UNIT-I

Unified Process and use case diagram.

Introduction to OOAD with OO Basics - unified process - UML diagram - use case - case study the Next Gen POS system, inception - use case modelling Relating use cases - include, extend and generalization when to use Use-cases.

Guideline for use case Modeling:-

↳ Guideline 1:- write in an essential UI free style

⇒ Keep the user interface out and focus on actor intent.

Essential style

We Assume that Manage users requires Identification and Authentication.

Concrete style Avoid During Early Requirements work.

↳ user interface decisions are embedded in the use case text.

Guideline 2:-

↳ write terse use cases:-

⇒ Delete Noise words

→ small changes add up such as "system authenticates" rather than "The system authenticates"

Guideline 3:-

↳ write Black Box use cases:-

↳ Black Box use cases are most common and recommended.

coz, they do not describe the internal working of the system.

Rather than describes as having Respon[illegible]

f

Guideline 4:-

↳ Take an actor and Actor Goal Perspective:-

A set of use case instances (sequence of acti[illegible]

yields an observable result of value to a partic[illegible]

actor.

⇒ Has two attitude.

① ⇒ writes requirements focusing on the users a[illegible]

asking their goals & situation

② ⇒ Focus on what a actor considers as

valuable result

Guideline 5

↳ To find use case.

~~Applying~~

Applying UML: use case Diagram.

↳ To illustrate the names of use cases and actors and the relationships between them.

↳ use cases are text documents.

↳ The work of use case to write text.

Guideline

↳ is an excellent picture of the system.

↳ it makes a good context diagram.

↳ it serves as a communication tool that summarizes the behavior of the system

Guideline: Diagramming

⇒ Actor box with the symbol «actor»

⇒ It includes guillement symbol («actor»)

Relating use cases

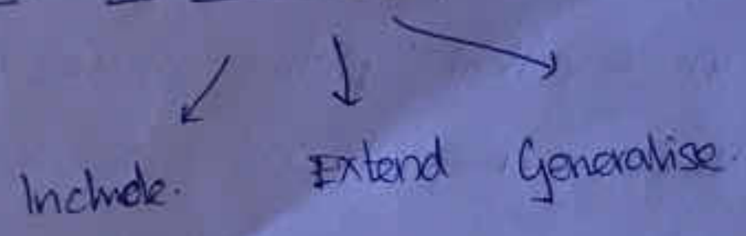
Use cases can be related to each other.

A sub function has a Process sale and Process Rental.

It is simply an organization mechanism to improve communication

Therefore reduce duplication of text and improve management of use case documents

Kinds of Relationship



Include Relationship

↳ They have partial behaviour across several use cases.

↳ Rather than duplicate text. it includes the description, Hence it is desirable to separate it own subfunction use case.

→ It is an slightly shorter notation to indicate a use case ~~or this to~~ is simply to underline or highlight it

⇒

Use cases use include relation when

↳ they are duplicate in other use cases

↳ when use case is long complex and separating into subunits

Concrete, Abstract, Base and Addition

↳ Initiated by an actor and performs the entire behaviour desired

Abstract use case

↳ never instantiated by itself.

↳ it is a subfunction use case that is a part of another use case.

Base use case

↳ a usecase that includes another use case, or extended or specialized by another use case.

↳ Process sale is a base use case with respect to Handle credit payment subfunction.

↳ Inclusion, Addition, extension is called additional use case.

Addition use abstract, Base use concrete

The extend Relationship

↳ Idea to develop an extending or addition use case.

↳ Describes the condition it extend the behavior of base use case.

Extention Point

↳ are Lables in the base use case with extending use case references

→ So that it can change without affecting extending use case.

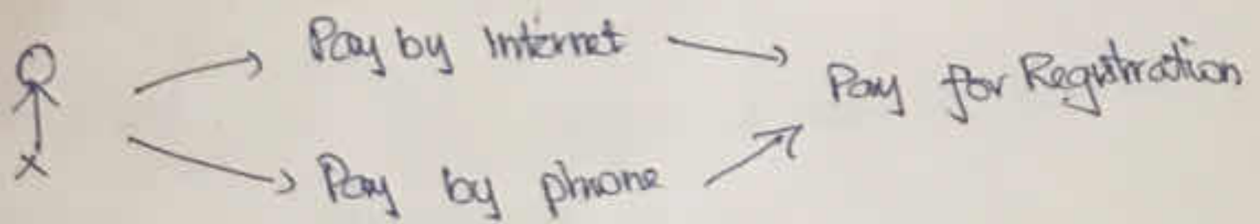
Generalization Relationship

↳ Refers to the relationship that exists between two use cases.

→ One use case (child) inherits the structure, behaviour and relationships of another actor (parent)

where child is referred to as more special and parent is referred to abstract

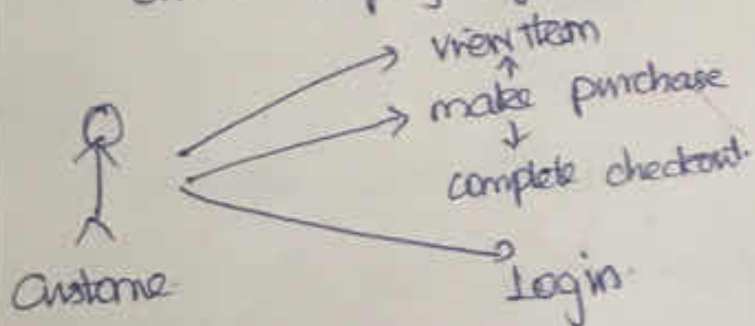
Generalization relationship ~~are referred to as~~ have same logic/behaviour



When to use USE CASE DIAGRAM

- ↳ Representing the goals of system.
- ↳ Defining & organizing functional Requirement
- ↳ specifying the context.
- ↳ Modeling the basic flow.

Ex online Shoping system.



→ Help to identify any internal or external factors

→ to specify the events of a system & flows.

→ They use black box where only Input, output and function is known.

→ They are not good candidate for forward & reverse engineering but still used in different ways.

OOAD - CS8592 [13, 18, 24, 27, 34 Absent (3rd CSE)] ①

OOAD

↳ essential for well designed, robust, maintainable. Software using OO technology.

→ Languages java & C++

→ Practical in some development process.

Development Process includes.

→ Apply principles & patterns (why) → to create better object designs.

→ To create frequently used diagrams.

Analysis

↳ do the right thing

↳ Analyse the problem & requirements rather than a solution.

Design

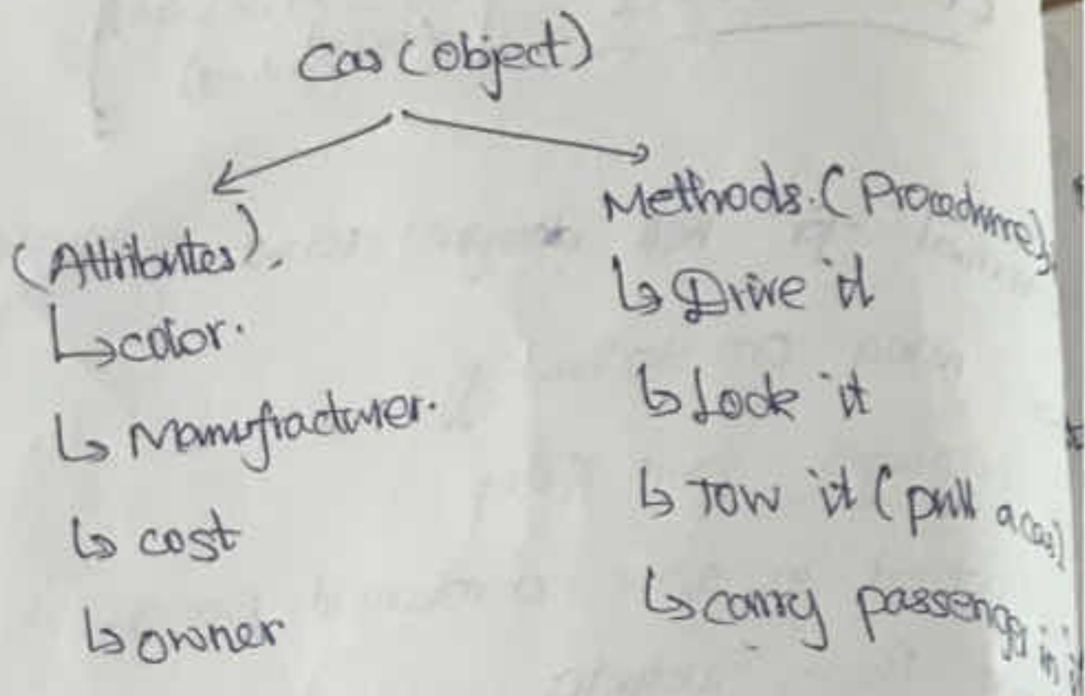
↳ do the thing right.

↳ that fulfills the requirements, than implementation

Object.

↳ cas is an real world entity, identifyble from its surroundings.

↳ it has a set of attributes in relation to other object



OBJECT → what?

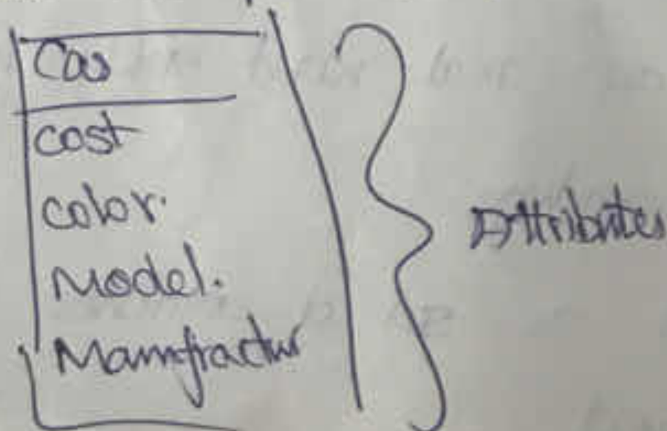
↳ combination ordata & Logic.

The programming in object Oriented System. Consists of adding new kind of object to the System & defining how they behave.

Object Properties (Attributes).

↳ represent the state of an object

↳ object oriented methods want to refer tho the description of properties.



programming Language if we take colour.

↳ can be declared as characters, to store sequence or character (ex, red, blue)

↳ can be declared to store ~~the~~ in numbers as stock number.

↳ can be declared as image or video file to refer a full color video image.

