UNIT 15 INTRODUCTION TO UML

Structure

15.0 INTRODUCTION

The UML diagrams are used for modeling the object oriented design. UML stands for Unified Modeling Language. UML provides a standards-base, general purpose modeling language that helps everyone to understand various aspects of the system. As UML is language independent, we can use UML for many different languages such as Java, Python, C sharp and others. UML provides the software blueprint in the visual representation.

15.1 OBJECTIVES

After going through this unit, you should be able to

- understand key concepts of UML,
- understand the object oriented design,
- understand the difference between structural UML diagrams and behavioral UML diagrams,
- learn to model the class diagram, sequence diagram, component diagram and deployment diagram
- understand state chart diagram, use case diagram, activity diagram

15.2 BACKGROUND OF OBJECT ORIENTED DESIGN

Function-oriented programming emphasize on algorithms. A function or procedure is the basic building block in function-oriented programming. This is mostly a top-down approach. The function-oriented programming suits well for relatively smaller and

less complex systems. Due to inherent coupling and behavioral relationship, usually there will be high complexity.

In the object-oriented programming, the data is exposed only through the interfaces. The object-oriented programming follows a bottom up approach. The object normally represents a logical unit or a real-world object. The object state contains all the object properties/attributes and the behavior of an object is expressed through its functions or methods.

The problems that are inherently complex are well suited for object oriented programming. The object oriented programming is designed for change, allows reuse, improves productivity, speeds up development time and improves maintainability. We can build scalable and modular applications with object oriented programming.

15.3 KEY CONCEPTS OF OBJECT ORIENTED DESIGN

In object oriented programming, the solution is composed of objects that are instances of classes interacting with each other via relationships. Objects represent real world entities. Objects encapsulate data and functions/behavior that control the data. While objects are the basic building blocks of object oriented design, classes are the blueprints of the objects.
The object oriented programming focuses on decomposition, encapsulation, abstraction, modularity and inheritance.

The abstraction hides the inner details of the entity and exposes the behavior through interface. It defines the clear boundaries of the object and provides the required characteristics and behavior of an object. Abstraction can be implemented through a well-defined interface. Abstraction plays a key role in the object modelling providing the integrations only with the exposed interfaces and hides the user from the inner details of the component.

The modularity partitions the individual components based on their responsibilities so that the modules can be easily reused and extended. The system is decomposed into loosely coupled cohesive modules.

In the **encapsulation based design**, the data is exposed only through the functions and the class provides methods for binding data. Encapsulation eliminates the direct dependencies between objects and hence avoids tight coupling. Encapsulation is also called data hiding or information hiding.

In Inheritance the child class obtains the characteristics and behavior of its parent class. It is possible to model the hierarchy through inheritance. You can also change the behavior of the subclass through overloading and overriding.

In **composition** one object are composed of other objects.

UML diagrams are effective way to visually model the object oriented design as the UML diagrams can depict inheritance, composition, modularity and abstraction.

15.4 INTRODUCTION TO UML DIAGRAMS

UML model was released in 1997 as a common design language building and modelling software applications. UML model provides the ways to depict the complex the interaction between components, the behavior, and sequence, static and dynamic nature of the software components. UML models are independent of the programming language.

UML diagrams visually model and depict the system from various perspectives such as use case, implementation, process and deployment. We can visually depict, model, document and build the system using UML diagrams.

Definition

A Unified Modeling Language is a structured language that provides the commonly agreed vocabulary to communicate, model and document the structural and behavioral aspects of the software system. UML is widely used as a blueprint to design, develop, document, analyze, and comprehend the software systems.

Importance of UML diagrams

When modeling the architecture of the overall software system, it is important to understand and analyze the system from various perspectives. A software architect has to understand the implementation aspects of the application, overall structure of the application, the system topology, flow of the messages and such. A model abstracts the system that helps in easier understanding. UML diagrams is a popular way to model the system. UML is also supported by popular Integrated Development Environments (IDE) such as Eclipse. Given below are the key advantages of the UML diagrams:

- UML diagrams model the big picture of the entire application and helps us to analyze the application from various dimensions.
- UML diagrams provide a common language and vocabulary to understand, model, document and discuss the system design.
- UML diagrams helps us to understand the conformance of the system to business and technical requirements.
- The team gets the clear understanding of the main classes, components, interfaces and their relationship through UML diagrams. This helps the developers to develop the low-level design faster and implement the system quicker.
- During impact analysis or during major bug fixes or during system enhancements, we can use the UML diagrams to analyze the key impact areas to come up with optimal design.
- We can implement the key architecture best practices such as loose coupling, layered architecture, and separation of concerns by modeling the system in UML.
- UML diagrams make the system easy to develop, maintain and change.
- We can document the system from various perspectives through the UML diagrams. UML diagrams also document the key architecture decisions.
- UML diagrams help us visualize the overall structure and behavior of the system and its various components.

