LECTURE HANDOUTS

L 01

MCA **I / II**

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction **Date of Lecture: 22.02.2021**

Topic of Lecture: An overview – Object basics

Introduction: Object-oriented analysis and design (OOAD) is a technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the software development process to guide stakeholder communication and product quality.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Class and Object
- Message, operation and method
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

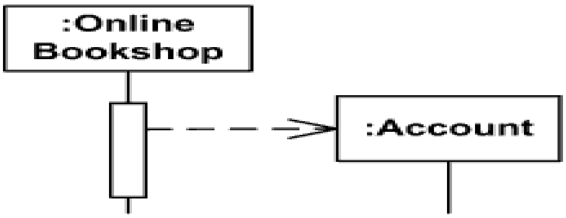
Detailed content of the Lecture:

Class and Object

UML class is a classifier which describes a set of objects that share the same features, constraints, semantics. Class may be modeled as being active, meaning that an instance of the class has some autonomous behavior. Object is an instance of a class.

Message, operation and method

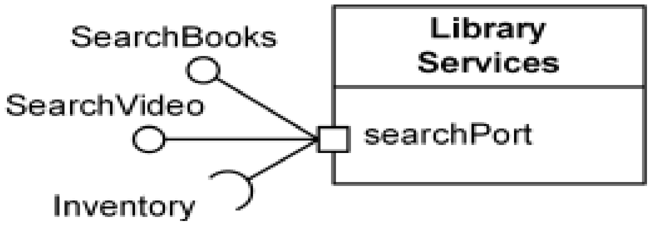
Messages are intrinsic elements of UML interaction diagrams. A message defines a specific kind of communication between lifelines of an interaction. A communication can be, for example, invoking an operation, replying back, creating or destroying an instance, raising a signal. It also specifies the sender and the receiver of the message. Create message is shown as a dashed line with open arrowhead, and pointing to the created lifeline's head.



Encapsulation

Encapsulation was describing abstraction mechanisms in programming language CLU in the context of hiding details of implementation**.** Encapsulation is a development technique which includes creating new data types classes by combining both information structure and behaviors, and restricting access to implementation details.

Encapsulated classifier in UML is a structured classifier isolated from its environment by using ports. Each port specifies a distinct interaction point between classifier and its environment.



Abstraction

Abstraction is a dependency relationship that relates two elements or sets of elements called client and supplier representing the same concept but at different levels of abstraction or from different viewpoints.

Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification the supplier and the other represents an implementation of the latter the client.

Inheritance

Inheritance as the mechanism by which those more specific elements incorporate structure and behavior of the more general elements. Inheritance supplements generalization relationship.

Generalization is defined as a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains some additional information. An instance of the more specific element may be used where the more general element is allowed.

Polymorphism

Polymorphism is ability to apply different meaning semantics, implementation to the same symbol message, operation in different contexts.

When context is defined at compile time, it is called static or compile-time polymorphism**.** When context is defined during program execution, it is dynamic or run-time polymorphism.

OOAD polymorphism means dynamic polymorphism and is commonly related to as late binding or dynamic binding.

Dynamic Polymorphism is ability of objects of different classes to respond to the same message in a different way.

Video Content / Details of website for further learning (if any):
https://www.uml-diagrams.org/uml-object-oriented-concepts.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:3-15)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 02

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **I - Introduction** **Date of Lecture: 23.02.2021**

Topic of Lecture: Object state and properties, Behavior, Methods

Introduction: All the objects have a state, behavior and identity. State of an object - The state or attributes are the built-in characteristics or properties of an object.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Class
- Object
- Attributes
- Function

Detailed content of the Lecture:

- **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
- **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

State of an object - The state or attributes are the built-in characteristics or properties of an object. For example, a T.V has the size, colour, model etc. Behaviour of the object - The behavior or operations of an object are its predefined functions.

Object − Objects have states and behaviors. Example: **A dog has states - color, name, breed as well as behaviors** – wagging the tail, barking, eating. An object is an instance of a class. Class − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

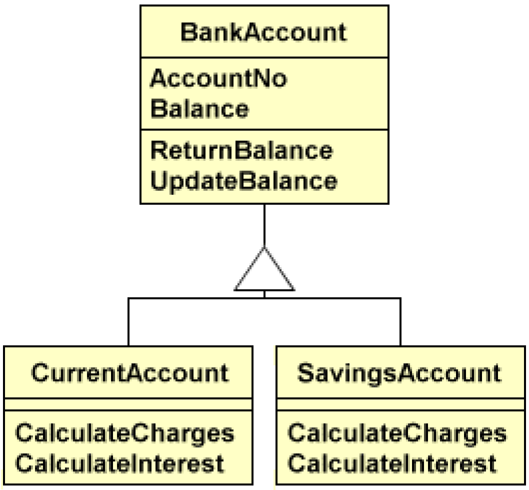
Behavior **binds the structure of objects to their attributes and relationships so that objects can meet their responsibilities**. Ultimately, an object's operations implement its behavior. There are means for constraining and controlling these primitive operations into permissible sequences.

- **State of an object** - The state or attributes are the built-in characteristics or properties of an object. For example, a T.V has the size, colour, model etc.

- **Behaviour of the object** - The behavior or operations of an object are its predefined functions. For example, a T.V. can show picture, change channels, tune for a channel etc. in object-oriented programming terminology the behavior is implemented through methods.
- **Object identity** - Each object is uniquely identifiable. For example, the fridge cannot become the T.V.

A **method** in object-oriented programming (OOP) is a procedure associated with a message and an object. An object consists of *data* and *behavior*; these comprise an *interface*, which specifies how the object may be utilized by any of its various consumers.

Data is represented as properties of the object, and behaviors are represented as methods. For example, a Window object could have methods such as open and close, while its state (whether it is open or closed at any given point in time) would be a property.



Video Content / Details of website for further learning (if any):
https://en.wikipedia.org/wiki/Method_(computer_programming)

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:16-18)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 03

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction **Date of Lecture: 24.02.2021**

Topic of Lecture: Messages, Information hiding

Introduction: Message is a named element that defines one specific kind of communication between lifelines of an interaction. Information hiding or data hiding in programming is about protecting data or information from any inadvertent change throughout the program

Prerequisite knowledge for Complete understanding and learning of Topic:

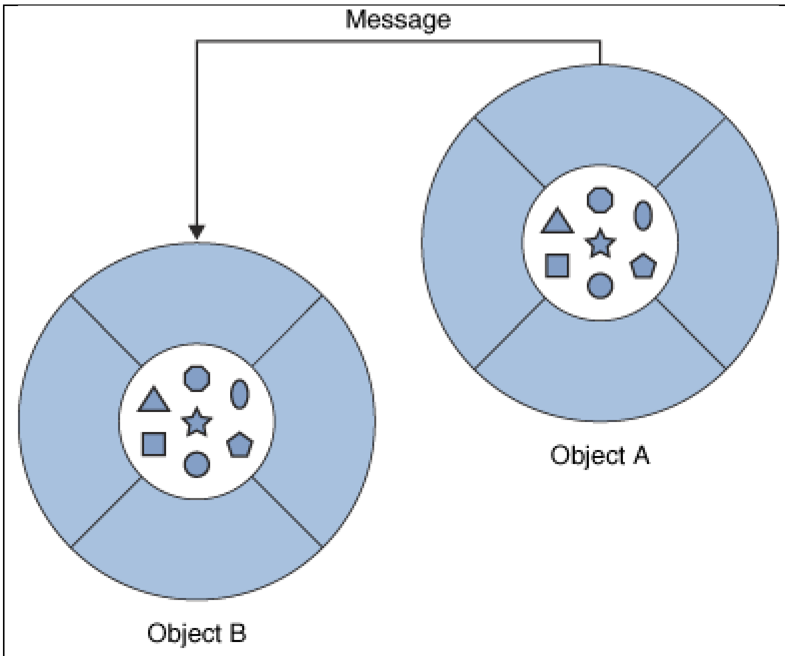
- Classes
- Objects
- Methods
- Data

Detailed content of the Lecture:

A message represents either an operation call or the sending and receiving of a signal. When the message represents an operation, the operation name identifies the message. The arguments of the message are passed to the target source. The return message contains the arguments of the resulting operation call.

A single object alone is generally not very useful. Instead, an object usually appears as a component of a larger program or application that contains many other objects. Through the interaction of these objects, programmers achieve higher-order functionality and more complex behavior. Your bicycle hanging from a hook in the garage is just a bunch of metal and rubber; by itself, the bicycle is incapable of any activity. The bicycle is useful only when another object (you) interacts with it (pedal).

Software objects interact and communicate with each other by sending messages to each other. When object A wants object B to perform one of B's methods, object A sends a message to object B (see the following figure).



- Information hiding is the principle of segregation of the design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed. The protection involves providing a stable interface which protects the remainder of the program from the implementation (whose details are likely to change). Written in another way, information hiding is the ability to prevent certain aspects of a class or software component from being accessible to its clients, using either programming language features (like private variables) or an explicit exporting policy. the requirements.

Information hiding definition

Information or data hiding is a programming concept which protects the data from direct modification by other parts of the program.

The feature of information hiding is applied using Class in most of the programming languages.

- One advantage of information hiding is yielding flexibility, such as allowing a programmer to more readily modify a program. This also may be done by placing source code within modules for easy access in the future, as the program develops and evolves

Video Content / Details of website for further learning (if any):
https://www.iitk.ac.in/esc101/05Aug/tutorial/java/concepts/message.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:18-20)

Course Faculty

Verified by HOD

LECTURE HANDOUTS **L 04**

MCA **I / II**

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction **Date of Lecture: 25.02.2021**

Topic of Lecture: Class hierarchy, Relationships

Introduction : A class hierarchy or inheritance tree in computer science is a classification of object types, denoting objects as the instantiations of inter-relating the various classes by relationships such as "inherits", "extends", "is an abstraction of", "an interface definition".

Prerequisite knowledge for Complete understanding and learning of Topic

- Building blocks of object-oriented modeling
- Link
- Dynamic structure of the system

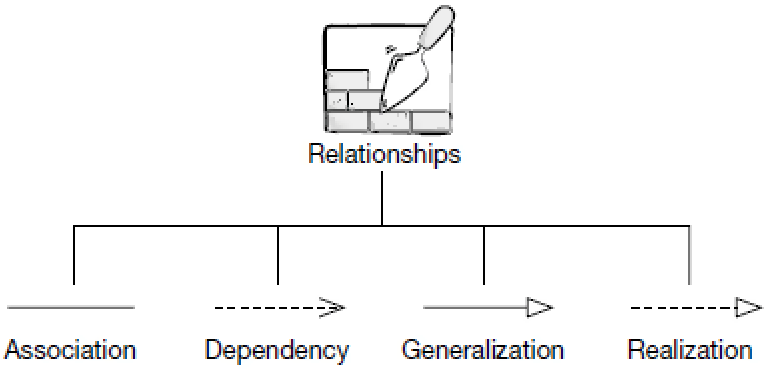
Detailed content of the Lecture:

A class hierarchy or inheritance tree in computer science is a classification of object types, denoting objects as the instantiations of classes (class is like a blueprint, the object is what is built from that blueprint) inter-relating the various classes by relationships such as "inherits", "extends", "is an abstraction of", "an interface definition". In object-oriented programming, a class is a template that defines the state and behaviour common to objects of a certain kind. A class can be defined in terms of other classes. The concept of class hierarchy in computer science is very similar to taxonomy, the classifications of species.

The relationships are specified in the science of object-oriented design and object interface standards defined by popular use, language designers (Java, C++, Smalltalk, Visual Prolog) and standards committees for software design like the Object Management Group.

The class hierarchy can be as deep as needed. The Instance variables and methods are inherited down through the levels and can be redefined according to the requirement in a subclass. In general, the further down in the hierarchy a class appears, the more specialized its behaviour. When a message is sent to an object, it is passed up the inheritance tree starting from the class of the receiving object until a definition is found for the method.

Class relationships define how various classes and kinds of classes are related to each other. For example, there is a special relationship between a class called Flower and a class called Rose: A Rose is a type of Flower. Relationships in UML are used to represent a connection between structural, behavioral, or grouping things. It is also called a link that describes how two or more things can relate to each other during the execution of a system. Type of UML Relationship are Association, Dependency, Generalization, and Realization.



Association

It is a set of links that connects elements of the UML model. It also defines how many objects are taking part in that relation.

Dependency

In a dependency relationship, as the name suggests, two or more elements are dependent on each other. In this kind of a relationship, if we make a change to a particular element, then it is likely possible that all the other elements will also get affected by the change.

Generalization

It is also called a parent-child relationship. In generalization, one element is a specialization of another general component. It may be substituted for it. It is mostly used to represent inheritance.

Realization

In a realization relationship of UML, one entity denotes some responsibility which is not implemented by itself and the other entity that implements them. This relationship is mostly found in the case of interfaces.

Association

It is a structural relationship that represents objects can be connected or associated with another object inside the system. Following constraints can be applied to the association relationship.

- {implicit} – Implicit constraints specify that the relationship is not manifest; it is based upon a concept.
- {ordered} – Ordered constraints specify that the set of objects at one end of an association are in a specific way.

Video Content / Details of website for further learning (if any):
https://www.guru99.com/uml-relationships-with-example.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:21-25)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 05

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **I - Introduction** **Date of Lecture: 26.02.2021**

Topic of Lecture: - Associations, Aggregations

Introduction: Association refers to "has a" relationship between two classes which use each other. Aggregation refers to "has a"+ relationship between two classes where one contains the collection of other class objects.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Relationship
- Class
- Object
- Interface

Detailed content of the Lecture:

Association:

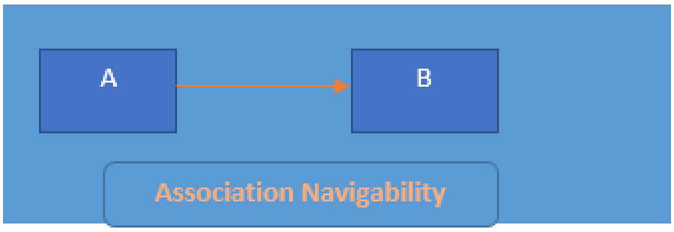
When we talk about the association in ooad, then this is nothing but a structural relationship, in object-oriented modeling, that specifies how objects are related to one another. This structural relationship can be shown in two forms:

- Class diagram associations
- Use case diagram associations.

Association represents the unidirectional or bidirectional relationship between two classes. If the Customer places an order, then this is a unidirectional association.

Bidirectional Association example: Person and Dog classes that setup a bidirectional association between a Person object and a Dog object. A Person object behaves like an owner for a Dog object and the Dog object behaves like a pet for the Person object. An association may exist between objects of different types or between the objects of the same type means of the same class. An association in UML is drawn as a solid line connecting the object of different classes or two objects of the same class.

If we want to show navigability, then we can use arrow (-->) at one end or both ends for bi-directional navigability. Here, if you see the below diagram then we have navigation from class A to class B. So class A somehow related to class B. So, we can say that class B is navigable from class A.



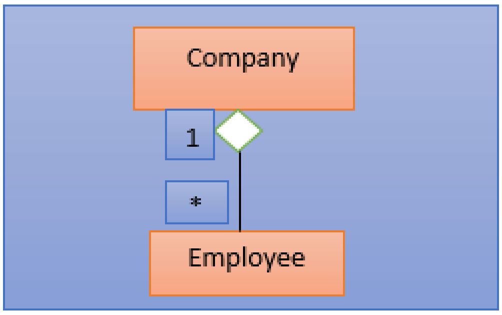
There are many types of associations in Java-like one to one, one to many, many to many and many to one.

Aggregation

In Aggregation, the based object is standalone and can exist even if the object of the control class is dead. Let’s take an example: suppose we have a class Car and car has a dependent object as a Wheel. Now when we destroy the car, the Wheel object is still alive because the wheel can be fit into different cars. So, here the association between Car and Wheel is the aggregation.

First of all, to exist an aggregation relationship, there would be some association. In a plain association, classes are considered at the same level with one class having no more important than the other. Sometimes we do need to define a “whole/part” relationship where one class considered as a “whole” that consists of smaller classes that is considered as a “part”. This kind of relationship is called aggregation. It is also known as “Has-A” relationship as object of the whole has the object of the parts.

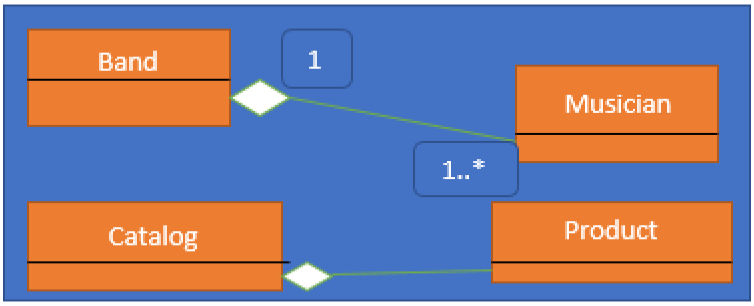
An aggregation in UML is drawn as an open diamond as below:



Multiplicity here is: 1 means Exactly one instance and * means zero or more instance

Aggregation and composition are both powerful forms of association. They both describe relationships between a whole and its parts. So, instead of "Has-A" relationship as a simple association, we deal with relationship that says is "part of" or reading relationship in other direction is "made of".

Examples of this kind of relationship is Band made up of Musician or in other words Musician is a part of Band. Another example is Catalog made up of Product or we can say that Product is part of a catalog.



1.*represents here At least one instance

Video Content / Details of website for further learning (if any):
https://www.nexsoftsys.com/articles/association-composition-aggregation-inheritance-java.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:27-30)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

LECTURE HANDOUTS

L 06

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction **Date of Lecture: 01.03.2021**

Topic of Lecture: Identity, Dynamic binding

Introduction: Object identity is a property of data that is created in the context of an object data model, where an object is assigned a unique internal object identifier, or oid. Dynamic binding also called dynamic dispatch is the process of linking procedure call to a specific sequence of code (method) at run-time. ... Dynamic binding is also known as late binding or run-time binding. Dynamic binding is an object-oriented programming concept and it is related with polymorphism and inheritance.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Behavior of the system
- Role
- Function
- Abstraction

Detailed content of the Lecture:

Identity of objects allows objects to be treated as black boxes. The object need not expose its internal structure. It can still be referred to, and its other properties can be accessed via its external behavior associated with the identity. The identity provides a mechanism for referring to such parts of the object that are not exposed in the interface. Thus, identity is the basis for polymorphism in object-oriented programming.

Identity allows comparison of references. Two references can be compared whether they are equal or not. Due to the identity property, this comparison has special properties. If the comparison of references indicates that the references are equal, then it's clear that the two objects pointed by the references are the same object. If the references do not compare equal, then it's not necessarily guaranteed that the identity of the objects behind those references is different. The object identity of two objects of the same type is the same, if every change to either object is also a change to the other object.

Identity allows the construction of a platonic ideal world, the ontology or conceptual model, that is often used as basis of object-oriented thinking. The conceptual model describes the client side view to a domain, terminology or an API. This world contains point-like objects as instances, properties of the objects and links between those objects. The objects in the world can be grouped to form classes. The properties of the objects can be grouped to form roles. The links can be grouped to form associations. All locations in the world together with the links between the locations form the structure of the world. These groups are types of the corresponding instances of the world

Dynamic binding also called dynamic dispatch is the process of linking procedure call to a specific sequence of code (method) at run-time. It means that the code to be executed for a specific procedure call is not known until run-time. Dynamic binding is also known as late binding or run-time binding. Dynamic binding(dispatch) means that a block of code executed with reference to a procedure(method) call is determined at run time.

Dynamic dispatch is generally used when multiple classes contain different implementations of the same method. It provides a mechanism for selecting the function to be executed from various function alternatives at the run-time.

Dynamic binding or late binding is the mechanism a computer program waits until runtime to bind the name of a method called to an actual subroutine. It is an alternative to early binding or static binding where this process is performed at compile-time. Dynamic binding is more expensive computationally, but it has the advantage of being more likely to avoid version conflicts when binding functions of a linked library.