⇒ Independent of its physical representation we can define its object abstract state.

Object behaviours & Methods

↳ set of things that an object can do on its own (or) we can do with it.

Example we can drive the car (or) we can stop the car.

These represents the objects behaviours.

⇒ Behaviour is described in methods or procedures.

Method is a function.

Procedures in a class can access to perform some operation

Behaviors is what an object is capable

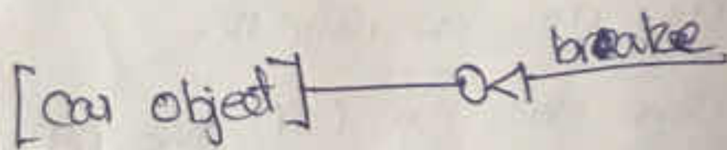
Doing.

Object respond to messages

To do an operation on message is sent to an object.

↳ when applying breake the we can send a stop message to the car object

↳ it knows how to respond to the stop message. since it's designed with specialized parts such as breake pads & drums respond to the message.



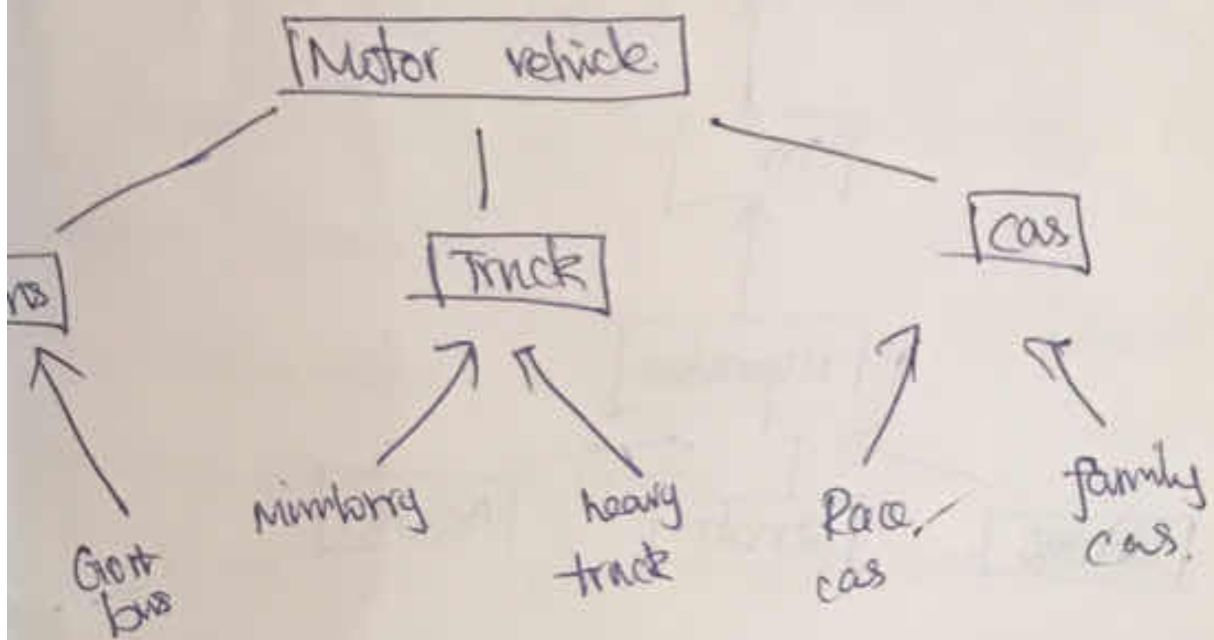
Different object can respond to the same message in different ways

Eg: car, motorcycle, & bicyck.

actual operations performed are object specified

Hierarchy.

An object oriented system organizes classes a subclass / super class.



[sub class / super class]

A subclass inherits all the propertise and ethods defined in its super class.

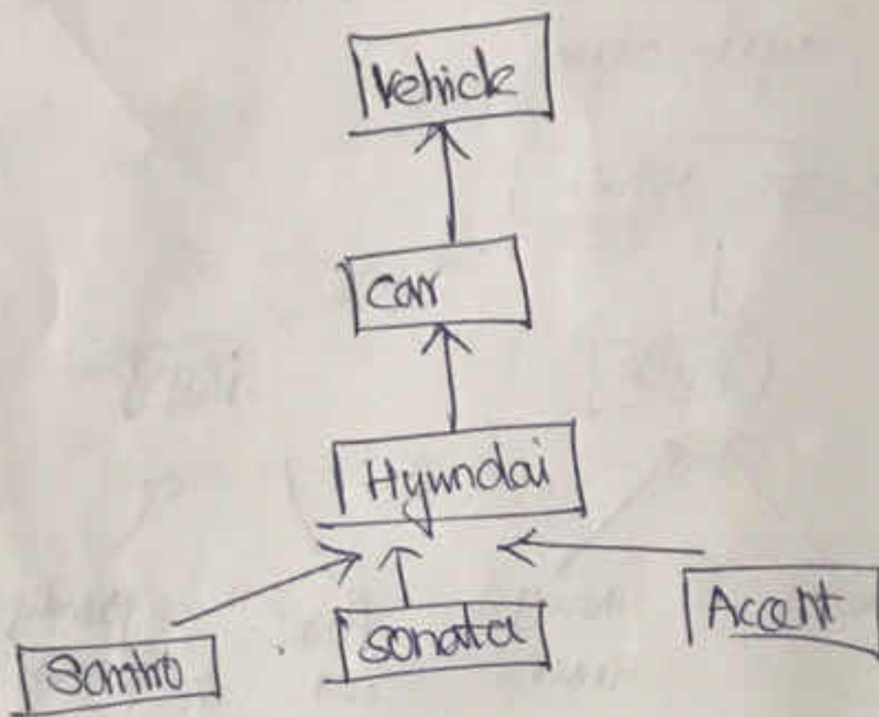
eritance

↳ allows objects to built from other objects.

↳ takes advantages of commonality of objects when creating new class

↳ it is a relationship between classes where one class is the parent class of another class (derived

→ Derived class holds the propetise and behaviour of base class



Dynamic Inheritance

↳ Allows objects to change & evolve over

Multiple inheritance.

system permits a class to inherit its state and behaviour from more than one Super class.

Encapsulation & information Hiding

→ concealing the internal data & procedures of an object.

→ Encapsulation example ⇒ car engine

orphism

Poly → "many" Morph → "form".

→ Objects can take many different forms.

→ same operations may behave differently different classes.

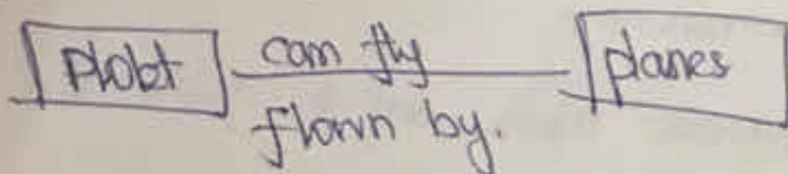
→ relationship of many objects of different sses by some super class

→ we can write reusable code easily

Ex: pay roll.

Object Relationship and Associations

Associations - relationship between class & object, bidirection (inverse)



Aggregations

Bringing up all the similar entities together.

Breaking down of the objects is called decomposition.

The car object is an aggregation of objects such as engine seat & wheel object.

Static & Dynamic Binding.

Functions to be involved at compile time called Static.

Functions to be involved at run time called Dynamic.

Object Persistence.

exists for a period of time that has been the duration of the process in which the are created

OOAD ?

OOA :- finding & describing the objects or concepts in the problem domain.

OOD :- Defining software objects and to fulfill the requirement.

Define Use case	Define Domain model	Define Interaction Diagram	Define design class Diagram

Use case

req analysis.

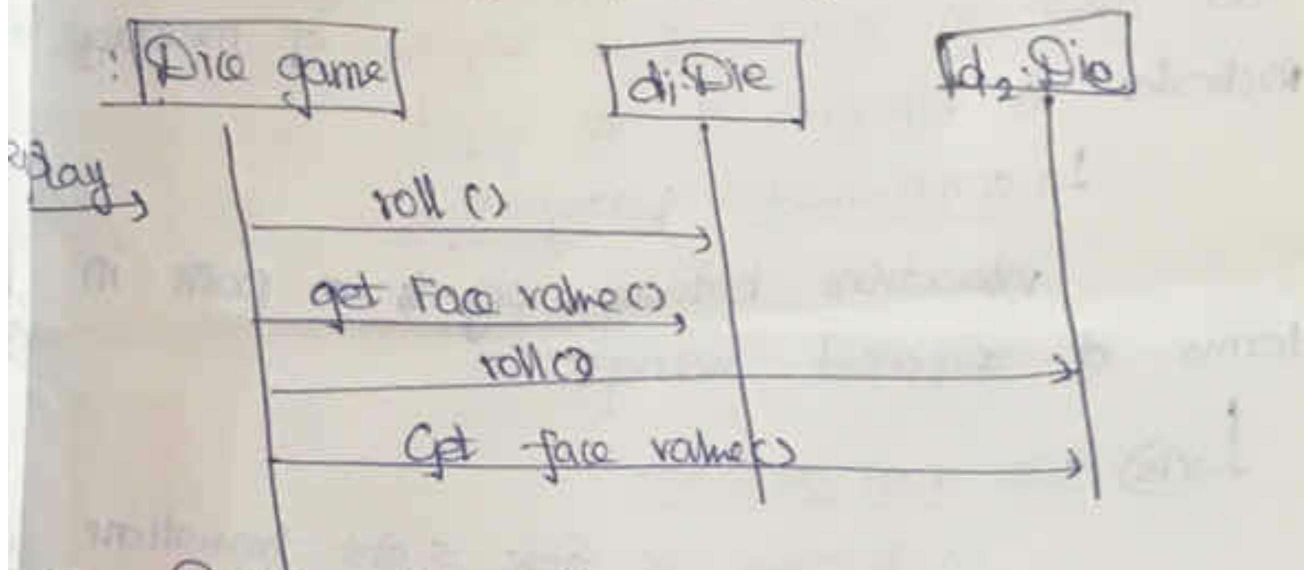
↳ Scenarios of how people use the application

play a Dice game.

Domain Model

Identification of concepts, attributes and associations

SD → defines flow of messages between software objects and thus the invocation of methods



~~Design class Diagram.~~

UML Diagram.

Unified Modeling Language → visual Language for specifying, constructing & Documenting the artifacts of the System

composed of 9 graphical Diagram.

