1. In a Scrum-based software development project, the Product Owner has defined the following user stories for an e-commerce application:

- As a user, I want to log in securely so that I can access my account.
- As a user, I want to search for products by category to find items easily.

I. Create a product backlog for these user stories by breaking them into tasks.
II. Describe how the development team can prioritize these user stories during a Sprint Planning meeting, considering value to the customer and technical feasibility.
III. Illustrate how these tasks will be tracked using a Scrum board. Use proper terms like "To Do," "In Progress," and "Done." **(6)**

Ans: to the question no-1

I

User Story 1:

"As a user, I want to log in securely so that I can access my account."
Tasks:

- **Design login page UI** (To Do)
- **Set up secure backend authentication mechanism (e.g., JWT or OAuth)** (To Do)
- **Implement input validation for email/password** (To Do)
- **Integrate secure communication protocols (e.g., HTTPS)** (To Do)
- **Create unit tests for login functionality** (To Do)
- **Conduct security tests to prevent vulnerabilities (e.g., SQL injection, XSS)** (To Do)

User Story 2:

"As a user, I want to search for products by category to find items easily."
Tasks:

- **Design search bar UI with category dropdown** (To Do)
- **Build database schema for product categories** (To Do)
- **Develop search API to fetch products by category** (To Do)
- **Integrate API with the frontend search functionality** (To Do)
- **Add error handling for no results found** (To Do)
- **Perform usability tests for the search feature** (To Do)

II

Prioritization During Sprint Planning

1) **Value to the Customer**:
    a) **User Story 1 (Login Securely)** is a high-priority feature since users cannot access the application without secure login.

    b) **User Story 2 (Search for Products)** provides utility but is secondary compared to user authentication.
2) **Technical Feasibility**:
    a) Tasks related to **User Story 1** involve foundational work (e.g., backend authentication, HTTPS setup) and are likely prerequisites for other features.
    b) Tasks for **User Story 2** require search API development, which can be complex but not critical for the first Sprint.
3) **Sprint Planning Decision**:
    a) Prioritize all tasks of **User Story 1** in the current Sprint.
    b) Begin **User Story 2** by tackling simpler tasks (e.g., UI design and database schema setup).
    c) Defer more complex tasks of **User Story 2** to a subsequent Sprint.

Tracking Tasks Using a Scrum Board

Columns:

- **To Do**: All tasks not yet started.
- **In Progress**: Tasks being actively worked on by the team.
- **Done**: Completed tasks verified by QA or team review.

Example Scrum Board:

Task	To Do	In Progress	Done
Design login page UI	✅		
Set up secure backend authentication	✅		
Implement input validation	✅		
Integrate secure communication	✅		
Create unit tests for login	✅		
Conduct security tests	✅		
Design search bar UI	✅		
Build database schema	✅		

How Tasks Progress During the Sprint

- Team members move tasks from **To Do** to **In Progress** when starting work.
- After completing a task, it moves to the **Done** column once verified.
- At the end of the Sprint, completed tasks for **User Story 1** ensure it is fully delivered, while partially completed **User Story 2** tasks can be reprioritized.

2. A software development team is about to start a project for a new innovative product. The project has several high-risk components due to its novelty, and there's uncertainty regarding the client's future needs. The client is open to iterative changes, but the team must ensure that the software evolves in a manageable, cost-effective way.
    - Considering the high risks and the evolving nature of the client's needs, discuss how the Spiral, Agile, and Extreme methodologies address risk management and adaptability. Which methodology would be the most suitable for a project with significant risk and evolving requirements, and why? **(6)**

Methodologies and Their Approach to Risk Management and Adaptability

i. Spiral Methodology

- **Risk Management**:
    - The Spiral model explicitly focuses on risk assessment at each phase.
    - Each iteration (or "spiral") begins with identifying and addressing the most significant risks, ensuring high-risk components are tackled early.
    - Prototyping and stakeholder feedback at every iteration reduce uncertainties and clarify client needs.
- **Adaptability**:
    - Allows incremental development and client feedback at each iteration, making it possible to adjust requirements and plans.
    - However, the structured approach can be slower compared to Agile methods and may be less responsive to rapidly changing requirements.

ii. Agile Methodology

- **Risk Management**:
    - Risk is mitigated by delivering working software in small, incremental sprints, allowing the team to address issues early.
    - Continuous feedback loops ensure that potential risks, such as misaligned client expectations, are identified and corrected quickly.
- **Adaptability**:
    - Highly flexible, Agile thrives in environments with evolving requirements.
    - Agile's iterative nature and focus on collaboration make it easy to adapt to changes while keeping development aligned with client needs.

iii. Extreme Programming (XP)

- **Risk Management**:
    - XP reduces technical risks through practices like Test-Driven Development (TDD), pair programming, and continuous integration.
    - Frequent releases minimize the risk of building the wrong product by ensuring ongoing alignment with client expectations.
- **Adaptability**:

    - XP's emphasis on constant feedback and close collaboration with the client allows for rapid adaptation to new or changing requirements.
    - Practices like refactoring ensure the codebase remains maintainable and adaptable as the product evolves.

Most Suitable Methodology

Recommendation: Agile Methodology

Reasons:

- **Evolving Requirements**:
    - Agile is specifically designed to handle projects with uncertain and changing requirements. Its iterative nature ensures that the software evolves alongside the client's needs.
- **Risk Management**:
    - By delivering incremental value in each sprint, Agile minimizes the impact of high-risk components and ensures continuous alignment with client expectations.
- **Flexibility and Collaboration**:
    - Agile allows for seamless collaboration between the development team and the client, which is crucial for refining the product based on feedback and reducing uncertainties.

Comparison:

- While the **Spiral Model** excels in risk management, its slower, structured approach may not suit rapidly evolving requirements.
- **Extreme Programming (XP)** is highly adaptable, but it requires an exceptionally disciplined team and may not address high-level risks as comprehensively as Agile.

Agile strikes the right balance between **risk management** and **adaptability**, making it the most effective choice for this project.

3. A company is working on two different projects. Project A has well-defined requirements and a strict deadline, while Project B has evolving requirements with an uncertain timeline and continuous customer feedback. Both projects involve high stakes, and the team must decide which development methodology to use.
   - Compare and contrast the Waterfall, Agile, Extreme, and Spiral development models. Based on the characteristics of both projects (Project A and Project B), which methodology would best suit each? Support your answer with a detailed analysis of how each methodology would address the specific needs of the projects, considering factors such as predictability, customer collaboration, and risk management. **(6)**

Ans: to the question no-3

Comparison of Methodologies

Aspect	Waterfall	Agile	Extreme Programming (XP)	Spiral
Adaptability	Low	High	Very High	Moderate
Predictability	High	Moderate	Low	Moderate
Risk Management	Moderate	High	High	Very High
Best For	Fixed requirements	Evolving needs	Frequent releases	High-risk projects

Methodology Recommendations for Projects

Project A:

- **Well-defined requirements and strict deadline.**
- **Recommended Methodology**: **Waterfall Model**

Reasoning:

- **Predictability**: The linear structure and well-defined phases ensure clear timelines and deliverables, making it easier to meet the strict deadline.
- **Stable Requirements**: With no significant changes expected, the rigidity of the Waterfall model is not a disadvantage.
- **Risk Management**: Since requirements are fixed, the project's risks can be addressed upfront during the planning phase.

