BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 5

DCA3103

SOFTWARE ENGINEERING

Unit 5

Object-Oriented Design

Table of Contents

1. INTRODUCTION

An organising and modelling strategy for software systems based on the idea of objects is known as object-oriented design or OOD. It offers a methodical approach to software development and places a focus on modular architecture, code reuse, and maintainability.

The design should be specific to the problem and general enough to address future problems and requirements. The goal of object-oriented programming is to make development easier, quicker, and more natural by raising the level of abstraction to the point where applications can be implemented in the same terms in which they are described the application domain. It must be possible to find pertinent objects, factor them into classes at the right granularity, and establish key relationships among them. Objects and classes, relationships, object-oriented design process, object identification and design dels such as sequence models and state diagrams are covered in this unit.

1.1 Learning Objectives:

After studying this unit, students should be able to:

- *Describe objects and object classes*
- *Explain the object-oriented design process*
- *Discuss various design models*

2. OBJECT-ORIENTED DESIGN

2.1 Object and Object Classes

Objects: An object is a real-world entity, comprising data and methods to manipulate the data. We can have many objects nested inside one object. We are aware of the meaning of an object, but we have to describe the exact use of an object in software development. Every real-time entity can be called an object, such as a person, a place or a thing. It can be a noun or noun phrase, either physical or conceptual. Some examples of objects of the college are shown in Table 5.1:

Table 5.1

Object Name	Description of the Object
Kushal	An employee of the college. A physical entity
Sruthi	Head of the computer department and also an employee of the college.
Vindya	An employee of the computer department and also an employee of the college.

The objects Kushal, sruthi and vindya have similarity. They can be categorized into a group (class) such as 'Employee'. An object contains data and functions as its integral parts. In object terminology, the data integral to an object are called attributes of the object and functions are called methods. This is shown in Figure 5.1. In the example stated above, the objects can have different attributes such as emp_id, emp_name, designation, address, basic pay etc. These attributes distinguish one object from another. The methods that operate on the data (attribute) of the object are also integral parts of the object. An object has a unique identity and refers to a single entity from a group of entities.

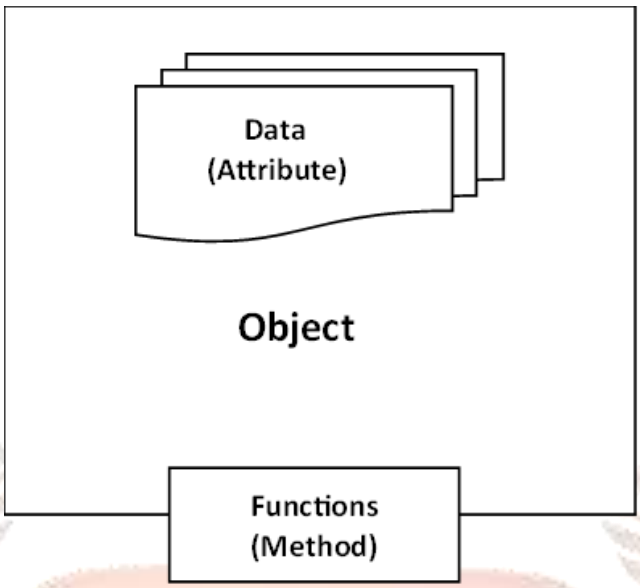

Fig 5.1: Structure of an Object

Object Classes

Now that we have seen the way to define an object using attributes, let us study the way to group these objects. Objects are grouped into a class. The class definition is given below.

Definition: An object class is defined as a group of objects with the same structure and behavior. We usually identify the properties and behavior of a real-world entity very easily. For example, we can easily identify a computer or a typewriter, based on the properties and behavior of a computer and a typewriter. Also, we can group all the properties of a particular object, say a computer, and call it a class.

All objects belong to some particular class and every object is identified as an instance (occurrence) of that class. A class helps us to easily classify objects and inherit properties from other classes. Classes define the properties of an object. For example, an object XYZ or let us say an instance XYZ of the class 'computer' has the property 'price'. This property can have different values such as Rs.36000, Rs.25000 etc.

We can call the class as a template for an object. Every object of a class has the same format and responds to the same instructions. For example, a class named 'Student' will have different objects of students (A, B, C) under it () which is shown in Figure 5.2.

