

1199

**DESIGN OF AN OBJECT-ORIENTED MODEL FOR
INFORMATION SYSTEM**

Dissertation submitted to the
Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & TECHNOLOGY

by

SUNIL KUMAR VERMA



**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI -110 067
(INDIA)**


JANUARY 1995.

Dedicated to
my
Mother

CERTIFICATE

This is to certify that the dissertation entitled "DESIGN OF AN OBJECT-ORIENTED MODEL FOR INFORMATION SYSTEM" , being submitted by SUNIL KUMAR VERMA to Jawaharlal Nehru University, New Delhi in partial fulfilment of the requirements for the award of the degree of MASTER OF TECHNOLOGY in Computer Science & Technology is a record of the original work done by him under the supervision of Dr. C.P. Katti, Asso. Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi during the year 1994, monsoon semester.

The results reported in this dissertation have not been submitted in part or full to any other university or Institution for the award of any degree or diploma.



12-1-95

Prof. K.K. Bharadwaj
Dean, SC & SS
Jawaharlal Nehru University
New Delhi - (INDIA)
Pin 110067.



Dr. C.P. Katti
Associate Professor,
SC & SS
J.N.U., New Delhi.

ACKNOWLEDGEMENT

I express my humble gratitude towards Dr. C.P Katti, Associate Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi who was always there to guide me to the safe exit in case of my crisis and was all the way with me to boost my morale and to provide his valuable guidance.

I am also highly thankful to Prof. K.K. Bharadwaj, Dean, School of Computer and Systems Sciences, who took great pains to provide me with the environment and all the facilities required for the successful completion of my dissertation.

I also take the pleasure of thanking Prof. P. C. Saxena and other faculty and staff members as well as my friends and well wishers who have been directly or indirectly helpful by providing me the material and moral support during the continuation of this dissertation.



SUNIL KUMAR VERMA

CONTENTS

1.	INTRODUCTION	1 - 5
2.	CONCEPTS IN OBJECT-ORIENTED DATABASES (OODB)	6 - 19
2.1	Object	7
2.2	Class	7
2.3	Object-Oriented Properties	8
2.4	Definition of OODB Systems	16
2.5	Merits of Object-Oriented Approach	17
3.	AN OBJECT-ORIENTED DEVELOPMENT PLATFORM FOR INFORMATION SYSTEM	20 - 36
3.1	Object-Oriented System Life Cycle	20
3.2	Object-Oriented Analysis (OOA)	21
3.3	Object-Oriented Design (OOD)	24
3.4	Object-Diagram Used in OOD Methodology	26
3.4.1	Object & Class Layer	26
3.4.2	Structure Layer	29
3.4.3	Subject Layer	30
3.4.4	Attribute Layer	31
3.4.5	Services Layer	34
4.	CASE STUDY	37 - 46
4.1	Problem Definition	37
4.2	Object-Diagram for Designing Problem Schema	40

5.	CODING	47-54
5.1	Functions of Class Modules	47
5.2	Programming Environment Used	54
6.	CONCLUSION	55-57
	BIBLIOGRAPHY	58
	PROGRAM	(i)-(xv)

CHAPTER - 1

INTRODUCTION

For the last decade of this century, computers of vastly greater capabilities have been evolved. The value of structured design has not changed. Also the as structured programming appears to fall apart when coding exceeds 100,000 lines. More recently, many design methods have been proposed, many of them invented to deal with the perceived shortcomings of structured design. These all methods are largely variations upon a similar theme. So most methods can be classified as one of following.

- (i) Top down structured design
- (ii) Data-driven design
- (iii) Object-oriented design

Many software have been developed using Top-Down structured design method. But this structured design method does not say anything about the data abstraction and information hiding. It also does not provide an adequate manner of dealing with concurrency. Structured design method is not well suited for designing extremely complex systems.

In the Data-driven method, the structure of a software system is derived by mapping system inputs to outputs. As with structured design, Data-driven design method has been applied to many complex problems but this method requires little concern for time-critical events.

The method which I am introducing in my project is object-oriented design method in which one can model software systems as collection of cooperating objects. by treating individual objects as instances of a class within a hierarchy of classes. Also this method directly reflects the topology of more recent high order programming languages such as small talk, Ada and C++.

Information system is not easy to model. The difficulty involved in modeling is it's complex nature and it's difficulty to be easily understood fully. The way to handle this problem is to develop unified framework for the modeling process such that models can be linked through that framework in a consistent, coherent way. The object oriented paradigm offers such a modeling framework. Object oriented design is the method that leads us to an object-oriented decomposition.

By applying object-oriented design method, we create software that is resilient to change and written with economy of expressions. I can achieve a greater level of confidence in the correctness of our software through an intelligent separation of its state space. Hence finally, we reduce the risks of developing complex Information systems.

The objective of my dissertation is to design and develop an object-oriented model for Banking Information System (BIS). There are several steps involved in the process of design and development.

The first step is to study about the basic definitions of database management systems. This include studying about the object-oriented methodology and it's properties. The definitions of all related terms and characteristics are discussed and elaborated in **chapter-2**.

The second step is the analysis of the case which is explained in **chapter -3**. In the analysis of object-oriented approach method, all the entities of information system is taken separately for the purpose of building model. This

includes discussing about the layers, namely;

- a) Object & Classlayer
- b) Structure layer
- c) Subject layer
- d) Attribute layer
- e) Services layer

In object & class layer the different objects and class which exist in the problem domain of the BIS will be defined. In structure layer, relationships among the objects will be described. These relationships will be nothing but basic properties of object-oriented approach. In subject layer, the overview of the problem domain is represented. This layer partition's the problem domain into problem subdomains and also establishes workpackage. Attributes associated with any object is shown by attribute layer. In services layer, function processing upon data will be described. Services like create, connect, access and release are discussed here. The idea of introducing message connection in services layer is just to support this layer. All these layers when combined, constructs a diagram called object-diagram.

This object-diagram is main building block for the designing of the problem domain. Object-diagram along with the definition of my case (i.e. Banking Information System) is neatly explained and sketched in **chapter-4**.

Next step is the algorithm for the problem domain. This algorithm is explained in **chapter-5** in different modules. Also brief introduction about the object-oriented programming language (C++) used in my coding will be given in chapter-5. I have ended my dissertation by summarizing about all chapters discussed and have proposed further research and development work in this area.

CHAPTER - 2

CONCEPTS IN OBJECT ORIENTED DATABASES (OODB)

The term object emerged almost independently in various fields in computer science to refer to notions that were different in their appearance, yet mutually related. All of these notions were invented to manage the complexity of software systems in such a way that objects represented components of a modular units of knowledge representation. The term object oriented means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This is in contrast to conventional programming in which data structure and behavior are only loosely connected. Object-oriented design methods have evolved to help developers exploit the expressive power of object-based. The term object model is a unifying concept in computer science, applicable not only to programming languages, but to design of user interfaces, databases, knowledge bases and even computer architecture. Now I will define object, class and characteristics which are required by an object oriented approach.

2.1 OBJECT: It is defined as "entities that combine the properties of procedure and data since they perform computations. Objects have certain integrity that can not be violated. An object can only change state, behavior or it can stand in relation to other objects. So finally we can say that an object has state, behavior and identity. The identity is that inherent property of an object which distinguishes it from all other objects. The term behavior means how an object acts and reacts, in terms of it's state changes and message passing. The state of an object is all the static properties of the object plus the current values of each of these properties.

2.2 CLASS: A class is a set of objects that share a common behavior and common structure. An object is an instance of a class. Classes with no instances are called abstract classes. The objects in a class share a common semantic purpose, above and beyond the requirement of common attributes and behavior. For example although a hut and horse both have a cost and age, they may belong to different classes. If hut and horse are regarded as purely financial assets, they may belong to the same class. If I think that I can paint a hut and feed a

horse, they would be modeled as different classes. The interpretation of semantics depends on the purpose of each application and is a matter of judgment. Each object 'knows' its class. Most object-oriented programming languages can determine an object's class at run time. An object's class is an implicit property of the object. Now the question arises that if objects are the focus of object modeling, why bother with class? The answer is that by grouping objects into classes, we abstract a problem.

2.3 OBJECT ORIENTED PROPERTIES :

There are several themes underlying object-oriented technology. If some of them are not unique to object oriented systems they are particularly well supported to object-oriented systems.

(i) INHERITANCE : Inheritance, is a relation among classes that allows the definition and implementation of one class to be based on that of other existing classes. Inheritance is an important conceptual tool that allows one to construct software systems from scratch. It is, therefore, natural to use objects to organize, abstract knowledge so that various

concrete situations can be generated once the relevant parameter values are supplied. Since 'string' is defined as a subclass of 'array'-this means that strings will inherit all the instance variables of variables of arrays. So a relationship among classes wherein one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) other classes is defined as inheritance relationship.

(ii) **GENERALIZATION** : It is a relationship between a class and one or more refined versions of it. The class being refined is called superclass and each refined version is called subclass. Generalization is sometimes called the 'is-a' relationship because each instance of subclass is an instance of superclass as well.

Generalization and inheritance are transitive across a arbitrary number of levels .each subclass not only inherits all the features of its ancestors but adds its own specific attributes and operations. We use generalization to refer to the relationship among classes,while inheritance refers to the mechanism of sharing

attributes and operations using the generalization relationship.

(iii) AGGREGATION : Here lower level concepts aggregate as part of higher level concepts. Aggregation is 'a-part-of' relation in which objects representing the components of something are associated with an object representing the entire assembly. The most important feature of aggregation is transitivity i. e. if A is part of B and B is part of C, then A is part of C. Aggregation is also antisymmetric i.e. if A is part of B then B is not part of A.