① class diagram → describes the structure of a system by showing the System's class, attributes and relationship among the classes

② Use case Diagram:

→ functionality provided by the in terms of actors, their goals represented use case

③ Behaviour Diagram

↳ a) interaction Diagram.

↳ Sequence Diagram.

· Shows how objects communicate with each other in terms of a sequence of messages indicates the lifespan of the object

↳ communication Diagram.

interactions between objects or parts in terms of sequenced messages

↳ ⓑ State chart diagram

↳ describes the state & state transitions of the system.

↳ ⓒ Activity diagram

Describes the business and operational. shows the overall flow & control.

④ Implementation Diagram.

a) Component Diagram - describes how a software system is split up into components.

b) Deployment Diagram - the hardware used in a system implementation & the execution environment deployed in hardware

Three ways to Apply UML.

→ UML as sketch:
→ UML as blueprint→
→ UML as programing Language.

Three Perpectives to apply UML.

1) conceptual perspective→
2) specification "
3) Implementation "

① The diagrams are interpreted as describing things in a situation of the real world

Specification Perspective

↳ The diagram describe the software abstractions or components with specification & interfaces

Implementation perspective

Describes in particular technology

UP → Unified process

A software development process describes an approach to building, deploying & possibly maintaining process.

⇒ used for building object oriented System

Rational Unified Process (RUP) is adopted

UP ~~accept~~

↳ Iterative

Why up

→ iterative process.

→ Structure for how to do and how to explain OOAD.

→ UP is flexible,

→ can be applied in Light weight

Iterative & Evolutionery Development

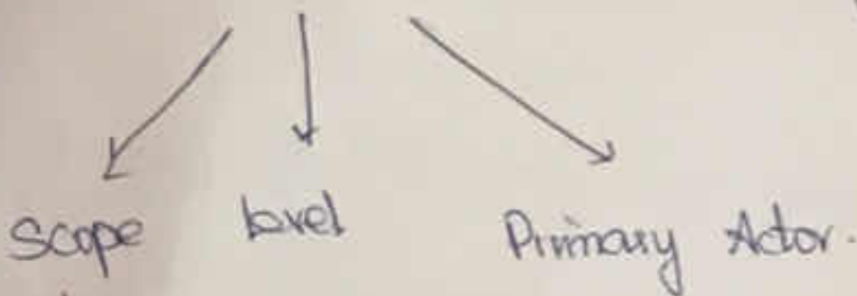
→ Development is Organized into a series of Short, fixed Length, mini project called iterations.

→ The output will be a tested, integrated ~~x~~ System.

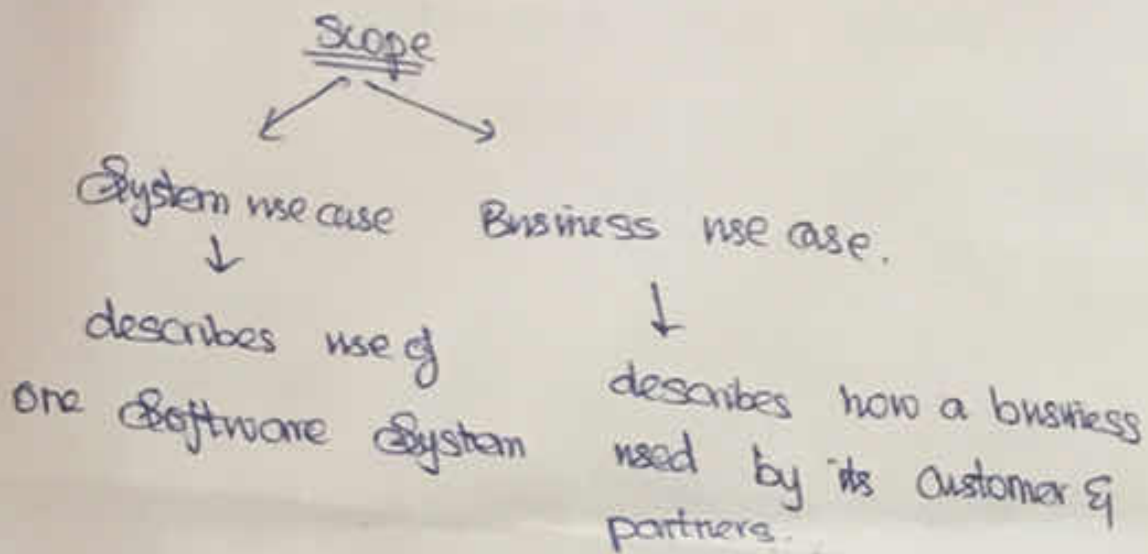
→ Each iteration includes its own requirement Analysis, implementation, design & testing activities

→

Format Descriptions



↳ can be used as use case or Business Usecase



Level

describes the scenario's to fulfill the goals of a primary actor to get work done.

Primary Actor

System calls upon service to fulfill a goal.

Stakeholders & interest List

We have a method to remind us what the more detailed responsibilities.

of the system be done.

Cashier :-

↳ Accurate fast entry and no payment error

Sales person

↳ want sales commissions updated.

Precondition and sucess gurantees

Pre condition

→ Must be true before a scenario is begun in the usecase.

→ Not tested within the use case. but are assumed to be true.

Main success Scenario & Steps.

↳ "happy path" scenario or more "Basic flow."

Three steps

↳ An interactions between actors

↳ A validation

↳ A State change by the system. (Recording or modifying)

xtension (or) Alternate flows.

Extension are branches [success or failure] from the main success scenario.

Noted with respect to the steps

Example :-

→ An extension is labled a "3a".

* It first identifies the condition and then the response.

→ Alternate extensions at step 3 are labeled as "3 b". and so forth.

Extensions

3a) Invalid identifier.

↳ error and rejects entry.

3b)

Guideline

Writes condition that can be detected by the system.

7a) Paying by credit

↳ enters the credit acc information

↳ sends request to payment Authorization.

↳ request approval.

2a) system detects failure

1) system signals error to cashier

2) cashier asks for alternate payment

Special Requirements

For non functional requirement it requires.

↳ quality Attributes.

↳ constraints.

Touch screen UI on a large flat panel monitor

⇒ Text Should be visible from 1 meter.

⇒ respone within 30 Second.

⇒ Language Internalization Should be done.

Technology & Data variants

→ Needed to understand varients in data schemes.

42, 44, 45, 49, 46
50, 51 52, 53, 58
45, 46.

UNIT-2.

Static UML Diagram.

Class Diagram- Elaboration - Domain model - Finding conceptual classes and description classes - Associations - Attributes - Domain model refinement Finding conceptual class Hiearchies - Aggregation and composition - Relationship between sequence diagram and use cases - when to use class diagram.

UNIT-II.

class diagram

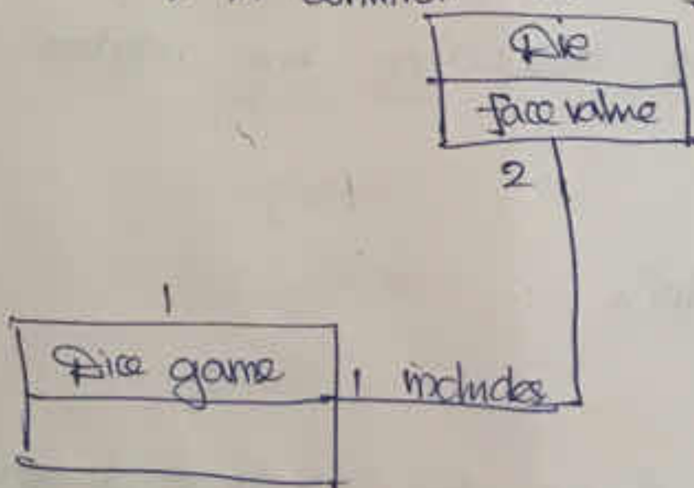
* UML includes class diagram to illustrate classes, interfaces and their associations.
  * used for static object modeling.
  * describes responsibilities of the system
  * Keywords are used.

Definition

→ To visualise a domain model.

→ clarify when a class diagram is used

→ A common modeling term.



class Diagram Representation

class is represented as rectangular box showing attributes operations

The main elements of class are

1. Attributes
2. operations & models.
3. Relationship between class.

1) Attributes

↳ logical Data value of an object.

↳ The full format of attribute is

<u>Syntax</u>

visibility name: type multiplicity = default {property-string}

visibility name: type multiplicity = default {property-string}.

<u>visibility</u> marks include +(public), -(private).

<u>Guideline:</u>

use the attribute text notation for data type objects and association line notation for others

2) <u>Operation</u> <u>and</u> <u>Methods</u>

<u>Operations</u>

⇒ class box shows the signatures of operation

⇒ Operations are usually assumed public if no visibility is shown.

* Both expressions are possible.
* An operation is not a method.

Syntax

Visibility name (parameter-list) : return-type {property string}

To show methods in class Diagram.

⇒ A UML method is implementation of a operation.

Illustration ways

→ In interaction Diagram.

→ In class diagram.

Relationship between class

A) Associations
B) Generalization & specialization
c) composition
d) Dependency
e) Interface.

Associations

Relationship between class. Relationship between two or more classifiers.

for many (*)

* [T] zero or many

1..* [T] one or more.

1...40 [T] one to 40

5 [T] exactly 5

3,5,8 [T] exactly 3, 5, or 8

Generalization & Specialization

Generalization

→ activity of identifying commonality among concepts and defining superclass & Subclass

Composition & Aggregation

Composition

↳ Known as composite Aggregation

→ useful to show some models

Implies

→ An instance of part

→

Associations

↳ Relationship between classes

↳ One class must know the other work.

↳ indicated by a straightline

Aggregations

→ one class belongs to a collection.

→ indicated by a empty diamond

Composition

→ Strong form of Aggregation

→ Lifetime control

→ indicated by a solid diamond.

Inheritance

→ Indicated by a triangle pointed to superclass.

Unary Associations

A Knows about B, but B Knows nothing about A.

Aggregation

Aggregation is an association with a Collection number.

Composition

Lifetime control.

Part object may belong to only one whole object.

Inheritance

The concept of having same functionalities from base class is inheritance.

UML Multiplicities

0....1 ⇒ zero or one.

0...* (or) * ⇒ no limit

1 ⇒ exactly one.

1...* ⇒ atleast one.

Elaboration

* Initial series of iterations

⇒ serious investigation

⇒ Discover & stabilize

⇒ Mitigate

⇒ Build core. architecture

Key Idea's

→ Not a waterfall model.