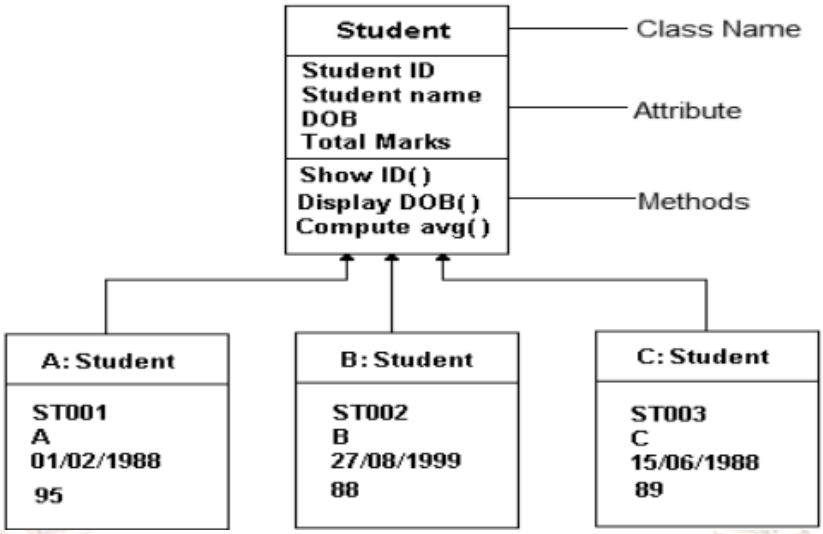

Fig 5.2: Example of a Class

Here, the student represents a group of students. All these students are instances of the class and are individually identified as objects. The Student class can have an unlimited number of such instances. The data associated with every object is managed by the object itself, but the instructions are managed by the class. If two students A and B of the class Student, are instructed to display their Date of birth using the 'Display DOB()' method, both students respond equally to the instruction, but with their unique DOBs. In object-oriented programming, we use predefined classes for which we create different instances and solve the problem.

2.2 Relationship

A relationship provides a channel through which instances of classes communicate with each other. They represent connections among instances of classes. Relationships are classified as follows:

- A Kind-Of relationship.
- Is-A relationship.
- Has-A relationship / Part-Of-relationship.

Consider for a moment the similarities and differences among the following objects/classes: Automobile, Ford, Truck, Car and Engine. We can make the following observations:

- A truck is a kind of an automobile.

- A car is a (different) kind of automobile.
- An engine is part of an automobile.
- An automobile has an engine.
- The Ford is a car.

A-Kind-Of Relationship:

Taking the example of a human being and an elephant, both are 'kind-of' relationships. As human beings and elephants are 'kind-of' mammals (living things), they share the attributes and behaviours of mammals. Human beings and elephants are subsets of the mammals class. The following Figure 5.3 depicts the relationship between the Mammals and Human Being classes:

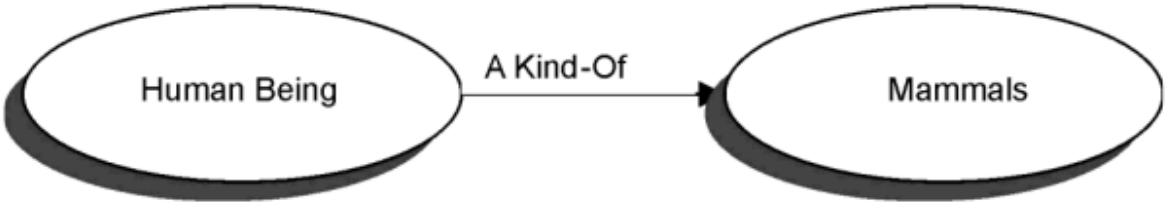

Fig 5.3: Example of A-Kind-Of relationship.

Is-A Relationship

Let's take an instance of the human being class – Peter, who 'is –a human being and, therefore, a mammal. The following Figure 5.4 depicts the 'is –a relationship.

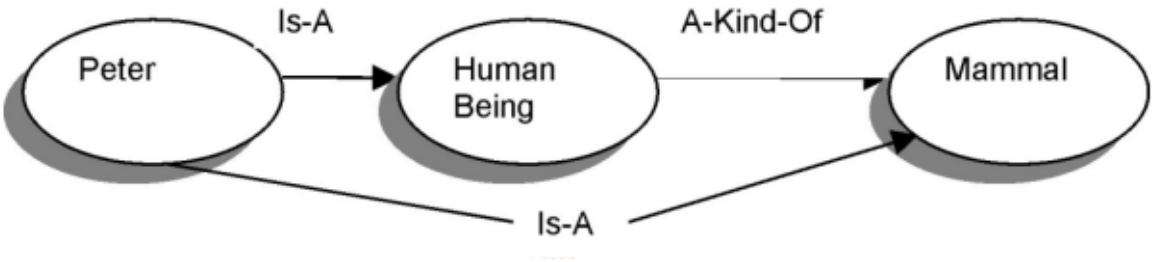

Fig 5.4: Example of Is-A relationship.