Project B:

- **Evolving requirements, uncertain timeline, and continuous feedback.**
- **Recommended Methodology**: **Agile Methodology**

Reasoning:

- **Adaptability**: Agile thrives in environments where requirements are dynamic and customer feedback is frequent.
- **Customer Collaboration**: Agile fosters continuous communication with the client, ensuring the product evolves according to their needs.
- **Risk Management**: Incremental delivery allows risks to be identified and mitigated early in the development process.

4. Explain the principles of software engineering ethics, highlighting the issues related to professional responsibility. Discuss how the ACM/IEEE Code of Ethics guides ethical decision-making in software engineering practices. **(4)**

Ans: to the question no-4

Principles of Software Engineering Ethics

- **Public Interest**: Prioritize safety, privacy, and welfare.
- **Client and Employer**: Serve clients/employers while protecting public interest.
- **Product Quality**: Ensure high-quality, reliable, and secure software.
- **Professional Integrity**: Maintain honesty and avoid conflicts of interest.
- **Fairness**: Avoid bias and treat everyone fairly.

Issues in Professional Responsibility

- **Accountability**: Responsibility for software outcomes.
- **Confidentiality**: Protect sensitive information.
- **Bias**: Avoid discriminatory designs.,

Role of ACM/IEEE Code of Ethics

- **Public Good**: Focus on societal benefit and harm avoidance.
- **Competence**: Work within expertise to ensure quality.
- **Transparency**: Communicate risks openly.
- **Conflict Resolution**: Address ethical dilemmas collaboratively and ethically.

The code guides ethical decision-making to ensure professional standards and societal values are upheld.

5. Given the story of the Airport Reservation System, identify at least five **functional** and five **non-functional** requirements for the system. In your answer, explain how each requirement contributes to the overall performance, usability, and security of the system. Consider factors such as performance, user experience, and system maintenance in your discussion. **(4)**

Functional Requirements

1. **User Registration and Login**:
    - Enables secure access for passengers and staff.
    - Contributes to **security** by restricting unauthorized access.
2. **Flight Search and Booking**:
    - Allows users to search and book flights based on preferences.
    - Enhances **usability** by providing easy access to available flights.
3. **Payment Gateway Integration**:
    - Facilitates secure online payments for tickets.
    - Supports **performance** by ensuring smooth transaction processing.
4. **Real-time Flight Updates**:
    - Provides users with updates on delays, cancellations, or gate changes.
    - Improves **user experience** by keeping users informed.
5. **Customer Support System**:
    - Offers chat or call support for resolving issues.
    - Enhances **usability** by addressing user concerns promptly.

Non-Functional Requirements

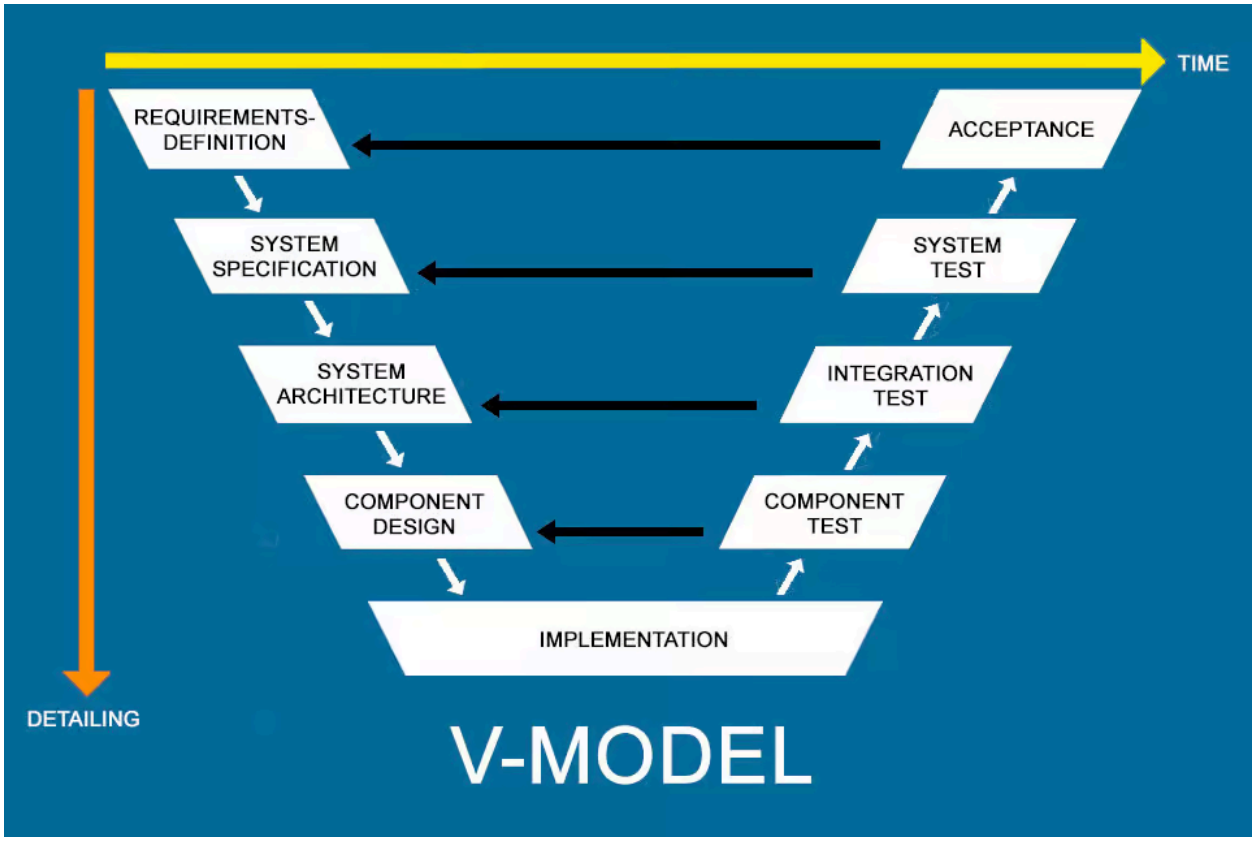
1. **System Availability**:
    - Ensures 24/7 access to the system.
    - Contributes to **performance** and user trust.
2. **Scalability**:
    - Handles a large number of simultaneous users during peak hours.
    - Supports **system maintenance** and adaptability.
3. **Response Time**:
    - Processes user actions (e.g., search or payment) within 2 seconds.
    - Enhances **usability** by ensuring smooth interactions.
4. **Data Security**:
    - Encrypts sensitive information like personal details and payment data.
    - Strengthens **security** and builds user confidence.
5. **Maintainability**:
    - Allows easy updates for adding new features or fixing bugs.
    - Supports **system maintenance** and reduces downtime.

