**Ch-1**

**Procedure**

A procedure is a block of code that performs a specific task. Procedure may not return value. It is a building block of a computer program but is a somewhat old fashioned phrase as nowadays computer languages use functions not procedures.

1. Same as routine, subroutine, and function. A procedure is a section of a program
2. that performs a specific task.

(2) An ordered set of tasks for performing some action.

A set of instructions that performs a specific task for a main routine, requiring direction back to the proper place in the main routine on completion of the task.

A section of a computer program that is stored only once but can be used when required at several different points in the program, thus saving space also called **procedure**

## Object

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that we find in everyday life.

An object is a location in memory having a value and referenced by an identifier. Object can be a table or column in a relational database, or an association between data and a database entity (such as relating a person's age to a specific person).

Objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process. Each object is made into a generic class of object and even more generic classes are defined so that objects can share models and reuse the class definitions in their code. Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables. An object is what actually runs in the computer.

## Properties of an object

**Three properties characterize objects:**

**Identity: the property of an object that distinguishes it from other objects**

**State: describes the data stored in the object**

**Behavior: describes the methods in the object's interface by which the object can be used**

Real-world objects share two characteristics: They all have *state* and *behavior*. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail).

objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state ­in *fields* (variables in some programming languages) and exposes its behavior through *methods* (functions in some programming languages).

**Event-Oriented Languages**

It is a programming paradigm in which the flow of program execution is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.

 An event driven application is designed to detect events as they occur and then deal with them using appropriate event handling procedure. An event-oriented language implies that an application (the computer program) waits for an event to occur before taking any action. These events include user input events in graphical user interface. Events are actions that are performed during application usage.

It might be the press of a key on the keyboard or the click of a mouse button.  With these events (these are many types), the computer waits for a key press or a mouse click (pushing a button on a hand-held mouse).

C# is **event driven**. You’ll write programs that respond to user-initiated **events** such as

mouse clicks, keystrokes, timer expirations and—new in Visual C# 2012—*touches* and *finger swipes*—gestures that are widely used on smart phones and tablets.

Event-oriented languages became possible with the advent of the Macintosh operating system for Apple Macintosh computers and Microsoft Windows for MS-DOS computer systems.  Both environments were designed to bring hardware and software together into a standard user interface by employing a graphical user interface, or GUI.

**Subject-oriented programming**

Subject means a collection of state and behavior specifications reflecting a particular gestalt, a perception of the world at large.

Subject-Oriented Programming is an object-oriented software paradigm in which the state (fields) and behavior (methods) of objects are not seen as intrinsic to the objects themselves, but are provided by various subjective perceptions (“subjects”) of the objects.

In the subject-oriented paradigm, each application is a subject or a composition of subjects: it defines just the state and behavior pertinent to the application itself, usually fragments of the state and behavior of collections of relevant classes. It discusses general characteristics of subject-oriented programming without introducing a specific model.

Subject-oriented programming advocates the organization of the classes that describe objects into “subjects”, which may be composed to form larger subjects. At points of access to fields or methods, several subjects’ contributions may be composed. These points were characterized as the join-points of the subjects. For example, if a tree is cut-down, the methods involved may need to join behavior in the bird and tax-assessor’s subjects with that of the tree’s own. It is therefore fundamentally a view of the compositional nature of software development, as opposed to the algorithmic (procedural) or representation-hiding (object) nature.

Subject-Oriented Programming is radical departure from the Object oriented as follows. In OO, objects are defined in terms of intrinsic (i.e. based on a model that independently describes it). and based on this its attributes (properties) and methods (behavior) are derived. The application makes only the use of these properties and behavior. Contrary to this, in subject oriented programming, no object exists (and modeled) in such an isolation. In the process, but behaviors of the objects has been provided by the various other "subjects” of the objects which are beyond the scope and control of the author of the original object. Think of it as a way of extending various "independently definable behaviors" on the object. I think this would be way beyond defining templates of inheritance compared to what is being discussed here.

