- **1 a) What are business to consumer (B2C) and business to business (B2B) Web applications, and what are some examples of each type? What are some business drivers for today's information systems?(3)**

Business-to-Consumer (B2C) vs. Business-to-Business (B2B) Web Applications

1. Business-to-Consumer (B2C) Web Applications

Definition: Online platforms where businesses sell products or services directly to individual consumers.

Key Features:

- Designed for ease of use, quick transactions, and mass-market appeal.
- Often include marketing tools (discounts, ads, loyalty programs).
- Payment methods optimized for end-users (credit cards, PayPal, mobile wallets).

Examples:

- **Amazon** – Sells products directly to shoppers.
- **Netflix** – Provides streaming services to consumers.
- **Uber** – Connects riders with drivers for transportation.
- **Spotify** – Offers music subscriptions to individual users.

---

2. Business-to-Business (B2B) Web Applications

Definition: Online platforms where businesses sell products, services, or software to other businesses.

Key Features:

- Focus on bulk orders, contracts, and long-term relationships.
- Often include invoicing, procurement workflows, and enterprise integrations.

- Pricing may be negotiated (not always publicly listed).

Examples:

- **Alibaba** – Connects wholesalers with retailers/manufacturers.
- **Salesforce** – Provides CRM software for businesses.
- **Slack** – Sells team collaboration tools to companies.
- **SAP Ariba** – A procurement platform for supply chain management.
- **What are some business drivers for today's information systems?**
    - Globalization.
    - E-commerce and e-business.
    - Security and privacy.
    - Knowledge asset management.
    - Business process redesign.

✓ **b. You are a new systems analyst and eager to prove your abilities on your first project. You are at a problem analysis meeting with the system owners and users and find yourself saying, "we need to do this to solve the problem," into what common trap are you in danger of falling? What technique could you use to avoid the trap?(3) (20-21/19-20/18-19)**

Why It's a Trap

As a new systems analyst, you might feel pressure to prove your competence quickly, leading you to propose solutions prematurely. When you say, "*We need to do this to solve the problem,*" without fully analyzing the issue, you risk:

- **Solving the wrong problem** (because you misunderstood the real issue).
- **Overlooking better solutions** (since you didn't explore alternatives).
- **Alienating stakeholders** (by dismissing their concerns or missing key requirements).

This is called the **"Jumping to Solutions"** (or **"Premature Solutioning"**) trap, a common mistake in problem-solving.

Why It Happens

- **Eagerness to impress** (wanting to show quick results).
- **Assumption bias** (believing you already know the answer).
- **Pressure from stakeholders** (who may demand immediate fixes).

How to Avoid the Trap

1. **Ask Open-Ended Questions First**
    - Instead of saying "*We should do X,*" ask:
        - "*Can you describe the problem in more detail?*"
        - "*What impact is this having?*"
        - "*How long has this been happening?*"
    - This helps uncover the **real** problem, not just symptoms.
2. **Use Root Cause Analysis Techniques**
    - **The 5 Whys**: Keep asking "*Why?*" until you find the underlying cause.
        - Example:
            - "*The system crashes at peak hours.*" (Why?)
            - "*Because server load is too high.*" (Why?)
            - "*Because the database queries are not optimized.*" (Now you have the real issue.)
    - **Fishbone (Ishikawa) Diagram**: Visually map possible causes to identify the main one.
3. **Practice Active Listening**
    - Let stakeholders explain without interrupting.
    - Paraphrase their points to confirm understanding ("*So, the main issue is…?*").
4. **Delay Solution Proposals**
    - Say: "*Before suggesting fixes, I'd like to make sure I fully understand the problem.*"

- This builds trust and ensures you address the right issue.

✓ **c. Why do many new systems analysis fail to effectively analyze problems? What can they do to become more effective?(2)**

Reasons for failure:

1. Lack of domain knowledge.
2. Inadequate communication with stakeholders.
3. Over-reliance on tools instead of understanding the actual problem.
4. Ignoring user feedback or real-world constraints.

How to become effective:

- Gain deep **understanding of the business domain**.
- Improve **interpersonal and communication skills**.
- Practice **active listening** and **requirement gathering techniques**.
- Use **structured methodologies and tools** (e.g., DFDs, UML).
- Validate assumptions with **real user feedback** and prototypes.

