2069 Bhadra

1. ans Encapsulation is one of the loosely defined OOAD concepts. The term is known in software development for many years. but 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.

Encapsulation is the process of compartmentalizing (classifying) the elements of an abstraction that constitute its structure and behavior, encapsulation serves to separate the contractual interface of an abstraction and its implementation.

for example: A relational database is encapsulated in the sense that its only public interface is a Query language (SQL for example) which hides all the machinery and data structures of the DBMS.

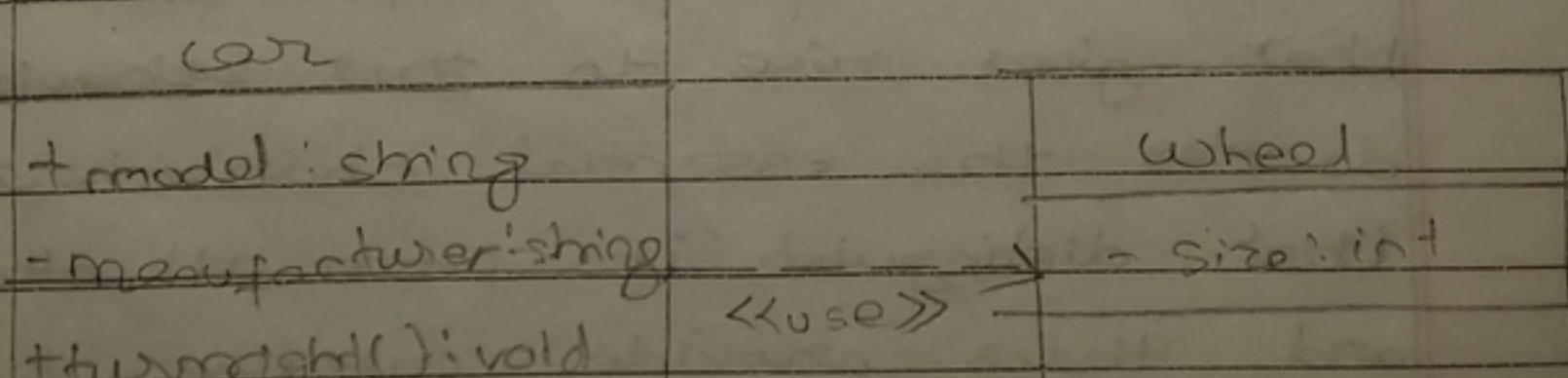
Encapsulation and abstraction works together in object-orientation. Abstraction and encapsulation are complementary concepts: abstraction focuses upon the observe behavior of an object, whereas encapsulation focuses upon the implementation that gives rise to this behavior. An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the

viewer. Encapsulation is the mechanism used to hide the data, internal structure and implementation details of some element, such as an object or subsystem. Abstraction is simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others. An abstraction focuses on the outside view of an object, and so serves to separate an object's essential behavior from it's implementation.

Deciding upon the right set of abstractions for a given domain is the central problem in OOD. Abstraction is thought process whereas encapsulation is implementation process.

3. ans Dependency:

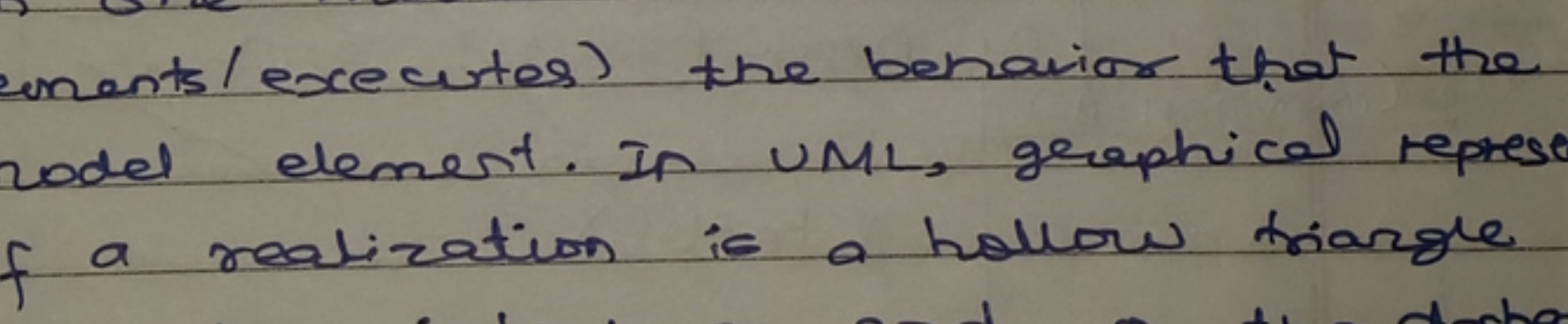
It's relationship between two things in which change in one element also effects other. Dependency is a weaker form of a relationship which indicates that one class depends on another because it uses it at some point in time. One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.



Association :

It's a set of links that connects elements of an UML model. An association represents a family of links. Binary associations (with two ends) are normally represented as a line. An association can be named and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility and other properties.

Types of association are: uni-direction, bi-directional, aggregation and reflexive.

