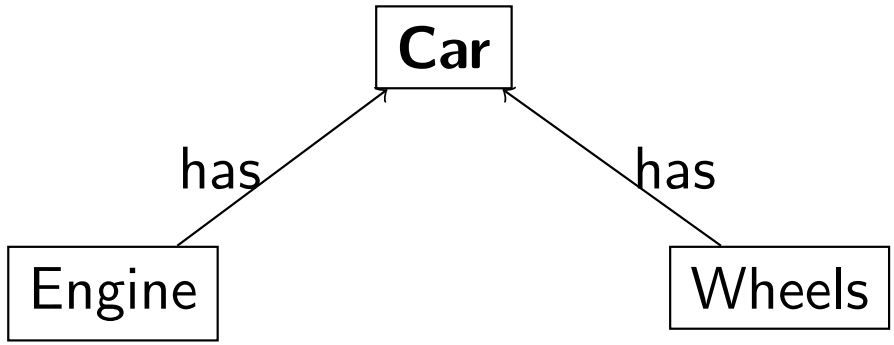


UML Composite Structure Diagram

Composite structure diagrams show the internal structure of a class and how its parts interact.

Key Points:

- Illustrating component-based architectures.
- Showing collaboration between objects within a system.

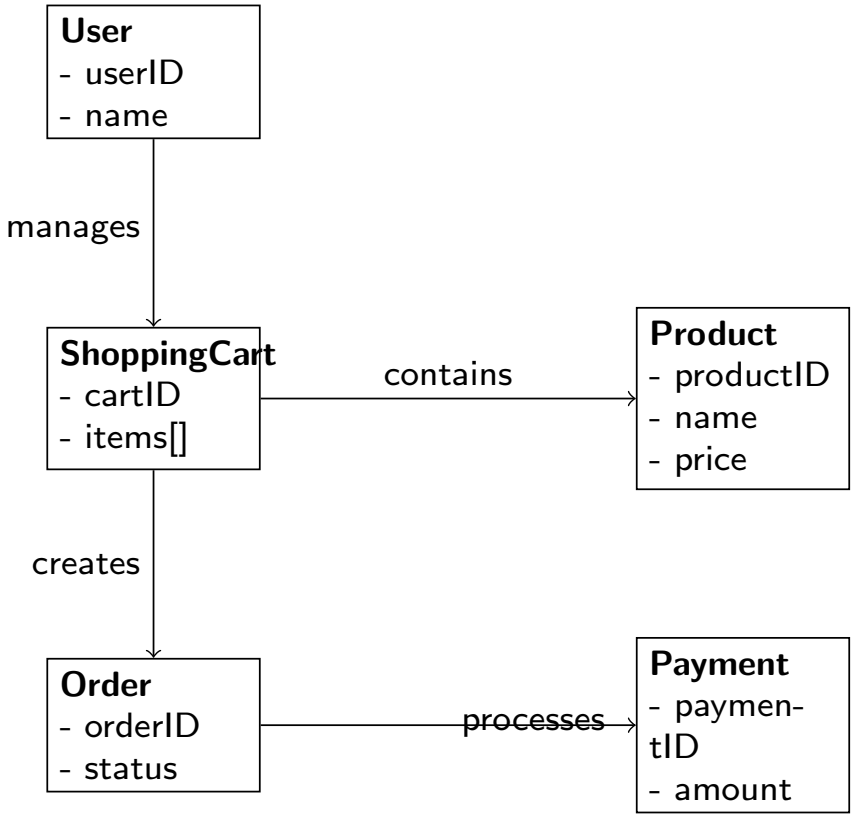


Use Case: Composite structure diagrams help illustrate internal object composition, component interactions, and hierarchical structures within a system.

Composite Structure Diagram: Online Shopping System

Explanation:

- Represents how components interact within an object.
- The **ShoppingCart** class contains multiple **Product** instances.
- The **Order** class aggregates multiple items and manages payment.
- The **User** class interacts with both the cart and the order process.

