General Informatics

Chapter 4. Object Oriented Programming And Windows Basics

Visual Basic Projects

A ***project*** is the collection of files you use to build an application. A VB project (stored in a file with **.vbp** extension) can include different types of files and objects.

In VB the object and class names are shown in the property window. All objects are created as identical copies of their class. Once they exist as individual object, their properties can be changed. In programming an object provide code you don't have to write. Each Object in VB is defined by a class.

Visual Basic Projects

A project can include different types of files and objects:
- **form modules (.frm and .frx);**
- **class modules (.cls);**
- **standard modules (.bas);**
- **resource files (.res);**
- **ActiveX Documents (.dob);**
- **user Control and Property Page Modules (.ctl, .pag);**
- **components;**
- **ActiveX Controls (.ocx);**
- **Insertable Objects (Worksheets, Word Documents, …)**
- **ActiveX Designers;**
- **Standard Controls (toolbox).**

Add and Remove files. Making executables.

Object Oriented Programming

Object. An object is an encapsulation of data structures (the static properties of the modeled system) with program code that manages the structures (dynamic properties of the modeled system) (figure 4.1).

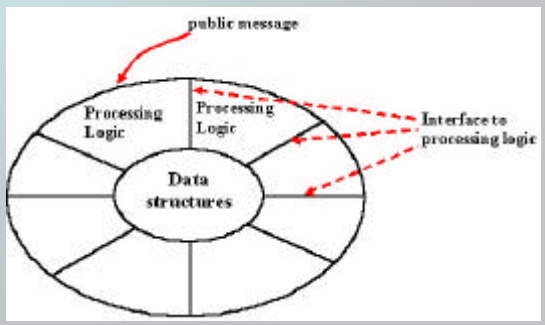


Figure 4. 1 Representation of an object

In programming environments the objects are identified by a unique name (we reference that name by *objectName*) assigned by the designer or by the system. The object name is used to qualify his components (attributes and methods) in constructions of the form:

objectName.**attributName**

or

objectName.**methodName.**

- **Messages.** All processing involves sending messages to objects. Messages are the language of interaction which you use to express yours computing requirements to objects. Messages request services from an object in terms of its external interface.

- **Methods** are the algorithms which are performed by an object in response to receiving a message. Methods represent the internal details of the implementation of an object.

An **object class** describes a set of object instance that have similar data characteristics, behavior, relationships to other objects and real-world meaning. Objects that are members of an object class share some attribute type and behavior type. Object instances that are members of the same class share a common real-world meaning in addition of their shared attributes and relationships (figure 4.2)

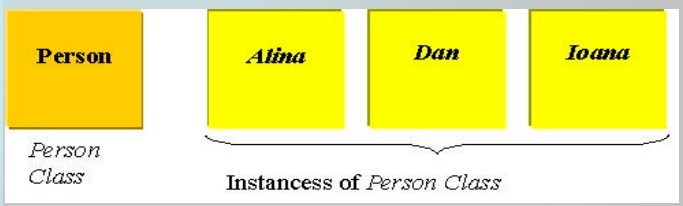


Figure 4. 2 Class and Objects

An **event** is an action recognized by an object (such as clicking the mouse or pressing a key). Events are things the object does; you can write code to be executed when event occur.

Event driven applications execute code in response to an event. In VB the objects automatically recognize a predefined set of events. A section of code - an event procedure - corresponds to each event. The types of events recognized by an object vary, but many types are common to most control. Events are triggered when some aspect of object is changed.

A typical sequence of events in an event_driven application is the following:

1°. the application starts and a form is loaded and displayed;

2°. the form (or a control on the form) receives an event. The event might be caused by the user (ex: pressing a button), by the system (ex: an execution that exceeds a time limit) or indirectly by the code of the application (for example a Load event to load and display a form).

3°. if there is code in corresponding procedure (for example in editing window a double-click can be nothing), it executes;

4°. the application waits for the next event.

Properties are data that describe an object. You can change an object's characteristics by changing its property. The properties are the attributes you set or retrieve. Some properties can be set at design time and others can be set only at runtime (you must write code to set them).