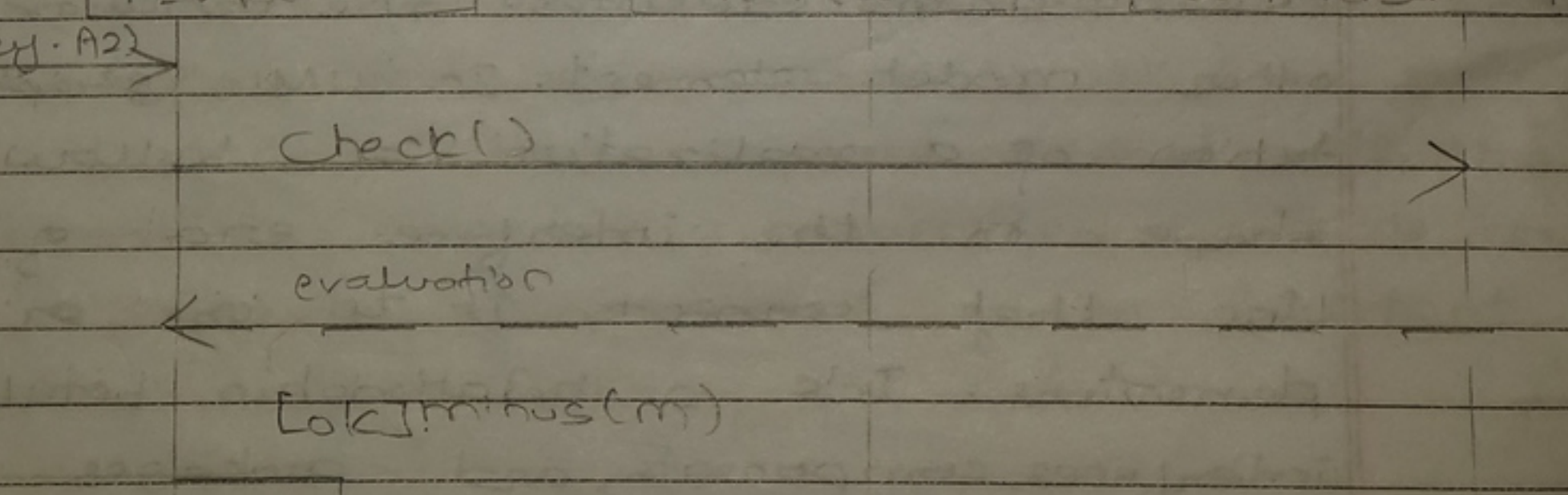


Realization :

In UML modelling, a realization relationship is a relationship between two model elements in which one model element (the client) realizes (implements/executes) the behavior that the other model element. In UML, geraphical representation of a realization is a hollow triangle shape on the interface end of the dashed line that connects it to one or more implementers. It's a relationship between classes, interfaces, components, and packages that connects a client element with a supplier element.

Generalization:

It indicates that one of the two related classes (the subclass) is considered to be a specialized form of the other (the super type) and superclass is considered as 'generalization' of subclass. The UML graphical representation of a Generalization is a hollow triangle shape on the superclass end of the line that connects it to one or more supertypes. The superclass is also called as parent class and subtype as child class.



4. ans

a) Two classes are involved in this transfer activity, viz. account and person.

b) The events followed for transfer function are check, evaluation, minus and plus.

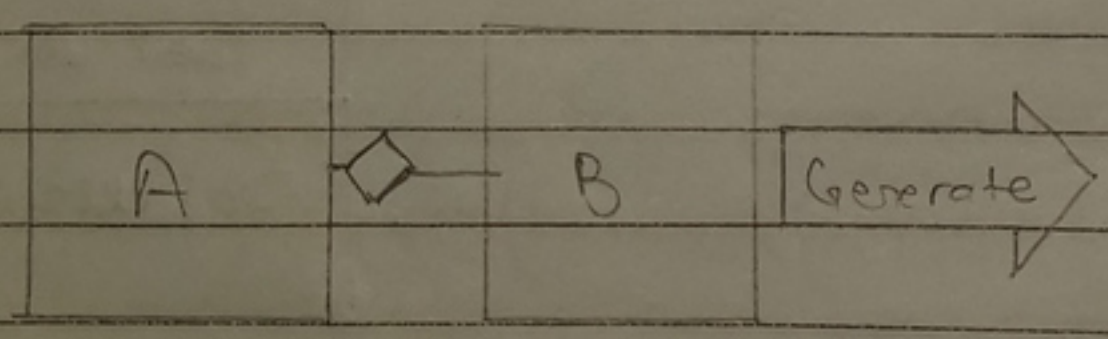
c) Check() function is required for transfer function so that only valid and acceptable transfer of money can be transferred from one account to another (A1. account to A2. account.

forward button/ increase time

5. ans

6. What is forward engineering:

The traditional process of moving from high-level abstractions and logical, implementation independent designs to the physical implementation of a system. It implies converting model into software code. The designs illustrated in UML diagrams will be incomplete, and serve as a "springboard" to the programming. Diagramming before programming relies on the function of the experience and cognitive style of the designers. Some people are very spatial/visual thinkers, and expressing their software design thoughts in a visual language complements their nature; others aren't. A large percentage of the brain is dedicated to visual or iconic thinking and processing rather than textual processing (code). The diagrams or designs drawn in UML or case tools are the implemented to coding. This process of implementing of design to code is called forward engineering.