Video Content / Details of website for further learning (if any):
https://www.defit.org/dynamic-binding/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:32-34)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 07

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction

Date of Lecture: 02.03.2021

Topic of Lecture: Persistence

Introduction: An object occupies a memory space and exists for a particular period of time. In files or databases, the object lifespan is longer than the duration of the process creating the object. This property by which an object continues to exist even after its creator ceases to exist is known as persistence.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Object
- Memory
- Databases
- OODBMS

Detailed content of the Lecture:

Persistence denotes a process or an object that continues to exist even after its parent process or object ceases, or the system that runs it is turned off. When a created process needs persistence, non-volatile storage, including a hard disk, is used instead of volatile memory like RAM.

A persistent process thus exists even if it failed or was killed for some technical reasons. The persistent state is the state that remains even after a process shuts down. Persistent process states are stored in persistent storage (non-volatile storage like hard drives) before the system fails or shuts down. They're easily retrieved once the system is turned on. The core processes of the operating system must be persistent for maintaining the data, device, or workspace in a similar state when the system is switched on. Types of Persistence

There are two types of persistence: object persistence and process persistence. In data terms, object persistence refers to an object that is not deleted until a need emerges to remove it from the memory. Some database models provide mechanisms for storing persistent data in the form of objects. In process persistence, processes are not killed or shut down by other processes and exist until the user kills them. For example, all of the core processes of a computer system are persistent for enabling the proper functioning of the system. Persistent processes are stored in non-volatile memory.

Object Persistence in Databases

A persistent database stores persistent data in the form of objects, or records that are durable when changing devices and software. Persistent data is stable and recoverable. Traditional relational database management systems (RDBMS) store persistent data in the form of records and tables. However, they cannot store objects and their relationships. Objects have necessary features (like encapsulation, inheritance, persistence, and polymorphism) that do not translate well into records and tables. Thus, certain special databases like object-oriented database management systems (OODBMS) and object relational database management systems (ORDBMS) are needed for storing objects and preserving object persistence.

Several approaches have been proposed to make the objects persistent:

1. persistence by class. Declare class to be persistent: all objects of the class are then persistent objects. Simple, not flexible since it is often useful to have both transient and persistent objects in a single class. In many OODB systems, declaring a class to be persistent is interpreted as ``persistable'' -- objects in the class potentially can be made persistent.
2. persistence by creation. Introduce new syntax to create persistent objects.
3. persistence by marking. Mark an object persistent after it is created (and before the program terminates).
4. persistence by reference. One or more objects are explicitly declared as (root) persistent objects. All other objects are persistent if they are referred, directly or indirectly, from a root persistent object. It is easy to make the entire data structure persistent by merely declaring the root of the structure as persistent, but is expensive to follow the chains in detection for a database system.

Video Content / Details of website for further learning (if any): https://study.com/academy/lesson/object-persistence-definition-overview.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:33-34)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 08

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction **Date of Lecture: 03.03.2021**

Topic of Lecture: Meta classes

Introduction: A meta class is used in object-oriented programming (OOP) and is typically thought of as a classes' class. A meta class is able to modify information from the class and can be linked to one or many classes, depending on the coding structure.

Prerequisite knowledge for Complete understanding and learning of Topic

- Classes
- Object Oriented Programming
- Groups
- Relationship

Detailed content of the Lecture:

In object-oriented programming, a metaclass is a class whose instances are classes. Just as an ordinary class defines the behavior of certain objects, a metaclass defines the behavior of certain classes and their instances. Not all object-oriented programming languages support metaclasses.

The class of a class is called a *metaclass*. A metaclass is created automatically for you whenever you create a class. Most of the time you do not need to care or think about metaclasses. However, every time that you use the browser to browse the "class side" of a class, it is helpful to recall that you are actually browsing a different class. A class and its metaclass are two separate classes, even though the former is an instance of the latter.

A MetaClass describes a real Class with the purpose of providing to an IDE class level information, and delaying the loading of that class to the last possible moment: when an instance of the class is required. A MetaClass binds the Class object from its class name using the appropriate class loader. Metaclasses expand the way to create a type beyond struct and class.

To make it clearer, a class describes what an object looks like, and is used at runtime to instantiate objects. A metaclass describes what a class looks like, and is used at compile time to instantiate classes.

The metaclass is then called with the name, bases and attributes of the class to instantiate it. However, metaclasses actually define the type of a class, not just a factory for it, so you can do much more with them.

Video Content / Details of website for further learning (if any):
https://www.fluentcpp.com/2018/03/09/c-metaclasses-proposal-less-5-minutes/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:34-35)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 09

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : I - Introduction **Date of Lecture: 04.03.2021**

Topic of Lecture: - Object oriented system development life cycle

Introduction:

- Software development process Process to change, refine, transform & add to existing product transformation 1(analysis) - translates user's need into system's requirements & responsibilities –how they use system can give insight into requirements

Prerequisite knowledge for Complete understanding and learning of Topic

- Analysis
- Design
- Implementation
- Testing
- Maintenance

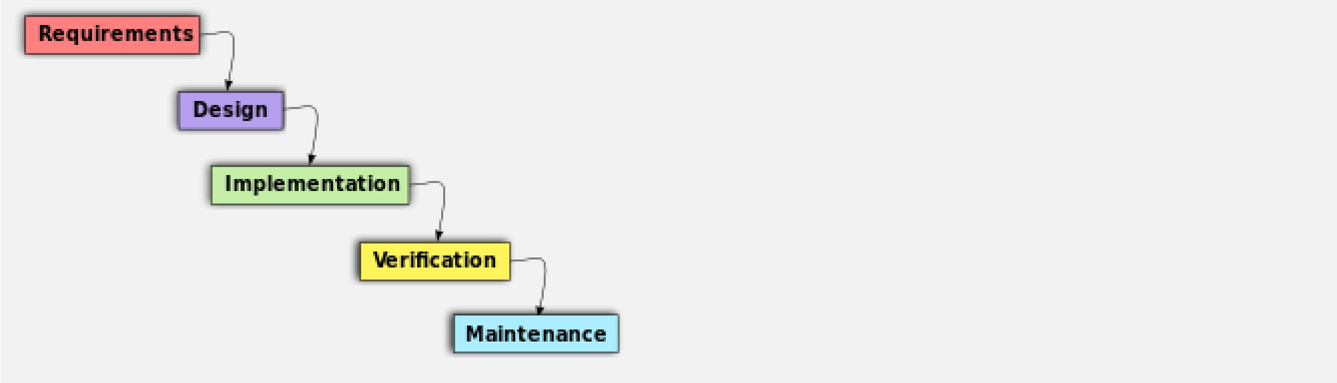
Detailed content of the Lecture:

Object-Oriented Analysis and Design (OOAD)

It's a structured method for analysing, designing a system by applying the object-orientated concepts, and develop a set of graphical system models during the development life cycle of the software.

OOAD In The SDLC

The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment.



The earliest stages of this process are analysis (requirements) and design. The distinction between analysis and design is often described as "what Vs how".

In analysis developers work with users and domain experts to define what the system is supposed to do. Implementation details are supposed to be mostly or totally ignored at this phase. The goal of the analysis phase is to create a model of the system regardless of constraints such as appropriate technology. This is typically done via use cases and abstract definition of the most important objects using conceptual model.

The design phase refines the analysis model and applies the needed technology and other implementation constrains. It focuses on describing the objects, their attributes, behavior, and interactions. The design model should have all the details required so that programmers can implement the design in code. They're best conducted in an iterative and incremental software methodologies. So, the activities of OOAD and the developed models aren't done once, we will revisit and refine these steps continually.

Object-Oriented Analysis

In the object-oriented analysis, we …

1. **Elicit** requirements: Define what does the software need to do, and what's the problem the software trying to solve.

2. **Specify** requirements: Describe the requirements, usually, using use cases (and scenarios) or user stories.

3. **Conceptual** model: Identify the important objects, refine them, and define their relationships and behavior and draw them in a simple diagram.

Object-Oriented Design

The analysis phase identifies the objects, their relationship, and behavior using the conceptual model (an abstract definition for the objects).

While in design phase, we describe these objects (by creating class diagram from conceptual diagram — usually mapping conceptual model to class diagram), their attributes, behavior, and interactions. The input for object-oriented design is provided by the output of object-oriented analysis. But analysis and design may occur in parallel, and the results of one activity can be used by the other.

In the object-oriented design, we …

1. Describe the classes and their relationships using class diagram.
2. Describe the interaction between the objects using sequence diagram.
3. Apply software design principles and design patterns.

A class diagram gives a visual representation of the classes you need. And here is where you get to be really specific about object-oriented principles like inheritance and polymorphism. Describing the interactions between those objects lets you better understand the responsibilities of the different objects, the behaviors they need to have.

System Modeling

System modeling is the process of developing models of the system, with each model representing a different perspectives of that system. The most important aspect about a system model is that it leaves out detail; It’s an abstract representation of the system.

The models are usually based on graphical notation, which is almost always based on the notations in the Unified Modeling Language (UML). Other models of the system like mathematical model; a detailed system description.

Models are used during the analysis process to help to elicit the requirements, during the design process to describe the system to engineers, and after implementation to document the system structure and operation.

Prototyping Prototype – version of software product developed in early stages of product's life cycle for specific, experimental purposes –Enables us to fully understand how easy/difficult it will be to implement some features of system –Gives users chance to comment on usability & usefulness of user interface design –Can assess fit between software tools selected, functional specification & user needs

Implementation: Component- based development: No more custom development, now assemble from prefabricated components –No more cars, computers, etc custom designed & built for each customer –Can produce large markets, low-cost, high-quality products –Cost & time reduced by building from pre-built, ready- tested components –Value & differentiation gained by rapid customization to targeted customers

Component-based development: Industrialised approach to system development, move form custom development to assembly of pre-built, pre-tested, reusable software components that operate with each other –Application development improved significantly if applications assembled quickly from prefabricated software components –Increasingly large collection of interpretable software components could be made available to developers in both general & specialist catalogs.

Video Content / Details of website for further learning (if any):
https://slideplayer.com/slide/5721776/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:39-53)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 10

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **II - Methodology & UML** **Date of Lecture: 05.03.2021**

Topic of Lecture: Introduction, Survey, Rumbaugh

Introduction: **Object Modeling Technique (OMT)** is real world-based modeling approach for software modeling and designing. It was developed basically as a method to develop object-oriented systems and to support object-oriented programming. It describes the static structure of the system.

Prerequisite knowledge for Complete understanding and learning of Topic

- Object
- Model
- Tool

Software

Detailed content of the Lecture:

The object-modeling technique (OMT) is an object modeling approach for software modeling and designing. It was developed around 1991 by Rumbaugh, Blaha, Premerlani, Eddy and Lorensen as a method to develop object-oriented systems and to support object-oriented programming. OMT describes object model or static structure of the system.

OMT was developed as an approach to software development. The purposes of modeling according to Rumbaugh are:

- testing physical entities before building them (simulation),
- communication with customers,
- visualization (alternative presentation of information), and
- reduction of complexity.

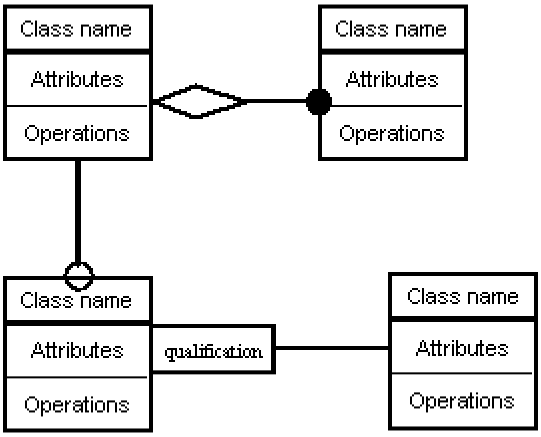
OMT has proposed three main types of models:

- **Object model:** The object model represents the static and most stable phenomena in the modeled domain. Main concepts are classes and associations with attributes and operations. Aggregation and generalization (with multiple inheritance) are predefined relationships.
- **Dynamic model:** The dynamic model represents a state/transition view on the model. Main concepts are states, transitions between states, and events to trigger transitions. Actions can be modeled as occurring within states. Generalization and aggregation (concurrency) are predefined relationships.

- **Functional model:** The functional model handles the process perspective of the model, corresponding roughly to data flow diagrams. Main concepts are process, data store, data flow, and actors.

OMT is a predecessor of the Unified Modeling Language (UML). Many OMT modeling elements are common to UML.

Functional Model in OMT: In brief, a functional model in OMT defines the function of the whole internal processes in a model with the help of "Data Flow Diagrams (DFDs)". It details how processes are performed independently.



- **Adv. Of OOM**

- It closely represents the problem domain

- Easy to understand

- Allow changes more easily

- Reusability-This approach is more nature

Video Content / Details of website for further learning (if any):
https://en.wikipedia.org/wiki/Object-modeling_technique

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:61-65)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 11

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : II - Methodology & UML **Date of Lecture: 06.03.2021**

Topic of Lecture: Booch, Jacobson methods

Introduction: The Booch method is a method for object-oriented software development. It is composed of an object modeling language, an iterative object-oriented development process, and a set of recommended practices. OOSE is the first object-oriented design methodology that employs use cases in software design. OOSE is one of the precursors of the Unified Modeling Language (UML), such as Booch and OMT.

Prerequisite knowledge for Complete understanding and learning of Topic

- Object Oriented Concepts
- Modeling
- Software
- Analysis
- Design

Detailed content of the Lecture:

In software engineering the Booch method, that is published in 1991 by Grady Booch, is a widely used method in object-oriented analysis and design. The Booch method has been superseded by UML, which features elements from the Booch method with OMT and OOSE.

The Booch method helps to design systems using the object paradigm. It covers the analysis- and design phases of an object-oriented system. The method defines different models to describe a system and it supports the iterative and incremental development of systems.

The Booch method includes six types of diagrams such as class diagrams, object diagrams, state transition diagrams, module diagrams, process diagrams an interaction diagram.

The process is organized around a macro and a micro process.

The macro process identifies the following activities cycle:

- Conceptualization: establish core requirements
- Analysis: develop a model of the desired behavior
- Design: create an architecture

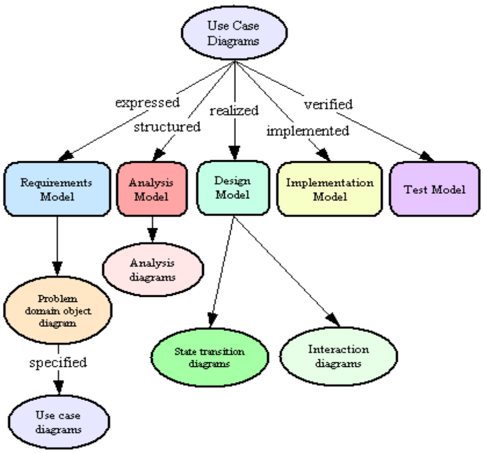
- Evolution: for the implementation
- Maintenance: for evolution after the delivery

The micro process is applied to new classes, structures or behaviors that emerge during the macro process. It is made of the following cycle:

- Identification of classes and objects
- Identification of their semantics
- Identification of their relationships
- Specification of their interfaces and implementation

Jacobson Methods OOSE is developed by Ivar Jacobson in 1992. OOSE is the first object-oriented design methodology that employs use cases in software design. OOSE is one of the precursors of the Unified Modeling Language (UML), such as Booch and OMT.

It includes a requirements, an analysis, a design, an implementation and a testing model.Interaction diagrams are similar to UML's sequence diagrams. State transition diagrams are like UML statechart diagrams.



Video Content / Details of website for further learning (if any):
https://bcanotes4u171863936.wordpress.com/jacobson-methodology/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:65-70)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 12

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **II - Methodology & UML** **Date of Lecture: 08.03.2021**

Topic of Lecture: Unified modeling language, Static and Dynamic models

Introduction: Static modeling is used to specify the structure of the objects, classes or components that exist in the problem domain. These are expressed using class, object or component. While dynamic modeling refers to representing the object interactions during runtime.

Prerequisite knowledge for Complete understanding and learning of Topic

- Modeling
- SDLC
- Object
- Class
- System

Detailed content of the Lecture:

Static modeling is used to specify structure of the objects that exist in the problem domain. These are expressed using class, object and USECASE diagrams. But Dynamic modeling refers representing the object interactions during runtime. It is represented by sequence, activity, collaboration and state chart diagrams.

The dynamic model is used to express and model the behavior of the system over time. It includes support for activity diagrams, state diagrams, sequence diagrams and extensions including business process modelling.

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. Unified Modeling Language (UML) is a general-purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language, it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis.

The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. It’s been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. **Structural Diagrams** – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
2. **Behavior Diagrams** – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

Important Books/Journals for further learning including the page nos.:
Martin Fowler, UML Distilled A Brief Guide to Standard Object Modeling Language, 3rd Edition, Addison Wesley,2003(Page No:89-91)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

DESIGNING YOUR FUTURE
Estd. 2000

IQAC

LECTURE HANDOUTS

L 13

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : II - Methodology & UML **Date of Lecture: 09.03.2021**

Topic of Lecture: Rational Rose Suite, UML diagrams

Introduction: Rational Rose is an object-oriented Unified Modeling Language (UML) software design tool intended for visual modeling and component construction of enterprise-level software applications.

Prerequisite knowledge for Complete understanding and learning of Topic

- Tools
- Packages
- Modeling
- Open-Source Software
- Framework

Detailed content of the Lecture:

Rational Rose is an object-oriented Unified Modeling Language (UML) software design tool intended for visual modeling and component construction of enterprise-level software applications. In much the same way a theatrical director blocks out a play, a software designer uses Rational Rose to visually create (model) the framework for an application by blocking out classes with actors (stick figures), use case elements (ovals), objects (rectangles) and messages/relationships (arrows) in a sequence diagram using drag-and-drop symbols. Rational Rose documents the diagram as it is being constructed and then generates code in the designer's choice of C++, Visual Basic, Java, Oracle8, Corba or Data Definition Language.

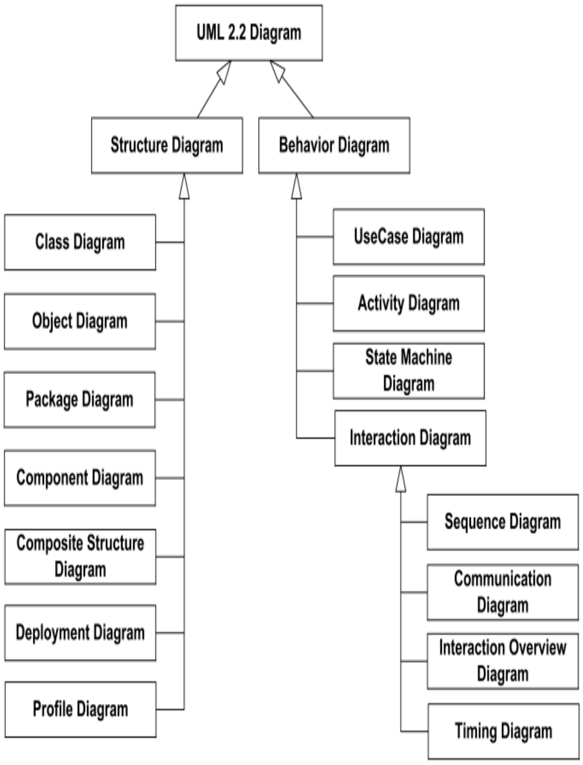
Two popular features of Rational Rose are its ability to provide iterative development and round-trip engineering. Rational Rose allows designers to take advantage of iterative development because the new application can be created in stages with the output of one iteration becoming the input to the next. Then, as the developer begins to understand how the components interact and makes modifications in the design, Rational Rose can perform what is called "round-trip engineering" by going back and updating the rest of the model to ensure the code remains consistent.

Rational Rose is extensible, with downloadable add-ins and third-party partner applications. It supports COM/DCOM (ActiveX), JavaBeans, and Corba component standards.

UML is an acronym that stands for **Unified Modeling Language**. Simply put, UML is a modern approach to modeling and documenting software. In fact, it’s one of the most popular business process modeling techniques.

It is based on **diagrammatic representations** of software components. As the old proverb says: “a picture is worth a thousand words”. By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

Mainly, UML has been used as a general-purpose modeling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficacy.



Video Content / Details of website for further learning (if any):
https://tallyfy.com/uml-diagram/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:94-99)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 14

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **II - Methodology & UML** **Date of Lecture: 10.03.2021**

Topic of Lecture: Static diagram, Class diagram, Use case diagrams

Introduction: A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time." Class diagrams are the backbone of almost every object-oriented method, including UML. They describe the static structure of a system.