By focusing on **understanding before solving**, new analysts can avoid common pitfalls and deliver better system solutions.

d) What are the four steps in a system development process? What happens in each step? ADI^2 (4) (20-21/19-20)

1. **System Initiation**: Define scope, goals, and budget.
2. **System Analysis**: Study business requirements.
3. **System Design**: Create technical specifications.
4. **System Implementation**: Build, test, and deploy the solution.

e) Assume you are a systems analyst who will be conducting a requirements analysis for an individually owned brick-and-mortar retail store with a point of sale system. Identify who the typical internal and exteral users might include.(2) (20-21/19-20)

As a systems analyst conducting a **requirements analysis** for a retail store's **Point of Sale (POS) system**, you would interact with the following **stakeholders**:

1. Internal Users (Employees & Management)

These users interact with the POS system daily to process sales, manage inventory, and oversee operations.

- ✓ **Store Owner**
  - *Role*: Approves system features, sets business rules, and reviews sales/reports.
  - *Needs*: Financial summaries, inventory alerts, and sales performance dashboards.
- ✓ **Cashiers / Sales Associates**
  - *Role*: Process transactions, handle returns, and apply discounts.
  - *Needs*: Fast checkout, easy-to-use interface, and receipt printing.
- ✓ **Inventory Manager**
  - *Role*: Tracks stock levels, places orders, and manages suppliers.
  - *Needs*: Real-time inventory updates, low-stock alerts, and purchase order integration.
- **Store Manager**
  - *Role*: Oversees staff, monitors sales trends, and handles promotions.
  - *Needs*: Employee performance tracking, shift scheduling, and sales analytics.
- ✓ **Accountant / Bookkeeper**
  - *Role*: Manages finances, tax reporting, and payroll.
  - *Needs*: Exportable sales data, tax calculation features, and integration with accounting software (e.g., QuickBooks).

---

2. External Users (Customers & Third Parties)

These users interact with the POS system indirectly or through integrations.

- **Customers**
    - *Role*: Make purchases, request refunds, and use loyalty programs.
    - *Needs*: Smooth checkout, digital receipts, and support for payment methods (credit cards, mobile wallets).
- **Payment Processors (e.g., Stripe, Square, PayPal)**
    - *Role*: Facilitate credit/debit card transactions.
    - *Needs*: Secure API integration for real-time payment processing.
- **Suppliers / Vendors**
    - *Role*: Provide inventory and restock products.
    - *Needs*: Automated reorder alerts and purchase order generation.
- **Tax Authorities (e.g., IRS, VAT systems)**
    - *Role*: Require accurate sales tax reporting.
    - *Needs*: Compliance with tax laws and automated tax calculations.
- **Loyalty Program Partners**
    - *Role*: Offer rewards and discounts.
    - *Needs*: Integration with membership databases and coupon validation.

A **POS system** must balance **internal operational efficiency** (cashiers, inventory, reporting) with **external stakeholder needs** (customers, payment processors, suppliers). As a **systems analyst**, you'd gather requirements from both groups to ensure the system is **user-friendly, secure, and scalable**.

2-a) As a new project manager in a rapidly growing organization, you have been asked to lead a project team for an important project. The scope of the project is not too broad, project time frames are somewhat on the tight side but definitely doable, and the budget is more than generous. In fact, you have been given the authority to hire as many people as you want for your project team. You estimate that 5 people would be about right for this type of project, 8 would provide a healthy amount of backup, and 10 could give you the resources to deliver an outstanding system in record time. What is something you might want to keep in mind before making your decision on how many people to hire? (2) (20-21/19-20)

Team Size Consideration Before Hiring

Before deciding how many people to hire (5, 8, or 10), you should consider:

- **Brooks' Law**: "*Adding more people to a late project makes it later.*"
    - More team members increase **communication overhead** (more meetings, coordination, and potential misalignment).
    - **Ramp-up time**: New hires take time to become productive.
- **Diminishing Returns**: Beyond an optimal team size, productivity may not increase proportionally.
- **Team Dynamics**: Larger teams may lead to **conflicts** or **diffused accountability**.