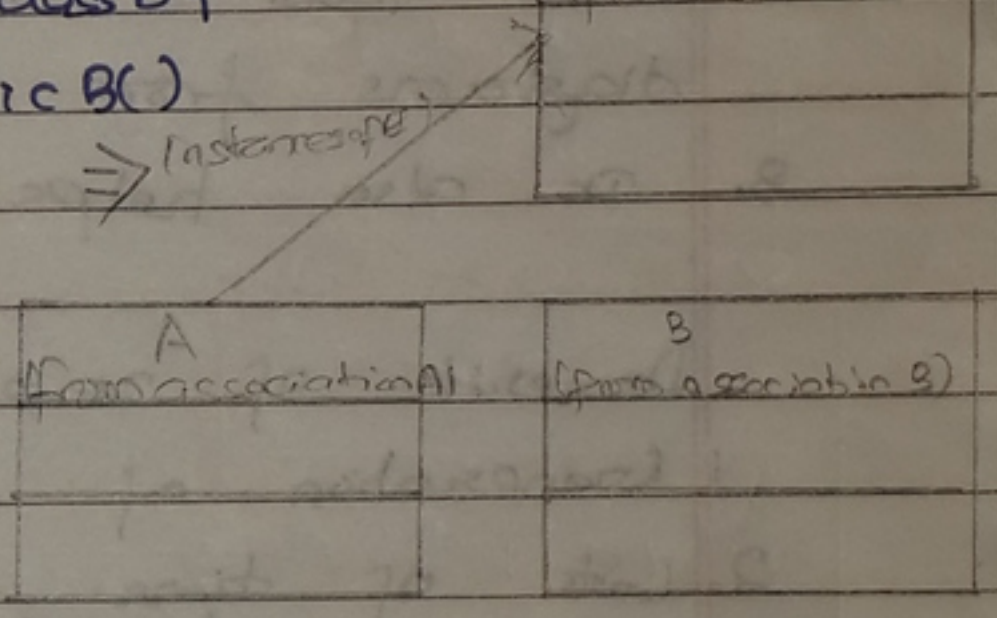



```
public class A{
    private B instance s[]
        fB[];
    public A()
    {
    }
}
public class B {
    public B()
    {
    }
}
```

Reverse engineering:

It means generation of UML diagrams from code. It's the reverse of forward engineering. In reverse engineering process the programmed code is used to generate the UML diagrams. Reverse engineering of object oriented code presents a unifying framework for the analysis of object oriented code.

```
public class A{                          public class B{
private B instances() {B[];
publicA()                                publicB()
{                                        {
}                                        }
}                                        }
```



It aims at supporting program comprehension by exploiting the source code as the major source of information about organization and behavior of the program, and by extracting set of potentially used views provided to programmers in the form of diagrams.

Merits of forward engineering:

1. Once the design is created, it's easy to code.
2. It also saves time during coding.
3. It's a time saving process for generation of code using UML diagrams.
4. Since UML provides overall system, it's easier to generate code for the system.

Demerits of forward engineering:

1. Since code is generated using case tools, it always doesn't guarantee correct code.
2. Code is dependent on UML diagrams.
3. Developing UML diagram is time-consuming.

Merits of reverse engineering:

1. It's time saving process of generating UML diagrams since we have code already with us.
2. Case tools make easy to generate design or UML diagrams from code.
3. It also helps in reducing the cost.

Demerits of reverse engineering:

1. Generation of design from code is difficult.
2. Lots of time is spent on analyzing code.
3. Case tools always doesn't guarantee correct design.

8. Write short notes on:

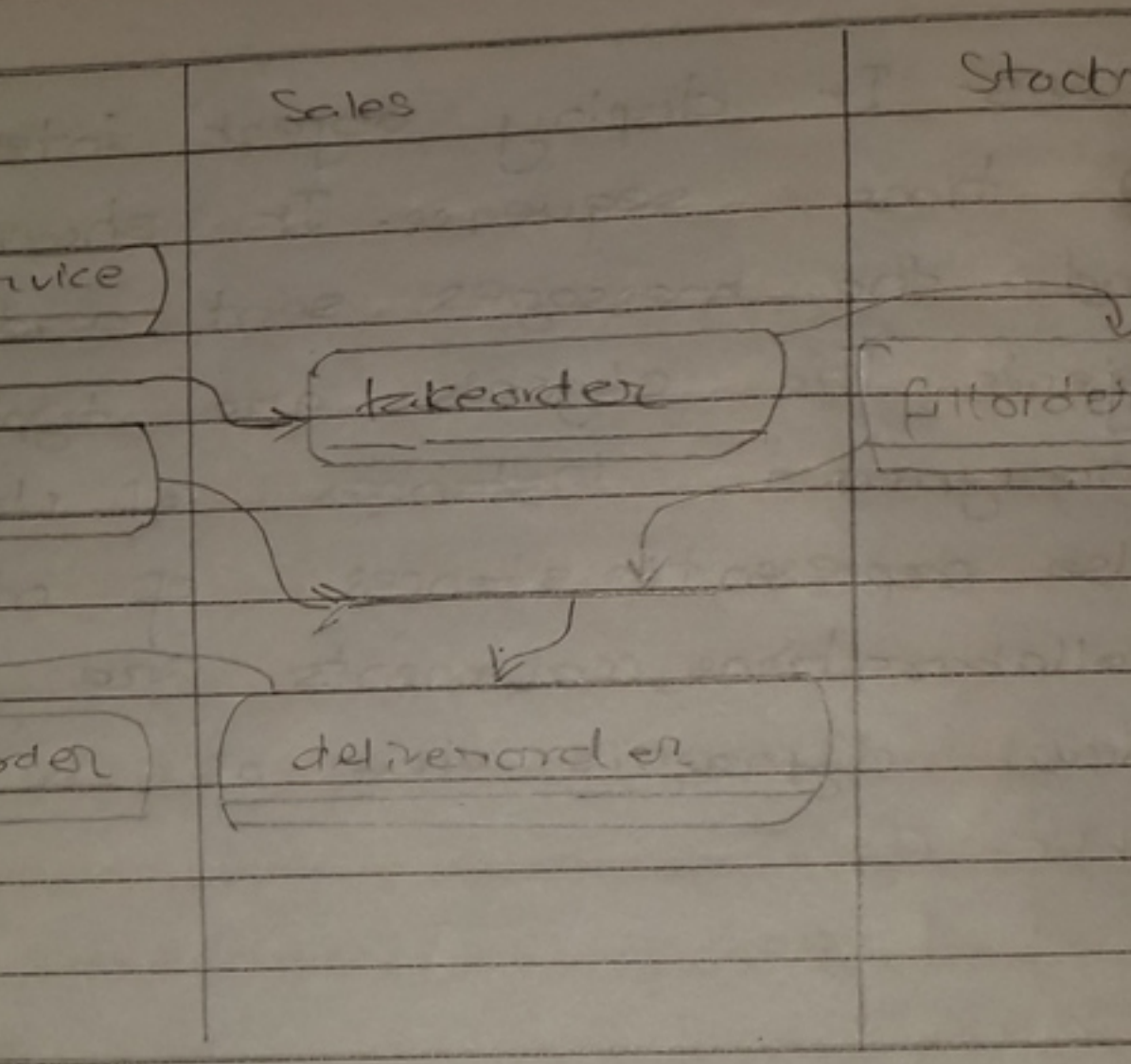
a) Sequence diagram:

It models the flow of control by time-ordering depicts the interaction between various objects by of messages passed, with a temporal dimension to it. It shows a particular sequence of messages exchanged between a number of objects. It also show the behavior by showing the ordering of message exchange. It shows some particular communication sequences in some run of the system. It doesn't characterize all possible

runs. It display object interactions arranged in time sequence. It shows a set of objects and the messages sent and received by those objects. The objects are typically named or anonymous instances of classes, but may also representin stances of other things, such as collaborations, components and nodes. It's used to view dynamic view of a system.

b) Swim lanes:

It represents the column in activity diagram to group the related activities. These are represented in form of partitioned region. They are useful when modelling business workflow because they can represent organizational units or roles within a business model. They're very similar to object because they provide a way to tell who is performing a certain role. When used to diagram a business process that involves more than one department, swimlanes often serve to clarify not only the steps and who is responsible for each one, but also how delays mistakes or cheating are mostly like to occur.



c) Polymorphic signal:

It's a signal delivered at run time to a specific state machine for a class in a generalization hierarchy. The polymorphic signal must be able to be received in every branch in the superclass's hierarchy, so that the polymorphic signal occurence always has a receiving state machine instance. When a supervisor tells a clerk to go off-duty, the supervisor doesn't care whether the clerk is a shipping clerk or a stock clerk. Similarly when an actions sends a signal to an object in a class hierarchy, the sender shouldn't need to know the subclass of the object. Rather, the signal is polymorphic. To keep distinction between signal and event we have polymorphic event which is an event that has many potential receiving state machines in a generalization hierarchy.

2069 Poush

1. one class:

In UML, a class is used to describe characteristics of any entity of the real world. A class, apart from characteristics has some functions to perform called as methods.

example:

A class named "Food" has attributes like 'price', 'quantity'. 'Food" class methods like serve-food(), bill-food().

object:

An object is a pattern of the class. An actual object created at runtime is called an instance.

Inheritance:

The main class or the root class is called as a base class. Any class which is expected to have all properties of the base class along with its own is called as a derived class.

for example:

for the "Food" class, a derived class can be "class Nepalese Food".

Abstraction:

It's creating models or classes of some broad concept. It can be achieved through inheritance or even composition.

Interface:

It's a description of the actions that an object can do.

for example:

When you flip a light switch, the light goes on, you don't care how, just that it does.

Encapsulation:

It's a collection of functions of a class and object. The "Food" class is an encapsulated form. It's achieved by specifying which class can use which members (private, public, protected) of an object.

Polymorphism:

It means existing in different forms. Inheritance is an example of an polymorphism. A base class exists in different forms of derived class. Operator overloading is also an example of polymorphisms which is applied in different situations.

2. The building blocks of UML are:

OR

- Things
- Relationships
- Diagrams

1. Things:

Things are the most important building blocks of UML.

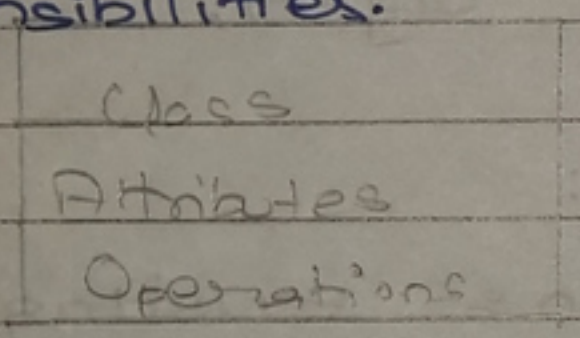
Things are:
a) Structural
b) Behavioral
c) Grouping
d) Annotation

a) Structural things:

The structural things define the static part of the model. They represent physical and conceptual elements.

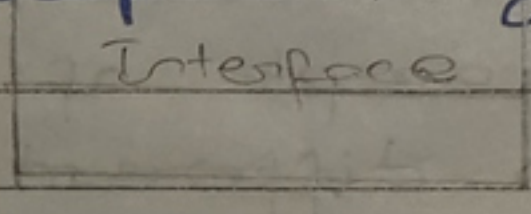
* i. Class:

It represents set of objects having similar responsibilities.



ii. Interface:

It defines a set of operations which specify the responsibility of a class.

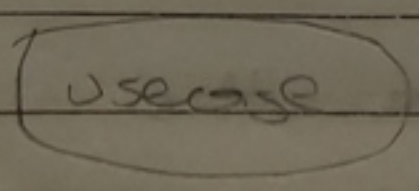


iii. Collaboration:

It defines interaction between elements.