(iv) ASSOCIATION : In association lower level concepts are associated in a higher level concepts on the basis of some meaningful connection. It is group of links with common structure and common semantics. An association describes a set of potential links in the same way that a class describes a set of potential objects.

(v) Polymorphism: It is a concept, according to which a name may denote objects of many different classes that are related by some common superclass. Thus the ability of different

objects to respond differently to the same message is known as polymorphism.

(vi) ABSTRACTION: The word abstraction arises from a recognition of similarities between certain objects, situation or processes in the real world, and the decision to concentrate upon these similarities and to ignore for the time being the differences. An abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.

An abstraction focuses on the outside view of an object, and so serves to separate an object's essential behavior from it's implementation. All abstractions have static as well as dynamic properties. For example, a file object takes up a certain amount of space on a particular memory device; it has a name, and it has contents. These are all static properties. The value of each of these properties is dynamic, relative to the lifetime of the object i.e. a file object may grow in size, it's name may change.

Abstraction must always be for some purpose, because the purpose determines what is important and what is not important. Many different abstraction of some thing are possible, depending on the purpose for which they are made. All abstractions are incomplete and inaccurate. All human words and language are abstractions-incomplete description of the real world. This does not destroy their usefulness. The purpose of an abstraction is to limit the universe so we can do things. So in building object oriented model we do not search for absolute truth but for adequacy for some purpose. There is no single correct model of a situation, only adequate and inadequate ones.

(vii) ENCAPSULATION:: Abstraction and encapsulation are complementary concepts. Abstraction focuses upon the outside view of an object, and encapsulation which is also known as information hiding, prevents clients from seeing it's inside view, where the behavior of the abstraction is implemented.

For abstraction to work, implementations must be encapsulated. It means that each class must have two parts i.e. an interface and an implementation. The interface of a

class captures only it's outside view, encompassing our abstraction of the behavior common to all instances of the class. The implementation of class comprises the representation of the abstraction as well as the mechanisms that achieve the desired behavior.

Intelligent encapsulation localizes design decisions that are likely to change. To summarize, we define encapsulation as it is the process of hiding all of the details of an object that do not contribute to it's essential characteristics. Here hiding is a relative concept; what is hidden at one level of abstraction may represent the outside view at another level of abstraction.

(viii) MODULARITY : In object oriented languages class and objects form the logical structure of a system. We place these abstractions in modules to produce the systems physical architecture. For large applications where classes are many, the use of modules is essential to help complexity. The inner workings of an module are expressed in terms of the module's interactions with other modules. System details that are

likely to change independently should be the secrets of separate modules. Every data structure is private to one module, it may be directly accessed by one or more programs within the module but not by programs outside the module. Any other program that requires information stored in a module's data structures must obtain it by calling module programs. Thus modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.

(ix) TYPING :- It has come from the theory of abstract data types. It is defined as enforcement of the class of an object, such that objects of different types may not be interchanged or if they are interchanged, it must be in restricted manner. An object oriented programming language may be untyped, strongly typed or weak typed. A strongly typed language is one in which all expressions are guaranteed to be type-consistent. If we don't do type checking, program may crash at runtime in unusual way. Also by declaring type, it helps in program documentation and compiler can generate most efficient object code. There is static typing which

means that the types of all variables and expressions are fixed at the time of compilation. Opposite to static typing there is dynamic typing which means that the types of all variables and expressions are not known until runtime.

(x) PERSISTENCE : By defining persistence in object model, it gives idea to the object oriented databases. In object oriented databases, apart from the persistence of the state of the object, class must also transcend any individual program so that any program interprets this saved state in same way. For any programming language to persist it must include transient results in expression evaluation, local variables in procedure activations, own global variables. Apart from these, object oriented programming languages also includes data that exists between execution of a program, between various versions of a program for showing persistence. To summarize all above we can define persistence as the property of an object that continues to exist even after it's creator stop existing.

(xi) CONCURRENCY: This emphasizes upon process abstraction

and synchronization. Each object may represent a process abstraction. Such objects may be called active. So we can say that concurrency is the property that distinguishes an active object from one that is not active. Once we apply concurrency into a system, we must see how active objects, synchronize their activities with one another and objects which are sequential. For example, if two objects try to send messages to a third object, we must be certain to use some means of mutual exclusion, so that the state of the object being acted upon is not corrupted when both active objects update their state simultaneously. My example includes the concepts of abstraction, encapsulation and concurrency also.

2.4 Definition of Object-Oriented Database Management System (OODBMS):

In OODBMS, the basic unit is object-class. For this encapsulated combinations of data and procedures must be taken together as an integrated unit. An object oriented database system should satisfy two conditions. The first is that it must be a database management system and the second

is that it must be an object-oriented system. The first criteria i.e. database management system is satisfied, with the features of concurrency, persistence, secondary storage, data reliability transaction management and schema modification. Now the second criteria i.e. object oriented system (the work of my project) is satisfied by the properties discussed so far i.e. encapsulation, data abstraction, type class, object inheritance, polymorphism, generalization, aggregation.

2.5 MERITS OF OBJECT-ORIENTED APPROACH

1. The main idea behind using this approach is that it allows systems life cycle to use integrated and coherent structure and behavior. Other modeling concepts do not permit this in a natural way.

2. The object-oriented approach eliminates the transformations that currently occur from analysis and specification to design and construction. Say for example from entity relationship modeling to design and construction of a relational database where tables are complex

transformations of entities. In the object-oriented development, the same objects are considered throughout the complete system development cycle and no mapping is done from the type of object to another, also, such objects will have clear relationship to real life entity.

3. As described above that objects have relationship with real world entity. So in general they are not static, hence flexibility has to be provided. Object oriented approach allows the same object to be viewed at different levels of abstraction.

4. Normally the interface between programming and database languages is crude since each language provides a different type system. Moreover, interfaces are often designed as an afterthought. This may create complexity. But in object-oriented method we can overcome such type of complexities.

Another merits of object-oriented approach include the following :

a) It is a full life-cycle methodology from external design to code generation.

- b) It exploits the expressive power of all object-based and object-oriented programming languages.
- c) It reduces development risk.
- d) This approach encourages the reuse of software components.

CHAPTER - 3

AN OBJECT ORIENTED DEVELOPMENT PLATFORM FOR
INFORMATION SYSTEM

(3.1) OBJECT-ORIENTED SYSTEM LIFE CYCLE:

The system life cycle must be represented in a realistic way to provide intellectual and management control of the system's development.

The system life cycle models has led to the description of spiral models for system development. The spiral is flexible arrangement of sequential and concurrent activities with well defined point of progress review and approval. The spiral model defines a set of limited, time phased activity loops to manage a system development. Each activity loop is initiated and managed dynamically on the basis of the outcome of previous loops. The activity loop in the spiral can be sequential or concurrent and can be one of three types; investigation, specification or implementation.

Each loop, regardless of type has following steps:

Planning the activity objectives, statement of work

and schedules, performance of the work needed to achieve the activity objective, evaluation to determine if the objectives are met and to plan for next activities. Activity loops are performed in the spiral until the system development is completed.

This spiral model provides the flexibility to plan and record system development activities in a controlled, user defined patterns. The spiral system life cycle is a natural one for object-oriented system development. The spiral model supports flexible patterns of object oriented analysis (OOA), object oriented design (OOD) and object oriented implementation and testing (OOIT) activity loops during a system development.

(3.2) OBJECT ORIENTED ANALYSIS: (OOA)

The goal of OOA is to achieve an adequate understanding of the problem domain in order to describe the requirements of the desired system. This is the most critical phase of system development. Most system development projects fail because of inadequate understanding of problem requirements.



TH-5596

Many times, even when a system is completely developed, it does not solve the target problem. Thus, OOA explains about close cooperation between the system developer and the system customer.

The approach which is used here to develop OOA method consists of five activities as listed here.

(a) **Finding class and object:** Information gathering techniques is used here to get an in-depth understanding of the problem domain. Thus understanding is used here to specify the required objects and classes of the desired system. Generic classes with no objects are just termed a class.

(b) **Identifying structures:** Hierarchical structures are defined here based on the relationships among classes. A generalization-specialization structure and the whole part structure shows the generalization and the aggregation relationships. Inheritance is provided here through the generalization specification structure.

(c) **Identifying subject:-** There should exist some sort of

real world description in order to derive objects. Objects can be identified by looking the roles, locations, organizational units and the like. The approach, I have proposed here is based on generalized process modeling which assumes that information system exists for the regulation of resources of one or more types. Thus, information system maintains the information about the state of such resources. So I have started object identification with the assumption that resource identities, which we call primary entities, have been identified. Also I assume that, I know how to regulate the entities and the state information required to manage their behavior. These entities are conceptual models of real-world objects.

(d) Defining objects and attributes:

To define objects, we start with the idea that each resource should correspond to an object. This guarantees that each object which will be finally implemented in the computer bears a direct relationship to a real-life entity i.e. a resource. So I have found out the relevant attributes of such objects that characterize their state.

(e) **Defining Services:** Here interaction between objects by means of services is defined along with the message connections that support the services. Object oriented approach requires the approach that classes are encapsulated here and we can access data only through class methods. Thus, there are methods to process inputs to classes coming from entities external to the computerized portion of the system that will change some attributes in the class, some will produce outputs defined by the requirements and others will provide data needed by another class to execute a given method. This concept introduces the idea of object (class) collaboration whereby a class requires a specific service from another class. Object collaboration implies the passing of data and/or control i.e., messages, from one class to another. Object collaboration is done here by object diagram.

(3.3) **OBJECT ORIENTED DESIGN (O.O.D.) :**

The goal of object-oriented design is to transform the system models of OOA into system specifications in the solution domain. Designing is a creative attempt in which trade off must be made to take advantage of opportunities and

to deal with constraints of the implementation environment. The following four steps are to be performed iteratively and concurrently in a system development spiral.