Classification of UML Diagrams

We can broadly classify UML diagrams into two categories - static diagrams or structure diagrams and dynamic diagrams or behavior diagrams.
Static or structural diagrams represent the static physical elements of the system. Static diagram includes class diagrams, object diagrams, component diagrams and deployment diagrams. The dynamic or behavior diagrams depict the runtime and dynamic behavior of the software systems. The dynamic diagrams include sequence diagram, collaboration diagram, state diagram and activity diagram. We have depicted the main categories of UML diagrams in Figure 15.1.

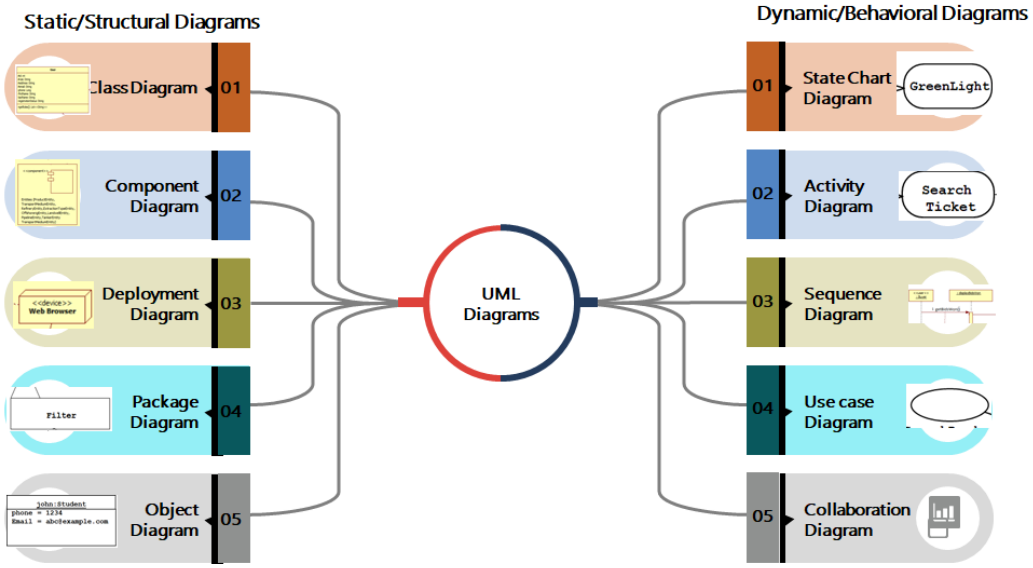

Figure 15.1 : Categories of UML diagrams

Dynamic or behavioral UML Diagrams

Dynamic diagrams or behavioral diagrams depict the dynamic and moving parts of the system. The main behavioral elements depicted in the dynamic UML diagram are object state, control flow, interaction, state machines and such. The main dynamic UML diagrams are sequence diagram, activity diagram, state chart diagram, use case and collaboration diagram.

The **sequence diagram** models the behavior of a single use case or a single scenario. The sequence diagram visually depicts how each objects interact with each other. In the sequence diagram we have a time ordered messages where vertical dimension represents the time axis and horizontal dimension represents the roles.

The **activity diagram** describes the business logic the business process and work flows. The activity diagram models the use case.

The **state diagrams** describe the systems behavior through state transitions. The state diagram depicts all possible states of an object when an event occurs.

Static or structural UML Diagrams

Static or structural UML diagrams depict the overall structure of the system that remains static. The main structural elements depicted in the UML diagram are classes, objects, nodes, interfaces, use cases, collaboration and components. The main static UML diagrams are package diagram, class diagram, component diagram, deployment diagram and object diagram.

A **class diagram** depicts the classes along with its attributes and operations. The **object diagram** depicts the instance of class encompassing the state.

The **package diagram** organizes the elements of system into logical groups to minimize the dependencies among them.

The **component diagram** represents the physical modules, known as components of the solution. The components are logical, independent autonomous and encapsulated units of a system and exposes one or more

interfaces. The component diagram depicts the relationship between various components of the system.

The **deployment diagram** defects the relationship between the software and hardware components of the system. The nodes in the deployment diagram represent servers or other computational elements. Deployment diagram depicts how the components are organized in the system

Views Covered by UML Diagrams

We can comprehend a complex system only by looking at various aspects (also known as views or perspectives) of the system.

The UML diagrams cover various views. We have depicted various views covered by the UML diagrams in Figure 15.2.