→ Two to six weeks for each iteration

→ Timeboxed.

→ Each product ends in a stable state.

Best practices

→ Start program early

→ Adapt based on feedback.

→ Design, implement & test.

→ Test early and realistically

→ Requirements & use case details.

Artifacts

↳ Domain model.

↳ Design model.

↳ Software Architecture Document

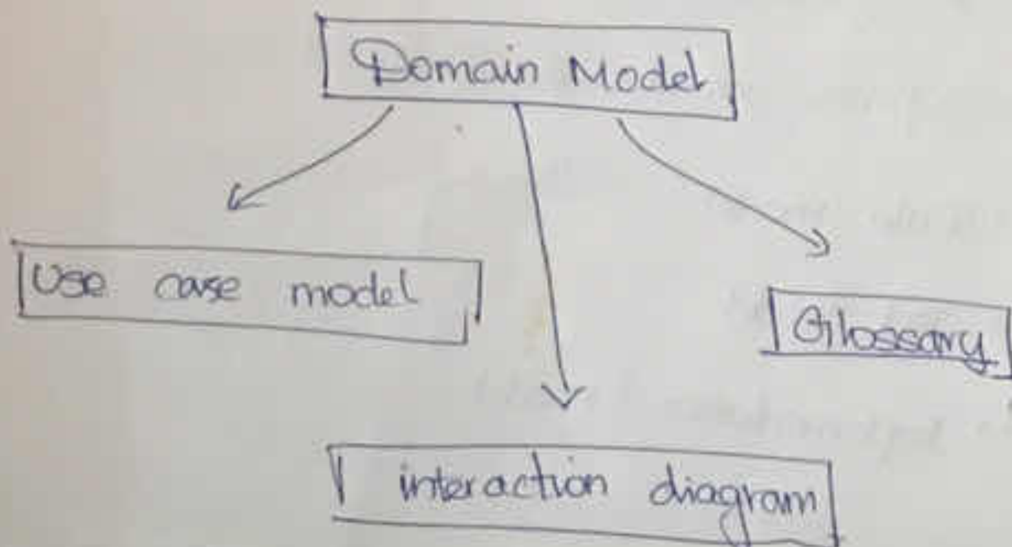
↳ Data model.

↳ Test model.

↳ Implementation model.

Elaboration is tedious when

* No timebox scheduled.
* single iteration
* Most requirements already defined.
* No Risk mitigation
* NO Executable Architecture
* Requirement Phase
* Attempt full and careful design
* Minimal feedback and adaptation
* No early and realistic testing.
* Frozen Architecture.
* Noc proof-of concept programming.
* No multiple requirement workshop



Domain Model.

* Illustrates meaningfull classes in a problem domain

* Representation of real world concepts.

* NOT a set of diagrams describing software classes

* May show.

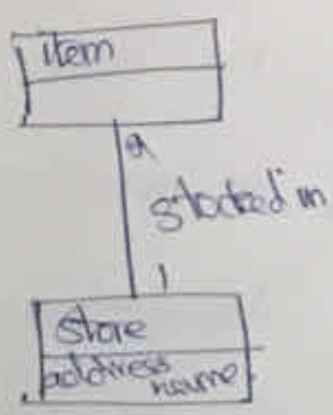
→ concepts.

→ associations between concepts

→ attributes of concepts

UML notation

→ Illustrates using a set of diagrams for which no operations defined.

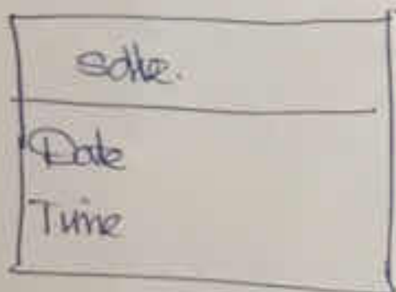


* Description of things in the real world.

* Not a description of software design

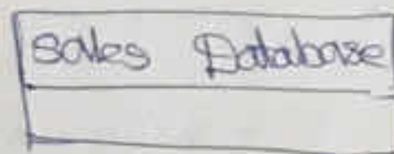
* An concept is an idea, thing or object

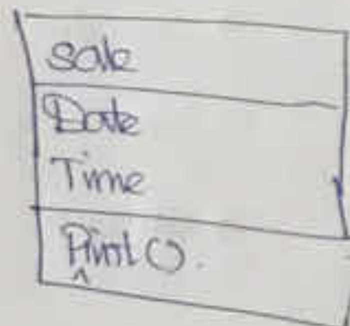
A conceptual class



vs

Software Artifacts.





Identify conceptual classes

* Identify nouns and phrases in textual Description

* Fully Dressed use cases are good.

* Not strictly a mechanical process.

* words may be ambiguous.

* Different phrases may represent same concept

Steps to create Domain Model

* Identify candidate classes
* Draw them in Domain Model
* Apply existing analysis patterns

Apply the MapMaker Strategy

* using existing names for things, the vocabulary of the Domain
* Exclude irrelevant features
* Do not add things that are not there.

Common Mistake

If takes up space, then it is likely a conceptual class

Example

→ A Store is note an attribute of sale.

→ A destination is not a attribute of flight

Specification

→ class Records information

* If all the instances are sold out the description remains

* Avoids duplication of recordings

Associations

Relationship between the instances of types of indicate some meaningful connection

Introduction

→ Identify associations

→ Identify the aid in comprehending the domain model.

Useful Associations

* Relationship needed to be preserved for some duration

* Derived from common association List.

UML ASSOCIATION NOTATION

* Association is represented as a line between classes
* Inherently Bibidirectional.

Categories

* A is a physical part of B.
* A is a logical part of B.
* A is phically contained in B

Common Association List

* A is logically contained in P.
* A is description of B.
* A is line item of report B
* A is a member of B.
* A uses B.
* A is next to B
* A communicates with B.
* A is owned by B.
* A is next to B.

High priority Associations

* A is physical part of B.
* A is recorded in B.
* A is contained in B

Associations Guidelines

* The knowledge should be preserved for some durations.
* Avoid showing redundant associations
* Identifying classes is more important than identifying Associations.

Roles.

Each end of an association is called role.

Roles may have

* name
* multiplicity
* navigability.

Aggregation

Kind of associations used to model whole part relationships

The whole is called composite.

Aggregation in UML

Aggregation is shown with a hollow or filled diamond symbol

Aggregation is a property of an association role.

Composite Aggregation.

A finger is in a part of whole one hand, thus the diamond is filled to intricate composite aggregation

Shared Aggregation

A UML Package is consided to aggregate its elements

usually present in physical assemblies.

sometimes; not clear

System sequence Diagram

class diagram represent static modeling.

Interaction Diagram show how groups of objed collaborate

typically captures the behavior of single use case

SSD

The system as a black box.

The external actors

The system events that generates their temporal order.

How to construct an SSD from Usecase.

Draw a line representing the system as a black box.

Identify each actor that directly operates on the System.

Draw a line for each other.

From Detailed use case System events that each generates.

Class Diagram.

↳ backbone of all the OO methods. used in all the time

Domain Refinement Model.

↳ identify superclasses and subclasses when they help us to understand concepts in more general.

↳ Expand class hierarchy relevent to current iteration.

100% eq. Is a rule.

↳ subclasses must conform to 100% of superclass attributes and associations.

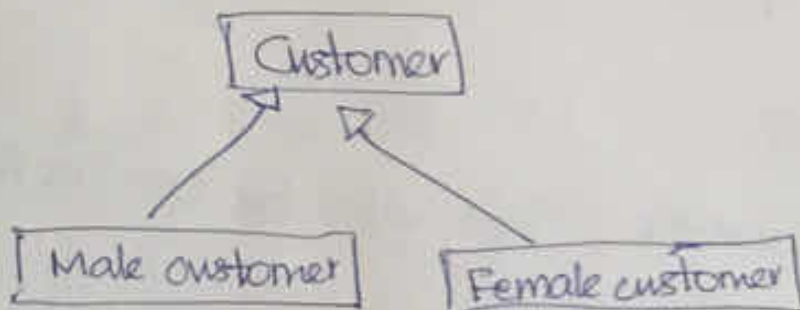
Is a rule.

superclass is a subclass

Hearchy

↳ subclasses has additional attributes

↳ subclasses



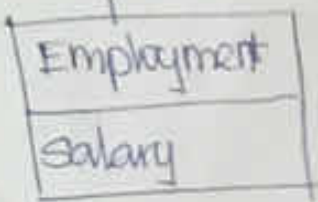
classes should be invarient.

classes can maintain state information

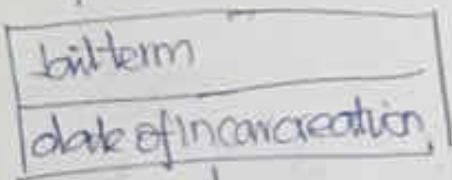
Rather than subclasses, model changing states.

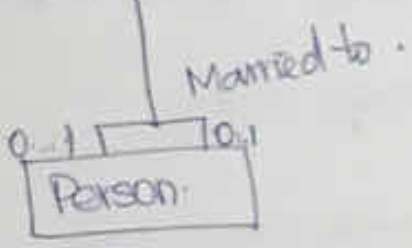
More Association classes

company* Employee *Person



Jail Incarcerates Person





Conceptual class Hierarchy

- ↳ facilitates the reuse of codes
- ↳ Reuse of solutions to design

Problems

class Hiearches as sets - supersets and subsets

Example

Design of class Hiearchy.

→ Put common Data.

→ functionality in a super class.

→ The subclass inherit the data and functionality of super class

→ Avoid duplicating the codes

→ Add or change any subclassed without modyfing the superclass.

→ If a superclass changes then subclass automatically changes.

UNIT-3.

1) Dynamic Diagrams.

→ To represent the behaviour of software

→ To represent time dependent aspects of a system

→ Represent the behaviour, workflow, states and so on.

* use case modeling
* Interaction modeling
*   ↳ Sequence diagram
    ↳ communication diagram.

State Modeling:-

* State chart diagram.
* Activity diagram.

UML Interaction Diagram

To illustrate how object interact via messages

used for dynamic modeling.

Types.

* sequence diagram
* collaboration diagram.

Sequence Diagram

→ illustrates in fence format

→ shows the sequence / modeling.

→ consumes the vertical space of the diagram

→ large set of detailed notation options.

