Objected Oriented Analysis and Design with Java - I

UNIT 1 - Diagrammatic representations

▼ **Introduction to OO programming**

- Emphasis is on data rather than programs. Programs are divided into objects and can communicate with each other through functions.
- Follows bottom-up approach

Requirement engineering - A process of working proactively with all stakeholders gathering their needs, articulating their problem, identifying and negotiating potential conflicts thereby establishing a clear scope and boundary for a project.

1. Understand requirements in depth from product and process perspective
2. Classify into functional/non functional requirements and system requirements
3. Model requirements
4. Analyse requirements using fishbone diagram
5. Recognise and resolve conflicts
6. Negotiate requirements
7. Prioritize requirements suing MOSCOW
8. Identity risks
9. Decide on Build or Buy

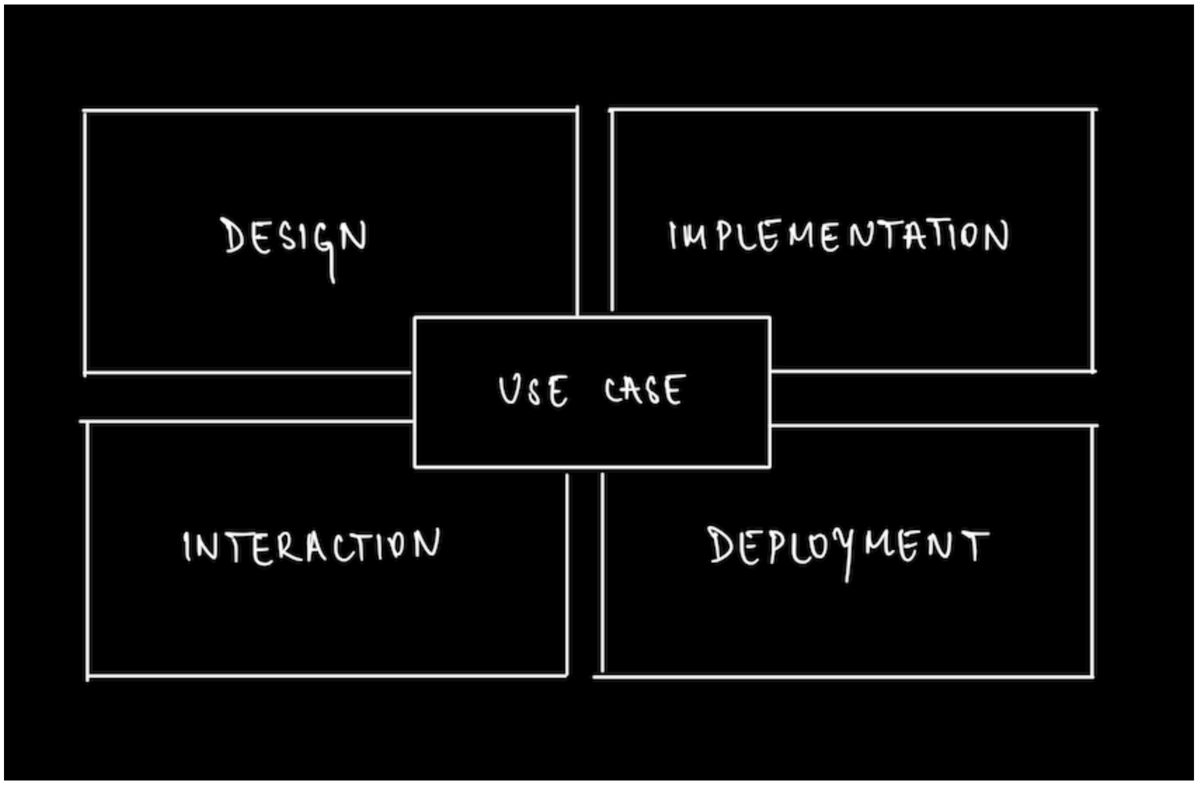
▼ **Unified Modelling Language (UML)**

UML is a language to express different types of models that defines the syntax and semantics. UML is used to visualise, specify, construct and document the artifacts of software intensive systems.

UML is made up of three conceptual elements -

1. **Building block**s that make up the UML
2. **Rules (name, scope and visibility**) that dictate how the blocks can be put together
3. **Common mechanisms** that apply throughout the UML

Views of a Solution system -



- Use case view. - Describes behaviour of a system as seen by end users, analysts and testers.