## Relationship to aspect-oriented programming

* the aspect program comprises both a) a cross-cut that comprises a point in the execution where cross-cutting behavior is to be included; and b) a cross-cut action comprising a piece of implementation associated with the cross-cut, the piece of implementation comprising computer readable program code that implements the cross-cutting behavior.
* the aspect transparently forces the cross-cutting behavior on object classes and other software entities

In the subject-oriented view, the cross-cut may be placed separately from the aspect (subject) and the behavior is not forced by the aspect, but governed by rules of composition. Hindsight makes it also possible to distinguish aspect-oriented programming by its introduction and exploitation of the concept of a query-like point cut to externally impose the join-points used by aspects in general ways.

In the presentation of subject-oriented programming, the join-points were deliberately restricted to field access and method call on the grounds that those were the points at which well-designed frameworks were designed to admit functional extension. The use of externally imposed point cuts is an important linguistic capability, but remains one of the most controversial features of aspect-oriented programming.

# Aspect Oriented Programming

# It is programming technique that makes it possible to clearly express those programs that OOP fail to support. Some design decisions have been so difficult to cleanly capture in actual code. AOP involves such aspects, including appropriate isolation, composition and reuse of the aspect code.

# Components are properties of a system, for which the implementation can be cleanly encapsulated in a generalized procedure. Aspects are properties for which the implementation cannot be cleanly encapsulated in a generalized procedure. Aspects and cross-cut components cross-cut each other in a system’s implementation.

# AOP technology that supports clean abstraction and composition of both components and aspects.

# Aspect oriented programming has originated from the concept of "Separation of concerns". Basically it extends the either procedural or object oriented programming for concerns which are cross-cutting. The software has functional requirements and non-functional ones.

# These cross cutting requirements includes examples like logging, exception handling, thread synchronization, memory management, optimization and so on. These cross cutting ASPECTS should be expressed and implemented separately and independently to any other functional parts. each such concerns or aspects can be independent from each other forming multidimensional "concern space". Good practical implementations of Aspect Oriented AspectJ and AspectC++

# Integrated development environment

# Abbreviated as IDE, a programming environment integrated into a software application that provides a GUI builder, a text or code editor, a compiler and/or interpreter and a debugger. Visual Studio, Delphi, JBuilder, FrontPage and DreamWeaver are all examples of IDEs.

# An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Several modern IDEs integrate with Intelli-sense coding features.

# Some IDEs contain a compiler, interpreter, or both, such as Net Beans and Eclipse; others do not, such as SharpDevelop and Lazarus. The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a version control system and various tools are integrated to simplify the construction of a GUI. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use with object-oriented software development

# IDEs are designed to maximize programmer productivity by providing tight-knit components with similar user interfaces. IDEs present a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software. This contrasts with software development using unrelated tools, such as vi, GCC or make.

# One aim of the IDE is to reduce the configuration necessary to piece together multiple development utilities, instead providing the same set of capabilities as a cohesive unit. Reducing that setup time can increase developer productivity, in cases where learning to use the IDE is faster than manually integrating all of the individual tools. Tighter integration of all development tasks has the potential to improve overall productivity beyond just helping with setup tasks. For example, code can be continuously parsed while it is being edited, providing instant feedback when syntax errors are introduced. That can speed learning a new programming language and its associated libraries.

# Some IDEs are dedicated to a specific programming language, allowing a feature set that most closely matches the programming paradigms of the language. However, there are many multiple-language IDEs, such as Eclipse, ActiveState Komodo, IntelliJ IDEA, Oracle JDeveloper, NetBeans, Microsoft Visual Studio, GenuitecMyEclipse and WinDev. Xcode is dedicated to a closed set of programming languages.

# Visual programming language

# A visual programming language (VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols, used either as elements of syntax or secondary notation.

# Visual programming language (VPL) is a programming language that uses graphical elements and figures to develop a program. VPL employs techniques to design a software program in two or more dimensions, and includes graphical elements, text, symbols and icons within its programming context. Visual programming language is also known as executable graphics language.