Prerequisite knowledge for Complete understanding and learning of Topic

- Class
- Object
- Methods
- Link
- Relationship

Detailed content of the Lecture:

The Unified Modeling Language is a standardized general-purpose modeling language and nowadays is managed as a de facto industry standard by the Object Management Group (OMG). The creation of UML was originally motivated by the desire to standardize the disparate notational systems and approaches to software design. It was developed by Grady Booch, Ivar Jacobson, and James Rumbaugh at Rational Software in 1994–1995, with further development led by them through 1996.Static modeling is used to specify the structure of the objects, classes or components that exist in the problem domain.

The UML Class diagram is a graphical notation used to construct and visualize object-oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:

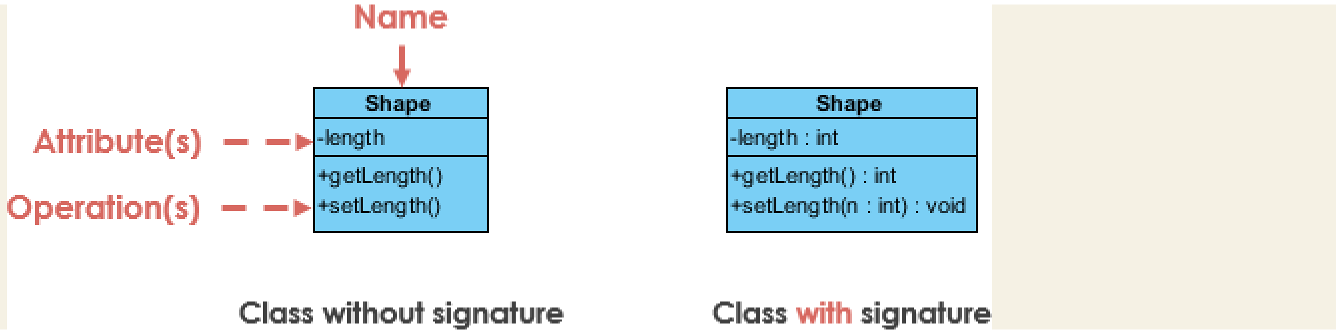
- classes,
- their attributes,
- operations (or methods),
- and the relationships among objects.

A Class is a blueprint for an object. Objects and classes go hand in hand. We can't talk about one without talking about the other. And the entire point of Object-Oriented Design is not about objects, it's about classes, because we use classes to create objects. So, a class describes what an object will be, but it isn't the object itself.

In fact, classes describe the type of objects, while objects are usable instances of classes. Each Object was built from the same set of blueprints and therefore contains the same components (properties and methods). The standard meaning is that an object is an instance of a class and object - Objects have states and behaviors.

UML Class Notation

A class represent a concept which encapsulates state attributes and behavior operations. Each attribute has a type. Each operation has a signature. The class name is the only mandatory information.



Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases. We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

- Functionalities to be represented as use case
- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram

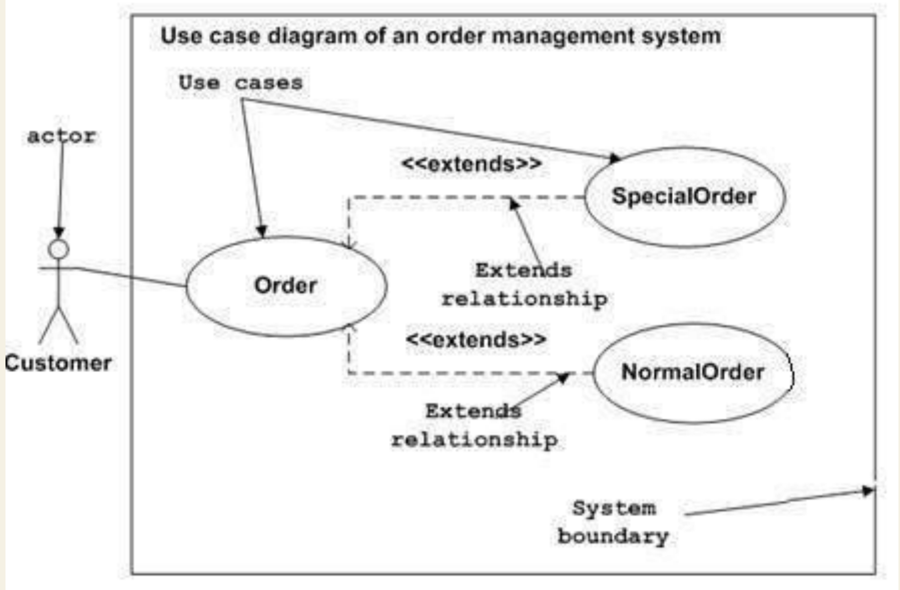


Figure: Sample Use Case diagram

Video Content / Details of website for further learning (if any):
https://www.smartdraw.com/uml-diagram/

Important Books/Journals for further learning including the page nos.:

Martin Fowler, UML Distilled A Brief Guide to Standard Object Modeling Language, 3rd Edition, Addison Wesley,2003(Page No:35-38)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 15

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : II - Methodology & UML **Date of Lecture: 11.03.2021**

Topic of Lecture: Behavior Diagram, Interaction diagram, State chart diagram

Introduction: Behavioral (or Dynamic) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams, and state machine diagrams. This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram

Prerequisite knowledge for Complete understanding and learning of Topic

- To capture the dynamic behavior of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

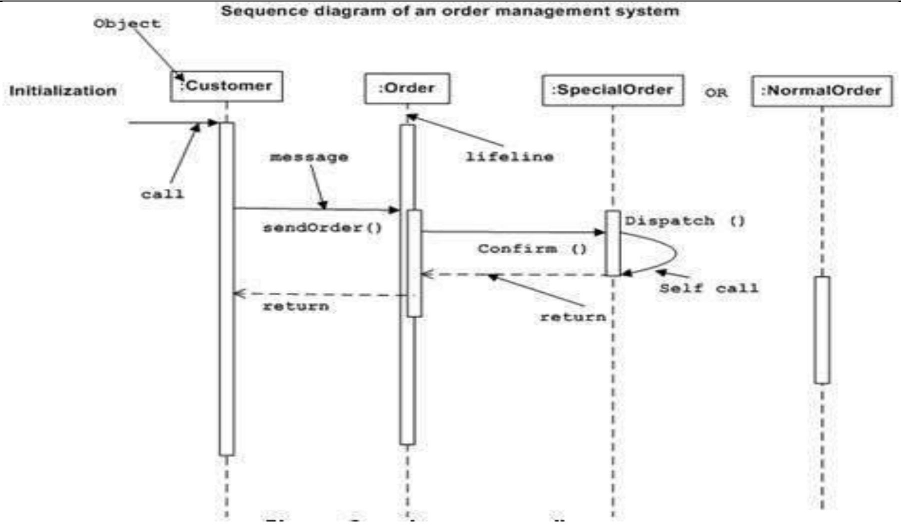
Detailed content of the Lecture:

Two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram.

Sequence Diagram

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder). The following diagram shows the message sequence for SpecialOrder object and the same can be used in case of NormalOrder object. It is important to understand the time sequence of message flows. The message flow is nothing but a method calls of an object.

The first call is sendOrder () which is a method of Order object. The next call is confirm () which is a method of SpecialOrder object and the last call is Dispatch () which is a method of SpecialOrder object. The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

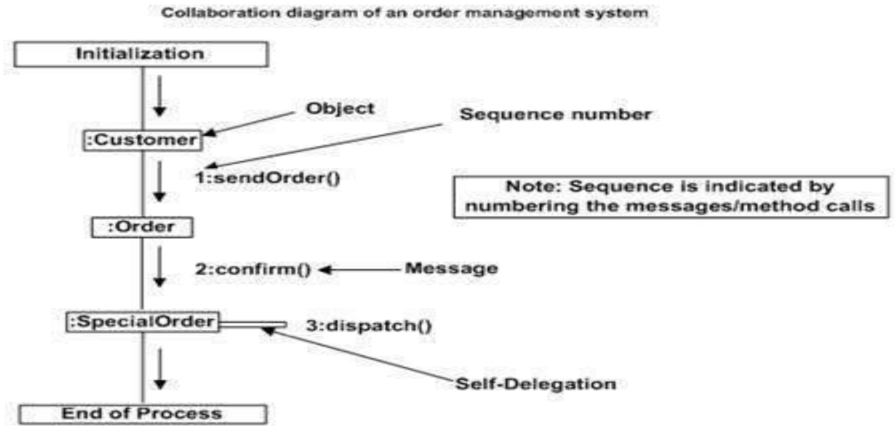


The Collaboration Diagram

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.



Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=Ba7SyM78cUM

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:106-108)

Course Faculty

Verified by HOD

DESIGNING YOUR FUTURE
Estd. 2000

IQAC

LECTURE HANDOUTS

L 16

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **I - Methodology & UML** **Date of Lecture: 12.03.2021**

Topic of Lecture: Activity diagram, Implementation diagram, Component diagram Deployment diagram, example

Introduction: Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Prerequisite knowledge for Complete understanding and learning of Topic

- Behavior of a system
- Interaction and concurrency
- Node

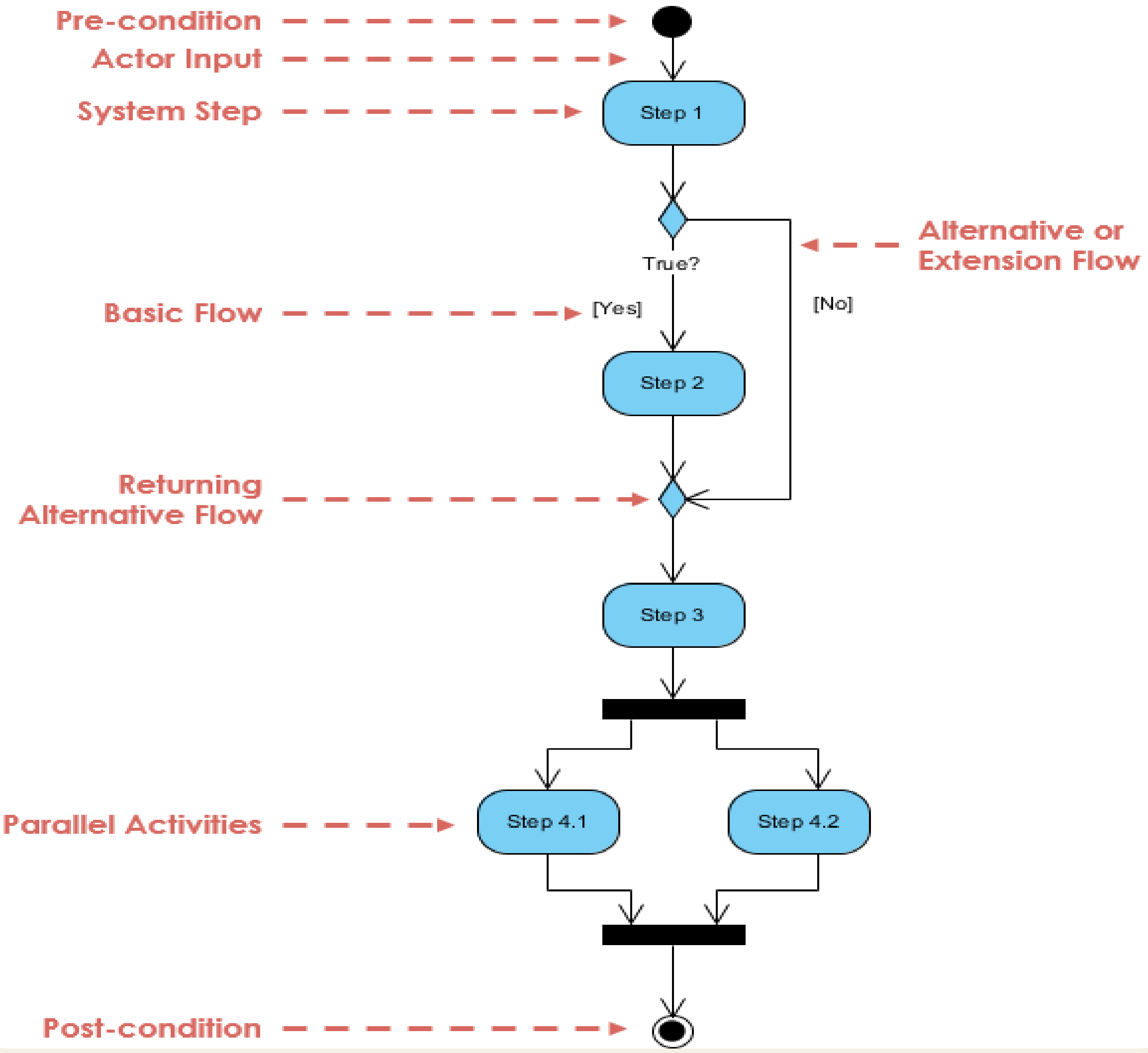
Detailed content of the Lecture:

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modeling how a collection of use cases coordinates to represent business workflows

1. Identify candidate use cases, through the examination of business workflows
2. Identify pre- and post-conditions (the context) for use cases
3. Model workflows between/within use cases
4. Model complex workflows in operations on objects
5. Model in detail complex activities in a high-level activity Diagram

Activity Diagram - Learn by Examples

A basic activity diagram - flowchart like



Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries, etc

The purpose of this diagram is different. Component diagrams are used during the implementation phase of an application. However, it is prepared well in advance to visualize the implementation details.

Initially, the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation. This diagram is very important as without it the application cannot be implemented efficiently. A well-prepared component diagram is also important for other aspects such as application performance, maintenance, etc.

Before drawing a component diagram, the following artifacts are to be identified clearly –

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

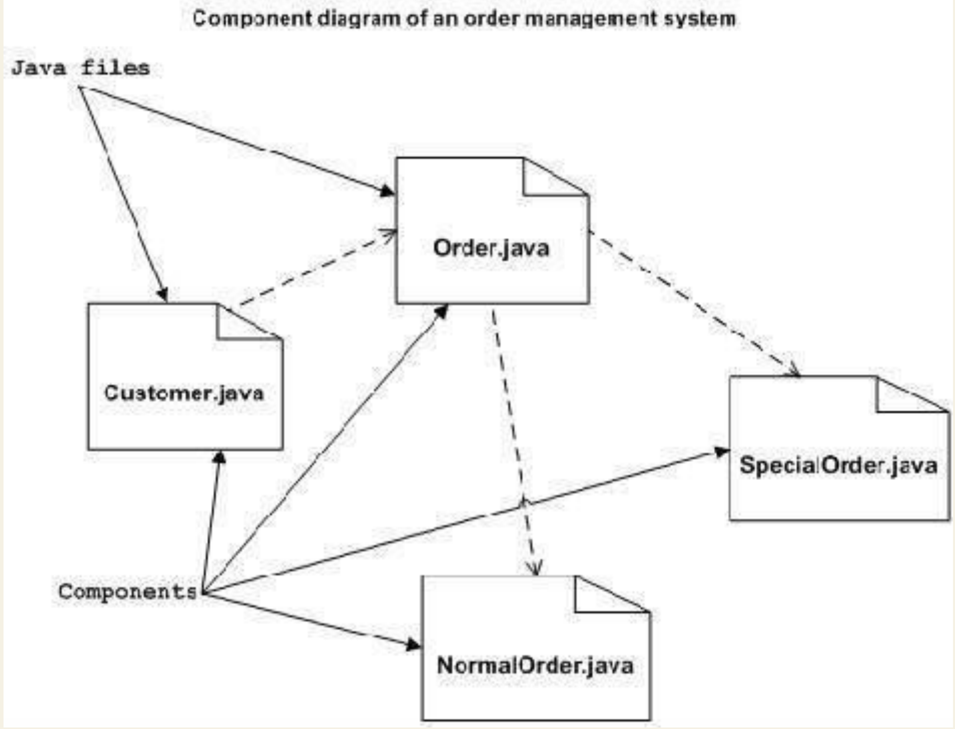
After identifying the artifacts, the following points need to be kept in mind.

- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing the using tools.
- Use notes for clarifying important points.

Following is a component diagram for order management system. Here, the artifacts are files. The diagram shows the files in the application and their relationships. In actual, the component diagram also contains dlls, libraries, folders, etc.

In the following diagram, four files are identified and their relationships are produced. Component diagram cannot be matched directly with other UML diagrams discussed so far as it is drawn for completely different purpose.

The following component diagram has been drawn considering all the points mentioned above.



Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Video Content / Details of website for further learning (if any):
https://www.youtube.com/watch?v=3bfJ5AORAjQ

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:109-112)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

DESIGNING YOUR FUTURE
Estd. 2000

IQAC

LECTURE HANDOUTS

L 17

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **II - Methodology & UML** **Date of Lecture: 13.03.2021**

Topic of Lecture: Design of online railway reservation system using UML diagrams

Introduction: To model the "**Online Railway Ticket Reservation System**" using the software Rational Rose with various UML (Unified Modeling Language) diagrams.

Prerequisite knowledge for Complete understanding and learning of Topic

- Dependencies between packages
- Visualize physical hardware and software
- Open-source software
- UML Diagrams
- Classes & Objects

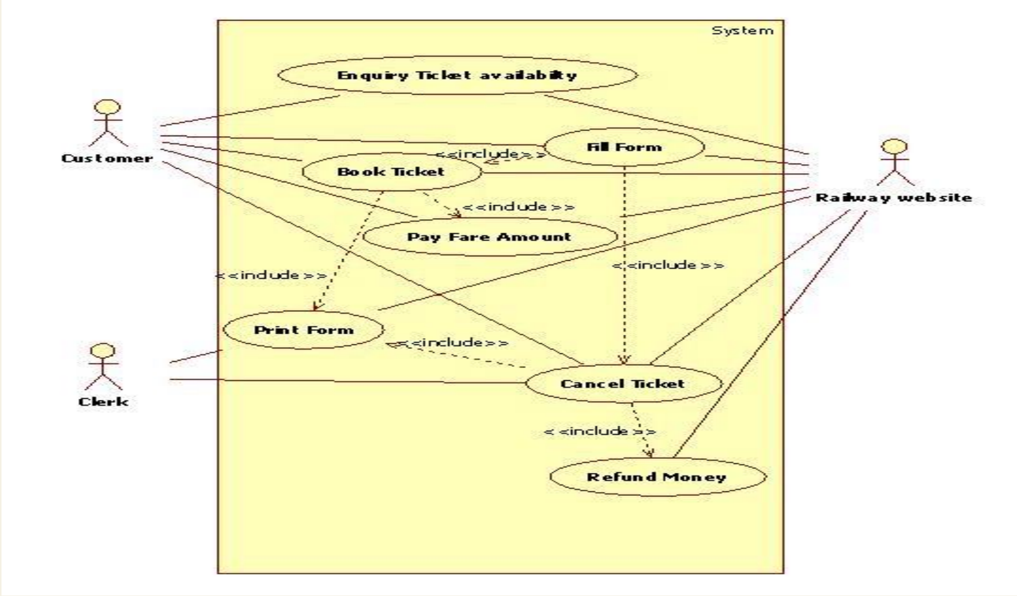
Detailed content of the Lecture:

Railway Reservation System is a system used for booking tickets over internet. Any Customer Can book tickets for different trains. Customer can book a ticket only if the tickets are available. Customer searches for the availability of tickets then if the tickets are available, he books the tickets by initially filling details in a form. Tickets can be booked in two ways by i-ticket or by e-ticket booking.