(a) Identify the classes and objects at a given level of abstraction:- This step emphasizes the discovery of information from the application domain in order to identify potential classes and objects. Candidate classes and objects are proposed and analyzed.

(b) Identify the semantics of these classes and objects: Considerable effort is spent here to understand the semantics of each class and object in the system. The design templates for classes and objects are started in this step.

(c) Identify the relationships among these classes and objects:- The relationships (e.g. generalization, aggregation and association) among classes and objects are discovered, analyzed and represented in the design templates. This is a highly creative exercise that determines the structure of the system.

(d) Implement these classes and objects:- The final design

decisions are made to complete the specification of these classes and objects. Additional design templates are defined here. In the module template, classes and objects are allocated to physical modules for implementation. Required processes are defined in the process template and is assigned to processors in the processor template.

(3.4) Object Diagram Used in O. O. Design for Information System

Object diagram is the outcome of an object oriented analysis. In problem domain of the information system, object diagram contains five different layers of the objects, similar to the five activities described in the object oriented analysis :

1. Object & Class Layer
2. Structure Layer
3. Subject Layer
4. Attribute Layer
5. Services Layer

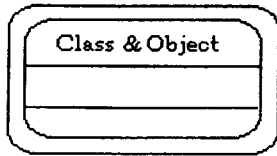
Now I will describe each of these layers separately.

(3.4.1) Object & Class Layer :

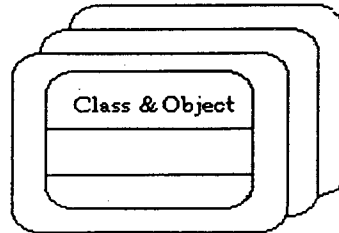
The object & class layer gives an abstraction in a

problem domain, reflecting the capabilities of a system to keep information about it, and interact with it or perform the both. In a class, we do the description of one or more objects with a uniform set of attributes and services. It also includes description of how to create new objects in the class.

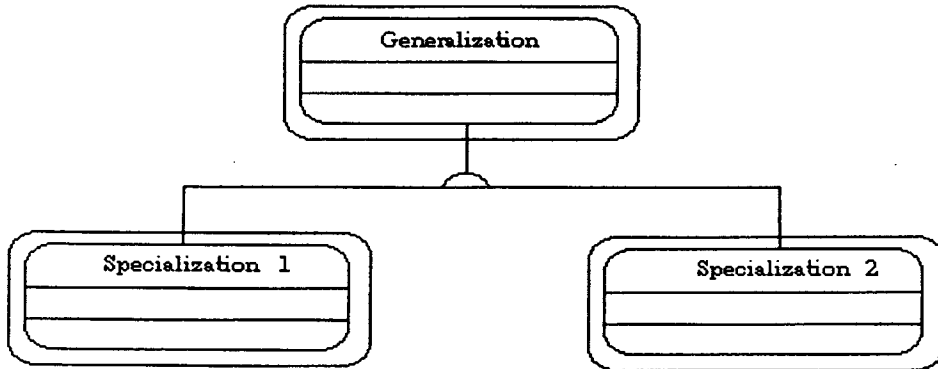
The class & object layer is represented by the bold rounded rectangle, divided into horizontal section (see fig. (1a)). Certain object oriented analysis connections map an object to another. Other OOA connections may map an class to another class or there may be the case that some OOA connections may map an object to a class. In object & class layer, it's name should describe a single object within the class. We give name of an object & class, a noun or adjective and noun which are taken from the standard vocabulary. Fig. (1b), Shows a class and object symbol as a class with one or more objects in the class. Now we consider upon the things which are taken into consideration while taking objects. Firstly we see that does an object need to provide some behavior. Next we think about the classes with only one object. Also we see that whether a class with just a single object really does



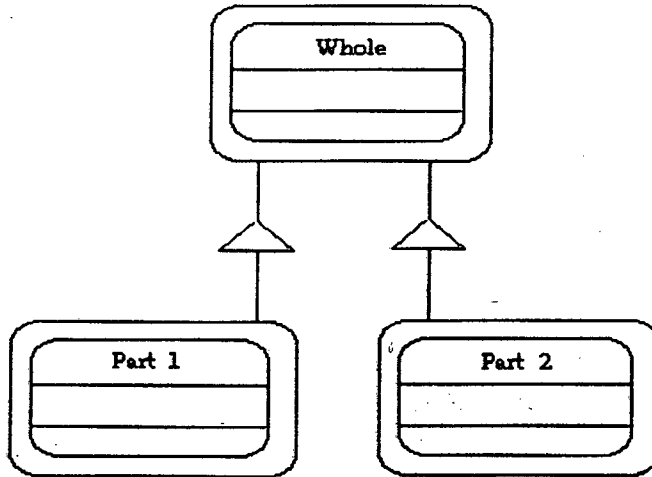
Class & Object symbol
Fig. 1 (a)



Picturing some number of objects within a class Fig. 1 (b)



Generalization-Specialization notation Fig. 2 (a)
(is - a relation)



Whole - part structure notation (a - part - of relation)
Fig. 2 (b)

reflect the problem domain. Thus discussion about object & class layer concludes with the investigation of problem domain for an initial set of class and objects.

(3.4.2) Structure Layer :

This layer is defined in two ways. First one is that it reflects problem domain and second is that it shows the system's responsibility. Here we are using this layer to describe about Generalization-specialization structure and whole part structure both. Generalization- specialization is nothing but distinguishing between the classes. The property of inheritance comes in the generalization-specialization structure. On the other hand the whole part structure represents the aggregation property. Each class and object of object & class layer examines for generalization-specialization structure and whole part structure in the structure layer.

The notation for generalization-specialization structure layer is shown in fig. (2a). Here generalization class is shown at the top and the specialization classes below, with lines drawn between them. In the figure, a semicircle is shown to distinguish structure with class. We place

generalization at top and sepcialization at bottom only for the easyness to understand. The name of the specialization is the name of generalization followed by its qualifying name. The criteria for making generalization-specialization structure layer is that, whether it is reflecting the problem domain or not. We do not use generalization-specialization structure just for the sake of extracting out a common attribute.

Whole part or aggregation is one of the basic methods of organizing things. The notation for whole part structure is shown in fig. (2b). In this layer whole object (of an object and class layer) is at the top and then a part of object (of an object & class layer) below, with a line drawn between them. A triangle shown in the figure distinguishes objects as forming a whole part structure. when we take various combinations of generalization-specialization structure layer, or whole part structure layer or both we get the multiple structure layer.

(3.4.3) SUBJECT LAYER|

This layer gives an overview of a larger OOA model. The term subject is a mechanism for supervising us through a

large and complex model. The main basis for the identification of subject layer is problem domain complexity. This layer presents the overall model from an even higher perspective. Also the subject layer helps me in reviewing the object model and system's responsibility. While selecting the subject, what we do is to promote the uppermost class in each structure upwards to a subject. Then promote the each class & object which is not in a structure, upward to a subject. In subject layer as shown in Fig. (3), we draw each subject as a simple rectangular box, with a subject name and number inside. A class & object may be in more than one subject. Also one subject may contain other subjects. For small information system model it is not at all necessary to introduce this layer. But for larger information system (The case which I have taken) this layer is necessary to partition the problem domain into problem subdomains, and also establishes workpackage. So for larger information system it becomes useful to introduce subject layer.

(3.4.4) Attribute Layer :

We can define the term attribute as some data (or

1.subject 1

2.subject 2

Fig. 3(a) subject notation, collapsed

1.subject 1
class & object 1 class & object 2

2.subject 2
class & object 3 class & object 4

Fig. 3 (b) subject notation, partially expanded

1	1
1	1

2	2
2	2

Fig. 3(c) subject notation, expanded

Class & Object
Attribute 1 Attribute 2

Class
Attribute 1 Attribute 2

Attribute Notation Fig.(4)

Class & Object
Service 1 Service 2

Class
Service 1 Service 2

Service Notation Fig.(5)

information) for which each object in a class has it's own value. As shown in the fig. (4) this layer is placed in the centre section of the class & object and class symbol.

If some object has only one attribute then we are confused that whether it is attribute or address of the object. Because every object and class has one address. So it will be better and simplified if one object has multiple attributes. Also object must have value for each attribute. While analyzing we avoid merely derived results. For example, say that if 'client's age' in a system is given then it is not necessary to derive client's date of birth. Derived result when found directly complicate the picture. So we capture attributes for which we can reach to useful derived results.

For placing an attribute in the object diagram we apply the rule that for classes within a Generalization-specialization structure, I put an attribute at the uppermost point in the structure in which it remains applicable to each of it's specializations. If an attribute is applicable in entire level of specializations, then I move

it up to the corresponding generalization. Again when I find the situation where an attribute sometimes has a meaning but sometimes it's value is not meaningful then I recheck about the generalization-specialization structure and correct it in accordance with that each attribute must have meaningful value. Also while placing attributes in the object diagram I check each attribute whether it has single value or repeating values. If any attribute has repeating value, I revisit to the object and class layer and get it corrected by making an additional object & class. These were the various ways of putting an attribute layer in the object diagram :

(3.4.5) Services Layer :

The term service is defined as a specific behavior that an object is responsible for exhibiting. So this layer provides a set of operations that can be requested by other objects. The other issue while defining service layer is to define the necessary communication between objects. Such commands and requests will be the nature of human interaction with a system. So this interaction phenomenon is used between parts of the OOA model.