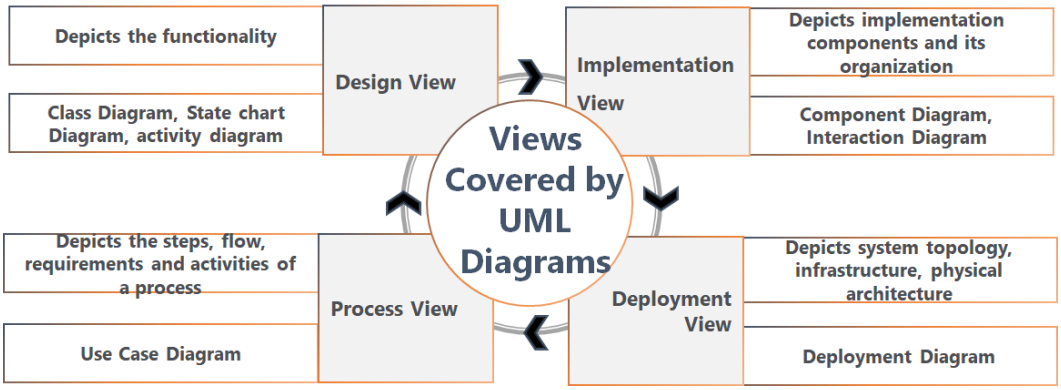

Figure 15.2 : Views covered by UML diagrams

As shown in figure 15.2, it is important to model the system from various views such as implementation, design, deployment and process. Each view provides the specific abstraction of the system. We can understand the system holistically by analyzing all the views.

The design view essentially captures the functionality of the system. The static part of the design is depicted through class diagram and the dynamic behaviour is captured in the activity diagram and state chart diagram.

The implementation view depicts the overall implementation components such as components (static aspects) and interactions (dynamic aspects) of the application. The implementation view specifies the physical view of the system.

In process view, the UML diagrams primarily depict the sequence of steps and flow and components involves in a use case. The Use case diagram models the requirement of the system. We can understand the throughput and scalability of the system through the use process view.

The deployment view covers the physical architecture of the system. We can understand the overall distribution and topology of the application components. We can assess the capacity and sizing of the overall solution in the deployment view.

Extensibility of UML Diagrams

We can use the below given features to extend the UML diagram:

- **Stereotypes** allow us to extend the UML vocabulary by adding the application specific elements. For example we can add the logging framework details for the application using stereotype
- **Constraints** allows us to add the business rules and conditions. For example we could specify the value of "date" attribute to be greater than 01-Jan-1970
- **Tagging** allows us to specify the additional metadata for the object. For instance we can specify the details such as author name, version using tagging.

15.5 STRUCTURAL UML DIAGRAMS

Structural elements depicted in the UML diagrams are the static parts of the system. Given below are the key structural elements depicted in the UML diagrams; normally they are "Nouns" of the system.

- **Class**: It is logical unit that encapsulate the state (through attributes), and behavior (through operations) of the objects and can optionally implement one or interfaces. A class can interact with other classes through relationships such as generalization, composition and such. A class is depicted in rectangle as shown in the Figure 15.3.

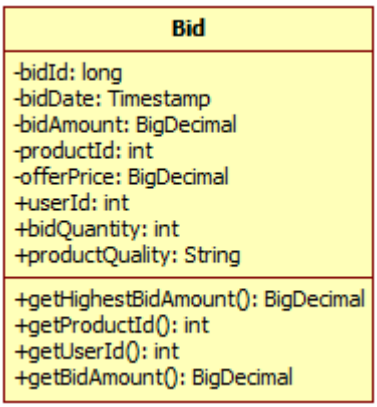

Figure 15.3 : Class

- **Interface**: An interface specifies the contract for the implementation through the operations. The implementation class has to implement the operations of the interface. We have depicted the interface in Figure 15.4.

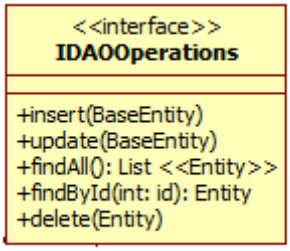

Figure 15.1 : Interface

- **Object**: An object is the instance of the class. Objects interact with each other by passing messages and expose the data through operations. We have depicted the Object in the figure 15.5.

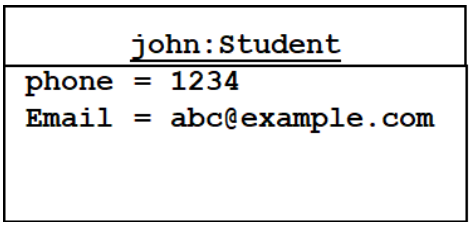

Figure 15.5 : Object

- **Component**: A component represents a logical module that exposes interfaces for interaction. The component is represented in Figure 15.6

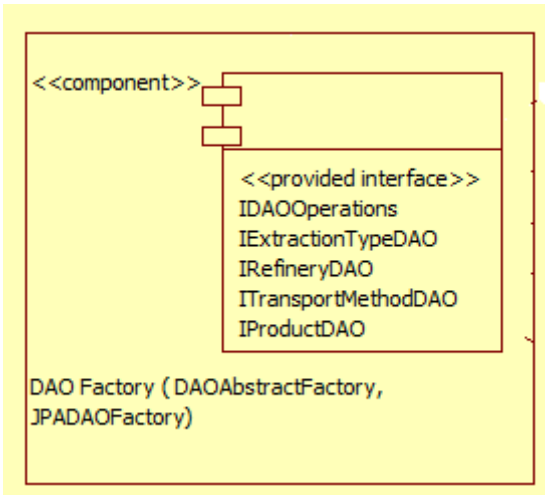

Figure 15.6 : Component

- **Use case:** A use case represents a specific functionality and represents the flow for implementing the use case. A use case is represented in Figure 15.7.

