**Q1. Draw a Use Case Diagram - 4 Marks**



**Q2. Derive Boundary Classes, Controller classes, Entity Classes. - 4 Marks**

**1. Boundary Classes (UI/Interaction Layer):**

These classes handle interactions between the user and the system. They are typically forms, screens, or input pages.

| **Boundary Class** | **Description** |
| --- | --- |
| PaymentPage | Interface to allow customer to choose and initiate payment |
| CardPaymentForm | UI to enter card details (Card No, Expiry, CVV) |
| WalletPaymentScreen | UI to log into wallet (like Paytm, PhonePe) |
| NetBankingLoginScreen | UI to select bank and enter credentials |

**2. Controller Classes (Process/Logic Layer):**

These classes handle the logic and flow of control for the use case.

| **Controller Class** | **Description** |
| --- | --- |
| PaymentController | Handles overall payment coordination |
| CardPaymentController | Validates and processes card payments |
| WalletPaymentController | Manages wallet API calls and validation |
| CashPaymentController | Handles confirmation of cash payment |
| NetBankingController | Handles net banking integration and logic |

**3. Entity Classes (Data/Business Layer):**

These classes represent the data and business rules.

| **Entity Class** | **Description** |
| --- | --- |
| Customer | Contains customer details |
| Payment | General payment information (amount, date, status) |
| Card | Card details (card number, expiry, etc.) |
| Wallet | Wallet details (wallet ID, balance) |
| BankAccount | Bank info for net banking |

**Q3. Place these classes on a three tier Architecture. - 4 Marks**

In a **Three-Tier Architecture**, classes are organized into the following layers:

1. **Presentation Layer (UI Layer)** – User interaction
2. **Business Logic Layer (BLL)** – Application logic & rules
3. **Data Access Layer (DAL)** – Data management and storage

**1. Presentation Layer (Boundary Classes)**

These classes interact with the user interface:

| **Class Name** | **Role** |
| --- | --- |
| PaymentPage | Allows customer to select payment mode |
| CardPaymentForm | UI to collect card details |
| WalletPaymentScreen | UI for wallet login |
| NetBankingLoginScreen | UI for net banking login |

**2. Business Logic Layer (Controller Classes)**

These classes process input, control application logic, and manage workflows:

| **Class Name** | **Role** |
| --- | --- |
| PaymentController | Main logic for processing payments |
| CardPaymentController | Handles card payment flow |
| WalletPaymentController | Handles wallet transactions |
| CashPaymentController | Logic for handling cash option |
| NetBankingController | Logic for net banking transactions |

**3. Data Access Layer (Entity Classes)**

These classes represent business objects and manage database interactions:

| **Class Name** | **Role** |
| --- | --- |
| Customer | Stores customer details |
| Payment | Contains payment info like amount, status |
| Card | Holds card-related data |
| Wallet | Stores wallet data |
| BankAccount | Represents bank account for net banking |

**Q4. Explain Domain Model for Customer making payment through Net Banking - 4 Marks**

A **Domain Model** is a visual representation of real-world objects (entities), their attributes, and relationships within the problem domain. It helps identify core business concepts and their interactions.

****

**Q5. Draw a sequence diagram for payment done by Customer Net Banking - 4 Marks**

****

**Q6. Explain Conceptual Model for this Case - 4 Mark**

A conceptual model is a high representation of a system that helps in understanding, visualizing and communicating the essential aspects of a domain.

It provides a clear and simplified view of the domain, making, it easier to understand key elements of a conceptual model.

Entities: Customer, Product, Order and Payment

Attributes: Customer id, name, email, phone number.

Relationships: For example, a customer place an order.

**Q7. What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture - 8 Marks**

**MVC Architecture Explanation:**

**MVC** stands for **Model-View-Controller**.

It is a **design pattern** that separates an application into three interconnected components:

**Model**: Manages data, business logic, and rules of the application.

**View**: Handles the display and presentation to the user.

**Controller**: Manages user inputs and updates the Model and View accordingly.

**Purpose**: To separate internal representations of information from the ways information is presented and accepted from the user.