Has-A Relationship/Part-Of Relationship

A human being has a heart. This represents a Has-a relationship. The heart is a part of the human being. This represents part of the relationship. The following Figure 5.5 depicts the relationship between a human being and a heart.

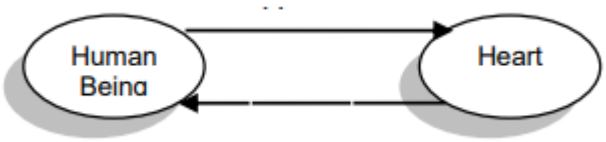

Fig 5.5: Example of Has-A/Part-Of Relationship

3. SERVICE USAGE

The term "service usage" in object-oriented design (OOD) describes how objects communicate with one another by calling methods or accessing attributes to carry out particular activities. Utilising services is necessary to give a software system the desired functionality and behaviour.

The following are some crucial elements of service consumption in OOD:

- Collaboration between objects: In an OOD system, objects communicate with one another by sending messages or calling methods. Through this cooperation, objects can cooperate to carry out their duties and accomplish system-level objectives.
- Method Invocation: To request actions or get information, objects employ the methods supplied by other objects by other objects through the use of services. An object can make use of the features and behaviours exposed by another object by calling its methods.
- Encapsulation and Interfaces: Encapsulation and specified interfaces make it easier to use services. By exposing only, the essential methods and attributes through an object's interface, encapsulation conceals the internal implementation details of an object. Interfaces establish the methods that can be used and the properties that can be accessible, defining the contract between objects.
- Dependency Injection: Managing dependencies between objects is done using the dependency injection approach. Objects can use services offered by other objects without being strongly tied to their particular implementations by injecting

dependencies through interfaces. This encourages the design's adaptability, modularity, and testability.

- Design patterns: Reusable solutions to typical design issues are provided by design patterns. Many design patterns use services to provide communication and coordination between objects, such as the Observer pattern and the Strategy pattern.
- IoC, or inversion of control, is a design approach that encourages loose coupling and concern separation. In IoC, a container or framework instead of the object that is being used has control over object instantiation and management. Thus, objects can rely on the external services offered by the container while concentrating on their primary duties.
- Service-Oriented Architecture (SOA): A key idea in SOA, which promotes the creation of systems based on loosely connected services. Services in SOA offer well-defined interfaces that other components or services can utilise to perform particular business tasks.

4. OBJECT-ORIENTED DESIGN PROCESS

In the object-oriented design process, we must revisit the classes identified during the analysis phase to implement them in the design. We must add more classes and attributes if required. Let us see the step-by-step process of object-oriented design.

4.1 Activities in the Object-Oriented Design Process

The activities in the object-oriented design process using a unified approach are shown in Figure 5.6.