In information system, every data processing system must have "data" and "processing". In attribute layer we discussed about the data in the system. Here in this layer function processing upon data is described. Fig. (5) shows the service layer in which services are placed in the bottom section of the object symbol. There are generally four services called create, connect, access and release. Create service creates and initializes a new object in a class. The connect services, connects or disconnects an object with another. The third service i.e. access services gets or sets the attribute values of an object. The last service i.e. release service, deletes an object. There may be also some other complex services like calculate and monitor.

In designing by object oriented approach, the service layer includes 'message connections' also. Message connection is defined to reflect both the problem domain and the system's responsibilities. In this connection is made with two objects (or class), in which one is 'sender' who sends a message to a 'receiver', who gets some processing done. We give three rules to identify needed message connections. The first one is that what other objects does it

need services from ? We put an arrow to each of those objects. The second rule is that what other objects need one of it's services. We put an arrow from each of those objects to the one under consideration. The last rule is that follow each message connection to the next object and repeat the previous two rules. The idea behind introducing message connection is just to support the service layer.

In designing object oriented model for information system we make object diagram with the help of these five layers and then go for coding in object oriented programming language.

CHAPTER - 4

CASE STUDY

4.1 PROBLEM DEFINITION :- Some bank X wants to develop a prototype system for processing and maintaining its transactions. This constructed prototype considers all of its customers to be of the same type. There are only four types of transactions possible. These transactions are:-

- (i) deposit
- (ii) withdrawal
- (iii) opening of account
- (iv) Closing of account.

These above four transactions have to be logged onto files, and their effect should reflect on the customer's balance and on the Cash-on-hand of the bank. The most important criteria of the prototype is that it should not lose its integrity if the power fails while it is running.

Now I want to develop a banking information system (BIS) which satisfies above requirements. The guidelines will be as follows:-

(a) On running BIS it should open the customer's file "CUSTOM.DAT". If the file does not exist, then it must create one by initializing it.

(b) After opening of the customer's file, it should load the names of all the customers into the memory, along with their account numbers. Now the name of each customer acts as the key field to access the account numbers. For easiness, I have assumed that same name is not given to two customers.

(c) Once CUSTOM.DAT is loaded, BIS opens today's transaction file. The name of the transaction file also contains today's date.

(d) After starting all of the above processes BIS will display a menu asking for the type of transaction the user wants. These required actions on each transaction are given here:-

(i) Deposit :- Firstly ask for customer name. If customer does not exist, issue an error message and return. Else, ask for the amount to be deposited. With the successful entry of both (i.e. name and amount), log the data onto the transaction

file, and update CUSTOM.DAT and the cash-on-hand.

(ii) Withdrawal:- We do same process in this as was done in deposit. One more process is done here that customer cannot withdraw more than what he has in his balance.

(iii) Opening of Account:- In this I open account of those who have no any account already. Also opening balance is given at the time of opening the account.

(iv) Closing of Account:- In this, I delete customer's account number and name. Here his rest balance is returned to him. This should be reflected on the cash--on-hand.

So the information contained in CUSTOM.DAT is as given here:-

Cash On Hand
Number of customers
customer Records
 Customer name
 Account number
 Balance

Information contained in the transaction file will be:-

Number of Transactions

Transaction records

Transaction number

Transaction type

Customer Name

Amount of Transaction (deposit, withdrawal etc.)

These were the all details regarding BIS and it's workings. Now I have to develop above BIS definition in C++.

4.2 OBJECT DIAGRAM FOR DESIGNING PROBLEM SCHEMA :

All mentioned guidelines in above problem definition is now converted into the object diagram. Each layers (i.e. Object and Class layer, Structure layer, Subject layer, Attribute layer and Services layer) are drawn separately, in figures (7 - 11) shown here. These are combined all together to construct object diagram, which is also drawn in figure (12) shown here.