Recommendation:

- Start with **5–6 skilled members** (efficient and manageable).
- Scale up only if necessary, ensuring proper onboarding and role clarity.

b) What is the trigger for communicating the project plan, and who is the audience? Why is communicating the project plan important?,(2) (20-21/19-20)

- **Trigger**: The project plan should be communicated **after approval** (when scope, budget, and timelines are finalized).
- **Audience**:
    - **Stakeholders** (sponsors, clients) – to set expectations.
    - **Team members** – for clarity on tasks and deadlines.
    - **Other departments** (e.g., IT, finance) – if coordination is needed.
- **Why It's Important**:
    - Ensures **alignment** on goals and deliverables.
    - Reduces **miscommunication** and **scope creep**.
    - Builds **trust** with stakeholders.

c)Show the business factors that are driving system analysis. Based on these factors, what should system analysis address?(2) (20-21/19-20)

Business Factors:

1. **Competitive Pressure** (Need for faster, better systems).
2. **Regulatory Compliance** (Legal/industry requirements).
3. **Cost Reduction** (Automation to cut operational expenses).
4. **Customer Demand** (Enhancing user experience).
5. **Technology Advancements** (Adopting new tools for efficiency).

What System Analysis Should Address:

- **Feasibility** (Can it be done within constraints?).
- **Requirements** (What do users/stakeholders need?).
- **Process Improvements** (How can workflows be optimized?).
- **Risk Assessment** (What could go wrong?).

d) Briefly describe about the eight major activities in the project management life cycle.(5) (20-21/19-20)

1. Project Initiation

Purpose:

- Define the project's **justification, objectives, and feasibility**.
- Identify **stakeholders** and secure approval to proceed.

2. Planning

Purpose:

- Create a **roadmap** for executing, monitoring, and controlling the project.

Key Tasks:

- Define **scope** (Work Breakdown Structure - WBS).
- Develop **schedule** (Gantt charts, critical path analysis).
- Estimate **budget** (cost management plan).

- Identify **risks** (risk assessment & mitigation strategies).

3. Requirements Gathering

Purpose:

- Understand **what the system must do** to meet stakeholder needs.

Key Tasks:

- Conduct **interviews, surveys, and workshops** with users.
- Document **functional & non-functional requirements**.

4. Design

Purpose:

- Plan **how the system will be built** (architecture, interfaces, data flow).

Key Tasks:

- **System Architecture** (monolithic vs. microservices).
- **UI/UX Design** (wireframes, prototypes).
- **Database Design** (ER diagrams, schema).
- **Technical Specifications** (APIs, frameworks, security).

5. Development

Purpose:

- Build the system based on **design specifications**.

Key Tasks:

- **Coding** (following coding standards).
- **Configuration** (setting up servers, databases).
- **Version Control** (Git, SVN).
- **Continuous Integration** (automated builds/testing).

6. Testing

Purpose:

- Ensure the system **works correctly** and **meets requirements**.

Key Tasks:

- **Unit Testing** (individual components).
- **Integration Testing** (modules working together).
- **System Testing** (full system validation).
- **Bug Fixing & Regression Testing** (ensure fixes don't break other features).

7. Deployment

Purpose:

- Release the system to **end-users**.

Key Tasks:

- **Data Migration** (transferring old system data).
- **User Training** (manuals, workshops).
- **Go-Live Support** (monitoring initial usage).

8. Closure

Purpose:

- Officially **end the project**, document lessons, and transition to operations.

Key Tasks:

- **Final Deliverable Handover** (to maintenance team).
- **Post-Mortem Review** (what went well, what didn't).
- **Stakeholder Sign-Off** (formal acceptance).
- **Release Resources** (team reassignment).

e) Which responsibility project managers do to manage changes that occur and/or are requested during a project? List out the factors to consider in estimating task durations.(3) (20-21/19-20)

Changes manage করার জন্য Project Manager-এর দায়িত্ব:

1. **Change Control Process স্থাপন করা** – পরিবর্তনের অনুরোধ গ্রহণ ও পর্যালোচনার জন্য একটি প্রক্রিয়া তৈরি করা।
2. **Impact Assessment করা** – পরিবর্তনের প্রভাব (সময়, বাজেট, স্কোপ ইত্যাদি) বিশ্লেষণ করা।
3. **Stakeholder Approval নেওয়া** – পরিবর্তন বাস্তবায়নের আগে সংশ্লিষ্ট পক্ষগুলোর অনুমোদন নেওয়া।
4. **Project Plan আপডেট করা** – পরিবর্তন অনুযায়ী প্রকল্প পরিকল্পনা সংশোধন করা।
5. **Communication নিশ্চিত করা** – পরিবর্তন সম্পর্কে টিম এবং স্টেকহোল্ডারদের জানানো।
6. **Documentation করা** – সব পরিবর্তন ও তার সিদ্ধান্ত নথিভুক্ত রাখা।