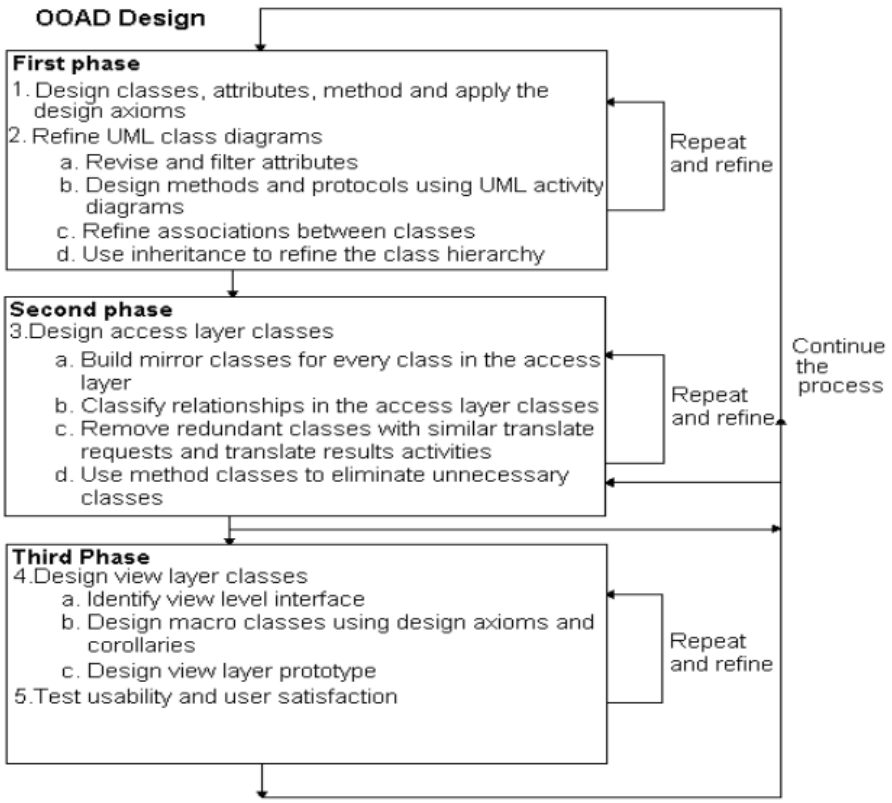

Fig 5.6: The Object-Oriented Design Using Unified Approach

In the OOAD design approach as shown in Figure 5.6, we have three phases. The first phase deals with the designing of classes and applying the design axioms to them. Here we have to create the UML diagrams and define class associations and hierarchy.

In the second phase, we have to create the mirror classes for the access layer corresponding to every class created in the first phase. We have to eliminate the redundant classes and use method classes to refine the class structure.

The third phase deals with designing view layer classes. Here we have to use the view level interface and create macro classes according to the axioms and corollaries. Here, we test the usability and user satisfaction also.

Finally, repeat the process until an effective design is formed. We can iterate each phase individually, a different number of times to refine it.

Using the UML class diagrams, we can create an inheritance structure. We can find out the classes to be created and the classes to be reused. We can create specific subclasses and

superclasses if required. In object-oriented systems, iteration is considered a good practice. We have to follow a step-by-step approach while designing and the design should match the user requirements.

4.2 Object Identification

The main problem in object-oriented design is identifying the objects that make up the system, their attributes and associated operations (methods). There is no simple formula, which allows objects to be identified. Designers must use their skills and experience in this task.

Several plans are made to identify objects, some of them are discussed below:

- Use grammatical analysis of a natural language description of a system. Objects and attributes are nouns, and operations or services are verbs.
- Use tangible entities (things) in the application domain such as aircraft, roles such as manager, events such as requestssssss, interactions such as meetings, locations such as offices, organizational units such as companies, and so on.
- Use a behavioural approach in which the designer first understands the overall behaviour of the system. The various behaviors are assigned to different parts of the system and an understanding is derived of who initiates and participates in these behaviours. Participants who play significant roles are recognized as objects.
- Use a scenario-based analysis where various scenarios of system use are recognized and analyzed in turn. As the individual scenario is analyzed, the team responsible for the analysis must recognize the required objects, attributes and operations. A method of analysis called Class- Responsibility-Collaboration (CRC) cards whereby analysts and designers take on the role of objects is effective in supporting this scenario-based approach. These approaches are not limited. Good designers may use all of them when trying to identify objects.

4.3 Object Interface Design

A key component of Object-Oriented Design (OOD) in software engineering is Object Interface Design. It involves developing the interfaces for objects, which specify how they communicate with one another and how other objects can access their services.

The following are some important considerations for Object Interface Design:

- Cohesion: Interfaces should have high cohesion, which means that their methods should be closely related and concentrated on a particular task. This encourages a distinct and well-defined interface purpose and aids in ensuring that objects follow the Single Responsibility Principle (SRP).
- Abstraction: The underlying implementation details should be abstracted using interfaces. They should hide unnecessary complexity and show a simplified, high-level picture of the services that an object provides. Abstraction makes it possible to handle and interact with objects based on their fundamental qualities rather than on the unique implementations they have.
- Consistency: Throughout the system, interfaces should adhere to the same naming conventions and design patterns. This enhances the interfaces' readability, maintainability, and clarity for developers working with them.
- Completeness: Interfaces should be created so that they offer all the essential methods and properties to carry out the functionality that is intended. They must faithfully reflect the agreement between the object and its customers, guaranteeing that they may rely on the specified services.
- Dependency Management: Interfaces are essential for managing dependencies between objects. Objects can be independent and dependent on abstractions by relying on interfaces as opposed to actual implementations. This encourages adaptability, modularity, and simplicity in testing and maintenance.
- Versioning and compatibility: When designing interfaces, versioning and compatibility should be taken into consideration. Future system changes and evolution should be carefully considered, ensuring that existing interfaces can be expanded or modified without compromising compatibility with current clients.
- Documentation: The intent, behaviour, and usage of each method and property should be clearly described in the interface documentation. Documentation reduces confusion and increases efficiency by assisting developers in comprehending how to interact with things through their interfaces.
- The Interface Segregation Principle (ISP): ISP states that interfaces have to be specialised and targeted towards particular client needs. It calls for several interfaces

that are suited for various clients as opposed to a single monolithic interface. This reduces needless coupling by preventing clients from depending on methods they do not use.