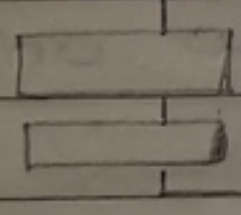
iv. Use case:

It represents a set of actions performed by a system for a specific goal.



v. Component:

It describes physical part of a system.



vi. Node:

It can be defined as a physical element that exists in run time.

b) Behavioral things:

It consists of the dynamic parts of UML models. following are behavioral things:

a) Interaction
b) state machine
c) Annotational things

2. Relationship:

It's another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application. four kinds of relationship:

a) Dependency
b) Association
c) Generalization
d) Realization

3. UML diagrams:

It's ultimate o/p of the entire discussion. All the elements, relationships are used to make a complete UML diagram that represents a system.

3. ans

Customer	View items	Catalog
Name	View unit price	Items
Address		Price

Provides credit card information
Add items
Pays
Confirm Sale
Returns amount

Sale
Total amount

5.

ans.

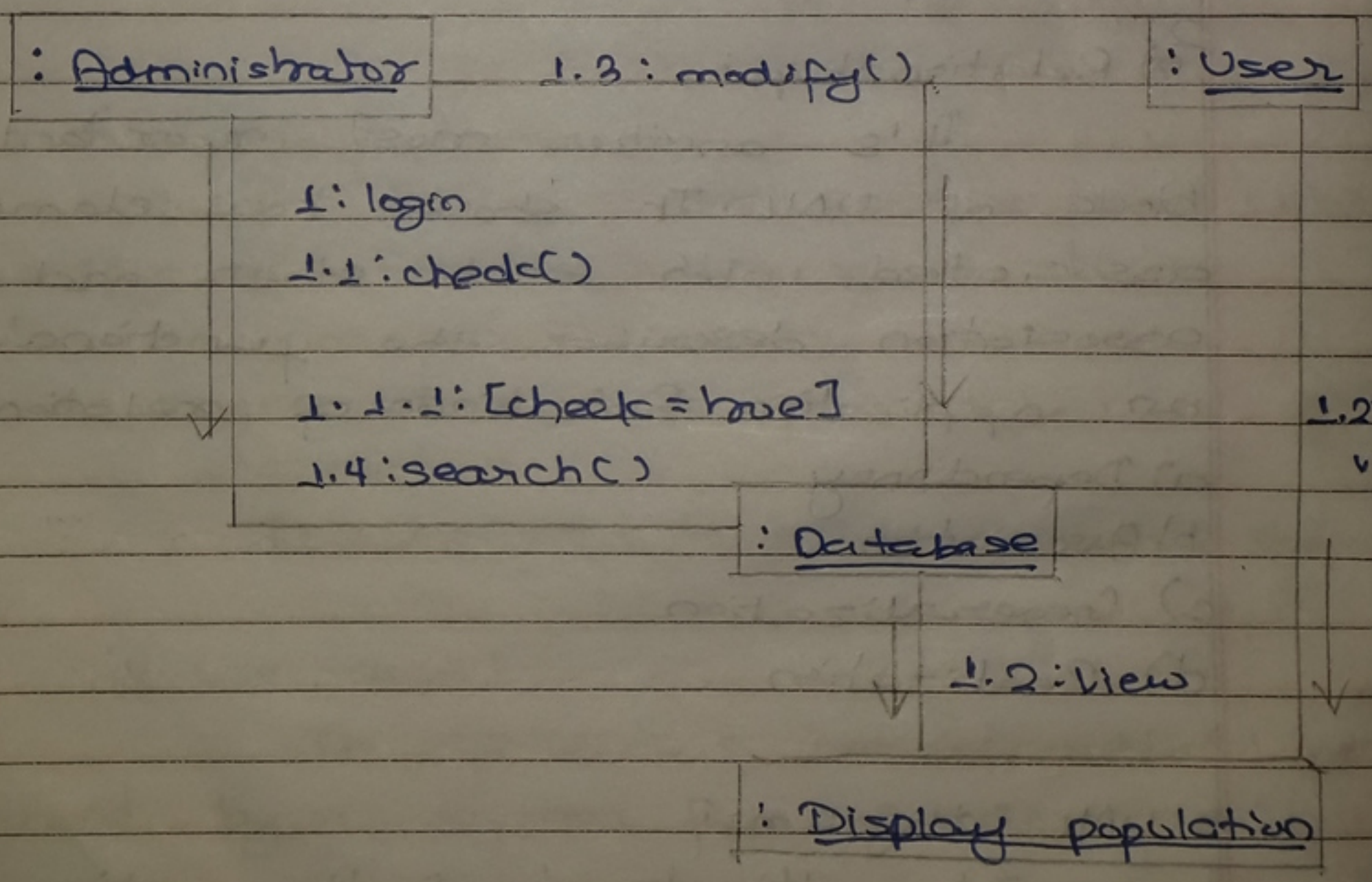


fig. Collaboration diagram

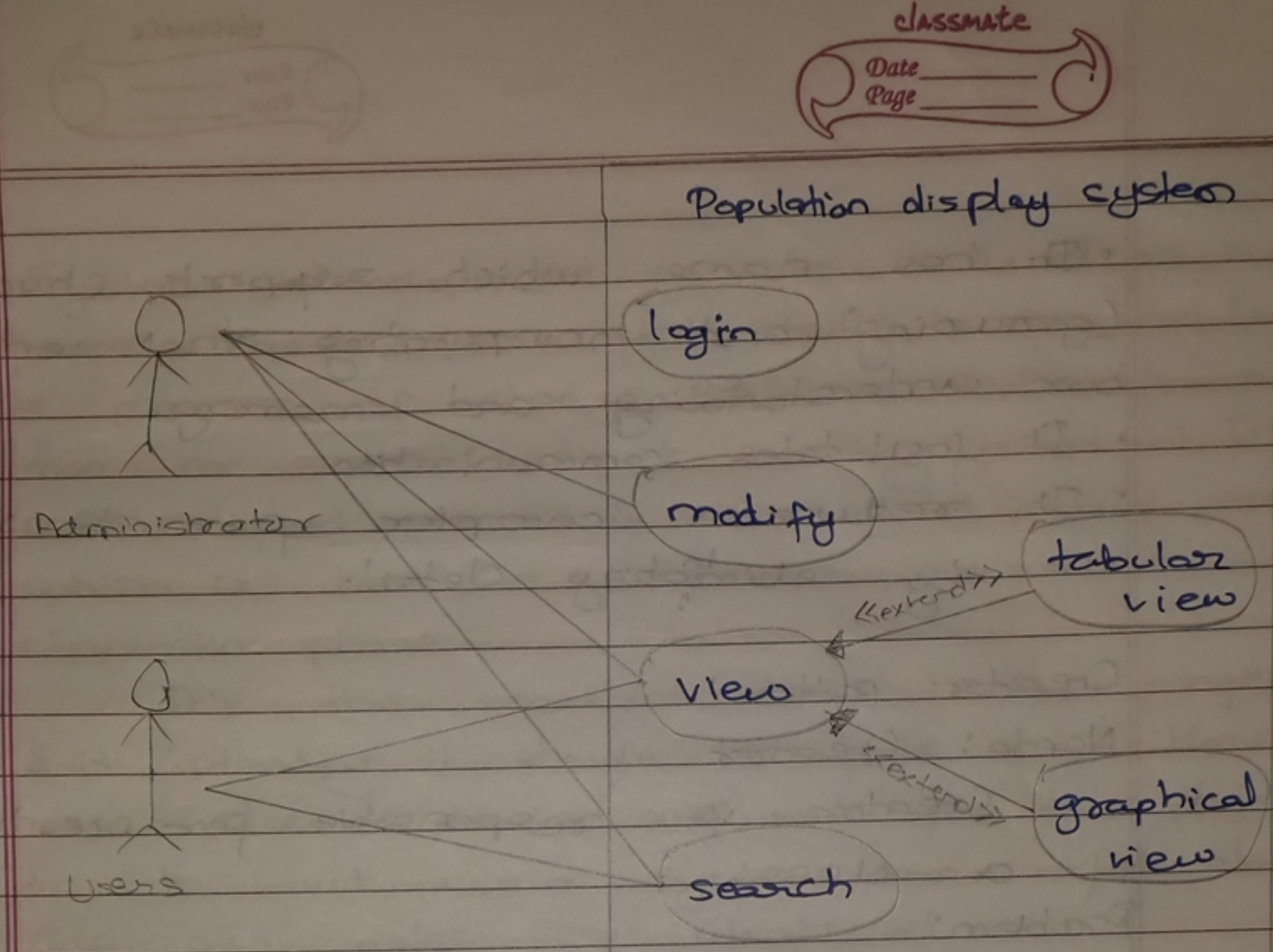


fig. use-case diagram

6. ans Object-oriented frameworks are established tools for domain-specific reuse. Many framework design patterns have been documented. The framework development cycle generally evolves from open framework to closed application. We describe a more flexible component-based approach to framework design that stresses a common interface for plugging in new components at different lifecycle stages.

In OOD, design pattern are useful because:

- Patterns give a specific category of problem, guide the assignment of responsibilities of objects.

- It has name which supports chunking (grouping) and incorporating that concept into our understanding and memory.
- It facilitates communication.
- It reduce a complex form to simple one by eliminating detail.

Creator pattern:
Name: Creator

Creator is responsible for creating object of a class.

Problem:

Who creates an A? (doing responsibility)

Solution:

Assign class B the responsibility to create an instance of class A if one of these is true.

- B contains or aggregates A.
- B records A.
- B closely uses A.
- B has the initializing data for A.

In monopoly game:
Name: Creater
Problem: Who creates square!
Solution: Board is assigned responsibility to create squares.

7. ans

System development can be viewed as a process. The development itself, in essence, is a process of change, refinement, transformation or additional to existing product. In OOAD, unified process is the development process which is iterative and incremental.

- Inception phase:

It's the smallest phase in the project and ideally it should be quite short. Here the documentation required for the project or to built up a system are made clear and also risks are identified.

- Elaboration phase:

In this phase, risk factors are noted out and a proper system architecture is established and validated. Primary tasks in this phase are use-case diagrams, conceptual diagrams and package diagrams.

- Construction phase:

It's the largest phase in SDC where system is built on the basis of last phases.

- Transition phase:

In this phase, the system is deployed to the targetted users and refined, if it's needed as desired by users.

8.