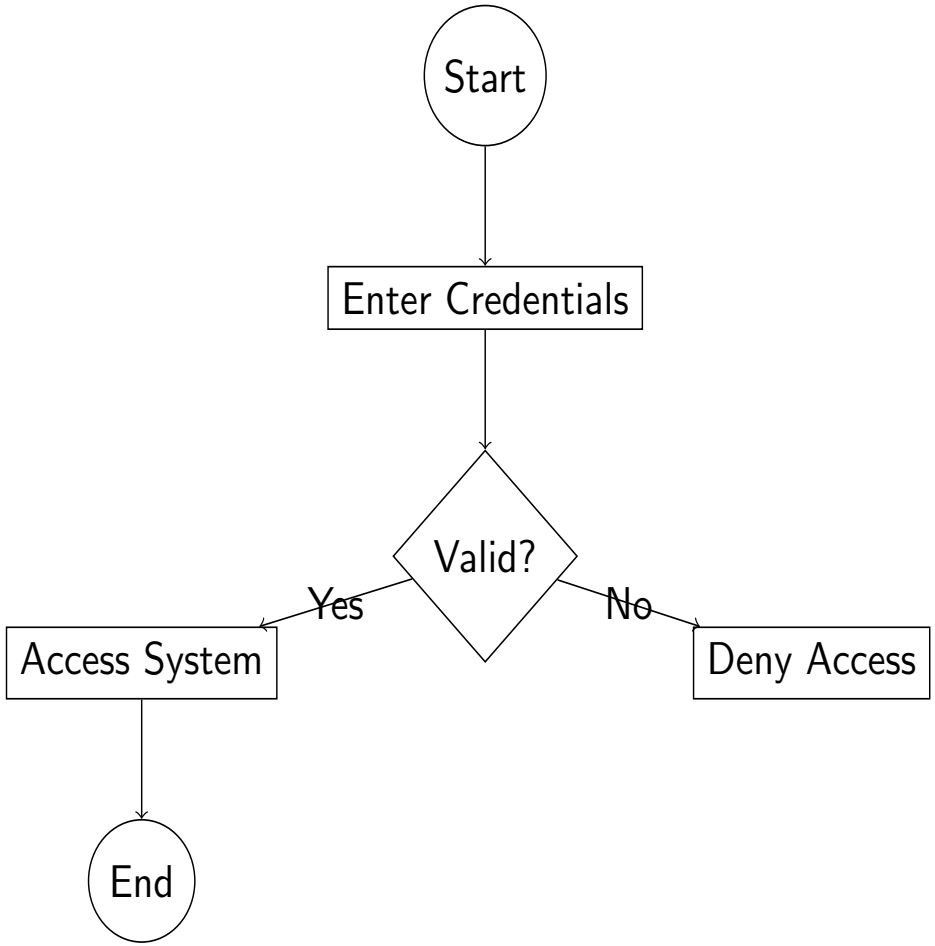


UML Activity Diagram

Activity diagrams represent workflows and processes.

Key Elements:

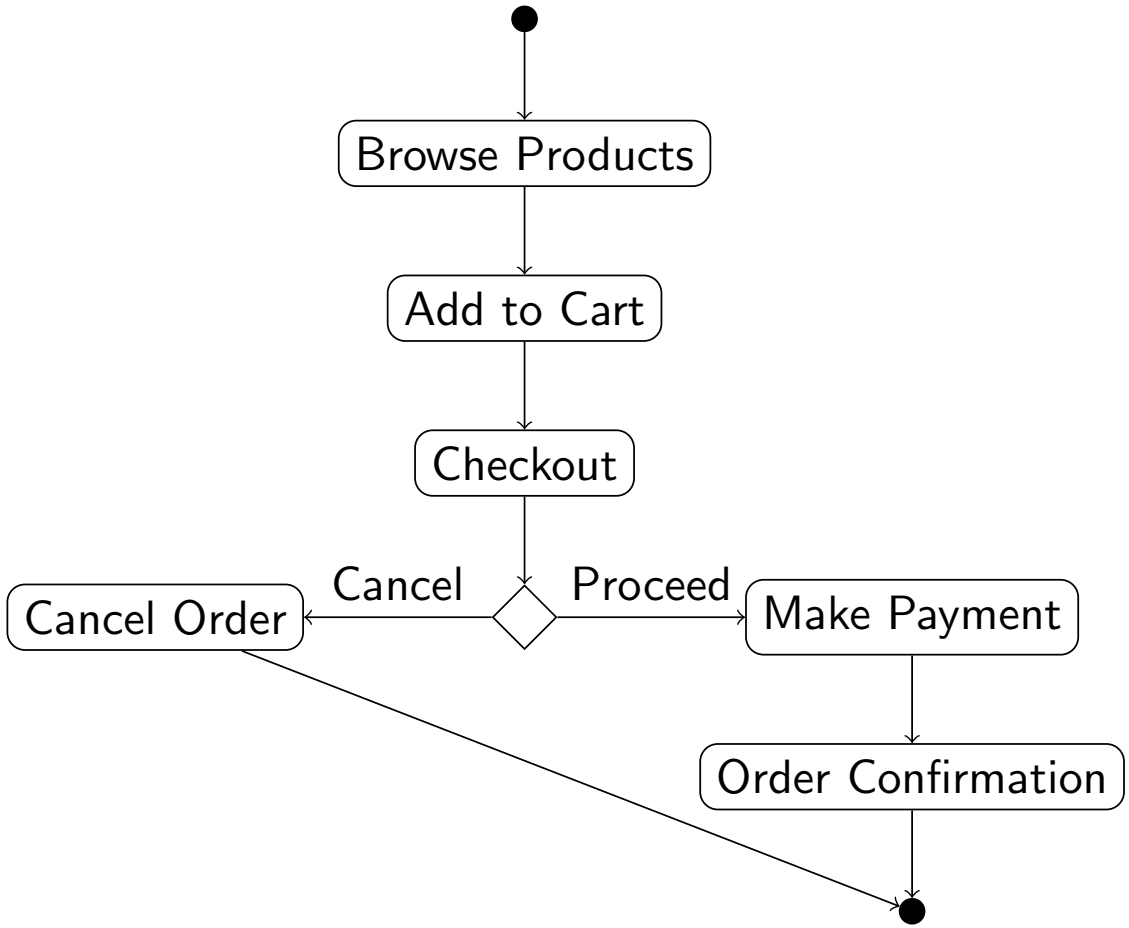
- Actions (Tasks or operations performed)
- Decision Nodes (Conditional branching points)
- Start and End Nodes (Indicating workflow initiation and completion)



Activity Diagram: Online Shopping Process

Explanation:

- The user starts by browsing products.
- Products can be added to the cart, followed by checkout.
- A decision point allows the user to either proceed with payment or cancel the order.
- After successful payment, the order is confirmed and the process ends.

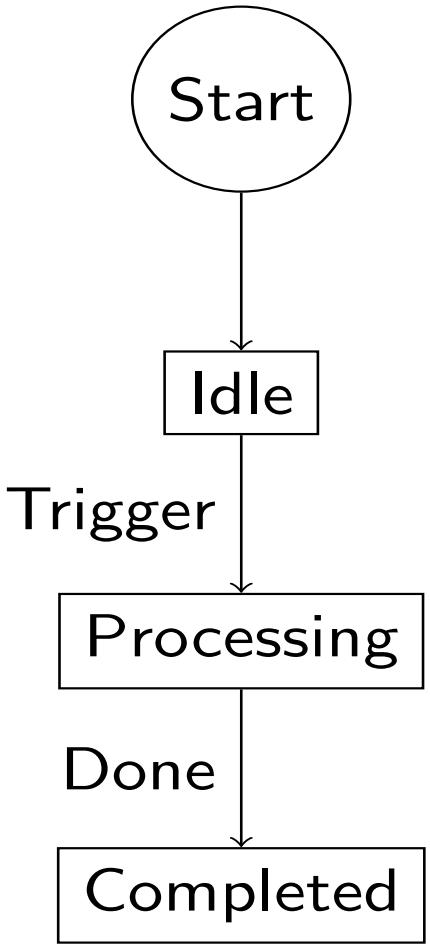


UML State Machine Diagram

State machine diagrams depict the different states of an object and how it transitions between states.