Task Duration **অনুমানের জন্য যেসব বিষয় বিবেচনা করতে হয়** (Factors to Consider in Estimating Task Durations):

1. **Task Complexity (কাজের জটিলতা)**
2. **Resource Skill Level (কর্মীদের দক্ষতা ও অভিজ্ঞতা)**
3. **Past Project Data (পূর্বের প্রকল্পের তথ্য)**
4. **Availability of Resources (রিসোর্স কতটা সময়ের জন্য পাওয়া যাবে)**
5. **Dependencies (অন্য টাস্কের ওপর নির্ভরতা)**
6. **Assumptions and Risks (অনুমান ও ঝুঁকি)**
7. **Working Hours and Calendars (কাজের সময়সূচি ও ছুটি)**
8. **Team Efficiency (দলের কাজের গতি ও মান)**

3->a) What is the objective of refining the Use-Case Model in object design? Why is it important? (20-21/19-20)

Objective:

- The main goal of refining the use-case model in object design is to **identify and define the classes, objects, and their responsibilities** in the system based on how users interact with the system (use-cases).

Importance:

- Ensures that the system's design **accurately reflects user needs**.
- Helps in identifying **key interactions**, objects, and behaviors early.

- Provides a **blueprint for code development** and helps maintain **traceability** from requirements to design.

উদ্দেশ্য:

- Use-case model পরিশোধনের মাধ্যমে আমরা সিস্টেমে **ব্যবহারকারী কীভাবে কাজ করে** তা বিশ্লেষণ করে **ক্লাস, অবজেক্ট ও তাদের দায়িত্ব নির্ধারণ** করি।

গুরুত্ব:

- এটি নিশ্চিত করে যে ডিজাইনটি **ব্যবহারকারীর চাহিদার সাথে মিল রয়েছে।**
- **কোড লেখার জন্য রূপরেখা** প্রদান করে।
- রিকোয়ারমেন্ট থেকে ডিজাইন পর্যন্ত **ট্রেসেবিলিটি বজায় রাখে।**

b)Why do many new systems analysts fail to effectively analyze problems? What can they do to become more effective? Show the categories of resources to be allocated to the project.(4) (20-21/19-20)

Reasons for failure:

5. Lack of domain knowledge.
6. Inadequate communication with stakeholders.
7. Over-reliance on tools instead of understanding the actual problem.
8. Ignoring user feedback or real-world constraints.

How to become effective:

- Gain deep **understanding of the business domain**.
- Improve **interpersonal and communication skills**.
- Practice **active listening** and **requirement gathering techniques**.
- Use **structured methodologies and tools** (e.g., DFDs, UML).
- Validate assumptions with **real user feedback** and prototypes.

Categories of resources to be allocated to the project:

1. **Human Resources** – Developers, testers, analysts, managers.
2. **Hardware Resources** – Servers, computers, network equipment.
3. **Software Resources** – IDEs, databases, OS, tools.
4. **Financial Resources** – Budget for labor, procurement, etc.
5. **Time Resources** – Estimated schedule and deadlines.

c)Show the commonly used technique for prioritizing system requirements.(2) (20-21/19-20)

MoSCoW Method

- **Must Have (M)** – Critical for success (e.g., login functionality).
- **Should Have (S)** – Important but not urgent (e.g., password reset).
- **Could Have (C)** – Nice-to-have if resources allow (e.g., dark mode).
- **Won't Have (W)** – Excluded from current scope.