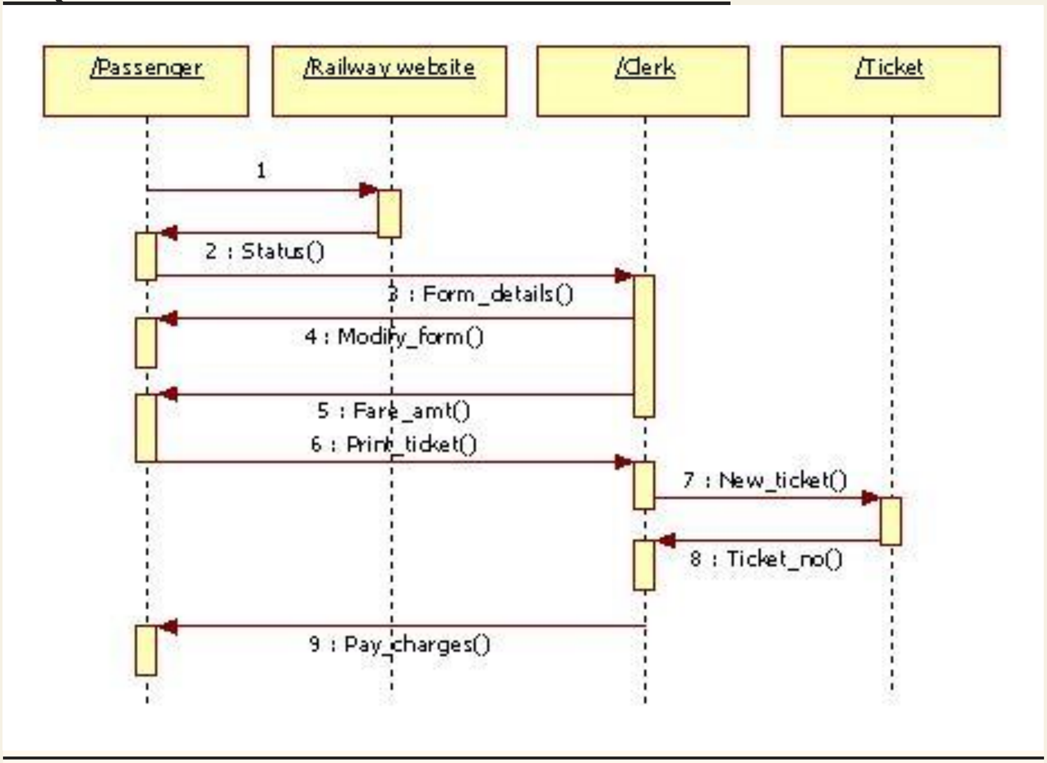
In case of i-ticket booking customer can book the tickets online and the tickets are couriered to Particular customer at their address. But in case of e-ticket booking and cancelling tickets are booked and cancelled online sitting at the home and customer himself has to take print of the ticket but in both the cases amount for tickets are deducted from customer’s account.

For cancellation of ticket the customer has to go at reservation office than fill cancellation form and ask the clerk to cancel the ticket than the refund is transferred to customer account. After booking ticket the customer has to checkout by paying fare amount to clerk.

Use Case Diagram :- Railway Reservation



SEQUENCE DIAGRAM FOR BOOKING TICKET:



Video Content / Details of website for further learning (if any):
http://www.programsformca.com/2012/03/uml-diagrams-for-railway-reservation.html /

Important Books/Journals for further learning including the page nos.: Martin Fowler, UML
http://www.programsformca.com/2012/03/uml-diagrams-for-railway-reservation.html /

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 18

MCA

I / II

Course Name with Code : 19C AB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : II - Methodology & UML

Date of Lecture: 15.03.2021

Topic of Lecture: Dynamic modeling, Model organization, Extensibility

Introduction: Dynamic Modeling represents **the temporal aspects of a system**, capturing the control elements through which the behavior of objects can be understood over time. ... It is necessary to model only those objects with a definite lifecycle or those which exhibit significant behavior.

Prerequisite knowledge for Complete understanding and learning of Topic

- System
- Modeling
- Behaviour
- Software Engineering

Detailed content of the Lecture:

The dynamic model represents the time–dependent aspects of a system. It is concerned with the temporal changes in the states of the objects in a system. The main concepts are −

- State, which is the situation at a particular condition during the lifetime of an object.
- Transition, a change in the state
- Event, an occurrence that triggers transitions
- Action, an uninterrupted and atomic computation that occurs due to some event, and
- Concurrency of transitions.

A state machine models the behavior of an object as it passes through a number of states in its lifetime due to some events as well as the actions occurring due to the events. A state machine is graphically represented through a state transition diagram.

The dynamic model represents the time–dependent aspects of a system. It is concerned with the temporal changes in the states of the objects in a system. The main concepts are −

- State, which is the situation at a particular condition during the lifetime of an object.
- Transition, a change in the state
- Event, an occurrence that triggers transitions
- Action, an uninterrupted and atomic computation that occurs due to some event, and
- Concurrency of transitions.

A state machine models the behavior of an object as it passes through a number of states in its lifetime due to some events as well as the actions occurring due to the events. A state machine is graphically represented through a state transition diagram.

States and State Transitions

State

The state is an abstraction given by the values of the attributes that the object has at a particular time period. It is a situation occurring for a finite time period in the lifetime of an object, in which it fulfils certain conditions, performs certain activities, or waits for certain events to occur. In state transition diagrams, a state is represented by rounded rectangles.

Parts of a state

- **Name** − A string differentiates one state from another. A state may not have any name.
- **Entry/Exit Actions** − It denotes the activities performed on entering and on exiting the state.
- **Internal Transitions** − The changes within a state that do not cause a change in the state.
- **Sub–states** − States within states.

Initial and Final States

The default starting state of an object is called its initial state. The final state indicates the completion of execution of the state machine. The initial and the final states are pseudo-states, and may not have the parts of a regular state except name. In state transition diagrams, the initial state is represented by a filled black circle. The final state is represented by a filled black circle encircled within another unfilled black circle.

Transition

A transition denotes a change in the state of an object. If an object is in a certain state when an event occurs, the object may perform certain activities subject to specified conditions and change the state. In this case, a state−transition is said to have occurred. The transition gives the relationship between the first state and the new state. A transition is graphically represented by a solid directed arc from the source state to the destination state.

The five parts of a transition are −

- Source State − The state affected by the transition.
- Event Trigger − The occurrence due to which an object in the source state undergoes a transition if the guard condition is satisfied.
- Guard Condition − A Boolean expression which if True, causes a transition on receiving the event trigger.
- Action − An un-interruptible and atomic computation that occurs on the source object due to some event.
- Target State − The destination state after completion of transition.

A model is a simplification at some level of abstraction We build models to better understand the systems we are developing. To help us visualize to specify structure or behavior to provide template for building system to document decisions we have made Principles of Modeling: The models we choose have a profound influence on the solution we provide Every model may be expressed at different levels of abstraction The best models are connected to reality No single model is sufficient, a set of models is needed to solve any nontrivial system

Purpose of Models:

1. Testing a physical entity before building it
2. Communication with customers
3. Visualization
4. Reduction of complexity

Extensibility is a software engineering and systems design principle that provides for future growth. Extensibility is a measure of the ability to extend a system and the level of effort required to implement the extension. Extensions can be through the addition of new functionality or through modification of existing functionality. The principle provides for enhancements without impairing existing system functions.

An extensible system is one whose internal structure and dataflow are minimally or not affected by new or modified functionality, for example recompiling or changing the original source code might be unnecessary when changing a system's behavior, either by the creator or other programmers. Because software systems are long lived and will be modified for new features and added functionalities demanded by users, extensibility enables developers to expand or add to the software's capabilities and facilitates systematic reuse. Some of its approaches include facilities for allowing users' own program routines to be inserted and the abilities to define new data types as well as to define new formatting markup tags.

Video Content / Details of website for further learning (if any):
https://www.fluentcpp.com/2018/03/09/c-metaclasses-proposal-less-5-minutes/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:114-117)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 19

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : III - Object Oriented Analysis Date of Lecture: 16.03.2021

Topic of Lecture: Identifying Use case, Business object analysis

Introduction: The traditional approaches to software development identified and defined rigid flows of procedures to support business processes or activities. With these approaches, data is defined separately from the procedures that act upon that data. Object-oriented analysis groups elements that interact with one another to create a model that represents the functionality and objectives of the system as a whole.

Prerequisite knowledge for Complete understanding and learning of Topic

- Actors
- Object
- Relationship
- Packages
- Analysis

Detailed content of the Lecture:

- Analysis is the process of transforming a problem definition from a fuzzy set of facts into a statement of a system's requirements.
- Analysis involves:
  - Identification of users who will be affected by the system.
- Analysis have following steps:
  - Examination of existing system documentation
  - Interviews
  - Questionnaire
  - Observation
- These activities must be done using use-case model which captures,
  - The user requirements

The input to the system

- Main process of analysis is understanding the problem, its associated constraints and methods.
- Three sources of requirement difficulties given by Norman:
- Fuzzy descriptions
- Fast response time, very easy and very secure
- Incomplete requirements

- Necessary requirements not included due to some reasons like forgetting to identify them and policies.

- Unnecessary features
- Every additional features affect the performance, complexity, stability, cost
- Use cases are scenarios for understanding system requirements.
- Use case is an interaction between users and a system.
- It represents the goal of the user and the responsibility of the system to its users.
- Also describes uses of the system and the set of events that can be performed by the system.
- Use case must have a name and short textual representation of not more than few paragraphs.

Business object analysis: understanding the business layer

- Understanding user requirements to set goals of system development.
- Identifying classes and relationships
- Use cases are used to understand system requirements.
- Understanding input and output
- What must be done and how it should be done
- Preparing prototype of an user interface
- Its valuable tool used in business analysis to understand well
- Once the user requirements are identified, those things need to be documented to proceed with design and implementation.
- This is an iterative process.

Video Content / Details of website for further learning (if any):
https://www.businessanalystlearnings.com/ba-techniques/2017/8/8/an-introduction-to-object-oriented-analysis

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:125-127)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 20

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : III - Object Oriented Analysis **Date of Lecture: 17.03.2021**

Topic of Lecture: Use case driven, object-oriented analysis

Introduction: **Use case** is a sequence of actions, performed by one or more **actors** (people or non-human entities outside of the system) and by the system itself, that produces one or more results of value to one or more of the actors. One of the key aspects of the Unified Process is its use of use cases as a driving force for development.

Prerequisite knowledge for Complete understanding and learning of Topic

- Object
- Classes
- Methods
- Analysis
- Actors

Detailed content of the Lecture:

Use-case driven object-oriented analysis: The unified approach

- OOA phase of the unified approach uses actors and use cases to describe the system.
- The actor is an external factor that interact with the system.
- Use cases are scenarios that describes how actors use the system.
- Identify the actors:
- Develop a simple business process model using UML activity diagram
- Develop the use case:
- Use cases are scenarios for understanding system requirements.
- Use case is an interaction between users and a system.
- It represents the goal of the user and the responsibility of the system to its users.
- Use case must have a name and short textual representation of not more than few paragraphs.

The Use Case model is at the conceptual center of the entire SDLC because it drives every development activity that follows. It is often developed in conjunction with a domain model where your (textual) problem statement was developed from. Basically, a use case model is more than just a single use case diagram but attached with all the related information of the use cases, for example:

- A use case may be originally derived from a problem statement where identify them from.
- Each of the use cases is attached with a use case description (textual) and in turn, we can detail the use cases by refining the use case description into the flow of events.
- Base on the flow of events we can then perform use case scenario analysis by representing them with a group of related sequence diagrams and etc.

A problem statement is developed for a problem

- Use Case identification (scoping and partitioning of a problem into a set of functional requirements)
- Detail each use case with the flow of events (from analysis to design)
- Based on the flow of events, we can develop a sequence diagram for representing each of the use case scenarios
- By refining a system-level sequence diagram into a more detailed MVC sequence diagram, we can partition a system functionality into a 3 layers reusable software architecture as a clear implementation specification.
- By consolidating the sequence diagrams for the corresponding use case, we can identify a group of candidate objects required by implementing the system functionality currently underdeveloped.

Video Content / Details of website for further learning (if any):
https://www.visual-paradigm.com/guide/agile-software-development/what-is-use-case-driven-approach-for-agile/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:127-129)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

DESIGNING YOUR FUTURE
Estd. 2000

IQAC

LECTURE HANDOUTS

L 21

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : III - Object Oriented Analysis Date of Lecture: 18.03.2021

Topic of Lecture: Use case model

Introduction: A use-case model is a model of how different types of users interact with the system to solve a problem. As such, it describes the goals of the users, the interactions between the users and the system, and the required behavior of the system in satisfying these goals.

Prerequisite knowledge for Complete understanding and learning of Topic

- Use case
- Process
- Links
- Methods
- System

Detailed content of the Lecture:

A use-case diagram is used to graphically depict a subset of the model to simplify communications. There will typically be several use-case diagrams associated with a given model, each showing a subset of the model elements relevant for a particular purpose. The same model element may be shown on several use-case diagrams, but each instance must be consistent. The use-case model may contain packages that are used to structure the model to simplify analysis, communications, navigation, development, maintenance and planning.

Much of the use-case model is in fact textual, with the text captured in the Use-Case Specifications that are associated with each use-case model element. These specifications describe the flow of events of the use case.

The use-case model serves as a unifying thread throughout system development. It is used as the primary specification of the functional requirements for the system, as the basis for analysis and design, as an input to iteration planning, as the basis of defining test cases and as the basis for user documentation

Basic model elements

The use-case model contains, as a minimum, the following basic model elements.

Actor

A model element representing each actor. Properties include the actors name and brief description.

Use Case

A model element representing each use case. Properties include the use case name and use case specification. See Artifact: Use Case and Concept: Use Case for more information.

Associations

Associations are used to describe the relationships between actors and the use cases they participate in. This relationship is commonly known as a “communicates-association”.

Use-Case Package

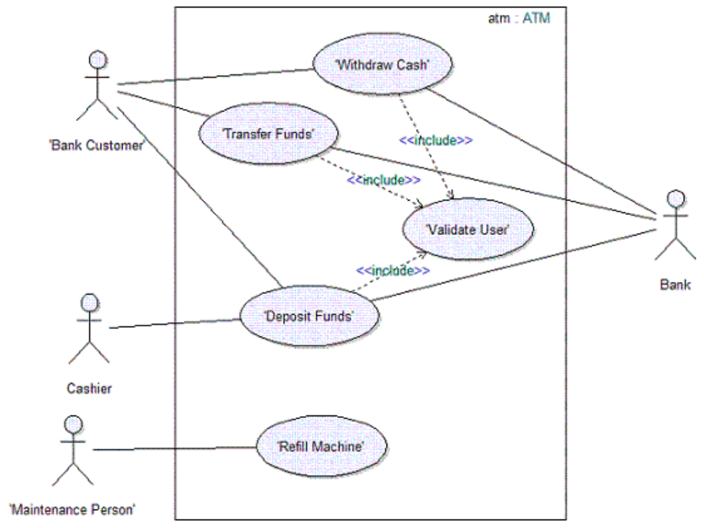
A model element used to structure the use case model to simplify analysis, communications, navigation, and planning. If there are many use cases or actors, you can use use-case packages to further structure the use-case model in much the same manner you use folders or directories to structure the information on your hard-disk.

Generalizations

A relationship between actors to support re-use of common properties.

Dependencies

A number of dependency types between use cases are defined in UML. In particular, <<extend>> and <<include>>.<<extend>> is used to include optional behavior from an extending use case in an extended use case.



Video Content / Details of website for further learning (if any):
https://www.utm.mx/~caff/doc/OpenUPWeb/openup/guidances/concepts/use_case_model_CD178AF9.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:129-137)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 22

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : III - Object Oriented Analysis **Date of Lecture: 19.03.2021**

Topic of Lecture: Documentation, Classification

Introduction: Classification is the means whereby we order knowledge. In object-oriented design, recognizing the sameness among things allows us to expose the commonality within key abstractions and mechanisms and eventually leads us to smaller applications and simpler architectures.

Prerequisite knowledge for Complete understanding and learning of Topic

- Dependencies between packages
- SDLC
- Visualize physical hardware and software

Detailed content of the Lecture:

Documentation is any communicable material that is used to describe, explain or instruct regarding some attributes of an object, system or procedure, such as its parts, assembly, installation, maintenance and use. Documentation can be provided on paper, online, or on digital or analog media, such as audio tape or CDs. Examples are user guides, white papers, online help, and quick-reference guides. Paper or hard-copy documentation has become less common. Documentation is often distributed via websites, software products, and other online applications.

Documentation as a set of instructional materials shouldn't be confused with documentation science, the study of the recording and retrieval of information.

The purpose of documentation is to: Describe the use, operation, maintenance, or design of software or hardware through the use of manuals, listings, diagrams, and other hard- or soft-copy written and graphic materials.

A single source of truth saves time and energy. ...

- Documentation is essential to quality and process control. ...
- Documentation cuts down duplicative work. ...
- It makes hiring and onboarding so much easier. ...
- A single source of truth makes everyone smarter.

Classification

Classification, the process of checking to see if an object belongs to a category or a class, is regarded as a basic attribute of human nature. Intelligent classification is part of all good science classification guides us in making decisions about modularization.

Superior Classification: Human beings classify information every instant of their waking lives. We recognize the objects around us, and we have and act in relation to them. A human being is a very sophisticated information system, partly because he (or) she possesses a superior classification capability.

The four alternative approaches for identifying classes:

- The noun phrase approach.
- The common class patterns approach.
- The use-case driven, sequence/collaboration modeling approach.
- The classes, responsibilities and collaborators (CRC) approach.

Noun phrase approach

- Look for the noun phrases through the use cases.

Three categories:

- Relevant classes.
- Fuzzy classes.
- Irrelevant classes.

Identifying tentative classes.

- Look for noun phrases and nouns in the use cases.
- Some classes are implicit or taken from general knowledge.
- All classes must make sense in the application domain.
- Carefully choose and define class names.

Selecting classes from the relevant and fuzzy classes.

- Redundant classes.
- Adjective classes.
- Attribute classes.
- Irrelevant classes.

The common class patterns approach.

The common class patterns approach is based on a knowledge base of the common classes that have been proposed researchers. The patterns used for finding the candidate class and object are:

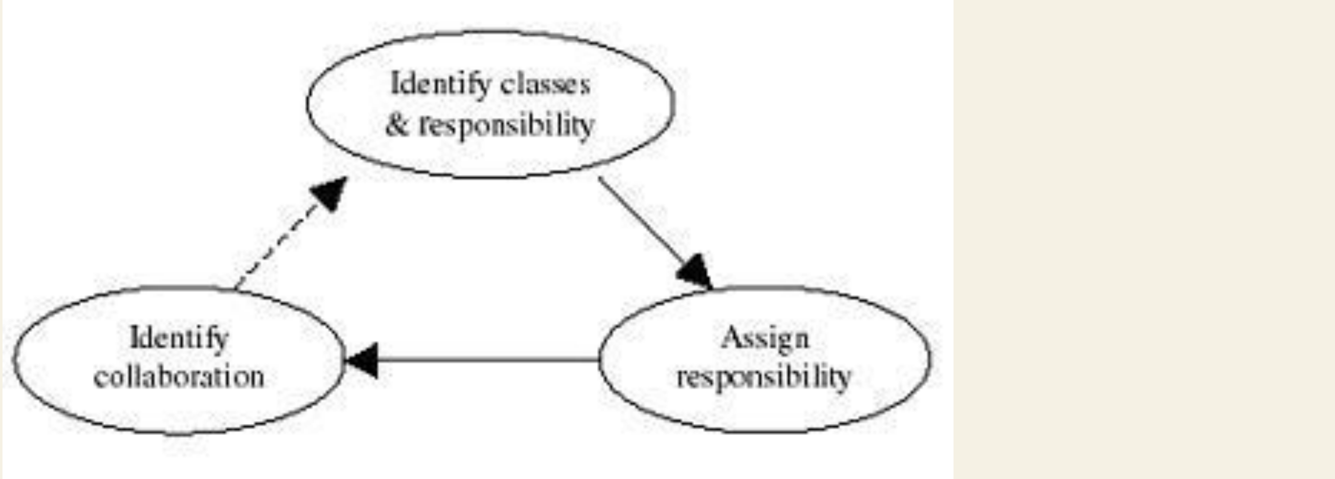
- Concept class
- Events class
- Organization class
- People class
- Places class
- Tangible things and devices class

The classes, responsibilities and collaborators (CRC) approach.
Classes, responsibilities, and collaborators is a technique used for identifying classes' responsibilities, and collaborators and therefore their attributes and methods. CRC is based on the idea that an object either can accomplish a certain responsibility itself or it may require the assistance of otherobjects.

Steps in CRC process

The classes, responsibilities and collaborators process consist of three steps:

- Identify classes' responsibilities (and identify classes).
- Assign responsibilities.
- Identify collaborators.