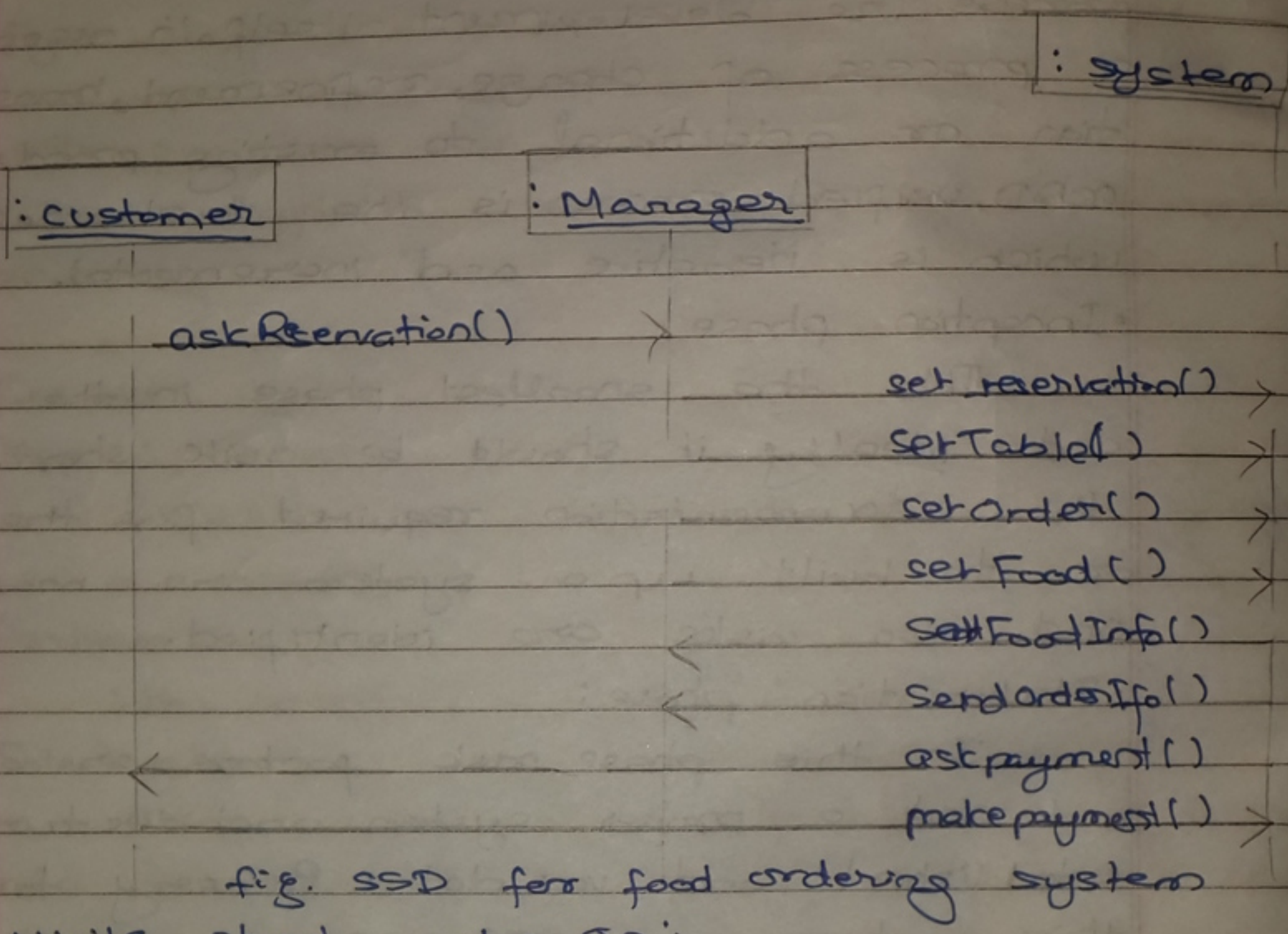


fig. SSD for food ordering system

10. Write short notes on:

a) Iterative cycles of development:

The word "iterative" means that it involves repetition. Iterative development is a development approach that "cycles" through the development phases, from gathering requirements to delivering functionality in a working release. Iterative and incremental development is any combination of both iterative design or iterative method and incremental build model for development. The combination is of long standing and has been widely suggested for large development efforts. During software development more than one ite-

ration of the SDC may be in progress at the same time, and this process may be described as an "evolutionary acquisition" or "incremental build" approach. The whole SDC ~~is~~ consists of systems being integrated from number of increments where there might be repetition of process while developing a system. The unified process groups increments/iterations into phases: inception, elaboration, construction & transition.

b) Synchronization bar:

A synchronization bar helps illustrate parallel transitions used in activity diagram. Synchronization is also called forking and joining.

During consolidation synchronization takes place, meaning the flow proceeds only after all incoming flows have reached the consolidation point. Join has two or more inputs and one output.

2070 Bhadra

1. Ans

Algorithmic decomposition is also known as functional decomposition of the system. It's the decomposition of system in functional parts. It's good in a procedural programming environment. It's even useful for understanding the modular nature of a large-scale application.

Object-oriented decomposition is also known as class-decomposition of system. It's the decomposition of a system into class and objects. It's good in a structural programming environment.

Algorithmic decomposition is done by function-oriented programming languages like C whereas ~~functi~~ object-oriented decomposition is done by structural-oriented programming languages like C++.

3. ans Behavior modelling in object-oriented analysis

Diagrams based on behavioral modelling are as follows:

1. Activity diagrams:

They're graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.

2. Interaction diagrams:

It is a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modelled. for example, the sequence diagram which

shows how objects are ~~intended~~ communicate with each other in sequence of messages.

9. Use-case diagrams:

It's representation of a user's interaction with the system and depicting the specification of a use-case. It can portray the different types of users of a system and the various type that they interact with the system. This is type of diagram is typically used in conjunction with the textual use-case and will often be accompained by other types of diagram as well.

A. State machine diagrams:

A state diagram is a type of diagram used in computer science and related fields to describe the behavior systems. State diagrams require that the system described is composed of a finite number of ~~states~~, sometimes. this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics.