4.4 Structural Decomposition

In Object-Oriented Design (OOD), the process of breaking a software system into smaller, independent modules or components is known as structural decomposition. It involves figuring out the main structural components of the system and arranging them in a layered or hierarchical fashion. Several important factors for structural disintegration in OOD are listed below:

- *Identify the primary Components:* To start, determine which primary elements or modules that form the system. These components stand in for the system's higher-order functional units or roles. It's crucial to describe these elements in terms of their distinct and clear roles.
- *Define Relationships:* Identify the connections and interdependencies among the components that have been discovered. Recognise how the parts work together, share information, and interact to produce the intended system behaviour. This involves identifying relationships, dependencies, and linkages between the components.
- *Encapsulation and Information Hiding:* Make sure that each component encapsulates its data and behaviour by applying the concept of encapsulation to it. This encourages information hiding, which is the technique of hiding implementation details of a component from other components so that they can only communicate through well-defined interfaces.
- *Layered Architecture:* Consider creating a layered architecture by dividing the components into layers or tiers according to their functions and degree of abstraction. A layered design enables modular creation, testing, and maintenance while separating different issues. Data access, domain/business logic, presentation, and application logic are typical layers.
- *Use Design Patterns:* Design patterns can be used to direct the structural decomposition process. Design patterns offer tested answers to frequent design issues and can facilitate the effective organisation of the components and relationships between them.

For instance, to differentiate the presentation layer from the business logic and data access layers, the Model-View-Controller (MVC) paradigm might be utilised.

5. DESIGN MODELS

Design models show the objects or object classes in a system and, the relationships between the entities. Design models essentially are the bridge between the system requirements and the system implementation. However, they also have to include enough details for programmers to make implementation decisions.

An important step in the design process is to choose which design model is essential and the level of detail of these models. This depends on the type of system that is being developed. There are two types of design models to describe an object-oriented design:

1. Static models describe the static nature of the system using object classes and their relationships.
2. Dynamic models describe the dynamic structure of the system and show the interactions between the system objects (not the object classes)

Unified Modeling Language (UML) provides twelve different static and dynamic models that may be produced to document a design. Some of them are seen below:

1. Subsystem models that show logical groupings of objects into coherent sub-systems. These are denoted using a form of the class diagram where each sub-system is shown as a package. Subsystem models are static models.
2. Sequence models that show the sequence of object interactions. These are denoted using a UML sequence diagram. Sequence models are dynamic models.
3. State machine models that show how individual objects change their state in response to events. These are represented in the UML using state chart diagrams. State machine models are dynamic models.

Figure 5.7 shows the objects in the sub-systems in the weather station. For example, the commscontroller object is linked with the weatherstation object, and the weatherstation object is linked with the data collection package. This means that this object is linked with one or more objects in this package.