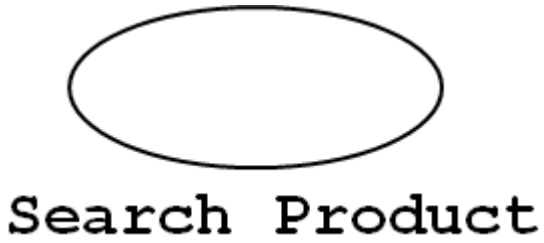

Figure 15.2 : Use case

15.5.1 Class diagram

The class diagram visually provides the object oriented design of the system and depicts the relationship between implementation classes. The class diagram visually provides the inheritance, composition, association relationship within classes.
In the class diagram we depict the class name, attributes, visibility (public, protected, private), associations and operations and optionally comment.

The relationships in the class diagram can be of the following types:

- **Association** relationship depicts the set of related links among the objects. For example a single "department" objects can be associated with multiple "employee" objects. Composition is type of association wherein a single object contains multiple dependent objects that cannot independently exist leading to whole-part kind of relationship. In aggregation relationship, a single object consist of multiple independent objects that can exist without the parent object. The association between "Department" class and "Employee" class is depicted in Figure 15.8. The association relationship indicates that each department has many employees. Aggregation is a weak form of association whereas composition is a strong form of association.

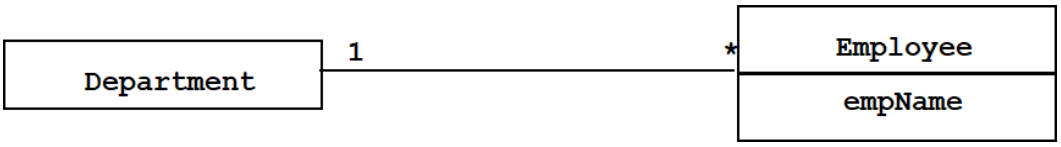

Figure 15.3 : Association Relationship

- **Dependency** relationship depicts the dependency between two elements (dependent and independent elements). When the independent element changes, the dependent element also changes.
- **Generalization** depicts the inheritance (parent-child relationship) in the class diagram. The child inherits the operations and attributes of the parent. The subclass is a specific kind of the super class.
- **Realization** depicts the contract between two entities. Normally we depict the realization relationship between interface (that specifies the contract) and the implementation class (that implements the contract).

The class diagram depicted in Figure 15.9 provides the details of three classes: User, Seller and Buyer.

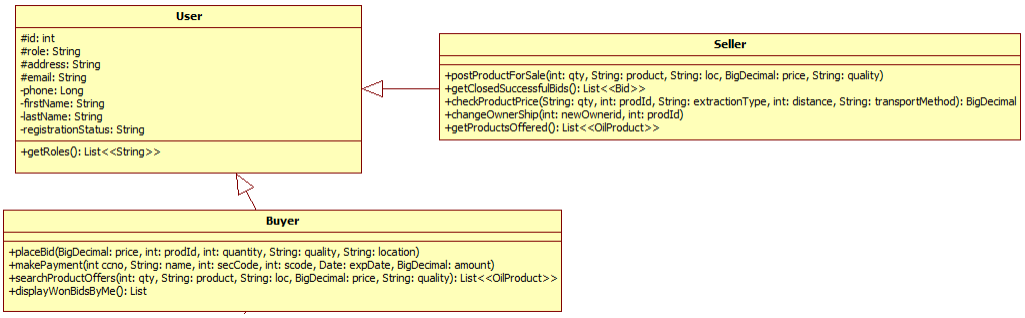

Figure 15.9 : Class diagram with Inheritance

The User class is the parent class that has attributes such as “id” and “role” are protected whereas the attributes such as “firstName”, “lastName” are private. The User class has a public method “getRoles ()” the return the list of String objects. The Figure 15.9 also depicts the generalization relation wherein the “Buyer” and “seller” classes are sub/child classes of “User” class.

The interface and its implementation is depicted in Figure 15.10.