The properties can be:

- **read-write** properties – you can set or get them at runtime**;**
- **read-only** properties – you can only read them at runtime**.**

To set a value of a property, use the following syntax:

object.property=expression

To get the value of a property, use the following syntax:

variable=object.property

Examples of manipulating some properties of objects:

- Caption:
  object.**Caption[=***string***]**
- Visible:
  object.**Visible[=***boolean***]**
- Enabled:
  object.**Enabled[=***boolean***]**
- Auto Redraw:
  object.**AutoRedraw[=***boolean***]**
- Text boxes the text property:
  object.**Text[=***string***]**

Methods are part of objects just as properties are. They can affect the values of properties. Because the methods perform actions they can have arguments. When you use a method in code, how you write the statement depends on how many arguments the method requires, and whether the method returns a value.

The general syntax can be expressed as:

[*variable*=]*object*.**method**[(*argument_list*)]

When a method does not take arguments, you write the code using the following syntax:

object.**method**

Some Events, Methods and Properties on Forms and/or controls:

- To load/unload a control or form into memory:
  [**Load**/**Unload**] *object*
- To hide a form object:
  object.**Hide**
- To clear graphics and text generated at runtime from a form, image or Picture Box:
  object.**Cls**
- To refresh an object (a complete repaint)
  object.**Refresh**
- To move the focus to a specified object:
  object.**SetFocus**
- To display a form:
  object.**Show**[[*style*],[*ownerform*]]

Encapsulation provides a conceptual barrier around an object, preventing clients of an object from viewing its internal details. The encapsulation (figure 4.3) can provide appropriate barriers for various levels of abstraction in a system model.

Each object class must have an external interface defining the outside view of the class and an implementation that defines the mechanisms that provide the behaviors the object must be exhibit.

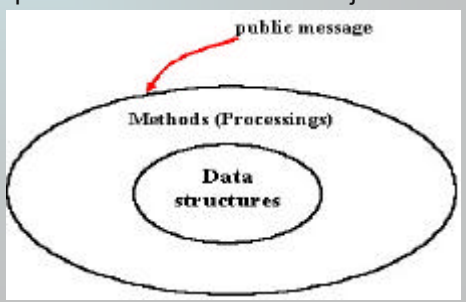


Figure 4. 3 Encapsulation

Hierarchy and Inheritance

Objects and their organization can provide the extra benefits of reusability of data and code. Programming procedures implemented in one object can be used in another object through a system of classes, hierarchies and inheritance. We need to combine data structures and processes to form a single unified set of structures and associated processing (figure 4.4)

Generalization: defines a relationship among classes

Aggregation: “part of” hierarchies

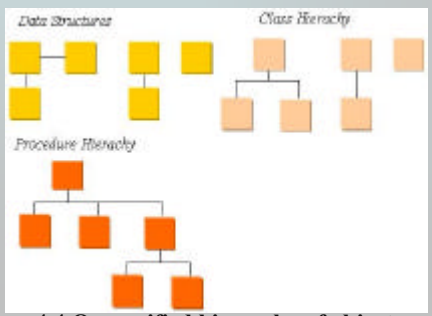


Figure 4.4 One unified hierarchy of objects

How VB implements the hierarchy ?

Suppose that our application contains two command buttons with distinct name Command1 and Command2. They share the same class Command Button and the characteristics of the form where they are located. All controls have common characteristics that make them different from forms and other objects in VB environment.

An object hierarchy provides the organization that determines how objects are related to each other, and how they can be accessed.

The objects can be grouped together in so called collection objects that have their own properties and methods. The objects in a collection object are referred to as *members* of the collection. In VB each member of a collection is numbered sequentially beginning at 0; this number is the member’s index number.

A member of a collection can be addressed by using two general techniques:

1st . By specifying the name of the member, for example:

CollectionName(“*MemberName*”)

or

CollectionName!*MemberName*

2nd . By using the index:

Controls(*index***)**

or

CollectionName(“*MemberIndex*”**)**

The common collections in VB are:

Forms - contains loaded forms;
Controls - contains controls on a form;
Printers - contains the available Printer Objects.