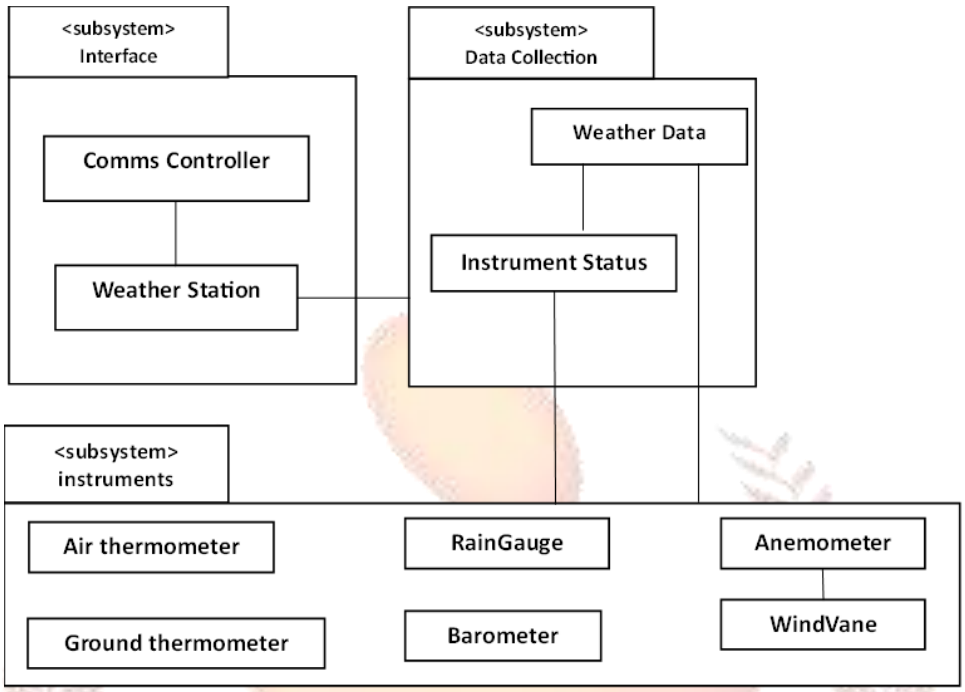

Fig 5.7: Weather Station Packages

5.1 Sequence Model

Sequence models are dynamic models that document each mode of interaction in which a sequence of object interactions takes place. Figure 5.8 is an example of a sequence model that shows the operations involved in collecting the data from a weather station. In a sequence model:

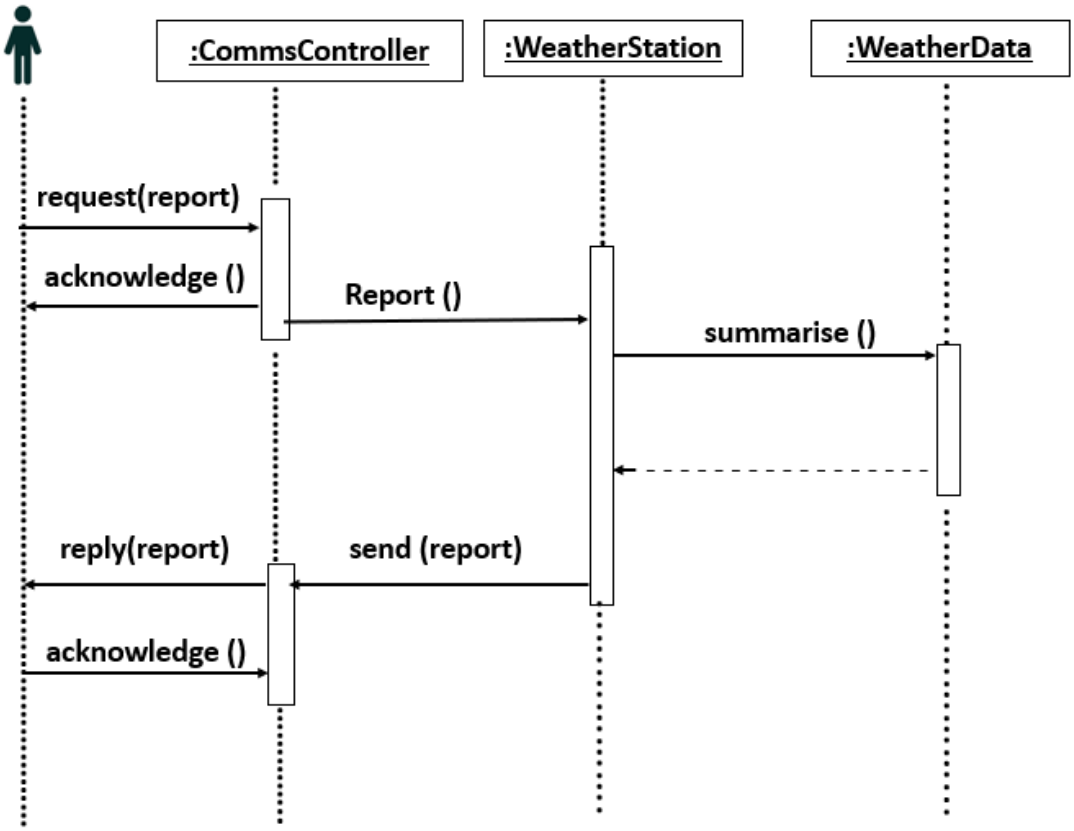

Fig 5.8: Sequence of Operations – Data Collection

1. The objects involved in the interaction are arranged horizontally with a vertical line linked to each object.
2. Time is denoted vertically so that time progresses down the dashed vertical lines. Therefore, the sequence of operations can be read easily from the model.
3. Labelled arrows linking the vertical lines represent interactions between objects. These are not data flows but denote messages or events that are fundamental to the interaction.
4. The thin rectangle on the object lifeline denotes the time when the object is controlling the object in the system. An object takes over control at the top of this rectangle and relinquishes control to another object at the bottom of the rectangle. If the hierarchy of calls, control is not relinquished until the last return then the initial method call has been completed.

