1. Thinking Object Oriented

1.1 Object Oriented Programming a New Paradigm

Procedural-oriented programming (POP) created a step by step program that guides the application through a sequence of instructions. Each instructions is executed in order. Procedural programming also focuses on the idea that all the algorithms are executed with functions and data that the programmer has access to and is able to change, where more attention is given to function and this leads to the bugs.

To remove the flaws encountered in POP an OOP was introduced as a new model or example with organizational approach among programming paradigms and still means different things to different people, however deals with decomposition of problem into a number of interacting agents called objects and their interaction to solve the problem.

So Object-Oriented Programming (OOP) is a programming paradigm that uses “objects” and their interaction to design applications and computer programs. An object commonly means a data structure consisting of data fields and procedures (methods) that can manipulate those fields. Typically, when calling a method from same object, the object itself should be passed as an parameter to the method.

OOP is much more similar to the way the real world works; it is analogous to the human brain. Each program is made up of many entities called objects. Objects becomes the fundamental units and have behavior, or a specific purpose, associated with them.

Objects cannot directly access other object’s data. Instead a message must be sent requesting the data, just like people must ask one another for information: we cannot see inside each others’ heads.

Benefits of OOP includes:

1. Ability to simulate real-world event much more effectively.
2. Code is reusable thus less code may have to be written.
3. Data becomes active.
4. Better able to create GUI applications.
5. Programmers are able to reach their goals faster.
6. Programmers are able to create more faster, more accurate and better-written applications (in case of a veteran programmer, by a factor of as much as 20 times compared with a procedural programming.

Features of OOP are:

1. Emphasis on data rather than procedure.
2. sdData structures are designed such that they characterized the objects.
3. Function that operate on the data of an object are type together in data structure.
4. Object may communicate each other through function.
5. Data is hidden and can't be accessed by external functions.
6. New data and functions can be easily added whenever necessary.
7. Follow bottom up approach in programming design.

1.2 A way of viewing World Agent

As in handling real world situations, OOP is also based on interacting agents and their communications where the terms agent, message, responsibility and methods have major roles.

To illustrate the idea, let us suppose, I wish to send flowers to my grandmothers for her birthday. She lives in a city many miles away, so my picking the flowers and carrying them to her doors by myself is out of the question. Nevertheless, sending her the flowers in an easy way; for that I merely go down to my local florist (named flo), tell her kinds and the number of flowers I want to send and my grandmother's address, and I can be assured the flower will be delivered expediently and automatically.

1.2.1 Agents, Responsibility, Messages and Methods

The mechanism I used to solve my problem was to find an appropriate agent and to pass to her message containing my request. It is the responsibility of Flo to satisfy my request. There is some method used by Flo to do this. I do not need to know the particular method she will use to satisfy my request. This information is usually hidden from my inspection.

So, our first principle of object oriented problem solving is the vehicle by which activities are initiated. Actions is initiated in OOP by the transmission of message to an agent (an object) responsible for the action and is accompanied by any additional information (arguments) needed to carry out the request. The receiver is agent to whom the message is sent. If the receiver accepts the message, it accepts the

responsibility to carry out the indicated actions. In response to a message, the receiver will perform some method to satisfy the request.

1.2.2 Responsibility

Responsibilities permits greater greater independence between agents, a critical factor in solving complex program. The entire collection of responsibilities is often described by the term protocol.

1.2.3 Classes and Instances

All objects are instances of class and classes are collection of objects of similar type. The method invoked by an object in response to a message is determined by the class of the receiver. All objects of a class use the same method in response to similar message.

1.2.4 Class Hierarchies- Inheritance

The categories of florist is more specialized form of the category of shopkeeper. Any knowledge of shopkeeper is also true of florist and hence of Flo.

Flo is florist, but florist is specialized form of shopkeeper. Furthermore, shopkeeper is also human. A human is mammal and mammal is an animal and animal is material object (it has mass and weight).

The principle that knowledge of a more general category is also applicable to a more specific category is called inheritance. The class florist will inherit attributes of the class or category shopkeeper.

1.2.5 Computation as Simulation

The traditional model describing the behavior of a computer executing a program is a process-state or pigeon-hole model. In this view, the computer is a data manager, following same pattern of instruction, wandering through memory, pulling values out of various slots (memory address), transforming them in some manner, and pushing the result back into other slots.

In contrast, in the object-oriented framework we never mention memory address, variables, assignments, or any of the conventional programming terms,. Instead, we speak of objects, messages, and responsibilities for some action, This view of programming is in many ways similar to a style of Computer simulation (process of representing real situation by computer) called “discrete event-driven simulation.”

In brief, in a discrete event- driven simulation the user crates computer models of the various elements of the simulation, describes how they will interact with one another, and sets them moving. This is almost identical to the average object-oriented program, in which the user describes what the various entities in the universe for the program are, and how they will interact with one another, and finally sets them in motion. Thus OOP can be viewed as a Computation Simulation.