Video Content / Details of website for further learning (if any):
https://www.coursehero.com/file/13903479/Ch4-OOAD-Classification/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:151-161)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 23

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **III - Object Oriented Analysis** **Date of Lecture: 20.03.2021**

Topic of Lecture: Identifying object, relationships, attributes, methods, Super-sub class

Introduction: A meta class is used in object-oriented programming (OOP) and is typically thought of as a classes' class. A meta class is able to modify information from the class and can be linked to one or many classes, depending on the coding structure.

Prerequisite knowledge for Complete understanding and learning of Topic

- Class
- Object Oriented Concepts
- Link
- Interaction

Detailed content of the Lecture:

Identifying object relationships, attributes and methods

- Identifying associations begins by analyzing the interactions between classes.
- Three types of relationships among objects are
- Association.
  - How are objects associated? This information will guide us in designing classes.
- Super-sub structure (also known as generalization hierarchy).
  - How are objects organized into superciasses and subclasses?
  - This information provides us the direction of inheritance.
- Aggregation and a-part-of structure.
  - What is the composition of complex classes?
  - This information guides us in defining mechanisms that properly manage object within-object.
- Generally speaking, the relationships among objects are known as associations.
- Association represents a physical or conceptual connection between two or more objects.
- For example, if an object has the responsibility for telling another object that a credit card

number is valid or invalid, the two classes have an association.

- Binary associations are shown as lines connecting two class symbols
- Ternary and higher-order associations are shown as diamonds connecting to a class symbol by lines, and the association name is written above or below the line.
- The association name can be omitted if the relationship is obvious.

Guidelines for Identifying Association

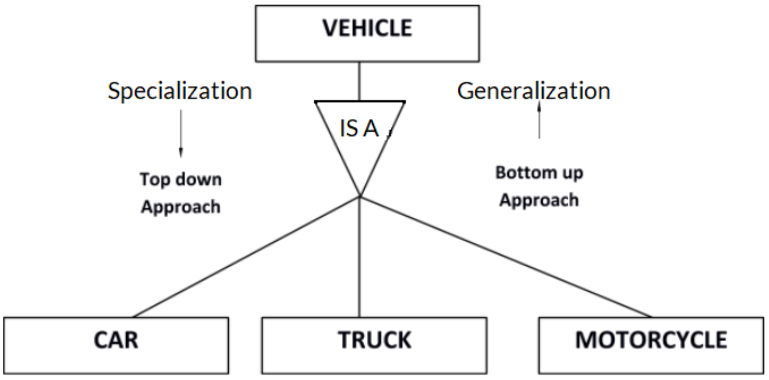
- The following are general guidelines for identifying the tentative associations:
  - A dependency between two or more classes may be an association.
  - Association often corresponds to a verb or prepositional phrase, such as part of, next to, works for, or contained in.
  - A reference from one class to another is an association.
  - Some associations are implicit or taken from general knowledge.

Super classes

A superclass is the class from which many subclasses can be created. The subclasses inherit the characteristics of a superclass. The superclass is also known as the parent class or base class. In the above example, Vehicle is the Superclass and its subclasses are Car, Truck and Motorcycle.

Sub classes

A subclass is a class derived from the superclass. It inherits the properties of the superclass and also contains attributes of its own. An example is:



Video Content / Details of website for further learning (if any):
https://slideplayer.com/slide/12696768/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:177-181)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 24

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **III - Object Oriented Analysis** **Date of Lecture: 22.03.2021**

Topic of Lecture: A part of relationships Identifying attributes and methods

Introduction: In UML modeling, a relationship is a connection between two or more UML model elements that adds semantic information to a model. In the product, you can use several UML relationships to define the structure between model elements. Examples of relationships include associations, dependencies, generalizations, realizations, and transitions.

Prerequisite knowledge for Complete understanding and learning of Topic

- Attributes
- Methods
- Relationships
- Class
- Object

Detailed content of the Lecture:

A-PART-OF RELATIONSHIPS—AGGREGATION

- Apart-of relationship, also called aggregation, represents the situation where a class consists of several component classes.
- A class that is composed of other classes does not behave like its parts; actually, it behaves very differently.
- For example, a car consists-of many other classes, one of which is a radio, but a car does not behave like a radio
- Two major properties of a-part-of relationship are transitivity and antisymmetric:
  - Transitivity. The property where, if A is part of B and B is part of C, then A is part of C.
  - Antisymmetty. The property of a-part-of relation where, if A is part of B, then B is not part of A.

A-Part-of Relationship Patterns

To identify a-part-of structures, Coad and Yourdon provide the following guidelines:

In the product, you can use several UML relationships to define the structure between model elements. Examples of relationships include associations, dependencies, generalizations, realizations, and transitions.

- **Assembly**
    - An assembly is constructed from its parts and an assembly-part situation physically exists;
    - For example, a French onion soup is an assembly of onion, butter, flour, wine, French bread, cheddar cheese, and so on.
- **Container**
    - A physical whole encompasses but is not constructed from physical parts; for example, a house can be considered as a container for furniture and appliances.
- **Collection-member**
    - A conceptual whole encompasses parts that may be physical or conceptual; for example, a football team is a collection of players.

Video Content / Details of website for further learning (if any):
https://www.ibm.com/docs/en/rational-soft arch/9.5?topic=SS8PJ7_9.5.0/com.ibm.xtools.modeler.doc/topics/rreltyp.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:182-183)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 25

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **III - Object Oriented Analysis** **Date of Lecture: 23.03.2021**

Topic of Lecture: Object responsibility

Introduction: An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain.

Prerequisite knowledge for Complete understanding and learning of Topic

- Object
- Classes
- Methods
- Response

Detailed content of the Lecture:

One of the most important mantras used in object-oriented design is that of “an object must be responsible for itself “. I first heard this phrase while taking my first class in object-oriented programming class in Smalltalk. Because I had never programmed in an object-oriented language before, I was curious as to what this really meant.

The translation of responsibilities into classes and methods is influenced by the granularity of the responsibility. Big responsibilities take hundreds of classes and methods. Little responsibilities might take one method. For example, the responsibility to "provide access to relational databases" may involve two hundred classes and thousands of methods, packaged in a subsystem. By contrast, the responsibility to "create a Sale" may involve only one method in one class.

A responsibility is not the same thing as a method - it's an abstraction - but methods fulfill responsibilities. RDD also includes the idea of collaboration. Responsibilities are implemented by means of methods that either act alone or collaborate with other methods and objects.

Responsibility-driven design is a design technique in object-oriented programming, which improves encapsulation by using the client–server model. It focuses on the contract by considering the actions that the object is responsible for and the information that the object shares. It was proposed by Rebecca Wirfs-Brock and Brian Wilkerson.

Responsibility-driven design is in direct contrast with data-driven design, which promotes defining the behavior of a class along with the data that it holds. Data-driven design is not the same as data-driven programming, which is concerned with using data to determine the control flow, not class design.

Responsibilities and Methods

The UML defines a responsibility as “a contract or obligation of a classifier”. Responsibilities are related to the obligations of an object in terms of its behaviour.

Doing responsibilities of an object include:

- Doing something itself, such as creating an object or doing a calculation
- Initiating action in other objects
- Controlling and coordinating activities in other objects

Responsibilities are assigned to classes of objects during object design

A straightforward definition for object-responsibility is this: An object must contain the data (attributes) and code (methods) necessary to perform any and all services that are required by the object. This means that the object must have the capability to perform required services itself or at least know how to find and invoke these services. Rather than attempt to further refine the definition, take a look at an example that illustrates this responsibility concept

Video Content / Details of website for further learning (if any):
https://en.wikipedia.org/wiki/Responsibility-driven_design

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:191-192)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 26

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **III - Object Oriented Analysis** **Date of Lecture:24.03.2021**

Topic of Lecture: Construction of class diagram for generalization

Introduction: In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent). Generalization. In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an "is – a – kind – of" relationship.

Prerequisite knowledge for Complete understanding and learning of Topic

- Generalization
- Class Hierarchy
- Relationship
- Link

Detailed content of the Lecture:

Generalization and Specialization

Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes.

Generalization

In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an "is – a – kind – of" relationship. For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle".

Specialization

Specialization is the reverse process of generalization. Here, the distinguishing features of groups of objects are used to form specialized classes from existing classes. It can be said that the subclasses are the specialized versions of the super-class.

Links and Association

Link

A link represents a connection through which an object collaborates with other objects. Rumbaugh has defined it as “a physical or conceptual connection between objects”. Through a link, one object may invoke the methods or navigate through another object. A link depicts the relationship between two or more objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association.

Degree of an Association

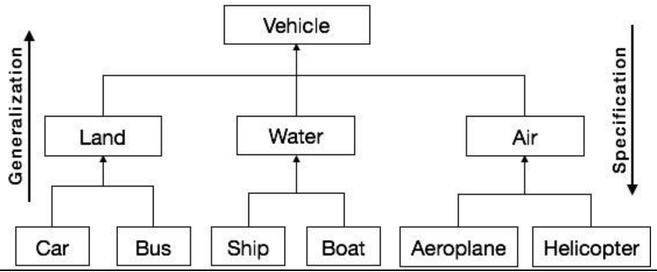
Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A unary relationship connects objects of the same class.
- A binary relationship connects objects of two classes.
- A ternary relationship connects objects of three or more classes.

Cardinality Ratios of Associations

Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely −

- One–to–One − A single object of class A is associated with a single object of class B.
- One–to–Many − A single object of class A is associated with many objects of class B.
- Many–to–Many − An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.



Video Content / Details of website for further learning (if any):
https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_model.htm

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:184-187)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 27

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **III - Object Oriented Analysis** **Date of Lecture: 25.03.2021**

Topic of Lecture: Aggregation, Example, Vehicle class.

Introduction: Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a "part–of" or "has–a" relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Prerequisite knowledge for Complete understanding and learning of Topic

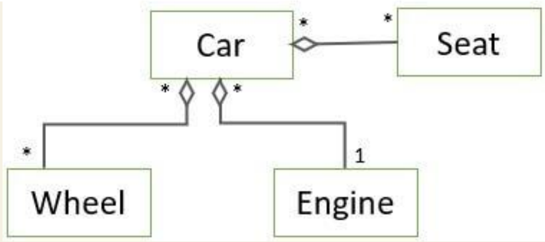
- Dependencies between Objects
- Relationship
- Classes & Objects
- Link

Detailed content of the Lecture:

Aggregation in object orientation is a kind of association which relates two objects by 'part-of whole' relationship that's why it is possible only in a binary association. Aggregation is a 'has-a' relationship between two class like a car has a wheel, an engine has a gearbox and so on.

Aggregation is a kind of association that is used to establish a relationship between the assembly class and one constituent part class. Aggregation is a binary association where one end of the association is aggregate and the other end is constituent.

Aggregation is the 'has-a' relationship. An aggregate object is derived from the several lesser constituent parts where each constituent is a part of the aggregation. We can refer an aggregate object as an extended object which is operated as a single unit though it is derived from the several constituent objects. The aggregation is denoted by the straight line with the empty arrow by the side of assembly class.



For a Car has wheels, an engine, seats. Here the class Car is an assembly class and the others are the constituent part classes. Each individual pairing i.e. Car to Wheel is one aggregation relation, Car to Engine is an aggregation relation and Car to Seats is an aggregation relation.

This concludes that aggregation is a binary association. Let us discuss some interesting properties of aggregation.

- Aggregation is transitive which means if A belongs to B and B belongs to C then we can say that A belongs to C. For example, the gearbox is a part of engine and engine is a part of car then gearbox is the part of the car.
- Aggregation is antisymmetric i.e. when we say that A is part of B it doesn't mean that b is also a part of A. For example, a car has an engine but we cannot say that engine will also behave likeacar.It can also be put up like this, aggregation is unidirectional which means only one end of the association can be marked as aggregation.

Aggregation is used to express the 'Part-of'/'has-a' relationship. That is, if you would use the words "part-of" or "has-a" to describe the relationship between two classes than that relationship would be described as an aggregation. For example, a "wheel is part of a car" and "a car has an engine". Essentially, aggregation is a type of association whose relationship role could be described using either of the terms "part-of" or "has-a".

Managing complex attributes is typically achieved by the use the aggregation or "whole/part" relationships, decomposing the parts of a complex class and providing ownership of these by the whole. In UML models, an aggregation relationship shows a classifier as a part of or subordinate to another classifier.

An aggregation is a special type of association in which objects are assembled or configured together to create a more complex object. An aggregation describes a group of objects and how you interact with them. Aggregation protects the integrity of an assembly of objects by defining a single point of control, called the aggregate, in the object that represents the assembly. Aggregation also uses the control object to decide how the assembled objects respond to changes or instructions that might affect the collection.

Video Content / Details of website for further learning (if any):
https://binaryterms.com/aggregation-in-object-orientation.html

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:187-189)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 28

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV Object Oriented Design** **Date of Lecture: 26.03.2021**

Topic of Lecture: Design process and benchmarking

Introduction: Benchmarking is a process used to measure the quality and performance of your company's products, services, and processes. These measurements don't have much value on their own—that data needs to be compared against some sort of standard.

Prerequisite knowledge for Complete understanding and learning of Topic

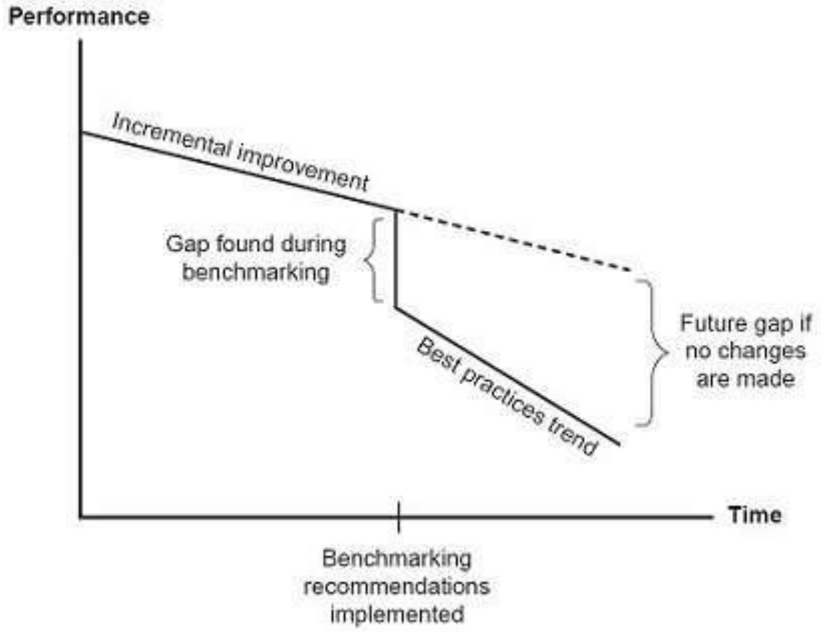
- Standards
- Quality
- Processes
- Tool
- Services

Detailed content of the Lecture:

Benchmarking is defined as the process of measuring products, services, and processes against those of organizations known to be leaders in one or more aspects of their operations. Benchmarking provides necessary insights to help you understand how your organization compares with similar organizations, even if they are in a different business or have a different group of customers. Benchmarking can also help organizations identify areas, systems, or processes for improvements—either incremental (continuous) improvements or dramatic (business process re-engineering) improvements.

Benchmarking as a Tool

Benchmarking is a process for obtaining a measure – a benchmark. Simply stated, benchmarks are the "what," and benchmarking is the "how." But benchmarking is not a quick or simple process tool. Before undertaking a benchmarking opportunity, it is important to have a thorough understanding of the company's guidelines. Some companies have strict guidelines as to what information can be gathered, and whom practitioners can contact to get that information. Depending on the size of the company, practitioners may be surprised at what is readily available in-house.


Performance
Incremental improvement
Gap found during benchmarking
Future gap if no changes are made
Best practices trend
Time
Benchmarking recommendations implemented

8 steps in the benchmarking process

- Select a subject to benchmark. ...
- Decide which organizations or companies you want to benchmark. ...
- Document your current processes. ...
- Collect and analyse data. ...
- Measure your performance against the data you've collected. ...
- Create a plan. ...
- Implement the changes. ...
- Repeat the process.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:199-200)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 29

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV - Object Oriented Design** **Date of Lecture: 27.03.2021**

Topic of Lecture: Axioms, Corollaries

Introduction: An axiom is a fundamental truth that always is observed to be valid and for which there is no counterexample or exception. The axioms cannot be proven or derived but they cannot be invalidated by counterexamples or exceptions. A corollary is a proposition that follows from an axiom or another proposition that has been proven.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Design
- Model
- Relationship
- Rules
- Object

Detailed content of the Lecture:

Axiom is a fundamental truth that is always observed to be valid and for which there is no counterexample or exception. They cannot be proven or derived. A theorem is a preposition that may not be self-evident but can be proven by accepted axioms. It is a law or principle. Theorem is valid if its referent axioms and deductive steps are valid.

There are two axioms followed.

Axiom 1: The independence Axiom. (Relationships between components) Maintain the independence of components The independence Axiom This Axiom states that, during the design process, as we go from requirements to a system component, each component must satisfy that requirement without affecting other requirements E.g.., designing a refrigerator door with two requirements: Door should provide access to the foodMinimal loss of energy on opening and closing it (opening the door is independent of loss of energy)Requirements are coupledVertical door – coupledThis can be designed by providing horizontally as like chest-type freezers. Which satisfies both the requirements, without violating the first axiom

Axiom 2: The Information Axiom. (complexity of design) Minimize the information content of the design. The Information Axiom. This Axiom is concerned with simplicityEach fact should be with a minimum amount of complexity and maximum simplicity and straightforwardness.Minimal complexity produces the most easily maintained and enhance applicationOOS- to use inheritance, system's built in classes to minimize complexity

COROLLARIES

A Corollary is a proposition that follows from an axiom or another proposition that has been proven Can be valid or invalid as theorems. They are also called as design rules. Derived from design axioms Useful in making design decisions, since they are applied to actual situations more easily than original axioms

From the two axioms, the following corollaries are formed Uncoupled design with less information content Highly cohesive objects can improve coupling because only a minimal amount of information need to be passed between objects Single purpose Each Class must have a single, clearly defined purpose Large number of simple classes Allows reusability Strong mapping Strong association between the analysis object and design object Standardization Promote standardization by designing interchangeable components and reusing existing classes or components Design with inheritance

Corollary 1: Uncoupled design with less information content The main goal here is to maximize objects cohesiveness among objects and software components in-order to improve coupling. This is the measure of strength of association established by a connection from one object or software component to another Change to one component of the system should have minimal impact on the other system.

Strong coupling among the objects complicates the system. The degree of coupling is a function of How complicated the connection is Whether the connection refers to the object itself or something inside it What is being sent or received OOD – Interaction and Inheritance coupling Interaction Coupling involves the amount and complexity of the messages between the components Hence its better to keep the messages simple Also reduce the number of messages sent & received by an object(infrequent as possible)Inheritance coupling: coupling between super and sub classes A subclass is coupled with its super class in terms of attributes and methods High inheritance coupling is desirable If the sub class is overriding all of the methods (low inheritance coupling)

Types of coupling among different objects or components

Content coupling – degree is very high

Common coupling – degree is high

Control coupling – degree is medium

Stamp coupling – degree is low

Data coupling – degree is very low

Video Content / Details of website for further learning (if any):
https://slideplayer.com/slide/12765789/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:202-211)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 30

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV - Object Oriented Design   Date of Lecture: 29.03.2021**

Topic of Lecture: Designing classes, Class visibility

Introduction: A **design class** represents an abstraction of one or several classes in the system's implementation, exactly what it corresponds to depends on the implementation language.

Prerequisite knowledge for Complete understanding and learning of Topic

- Class
- Object
- Access Specifier
- Software
- Operation

Detailed content of the Lecture:

The set of design classes refine analysis classes and providing design detail that enables classes to execute a software infrastructure that supports business solutions.

Operations

The only way other objects can get access to or affect the attributes or relationships of an object is through its operations. The operations of an object are defined by its class. A specific behavior can be performed via the operations, which may affect the attributes and relationships the object holds and cause other operations to be performed. An operation corresponds to a member function in C++ or to a function or procedure in Ada. What behavior you assign to an object depends on what role it has in the use-case realizations.

Parameters

In the specification of an operation, the parameters constitute formal parameters. Each parameter has a name and type. You can use the implementation language syntax and semantics to specify the operations and their parameters so that they will already be specified in the implementation language when coding starts.