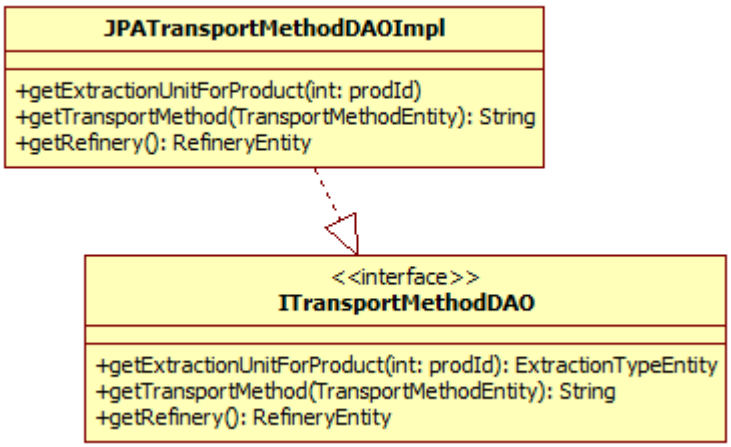

Figure 15.4 : Interface and its Implementation

The interface ITransportMethodDAO specifies three operations as part of contract and the implementation class JPATransportMethodDAOImpl provides the complete implementation of the three operations.

15.5.2 Object diagram

The object diagrams depict the instantiated objects and their state. The object diagram also represent the relationship between objects. We have depicted an object diagram in Figure 15.11.

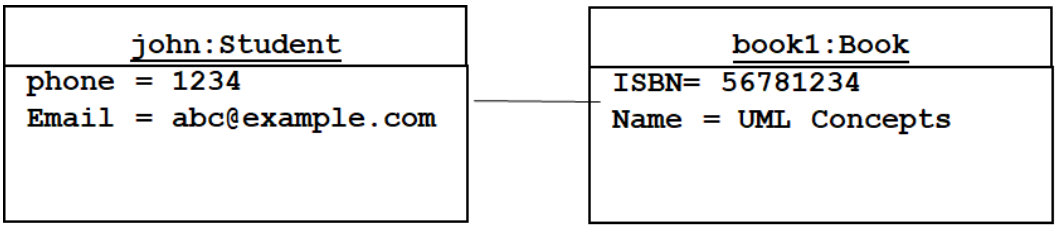

Figure 15.5 : Object Diagram

15.5.3 Component diagram

A component represent logically related entities such as classes and their dependents that interact with other components through well-defined interfaces.
The component diagram provides the physical view of the application wherein we model the software components and their dependencies on other components. The component diagram depicts the organization of physical entities of the application. We can depict the high order solution components or packages in the component diagram.

The component is depicted in Figure 15.12.

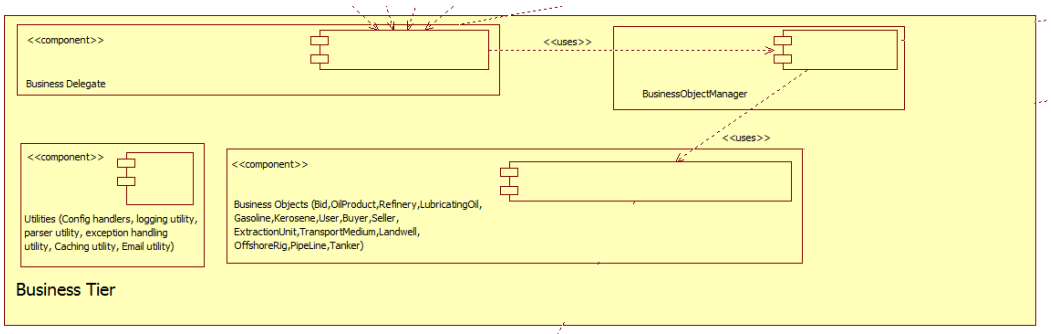

Figure 15.12 : Component Diagram

We have also depicted the interfaces exposed by the component and its interaction with other component in Figure 15.13.