**2. MVC Rules to Derive Classes from Use Case Diagram:**

When analyzing a **Use Case Diagram**, we can apply these rules:

| **MVC Layer** | **Rule to Derive Classes** |
| --- | --- |
| **Model** | Identify business objects (nouns) from the use cases and descriptions. Example: Customer, Order, Product. |
| **View** | Identify screens, forms, reports where users interact with the system. Example: Order Entry Screen, Invoice Report. |
| **Controller** | Identify actions (verbs) that involve interaction between user and system. Example: Place Order, Update Customer Profile. |

**Q8. Explain BA contributions in project (Waterfall Model – all Stages) – 8 Marks**

| **Stage** | Activities | Artifacts & Resources |
| --- | --- | --- |
| **Pre Project** | - Market research - Feasibility study - Identifying stakeholders | Business Case, Feasibility Report, Stakeholder List |
| **Planning** | - Scope definition - Estimating effort and timelines - Risk analysis | Scope Document |
| **Project Initiation** | - Conducting stakeholder meetings - Understanding business needs | Project Initiation Document (PID), Stakeholder Analysis Matrix |
| **Requirements Gathering** | - Conducting interviews/workshops - Documenting requirements | BRD (Business Requirements Document), Meeting Minutes, Use Cases |
| **Requirements Analysis** | - Validating and prioritizing requirements - Gap analysis | SRS (Software Requirements Specification), Requirements Traceability Matrix (RTM) |
| **Design** | - Assisting in translating requirements into system design - Reviewing design documents | Functional Specification Document, Review Notes |
| **Development** | - Clarifying requirements for developers - Participating in sprint/demo discussions | Updated RTM, Developer Queries Log |
| **Testing** | - Supporting test case creation - Reviewing test scenarios - Validating fixes | Test Plan, Test Cases, Defect Log, RTM |
| **UAT (User Acceptance Testing)** | - Coordinating UAT activities - Collecting feedback - Assisting in final sign-off | UAT Plan, UAT Feedback Report, Sign-Off Document |

**Q9. What is conflict management? Explain using Thomas – Kilmann technique – 6 Marks**

**What is Conflict Management?**

Conflict Management refers to the process of identifying, addressing, and resolving disagreements or conflicts between individuals or teams in a constructive manner. Effective conflict management helps maintain a healthy work environment, improves collaboration, and ensures project progress is not disrupted.

Thomas–Kilmann Conflict Management Technique:

The Thomas–Kilmann Conflict Mode Instrument (TKI) is a widely used model that identifies five conflict-handling styles based on two dimensions:

* Assertiveness – the extent to which a person tries to satisfy their own concerns.
* Cooperativeness – the extent to which a person tries to satisfy others’ concerns.

Five Conflict Handling Styles:

| Style | Description | When to Use |
| --- | --- | --- |
| 1. Competing | High assertiveness, low cooperativeness. Focus on winning. | In emergencies or when quick, firm decisions are needed. |
| 2. Collaborating | High assertiveness, high cooperativeness. Win-win solution. | When both sides' concerns are important and time allows. |
| 3. Compromising | Moderate assertiveness and cooperativeness. Middle-ground solution. | When a temporary solution is acceptable. |
| 4. Avoiding | Low assertiveness and cooperativeness. Ignoring or postponing conflict. | When the issue is trivial or emotions are high. |
| 5. Accommodating | Low assertiveness, high cooperativeness. Yielding to others. | When preserving relationships is more important than the issue. |

**Q10. List down the reasons for project failure – 6 Marks**

Projects can fail for various reasons, especially when there is a lack of planning, communication, or alignment with business goals. Below are some key reasons why projects fail:

1. Unclear Requirements

* Poorly defined or incomplete requirements lead to confusion during development.
* Misunderstanding between what the client wants and what is delivered.

2. Lack of Stakeholder Involvement

* Stakeholders are not actively involved in reviews or decision-making.
* Delays in feedback and approvals impact project timelines.

3. Poor Planning and Estimation

* Inaccurate time and cost estimates lead to missed deadlines and budget overruns.
* Lack of risk analysis and mitigation strategies.

4. Ineffective Communication

* Miscommunication between team members, departments, or stakeholders.
* Important information gets lost or misunderstood.

5. Scope Creep

* Uncontrolled changes or continuous addition of new features.
* Results in resource strain and reduced focus on core objectives.

6. Inadequate Testing

* Insufficient or rushed testing causes defects in the final product.
* Leads to poor user experience and project rejection.

**Q11. List the Challenges faced in projects for BA – 6 Marks**

A Business Analyst (BA) plays a key role in bridging the gap between business needs and technical solutions. However, BAs often face several challenges during project execution. Below are some common ones:

1. Unclear or Changing Requirements

* Stakeholders may not clearly define what they need.
* Requirements often change mid-project, causing rework and confusion.

2. Communication Gaps

* Miscommunication between stakeholders, developers, and testers.
* Lack of clarity in explaining business needs to technical teams.

3. Conflicting Stakeholder Interests

* Different stakeholders may have opposing views or priorities.
* BA has to balance and negotiate to meet the overall business goal.

4. Scope Creep

* Gradual increase in project scope due to additional feature requests.
* Leads to delays, budget overruns, and quality issues.

5. Limited Access to Stakeholders

* Difficulty in scheduling meetings or feedback sessions.
* Delays in decision-making and requirement finalization.

6. Technical Constraints

* Business requirements may not always align with system capabilities.
* BA must work with developers to find feasible alternatives.