Class Operations

An operation nearly always denotes object behavior. An operation can also denote behavior of a class, in which case it is a class operation. This can be modeled in the UML by type-scoping the operation.

Operation Visibility

The following visibilities are possible on an operation:

- Public: the operation is visible to model elements other than the class itself.
- Protected: the operation is visible only to the class itself, to its subclasses, or to friends of the class (language dependent)
- Private: the operation is only visible to the class itself and to friends of the class
- Implementation: the operation is visible only within to the class itself.

Implementation visibility is the most restrictive; it is used in cases where only the class itself is able to use the operation. It is a variant of Private visibility, which for most cases is suitable.

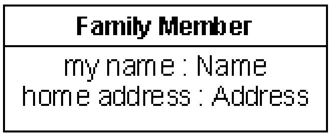
States

An object can react differently to a specific message depending on what state it is in; the state-dependent behavior of an object is defined by an associated state chart diagram. For each state the object can enter, the state chart diagram describes what messages it can receive, what operations will be carried out, and what state the object will be in thereafter. Refer to Guidelines: State chart Diagram for more information.

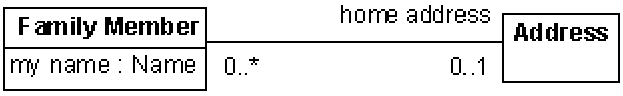
Attributes

An attribute is a named property of an object. The attribute name is a noun that describes the attribute's role in relation to the object. An attribute can have an initial value when the object is created.

Example:



An example of how an attribute is modeled. Each member of a family has a name and an address. Here, we have identified the attributes my name and home address of type Name and Address, respectively:



Video Content / Details of website for further learning (if any):
https://www.slideshare.net/gopal10scs185/unit-4-designing-classes

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:217-221)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 31

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : IV - Object Oriented Design **Date of Lecture: 30.03.2021**

Topic of Lecture: Refining attributes, Methods and protocols

Introduction: Static modeling is used to specify the structure of the objects, classes or components that exist in the problem domain. These are expressed using class, object or component. While dynamic modeling refers to representing the object interactions during runtime.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Object
- Classes
- Attributes
- CASE Tools
- Methods

Detailed content of the Lecture:

Designing classes

The three basic types of attributes are

1. single-value attributes.
2. Multiplicity or multivalued attributes.
3. Reference to another object, or instance connection.

Designing Methods and Protocols:

Once We have designed your methods in some formal structure such as UML activity diagrams with an OCL - object constraint language description, they can be converted to programming language manually or in automated fashion

A class can provide several types of Methods: -

1. **Constructor:** Method that creates instances (objects) of the class.
2. **Destructor**: The Method that destroys instances.
3. **Conversion method**: The method that converts a value from one unit of measure to another.
4. **Copy method**: The method that copies the contents of one instance to another instance.
5. **Attribute set**: The method that sets the values of one or more attributes.
6**. Attribute get**: The method that returns the values of one or more attributes.
7. **I/O methods**: The methods that provide or receive data to or from a device.

Private and Protected Protocol Layers: Internal

- Items in these layers define the implementation of the object.
- Apply the design axioms and corollaries, especially Corollary 1 (uncoupled design with less information content) to decide what should be privates what attributes (instance variables)?
- What methods? Remember, highly cohesive objects can improve coupling because only a minimal amount of essential information need be passed between objects.
  Public Protocol Layer: External
- Items in this layer define the functionality of the object.
- Here are some things to keep in mind when designing class protocols:
  - Good design allows for polymorphism.
  - Not all protocol should be public; again, apply design axioms and corollaries.
- The following key questions must be answered:
  - What are the class interfaces and protocols?
  - What public (external) protocol will be used or what external messages must the system understand?
  - What private or protected (internal) protocol will be used or what internal messages or messages from a subclass must the system understand?

DESIGNING CLASSES: REFINING ATTRIBUTES

- The main goal of this activity is to refine existing attributes (identified in analysis) or add attributes that can elevate the system into implementation.
- Attribute Types
  - The three basic types of attributes are
    - Single-value attributes.
    - Multiplicity or multivalue attributes.
    - Reference to another object, or instance connection.

Attribute Types

- Attributes represent the state of an object.
- When the state of the object changes, these changes are reflected in the value of attributes.
- The single-value attribute is the most common attribute type.
- It has only one value or state.
- For example, attributes such as name, address, or salary are of the single-value type
- The multiplicity or multivalue attribute is the opposite of the single-value attribute since, as its name implies, it can have a collection of many values at any point in time.
- For example, if we want to keep track of the names of people who have called a customer support line for help, we must use the multivalues attributes.

Video Content / Details of website for further learning (if any):
https://gacbe.ac.in/pdf/ematerial/18MIT13C-U4.pdf

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:221-230)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 32

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV Object Oriented Design** **Date of Lecture: 31.03.2021**

Topic of Lecture: Object storage and object interoperability

Introduction: Object storage, also known as object-based storage, is a strategy that manages and manipulates data storage as distinct units, called objects. These objects are kept in a single storehouse and are not ingrained in files inside other folders.

Prerequisite knowledge for Complete understanding and learning of Topic

- Object
- Classes
- Attributes
- CASE Tools
- Methods

Detailed content of the Lecture:

Object storage takes each piece of data and designates it as an object. Data is kept in separate storehouses versus files in folders and is bundled with associated metadata and a unique identifier to form a storage pool.

File storage stores data as a single piece of information in a folder to help organize it among other data. This is also called hierarchical storage, imitating the way that paper files are stored.

Object storage is a technology that manages data as objects. All data is stored in one large repository which may be distributed across multiple physical storage devices, instead of being divided into files or folders.

It is easier to understand object-based storage when you compare it to more traditional forms of storage – file and block storage.

Interoperability is one of the key aspects related to the construction of large object-oriented systems, and can be defined as the ability of two or more en- tities to communicate and cooperate despite differences in the implementation language, the execution environment or the model abstraction.

There are three levels of interoperability: foundational, structural, and semantic.

- Foundational interoperability. Foundational interoperability is the ability of one IT system to send data to another IT system. ...
- Structural interoperability. ...
- Semantic Interoperability.

Data interoperability addresses the ability of systems and services that create, exchange and consume data to have clear, shared expectations for the contents, context and meaning of that data.

Interoperability is the property that facilitates unrestricted sharing and use of data or resources between disparate systems via local area networks (LANs) or wide area networks (WANs). There are two types of data interoperability - syntactic interoperability, which is a prerequisite to semantic interoperability and enables different software components to cooperate, facilitating two or more systems to communicate and exchange data; and semantic interoperability, which refers to the ability of computer systems to exchange meaningful data with unambiguous, shared meaning.

Efficient automated data sharing between applications, databases, and other computer systems is a crucial component throughout networked computerized systems, especially interoperability in healthcare information and management systems. Interoperability is a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, at present or in the future, in either implementation or access, without any restrictions. While the term was initially defined for information technology or systems engineering services to allow for information exchange, a broader definition takes into account social, political, and organizational factors that impact system-to-system performance.

Video Content / Details of website for further learning (if any):
https://gacbe.ac.in/pdf/ematerial/18MIT13C-U4.pdf

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:237-238)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 33

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV Object Oriented Design** **Date of Lecture: 01.04.2021**

Topic of Lecture: Databases, Object relational systems, Designing interface objects

Introduction: An object-oriented database (OOD) is a database system that can work with complex data objects — that is, objects that mirror those used in object-oriented programming languages.

Prerequisite knowledge for Complete understanding and learning of Topic

- Database
- Object
- Methods
- Messages
- OODBMS

Detailed content of the Lecture:

An object–relational database (ORD), or object–relational database management system (ORDBMS), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data types and methods.

The ODBMS which is an abbreviation for object-oriented database management system is the data model in which data is stored in form of objects, which are instances of classes. These classes and objects together make an object-oriented data model.

Components of Object-Oriented Data Model:

The OODBMS is based on three major components, namely: Object structure, Object classes, and Object identity.

These are explained below.

1.ObjectStructure:

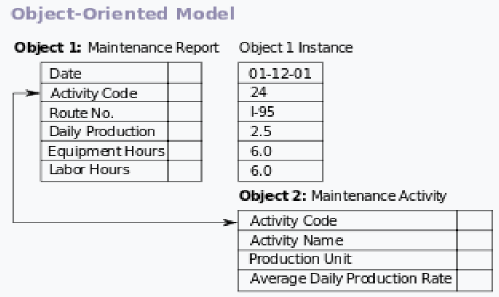
The structure of an object refers to the properties that an object is made up of. These properties of an object are referred to as an attribute. Thus, an object is a real-world entity with certain attributes that makes up the object structure. Also, an object encapsulates the data code into a single unit which in turn provides data abstraction by hiding the implementation details from the user.

The object structure is further composed of three types of components: Messages, Methods, and Variables. These are explained below.

1. **Messages** –
   A message provides an interface or acts as a communication medium between an object and the outside world. A message can be of two types:
   - Read-only message: If the invoked method does not change the value of a variable, then the invoking message is said to be a read-only message.
   - Update message: If the invoked method changes the value of a variable, then the invoking message is said to be an update message.

2. **Methods** –
   When a message is passed then the body of code that is executed is known as a method. Whenever a method is executed, it returns a value as output. A method can be of two types:
   - Read-only method: When the value of a variable is not affected by a method, then it is known as the read-only method.
   - Update-method: When the value of a variable change by a method, then it is known as an update method.

3. **Variables** –
   It stores the data of an object. The data stored in the variables makes the object distinguishable from one another.



Object 1: Maintenance Report	Object 1 Instance
Date	01-12-01
Activity Code	24
Route No.	I-95
Daily Production	2.5
Equipment Hours	6.0
Labor Hours	6.0

Object 2: Maintenance Activity
Activity Code
Activity Name
Production Unit
Average Daily Production Rate

Example of an object-oriented database model

An object–relational database can be said to provide a middle ground between relational databases and object-oriented databases. In object–relational databases, the approach is essentially that of relational databases: the data resides in the database and is manipulated collectively with queries in a query language; at the other extreme are OODBMSes in which the database is essentially a persistent object store for software written in an object-oriented programming language, with a programming API for storing and retrieving objects, and little or no specific support for querying.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

Important Books/Journals for further learning including the page nos.:
Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:239-248)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 34

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV - Object Oiented Design** **Date of Lecture: 03.04.2021**

Topic of Lecture: Macro and Micro level processes

Introduction: Software Development Macro Process (SDMaP): this is the overall software development lifecycle. In this process model we define our requirements, we execute analysis, design, implementation, test and we deploy the software into production.

Prerequisite knowledge for Complete understanding and learning of Topic

- Software
- Process Model
- Object
- UML Diagram
- Methods

Detailed content of the Lecture:

The macro development process consists of the following steps:

Conceptualization
Establish the core requirements and develop a prototype.

Analysis and development of the model
Use the class diagram to describe the roles and responsibilities of objects. Use the object diagram to describe the desired behavior of the system.

Design or create the system architecture.
Use the class diagram to decide what classes exist and how they relate to each other, the object diagram to decide what mechanisms are used, the module diagram to map out where each class and object should be declared, and the process diagram to determine to which processor to allocate a process.

Evolution or implementation-
Refine the system through much iteration.

Maintenance-Make localized changes to the system to add new requirements and eliminate bugs.

Micro Development process

The micro process is a description of the day-to-day activities by a single or small group of software developers.

It consists of the following steps.

- Identify classes and objects.
- Identify class and object semantics.
- Identify class and object relationships.
- Identify class and object interface and implementation.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:281-292)

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 35

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV - Object Oriented Design** **Date of Lecture: 05.04.2021**

Topic of Lecture: The purpose of a view layer interface, OOUI

Introduction : An object-oriented user interface (OOUI) is a type of user interface based on an object-oriented programming metaphor. In an OOUI, the user interacts explicitly with objects that represent entities in the domain that the application is concerned with.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Object
- Database
- Interface
- Design
- Process

Detailed content of the Lecture:

Designing view layer classes

UI layer consists of objects with which user interacts.
Objects needed to manage or control UI
Responsibility of view layer objects
Input: responding to user interaction
Translating user’s action into an appropriate response
Output: displaying or printing business objects

Process of designing view layer classes

It has 4 major activities –

- Macro level UI design process: (identifying view layer object)
- Takes place during analysis phase
- Identifying classes that interact with human actors by analyzing use cases
- sequence and collaboration diagram help to identify UI classes

Micro level UI design activities

• Designing the view layer objects by applying design axioms and corollaries
Decide how to use and extend the components so they best support application specific functions and provide the most usable interface
• Prototyping the view layer interface – Useful in the early design process – Testing usability and user satisfaction – Refining and iterating the design

Macro level design process

The class interacts with a human actor Zoom in by utilizing sequence or collaboration diagrams Identify the interface NeXT class objects for the class refine and iterate. The class Define the relationships doesn't among the view objects interacts with the human actor

Design rules

Making the interface simple (application of corollary 2) – Ex: • Car engine is complex • But driver interface remains simple – User must be able to work with our screen without asking much questions – Factors affecting UI • Deadlines • Comparative evaluations • Addition features • Shortcuts – Things to be considered while designing UI • Additional features affect performance, complexity • Fixing problem, after the release of product is difficult

OOUIs are becoming more and more popular, thanks in part to their versatility. Traditional user interfaces often have limited functions, which restricts their utility for various applications. Whether it's a commercial or consumer-based application, OOUIs allow for greater customization along with a higher level of utility. Using the example cited above, vector drawing applications can be used to create and design a wide range of illustrations and that's just one of the many uses for an OOUI. Traditional user interfaces, on the other hand, have limited functions and utility, restricting the user from manipulating the objects.

Video Content / Details of website for further learning (if any):
https://www.nelson-miller.com/what-is-an-object-oriented-user-interface/

Important Books/Journals for further learning including the page nos.:

Ali Bahrami, Object Oriented System Development, McGraw Hill International Edition, 2008(Page No:292-302)

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 36

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **IV - Object Oriented Design** **Date of Lecture: 06.04.2021**

Topic of Lecture: MVC Architectural Pattern and Design, Designing the system.

Introduction: MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application.

Prerequisite knowledge for Complete understanding and learning of Topic

- Model
- Pattern
- User Interface
- Controller
- GUI

Detailed content of the Lecture:

MVC design pattern is also known as Model-View-Controller. It is a common architectural pattern that is used to design and create interfaces and the structure of an application.

This pattern divides the application into three parts that are dependent and connected to each other. These designs are used to distinguish the presentation of data from how the data is accepted from the user to the data shown. These design patterns have become common in the use of web applications and for developing GUIs.

Understanding MVC Design Pattern

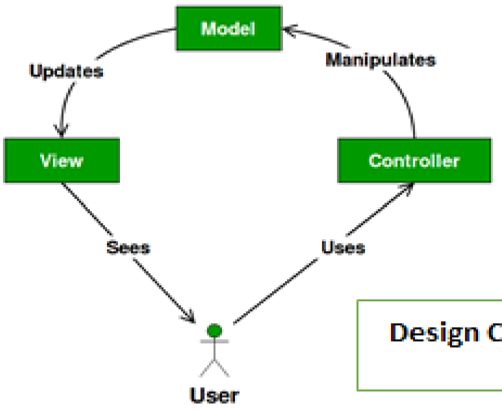
Understanding these Design patterns is easy and simple. The theory stands for Model-View-Controller Pattern.

The functions of the three parts are-

1.Model: This part of the design pattern is the primary part and contains application information purely. It doesn’t contain any information on how to show the data to the user. It is independent of the user interface. It controls the logic and rules of application.

2. View: This part helps the user to see the model’s data. The main concern of this part is to access the model’s data. The view section uses a chart, table or diagrams to represent the information. It can also show similar data and use bar graphs and tables for different purposes. It is a visualization of information that the application contains.

3. Controller: Most of the work is done by the controller. It provides the support for input and converts the input to commands for the application. It is used between the model and view part. The model and the view are interconnected, so the execution is reflected in the view part.



Design Components

The Model View Controller architectural pattern separates concerns into one of 3 buckets: Model: stores & manages data. Often a database, in our quick example we'll use local web storage on a browser to illustrate the concept. The view is a visual representation of the data- like a chart, diagram, table, form.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/mvc-design-pattern//

Important Books/Journals for further learning including the page nos.:
https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 37

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : V - Case Tools **Date of Lecture: 07.04.2021**

Topic of Lecture: Railway domain: Platform assignment system for the trains in a railway station

Introduction: The Railway Reservation System facilitates the passengers to enquire about the trains available on the basis of source and destination, Booking and Cancellation of tickets, enquire about the status of the booked ticket, etc. The aim of case study is to design and develop a database maintaining the records of different trains, train status, and passengers.

Prerequisite knowledge for Complete understanding and learning of Topic

- Object
- Records
- UML Diagrams
- Class
- Case Tools

Detailed content of the Lecture:

PROBLEM STATEMENT Our project is carried out to develop software for online Railway Reservation System. This system has various options like reservation, cancellation and to view details about available seats. Our project mainly simulates the role of a Railway ticket booking officer, in a computerized way. The reservation option enables a person to reserve for a ticket at their home itself. All he/ she has to do is to just login and enter the required details. After this the reservation database is updated with the person details, train name and also the source and destination place. The cancellation option enables the passenger to cancel the tickets that has been already booked by him/her. The availability option prompts the person to enter train number, train name and date of travel. After this the availability database is accessed and available positions are produced.

USE-CASE NAME: CHECK STATUS The passenger can view the status of the reserved tickets. So the passenger can confirm his/her travel.

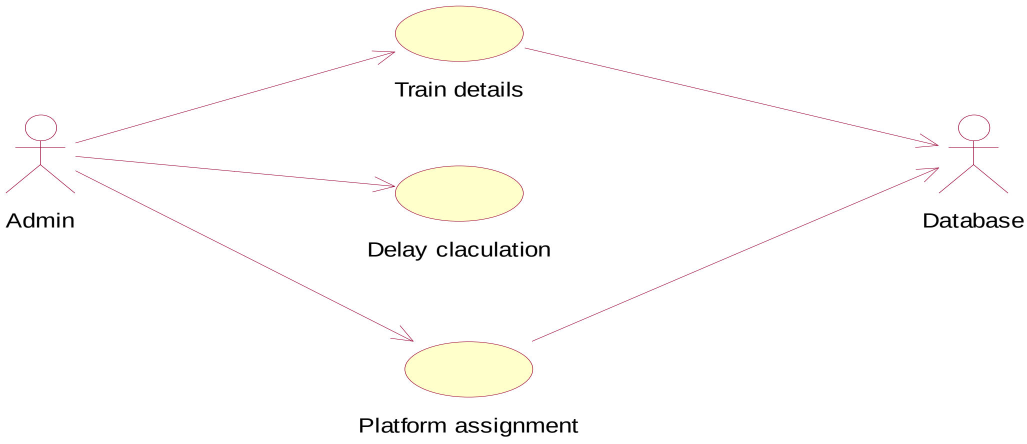
Login:
Basic flow: To authenticate the user,the admin has to enter username and password
Alternate flow: If the password is wrong, it will ask the admin to answer security question and retrieve the password
Pre-condition: The system asks the admin to enter the password
Post condition: On success,the admin displays the admin information

Train Details:
Basic flow: The train details like train no, scheduled time, expected time to arrive, scheduled time for departure, expected time to start, platform number and current number is given as input to calculate the time delay.
Alternate flow: If the train details is not correct admin can report it.
Pre-condition: The train details should be known.
Post condition: After verifying all the details, admin is moved to the next state.

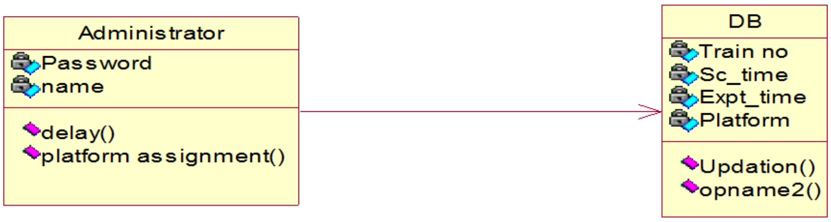
Delay calculation:
Basic flow: The entered details are verified with the database and time delay is calculated and also checks whether there is any clash with the next train timing
Alternate flow: If the details are incorrect, error message is generated.
Precondition: The details of the database should be up to date.
Postcondition: After verification the platform assignment is done.