By addressing these functional and non-functional requirements, the system ensures a secure, efficient, and user-friendly experience while remaining robust and adaptable for future needs.

6. Illustrate and explain the V-model of testing phases in a plan-driven software process, detailing the relationships between development activities and corresponding testing activities. **(4)**

The V-Model of Testing Phases

The **V-Model** (Verification and Validation model) is a plan-driven approach where development and testing activities are closely aligned. It emphasizes the relationship between each development phase and its corresponding testing phase, forming a "V" shape.



Explanation

1. **Left Side (Development Activities)**:
    - **Requirements**: Define the system's functionality and user expectations.
    - **System Design**: Break down requirements into detailed specifications.
    - **Architectural Design**: Define the system structure and module interactions.
    - **Module Design**: Specify the functionality of individual components.
    - **Coding**: Implement the designs into actual code.
2. **Right Side (Testing Activities)**:
    - **Acceptance Testing**: Validates the system against user requirements.
    - **System Testing**: Tests the complete system for overall functionality.
    - **Integration Testing**: Verifies the interactions between modules.
    - **Unit Testing**: Tests individual modules or components for correctness.

Relationships

- Each testing activity on the right corresponds directly to a development activity on the left.
- For example:
    - **Requirements ↔ Acceptance Testing**: Ensures the final product meets the initial requirements.
    - **Module Design ↔ Unit Testing**: Verifies the correctness of individual modules as per their design.

The V-Model ensures that testing is planned early and conducted methodically, reducing defects and aligning with the overall project goals.

7. Explain the process of prototype development in software engineering. Discuss the key stages involved in creating a prototype and how it helps in refining software requirements. Analyze the benefits of using the prototyping model, particularly in terms of user feedback, risk reduction, and iterative development. **(4)**

Prototype Development Process

1. **Requirement Identification**: Gather initial, essential requirements.
2. **Prototype Design and Development**: Build a basic version with core functionality.
3. **User Evaluation and Feedback**: Present the prototype to users for feedback.
4. **Refinement and Iteration**: Modify the prototype based on feedback and refine it.
5. **Final System Development**: Develop the complete system based on refined requirements.

Benefits of Prototyping

1. **User Feedback**: Continuous input ensures the system meets user needs.
2. **Risk Reduction**: Identifies and resolves issues early, preventing costly revisions.
3. **Iterative Development**: Allows for ongoing refinement, adapting to evolving requirements.

Prototyping helps clarify requirements early, reduces risks, and promotes iterative, user-driven development.

8. Explain the process improvement cycle in software engineering and describe its key stages. Name and explain some commonly used process metrics, highlighting how they help in monitoring and improving software processes. **(4)**

Process Improvement Cycle in Software Engineering

1. **Planning**: Define improvement goals and areas needing enhancement.
2. **Measuring**: Collect data on current process performance.
3. **Analyzing**: Identify inefficiencies or issues based on the data.
4. **Improving**: Implement changes to optimize the process.
5. **Controlling**: Monitor the improvements to ensure long-term effectiveness.

Commonly Used Process Metrics

1. **Defect Density**: Measures defects per unit of software size, helping monitor code quality.
2. **Cycle Time**: Tracks time taken for tasks, helping identify delays or inefficiencies.
3. **Code Churn**: Measures code changes, indicating stability or rework needs.
4. **Customer Satisfaction**: Assesses user satisfaction, ensuring product meets needs.
5. **Velocity**: Measures work completed per iteration, helping track team productivity.

These metrics help monitor, evaluate, and continuously improve software processes.

9. Explain the Software Engineering Institute Capability Maturity Model (SEI CMM) and its five levels of capability and maturity. Analyze how each level contributes to improving the software development process and organizational performance. **(4)**

SEI Capability Maturity Model (CMM)

The **Capability Maturity Model (CMM)** provides a framework for assessing and improving software development processes. It is structured across five levels, where each level focuses on enhancing organizational processes and capabilities.

1. **Level 1: Initial**
    - **Characteristics**: Processes are informal, unpredictable, and depend on individual efforts. There is no consistent approach to software development.
    - **Contribution**: This level highlights the need for establishing structured processes to improve the predictability of software delivery.
2. **Level 2: Managed**
    - **Characteristics**: Basic project management processes are introduced, such as tracking cost, schedule, and performance.
    - **Contribution**: The introduction of formal management processes helps reduce project risks and makes software delivery more predictable.
3. **Level 3: Defined**
    - **Characteristics**: Standardized, documented processes are implemented across the organization, ensuring consistency and integration.
    - **Contribution**: Defined processes allow for greater efficiency and consistency, improving quality and reducing errors in the software development lifecycle.
4. **Level 4: Quantitatively Managed**
    - **Characteristics**: Processes are measured and controlled using metrics, allowing for better management of performance and quality.
    - **Contribution**: The use of data-driven decisions enhances process stability and helps identify areas for improvement based on quantitative insights.
5. **Level 5: Optimizing**
    - **Characteristics**: Focus is on continuous process improvement through innovation, learning, and adapting based on performance data.
    - **Contribution**: The organization continuously refines its processes, fostering a culture of improvement that leads to highly optimized, efficient development practices.

10. Describe the core principles of agile software development methods. Analyze how these principles are applied in different software development environments, and assess the benefits and challenges of using agile methods in various project types and organizational settings. **(4)**

Core Principles of Agile Software Development

1. **Customer Collaboration Over Contract Negotiation**
   Emphasizes working closely with customers to meet their evolving needs.
2. **Responding to Change Over Following a Plan**
   Agile values flexibility and adapting to change during development.
3. **Individuals and Interactions Over Processes and Tools**
   Focuses on communication and collaboration among team members.
4. **Working Software Over Comprehensive Documentation**
   Prioritizes delivering functional software over extensive documentation.
5. **Continuous Delivery of Valuable Software**
   Regularly delivers software to provide ongoing value to customers.
6. **Self-Organizing Teams and Sustainable Development Pace**
   Teams manage themselves and maintain a steady, sustainable work pace.

Application in Different Environments

- **Small Projects**: Agile works well due to flexibility and iterative delivery.
- **Large Projects**: Scaled Agile frameworks (e.g., SAFe) can be applied but face challenges with coordination.
- **Startups**: Agile is ideal for quickly iterating and adapting products based on customer feedback.
- **Regulated Environments**: Requires balancing flexibility with compliance requirements.

