UNIT-3

object oriented Design

object oriented design is the process of planning a system of interacting objects for the purpose solving a software problem.

object Design

object design is the process of using an object oriented methodology to design a computing system or application. This technique enables the implementation of a software solution based on the concepts of objects. OOD serves as part of the object oriented programming (OOP) process or life cycle.

Following are the steps of object Design or Software design :-

1) Combining the three models to obtain operations on classes.
2) Design algorithms to implement operations.
3) optimize access paths to data.
4) Implement control for external interactions.
5) Adjust class structure to increase inheritance.
6) Design associations.
7) Determine object representation.
8) Package classes and associations into modules.

Combining Three Models

In this step, the operation to be performed on objects are defined by combining the three models developed in the object oriented analysis phase.

These three models are :-

1) object model
2) Dynamic model
3) Functional model

After analysis, we have object, dynamic and functional model, but the object model is the main framework around which the design is constructed. The object model from analysis may not show operations. The designer must convert the actions and activities of the dynamic model and the processes of the functional model into operations attached to classes in the object model. Each state diagram describes the life history of an object. A transition is a change of state of the object and maps into an operation on the object.

Purpose of Models

→ Testing a physical entity before building it.
→ Communication with customers.
→ Visualization.
→ Reduction of Complexity.

Designing Algorithms

Each operation specified in the functional model must be formulated as an algorithm. The analysis specification tells what the operation does from the view point of its clients, but the algorithm shows how it is done.

The algorithm designer must decide on the following :-

1) Choosing Algorithms :-

for example if Designer wants to select algorithm for search operation. Then these are the following optians :-

a) Computational Complexity
b) Ease of implementation and understand ability.
c) Flexibility.
d) Fine tuning the object model.

2) Choosing data Structures.

3) Defining internal classes and operations.

4) Assigning responsibility for operations.

Design Optimization

The basic design model uses the analysis model as the framework for implementation. The analysis model captures the logical information about the system, while the design model must add details to support efficient information access. The designer must strike an appropriate balance between efficiency and clarity. During design optimization, the designer must add redundant associations

for efficient access during analysis, it is undesirable to have redundancy in association network because redundant associations do not add any information. During design, however we evaluate the structure of the object model for an implementation. For that, we have to answer the following questions :-

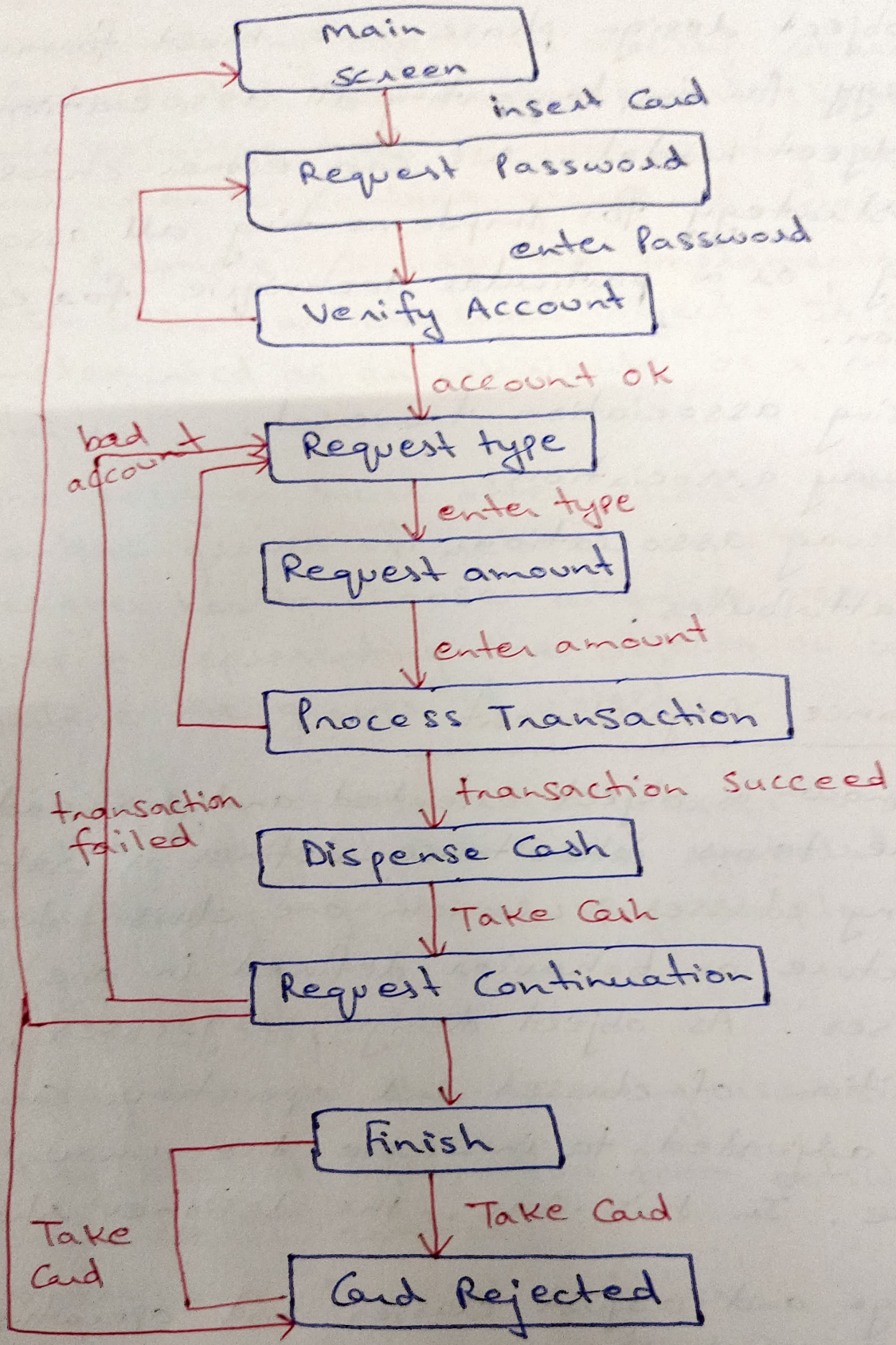
1) Is there a specific arrangement of the network that would optimize critical aspects of the completed system?