Collaboration Diagram

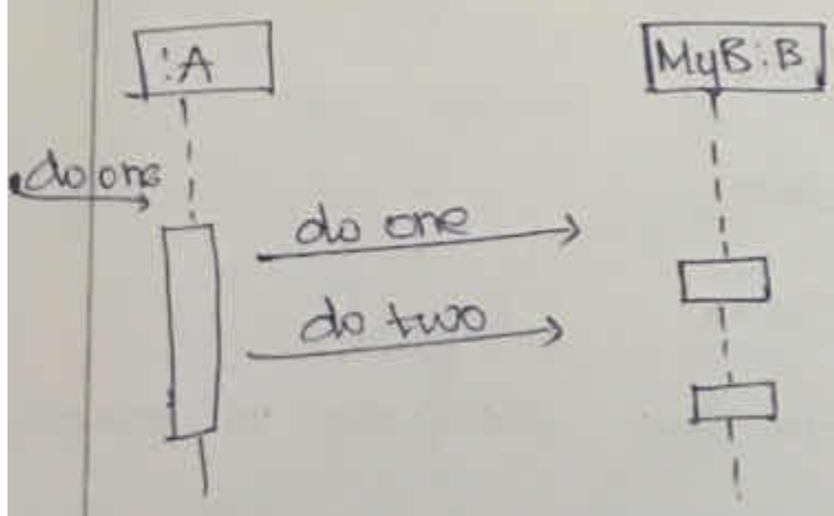
→ illustrates object interaction in graph format

→ it is difficult to see the sequence of messages

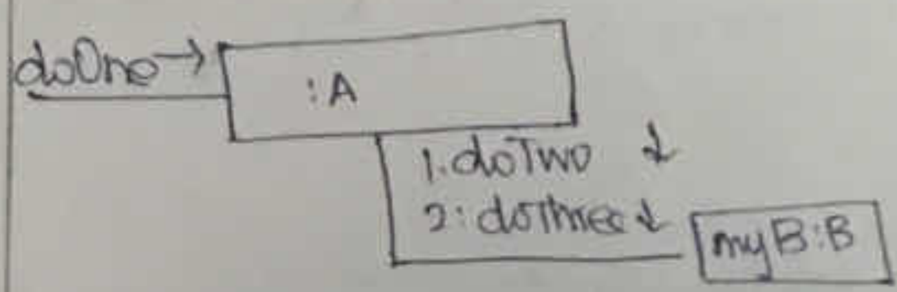
→ has fewer notation options

Example

```
public class A
{
 Private B myB= new B()
 public void do one()
 {
 MyB.doTwo();
 myB.doThree();
 }
}
```



Sequence Diagram.



Communication Diagram

When to use interaction Diagram

→ To capture the dynamic behaviour of a syst

→ To describe the message flow.

→ To describe the Structural organizations

→ To describe the interaction

How to drawn an interaction

* object taking part in interaction

* Message flows

* The sequence.

* object Organization

Interaction Diagram Notation

Lifeline boxes to show participants in interaction

Basic syntax.

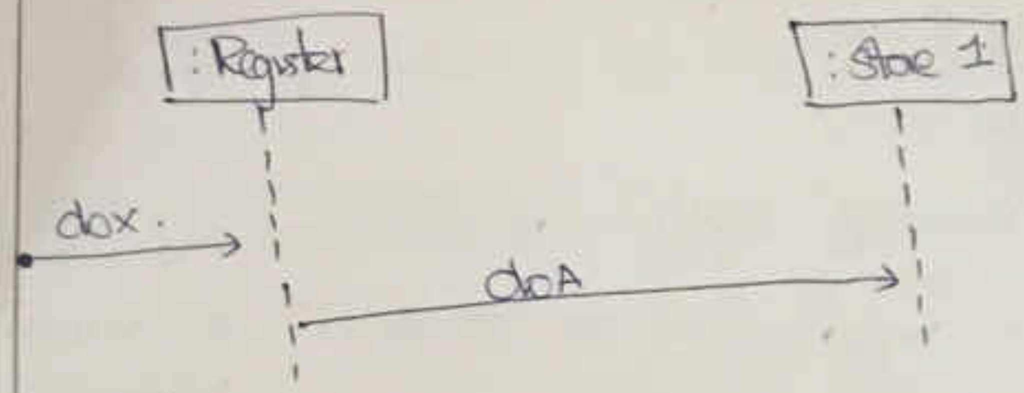
return = message (parameter: parameter Type): returnType

Singleton object

* only one instance of a class.

* An object is marked with '1' in the

upper right corner of the lifeline box



Lifeline Box.

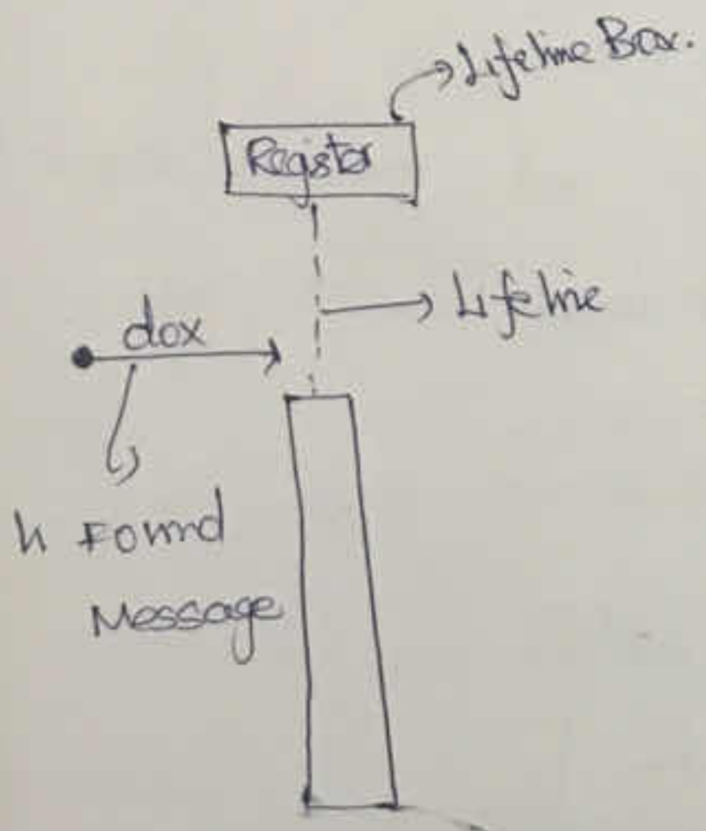
Represents the participants in interaction

Life line.

vertical dashed line

Found message

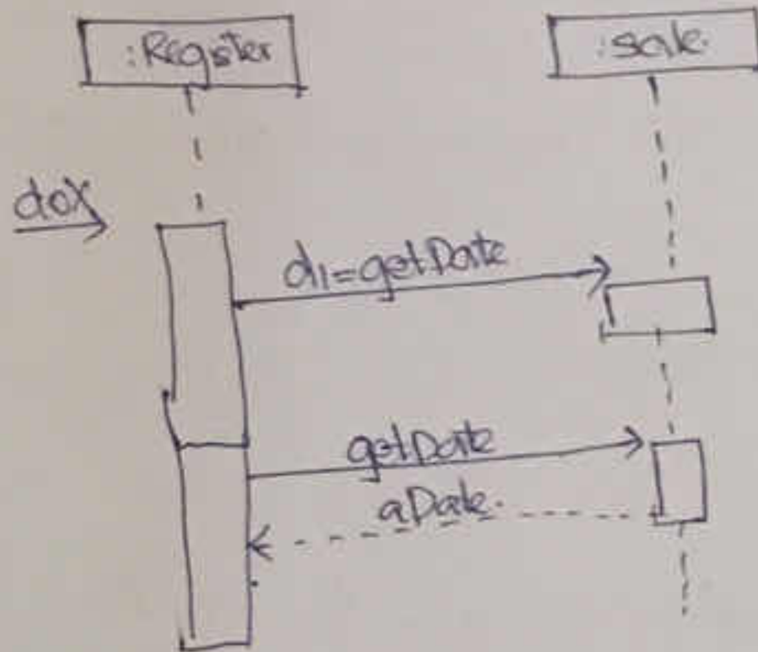
The sender will not be specified.



Reply or Return message.

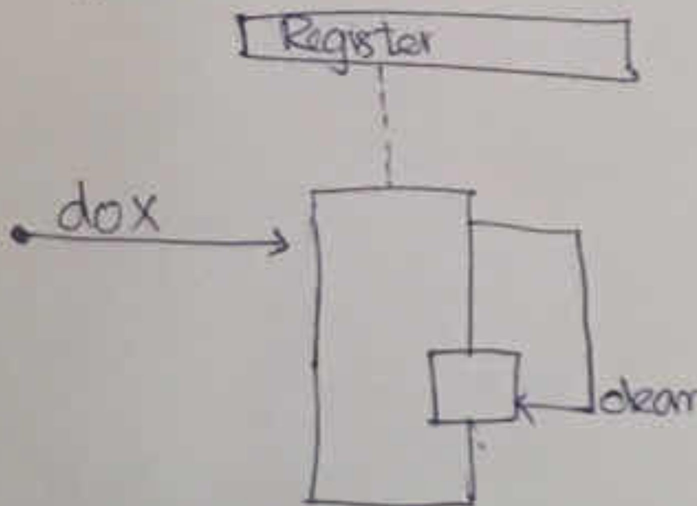
using Message syntax. returnVar = message (param

using a replymessage line at the end of activation bare.



Message to self (or) this.

Message is sent from an Object to itself by using a nested activation bar.



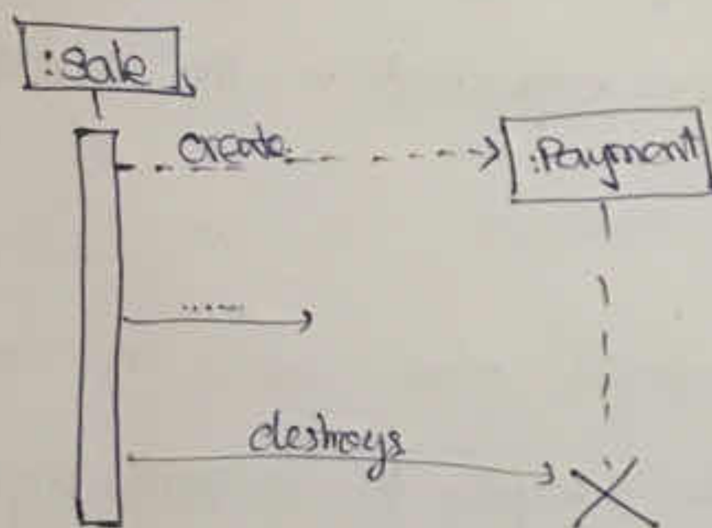
Creation of instances and object Destruction

Creation of Instances:-

objects are placed at their creation heights

Object Destruction

If an object is no longer useful its destroyed.