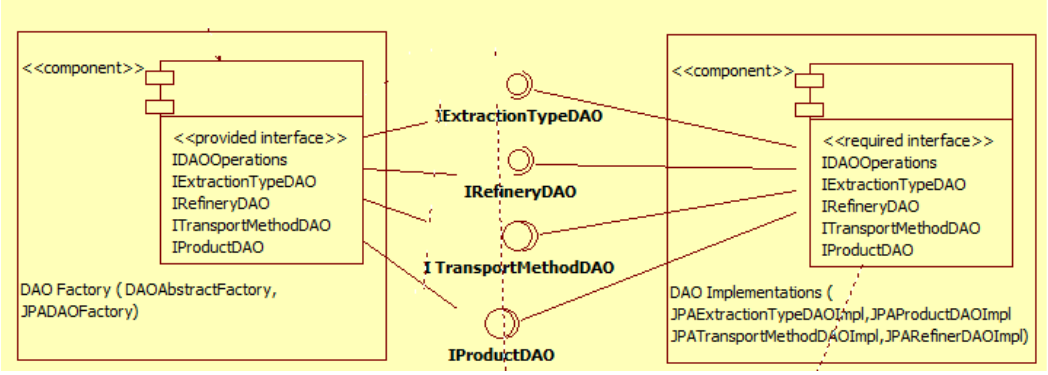

Figure 15.13 : Component Interfaces

15.5.4 Deployment diagram

We depict the details of solution components deployed on the infrastructure in the deployment diagram. Deployment diagram provide details of the hardware components (such as servers, load balancers, firewalls, virtual machines, routers, storage) and the software artefacts (such as files, Java Archives, libraries, executable etc.) that are deployed on them. Nodes (infrastructure elements) and their dependencies/relationships are modeled in the deployment diagram.
The systems support team and infrastructure team can understand the details of hardware and other physical components needed for the solution deployment. We could also understand the overall system topology, software used, the distribution model, machine configurations and such details. The infrastructure team can use this information to appropriately size the system and do the proper capacity planning. We have depicted the deployment diagram in Figure 15.14.

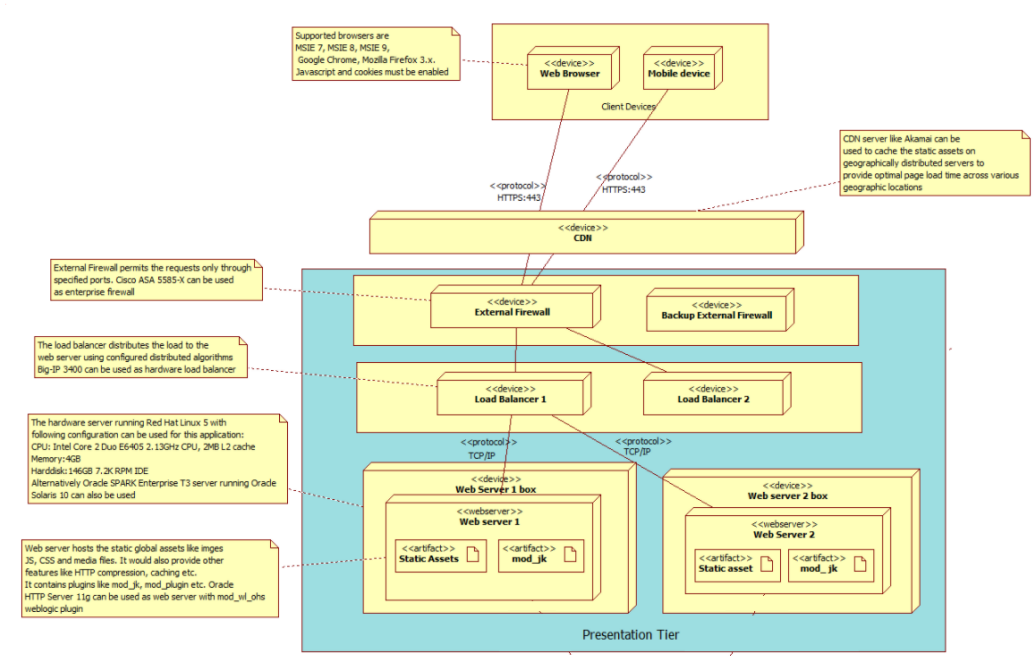

Figure 15.14 : Deployment Diagram

15.5.5 Package diagram

The package diagram depicts the packages and its constituent elements. We can organize the classes into logical groups through packages to enhance readability. The package diagram also depicts the relationship between packages. Package diagrams help us to logically group and organize the classes. By decomposing the system into subsystems and subsystems into packages we can organize the overall system better. We have depicted the package diagram in Figure 15.15.

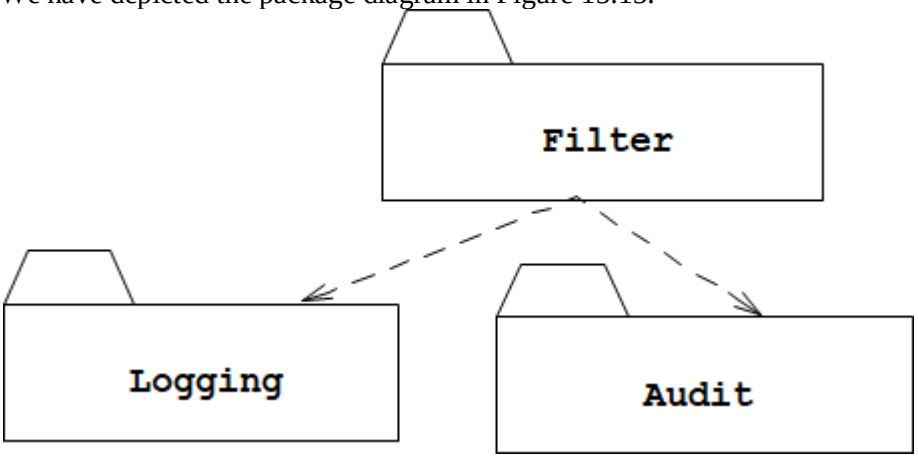

Figure 6.15 : Package Diagram

☞ **Check Your Progress 1**

1. _____ hides the inner details of the entity.
2. Encapsulation is also called _____
3. Two broad categories of UML diagrams are_____.
4. Four views covered by UML diagrams are_____.
5. _____ depicts the inheritance (parent-child relationship) in the class diagram
6. _____ Diagram depicts the software artefacts and the physical components of the infrastructure.