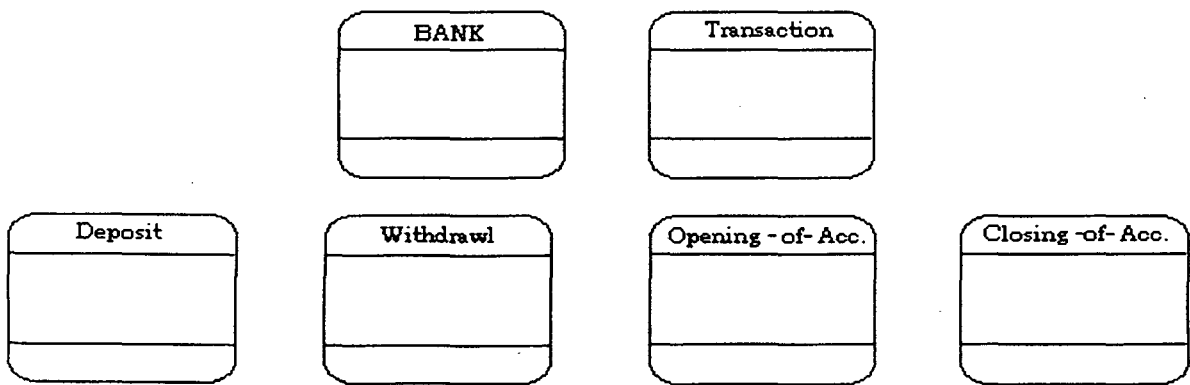


Fig (7) Object layer for BIS

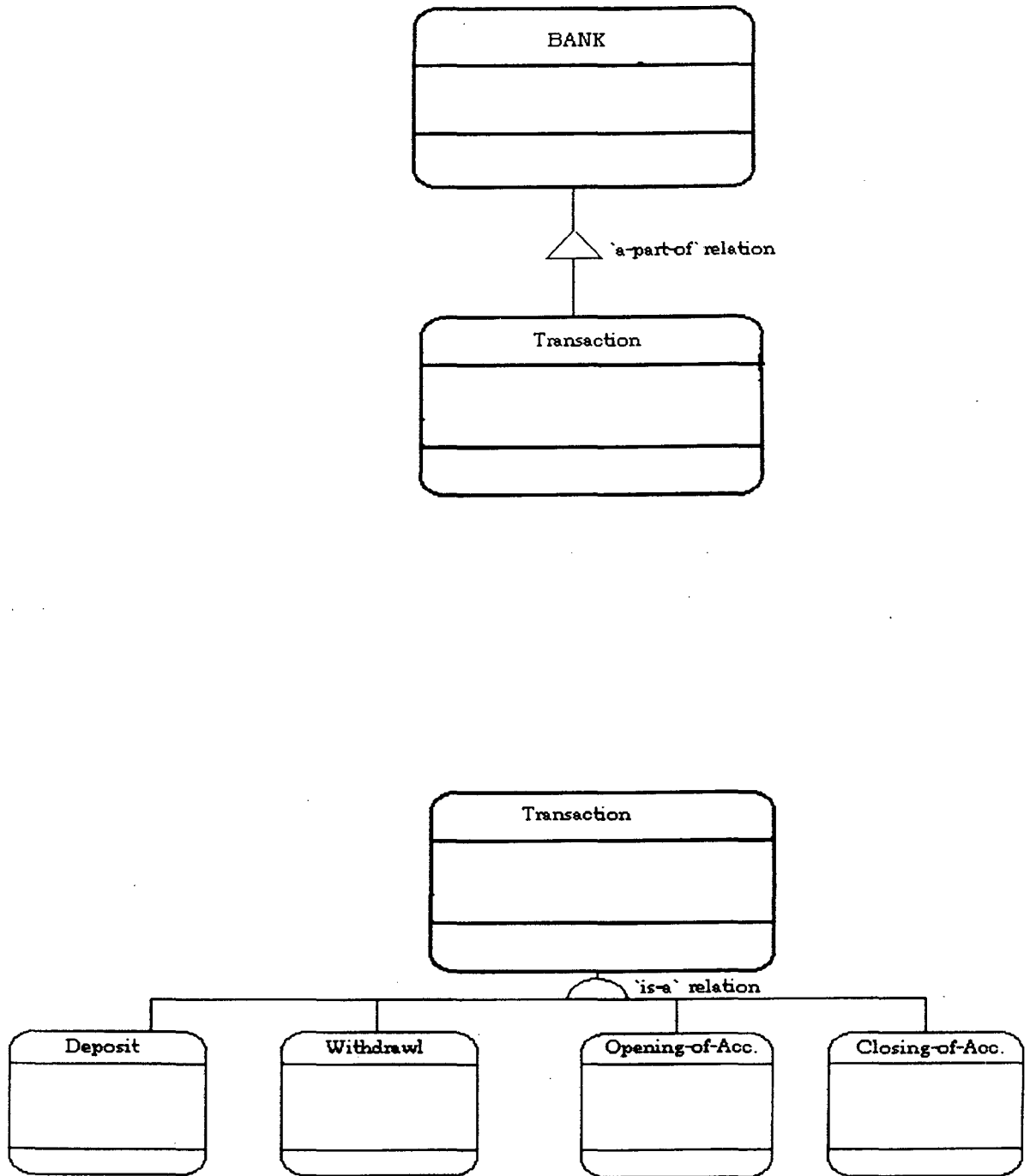


Fig (8) Structure Layer for BIS

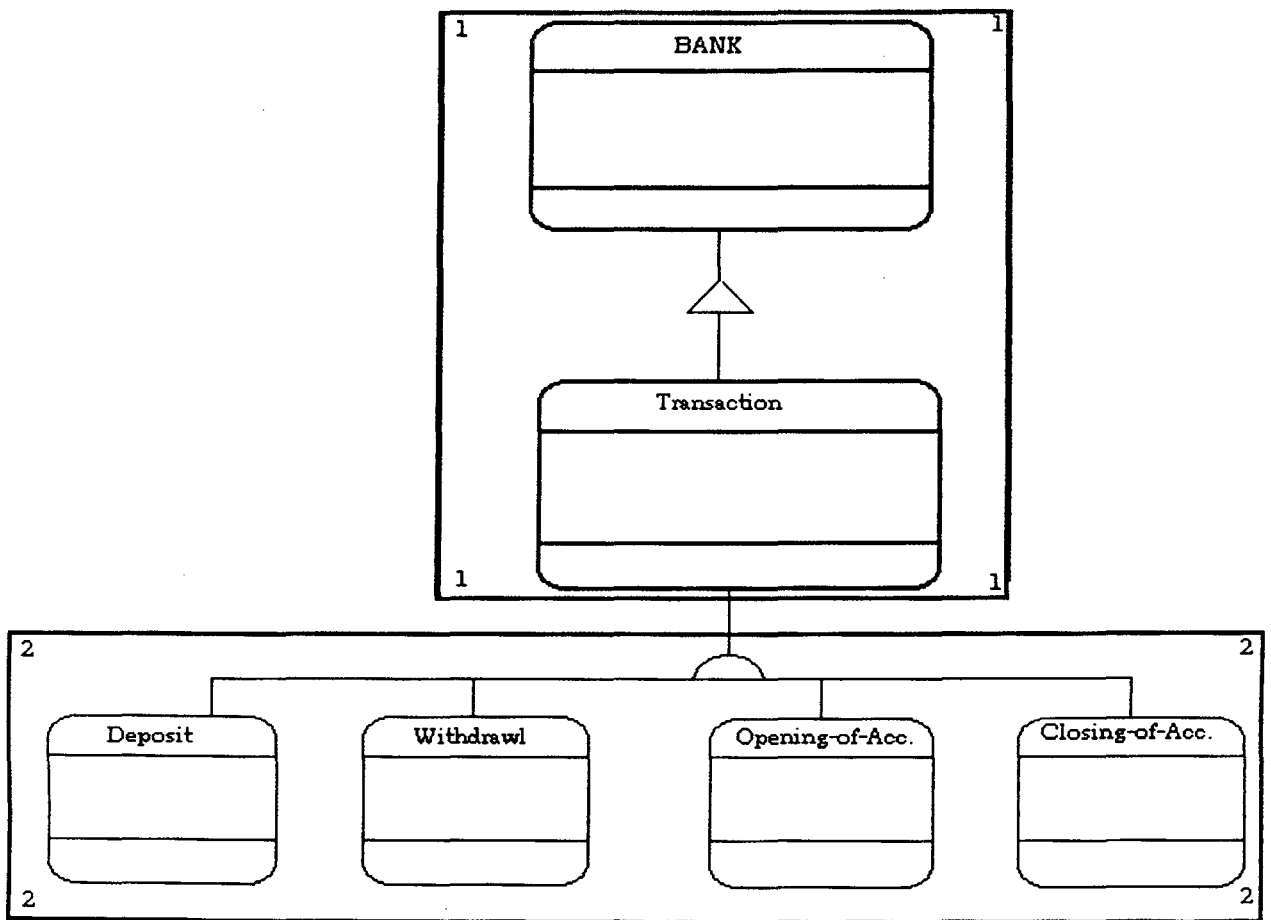


Fig (9) Subject layer for BIS

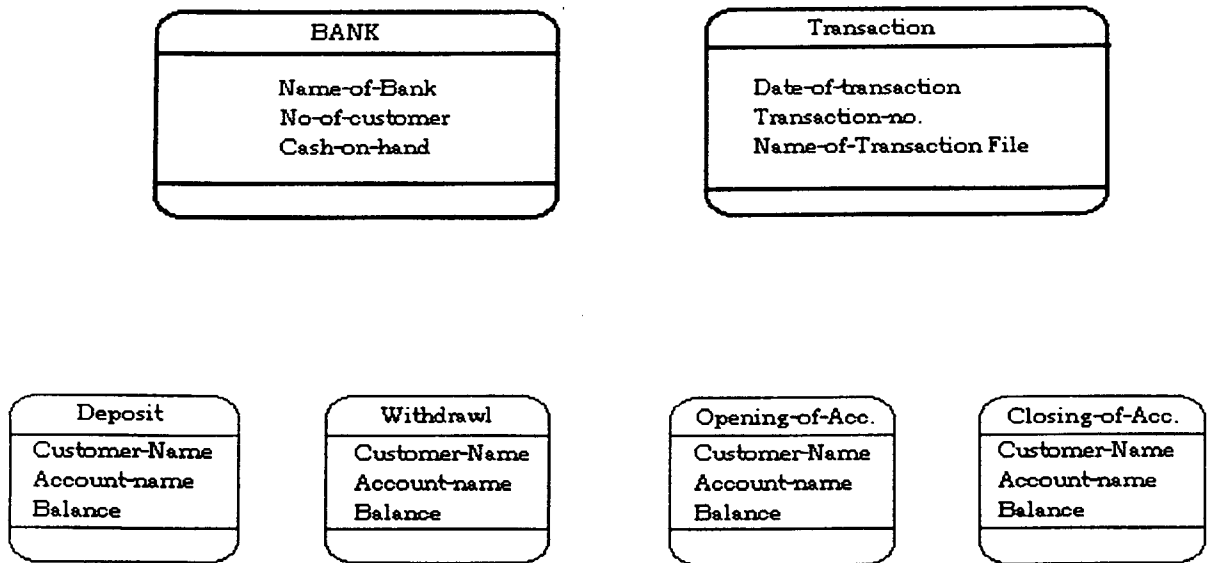


Fig .(10) Attribute layer for BIS

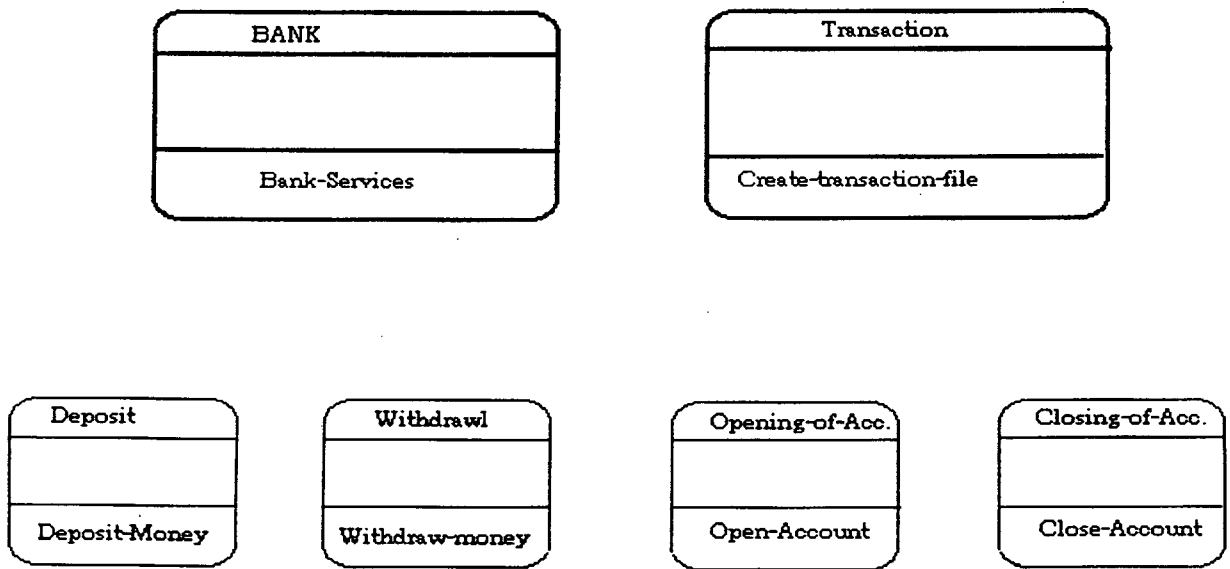
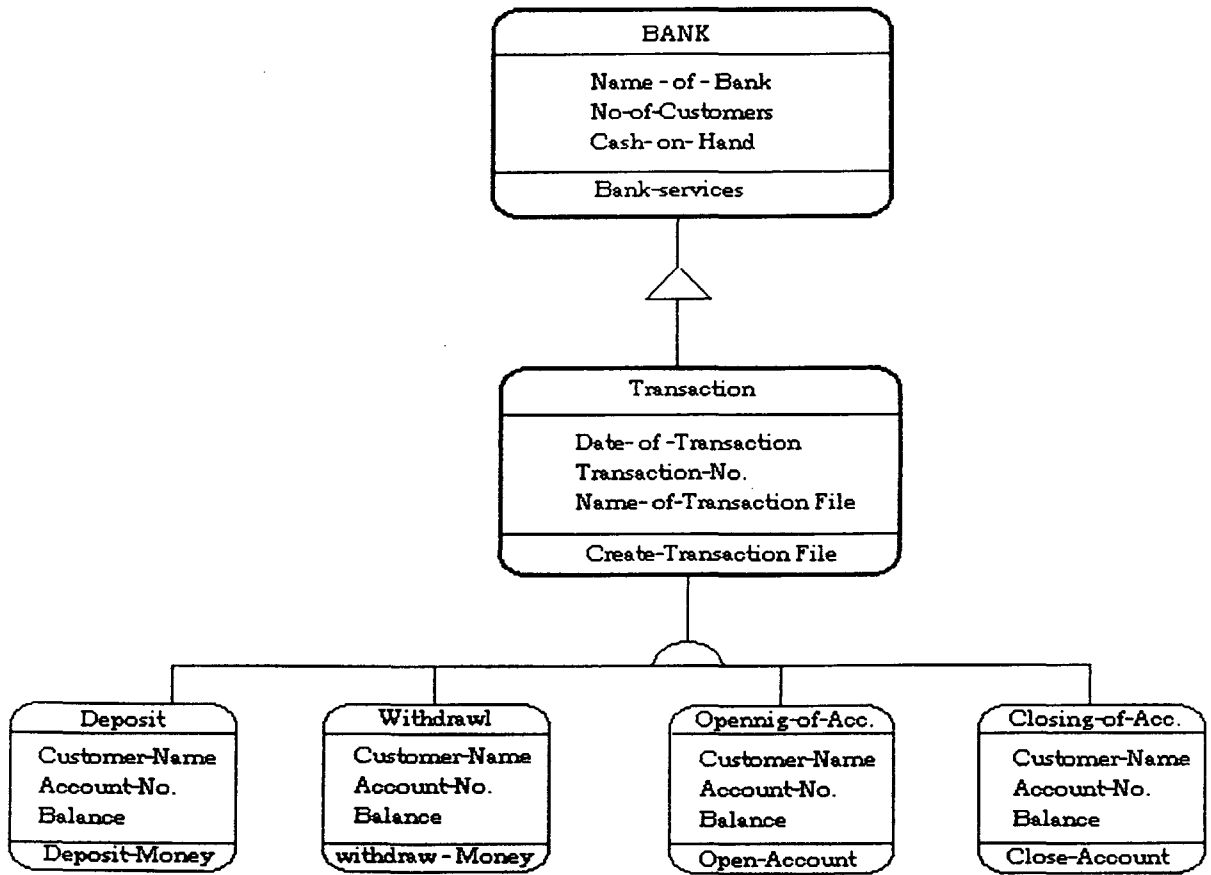