Benefits of Agile

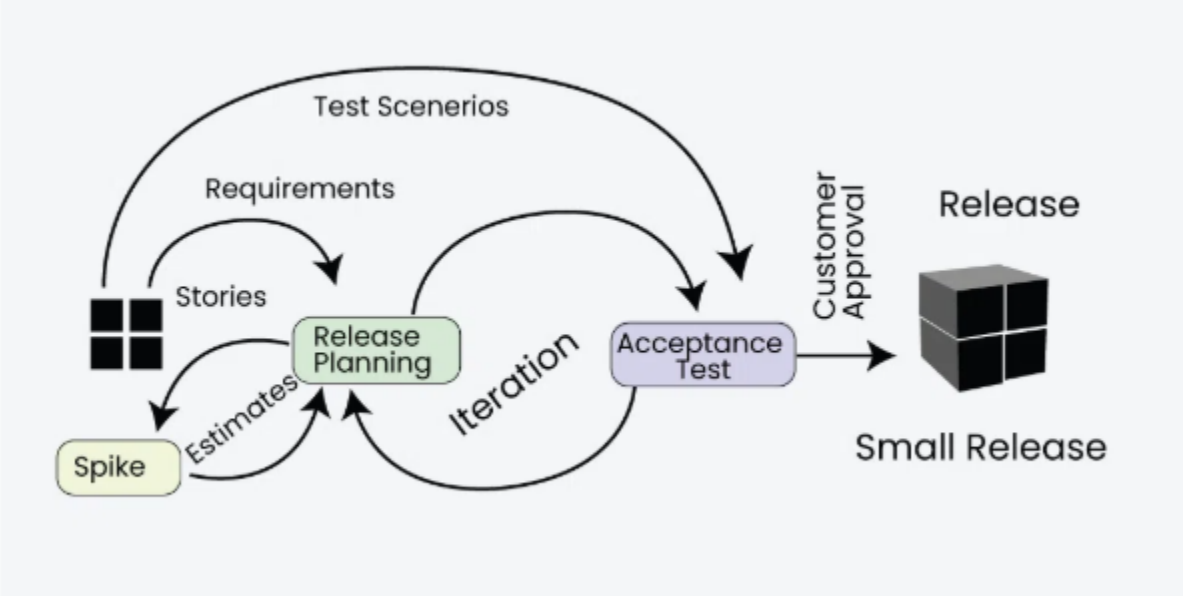
- **Customer Satisfaction**: Frequent feedback and releases ensure alignment with customer needs.
- **Flexibility**: Agile adapts to changing requirements and evolving markets.
- **Faster Delivery**: Short iterations lead to quicker releases and faster time to market.

Challenges of Agile

- **Scope Creep**: Ongoing changes can lead to uncontrolled scope expansion.
- **Coordination**: Large teams may struggle with dependencies and integration.
- **Documentation**: Less emphasis on documentation can be problematic for regulated industries.

11. Draw the release cycle of (Extreme Programming (XP) and explain the influential programming practices. **(4)**

Release cycle of Extreme Programming (XP) is given below:



Extreme Programming (XP) emphasizes flexibility, collaboration, and rapid iterations. Key practices include:

1. **Pair Programming**: Two developers work together, improving code quality and sharing knowledge.
2. **Test-Driven Development (TDD)**: Writing tests before code to ensure correctness and reduce bugs.
3. **Continuous Integration**: Frequently integrating code to avoid issues and run automated tests.
4. **Collective Code Ownership**: All developers are responsible for the entire codebase.
5. **Coding Standards**: Consistent coding practices to ensure clarity and maintainability.
6. **Simple Design**: Keeping the design simple and avoiding unnecessary complexity.
7. **Refactoring**: Continuously improving code structure without changing functionality.
8. **Small Releases**: Delivering software in frequent, small updates for early feedback.
9. **Customer Involvement**: Continuous customer feedback to align with their needs.
10. **Sustainable Pace**: Working at a manageable pace to avoid burnout and ensure long-term productivity.

12. A local library wants to create a digital system to manage its operations. The system will track books, members, and borrowing activities. Each book has attributes like title, author, ISBN, and genre. Members have attributes such as name, membership ID, and contact details. When a member borrows a book, the system records the borrowing date, return due date, and return status. The library also wants to maintain a catalog of overdue books and their respective fines.
    - Using the scenario of a digital library management system, design an Entity-Relationship Diagram (ERD) to represent the entities (e.g., books, members, borrowing activities) and their relationships. Clearly explain the attributes of each entity and how they are interconnected. **(6)**

Answer :

To design an Entity-Relationship Diagram (ERD) for the digital library management system, we'll break down the entities, their attributes, and the relationships between them based on the requirements provided. Here's how it can be structured:

1. Entities and Attributes :

Book

Attributes :

- Book ID (Primary Key) : Unique identifier for each book.
- Title : Title of the book.
- Author : Author of the book.
- ISBN : International Standard Book Number
- Genre : Genre of the book.

Member

- Attributes
- Member ID (Primary Key): Unique identifier for each member.
- Name : Name of the member
- Contact Details : Contact information (phone number, email, etc.)
- Address : Physical address of the member.