# Visual programming language enables the development of software programs by eliminating textual software code with a series of visual graphics elements. VPL incorporates these graphical elements as the primary context of the language arranged in a systematic order. The graphics or icons included within a visual program serve as input, activities, connections and/or output of the program. Visual language has a few types, such as icon-based languages, diagramming languages and form-based language. Visual languages should not be confused with GUI-based programming language as they only provide graphical program authoring services. However, their code/context is completely textual. Kodu, Blockly and executable UML are popular examples of visual programming languages.

**What is an application object**

This is a component of the top portion of the Application Layer defined by the manufacturer that actually implements the application

Application object is used to store and access information through variables from any page in application. Just like the session object. The difference is that all users share one application object (with session there is one session object for each user). The application object holds information that can access throughout all the pages in application (like database connection information) and we can change the application object in one place those changes automatically reflected in all the pages.

You can create application objects in Global.asax file and access those variables throughout the application.

The application object's events are described below:

**Events**

|  |  |
| --- | --- |
| Application\_OnStart | Occurs before the first new session is created(when the application object is first referenced) |
| Application\_OnEnd | Occurs when all user sessions are over, and the application ends |

**Main Window object**

A main window is a visual area containing some kind of user interface. It usually has a rectangular shape that can overlap with the area of other windows. It displays the output of and may allow input to one or more processes. Windows are primarily associated with graphical displays, where they can be manipulated with a pointer.

The window object represents an open window in GUI. MainWindow is automatically set with a reference to the first Window object to be instantiated in the AppDomain.

Windows are two dimensional objects arranged on a plane called the desktop. In a modern full-featured windowing system they can be resized, moved, hidden, restored or closed.

Main windows usually include other graphical objects, possibly including a menu-bar, toolbars, controls, icons and often a working area for showing relevant information.

The working area of a single document interface holds only one main object. "Child windows" in multiple document interfaces, and tabs for example in many web browsers, can make several similar documents or main objects available within a single main application window.

The Main window contains:

Menu bar: the Main window has six menus: File, Edit, Operations, View, Window, and Help.

Standard toolbar: by default located just below the menu bar.This toolbar provides short cuts for some regularly used menu commands.

Status bar: located at the bottom of the Main window.

# View Object

# The **View** object represents a result set obtained when a query processes using the OpenView method of the Database object.

# A view object is a class that lets to define and work with a set of rows, often in service of a user interface. Typically, a view object contains a SQL query that selects data from a database. It can be associated with underlying entity objects so we can modify data in the database, or not be associated with entity objects at all.

# View objects are based on what the client needs to display

# A view object uses a SQL query to specify filtered subsets of business data that can be related to attributes from entity objects. You create views based on what the client needs to display. The views of data can be based on but are independent of the underlying entity objects, enabling flexible data retrieval to support the required UI. In other words, you can query a set of data exactly as you want to show it in the display. The view object defines the attributes of the view row class, which represents a row in the query result, and optionally refers to underlying entity objects. View objects provide clients with row sets they can scroll through and update without concern for or knowledge of the underlying entity objects. Clients manipulate data by navigating through the result set, getting and setting attribute values; changes are made to the data in the underlying database when the transaction is committed. Relationships between view objects are expressed using view links. Each view object provides a default iterator that you can use to navigate through its result set.

# For example, the following figure shows the relationship between a view object, entity object, and the underlying database table. A view object named EmpNames operates on the Emp entity object to provide a view of the EMPNO and ENAME columns of the EMP table.

# C:\Users\generic\Desktop\view_object.gif

View objects are often used to:

* Provide an additional level of security by restricting access to a predetermined set of rows and columns. For example, you could create a view object where columns containing sensitive data (such as salaries) are not selected.
* Hide data complexity. For example, a view object can display columns or rows from multiple entity objects. Such a view object hides the fact that the data is coming from several tables.
* Customize presentation. Using a view object, you can rename columns without affecting the entity objects on which the view object is based.
* Store complex queries. A query could perform extensive calculations on table data. By saving this query in a view object, the calculations are performed only when the view object's query is executed. The calculation is executed before the data is retrieved from the database.
* Improve efficiency of the application by using fast-executing, optimized SQL, selecting only the data you need.