1.4 Coping with Complexity

People deal with complex artifacts and situation every day, they nevertheless experience complexity if not yet have created computer program (Software). Because of the non linear behavior of complexity, the software development task is really complex. However, by using technique called abstraction complexity can be managed.

1.4.1 The Non-Linear behavior of Complexity

Complexity means difficult to understand. Software is inherently complex i.e complexity of software development task is essential property not an accidental one, because software complexity evolved form different factors such as:

* problem domain
* development process management
* software flexibility
* Change in digital system technology

In computer programming, we have the view that it exhibits nonlinear behavior of complexity and is not easy to dealing successfully (coping). However these constraints can be addressed by abstraction mechanism.

1.4.2 Abstraction Mechanism

Abstraction Mechanism

Layers of Abstraction Other forms of Abstraction History of

Abstraction

-Higher Lever	- Division into parts	- Procedure
-Next Level	- Encapsulation	-Module
-Next Level	- Interface and implementation	- ADT
-Last Level	- The service view	- Object
- Composition		
- Pattern		

The term Abstraction refers to the act of representing essential features without including unnecessary detail. Programmer have had to deal with the problem of complexity for a long time. To understand more fully the importance of object-oriented techniques, we should review the variety of mechanism programmer have used to control complexity. Among them abstraction is one of the most basic tools, the ability to encapsulate and isolate design and exaction information. To create, understand and manage complex system, abstraction mechanism can be used in varieties of layers and forms. However, one should know that the object oriented techniques are not at all revolutionary, but can be seen as natural outcome of a long historical progression form procedures, to modules, to abstract data types, and finally to object which can be described as history of abstraction mechanism.

Layers of Abstraction

In OOP, different levels of abstraction can be used to manage complexity of a problem where lower the level search for more detail information as:

Higher Level: viewed as community of object that interact with each other to achieve their common goal.

Next Level: a group of objects working together combined into an unit which perform similar types of action.

Next Level: deals with interaction between two objects

Last Level: deals with activity of single object

Other forms of abstraction

Idea of abstraction can be further subdivided into different forms to implement in OOP as:

Division into parts: A problem is divided into parts and each parts are analyzed individually as a single unit.

Encapsulation and Interchangeability: Encapsulation is the process of wrapping up related data and method into a single unit which further permits possibility of interchangeability.

Interface and Implementation: Interface describes what a system is designed to do where as implementation describes how the task is performed by the system.

The service view: an object is provide a service that is used by other object, the service view and the relationship between the objects give the clear picture about the system.

Composition: Technique used to create a system from various single parts.

Pattern: approach to solve the problem with the help of previous solution.

History of Abstraction

Procedures:

Procedures functions were first abstraction mechanism to be widely used in programming languages. Procedures allowed the tasks that were executed repeatedly, or executed with only slight variation, to be collected in one place and reused rather than being duplicated several times. The procedure gave the first possibility of information hiding.

One programmer could write that was used by many others. The other programmers did not need to know the exact details of the implementation- they needed only the necessary interface. But procedures were not effective mechanism for information hiding, and they only partially solved the problem of of multiple programmers making use of the same names.

Modules:

Modules can be viewed simply as an improved technique for creating and managing collections of names and their associated values. A module provides the ability to divide a name space into two parts. The public part is accessible only within module.

Suppose a programmer announces that he has developed a new types of numeric abstraction called Complex. He has defined the arithmetic operation for complex numbers- addition, subtraction, multiplication and so on, and had defined routines to convert number from conventional to complex number will be manipulated. Modules provide an effective method of information hiding, but they do not allow us to perform instantiation, which is the ability to make multiple copies if data areas.

Abstract data types (ADT):

An abstract data types is a programmer defined data types that can be manipulated like the system defined data types. As with system defined data types, as abstract data types correspond to set of legal data values and a number of primitive operation that can be performed on those values. User can create variables with value that range over the set of legal values and can operate on those values using the defined operations. To build the abstract data types, we must able to:

1. Export a type definition
2. Make available a set of operations that can be used to manipulate instances of the type.
3. Protect the data associated with the type so that they can be operated on only by the provided routines.
4. Make multiple instances of data type.

Objects:

The central concept of OOP is the object which is a kind of module containing data and sub-routines. An object is a kind of self-sufficient entity that has an internal state (the data it contains) and that can respond to messages (calls to its subroutines). Programming problem is analyzed in terms of objects and the nature of communication between them. They may represent a person, a place, a bank account, a table or any item that the program has to handle.

Basic Concepts of OOP

General concepts/characteristics used in OOP are

1. Objects and Classes
2. Data abstraction and encapsulation
3. Inheritance
4. Polymorphism
5. Dynamic Binding
6. Message Passing