15.6 BEHAVIORAL UML DIAGRAMS

Behavioral elements depicted in the UML diagrams are the dynamic parts of the system. Given below are the key behavioral elements depicted in the UML diagrams; normally they are "verbs" of the system.

- **Interaction**: An interaction between two elements mainly happens through message passing and is represented as arrow as shown in Figure 15.16.
-

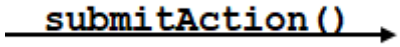

Figure 15.16 : Interaction

- **State machine**: The state machine represents various states of an object upon receiving an event. We have represented the state in Figure 15.17.

Figure 15.17 : State

15.6.1 State chart diagram

The state chart depicts the event-driven state change behavior through various states of a class and how the class can transition from one state to another for a specific event. The state chart diagrams depict the system behavior for external and internal events. The state chart has various element types – starting state or initial state which is the starting point of the process. When the object transitions to various States it is shown in the state chart diagram. The object in various states are connected by transition lines. We also depict the decision points in the state chart diagram. Statechart diagram helps us to understand the response of the system for the internal and external events. We have depicted the state chart diagram in Figure 15.18.

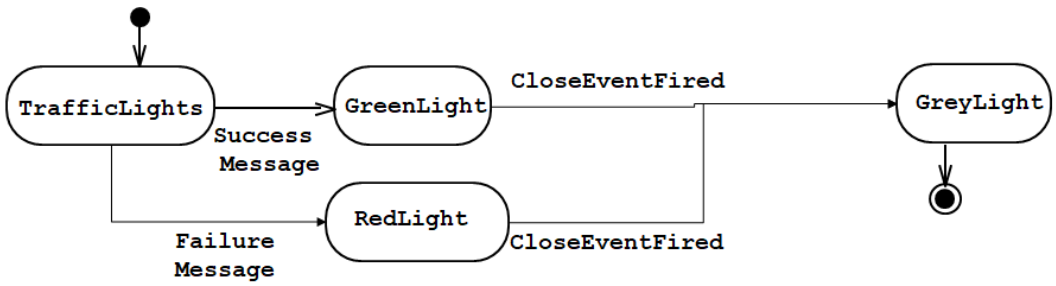

Figure 15.18 : State chart diagram

15.6.2 Activity diagram

The activity diagram depicts the control flow from one activity to another. We can model the system workflows, business rules, business process and understand the parallel and sequential activities through the activity diagram. Activity diagram also depicts the use case execution steps. Activity diagram is a type of state chart diagram in which states are depicted as activities. Activity diagrams are business-friendly. Similar to state chart diagram activity diagrams have initial stage with transition lines connecting various activities in the business process. The conditions can to be depicted in the activity diagram. The final activities that terminate the business process connected to the termination point.

We have depicted activity diagram in Figure 15.19.

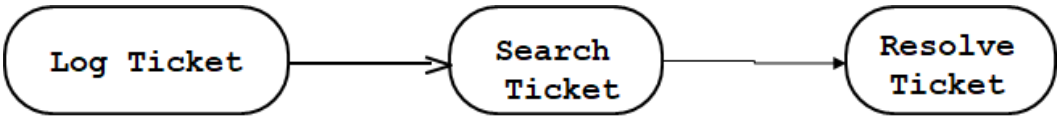

Figure 15.19 : Activity Diagram

15.6.3 Sequence diagram

The sequence diagram depicts the flow for a specific use case or a scenario. The objects perform the "call" by passing the messages. The sequence of messages are ordered by time order in which they occur. Sequence diagrams help us to understand the objects and the order in which they are invoked for the execution of a specific use case.
As depicted in Figure 15.20, arrows represent the messages, narrow vertical rectangle depict the activations and classes are in the top columns