Platform Assignment:
Basic flow: Based on the delay and arrival of the next train the platform assignment is done.
Alternate flow: If no platform is free then the train is made to wait.
Precondition: The availability of the platform should be known.
Postcondition: The platform is confirmed and the signal is given for the train to stop.

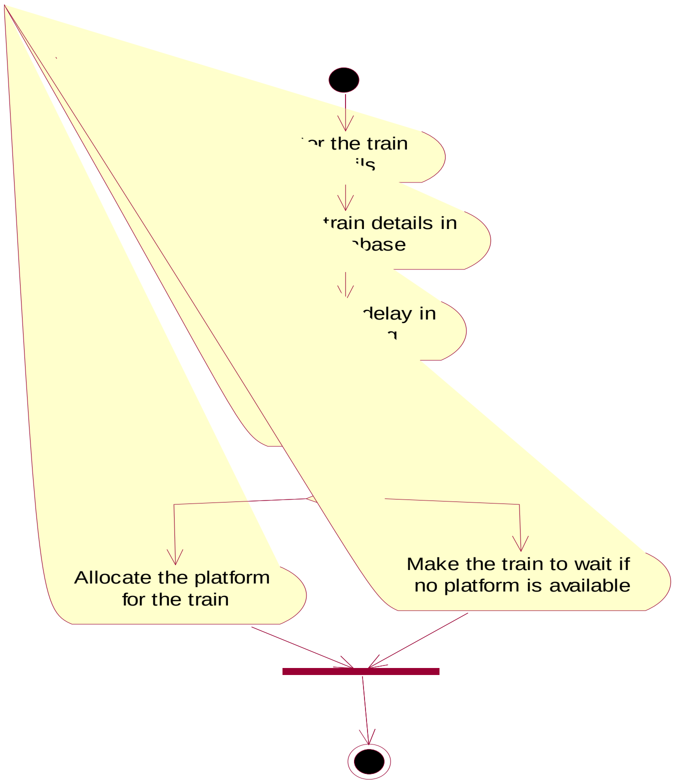
3.1 Usecase Diagram:



3.2 Class Diagram:



3.3 Activity Diagram:



Video Content / Details of website for further learning (if any):
https://www.freeprojectz.com/uml-diagram/railway-reservation-system-uml-diagram

Important Books/Journals for further learning including the page nos.:

Martin Fowler, UML Distilled A Brief Guide to Standard Object Modeling Language, 3rd Edition, Addison Wesley,2003

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 38

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : V - Case Tools **Date of Lecture: 08.04.2021**

Topic of Lecture: Academic domain: Students marks analyzing system

Introduction: The Student Mark Analysis System deals with the complete academic details of the students. It comprises of the Roll No, Name, Mark, Total and average. It can be accessed by the faculty who alone can change or update the marks of the student. It is also the duty of the faculty to maintain the records, the duty of the administrator is to generate the report cards to the faculty members. The faculty will calculate the total marks based on the percentage obtained by the student.

Prerequisite knowledge for Complete understanding and learning of Topic

- Dependencies between packages
- Visualize physical hardware and software

Detailed content of the Lecture:

Problem analysis and project planning

1.1 Introduction Student mark analyzing system has been designed to carry out the mark analysis process in an educational institution. The results of respective departments can be efficiently computed without much of manual involvement.

1.2 Objectives The purpose of this document is to define the requirements of mark analysis system. This system reduces manual work to great extent. The mark analysis is carried out by the system in an efficient manner.

1.3 Scope This system is very essential for every educational institution as it reduces man power. This system can be used for all kinds of educational institutions to evaluate and analyze the marks and generate reports of specified criteria.

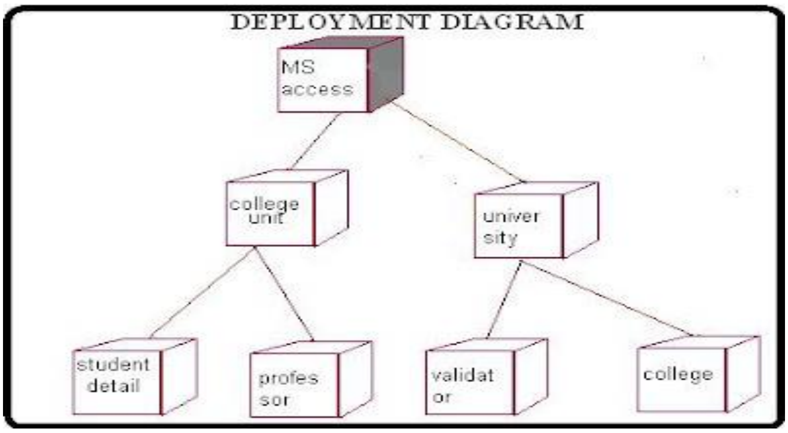
1.4 Problem Statement For analyzing the marks obtained by students in an educational institution. We are tasked to build up student mark analyzing system. This is done to replace the manual entering and processing of marks which are error prone and tedious. This system also maintains information about student. The system will have a Windows based desktop interface to allow the faculty to enter marks obtained by the students, update them and generate various reports.

For security reasons, the administrator and faculty only can update the marks and other information. First the user needs to login to the system for accessing it. The system will retain information on all the students and the institution. The system analyses the marks and generate

Ex.No: STUDENT MARK ANALYZING SYSTEM Date: the result reports. The marks and information about the students are stored in a database and the system works with the database. The faculty can enter the marks and student information through a visual environment.

One of the most important features of the system is creating reports based on the given criteria. The user can create the following reports: Overall Class, Department result, Individual student result, Topper's list, Arrear's list and Improvement rate for the academic year report has to be generated by entering the register number of the student. These reports can also be viewed by the management and placement officers.

The administrator is responsible for adding, deleting student details form the system and updating the marks to the system with the external queries. So, the system design will generate reports automatically and there will be no need for manual intervention.



IDENTIFICATION OF ACTORS AND USECASE

ACTORS:

1. Student
2. Administrator

USECASE:

1. student detail
2. student mark
3. view student list
4. department detail
5. calculate Tot, Ave ,Gra, Rem

Video Content / Details of website for further learning (if any):
https://www.studentcpu.com/2009/08/student-mark-analysis-actors-use-case.html

Important Books/Journals for further learning including the page nos.:
https://www.sourcecodesolutions.in/2010/12/student-marks-analysis-system-case.html

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 39

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **V - Case Tools** **Date of Lecture: 09.04.2021**

Topic of Lecture: ATM system

Introduction: An automated teller machine (**ATM**) or the automatic banking machine (**ABM**) is a banking subsystem (subject) that provides bank customers with access to financial transactions in a public space without the need for a cashier, clerk, or bank teller.

Prerequisite knowledge for Complete understanding and learning of Topic

- Dependencies between packages
- Visualize physical hardware and software
- Use cases
- Generalization
- Abstraction

Detailed content of the Lecture:

An automated teller machine (**ATM**) or the automatic banking machine (**ABM**) is a banking subsystem (**subject**) that provides bank customers with access to financial transactions in a public space without the need for a cashier, clerk, or bank teller.

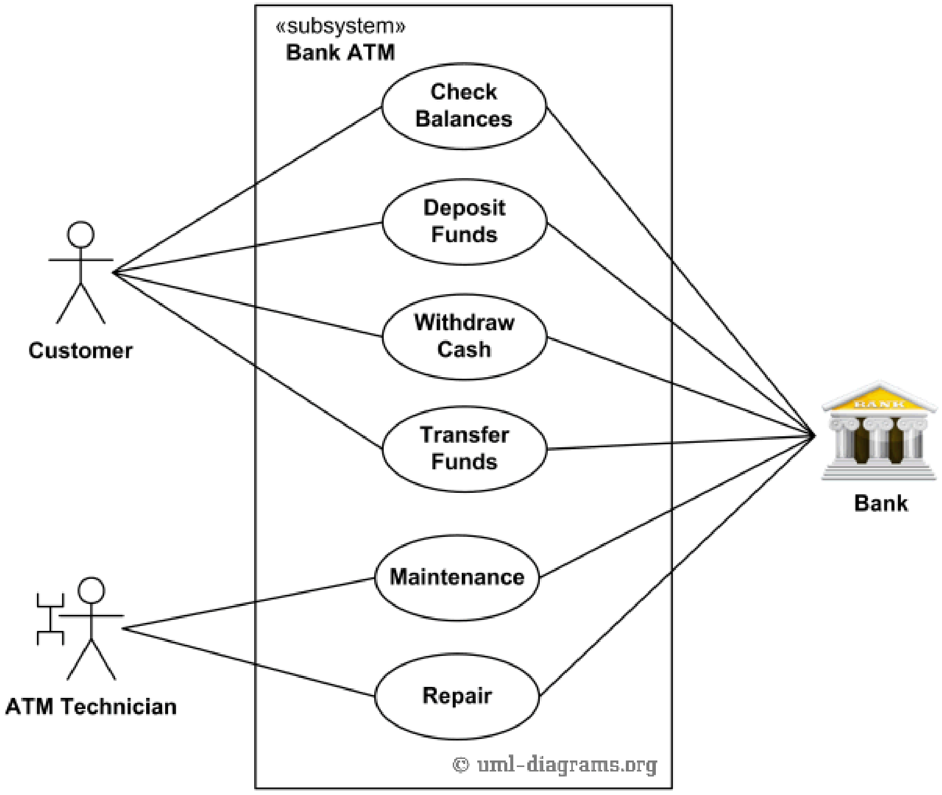
Customer (actor) uses bank ATM to Check Balances of his/her bank accounts, Deposit Funds, Withdraw Cash and/or Transfer Funds (use cases).

ATM Technician provides Maintenance and Repairs. All these use cases also involve Bank actor whether it is related to customer transactions or to the ATM servicing.

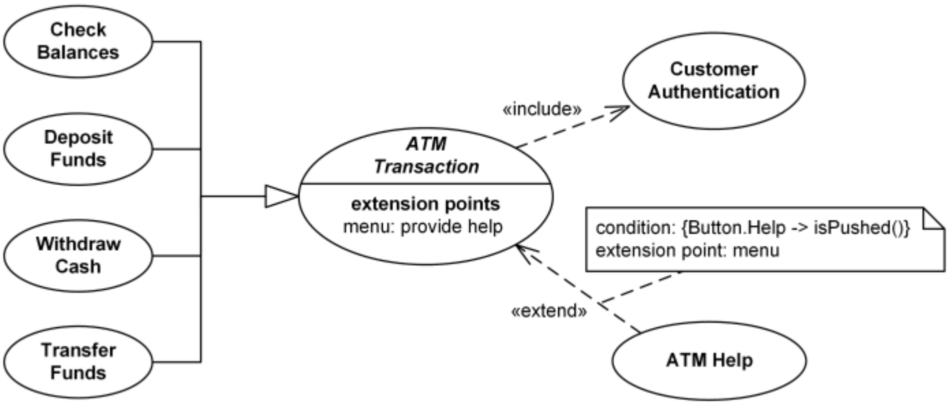
On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). Customer Authentication use case is required for every ATM transaction so we show it as include relationship.

Including this use case as well as transaction generalizations make the *ATM Transaction* an abstract use case. On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN).

Customer Authentication use case is required for every ATM transaction so we show it as include relationship. Including this use case as well as transaction generalizations make the ATM Transaction an abstract use case. On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN).



On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). Customer Authentication use case is required for every ATM transaction so we show it as include relationship. Including this use case as well as transaction generalizations make the ATM Transaction an abstract use case.



Video Content / Details of website for further learning (if any):
https://www.uml-diagrams.org/bank-atm-uml-use-case-diagram-example.html

Important Books/Journals for further learning including the page nos.:
https://www.startertutorials.com/uml/uml-diagrams-atm-application.html

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 40

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **V - Case Tools** **Date of Lecture: 10.04.2021**

Topic of Lecture: Stock maintenance

Introduction: The stock maintenance system must take care of sales information of the company and must analyze the potential of the trade. It maintains the number of items that are added or removed. The sales person initiates this Use case. The sales person is allowed to update information and view the database.

Prerequisite knowledge for Complete understanding and learning of Topic

- Software
- Analysis
- Use case
- Classes
- Interaction Diagram (UML Diagram)

Detailed content of the Lecture:

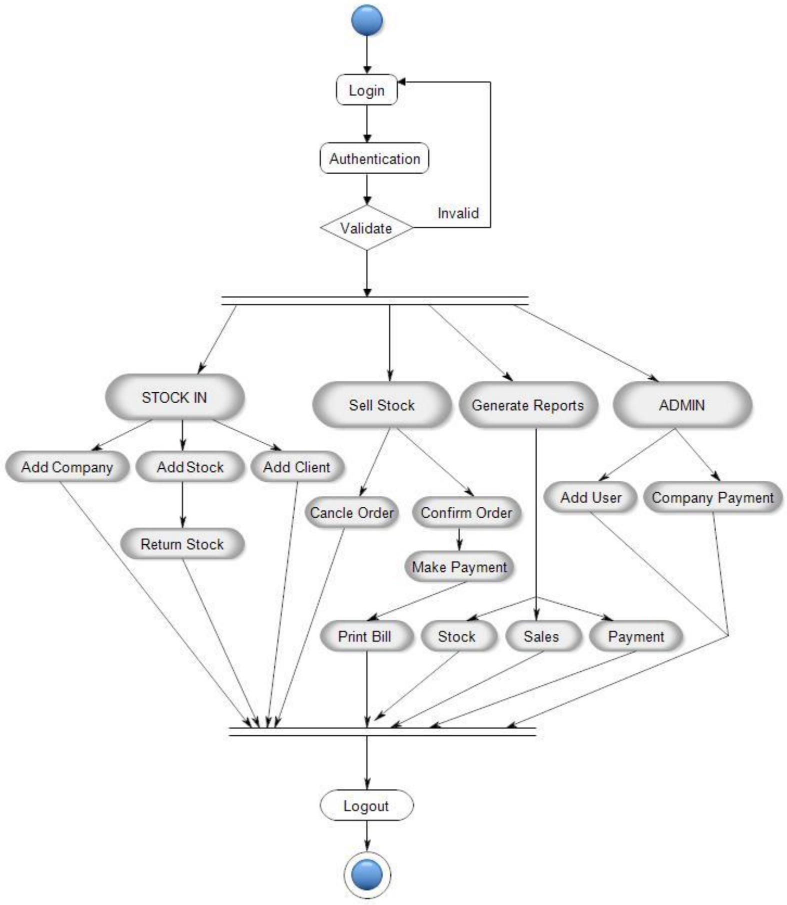
The stock maintenance system must take care of sales information of the company and must analyze the potential of the trade. It maintains the number of items that are added or removed. The sales person initiates this Use case. The sales person is allowed to update information and view the database. Stock maintenance is an interface between the customer and the sales person. It aims at improving the efficiency in maintaining the stocks. The entire process of Stock maintenance is done in a manual manner Considering the fact that the number of customers for purchase is increasing every year, a maintenance system is essential to meet the demand. So, this system uses several programming and database techniques to elucidate the work involved in this process.

The System provides an interface to the customer where they can fill in orders for the item needed. The sales person is concerned with the issue of items and can use this system. Provide a communication platform between the customer and the sales person.

- **Market Data provider**: One who analyze the product and distribute the news.
- **Customer:** One who takes order of product
- **Sales person:** One who maintains the stock details

The activity diagram used to describe flow of activity through a series of actions. Activity diagram is a important diagram to describe the system.

Activity Diagram- Stock Management System



Video Content / Details of website for further learning (if any):
https://meeraacademy.com/activity-diagram-for-stock-management-system/

Important Books/Journals for further learning including the page nos.:
http://www.programsformca.com/2012/03/uml-diagrams-for-stock-maintenance.html

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 41

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : V - Case Tools **Date of Lecture: 12.04.2021**

Topic of Lecture: Quiz System

Introduction: Quiz Management System Data flow diagram is often used as a preliminary step to create an overview of the Quiz without going into great detail.

Prerequisite knowledge for Complete understanding and learning of Topic

- Dependencies between packages
- Dataflow Diagram
- UML
- Reservation

Detailed content of the Lecture:

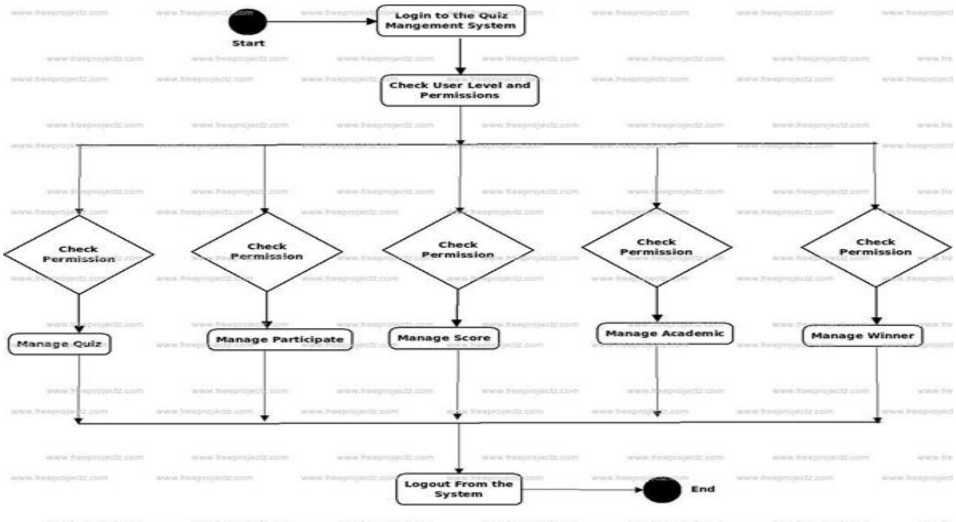
The Quiz Management application aims at the Employee assessment programs carried out in the organization by conducting Quizzes. The application focuses on creating a well-defined exam/quiz for individual departments. When the authorized examinee attends the exam, he gets to view the result on exam submission. The application administrator can view all the results with statistics as the total number of passed and failed candidates and the respective percentages. The application allows creating a scheduled exam / time specific exam.

The application comprises a strong mailing functionality to intimate the examinee of the exam details before and exam/result details after attending the same. The application simplifies, and completely automates the Quiz management system in the organization.

A quiz is a quick and informal assessment of student knowledge. Quizzes are often used in North American higher education environments to briefly test a students' level of comprehension regarding course material, providing teachers with insights into student progress and any existing knowledge gaps.

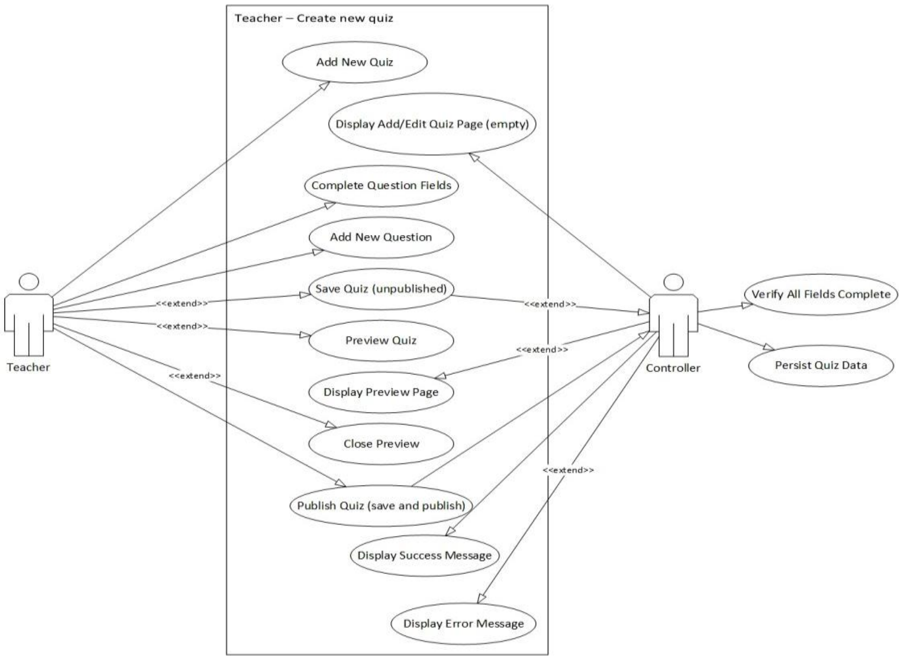
Online Quiz System (OQS), is a web–based quiz system; a system that can be used by lecturers to evaluate students effectively, efficiently and perfectly. The purpose of Online Quiz System is to save lecturer’s time since the answers are automatically marked. Online Quiz System is developed by using a rule-based algorithm. Rule-based algorithm used in this system is short answer based on keyword.