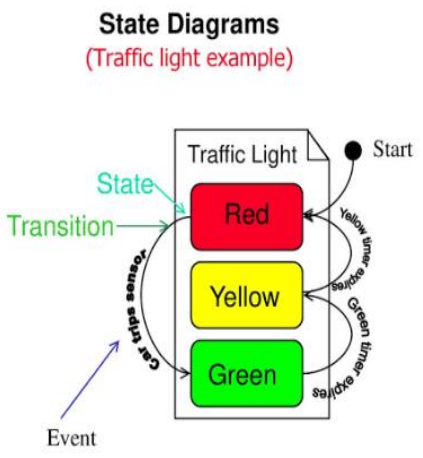
d) Describe the steps needed to construct the state chart diagram. Show the relationship between an object state and state transition event.(3) (20-21/19-20)

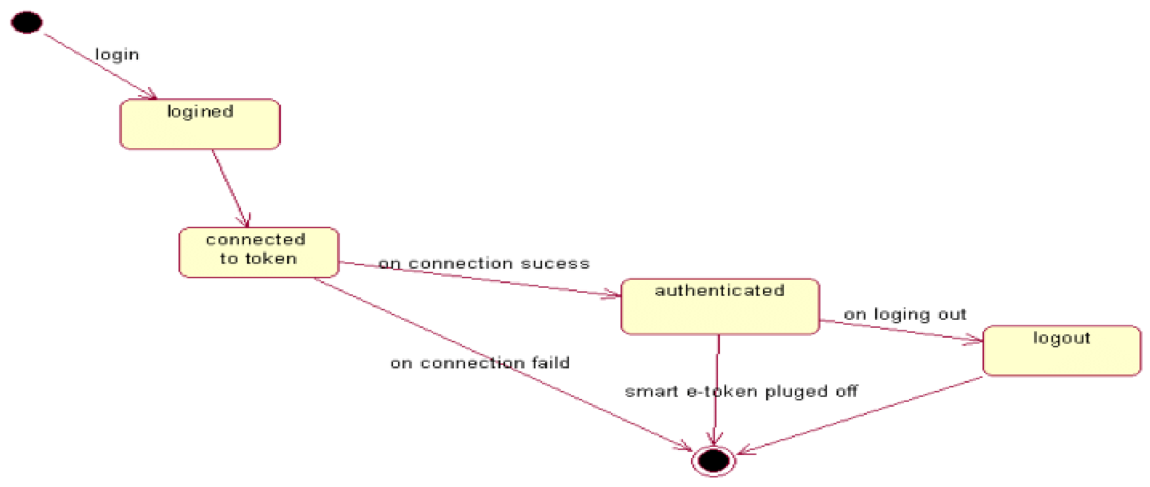
Steps to construct a state chart diagram:

1. Identify the **class** or **object** whose states are being modeled.
2. List all possible **states** the object can be in.
3. Identify **events or conditions** that trigger state changes.
4. Define **transitions** between states with associated **events/actions**.
5. Add **entry/exit actions**, if applicable.
6. Draw the diagram with:
    - **Rounded rectangles** for states.
    - **Arrows** for transitions.
    - **Labels** for events causing transitions.

Relationship Between Object State & Transition Event:

- A **state** represents a condition (e.g., "*Shipped*").
- A **transition** is triggered by an **event** (e.g., "*Dispatch Order*").





e) Define the visibility in object-oriented design. Explain the different levels of visibility. Why are the three kinds of objects needed in object-oriented design?(3) (20-21/19-20/18-19)

Visibility defines **how and where an object's attributes or methods can be accessed** by other objects or classes.

Levels of Visibility:

1. **Public (+)** – Accessible from anywhere.
2. **Private (-)** – Accessible only within the class.
3. **Protected (#)** – Accessible within the class and its subclasses.
4. **Package (default)** – Accessible only within the same package (in Java-like languages).

Why Are 3 Object Types Needed?

1. **Entity Objects** – Represent business data (e.g., *Customer, Order*).
2. **Boundary Objects** – Handle UI/system interactions (e.g., *Login Screen, API Gateway*).
3. **Control Objects** – Manage business logic (e.g., *PaymentProcessor*).

19-20

1 a) **How do communication and collaboration systems improve efficiency and effectiveness? What are some of the communication and collaboration systems that are being used by an increasing number of organizations?**

Communication/Collaboration Systems Benefits & Examples

Efficiency/Effectiveness Improvements:

- Reduce email overload through organized messaging (Slack/Microsoft Teams)
- Enable real-time document co-editing (Google Workspace)
- Automate approval workflows (ServiceNow)
- Provide unified customer communication channels (Zendesk)

Growing Adoption Systems:

- Team collaboration: Microsoft Teams (used by 1M+ organizations)
- Project management: Asana, Trello
- Video conferencing: Zoom (300M+ daily participants)
- Internal knowledge bases: Confluence

c) **What kind of knowledge and skills should a system analyst possess?**

  - Working knowledge of IT and programming.
  - General business knowledge.
  - Problem-solving, communication, interpersonal skills.