**Q12. Write about Document Naming Standards – 4 Marks**

A document numbering standard is systematic approach to assigning unique identifies to various documents created and used throughout the development process.

Ex- Suppose we have project with ID”PROJ123” and we working with a Requirements Specification Document.

Project ID: PROJ123

Document Type: REQ

Version: 1.0

Date: 2025-04-19

The document identifier could be PROJ123-REQ- 1.0.2025.05-26.

**Q13. What are the Do’s and Don’ts of a Business analyst – 6 Marks**

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Do's** | **Don’ts** |
| 1 | Consult the SME clairficaiton in requirements. | Never say NO to client. |
| 2 | Go to the client with a plain mind with no assumptions, Listent carefully and completely until the client is done, and then you can ask queries. | There is no word as "By default". |
| 3 | Try to extract maxumum leads to solution from the client himself. | Never imaging anything in terms of GUI. |
| 4 | Consentrate on the important requirements. | Don't interupt the client when he is giving you the problem. |
| 5 | Question the existence of existence/Question everhthing | Never try to give solution to the client straight away with your previous experience and assumptions. |

**Q14. Write the difference between packages and sub-systems – 4 Marks**

| **Aspect** | Packages |  | Sub-Systems |
| --- | --- | --- | --- |
| **Definition** | A package is a logical grouping of related elements such as classes, interfaces, or use cases. |  | A sub-system is a larger unit representing a complete, independently functioning part of the system. |
| **Scope** | Used for code or model organization within a system. |  | Represents a functional component of a larger system. |
| **Dependency** | Packages often depend on other packages and are not standalone. |  | Sub-systems can work independently or be integrated with other subsystems. |
| **UML Representation** | Represented using a folder icon in UML diagrams. |  | Represented as a component or a separate block in system architecture diagrams. |

**Q15. What is camel-casing and explain where it will be used- 6 Marks**

Camel-casing is a naming convention used in programming and documentation where each word in a phrase is capitalized except the first one, and no spaces or underscores are used between words. Example:

* customerName
* orderTotal
* getUserData

The name comes from the way the capital letters in the middle of the word resemble the humps of a camel

Where Is Camel-Casing Used?

Programming Languages:

Used in Java, C#, JavaScript, Python, etc., for naming variables, methods, and classes.

Example: calculateTotalAmount(), orderDate

APIs and Data Fields:

Used in JSON keys and API parameters.

**Q16. Illustrate Development server and what are the accesses does business analyst has? -6 Marks**

**Development Server:**

A Development Server is an environment where software applications are built, developed, and initially tested before being deployed to staging or production environments.

It is mainly used by developers and testers to:

* Write and test new code
* Fix bugs
* Integrate new features
* Perform unit testing and integration testing

Key Features of a Development Server:

* Contains in-progress code (not final)
* Frequently updated with new builds
* May have test data or dummy data
* Not accessible to end-users or clients
* Separate from production and staging servers

**Accesses for a Business Analyst (BA):**

A Business Analyst generally has limited but important access to the development server:

| Access Type | Description |
| --- | --- |
| Read-Only Access to Builds | BA can view the application builds to verify if features align with requirements. |
| Access to Test Environment | BA may test use cases or assist QA with validating business rules. |
| Access to Logs or Reports | BA can analyze logs or reports for issue tracking or requirement validation. |

**Q17. What is Data Mapping 6 Marks**

Data Mapping is the process of connecting data fields from one source to corresponding fields in another system or database. It defines how data from a source (such as an API, form, or database) is translated, transformed, or matched to the destination format or structure.

**Why is Data Mapping Important?**

Ensures data consistency and accuracy between systems.

Essential for data integration, migration, transformation, and reporting.

Helps in handling differences in data formats, units, field names, or structures.

**Q18. What is API. Explain how you would use API integration in the case of your application Date format is dd-mm-yyyy and it is accepting some data from Other Application from US whose Date Format is mm-dd-yyyy 10 Marks**

**What is an API?**

API (Application Programming Interface) is a set of protocols and tools that allows different software applications to communicate with each other. It acts as a bridge between two systems, enabling them to share data and functionality securely and efficiently.

For example, when two different systems (like our application and a US-based application) need to exchange data (such as user information, product details, or dates), an API ensures this communication is smooth and standardized.

**Implementation Steps:**

**Receive API Data:**

Use an HTTP client in your backend (e.g., in Python, Node.js, Java, etc.) to receive data via REST API.

**Parse the Date Field:**

Extract the date string from the received data.

Use date parsing libraries (e.g., moment.js, datetime, or DateTime.ParseExact) to interpret mm-dd-yyyy format.

**Convert the Format:**

Convert the date into the required format (dd-mm-yyyy) before saving it to your database or displaying it to users.