Activity Diagram



Activity Diagram for Quiz Management System

Usecase Diagram



Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

Important Books/Journals for further learning including the page nos.:

https://www.sourcecodesolutions.in/2010/12/quiz-system-using-case-tools.html

Course Faculty

Verified by HOD

DESIGNING YOUR FUTURE
Estd. 2000

IQAC

LECTURE HANDOUTS

L 42

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **V - Case Tools** **Date of Lecture: 15.04.2021**

Topic of Lecture: E-mail Client system

Introduction: Emails are stored in the user's mailbox on the remote server until the user's email client requests them to be downloaded to the user's computer, or can otherwise access the user's mailbox on the possibly remote server. The email client can be set up to connect to multiple mailboxes at the same time and to request the download of emails either automatically, such as at pre-set intervals, or the request can be manually initiated by the user.

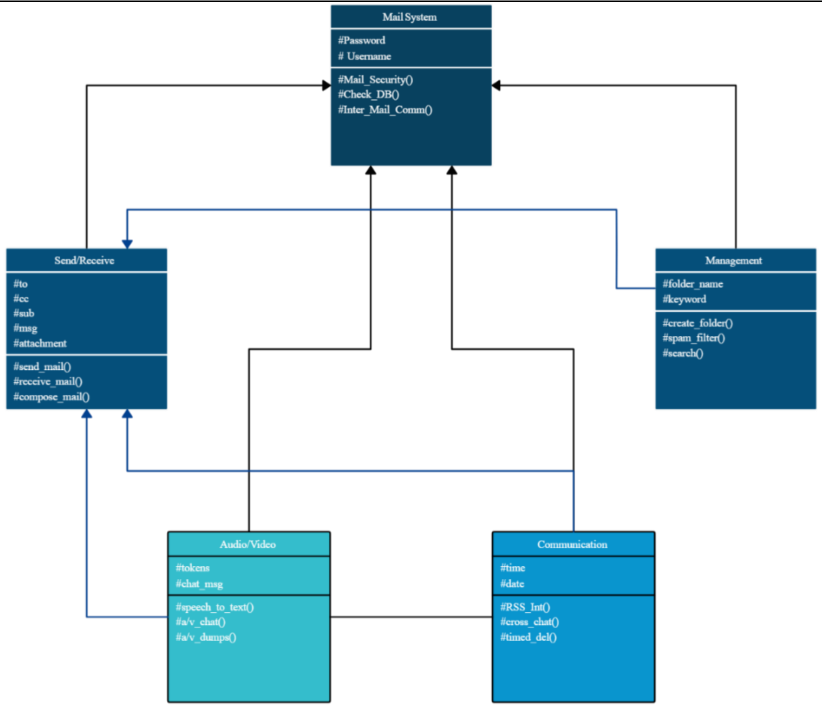
Prerequisite knowledge for Complete understanding and learning of Topic

- Client
- Server
- Network
- UML Diagrams
- World Wide Web

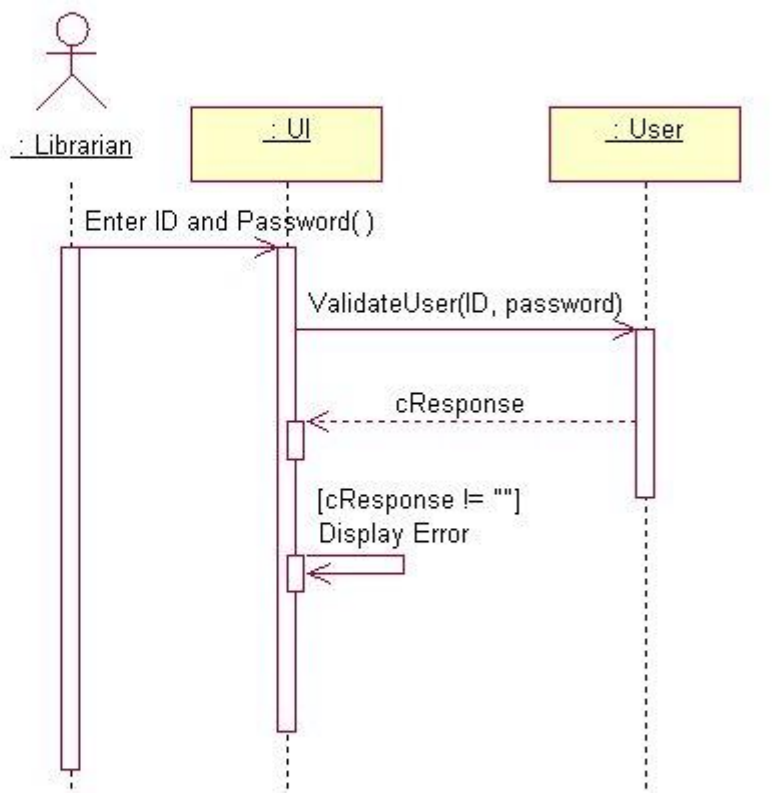
Detailed content of the Lecture:

The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients.

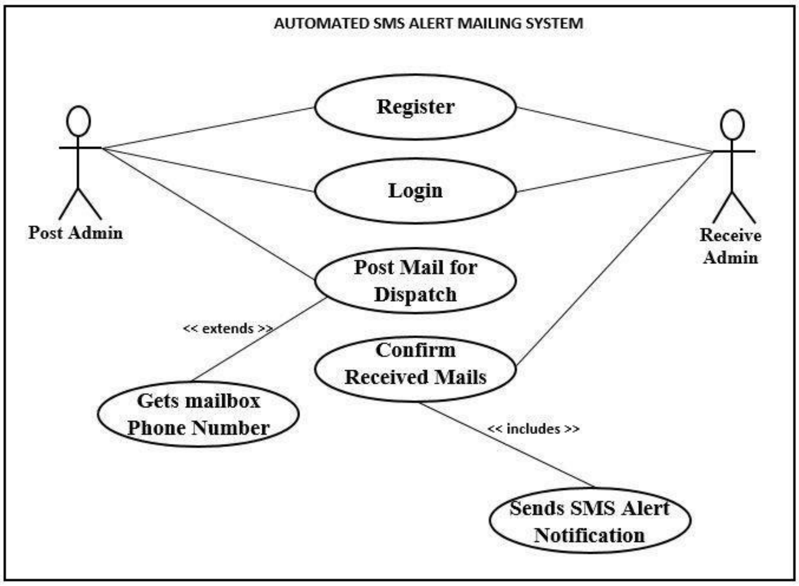
A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web."



Sequence Diagram



Usecase Diagram



Video Content / Details of website for further learning (if any):
https://reeprojectz.com/uml-diagram/mailing-system-activity-diagram

Important Books/Journals for further learning including the page nos.:
https://www.conceptdraw.com/examples/uml-diagrams-for-email-client-system

Course Faculty

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L 43

MCA

I / II

Course Name with Code : **19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN**

Course Faculty : **M. MENAKAPRIYA**

Unit : **V - Case Tools** **Date of Lecture: 17.04.2021**

Topic of Lecture: Cryptanalysis

Introduction: Cryptanalysis is **the decryption and analysis of codes, ciphers or encrypted text**. Cryptanalysis uses mathematical formulas to search for algorithm vulnerabilities and break into cryptography or information security systems.

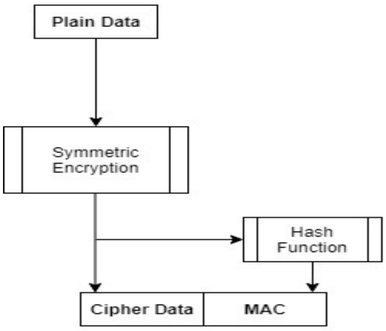
Prerequisite knowledge for Complete understanding and learning of Topic

- Encryption
- Decryption
- Security
- Hashing
- UML Diagram

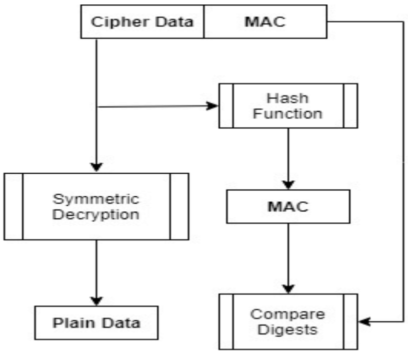
Detailed content of the Lecture:

Cryptanalysis is the study of ciphertext, ciphers and cryptosystems with the aim of understanding how they work and finding and improving techniques for defeating or weakening them. For example, cryptanalysts seek to decrypt ciphertexts without knowledge of the plaintext source, encryption key or the algorithm used to encrypt it; cryptanalysts also target secure hashing, digital signatures and other cryptographic algorithm

UML Diagram



Authenticated Encryption

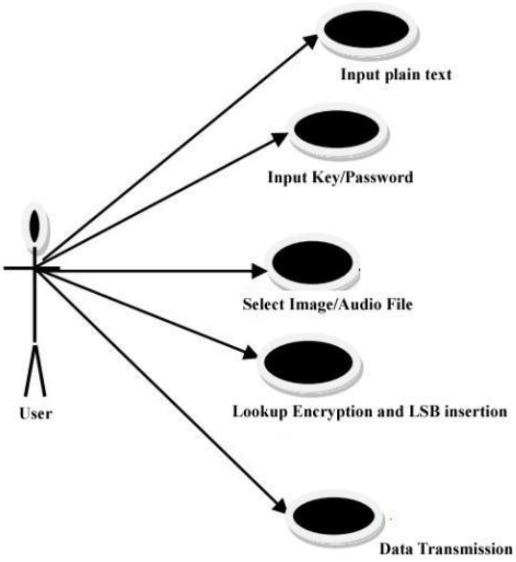


Authenticated Decryption

While the objective of cryptanalysis is to find weaknesses in or otherwise defeat cryptographic algorithms, cryptanalysts' research results are used by cryptographers to improve and strengthen or replace flawed algorithms. Both cryptanalysis, which focuses on deciphering encrypted data, and cryptography, which focuses on creating and improving encryption ciphers and other algorithms, are aspects of cryptology, the mathematical study of codes, ciphers and related algorithms.

Cryptanalysis is practiced by a broad range of organizations, including governments aiming to decipher other nations' confidential communications; companies developing security products that employ cryptanalysts to test their security features; and hackers, crackers, independent researchers and academicians who search for weaknesses in cryptographic protocols and algorithms. It is this constant battle between cryptographers trying to secure information and cryptanalysts trying to break cryptosystems that moves the entire body of cryptology knowledge forward.

Use case Diagram



Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

Important Books/Journals for further learning including the page nos.:

http://sussmanagency.com/fullcontactpolitics/wordpress/wp-includes/js/plupload/06-2010.cryptanalysis-case-study-in-ooad_3129.php

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 44

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : V - Case Tools Date of Lecture: 19.04.2021

Topic of Lecture: Health Care Systems

Introduction: Healthcare Management deals with the management or administration of healthcare organizations such as hospitals, nursing homes, diagnostic centers, and other allied centers that provide care to the needy.

Prerequisite knowledge for Complete understanding and learning of Topic

- Classes
- Models
- Extensibility
- Package
- UML Diagram

Detailed content of the Lecture:

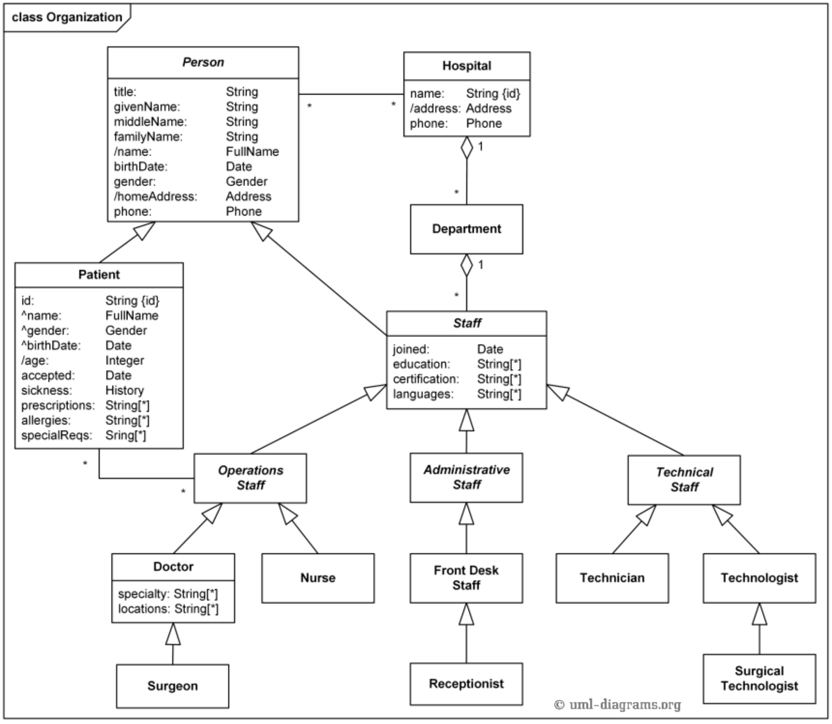
Healthcare management, also referred to as healthcare administration, is the administration, management or oversight of healthcare systems, public health systems, hospitals, entire hospital networks or other medical facilities. Duties of these professionals include ensuring that individual departments run smoothly, qualified employees are hired, information is disseminated efficiently throughout the organization, specific outcomes are reached and resources are used efficiently, among many other responsibilities.

There are general healthcare managers and those who are considered specialists. Generalists oversee entire facilities, while specialists focus on the administration of specific departments like marketing, finance, policy analysis or accounting.

Healthcare management” is an umbrella term that covers a wide variety of job titles. Clinical directors, healthcare supervisors, health coordinators, and nursing home facilitators often have degrees in healthcare management. And while you may think of healthcare managers as people who work at hospitals or in private practice, they may also work at colleges or universities, public health centers, urgent care clinics, insurance companies, or pharmaceutical companies.

A healthcare administrator will determine the best ways to help staff be more efficient at their jobs and will understand the type of treatment the facility will provide to patients. The healthcare manager is the one who decides on the treatment, number of staff and how each department should be run. Healthcare managers focus on the big-picture needs and direction of a hospital or other medical setting, while administrators focus largely on working with the staff. In hospitals, healthcare managers typically oversee hospital-wide matters, and administrators oversee individual departments.

UML Class Diagram



Video Content / Details of website for further learning (if any):
https://itsourcecode.com/uml/hospital-management-system-use-case-diagram/

Important Books/Journals for further learning including the page nos.:

https://www.conceptdraw.com/examples/healthcare-system-in-ooad

Course Faculty

Verified by HOD

LECTURE HANDOUTS

L 45

MCA

I / II

Course Name with Code : 19CAB12 / OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty : M. MENAKAPRIYA

Unit : V - Case Tools **Date of Lecture: 20.04.2021**

Topic of Lecture: Use Open-Source CASE Tools: StarUML / UML Graph for the above case studies.

Introduction: StarUML is an open-source software modeling tool that supports the UML (Unified Modeling Language) framework for system and software modeling. It is based on UML version 1.4, provides eleven different types of diagrams and it accepts UML 2.0 notation. It actively supports the MDA (Model Driven Architecture) approach by supporting the UML profile concept and allowing to generate code for multiple languages.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Case Tools
- UML
- Model
- Open Source

Detailed content of the Lecture:

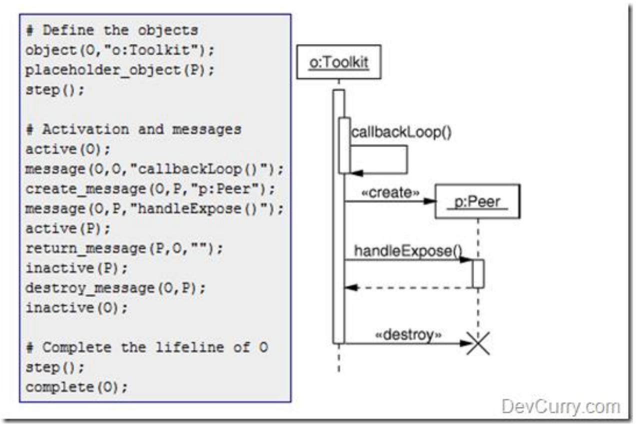
Computer-Aided Software Engineering (CASE) technologies are tools that provide automated assistance for software development. The goal of introducing CASE tools is the reduction of the time and cost of software development and the enhancement of the quality of the systems developed. The interest in CASE tools and environments is based on expectations about increasing productivity, improving product quality, facilitating maintenance, and making software engineers' task less odious and more enjoyable.

CASE is the use of computer-based support in the software development process; a CASE tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process; a CASE environment is a collection of CASE tools and other components together with an integration approach that supports most or all of the interactions that occur among the environment components, and between the users of the environment and the environment itself.

StarUML is an open-source project to develop fast, flexible, extensible, featureful, and freely-available UML/MDA platform running on Win32 platform. The goal of the StarUML project is to build a software modeling tool and also platform that is a compelling replacement of commercial UML tools

ArgoUML - ArgoUML is the leading open-source UML modeling tool and includes support for all standard UML 1.4 diagrams. It runs on any Java platform and is available in ten languages.

UMLGraph - UMLGraph allows the declarative specification and drawing of UML class and sequence diagrams. The current features are part of an ongoing effort aiming to provide support for all types UML diagrams.

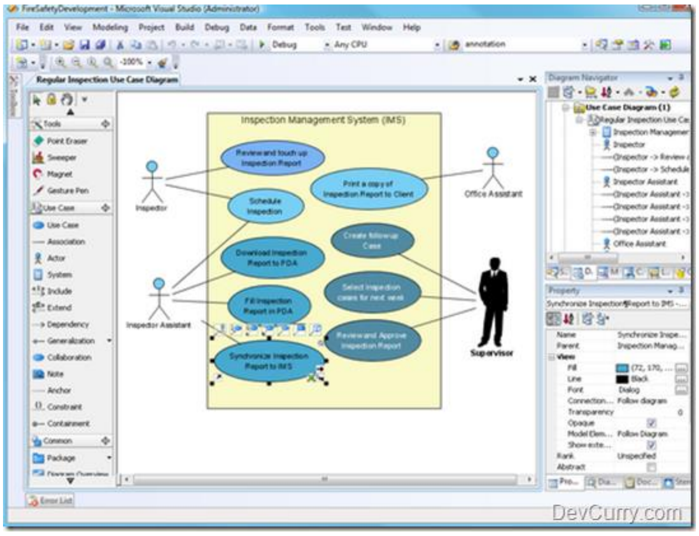


```
# Define the objects
object(O,"o:Toolkit");
placeholder_object(P);
step();

# Activation and messages
active(O);
message(O,O,"callbackLoop()");
create_message(O,P,"p:Peer");
message(O,P,"handleExpose()");
active(P);
return_message(P,O,"");
inactive(P);
destroy_message(O,P);
inactive(O);

# Complete the lifeline of O
step();
complete(O);
```

Visual Paradigm SDE for Visual Studio - Smart Development Environment Community Edition for Visual Studio (SDE-VS CE) fully supports the latest version of UML. Open-source projects' developers can use SDE-VS CE to design system with UML. SDE-VS CE is free non-commercial use only. SDE-VS is embedded in Visual Studio



FireSafetyDevelopment - Microsoft Visual Studio (Administrator)
File Edit View Modeling Project Build Debug Data Format Tools Test Window Help
Debug
Any CPU
annotation
Regular Inspection Use Case Diagram
Inspection Management System (IMS)
Review and touch up Inspection Report
Schedule Inspection
Print a copy of Inspection Report to Client
Create follow-up Case
Download Inspection Report to PDA
Select inspection cases for next visit
Fill Inspection Report in PDA
Review and Approve Inspection Report
Synchronize Inspection Report to IMS
Inspector
Inspector Assistant
Office Assistant
Supervisor
Diagram Navigator
Property
DevCurry.com

Video Content / Details of website for further learning (if any):
https://www.methodsandtools.com/tools/staruml.php

Important Books/Journals for further learning including the page nos.:
https://docs.staruml.io/working-with-uml-diagrams/use-case-diagram

Course Faculty

Verified by HOD