Figure 5.8 shows the sequence of interactions when the external mapping system requests the data from the weather station. Read the sequence diagram from top to bottom:

1. An object that is an instance of commscontroller (:commscontroller) receives a request from its environment to send a weather report. It acknowledges receipt of this request. The half-arrowhead on the acknowledge message indicates that the message sender does not expect a reply.
2. This object sends a message to an object that is an instance of a weatherstation to create a weather report. The instance of the commscontroller then suspends itself (its control box ends). The style of arrowhead used indicates that the commscontroller object instance and the weatherstation object instance are objects that may execute concurrently.
3. That object that is an instance of weatherstation sends a message to a weather data object to summarise the weather data. in this case, the squared-off style of the arrowhead indicates that the instance of the weather station waits for a reply.
4. This summary is computed and the control returns to the weather station object. The dotted arrow indicates a return of control.
5. This object sends a message to the commscontroller requesting it to transfer the data to the remote system. The weather station object then suspends itself.
6. The commscontroller object sends the summarized data to the remote system, receives an acknowledgement, and then suspends itself waiting for the next request.

From the sequence diagram, we can see that the commscontroller objects and the weatherstation objects are concurrent processes, where execution can be suspended and resumed. Essentially, the commscontroller object instance listens for messages from the external system, decodes these messages and initiates weather station operations.

5.2 State Machine Model

Sequence diagrams are used to model the combined behaviour of a group of objects, but you may also want to summarize the behaviour of a single object in response to the messages it can process. To do this, you can use a state machine model that shows how the object instance changes state depending on the messages that it receives.

Figure 5.9 is a state chart for the weatherstation object that shows how it responds to requests for various services. Read Figure 5.9 as follows:

1. If the object state is shutdown then it can only respond to a startup() message. It then moves into a state where it is waiting for further messages. The unlabeled arrow with the black blob denotes that the shutdown state is the initial state.
2. In the waiting state, the system expects further messages. If a shutdown() message is received, the object returns to the shutdown state.
3. If a report weather () message is received, the system moves to the summarizing state. When the summary is complete, the system moves to s transmitting state where the information is transmitted through the commscontroller. It then returns to the waiting state.
4. If a calibrate() message is received, the system moves to the Calibrating state, then the Testing state, and then the Transmitting state, before returning to the Waiting state. If a test() message is received, the system moves directly to the Testing state.
5. If a signal from the clock is received, the system moves to the Collecting state, where it is collecting data from the instruments. Each instrument is instructed in turn to collect its data.