Borrowing Activity

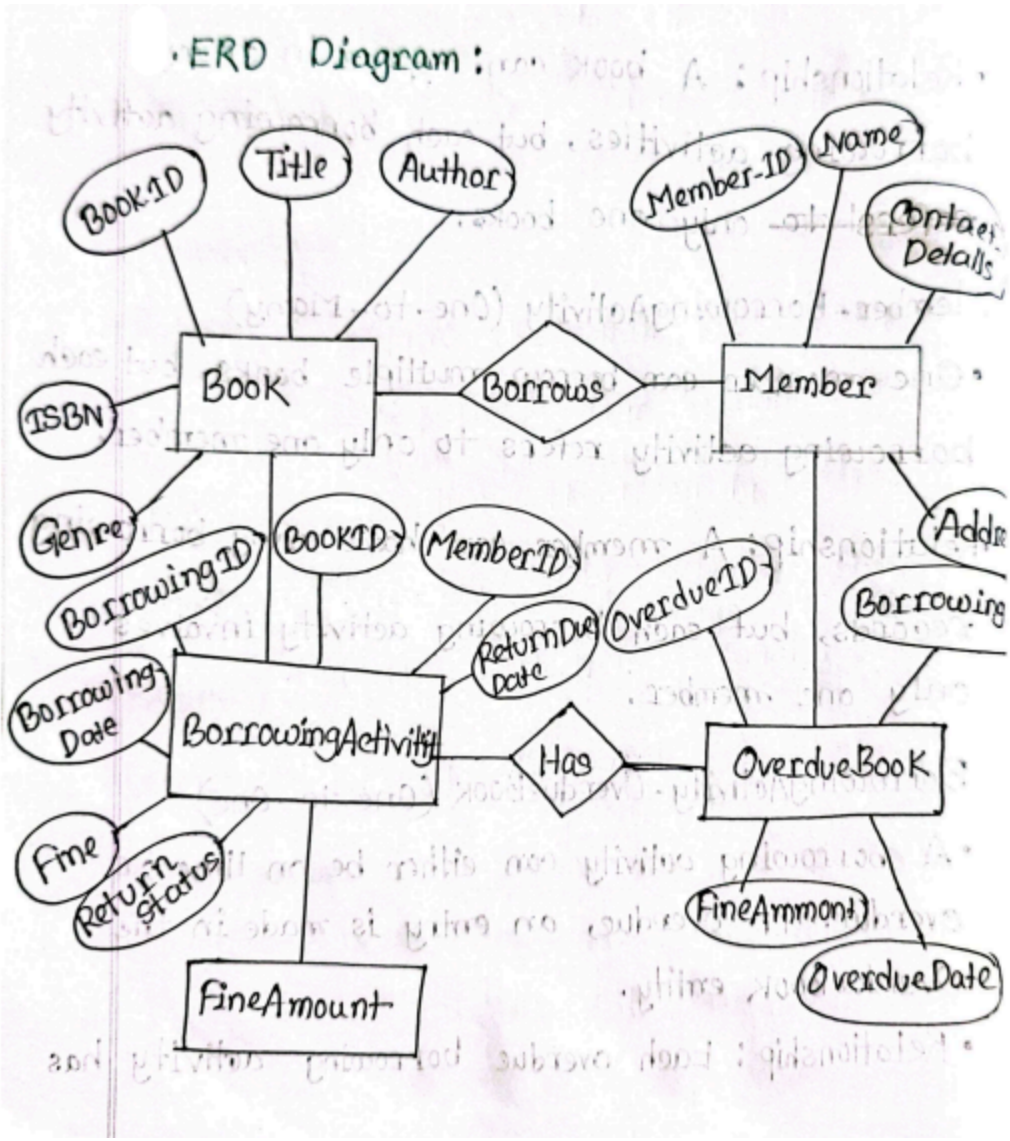
- Attributes:
- BorrowingID (Primary Key) : Unique identifier for each borrowing activity.
- BookID (Foreign Key) : Refers to the borrowed book (links to the Book entity).
- MembemID (Foreign Key) : Refers to the member who borrowed the book (links to the Member entity).
- Borrowing Date : The date the book was borrowed.
- Return Due Date : The date the book is due for return.

- ReturnStatus : Whether the book has been returned or not (could be "Returned" or "Not Returned").
- Fine : The fine imposed if the book is returned late (can be NULL if returned on time).

Overdue Book

- Attributes :
- OverdueID (Primary Key) : Unique identifier for each overdue book entry.
- BorrowingID (Foreign Key) : Refers to the borrowing activity of the overude book (links to the BorrowingActivity entity).
- FineAmount : Fine imposed on the overdue book.
- OverdueDate : Date when the book became Overdue.

ERD Diagram :


BookID
Title
Author
Member-ID
Name
Contact Details
Book
Borrows
Member
ISBN
Genre
BookID
MemberID
BorrowingID
OverdueID
Borrowing
ReturnDue Date
Borrowing Date
BorrowingActivity
Has
OverdueBook
Fine
Return status
FineAmmont
FineAmount
OverdueDate

2. Relationships:

Book - BorrowingActivity (One-to-Many)

- One book can be borrowed many times, but each borrowing activity refers to a specific book.
- Relationship: A book can appear in many borrowing activities, but each borrowing activity involves only one book.

Member - BorrowingActivity (One-to-Many)

- One member can borrow multiple books, but each borrowing activity refers to only one member.
- Relationship: A member can have many borrowing records, but each borrowing activity involves only one member.

BorrowingActivity - OverdueBook (One-to-One)

- A borrowing activity can either be on time or overdue. If overdue, an entry is made in the OverdueBook entity.

13. What is called Testing? Differentiate between Validation and Verification. **(4)**

What is Testing?

Testing is the process of evaluating a system or its components to identify errors, ensure quality, and verify that it meets specified requirements. It involves executing software to detect defects and ensure that the product functions correctly.

Difference between Validation and Verification:

Aspect	Verification	Validation
Definition	Ensures that the product is being built correctly (as per specifications).	Ensures that the right product is being built (meets user needs).
Objective	Checks whether the software conforms to design, requirements, and standards.	Checks whether the software fulfills its intended purpose.
Process Type	Static process (reviews, inspections, walkthroughs).	Dynamic process (actual testing and execution).
Performed During	Development phase.	After development or during testing phase.
Example	Code reviews, design verification.	Functional and user acceptance testing.

This differentiation ensures a clear understanding of software quality assurance processes.