2) Should the network be restructured by adding new associations?

Implementation of Control

The designer must refine the strategy for implementing the state - event models present in the dynamic model. As part of system design, you will have chosen a basic strategy for realizing dynamic model, during object design flesh out this strategy. There are three basic approaches to implementing the dynamic model :-

→ State as location within a program.
→ State machine engine.
→ Control as concurrent tasks.


Main Screen
insert Card
Request Password
enter Password
Verify Account
account ok
bad account
Request type
enter type
Request amount
enter amount
Process Transaction
transaction failed
transaction succeed
Dispense Cash
Take Cash
Request Continuation
Finish
Take Card
Take Card
Card Rejected

Design of Associations

During object design phase, we must formulate a strategy for implementing all associations in the object model. We can either choose a global strategy for implementing all associations uniformly, or a particular technique for each association.

1) Analysing association traversal.
2) one way association.
3) Two way association.
4) Link attributes.

Inheritance Adjustment

As we know, in object oriented analysis and design the terms inheritance defines a relation ship among classes, wherein one class shares the structure or behavior defined in one or more classes. As object design progresses, the definitions of classes and operations can often be adjusted to increase the amount of inheritance. In this case, the designer should:-

→ Rearange and adjust classes and operations to increase inheritance.

→ Abstract common behavior out of groups of classes.

→ use delegation to share behavior when inheritance is semantically invited.

Object Representation

→ Implementing objects is mostly straight forward, but the designer must choose when to use primitive types in representing objects and when to combine groups of related objects. for example: Consider the implementation of a SSN within an employee object. It can be implemented as an attribute or a separate class.

→ The designer must often choose whether to combine groups of related objects. The object designer has to choose when to use primitive types in representing the objects or when to combine the groups of objects.

→ The object representation means "to represent object by using objects model symbols". Implementing objects are very simple.

Physical Packaging

Packaging involves the following issues :-

1) Hiding internal information from outside view.
2) Coherence of entities.
3) Constructing Physical modules.
   ↳ During analysis and system design phase we partitioned the object model into modules.
   * The initial organisation may not be suitable for final packaging of system implementation. New classes added to existing module or layer or separate module.

Information Hiding

During analysis phase, we are not concerned with information hiding. So, visibilities of class members are not specified during analysis phase. It is done during object design phase. In a class, data members and internal operations should be hidden, so they should be specified as private. External operations form the interface so they should be specified as public.

Following Design Principles can be used to design classes :-

→ A class should be given the responsibilities of performing the operations and proving information contained in it to other classes.

→ Calling a method of that class should access attributes of other class.

→ Avoid traversing associations that are not connected to this class.

→ Define interfaces at the highest level possible.

→ External objects should be hidden. Defining interface classes could do this.

→ Avoid applying a method to the result of another class unless the result class is already a supplier of methods to the caller class.

Coherence of Entities

Module class, method etc. are entities. An entity is said to coherent, if it is organized on a consistent plan and all of its parts fit together toward a common goal.

Documenting Design Considerations

Documenting design decisions are the best way of transmitting the design to others and recording it for reference during maintenance. The design document is an extension of the requirement analysis document.

1) The design document includes revised and much more detailed description of the object model - both graphical and textual.

2) Functional model will also be extended. It specifies all operation interfaces by giving their arguments, results, input - output mappings and side effects.