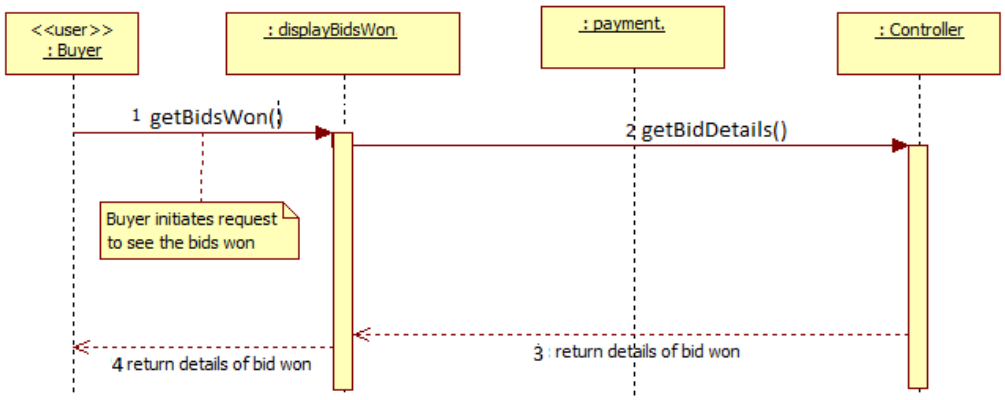

Figure 15.7 : Sequence Diagram

15.6.4 Use case diagram

Use case diagram depicts the functional behavior of the system in the visual format; use case diagram usually represent a single logical use case. The use case diagram provides visual depiction of relationship between the actors, use cases and various components for a specific functionality from the user's point of view. We can also depict the interactions of the system with external interfaces and actors.

We have depicted the use case diagram in Figure 15.21.

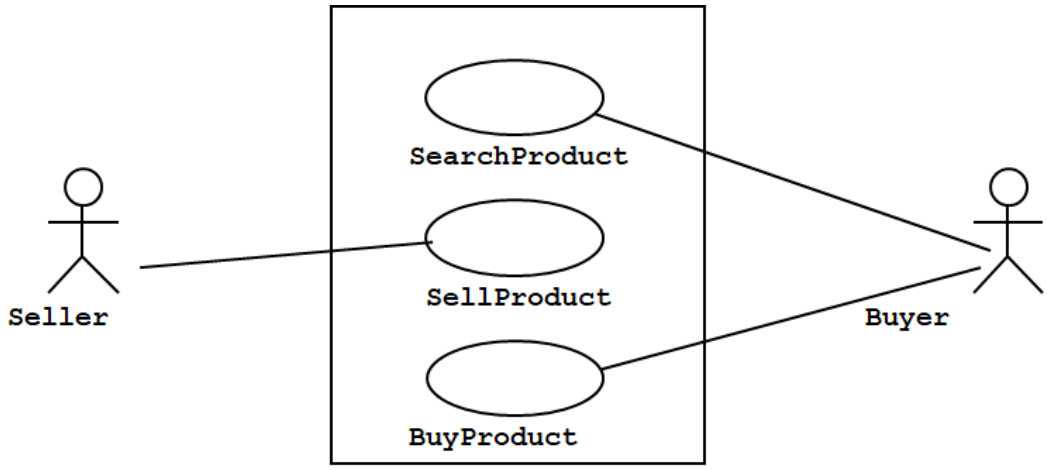

Figure 15.8 : Use case diagram

15.6.5 Collaboration diagram

A collaboration diagram consists of objects and links that represent the systems and the messages sent across the systems. A collaboration diagram is similar to a sequence diagram.

Check Your Progress 2

1. In sequence diagram _____ depicts the activations
2. _____ diagram depicts the control flow from one activity to another
3. The sequence of messages are ordered by _____
4. _____ diagram depicts relationship between the actors, use cases and various components
5. An interaction is normally represented by _____

15.7 SUMMARY

In this unit, we started discussing the key aspects of object oriented programming such as encapsulation, abstraction, inheritance and others. UML diagrams are industry standard for modeling the objects in the object oriented design. There are broadly two types of UML diagrams – structural diagrams that depict the static elements and the behavioral diagrams that depict the dynamic nature of the elements. Class diagrams depict the classes along with their relationship. Object diagrams depict the class instance along with its state. Component diagram depicts the components along with its interfaces. The deployment diagram depicts the software artefacts and the infrastructure elements. State chart diagram depicts the event-driven state transitions, Activity diagrams model the control flow and sequence diagrams provide time order sequence of objects. The use case diagram model a specific use case and the actors involved in the use case. Collaboration diagram depicts the objects and the links.

15.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1. Abstraction.
2. Information hiding or data hiding.
3. process view, implementation view, deployment view and design view

4. Generalization
5. Deployment

Check Your Progress 2

1. Narrow vertical rectangle.
2. Activity
3. Time order
4. Use case
5. arrow

15.9 FURTHER READINGS

References

The Unified Modeling Language User Guide, [G. Booch, J. Rumbaugh, I. Jacobson, 2000]

UML Explained, [Kendall Scott, 2001]

Applying UML and Patterns 2nd Ed., [Craig Larman, 2002]

UML Distilled 2nd Ed., [Martin Fowler with K. Scott, 2000

Rumbaugh, James, Ivar Jacobson, Grady Booch, the Unified Modeling Language Reference Manual, Addison Wesley, 1999

Jacobson, Ivar, Grady Booch, James Rumbaugh, the Unified Software Development Process, Addison Wesley, 1999

http://en.wikipedia.org/wiki/Object-oriented_programming