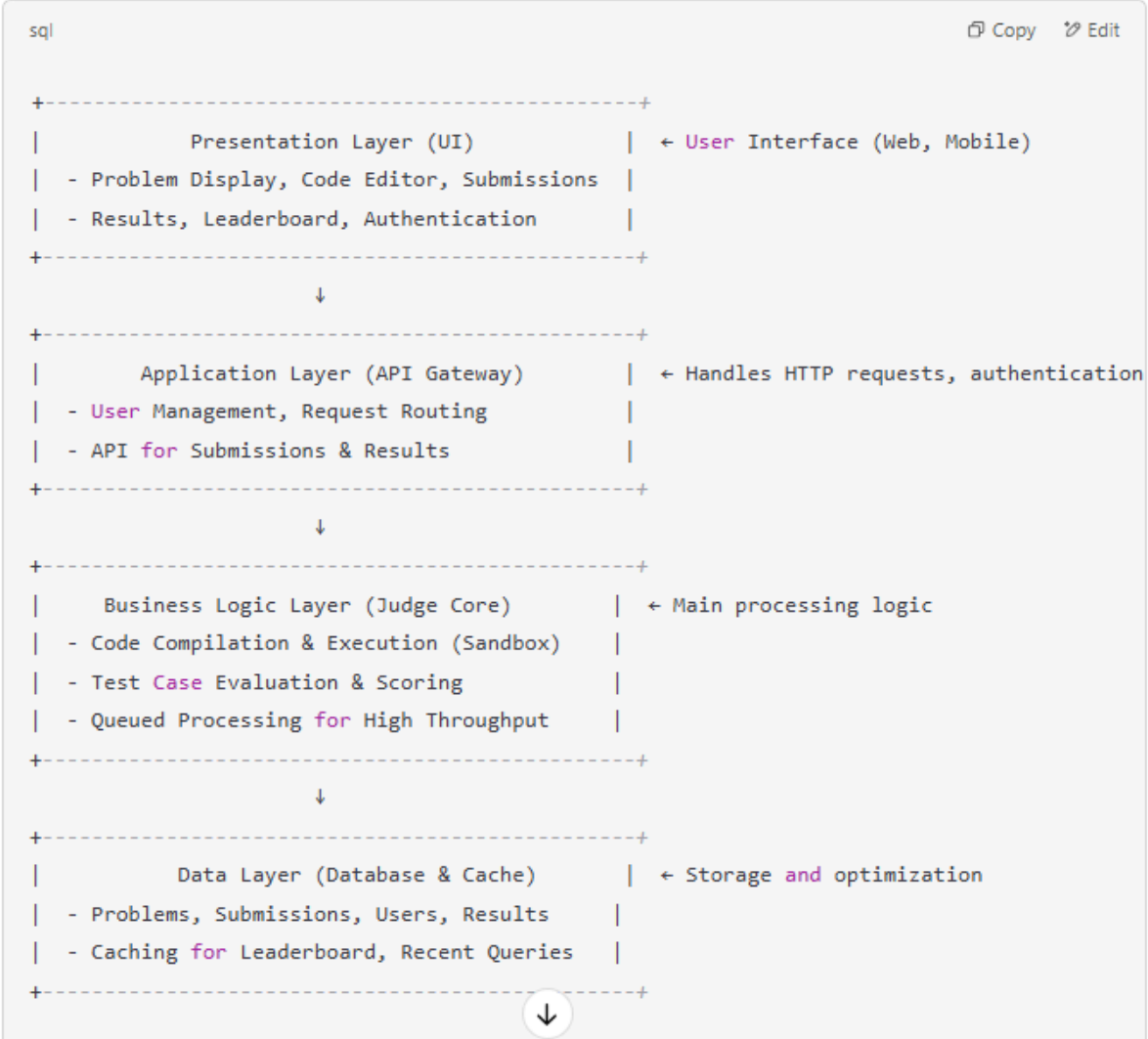
14. Design a layered architecture model for an online judge system, identifying the key layers (e.g., presentation, application, business logic, and data). Explain the responsibilities of each layer and analyze how this architecture ensures scalability, maintainability, and efficient performance. **(4)**

A **layered architecture model** for an **online judge system** consists of:

1. **Presentation Layer (Frontend)** – Handles user interface, problem display, and result viewing. (Technologies: React, Vue.js, REST APIs)
2. **Application Layer (API & Controller)** – Manages authentication, request routing, and user interactions. (Technologies: Django, Spring Boot, Express.js)
3. **Business Logic Layer (Judge Core & Execution Engine)** – Compiles, executes, and evaluates submitted code in a secure environment. (Technologies: Docker, RabbitMQ, C++/Python engines)
4. **Data Layer (Database & Storage)** – Stores problems, submissions, results, and rankings. (Technologies: PostgreSQL, MongoDB, Redis)

Layered Architecture Model for an Online Judge System

sql Copy Edit



Ensuring Scalability, Maintainability & Performance:

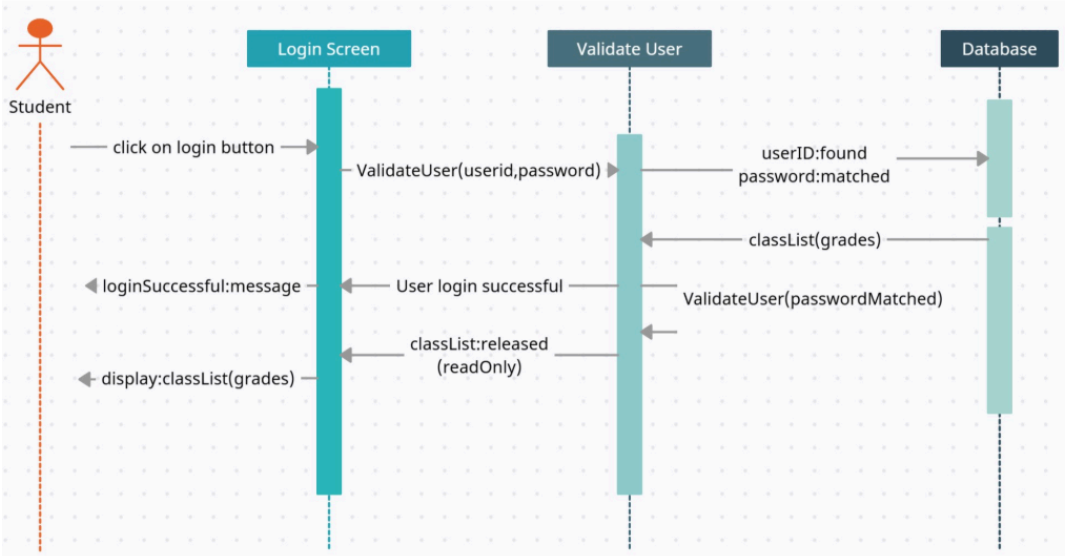
- **Scalability:** Load balancing, parallel execution, and caching.
- **Maintainability:** Modular design, microservices, and clear separation of concerns.
- **Performance:** Asynchronous processing, optimized database queries, and distributed execution.

This architecture ensures efficient and reliable online judging, even under high traffic. 🚀

15. Draw a DFD (Level-0 and Level-1) and UML Use Case Diagram for a Hospital Management System. A hospital management system is a large system that includes several subsystems or modules that provide various functions. Your UML use case diagram example should show actors and use cases for a hospital's reception.
    - **Purpose**: Describe major services (functionality) provided by a hospital's reception.
    - Consider the **Scenario** below:
    - The Hospital Reception subsystem or module supports some of the many job duties of a hospital receptionist. The receptionist schedules patient's appointments and admission to the hospital and collects information from the patients upon patient's arrival and/or by phone. The patient who will stay in the hospital ("inpatient") should have a bed allotted in a ward. Receptionists might also receive patients' payments, record them in a database and, provide receipts, file insurance claims and medical reports. **(5)**

Note: এটার উত্তরের জন্য ২ পৃষ্ঠা ফাঁকা রাখো।

16. Consider the following UML Sequence diagram and design the UML diagram for the same scenario.



- Now Draw a UML Class diagram for the scenario depicted by the Sequence Diagram. **(5)**

Ans:

To design a **UML Class Diagram** for the scenario depicted by the provided **Sequence Diagram**, we need to identify the key classes, their attributes, methods, and relationships. Here's the breakdown:

Scenario Description

1. A **Student** interacts with a **Login Screen** to log in.
2. The **Login Screen** validates the user by interacting with a **Database**.