8. ons Branching:

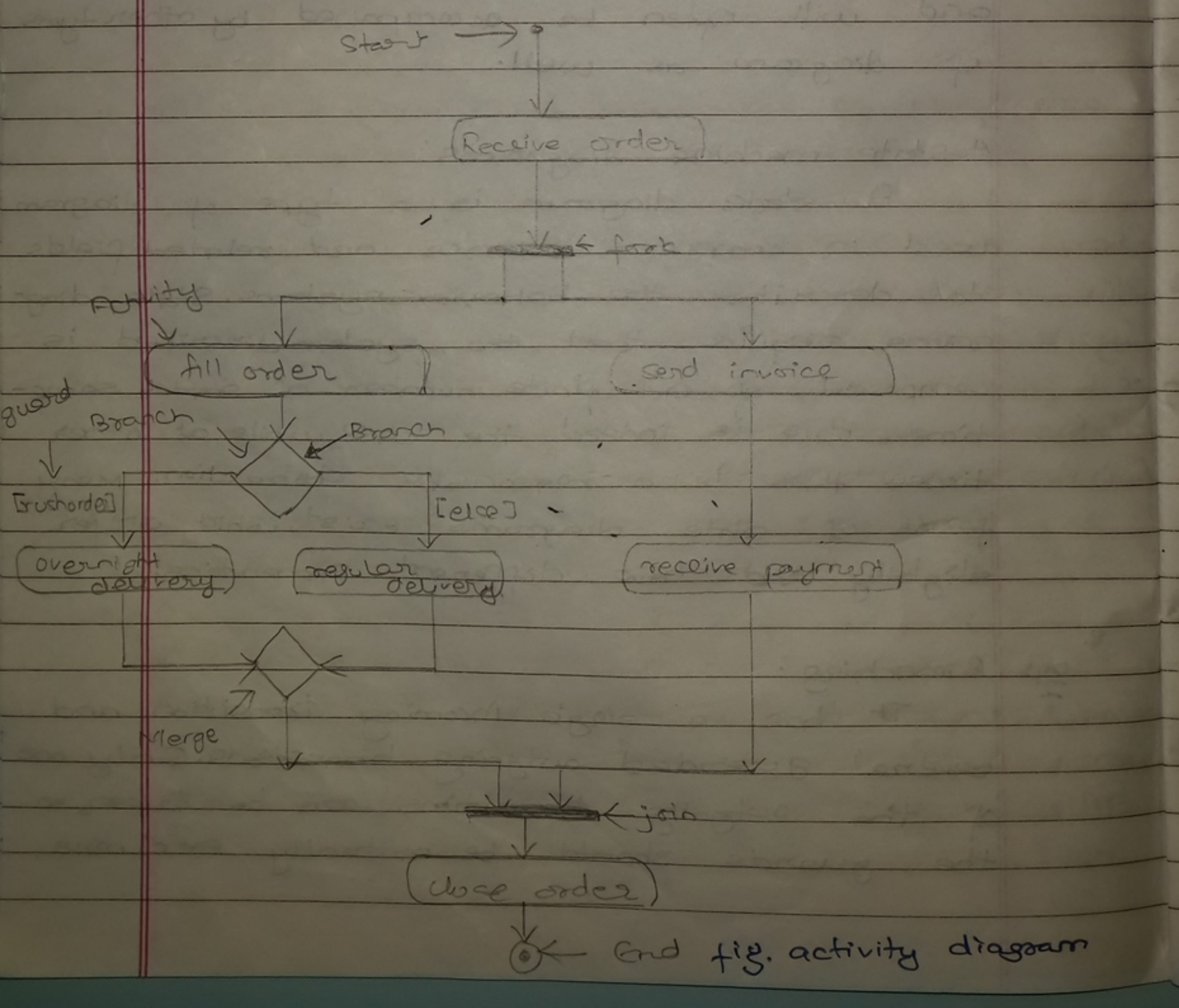
It has a single incoming transition and several guarded outgoing transitions. Only one of the outgoing transitions can be taken, so the guards should be mutually exclusive.

Using [else] as a guard indicates that "else" transition should be used if all the other guards on the branch are false.

Branching is belonged to conditional behavior of an activity diagram.

forking:

A fork has one incoming transition and several outgoing transitions when the incoming transition is triggered, all of the outgoing transitions are taken in parallel.



The above diagram shows the branching and forking in an activity diagram along with some joining and merging operations.

Branching is shown as: after an ordered is filled up; if it's such order, you do an over-night delivery and else, it's regular delivery.

forking operation is shown after receiving order and then filling up order and sending invoice parallel.

Joining:

Join operation is used to join together the threads that was started by the fork. So the join and fork operation must match with each other.

Joining operation is shown in above diagram before ending up the system i.e. it closes after receiving delivery and payment and then finally closes order.

9.

b) Methods of requirement gathering:

It's difficult to build a solution if you don't know the requirements. The "elicitation" step is where the requirements are first gathered from the client. Many techniques are available for gathering requirements. Each has value in certain circumstances, and in many cases, you need multiple techniques to gain a complete picture from a diverse set of clients and stakeholders.

Number of steps are:

1. One-on-one interviews
2. Group interviews
3. facilitated sessions
4. Joint application development (JAD)
5. Questionnaires
6. Prototyping
7. Use-cases
8. following people around
9. Request for proposals (RFPs)
10. Brainstorming

7. ans Exception handling is the process of responding to the occurence, during computation, of exceptions- anomalous or exception events requiring special processing - often changing the normal flow of program exception.

The implementation of exception handling in programming languages typically involves a fair amount of support from both a code generator and the runtime system accompanying a compiler. Two schemes are most common. The first, dynamic registeration, generates code that continually updates structures about that the program state in terms of exception handling. The second scheme and the one implemented in many production-quality C++ compilers, is a table-driven approach, which creates static tables at compile time and link time that relate ranges of the

program counter to the program state with respect to exception handling.

4. Ans