Key Components:

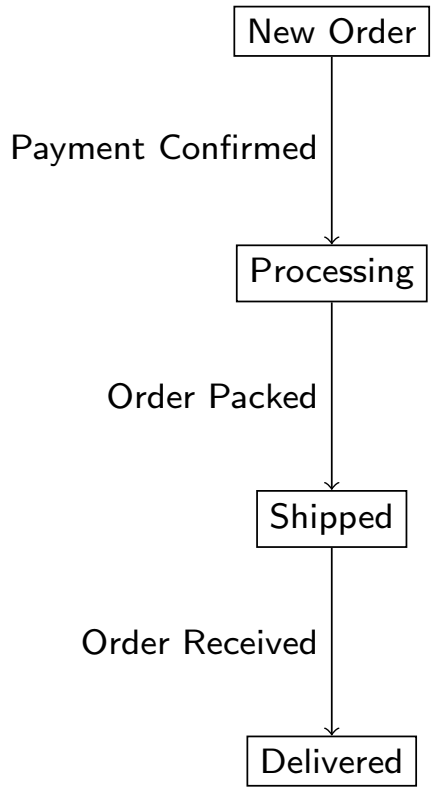
- States (Representing different conditions of an object)
- Transitions (Arrows indicating state changes)
- Events (Triggers causing state transitions)



State Machine Diagram: Order Processing

Explanation:

- Represents the states of an **Order** object in an online shopping system.
- The order transitions through different stages based on user and system actions.
- States include **New Order**, **Processing**, **Shipped**, and **Delivered**.

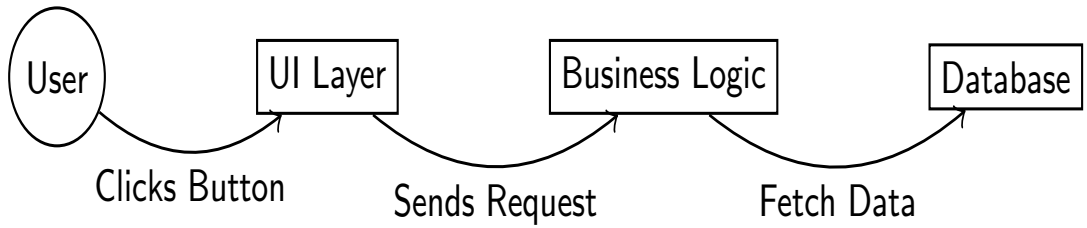


UML Communication Diagram

- Communication diagrams illustrate message exchanges between objects.
- Communication diagrams show interactions between objects, emphasizing the relationships rather than the order of messages.

Key Features:

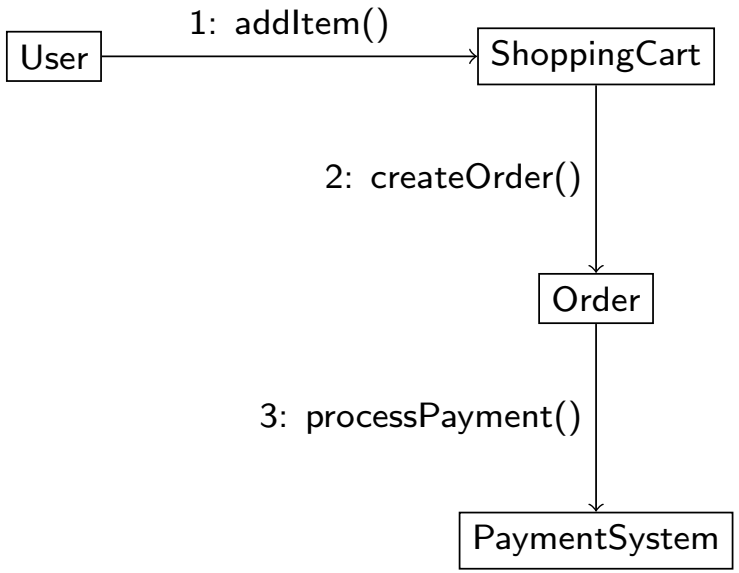
- Focus on message flow rather than time sequence.
- Depicts interactions between objects for a specific use case.