3. If the login is successful, the **Login Screen** displays the **classList** (grades) to the **Student**.

UML Class Diagram

Classes and Their Attributes/Methods

1. **Student**
    - Attributes:
        - `studentID: String`
        - `name: String`
    - Methods:
        - `login(username: String, password: String): boolean`
2. **LoginScreen**
    - Attributes:
        - `username: String`
        - `password: String`
    - Methods:
        - `validateUser(username: String, password: String): boolean`
        - `displayClassList(grades: List<String>): void`
3. **Database**
    - Attributes:
        - `userData: Map<String, String>` (stores username and password)
        - `classListData: Map<String, List<String>>` (stores grades for each user)
    - Methods:
        - `validateUser(username: String, password: String): boolean`
        - `getClassList(username: String): List<String>`
4. **ClassList**
    - Attributes:
        - `grades: List<String>`
    - Methods:
        - `getGrades(): List<String>`

Relationships

1. **Student** interacts with **LoginScreen** to log in.
    - Relationship: **Association** (Student → LoginScreen)

2. **LoginScreen** interacts with **Database** to validate the user and fetch the class list.
    - Relationship: **Association** (LoginScreen → Database)
3. **LoginScreen** displays the **ClassList** to the **Student**.
    - Relationship: **Association** (LoginScreen → ClassList)

Note: ১ টা চিত্রের জন্য ১ পৃষ্ঠা ফাঁকা রেখের পরেরটা লেখা শুরু করো।

17. We know that Quality Assurance (QA) is not just Quality Control (QC). For example, QA is process-oriented, and QC is product-oriented. Now, suppose you are said to get a straight explanatory description of it along with the Differences and Impediments between and to QA and QC. **(5)**

Quality Assurance (QA) vs. Quality Control (QC):

Quality Assurance (QA): It is process-oriented and proactive, focusing on preventing defects by improving processes and methodologies throughout the SDLC.

Quality Control (QC): It is product-oriented and reactive, concentrating on identifying and fixing defects in the final product through testing and inspections.

Differences between QA and QC:

Aspect	Quality Assurance (QA)	Quality Control (QC)
Focus	Focuses on process improvement and preventing defects.	Focuses on identifying and fixing defects in the product.
Orientation	Process-oriented (how the product is made).	Product-oriented (what the final product is).
Approach	Proactive, aiming to improve processes for consistent quality.	Reactive, focusing on detecting and fixing defects.
Activities	Process audits, training, defining standards, and reviews.	Inspection, testing, defect identification, and validation.
Tools Used	Process models, checklists, guidelines, and methodologies.	Test cases, bug reports, test execution, and reviews.
Timing	Done throughout the SDLC, from planning to maintenance.	Primarily done during or after development and testing.

Impediments to QA and QC:

Impediments to QA:

1. **Lack of clear processes**: Without well-defined processes and standards, QA efforts can be inconsistent and ineffective.
2. **Poor communication**: If QA teams are not in sync with developers, misunderstandings can arise, affecting process improvement efforts.
3. **Lack of management support**: Without strong support from management, it can be challenging to implement and maintain effective QA processes.
4. **Resistance to change**: Team members may resist adopting new processes or improvements, slowing down the implementation of QA best practices.
5. **Inadequate training**: If the team is not adequately trained in process management, QA efforts can fall short.

Impediments to QC:

1. **Insufficient test coverage**: Inadequate test coverage may result in undetected defects, leading to poor product quality.
2. **Limited resources**: Limited time or manpower can hinder effective testing, reducing the ability to catch defects.
3. **Unclear specifications**: If the product requirements are not clear, QC teams may face challenges in identifying what to test for, leading to incomplete testing.
4. **Time pressure**: Tight deadlines may lead to rushing through QC processes, which can result in defects being overlooked.
5. **Defect complexity**: Complex defects may be harder to identify with traditional testing methods, especially when the product is large or intricate.

18. Do you think the goal of QA is just to find the bugs as early as possible? The goal of the QA is to find the bugs as early as possible and make sure they get fixed. Quality Assurance was introduced after World War II when the weapons were tested before they came into action. Explain the role of Quality Assurance (QA) at each phase of the Software Development Life Cycle (SDLC). **(5)**

Role of Quality Assurance (QA) in the Software Development Life Cycle (SDLC)

Quality Assurance (QA) is not just about finding bugs early, but also about ensuring that the software meets the required quality standards at each stage of development, optimizing the entire development process, and preventing defects rather than just detecting them. QA involves the creation of processes, procedures, and methodologies that help improve the quality of the software product.

Role of QA in SDLC:

1. **Requirement Gathering & Analysis:**
   QA ensures that requirements are clear, complete, and testable, identifying any ambiguities or inconsistencies early.

2. **Design:**
   QA reviews design documents to ensure they meet requirements, are testable, and address usability, security, and performance concerns.
3. **Implementation (Coding):**
   QA prepares test cases, performs unit tests, and ensures adherence to coding standards and specifications.
4. **Testing:**
   QA conducts various types of testing (functional, non-functional, and performance) to identify defects and works with developers to fix them.
5. **Deployment:**
   QA verifies deployment procedures, ensuring that the software works in the production environment.
6. **Maintenance:**
   QA continues regression testing and supports updates, ensuring stability and addressing any new issues.

19. Explain the Rapid Application Development (RAD) model in software engineering. Discuss its key phases, principles, and advantages. Analyze how the RAD model supports faster delivery of software solutions while maintaining quality and user satisfaction. **(5)**

Rapid Application Development (RAD) Model:

The **RAD model** is a software development methodology that emphasizes rapid prototyping, quick iterations, and active user participation to deliver software solutions quickly and efficiently.

Key Phases:

1. **Requirements Planning**: Gather and prioritize initial requirements from users and stakeholders.
2. **User Design**: Develop prototypes and refine them based on user feedback, ensuring the design meets user needs.
3. **Construction**: Build and integrate the system incrementally, using reusable components for faster development.
4. **Cutover (Transition)**: Perform final testing, user training, and deployment, transitioning the system to the user environment.

Principles:

- **Prototyping**: Users interact with working prototypes, providing feedback that drives system improvements.
- **Iterative Development**: Software is built in iterations, allowing for frequent adjustments and refinements.
- **User Involvement**: Continuous user feedback ensures the software aligns with their needs.

- **Time-boxing**: Development is done within fixed, short time frames to ensure rapid delivery.
- **Modularization**: Components are developed separately and integrated for efficiency.

Advantages:

- **Faster Delivery**: Prototyping and parallel development allow quicker delivery of functional software.
- **User Satisfaction**: Active involvement ensures the final product matches user expectations.
- **Flexibility**: Changes can be incorporated easily as user needs evolve.
- **Cost-Effectiveness**: Reduces the cost of rework and late-stage changes by gathering user feedback early.

How RAD Supports Faster Delivery and Quality:

- **Faster Delivery**: Prototypes and iterative cycles reduce development time by delivering working software quickly.
- **Quality Maintenance**: Constant user feedback and testing catch issues early, ensuring the final product is robust and meets user needs.
- **User Satisfaction**: By involving users in every phase, the software is tailored to their requirements, ensuring high satisfaction.