Frames

* to support conditional and looping constructions
* Frames are regions or fragments of diagram
* They have an operator or label.

Frame operator

Alt → Alternate fragment
Loop → Loop fragment
opt → optional fragment.
Par → parallel fragment
Region. → Critical region.

Alt frame.

Alternate fragment for mutual exclusion conditional logic expressed in the guards

Loop frame

Loop fragment while guard is true. can also write loop(n) to indicate looping n times.

The specification will be enhance[illegible] to define a for Loop, such as loop(i,[illegible]

opt frame.

Optional fragment that executes if guard is true.

Nesting of frames

* Frames can be nested.
* Frames within a frame.

Relating Interaction Diagram.

An interaction occurrence is a reference to an interaction within another interaction.

To invoke static

The classes class and Type are metaclasses, which means their instances are themselves classes.

Asynchronous & Synchronous.

Does not wait for a response.

is a Stick arrow message; regular Synchronous calls are shown with in a filled arrow.

Link message

A Link is connection path between. two objects.

it indicates some form of navigation and visibility between visibility between the objects

Message to 'self' or 'this'

A message can be sent from an object to itself.

Creation of instance.

The message may be annotated with an UML stereotype, like <<create>>

Message Number Sequencing

The first number is not numbered. Thus msg is unnumbered.

The order and nesting of subsequent messages is shown.

Conditional Messages

A conditional message by following a sequence number with a conditional.

Similar to an iteration clause.

Mutually Exclusive

Modify the sequence expression with conditional letter path.

The first letter used is by a convention.

Both are sequence number 1 since could be.

Iteration or Looping

An simple * can be used.

Message to a classes to Invoke static.

method call

Asynchronous & synchronous call

Asynchronous are shown with a

stick arrow.

Synchronous calls with a filled arro

System Sequence Diagram

Guidelines

Draw an SSD for a main sucess

of each use case.

System behaviour is the description.

of what a system does without explainin

how it does

Types of Event in SSD

* External Event
* Fault event
* Timer Event

SSD in UP Phases

Inception

SSD are not motivated. in inception involves only cost estimation of identifying System operations.

Architecture Types

Strict Layered Architecture

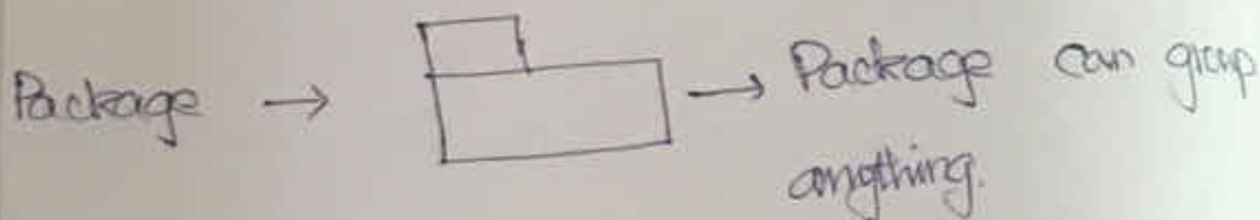
a Layer only calls upon. the services of the directly below it.

This common design in network. protocol stacks

Relaxed layered architecture.

A higher layer calls upon several layer.

The UI layer may call upon its directly subordinate application logic layer.



Dependency → - - - - - -> depend on package

Fully qualified Name Java::util::Date → To represent a namespace.

When to use package diagram?

It is used large scale system. to picture dependencies between major elements

UML Deployment and component digiam.

Deployment Diagram

The assignment of concrete software artifacts.

It shows the deployment of software elements to the physical architecture and communication

Device Node:-

This is a physical computing Resource representing a computer with memory or mobile

Execution Environment Node

This is software computing. resource that runs within an outer node and. provides a service

* an operating system (os) is software that hosts and executes programs.

* a workflow engine

* A sorlet containner or EJB container.

Deployment Diagram.

Component Diagram

A component represent a modular part of the system. that encapsulates its comments and whose manifestation is replacable within its renvironment

Features of UML component

1) it has interfaces

2) it is modulare

=> Self contained.

=> replaceable.

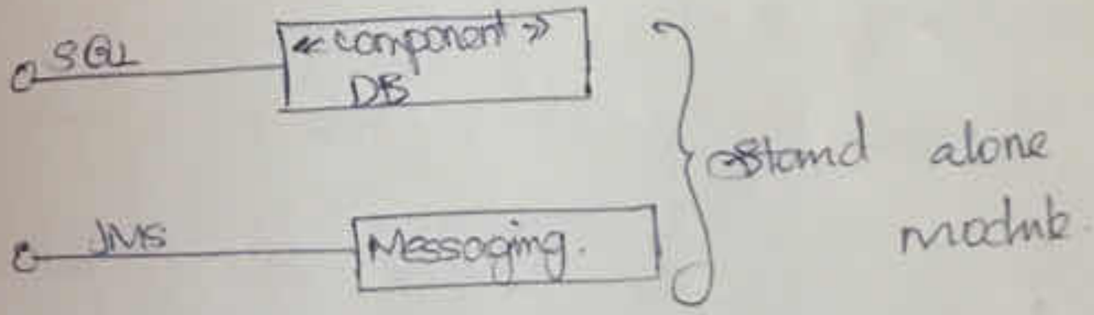
Example

A good analogy for software component modeling is a home entertainment System.

They are modular, self contained, replacable and work.

Elements

Name: component with provided Interface.



Dependency.

- - - - - ->

System getting services from component

When to use component diagram:

component diagram are used to Visualize the static implementation view of a system.

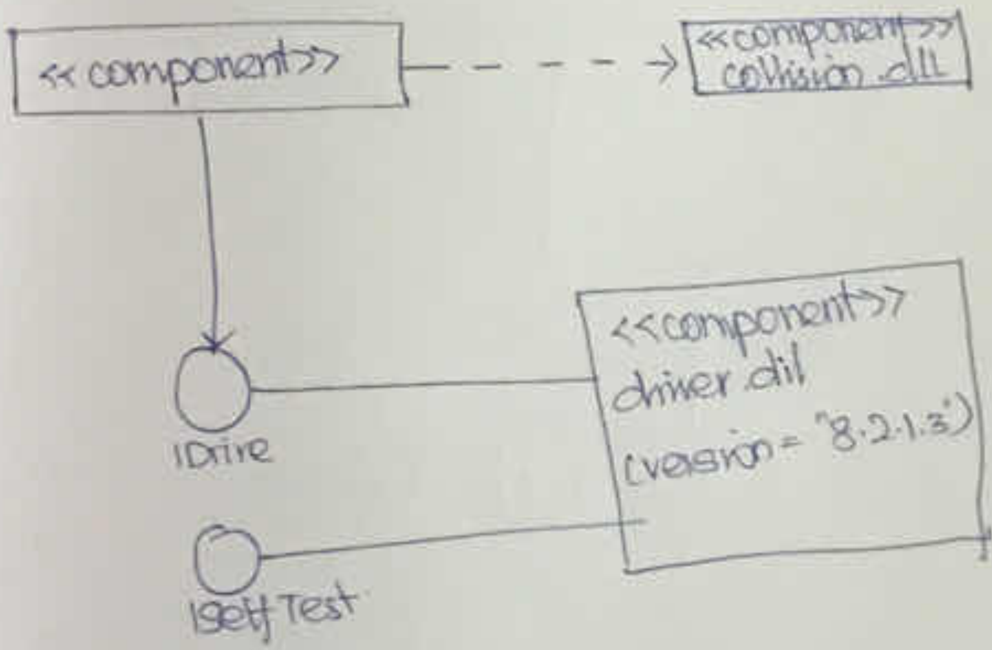
* Model the components of a system.
* Model the database schema.
* Model the executables of an applica
* Model the system's source code

Modelling Source Code:-

* Either by forward or Reverse engineering
* For Larger systems use packages to show groups.
* consider exposing a tagged value indicating such information as the version

Modeling on Executable Releses.

* identify the set of components of model
* consider the sterotype of each component
* For each component in the set, relate with its neighbours.

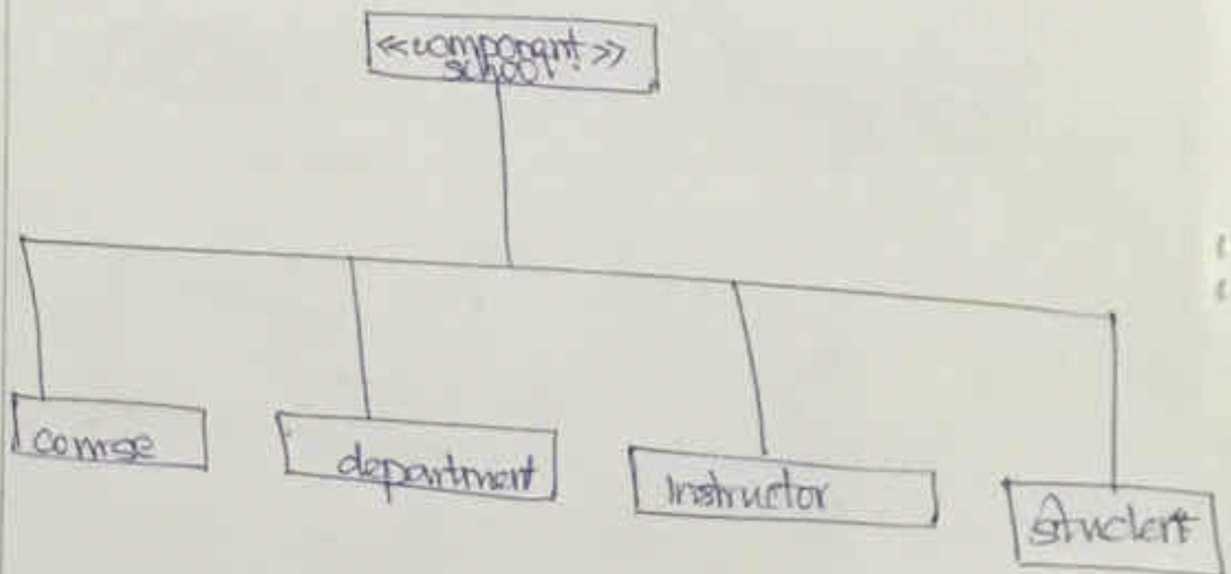


Modeling a physical Database

Identify the classes in the model that represent the Logical database schema.

select a strategy for mapping these classes to tables

When possible, use tools to help you transform Logical design into a Physical Design.