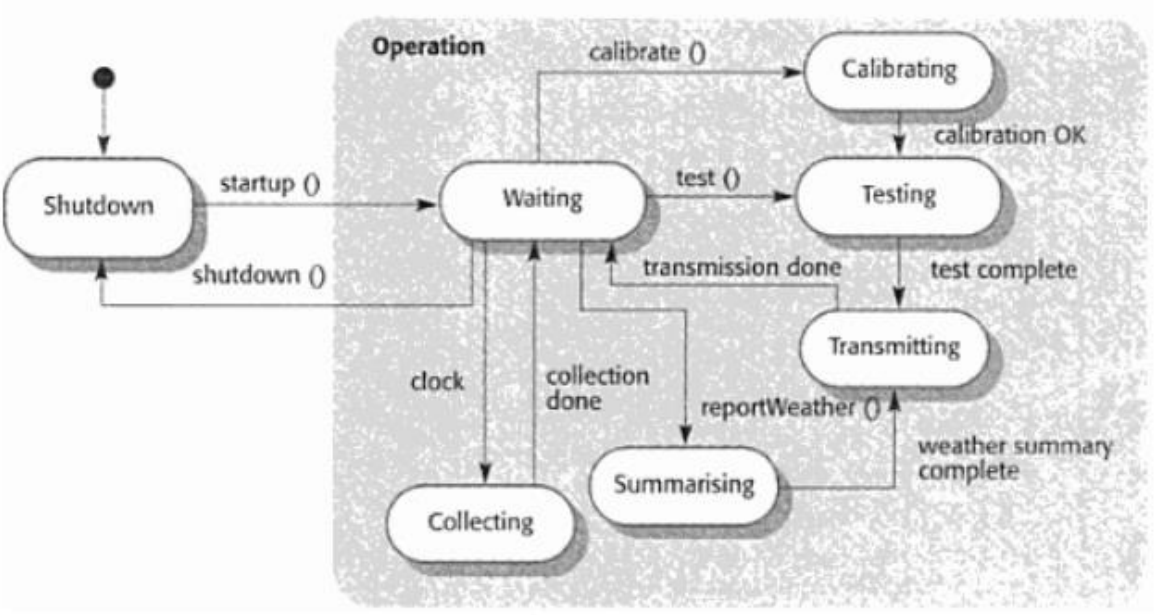

Fig 5.9: State Diagram for Weatherstation

6. SUMMARY

- An object is a real-world entity, comprising data and methods to manipulate the data. Every real-time entity can be called an object, such as a person, a place or a thing.
- An object contains data and functions as its integral parts. In object terminology, the data integral to an object are called attributes of the object and functions are called methods.
- An object class is defined as a group of objects with the same structure and behaviour.
- A relationship provides a channel through which instances of classes communicate with each other. They represent connections among instances of classes. Relationships are classified as follows: A Kind-Of relationship, Is-A relationship, Has-A relationship / Part-Of-relationship.
- In the object-oriented design process, we have to revisit the classes identified during the analysis phase to implement them in the design.
- Several plans are made to identify objects, some of them: Use grammatical analysis, Use tangible entities, Use a behavioural approach, Use scenario-based analysis and many more.
- Design models show the objects or object classes in a system and, the relationships between the entities. There are two types of design models to describe an object-oriented design: static models and dynamic models.
- Static models describe the static nature of the system using object classes and their relationships.
- Dynamic models describe the dynamic structure of the system and show the interactions between the system objects(not the object classes)
- Unified Modeling Language (UML) provides twelve different static and dynamic models to document a design
- Sequence models that show the sequence of object interactions. These are denoted using a UML sequence diagram. Sequence models are dynamic models.
- State machine models that show how individual objects change their state in response to events. These are represented in the UML using state chart diagrams. State machine models are dynamic models.

7. SELF–ASSESSMENT QUESTIONS

SELF-ASSESSMENT QUESTIONS – 1

1. An _____ is a real-world entity, comprising data and methods to manipulate the data.
2. An object contains _____ and _____as its integral parts.
3. In the object terminology, the data integral to an object are called _____ of the object and _____ are called methods.
4. An _____ is defined as a group of objects with the same structure and behaviour.
5. A _____ provides a channel through which instances of classes communicate with each other
6. List various types of relationships
7. In the object-oriented design process, we have to revisit the classes identified during the _____ phase to implement them in the design.
8. Mention any two methods to identify objects.
9. CRC stands for _____.
10. List various types of design models.
11. _____describes the static nature of the system using object classes and their relationships.
12. Dynamic models describe the dynamic structure of the system and show the interactions between the system objects (not the object5 classes) (True/False).
13. UML stands for _____.
14. In UML, _____ model shows the sequence of object interactions.
15. In UML, _____ models that show how individual objects change their state in response to events.

SELF-ASSESSMENT QUESTIONS – 1

16. _____ is the technique of hiding the implementation details of a component from other components.
17. _____ that interfaces have to be specialised and targeted towards particular client needs.
18. _____ identifies the connections and interdependencies among the components that have been discovered.

8. SELF-ASSESSMENT ANSWERS

1. Object
2. data and functions
3. attributes, functions
4. object class
5. relationship
6. A Kind-Of relationship. Is-A relationship. Part-Of-relationship/ Has-A relationship.
7. Analysis
8. Use tangible entities, Use a behavioural approach
9. Class-Responsibility-Collaboration
10. Static model and dynamic model
11. Static models
12. True
13. Unified Modeling Language
14. Sequence
15. State machine
16. Encapsulation
17. The Interface Segregation Principle (ISP) states
18. Relationship

9. TERMINAL QUESTIONS

1. Explain object and object class with an example.
2. List and explain various types of relationships with an example.
3. With the help of a diagram, describe the three phases of an object-oriented design process.
4. Write a short note on object identification.
5. What is the sequence model in unified modelling language? Explain with an example.
6. What is the state machine model in a unified modelling language? Explain with an example.

10. TERMINAL ANSWERS

1. An object is a real-world entity, comprising data and methods to manipulate the data. An object class is defined as a group of objects with the same structure and behaviour. (Refer to section 5.2)
2. A relationship provides a channel through which instances of classes communicate with each other. (Refer to section 5.3)
3. In the OOAD design approach, there are three phases. The first phase deals with the designing of classes and applying the design axioms to them. Here we must create the UML diagrams and define class associations and hierarchy. (Refer to section 5.4)
4. The main problem in object-oriented design is identifying the objects that make up the system, their attributes and associated operations (methods). (Refer to section 5.4.2)
5. Sequence models that show the sequence of object interactions. These are denoted using a UML sequence diagram. Sequence models are dynamic models. (Refer to section 5.5)
6. State machine models that show how individual objects change their state in response to events. These are represented in the UML using state chart diagrams. State machine models are dynamic models. (Refer to section 5.5)