4o mini

20. White box testing consists of code coverage and a data coverage method. Consider the following decision (e.g. if, switch, while etc.) and make one test for each side of each decision using a table with column caption Decision, x input and y input. Then implement JUnit test class that tests each decision described in the table using Java (follow the sample in Python) **(6)**

```
int x, y;
x = c.readInt();
y = c.readInt();

if (y == 0)
    c.println("y is zero");
else if (x == 0)
    c.println("x is zero");
else {
    for (int i = 1; i <= x; i++) {
        if (i % y == 0)
            c.println(i);
    }
}

// JUnit test class that tests each decision described in the table using Python as follows, do the same in Java
```

```
import unittest

class DecisionTest(unittest.TestCase):

    def setUp(self):
        # Mocking the behavior of `c.println` using a list to collect output
        self.output = []

    def println(self, message):
        # Simulates the `c.println` function
        self.output.append(str(message))

    def process(self, x, y):
        # The translated logic of the Java code
        if y == 0:
            self.println("y is zero")
        elif x == 0:
            self.println("x is zero")
        else:
            for i in range(1, x + 1):
                if i % y == 0:
                    self.println(i)

    def test_y_is_zero(self):
        self.output.clear()
        self.process(5, 0)
        self.assertEqual(self.output, ["y is zero"])

    def test_x_is_zero(self):
        self.output.clear()
        self.process(0, 3)
        self.assertEqual(self.output, ["x is zero"])

    def test_loop_does_not_run(self):
        self.output.clear()
        self.process(0, 2)
        self.assertEqual(self.output, [])

    def test_numbers_divisible_by_y(self):
        self.output.clear()
        self.process(4, 2)
        self.assertEqual(self.output, ["2", "4"])
```

```
    def test_numbers_not_divisible_by_y(self):
        self.output.clear()
        self.process(4, 3)
        self.assertEqual(self.output, ["3"])

    def test_edge_case_y_negative(self):
        self.output.clear()
        self.process(5, -2)
        self.assertEqual(self.output, ["2", "4"])

    def test_edge_case_x_negative(self):
        self.output.clear()
        self.process(-3, 2)
        self.assertEqual(self.output, [])

if __name__ == "__main__":
    unittest.main()
```

Here's the **Java JUnit test class** that replicates the **Python unittest logic**.

📌 Test Cases Table

Decision	x input	y input	Expected Output
`if (y == 0)`	5	0	"y is zero"
`else if (x == 0)`	0	3	"x is zero"
Loop does not run	0	2	(No output)
Numbers divisible	4	2	"2", "4"
Numbers not div.	4	3	"3"
Edge case (y < 0)	5	-2	"2", "4"
Edge case (x < 0)	-3	2	(No output)

JUnit Test Class (Java):

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import java.util.ArrayList;
import java.util.List;

public class DecisionTest {

    private List<String> output;

    @Before
    public void setUp() {
        output = new ArrayList<>();
    }

    private void println(String message) {
        output.add(message);
    }

    private void process(int x, int y) {
        if (y == 0) {
            println("y is zero");
        } else if (x == 0) {
            println("x is zero");
        } else {
            for (int i = 1; i <= x; i++) {
                if (i % y == 0) {
                    println(String.valueOf(i));
                }
```

```
        }
    }
}

@Test
public void testYIsZero() {
    output.clear();
    process(5, 0);
    assertEquals(List.of("y is zero"), output);
}

@Test
public void testXIsZero() {
    output.clear();
    process(0, 3);
    assertEquals(List.of("x is zero"), output);
}

@Test
public void testLoopDoesNotRun() {
    output.clear();
    process(0, 2);
    assertEquals(List.of(), output);
}

@Test
public void testNumbersDivisibleByY() {
    output.clear();
    process(4, 2);
    assertEquals(List.of("2", "4"), output);
}

@Test
public void testNumbersNotDivisibleByY() {
    output.clear();
    process(4, 3);
    assertEquals(List.of("3"), output);
}

@Test
public void testEdgeCaseYNegative() {
    output.clear();
    process(5, -2);
    assertEquals(List.of("2", "4"), output);
}

@Test
public void testEdgeCaseXNegative() {
    output.clear();
```

```
    process(-3, 2);
    assertEquals(List.of(), output);
  }
}
```

21. Black Box Unit testing is earlier and more precise than Black Box System testing - it can find errors very early, even before the entire first version is finished. Now, Consider the production codes that need function testing. Suppose you have JUnit 4 API in your IDE and you are said to develop test codes for these production codes showing the application of Exception, Setup Function and Timeout Rule. How do you solve it? **(5)**

To implement exception handling, setup, and timeout rules in JUnit 4:

1. **Setup Function**:
    - Use @Before to set up resources before each test, or @BeforeClass for setup before any tests.

Example:

```
@Before
public void setup() { ... }
```

2. **Exception Handling**:
    - Use @Test(expected = Exception.class) to expect an exception or use the ExpectedException rule for more control.

Example:

```
@Test(expected = IllegalArgumentException.class)
public void testException() { ... }
```

3. **Timeout Rule**:
    - Use the Timeout rule to set a time limit for a test. If the test exceeds the limit, it fails.

Example:

```
@Rule
public Timeout globalTimeout = Timeout.seconds(1);
```

These features help ensure your tests are properly set up, handle exceptions, and enforce time limits.

Full Example:

```
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;
import org.junit.rules.Timeout;

public class MyTest {

    @BeforeClass
    public static void setupBeforeClass() {
        System.out.println("Setting up resources before all tests...");
    }

    @Before
    public void setup() {
        System.out.println("Setting up resources before each test...");
    }

    @Rule
    public ExpectedException exceptionRule = ExpectedException.none();

    @Rule
    public Timeout globalTimeout = Timeout.seconds(1);

    @Test
    public void testException() {
        exceptionRule.expect(IllegalArgumentException.class);
        exceptionRule.expectMessage("Test exception");
        throw new IllegalArgumentException("Test exception");
    }

    @Test
    public void testTimeout() throws InterruptedException {
        Thread.sleep(1500);  // This will fail due to timeout
    }
}
```

Explanation:

- **Exception Handling**: In the `testException()` method, we expect an `IllegalArgumentException` to be thrown, and we also check that the message is correct using the `ExpectedException` rule.
- **Setup**: The `setupBeforeClass()` runs once before any tests, while the `setup()` runs before each individual test method.
- **Timeout**: The `testTimeout()` method will fail because it exceeds the specified timeout of 1 second (simulated by `Thread.sleep(1500)`).

By using these techniques, you can write robust JUnit test cases that handle exceptions, set up necessary resources, and enforce time constraints efficiently.