UNIT-4.

Low Coupling Principle.

coupling is a device to connect parts of machinery.

It assign responsibilities so that coupling remains low.

Coupling is a measure of how strongly one class is

- connected to
- has a knowledge of
- relies upon other clases.

coupling between classeses is dependency of one class on another class.

class A has an attribute that refers to class B.

Low coupling.

A creator pattern suggests Register should create payment.

A register records a payment in real world.

Register is coupled to both sale and payment.

Assuming that the sale must eventually be coupled to knowledge of a payment having sale create the payment does not increase

A class with low cohesion does too much unrelated work.

Hard to comprehend

Hard to reuse.

Hard to maintain.

Delicate and constantly affect by change

Cohesion → act of sticking together tightly.

Redmood cohesion of Register

Low cohesion.

Register is taking part of responsibility for fulfilling " make payment operation

it will become burden with tasks and become incohesive

Delegate the payment creation responsibility to Sale the support high cohesion

If a program receive events from external sources other than its graphical interface

Assign the responsibility to class that represent one of following options

Represent the overall system or root object.

1. Suitable when there are not too many events or when UI cannot choose between multiple controllers

2. A controller for each use case

eg:- process Sale Handler.

Register: is specialized device with software running in it

Register:-

specilized device with software running in it

Process Handler

Represents a receiver of all system events of use case scenasio

Design patterns

A pattern is a recurring solution to a standard problem.

A pattern describes a problem of which occurs over again & again.

Dress patterns

Finding appropreate objects.

Determining object granlality.

Specifying object interface.

specifying object implementation

Programming to an interface.

Elements of Design patterns.

pattern name.

Problem

Solution

Consequences

Types of patterns

1) creational Pattern.

2) Structural Pattern.

3) Behavioral pattern.

CREATIONAL PATTERNS.

Abstract factory.

Factory for building related objects.

Builder

Factory for building complex object incrementally.

Factory Method.

Method is a derived class creates associations.

Prototype.

Factory for cloning new instances from a prototype.

Singleton.

Factory for a singular instance.

Structural Pattern

Adapter:-

Translator adapts a server instance for a client

Bridge.

Abstraction for binding one of many implementation.

Composite.

Structure for building recursive aggregatio

Decorator:

Extends an object transparently.

Facade

simplifies the interface.

Flyweight

Many fine grained objects shared efficiently.

Proxy.

one object approximates another

Observer.

Depends update automatically which subjects

chain of Responsibility

Request delegate to the responsible service povider.

Command

Request or action is first class object.

Iterator

Language interpreter for a small grammer.

Mediator

Co ordinates interactions between its associations.

Momentor

Snapshot captures and restores object states.

GOF design pattern categories

What are creational patterns.

Design patterns that deals with object creation.

Make a system independent of the way in which objects are created.

Recurring themes.

Encapsulates knowledge about wh concrete classes the system uses.

Hide how instances of these classes are created and put together.

Benefits of creational patterns

creational patterns let you prog to an interface defined by an abstract

Example

* Interview GUI class library
* Multiple Looks
* Abstract Factories

Benefits of creational patterns

Generic instantiation

Objects are instantiated without having to identify a specific class type.

simplicity.

Make instruction easier: caller do not have to write large complex code.

Creation constraints

can put bounds on who can create objects, how they are created and when they are created.

Factory Method

TO create object as Template method to implement an algorithm

Makes design more customizable and only a little more complicated.

Specified by an architectural framewor[illegible] implemented by user of framework.

Definition

Product Defines the interface of objects.

Concrete Product

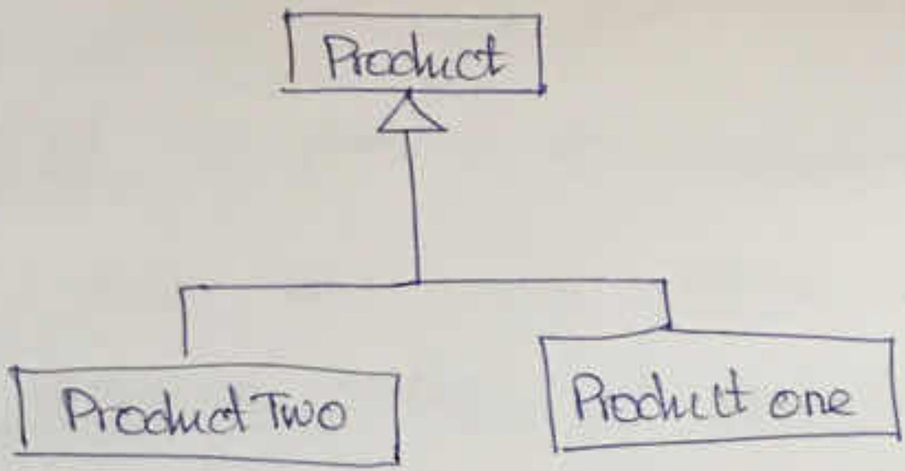
implements the product interface

Creator

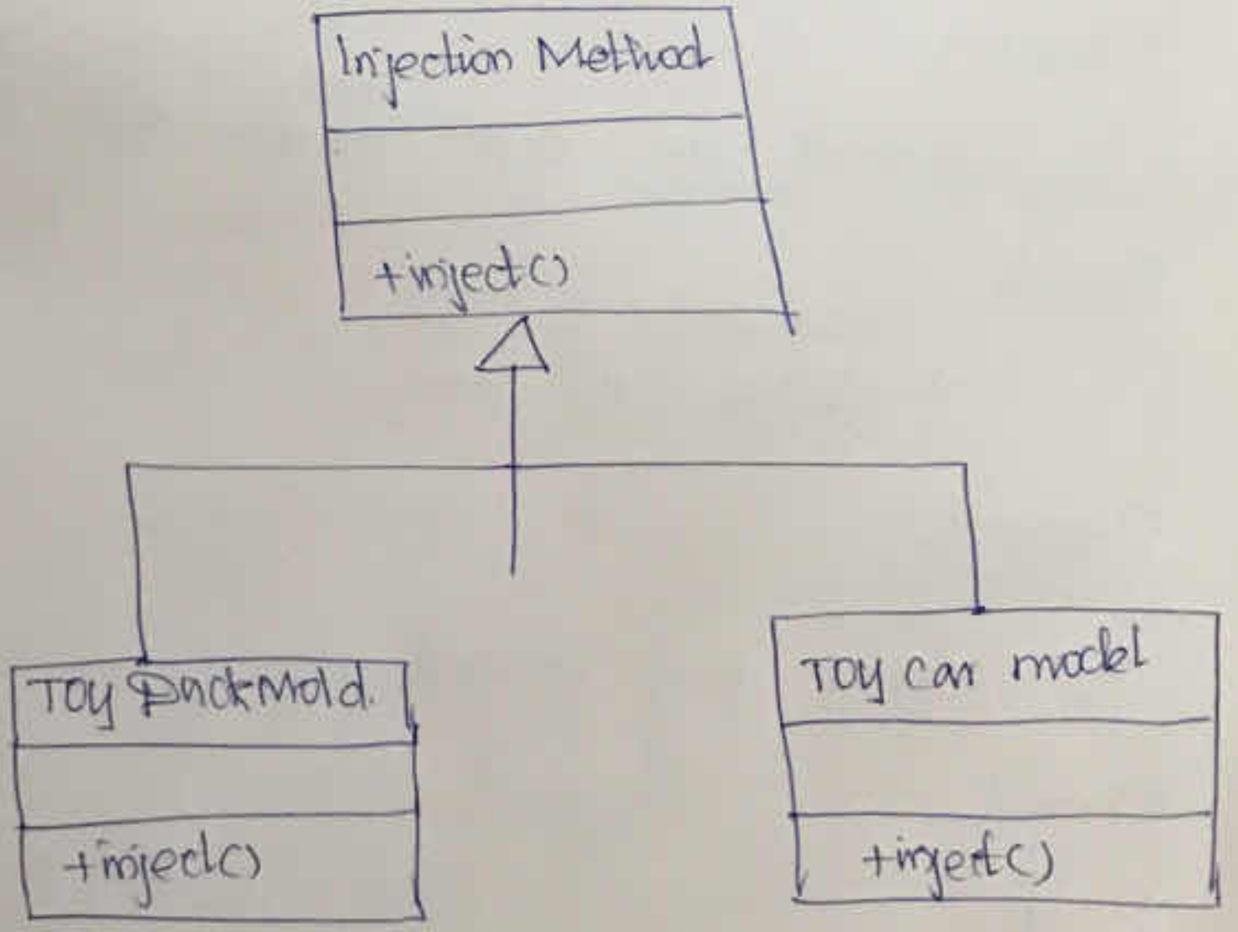
declares the method. May call the generating method for creating product object.

Concrete Creator

overrides the generating method for creating concrete objects



Example:



Structural Pattern

Adapter

Translator adapts a server interface.

Bridge

Abstraction for binding one implementat

compositor

Structure for building recursive aggregations.

Decorator

Extends an object transparently.

Facade

Simplifies the interface.

Flyweight

Many fine grained objects Shared.

Proxy

one objects approximate othes.