Flexibility, adaptability, and ethical judgment

18-19

1)a)Suppose your company has a contract with a XYZ local firm to link all of their system so they can transparently work together. Their applications include a number of existing legacy systems, which were built at different times by different developers using a variety of languages and platforms, as well as several newer contemporary applications. What is the term for this type of linking? What type of tool would you most likely use, and shows some examples of these tools?

The term for linking disparate systems so they can work together transparently is **Enterprise Application Integration (EAI)** or **System Integration**.

Since the systems in question include legacy applications built on different platforms and newer applications, the most suitable approach is **Middleware-based Integration** or **Service-**

Oriented Architecture (SOA). The tool used for this purpose is typically an **Enterprise Service Bus (ESB)** or an **Integration Platform as a Service (iPaaS)**.

Basic Terms:

1. **Enterprise Application Integration (EAI)** – **এন্টারপ্রাইজ অ্যাপ্লিকেশন ইন্টিগ্রেশন** (বিভিন্ন সিস্টেমকে একসাথে যুক্ত করার পদ্ধতি)
2. **System Integration** – **সিস্টেম ইন্টিগ্রেশন** (বিভিন্ন সফটওয়্যার ও হার্ডওয়্যারকে একসাথে কাজ করানো)
3. **Legacy Systems** – **লিগ্যাসি সিস্টেম** (পুরানো, কিন্তু এখনও ব্যবহৃত সফটওয়্যার)
4. **Middleware** – **মিডলওয়্যার** (সফটওয়্যার যা বিভিন্ন অ্যাপ্লিকেশনকে সংযুক্ত করে)
5. **Service-Oriented Architecture (SOA)** – **সার্ভিস-ওরিয়েন্টেড আর্কিটেকচার** (সেবা-ভিত্তিক সফটওয়্যার ডিজাইন)

Tools & Technologies:

6. **Enterprise Service Bus (ESB)** – **এন্টারপ্রাইজ সার্ভিস বাস** (একটি সফটওয়্যার যা বিভিন্ন অ্যাপ্লিকেশনকে যুক্ত করে)
   - Example: **MuleSoft** – **মিউলসফট**
   - Example: **IBM App Connect** – **আইবিএম অ্যাপ কানেক্ট**
7. **Integration Platform as a Service (iPaaS)** – **ইন্টিগ্রেশন প্ল্যাটফর্ম অ্যাজ আ সার্ভিস** (ক্লাউড-ভিত্তিক সিস্টেম সংযোগের টুল)
   - Example: **Boomi** – **বুমি**
   - Example: **Azure Logic Apps** – **অ্যাজুর লজিক অ্যাপস**
8. **API Gateway** – **এপিআই গেটওয়ে** (একটি সিস্টেম যা বিভিন্ন API কে ম্যানেজ করে)
   - Example: **Kong** – **কং**
   - Example: **Apigee** – **এপিজি**
9. **Message Broker (e.g., Kafka, RabbitMQ)** – **মেসেজ ব্রোকার** (সফটওয়্যার যা বিভিন্ন সিস্টেমের মধ্যে ডাটা আদান-প্রদান করে)
   - Example: **Apache Kafka** – **অ্যাপাচি কাফকা**
   - Example: **RabbitMQ** – **র‍্যাবিট এমকিউ**

3) a)Discuss about the main activities of object-oriented design. Show the steps needed to construct the state chart diagram.

Object-Oriented Design (OOD) focuses on defining classes, objects, and their relationships to create a modular, reusable, and maintainable software system. The key activities include:

1. Identifying Objects and Classes

- Recognize real-world entities and model them as **objects**.
- Group similar objects into **classes** (e.g., `Customer`, `Order`, `Product`).
- Define **attributes (data)** and **methods (behavior)** for each class.

2. Defining Class Relationships

- **Inheritance** (Is-A relationship) – Child classes extend parent classes (e.g., `Car` extends `Vehicle`).
- **Association** (Has-A relationship) – Objects interact with each other (e.g., `Student` enrolls in a `Course`).
- **Aggregation & Composition** – Special forms of association (weak vs. strong ownership).