3) Dynamic model → If it is implemented using explicit state control or concurrent tasks then the analysis model or its extension is adequate.

* Keep the design document different from analysis document. The design document includes many optimizations and implementation artefacts.

* The design document will include a revised and much more detailed description of the object model.

Jackson System Development (JSD)

JSD is a method of system development that covers the software life cycle either directly or by providing a framework into which more specialized techniques can fit.

Phases of JSD

JSD has 3 phases :-

1) modelling Phase → The designer creates a collection of entity structure diagrams and identifies the entities in the system, the actions they perform, the attributes of the actions and the time order of the actions in the life of the entities.

2) Specification Phase → This phase focuses on actually what is to be done? This step maps progress in the real world on progress in the system that models it.

3) Implementation Phase → This phase determines how to obtain the required functionality. This step involves transformation of the specification into an efficient set of processes.

JSD steps

1) Entity / action step
2) Initial model step
3) Interactive function step
4) Information function step
5) System Timing step
6) System implementation step

Merits of JSD

- → It is designed to solve the real time problems.
- → JSD modelling focuses on time.
- → It considers simultaneous processing and timing.
- → Provides functionality in the real world.
- → It is a better approach for microcode applications.

Demerits of JSD

- → It is a poor methodology for high level analysis and database design.
- → JSD is a complex methodology due to pseudo code representation.
- → It is less graphically oriented as compared to SA/SD or OMT.
- → It is a bit complex and difficult to understand.

How do you map the object oriented concepts using non object oriented languages? Explain with an example. AKTU (2010-11), (2011-12)

Implementing an object oriented concept in a non object oriented language requires the following steps :-

1) Translate classes into data structure.

→ Each class is implemented as a single contiguous block of attributes. Each attribute contains a variable. Now an object has state and identity and is subject to side effects.

→ A variable that identifies an object must therefore be implemented as a shalable reference.

2) Pass arguments to methods

→ Every method has at least one argument. In a non object oriented language, the argument must be made explicit.

→ Methods can contain additional objects as arguments. In passing an object as an argument to a method, a reference to the object must be passed if the value of the object can be updated within the method.

3) Allocate Storage for objects

→ objects can be allocated statically, dynamically or on a stack.

→ Most temporary and intermediate objects are implemented as stack based variables.

→ Dynamically allocated objects are used when there number is not known at compile time.

→ A general object can be implemented as a data structure allocated on request at run time from a heap.

4) Implement inheritance in data structures

Following ways are used to implement data structure for inheritance in non object oriented programming language.

→ Avoid it
→ Flatten the class hierarchy
→ Break out separate objects

5) Implement method resolution

Method resolution can be implemented in a following ways :-

→ Avoid it
→ Resolve methods at Compile time
→ Resolve methods at run time

6) Implement Associations

Implementing associations in a non oriented language can be done by :-

→ Mapping them into pointers
→ Implementing them directly as association Container objects.

7) Deal with Concurrency

→ Most languages do not explicitly support Concurrency.
→ Concurrency is usually needed only when more than one external event occurs, and the behavior of the program depends on their timing.

8) Encapsulate internal details of classes

→ object oriented languages provide constructs to encapsulate implementation.
→ Some of this encapsulation is lost when object oriented Concepts translated into a non object oriented language, but we can still take advantage of the encapsulation facilities provided by language.

Object oriented Programming Style

Reusability

Inheritance supports the concept of "reusability" i.e when we want to create a new class and there is already a class that includes some of the code that we want, we can ~~drive~~ derive our new class from the existing class.

→ Reusability is the ease with which something can be used repeatedly.

Why Reusability is important in OOPs

In object oriented systems, assessing reusability plays a key role in reducing a cost and improving the quality of the software. object oriented programming helps in achieving the concept of reusability through different types of inheritance programs, which further helps in developing reusable software modules.

Extensibility

Extensibility means permitting language users to define new language features. Starting with a base language and using various definition facilities an extensible language user can create new notations, new data structures, new operations and sometimes new regimes of control.

→ Extensibility is a measure of the ability to extend a system and the level of effort required to implement the extension.

→ Extensibility is a system design principle where the implementation takes future growth into consideration.

Robustness

It is the measure of stability of software applications in extreme situations (eg. maximum load conditions, erroneous user inputs). Robust applications have less downtime and can reduce maintenance costs.

→ Robust programming is a style of programming that focuses on handling unexpected termination and unexpected actions.

→ System robustness is defined as a system's ability to remain functioning under disturbances.

Programming in the Large

→ Programming in the large and programming in the small refer to two different aspects of writing software, namely, designing a large system as a composition of smaller parts, and creating those smaller parts by writing lines of code in a programming language respectively.

→ we distinguish the activity of writing large programs from that of writing small ones.

object oriented language Features

- → Encapsulation
- → Abstraction
- → Inheritance
- → Polymorphism