Adapter

context / problem

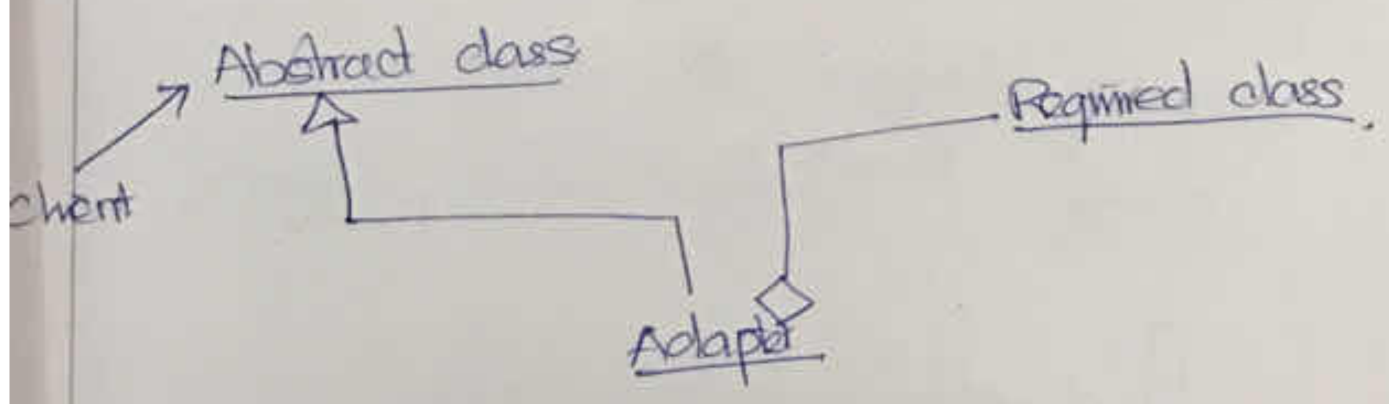
How to provide a stable interface to similar components?

Solution

Convert the orginal interface of a component into another interface through an intermediate adapter object.

Adapter pattern

Adapters use interfaces and polymorphism to add a level of indirections.



Adapter Example

Target:-

Defines the domain specific interfaces

Adapter

Adapts the interface Adaptee to the Target

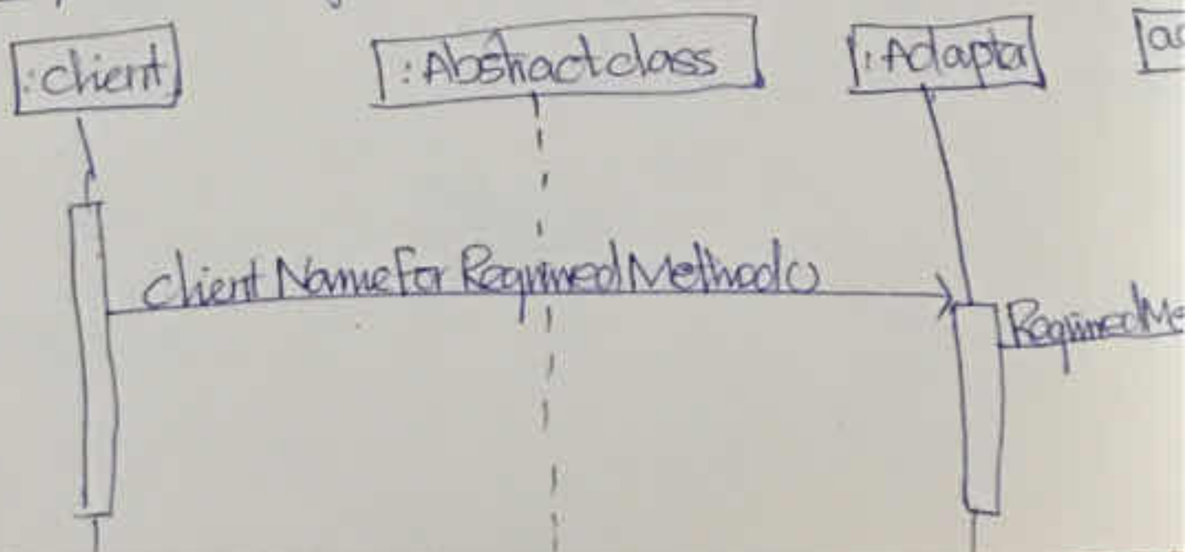
Adaptee

Defines an existing interface that needs adapting.

Client.

collaborates with objects conforming to the Target interface

Sequence Diagram for adapter



Behavioral Pattern.

- Describes algorithm assignments of responsibilities and interaction between objects
- Behavioral object pattern use composition

chain of Rep.

A way of passing a request between a chain of projects

Avoid coupling the sender of request.

command.

Encapsulate a command request as an object

Encapsulate a request as an. object

Interpreter

A way to include language elements

Iterator

Sequently access the elements of a collection Provide a way to access the elements of an aggregate

Memento

Capture and restore an objects' internal State.

Observer.

A way of notifying change to a number of classes.

State.

Alter an object behavior when its State changes.

Strategy:

Encapsulate an algorithm inside a class Defines a family of algorithms

Example.

Name	Strategy
Problem	How to design for varying, but related, algorithm or policies?
solution	Define each algorithm /policy/ strategy in a Separate class, with a common interface

creating a Strategy with a factory

There are different pricing algorithm or Strategies. and they change over time who should create the pattern.

The major ideas and steps in Example

An interface is defined

Define the window to implement the interface.

When the SaleFrame / window is initialized.

The saleframe window registers

or subscribe.

The Sale does not know about

sale-frame

The sale instance is thus a

publisher of property events.

Implementation in an object Oriented Language

requires writing source code.

1) create the fixture.

2) Do Something.

3) Evaluate the results are as

expected.

Object Oriented Methodologies

set of methods, models and rules for developing systems.

can be done during any phase of Software Life cycle.

. A model is an abstraction of a phenomeon for the purpose of understanding.

Types

1) The Rumbaugh.
2) The booch methodology
3) Jacobson's methodology

A methodology is explained as the science of methods

A method is a set of procedures specifyies in step by step.

Advantages / characteristics

The Rumbaugh is well suited for describing the object model or static structure.

The Jacobson method is good for producing user driven analysis method.

OMT consists of four phases, which can be performed

1) Analysis:- The results are objects.
2) System design
3) Object design
4) Implementation

OMT #4 Parts.

1) An object model.
2) Dynamic model.
3) Domain model
4) Functional Model

OMT object Model.

* The object Model describes the structure of objects
* Identify, relationship to other object
* object diagram contains classes interconnected by association lines

OMT Dynamic Model

OMT dynamic model depicts states, transitions events and actions.

OMT state transition is a network of states and events.

Each state receives one or more events

OMT Functional Model.

=> DFD use form primary symbol.

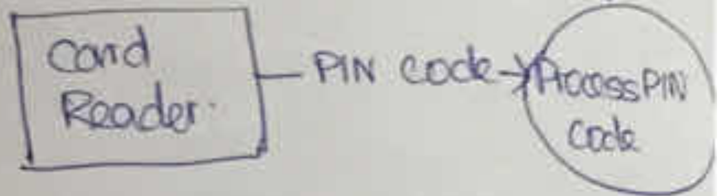
* Process
* Data flow.

* Data store.

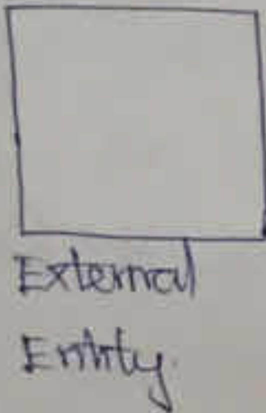
* External entity

Data
Store

process



Data flow.



External
Entity.

The Booch Methodology.

It is widely used object oriented method that helps us to design the System using object paradigm

The booch method consists of

- class diagram.
- object diagram.
- state transition Diagram.
- Module diagram.
- Process diagram.
- Interaction Diagram.

Patterns and the various Pattern Templates.

⇒ Identifies a common structure that makes it useful for design.

⇒ help document software architecture.

⇒ describes software abstractions.

Patterns do not

Provide an exact solution.

Solve all design patterns

Only apply for object oriented design.

Generative Patterns.

Tell us how to generate something.

* observed in a system.
* Descriptive and passive.

Non generative patterns:-

Ogenerate systems or parts of the system.

Operspective and active.

Patterns templates

Name:-

A meaningful name allows us to use a single word. or short phrase.

Problem:-

A statement of the problem that describ its intent.

A force oppose these objectives

Context.

The preconditions under which the problem

Forces

A description of the relevant forces and how they interact or with one another and with the goals.

Solution.

Static relationships and dynamics rules describing how to realize the outcome. It describes how to construct the product.

Example

One or more sample applications of the pattern than a specific initial context.

Resulting Context.

The state or configuration of the system after the pattern has been applied.

Frame works

Frame works are a way of delin[...]
application development patterns.

The major differences between design patterns [...]

* Design Patterns are more abstract than frameworks.
* Less specialized.
* Smaller Architectural elements

The Unified Approach:

Establishes a unifying and unitary framework by utilizing UML.

→ use case driven development.

→ object Oriented Analysis.

→ object Oriented design.

→ continuous Testing.

Object oriented design.

* Design classes.
* Designs the Access Layer.
* iterate and refine the design.

Continuous testing.

UML- modeling Language.

Repository:

* Allows maximum reuse of previous experiance.
* should be accessible by many people.

Quality Assurance Tests.

Debugging is a process of finding out where something went wrong and correcting the code.

Kinds of errors

1) Language
2) RUN time.

Testing strategies:-

The objective of s/w testing is to uncover errors

UNIT TESTING:-

Black box testing, white black

Integration Testing

Top down testing, bottom up testing

Regression testing.

Systematic technique for constructing the program.

Top down testing.

It assumes that main logic or object interaction and system messages of application.

Bottom-up testing

It starts with details of system.

<u>Stress testing</u>

Executes a system in a manner that demands resources

In some situations, a very small range of data contained within the bounds of valid data.

<u>Performance testing</u>

To test run-time performance of s/w within context of an integrated system.

<u>Impact of object orientation on Testing</u>

1) Some type of errors could become less plausible.

2) Some type of errors should become more plausible.

3) Some new types of error might appear.

Develope Test case and Test plans.

Develop Test

Myers describes the object of testing

* Testing is a process of executin[g] a program with the intent of findin[g] errors

Guidelines for developing Quality Assu[rance]

* Test abnormal but reasonable use of object's method

* Test boundary conditions of number of parameters

* Drop down the file menu and select open.

* A file type that the progra[m] does not open

Develope Test case and Test plans.

Develop Test.

Myers describes the object of testing

* Testing is a process of executin[g] a program with the intent of findin[g] errors

Guidelines for developing Quality Assur[ance]

* Test abnormal but reasonable use of object's method

* Test boundary conditions of number of parameters

* Drop down the file menu and select open.

* A file type that the program does not open

Develope Test case and Test plans.

Develop Test.

Myers describes the object of testing

* Testing is a process of executin a program with the intent of findin errors

Guidelines for developing Quality Assu

* Test abnormal but reasonable use of object's method

* Test boundary conditions of number of parameters

* Drop down the file menu and select open.

* A file type that the progra does not open