3. Applying Design Principles

- **SOLID Principles:**
  - **Single Responsibility Principle (SRP)** – A class should have only one reason to change.
  - **Open/Closed Principle (OCP)** – Classes should be open for extension but closed for modification.
  - **Liskov Substitution Principle (LSP)** – Subclasses should be replaceable with their parent class.
  - **Interface Segregation Principle (ISP)** – Clients should not be forced to depend on unused interfaces.

- **Dependency Inversion Principle (DIP)** – Depend on abstractions, not concrete implementations.

4. Designing Interactions Between Objects

- Define how objects communicate (e.g., method calls, events).
- Use **design patterns** (e.g., Factory, Observer, Singleton) for reusable solutions.

5. Encapsulation & Data Hiding

- Restrict direct access to an object's data (use **getters/setters**).
- Expose only necessary functionalities (**abstraction**).

6. Modeling System Behavior

- Use **UML diagrams** (e.g., Class, Sequence, State Diagrams) to visualize design.
- Define **use cases** and **scenarios** for system interactions.

7. Refining and Optimizing the Design

- Remove redundant classes/methods (**refactoring**).
- Ensure **low coupling** (minimize dependencies) and **high cohesion** (related functionalities stay together).

Steps to construct a state chart diagram:

1. Identify the **class** or **object** whose states are being modeled.
2. List all possible **states** the object can be in.
3. Identify **events or conditions** that trigger state changes.
4. Define **transitions** between states with associated **events/actions**.
5. Add **entry/exit actions**, if applicable.
6. Draw the diagram with:
   - **Rounded rectangles** for states.
   - **Arrows** for transitions.
   - **Labels** for events causing transitions.

b)What is visibility in object-oriented design??explain the different levels of visibility.

Visibility defines **how and where an object's attributes or methods can be accessed** by other objects or classes.

Levels of Visibility:

1. **Public (+)** – Accessible from anywhere.
2. **Private (-)** – Accessible only within the class.
3. **Protected (#)** – Accessible within the class and its subclasses.
4. **Package (default)** – Accessible only within the same package (in Java-like languages).

C) Show the key reason for object reusability. Which methods developer's use to achieve object reusability?

Key Reason:

Save time and effort by using existing, tested objects instead of rewriting code.

Methods to Achieve Reusability:

1. **Inheritance** – Child classes reuse parent class features.
2. **Composition** – Combine objects (e.g., a `Car` uses an `Engine` object).
3. **Design Patterns** – Reuse proven solutions (e.g., Singleton, Factory).
4. **Libraries/Frameworks** – Use pre-built code (e.g., Java Collections, React Components).

d)Describe the goal of constructing object robustness diagrams? What are the components of those diagrams?

Goal:

Bridge the gap between requirements (use cases) and detailed design by identifying objects and their interactions early.

Components:

1. **Actors** – Users or external systems.

2. **Boundary Objects** – UI elements (e.g., screens, forms).
3. **Entity Objects** – Core data/logic (e.g., `Customer`, `Order`).
4. **Control Objects** – Manage workflows (e.g., `PaymentController`).

Example:

- **Use Case:** "User places an order"
- **Robustness Diagram:**
  - **Actor:** Customer
  - **Boundary:** OrderForm (UI)
  - **Control:** OrderProcessor
  - **Entity:** Order, Product

e)Compare the relationship between an object state and state transition event?

Aspect	Object State	State Transition Event
Definition	The current condition or mode of an object at a given time.	An occurrence that triggers a change in the object's state.
Example	A `Door` can be **Open**, **Closed**, or **Locked**.	The event **"KeyTurned"** changes the `Door` from **Locked** → **Unlocked**.
Representation	Stored as **attributes/variables** (e.g., `state = "open"`).	Represented as **methods** or **external triggers** (e.g., `unlock()`).
Role in OOP	Describes **what the object is** at a moment.	Describes **how the object changes** over time.
Dependency	Passive (exists as data).	Active (causes state modification).
UML Diagrams	Shown in **State Machine Diagrams**.	Represented as **transitions (arrows)** between states.

Aspect	**Object State**	**State Transition Event**
Trigger	N/A (static until an event occurs).	Caused by **method calls**, **user actions**, or **external signals**.
Example Code	`python<br>class Door:<br> def __init__(self):<br> self.state = "closed"<br>`	`python<br>def unlock(self):<br> if self.state == "locked":<br> self.state = "unlocked"<br>`