Fig (11) Services layer for BIS



Fig(12) Object Diagram for BIS

CHAPTER - 5

CODING

(5.1) FUNCTIONS OF CLASS MODULES

As I have defined my problem, so I have developed some of the class modules accordingly. These modules have been used in my program. These modules are capable of performing some or all the functions mentioned here. Following are the modules used :

(i) Module List:

1. Append a customer in the linked list
2. Delete a customer from the linked list
3. to search a customer in the linked list

List of node contains name of the customer and account number. For that the structure I have used is :

```
struct node
{
    char name[25];
    int accnumber;
    struct node *next;
};
```

(ii) Module Customer:-

1. Read customer name
2. Set account_no.=no where no is passed from outside of the module.

(iii) Module CustFile:

1. Checks if the customer.dat file exists or not if file does not exists then create it.
2. Open customer.dat file for reading
3. Close customer.dat file
4. Read control record
5. Read record for account number = accno, where accno supplied from outside.
6. Write the control record
7. Write the record at the end of customer.dat, for that search for record whose account number is less than one to the account number of records to be written on the customer.dat file.

The customer record contains Name of the customer, Account Number, Balance and one flag. For that the structure

used is :

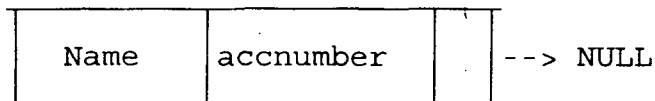
```
struct cust_rec
{
    char name[25]
    int accno;
    float balance;
    int delflag;
};
```

The control record contains cash-on-hand, number of customers and pointer to next customer record. For that the structure used is:

```
struct cntl_rec
{
    float cash-on-hand;
    int no-of-cust;
    int next_accno;
};
```

(iv) Module load list:

1. It uses, module llist and module custfile.
2. It returns new node of customer record with initialization of *lp.



So it loads data from file customer.dat to linked list and returns to header as first.

(v) Module Txn file:-

1. Uses module customer
2. If transaction file does not exist (fnamet) open it, if already opened, return the total no. of transactions.
3. Open the transaction file fnamet for reading
4. Close the transaction file
5. Read the transaction record for the supplied account number and return the transaction record.
6. Read the control of transaction record and return the transaction control record.
7. Go to the last of the transaction record and return the last transaction record.
8. Write to transaction control record
9. Write the transaction record file
10. Get commit flag for crash recovery
11. Get type of transaction

The translation record contains transaction type,

customer name and amount. The transaction control records contains number of transaction and commitflag for crash recovery.

Following is the structure used for transaction record:

```
struct TxnRec
{
    int transtype;
    char custname[20];
    float amount;
};
```

For the transaction control record, following structure is used:

```
struct TxnCntl
{
    int no_of_trans;
    int commitflag;
};
```

(vi) Module Account:

1. It uses module loadlist and module TxnFile
2. Read amount in Transaction record.
3. Deposits the amount into the customer's account. It checks that amount deposited is not negative and customer exists in the file (Transaction type=1).

4. This function withdraws the amount from the customer's account. It checks whether customer exists in the file and the amount to be withdrawn is not more than customer's balance.

(transaction type =2)

5. It opens account (new account), if customer already does not exist. (transaction type-3)

6. It closes the customer account, it checks whether customer already exists in the file. It tells to customer his net balance (Transaction type=4.)

(vii) Module Menu:-

1. Uses the public account

2. It displays main menu on the screen and accepts options, then according to the option, execute member function of module account.

(viii) Main Module:-

It calls module menu. If the commitflag=0 then customer.Bak is correct copy, so copy customer.Bak to

customer.dat, last transaction is automatically redone.

If commitflag=1 then transaction is done in customer.dat but might not have been done in customer.bak. So copy customer.dat to costomer.bak

RUNNING OF THE PROGRAM :

When I run the main program, following message will come:

BIS..

1. Deposit
2. Withdraw
3. Open New Account
4. Close Account
5. Press any other key to quit

Enter options (1...5)

Now the system is waiting for the input. If '1' is pressed the system will go into "Deposit". If '2' is pressed the system will go to the "Withdraw",if '3' is pressed then it goes to "Open Account", if '4' is pressed then "Close Account" and lastly if any other key is pressed the system

will quit.

Depending on the input the internal functions will be accomplished as already discussed in different modules.

(5.2) * PROGRAMMING ENVIRONMENT USED :

The use of object-oriented design is not limited to any one particular language. Here, we can not ignore the details of coding because ultimately our software will be expressed in some language. Some of the languages which satisfies the properties of object-oriented design approach are Smalltalk, Ada, Object Pascal and C++. The augmentation of pascal with object-oriented concepts has resulted in the language object pascal and Ada. The currently popular OOP language have combined to advantages of structured programming with the innovative concepts of object orientation. C++ and objective C are enhanced versions of C with object oriented features. Even the artificial intelligence language lisp has object-oriented versions in the languages LOOPS and CLOS.

Here in my coding I have used C++ as object-oriented programming language.

CHAPTER - 6

CONCLUSION

The object-oriented database systems are getting a lot of attention from various fields. Due to the large complexity of this field, it is no surprise that there will be continuous discussion about the features of object-oriented database systems. The work carried out in my dissertation work is just a snapshot in object-oriented system development methods. As discussed throughout the dissertation, we can say that object-oriented programming, object-oriented database systems, object oriented design and object-oriented analysis have reached a level of some maturity. In may case real object-oriented system has been analyzed, designed and coded.

While summarizing the discussion, the first question arises whether object-oriented concepts are answer to the many problems we face in developing system? I find that object orientation is not solution to all problems but it is new way of thinking about systems. Fresh insights are gained by looking at old problems in new ways, So, object oriented method has provided new solution approach for such problems

like reusability, concurrency. Also one question arises that the problem case (i.e. information system) which I have taken is best analyzed by object oriented method or not. I find that based on the requirements and taking into the consideration of the merits of object-oriented method, it is obvious that one should go for object-oriented approach. However it remains a very interesting question that deserves further research attention.

Now we conclude that the method presented in my work, provides a procedure for identifying objects and classes. How these classes and objects are related with each other is structured. For simplifying our problem case I have divided the whole problem in different sections which are termed as subjects. Procedures are also given for identifying attributes and operations. Specifically our method has used object diagram to represent the model. Our model has also captured the user's view of a system and reusability of object.

While doing the project what I found risky in design is the risk of system performance and design start-up

difficulty. Origin of system performance risk comes from object-oriented programming language and object-oriented database system in relation to traditional programming languages and database systems. Currently these implementation areas may preclude the use of object orientation in certain applications that demand high performance. Secondly the start-up difficulty may be the barrier for adopting this method. This start-up difficulty includes time and human stress. Because usually if some developer is using a particular object-oriented programming language for the first time, he has no established base of software. So he starts from scratch, how to interface his object orientations with existing non-object oriented ones, as I also felt through out the project work.

Last but not least I can say that our current object-oriented system development methods have revolutionary potential. However, important deficiencies, theoretical and practical, may hamper widespread use of object-oriented system development methods until they are remedied. So much additional research is needed to support the complete object-oriented system life cycle.

BIBLIOGRAPHY

1. B. Mayer, - Object-Oriented Software Construction, Prentice-Hall, New York, 1988.
2. Brehm, B. (1988)-A Siral Model of Software Development and Enhancement, IEEE Computer 21(5).
3. Booch, Grady,- "Object-Oriented Development"-IEEE Transactions on Software Engineering, February 1986.
4. Cardelli, L., and Wegner, P. (1985).-On Understanding Types, Data Abstraction, and Polymorphisom, : ACM Commuting Surveys 17(4).
5. Champeaux, Dennis De,-Toward an Object-Oriented Software Development Process HP Lab report HPL-92-149, Nov.-92.
6. Davis, G.B. and Olsm, M.H.,- Management Information Systems : Conceptual Foundations, Structure, and Development, (2nd ed), McGraw-Hill, New York, 1985.
7. Dickson, G.W. and Wetherbe, J.C.,-The Management of Information Systems, McGraw-Hill, New York, 1985.
8. Diebold, J., -"Information Resource Management : The next Challenge," Infosystems, Vol. 26, No. 6, June 1979.
9. Jalote, Pankaj,-"Functional Refinement and Nested Objects for Object-Oriented Design"-IEEE Transactions on Software Engineering, March 1989.