Association

An association defines a conceptual connection between object classes with common structure and semantics and provides a way to depict information that is not unique to a single class but that depends on two or more classes (figure 5.19)

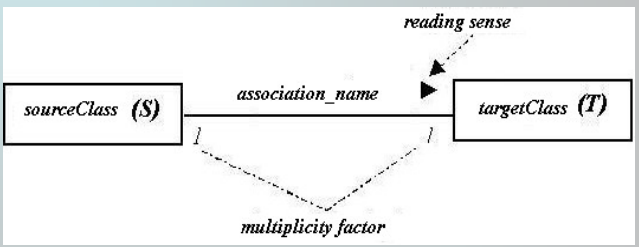


Figure 5.19. Associations between objects

Messages

When objects have been encapsulated to insulate the outside world from the details of the object structures and behaviors, there needs to be a way to interact with these structures and behaviors. Messages provide this mechanism.

A message is composed of the name of an operation to perform on object data and any necessary parameters to qualify the operation (figure 4.5).

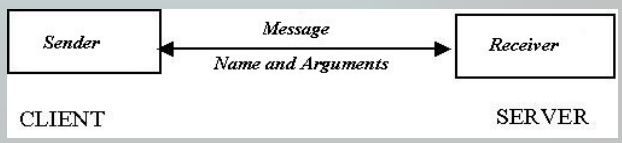
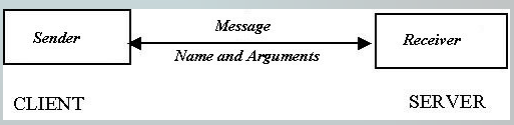


Figure 4. 5 Messages

When a client object (CLIENT) sends a message to another object (a SERVER), the client is asking the server to perform some operation and, perhaps, to return some information to the client. When a receiver of a message processes that message, it performs an operation in any way it can. The sender of message does not (indeed, should not) know how the operation will be performed. Because of encapsulation, the details of how an object performs an operation are hidden from view of outsiders.



Some messages might be internal ones that are not part of the objects public interface. An object could send a message to itself to perform recursive operations, for instance.

Polymorphism

Polymorphism is the ability of two or more classes to respond to the same message, but in different ways. Polymorphism means that many classes can provide the same property or method, and a caller doesn't have to know what class an object belongs to before calling the property or method. It allows the similarities between different object classes to be exploited (figure 4.6)

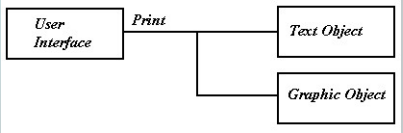


Figure 4.6 Polymorphism

Most object_oriented programming systems provide polymorphism through inheritance.

VB doesn't use inheritance to provide polymorphism; the polymorphism is provided by VB through multiple ActiveX interfaces. An interface is a set of related properties and methods.

VB Procedures

The methods and events are represented by various types of procedures. The procedures are small logical components in which you can break (split) a program for a specific task. They are very useful for condensing repeated or shared tasks (such as calculations frequently used). The procedures are called to do their job from other procedures. Generally a procedure can take arguments, perform a series of statements, and change the value of its arguments.

The general form of a VB procedure can be described as follows:

```
[Procedure_scope] Procedure_type Procedure_name [(Argument_list)][return_type]
        [declaration_statements]
        [executive_statements]      } procedure body
End Procedure_type
```

The *Procedure_scope* defines which parts of your code are aware of its existence:

Procedure_scope::={Private| Public}[Static]

Procedure_type::=Sub | Function

The *argument_list* declares the values that are passed in from a calling procedure.

A procedure can have two parts:

- a **declaration part** that tells the compiler which cells are needed to hold data and program results. The declaration part communicates to the compiler the names of all user_defined identifiers that can appear in the program and the usage of each identifier. It also tells the compiler what kind of information will be stored in each memory cell.