- Design view - Describes classes, interfaces and collaborations that form the vocabulary of the problem and its solution.
- Interaction view - Describes flow of control among its various parts. Addresses scalability, performance and throughput.
- Implementation view - Addresses configuration management of system releases.
- Deployment view - Addresses the distribution, delivery and installation of the system

▼ **Use Case Diagrams, Class Diagrams and CRC cards**

Describes

- interaction of users and system
- functionality provided by the system to it's users

Use case model has two important elements - actors and use cases

Relationships between use cases -

1. **Generalization** - Use cases that are specialized versions of other use cases
2. **Include** - Use cases that are included as parts of other use cases. Enabled to factor common behaviour
3. **Extend** - Use cases that extent the behaviour of other core use cases. Enable ot factor variants.

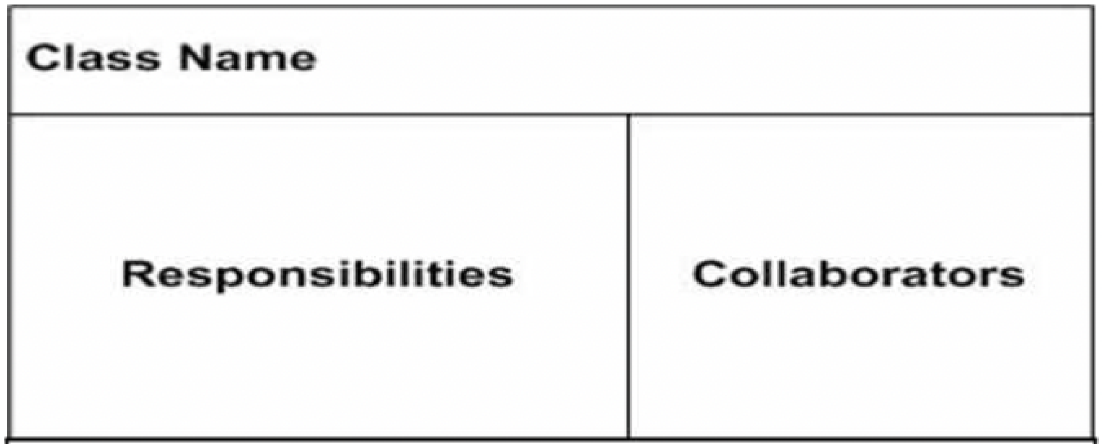
▼ Template for Use Case Specification:

- Name:
- Summary:
- Actor:
- Preconditions:
- Description:
- Exceptions:
- Alternate flows:

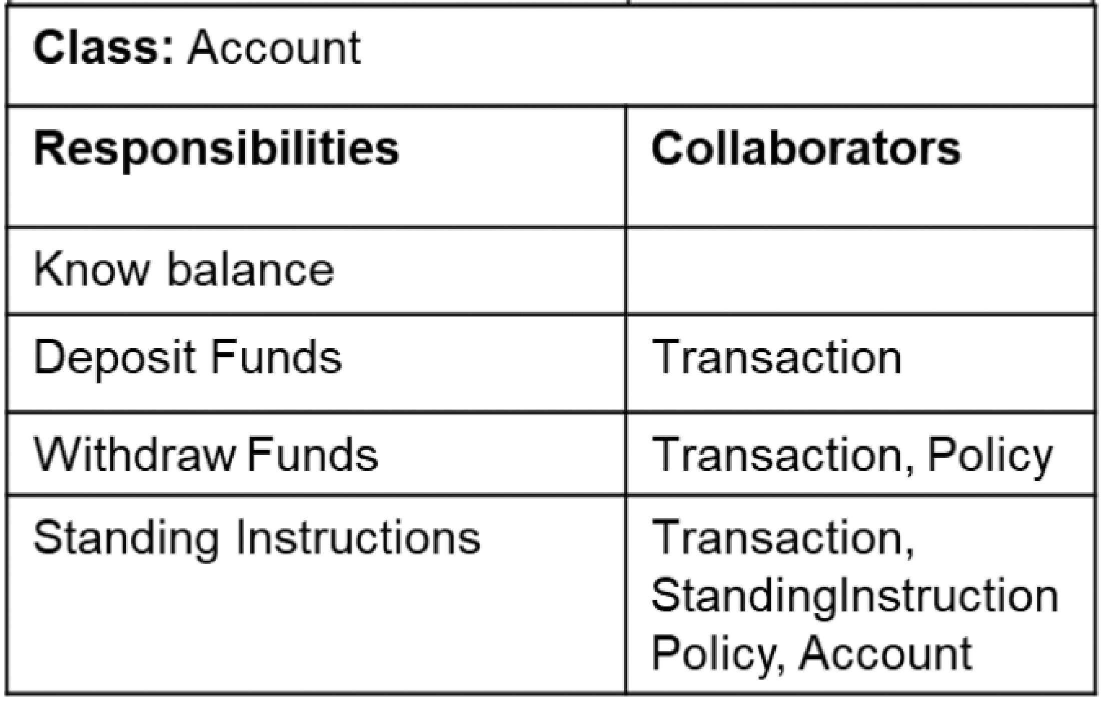
- Post conditions:

▼ **CRC cards**

CRC - Class Responsibility Collaboration



Class Name	
Responsibilities	Collaborators



Class: Account	
Responsibilities	**Collaborators**
Know balance	
Deposit Funds	Transaction
Withdraw Funds	Transaction, Policy
Standing Instructions	Transaction, StandingInstruction Policy, Account

▼ **OO Relationships**

▼ Generalization

"is a" or "is kind of a" relationship. Showcases reusable elements in the class diagram.

- ▼ Abstraction

  An abstract method is a method that does not have implementation.

- ▼ Realization

  A specialized abstraction relationship between two things where an interface specifies a contract and a class carries out the implementation.

- ▼ Dependency

  One class uses another. A dependency relationship is often used to show that a method has object of a class as arguments.

- ▼ Association

  If two classes in a model need to communicate with each other, there must be an association between them. Unidirectional and bidirectional.

  The multiplicity of an association is indicated by adding multiplicity adornments to the line denoting the association.

  The behaviour os an object in an association can also be indicated using role names.

- ▼ Aggregation

  "has a" relationship. Consists of an aggregate and a constituent part.

- ▼ Composition

  Strong ownership and coincident lifetime of parts by the whole.

- ▼ Enumeration

  User defined datatype that consists of a name and an ordered list of enumeration fields.

- ▼ Exception

  Key exceptions and SQL exceptions fall under this

▼ **Component model**

- Used for modelling large systems into smaller subsystems which can be easily managed.
- A component is a replaceable and executable piece of system whose implementation details are hidden.
- It is a modular part of the system that encapsulates its contents.

Units of a component diagram -

1. Component - Each component is responsible for one clear aim, similar to a black box whose external behaviour is defined by a provided interface and required interfaces
2. Interface - Required and Provided interfaces. Separated the specification of functionality from its implementation by a class diagram or a subsystem
3. Usage connector - component base that acts a decomposition unit for larger systems.
4. Port - an interaction point between a classifier and an external environment.

A component diagram is used to represent the structure and organization of components during any instance of time.

1. Represents the components of a system at runtime
2. Helps testing of a system
3. Visualizes the connection between various components

▼ **Deployment model**

Maps the software architecture created in design to the physical system architecture that executes it. Models runtime architecture of the system.

Deployment diagrams are used to visualize

1. The topology of physical components of a system and where the software components are deployed.
2. Describe the hardware components used to deploy the software components.

3. Describe runtime processing nodes.

Elements of a deployment diagram -

- Artifacts - concrete elements caused by a development process
- Components with artifacts - concrete real world entities related to sw development
- Associations

▼ **Interaction modelling - Activity Model**

Elements of an activity model -

1. Start point
2. Action state or Activity
3. Action flow
4. Object flow
5. Decisions and branching
6. Guards
7. Synchronization
8. Time event
9. Merge event
10. Sent and Received signals
11. Swimlanes

▼ **Behaviour modelling - State Machine Models**

State diagram shows the actual changes in state, not the processes or commands that created those things

Notations and symbols -

1. State
2. Transition

3. Synchronization and splitting of control

Advanced State diagrams-

- Nested state diagrams
- Concurrent state diagrams

▼ Unit 1 pdf for diagrammatic examples

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/bb8fb149-71bc-4370-a21f-733f6312630f/OOAD_U1.pdf