10. Karl Lieberherr, -"Formal Foundations for Object-Oriented Data Modelling,"-IEEE Transaction on Knowledge and Data Engineering, Vol. 5(3), June 1993.
11. Micallef, J.-April/May 1988:Encapsulation, Reusability, and Extensibility in Object-Oriented Programming Languages : Journal of Object-Oriented Programming Vol. 1(1).
12. Rainer Unland Gunter Schlageter-Object-Oriented Database Systems : Concepts & Perspectives.
13. Rambaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Loresen, W. (1991).- "Object-Oriented Modeling and Design" - Prentice-Hall, Englewood Cliffs, NJ.
14. Richard Baskerville - Information System Security Design Methods : Implementation for Information System Development - ACM Computing Surveys, Vol. 25, No. 4, Dec. 93.
15. Robson, D. - August 1981 : Object-Oriented Software Systems-Byte Vol. 6(8).
16. Salvatore T. March and Young-Gul Kim : Information Resource Management : Intergrating the Process - DATABASE - Summer 92.
17. Saunders, J. - March/April 1989 : A Survey of Object-Oriented Programming Languages - Journal of Object-Oriented Programming Vol. 1(6).
18. S.C. Bailin, - An Object-Oriented Requirements Specification Method, Commun. ACM 608-623 (1989).
19. Shdaer, S., and Mellor, S. (1988). - "Object-Oriented Systems Analysis" - Prentice-Hall, Englewood Cliffs, NJ.

20. Smith J., and Smith, D. - Database Abstractions : Aggregation and Generalization. ACM Transactions on Database Systems Vol. 2(2).
21. Unland, R. and Schlageter, G. - "An Object-Oriented Programming Environment for Advanced Database Applications," Journal of Object-Oriented Programming, May/June 1989.
22. Ward, P. (1989) - "How to Integrate Object-Orientation with structured Analysis and Design," - IEEE Software 8(1).
23. Wirfs-Brock, R., and Johnson, R. (1990) - "Surveying Current Research in Object-Oriented Design," Communications of the ACM 33(9).

PROGRAM

```
# include <dos.h>

# include <string.h>
# include <sys\stat.h>
# include <io.h>
# include <stdio.h>
# include <iostream.h>
# include <process.h>
# include <time.h>
# include <stdlib.h>

Struct node
{
    char name[25];
    int accnumber;
    struct node *next;
};

Class llist
{
protected:
    struct node *first, *temp,*forw,*prev;

public:
    void append (struct node*);
        // TO APPEND A CUSTOMER IN THE LINKED LIST.
    void delnode (char *);
        // TO DELETE A CUSTOMER FROM THE LINKED LIST.
    int search (Char *);
        // TO SEARCH A CUSTOMER IN THE LINKED LIST
};

Void llist :: append (struct node *a)
{
    temp = first;
    while (temp-> next != NULL)
```

```

        temp = temp-> next;
a-> next = NULL;
temp-> next = a;
    } ;

void    llist::delnode (char string[])
    {
    forw = first;
    prev = first;
    while (forw != NULL)
        if (strcmp (forw -> name, string))
            {

                prev = forw;
                forw = forw -> next;
            }
        else
            {
                prev -> next = forw -> next;
                delete (forw);
                return;
            }
    };

int    llist:: search (char string [25])
    {
    temp = first;
    do
        {
            if (strcmp (temp-> name, string))
                temp = temp -> next;
            else
                return (temp -> accnumber);
        }
    while (temp != NULL);
    return(-1);
    };

```

```

class customer
    {
    protected:
        char ccustname[25];
        int accno;
    public:
        void getcustomer()

```

```
        { cin >> custname; };  
  
void setaccno (int no)  
    {accno = no ; };  
  
};
```

```
struct cust_rec  
{  
    char    Name [25];  
    int     AccNo;  
    Float  Balance;  
    int    delflag;  
  
};
```

```
struct cntl_rec  
{  
    float cash_on_hand;  
    int no_of_cust;  
    int next_accno;  
  
};
```

Class CustFile

```
{  
protected:  
    struct cust_rec crec;  
    struct cntl_rec ccntl;  
    FILE *fp ;  
    char fname [25];  
  
public:  
    CustFile ();  
    void fileopen();  
    void fileclose();  
    cntl_rec readcntl();
```

```

    cust_rec readrec (int);
    void writecntl();
    void writerec (int);
    ~CustFile()
        // DESTRUCTOR FOR CLOSING THE CUSTOMER FILE
        {
            fileclose();
        }
};

CustFile:: CustFile()

    //THIS CONSTRUCTOR CHECKS WHETHER
    //CUSTOMER. DAT EXISTS OR NOT IF FILE DOES
    // NOT EXIST THEN CREATE IT.

    {
        strcpy (fname,"customer.dat");
        if (access(fname, 00) != 0 )
            {
                cout <<"\n Custfile not found creating new...\n"
                fp= fopen(fname,"w+" );
                ccntl.cash_on_hand = 1000 ;
                ccntl.no_of_cust = 0;
                ccntl.next_accno = 0;
                writecntl();
                fileclose();
            }
    };

void CustFile:: fileopen()
    {fp= fopen("customer.dat", "r+" );};

void CustFile:: fileclose()
    { fclose (fp);} ;

cntl_rec CustFile:: readcntl()
    {
        cntl_rec cntl;
        fseek (fp,0,0);
    }

```

```

        fread (&cntl, sizeof (struct cntl_rec),1,fp);
        return cntl;
    }

    cust_rec CustFile:: readrec (int accno)
    {
        cust_rec rec;

        fseek (fp, sizeof (cntl_rec),0);
        fseek (fp, (accno-1)* sizeof (cust_rec),1);
        fread (&rec, sizeof (cust_rec),1,fp);
        return rec;
    };

    void CustFile:: writecntl()
    {
        fseek(fp,0,0);
        fwrite(&cntl, sizeof (cntl_rec),1,fp);
    };

    void CustFile:: writerec (int accno)
    {
        fseek (fp, sizeof (cntl_rec),0);
        fsseek (fp, (accno-1)*sizeof (cust_rec),1);
        fwrite (&rec,sizeof (cust_rec), 1, fp);
    };

class loadlist: public llist, public CustFile
{
public:
    node* load (cust_rec *);
    loadlist();
};

node* loadlist:: load (cust_rec *lp)
{
    node* newnode;
    newnode = new (node);
    strcpy (newnode->name, lp->name);
    newnode->accnumber = lp->AccNo;
    newnode ->next = NULL;
    return newnode;
};

```

/* THIS CONSTRUCTOR LOADS DATA FROM FILE CUSTOMER.DAT TO THE LINKED LIST AND RETURNS THE HEADER AS FIRST*/

```
loadlist:: loadlist()
```

```

    {
        node *nodeptr;
        int lastacc;
        strcpy (crec.Name, " ");
        crec.AccNo = 0;
        first = load (&crec);
        fileopen();
        ccntl=readcntl();
        if ((lastacc==ccntl.next_accno)!=0)
            {
                for ( int j=1;j<=lastacc;j++)
                    {
                        crec=readrec(j);

                        if (crec.delflag ==0)
                            {
                                crec=readrec (j);
                                nodeptr = new (node);
                                nodeptr = load (&crec);
                                append (nodeptr);
                            }
                    }
            }
};

struct TxnRec
{
    int transtype;
    char custname[25];
    float amount;
};

struct TxnCntl
{
    int no_of_trans;
    int commitflag;
};

class TxnFile: public customer
{
    FILE *fpt;
    char fnamet[25];

```

```

protected:
    struct TxnRec trec;
    struct TxnCntl tcntl;
public:

    TxnFile()
    {
        strcpy (fnamet, "txnfile");
        if (access(fnamet,00) == -1)
        {
            cout <<"\n\n TXNFILE not found creating new.\n"

            fpt = fopen (fnamet, "w+");
            tcntl.no_of_trans=0;
            tcntl.commitflag=1;
            writecntlt ();
            cout <<"\n\n press any key to continue..\n";
            getch();
        }
        else
            fileopent ();
    };
~TxnFile()
{
    /* fseek (fpt, 0,0);
    fread (&tcntl,sizeof (struct TxnCntl),1,fpt);
    cout<<"\n\n Total no of transactions :";
    cout << tcntl.no_of_trans;

    */ closefilet ();
};

void fileopent ()
{
    fpt=fopen (fnamet,"r+");
};
void closefilet ()
{
    fclose (fpt);
};
TxnCntl readcntlt ();
TxnRec readrect (int);
TxnRec readlast ();
void writecntlt ();

```

```

void writerec ();
int getcommitflag ();
int gettxntype ();
char* getfilename();
};

TxnRec TxnFile:: readrect (int accno)
{
    TxnRec rec;
    fseek (fpt, sizeof (TxnCntl),0);
    fseek (fpt, (accno-1) * sizeof (TxnRec),1);
    fread (&rec, sizeof (TxnRec),1,fpt);
    return rec;
};

TxnCntl TxnFile: readcntlt()
{
    TxnCntl cntl;
    fseek (fpt,0,0)
    fread (&cntl, sizeof (TxnCntl),1, fpt);
    return cntl;
};

TxnRec TxnFile:: readlast()
{
    TxnRec rec;
    fseek (fpt,-i *sizeof (TxnRec), 2);
    fread (&rec, sizeof (TxnRec), 1, fpt);
    return rec;
};

void TxnFile:: writecntlt()
{
    fseek (fpt,0,0);
    fwrite(&cntl,sizeof(struct TxnCntl),1,fpt);
};

void TxnFile:: writerec()
{
    fseek (fpt, 0,2);
    fwrite (&rec,sizeof (TxnRec),1, fpt);
};

int TxnFile:: getcommitflag()

// GET COMMIT FLAG FOR CRASH RECOVERY

{