Communication Diagram: User Purchase Process

Explanation:

- Shows interactions during the purchase process.
- Objects involved: **User, ShoppingCart, Order, PaymentSystem**.
- Messages include adding items, checking out, and processing payments.



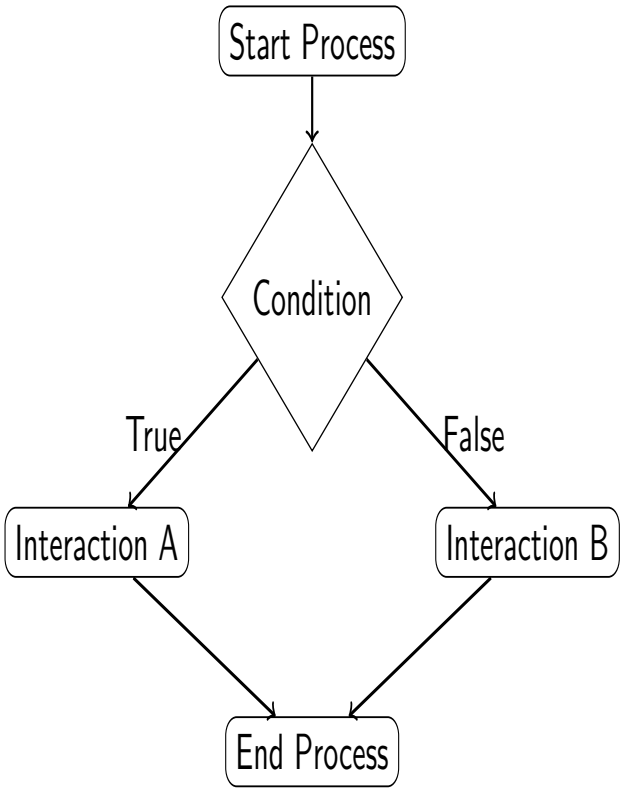
UML Interaction Overview Diagram

- Interaction overview diagrams combine sequence and activity diagrams.
- They provide a high-level view of interactions within a system, combining elements of activity and sequence diagrams.

Key Points:

- Representing high-level workflow processes.
- Showing different interaction sequences within a system.