**Document Object**

Document object is a document that represents object in HTML, XHTML, and XML documents. The nodes of every document are organized in a tree structure, called the DOM tree. The HTML document object is the owner of all other objects in web page.

When an HTML document is loaded into a web browser, it becomes a **document object**. The document object is the root node of the HTML document and the "owner" of all other nodes:(element nodes, text nodes, attribute nodes, and comment nodes).The document object provides properties and methods to access all node objects, from within JavaScript. The document is a part of the window object and can be accessed as window.document.

***Depending on the kind of the document (e.g. HTML or XML) different APIs may be available on the document object***.

All document objects implement the DOM Core Document and Node interfaces, meaning that the "core" properties and methods are available for all kinds of documents.

In addition to the generalized DOM Core document interface, HTML documents also implement the HTMLDocument interface, which is a more specialized interface for dealing with HTML documents (e.g. document.cookie, document.alinkColor).

XML documents (available to Mozilla add-on and application developers) implement their own additions to the core Document functionality.

**It defines the logical structure of documents and the way a document is accessed and manipulated.**

|  |  |
| --- | --- |
| **Properties** | **Description** |
| document.body | Returns the BODY node of the current document. |
| document.cookie | Returns all name/value pairs of cookies in the document |
| document.forms | Returns a collection of all the forms in the document |

|  |  |
| --- | --- |
| **Methods** | **Description** |
| document.close() | Closes the output stream previously opened with document.open() |
| document.open() | Opens an HTML output stream to collect output from document.write() |

The DOM is a programming API for documents. It is based on an object structure that closely resembles the structure of the documents it model. For instance, consider this table, taken from an XHTML document:

**Document-view architecture**

The document-view architecture is a Microsoft way of separating how and where a programs data ( objects ) are stored and how they are viewed.

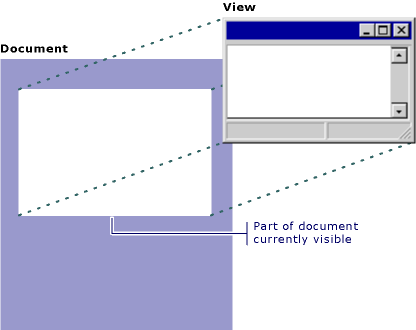
By default, the MFC(Microsoft Foundation Classes) Application Wizard creates an application skeleton with a document class and a view class. MFC separates data management into these two classes. The document stores the data and manages printing the data and coordinates updating multiple views of the data. The view displays the data and manages user interaction with it, including selection and editing.

In this model, an MFC document object reads and writes data to persistent storage. The document may also provide an interface to the data wherever it resides (such as in a database).A separate view object manages data display, from rendering the data in a window to user selection and editing of data. The view obtains display data from the document and communicates back to the document any data changes.

While you can easily override or ignore the document/view separation, there are compelling reasons to follow this model in most cases. One of the best is when you need multiple views of the same document, such as both a spreadsheet and a chart view. The document/view model lets a separate view object represent each view of the data, while code common to all views (such as a calculation engine) can reside in the document. The document also takes on the task of updating all views whenever the data changes.

The MFC document/view architecture makes it easy to support multiple views, multiple document types, splitter windows, and other valuable user-interface features.

The following figure shows the relationship between a document and its view.



**Advantages of the Document/View Architecture**

The key advantage to using the MFC document/view architecture is that the architecture supports multiple views of the same document particularly well.

Suppose your application lets users view numerical data either in spreadsheet form or in chart form. A user might want to see simultaneously both the raw data, in spreadsheet form, and a chart that results from the data. You display these separate views in separate frame windows or in splitter panes within a single window.

Both views(spreadsheet view and chart view) would be associated with a single document object. The document stores the data (or perhaps obtains it from a database). Both views access the document and display the data they retrieve from it.

That function notifies all of the document's views, and each view updates itself using the latest data from the document.