```



```

        tcntl=readcntl();
        return tcntl.commitflag;
    };
int TxnFile:: gettxntype()
{
    trec=readlast();
    strcpy (ccustname, trec.custname);
    return (trec.transtype);
};

```

```

class account: public loadlist, public TxnFile

```

```

{
    float Balance;
    public:
    float getamount()

    { cin >> trec.amount; return trec.amount;};
    void deposit ();
    void withdraw ();
    void openaccount ();
    void closeacc ();

};

```

/* THIS FUNCTION DEPOSITS THE AMOUNT INTO THE CUSTOMER'S ACCOUNT. IT CHECKS THAT AMOUNT DEPOSITED IS NOT NEGATIVE AND CUSTOMER EXISTS IN THE FILE*/

```

void account::deposit ()
{
    tcntl = readcntl();
    tcntl.no_of_trans++;
    tcntl.commitflag=0;
    writcntl();
    trec.transtype=1;
    strcpy (trec.custname, ccustname);
    writerec();
    ccntl=readcntl();
    ccntl.cash_on_hand +=trec.amount;
    writecntl();
    crec= readrec(accno);
    crec.balance+=trec.amount;
}

```

```

        writerec(accno);
        tcntl.commitflag=1;
        writecntl();
        system("copy customer.dat customer.bak>NUL");
    };

```

```

    /* THIS FUNCTION WITHDAWS THE AMOUNT FROM THE CUSTOMER'S
    ACCOUNT. IF CHECKS WAETHER CUSTOMER EXISTS IN THE FILE AND
    THE AMOUNT TO BE WITHDRAWN IS NOT MORE THAN CUSTOMER'S
    BALANCE.*/

```

```

void account ::withdraw()
{
    crec=readrec(accno);
    if (crec.Balance>trec.amount)
    {
        tcntl=readcntl();
        tcntl.no_of_trans++;
        tcntl.commitflag=0;
        writecntl();
        trec.transtype=2;
        strcpy(trec.custname,ccustname);
        writerec();
        ccntl=readcntl();
        ccntl.cash_on_hand-=trec.amount;
        writecntl();
        crec.Balance-=trec.amount;
        writerec(accno);
        tcntl.commitflag=1;
        writecntl();
        system("copy customer.dat customer.bak>NUL");
    }
    else
        cout<<"\n\n\n you can not withdraw more than you have.. \n"
};
/*THIS FUNCTION IS FOR OPENING NEW ACCOUNT.IT CHECKS IF CUSTOMER
EXISTS IN THE FILE*/
void account::openaccount()
{
    tcntl=readcntl();
    tcntl.no_of_trans++;
    tcntl.commitflag=0;
    writecntl();
    trec.transtype=3;
    strcpy(trec.custname,ccustname);
    writerec();
    ccntl=readcntl();
}

```

```

ccntl.cash_on_hand+=trec.amount;
ccntl.no_of_cust++;
ccntl.next_accno++;
writecntl();
strcpy(crec.Name, custname);
crec.Balance=trec.amount;
crec.delflag=0;
crec.AccNo=(cntl.next_accno);
writerec(crec.AccNo);
node *newnode;
newnode=new(node);
strcpy(newnode->name, ccustname);
newnode->accnumber=crec.AccNo;
newnode->next=NULL;
append(newnode);
tcntl.commitflag=1;
writecntlt();
system("copy customer.dat customer.bak>NUL");
};

```

/*THIS FUNCTION IS FOR CLOSING THE CUSTOMER ACCOUNT .IT CHECKS WHETHER THE CUSTOMER ALREADY EXISTS IN THE FILE.IT TELLS TO THE CUSTOMER HIS NET BALANCE*/

```

void account::closeacc()
{
    delnode(ccustname);
    erec=readrec(accno);
    trec.amount=crec.Balance;
    cout<<"\n\n\n amount you will get back :Rs. "<<trec.amount
    cout<<"/-";
    tcntl=readcntlt();
    tcntl.no_of_trans++;
    tcntl.commitflag=0;
    writecntlt();
    tree.transtype=4;
    strcpy(trec.custname, ccustname);
    writerec();
    ccntl=readcntl();
    ccntl.cash_on_hand-=trec.amount;
    ccntl.no_of_cust--;
    writecntl();
    crec=readrec(accno);
    crec.Balance-=trec.amount;
    crec.delflag=1;
    writerec(accno);
    tcntl.commitflag=1;
    writecntlt();
}

```

```
system("copy customer.dat customer.bak>NUL");
};
```

```
class menu:public account
```

```
/*THIS CLASS DISPLY MAIN MENU AND ACCEPTS OPTIONS THEN  
ACCORDING TO THE OPTION EXECUTE MEMBER FUNCTION OF ACCOUNT CLASS*/
```

```
{
    char choice;
public:
    void displaymenu();
    ~menu()
    {
        fileclose();
    };
    void setchoice(char cc)
    { choice=cc;
    };
    void executechoice();
    void Deposit();
    void Withdraw();
    void NewAcc();
    void DelAcc();
    };
    void menu :: displaymenu()
    {
        choice=' ';
        while(choice <'5')
        {
            clrscr();
            cout<< "\n\n\n\n\n\t";
            cout <<"BIS";
            cout <<"\n\n\t";
            cout <<"1. Deposit";
            cout<<"\n\n\t";
            cout<<"2. Withdraw";
            cout<<"\n\n\t";
            cout<<"3. Open new Account";
            cout<<"\n\n\t";
            cout <<"4. Close Account";
            cout<<"\n\n\t";
            cout<<"5. Press any other key to quit";
            cout<<"\n\n\t";
            cout<<"Enter option (1...5) :";
            cin>> choice;
        }
    }
};
```

```

        executechoice();
        if (choice<5)
        {
            cout<<"\n\n\n for main menu press any key";
            getch();
        }
};

void menu :: executechoice()
{
    switch (choice)
    {
        case '1':Deposit();
            break;
        case '2':Withdraw();
            break;
        case '3':NewAcc();
            break;
        case '4':DelAcc();
            break;
    }
};

void menu :: Deposit()
{
    clrscr();
    cout <<"\n\n\n\n\t\t DEPOSIT\n";
    cout <<"\n\n\n\n\t\t ENTER NAME OF CUSTOMER\n";
    getcustomer();
    cout <<"\n\n\t\t ENTER AMOUNT TO DEPOSIT :Rs.";
    if (getamount()>0)
    {
        accno=search(custname);
        if(accno==-1)
        cout <<"\n\n\nCUSTOMER NOT FOUND IN DATA BASE";
        else
        {
            deposit();
            cout <<"\n\n\n AMOUNT DEPOSITED..\n";
        }
        else
            cout <<"\n\n\n NEGATIVE AMOUNT..\n";
    }
};

void menu :: Withdraw()
{
    clrscr();
    cout <<"\n\n\n\n\t\t WITHDRAW\n ";
    cout <<"\n\n\n\n\t\t ENTER NAME OF CUSTOMER\n";
}

```

```

getcustomer();
cout <<"\n\n\t\t ENTRER AMOUNT TO WITHDRAW :Rs. " ;
if (getamount()>0)
{
accno=search(custname);
if(accno==-1)
cout <<"\n\n\nCUSTOMER NOT FOUND IN DATA BASE";
else
{
withdraw();
cout <<"\n\n\n AMOUNT WITHDRAWAL COMPLETED..\n"
}
else
cout <<"\n\n\n NEGATIVE AMOUNT..\n";
};
void menu :: NewAcc()
{
clrscr();
cout <<"\n\n\n\n\t\t OPEN NEW ACCOUNT\n";
cout <<"\n\n\n\n\t\t ENTER NAME OF CUSTOMER\n";
getcustomer();
cout <<"\n\n\t\t ENTRER AMOUNT :Rs.";
if (getamount()>0)
{
if(search(custname) !=-1)
cout <<"\n\n\n CUSTOMER WITH THIS NAME.";
else
openaccount();
cout<<"\n\n\n opened a new account:";
}
else
cout <<"\n\n\n NEGATIVE AMOUNT..\n";
};
void menu :: DelAcc()
{
clrscr();
cout <<"\n\n\n\n\t\t CLOSE ACCOUNT\n";
cout <<"\n\n\n\n\t\t ENTER NAME OF CUSTOMER\n";
getcustomer();
accno=search(ccustname);
if(accno==-1)
cout <<"\n\n\n\nCUSTOMER NOT FOUND IN DATA BASE";
else
{
closeacc();
}
};

```

```

void main( )
{
menu menuM;
/* IF COMMITFLAG = 0 THEN CUSTOMER.BAK IS CORRECT.COPY
CUSTOMER.BAK TO CUSTOMER.DAT.LAST TRANSACTION IS
AUTOMATICALLY REDONE.IF COMMITFLAG = 1 THEN TRANSACTION
IS DONE IN CUSTOMER.DAT BUT MIGHT NOT HAVE BEEN DONE IN
CUSTOMER.BAK SO COPY CUSTOMER.DAT TO CUSTOMER.BAK */

if (menuM.getcommitflag()==0 {
system("copy customer.bak customer.dat>NUL") ;
menuM.setchoice(menuM.gettxntype());
menuM.executechoice();
menuM.displaymenu();
}
else
{
system("copy customer.dat customer.bak>NUL") ;
menuM.displaymenu() ;
}
}

```