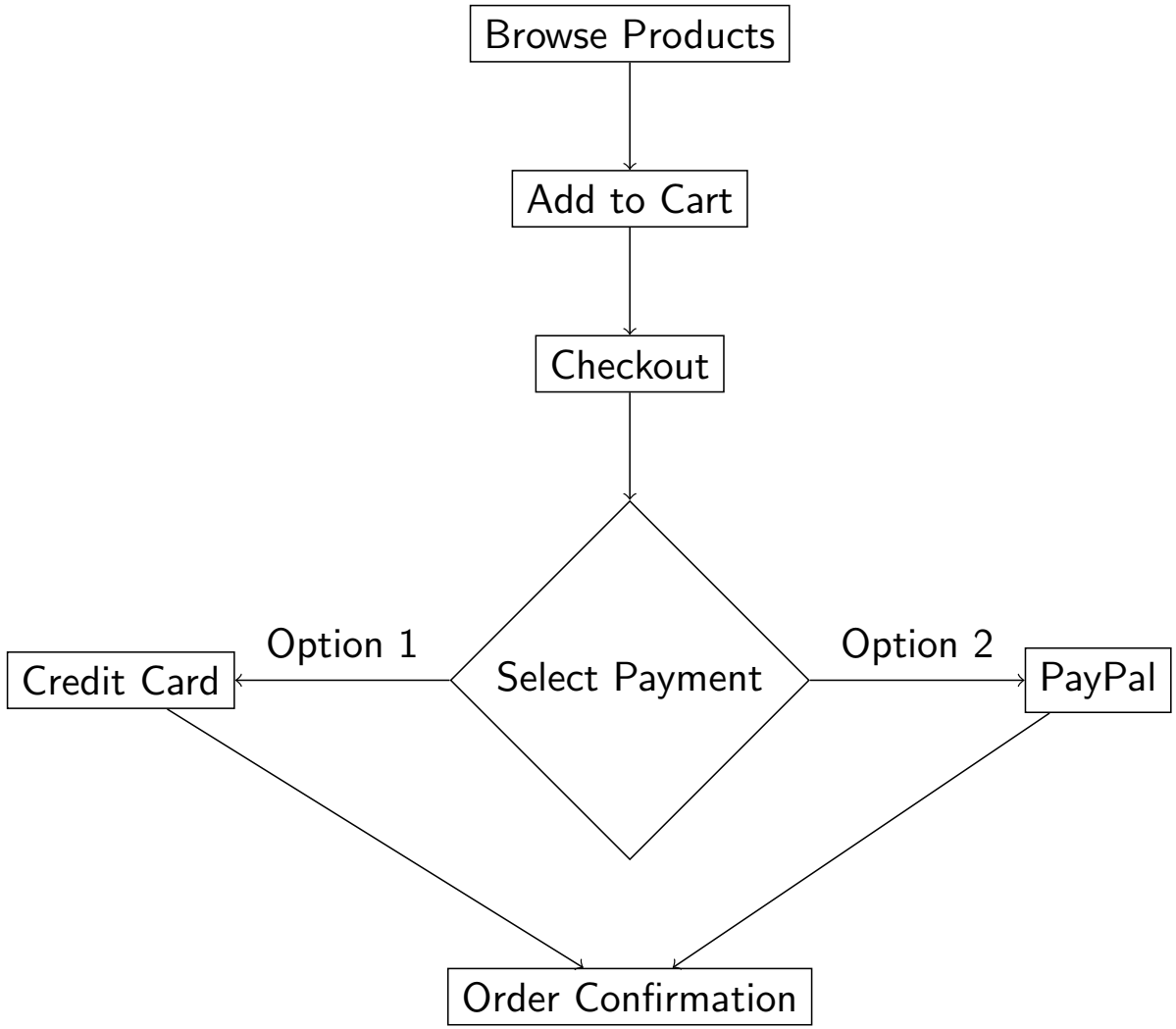
Use Case: Interaction Overview Diagrams are useful for modeling high-level workflows that include multiple interactions.



Interaction Overview Diagram: Online Shopping

Explanation:

- Represents an overview of the user shopping process with branching paths.
- Includes browsing, adding items, checkout, different payment methods, and order confirmation.
- Shows conditional paths for successful and failed transactions.

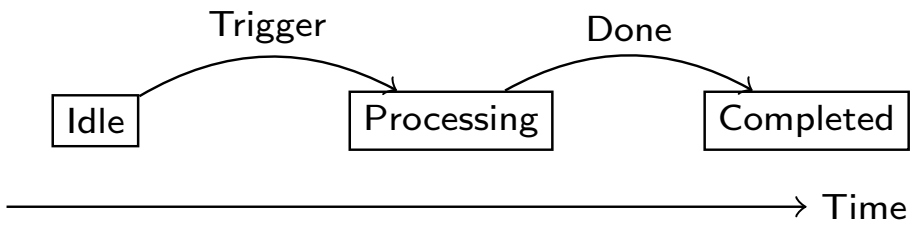


UML Timing Diagram

Timing diagrams show time-dependent behavior of objects.

Key Elements:

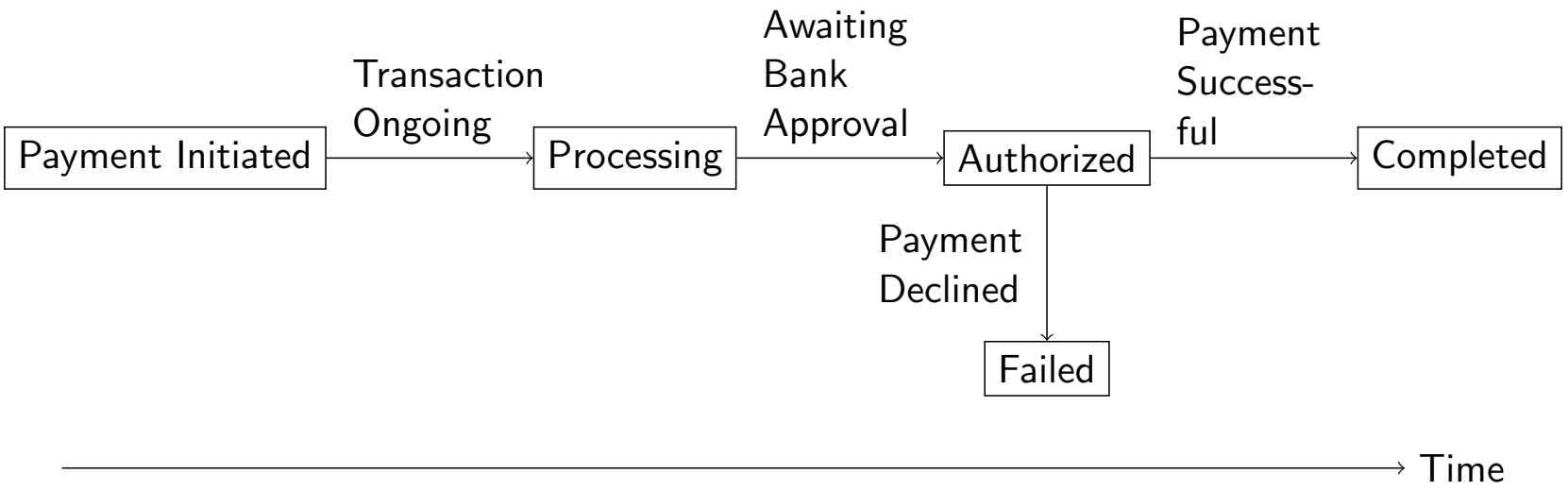
- Lifelines (Object states over time)
- State Changes (Transitions triggered by time events)
- Time Axis (Representing the progression of time)



Timing Diagram: Payment Processing

Explanation:

- Shows the detailed timing of events in payment processing.
- Tracks state transitions such as **Initiated**, **Processing**, **Authorized**, **Completed**, and **Failed**.
- Helps in understanding real-time event dependencies.



UML and Design Patterns

Connection Between UML and Design Patterns:

- UML diagrams help visualize and document design patterns in software architecture.
- Structural patterns are represented using Class and Component Diagrams.
- Behavioral patterns are modeled using Sequence and Activity Diagrams.

Class Diagrams and Structural Design Patterns

Examples:

- **Singleton Pattern** - Represented as a class with a private constructor and static instance.
- **Factory Pattern** - Shown as a class that creates instances of other classes.
- **Adapter Pattern** - Demonstrates how two incompatible interfaces can work together.
- **Composite Pattern** - Represents tree structures with parent-child relationships.

Sequence Diagrams and Behavioral Design Patterns

Examples:

- **Observer Pattern** - Sequence diagrams show how observers receive updates from subjects.
- **Strategy Pattern** - Shows the dynamic selection of algorithms at runtime.
- **Command Pattern** - Represents how requests are encapsulated and executed.

Component and Deployment Diagrams for Architectural Patterns

Examples:

- **Model-View-Controller (MVC)** - Component diagrams represent models, views, and controllers.
- **Microservices Architecture** - Deployment diagrams visualize distributed services.
- **Layered Architecture** - Showcases different software layers and their dependencies.

Summary of UML Diagrams

Main Features and Use Cases:

- **Class Diagram** - Represents system structure (Use for object-oriented design).
- **Use Case Diagram** - Shows user interactions (Use in requirements gathering).
- **Sequence Diagram** - Models interactions over time (Use for dynamic behavior).
- **Activity Diagram** - Represents workflows (Use for business process modeling).
- **State Machine Diagram** - Shows object states and transitions (Use for lifecycle modeling).
- **Component Diagram** - Illustrates software components (Use for modular system design).
- **Deployment Diagram** - Visualizes system deployment (Use for infrastructure planning).
- **Communication Diagram** - Emphasizes object relationships (Use for message-driven systems).