-an **executive part** that contains statement (derived from the algorithm you want to communicate to the computer) that are translated into machine language and later on executed.

There are several types of procedures used in VB:

1) **Sub** procedures do not return a value;

2) **Function** procedures return a value;

3) **Property** procedures can return and assign values, and set references to objects.

Sub procedures

The syntax of Sub procedure is:

[Private çPublic][Static] Sub *procedure_name*(*arguments*)
statements
End Sub

Each time the procedure is called the statements between Sub and End Sub are executed. Sub procedures are by default Public in all modules, that mean they can be called from anywhere in the application. A procedure can modify the values of any variables passed to it.

A Sub procedure can be a general procedure or an event procedure. A **general procedure** tells the application how to perform a specific task (ex. ButtonMgr)**.**
An **event procedure** establishes an association between the object and the code (they are said to be attached to forms and controls).

The naming conventions used are:

- an event procedure for a control combines the control's actual name (specified in the Name property), an underscore (_), and the event name;

- an event procedure for a form combines the word "Form", an underscore and the event name. If the forms are MDI the event name is prefixed with "MDIForm".

A call to a Sub procedure is a stand_alone statement (figure 4.7). The ways to call a Sub procedure can be realized in two ways:

Call *Procedure_name* (*argument_list*)

or

Call *Procedure_name argument_list*

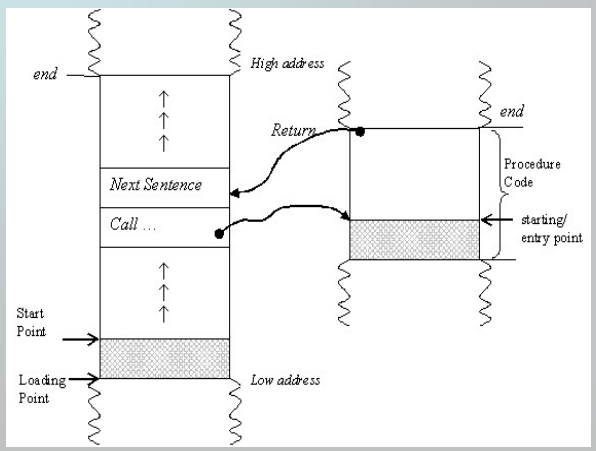


Figure 4.7 The call process

Public procedures in other modules can be called anywhere in the project. In order to reference the procedure in another module you must specify the module that contains it. The techniques for doing this vary, depending on whether the procedure is located in a form, class or standard module:

- Procedures in Forms. All calls from outside the form module must point the form module containing the procedure:

Call *Formname.Procedure_name*(*arguments*)

- Procedures in Class Modules. Calling a procedure in a class module requires that the call to the procedure be qualified with a variable that points to an instance of the class:

Dim *Variablename* **As New** *Classname*
Variablename.Procedure_name

- Procedures in Standard Modules. If the procedure name is unique you don't need to include the module name in the call. A call to a common procedure name from another module must specify the intended module:

Module_name.Procedure_name(*arguments*)

Function Procedures

VB include built_in functions (like Sqr, Sum, Min, …) that can be invoked anywhere in a project, as an assignment statement:

Variablename=Functionname(*arguments*)

or

Call *Functionname*(*arguments*)

or

Functionname arguments

The user can define his own function procedures by using the statement:

[Privateç**Public][Static]Function** *Functionname*(*arguments*) **[As** *data_type*]
statements (somewhere in the list of the statements must be an assignment:***Functionname=expression*** for the return)
End Function

A function procedure can return a value to the calling. There are three differences between Sub and Function procedures:

- generally, you call a function by including the function procedure name and arguments on the right side of a larger statement or expression (*Variablename*=*Functionname*());

- function procedures have data types, just as variables do. This determines the type of return value (As *type*).

- you return a value by assigning it to the procedure name itself.

Property procedures

Property procedures allow you to execute code when a property value is set or retrieved. In that way the property procedures allow an object to protect and validate its own data.

Control Categories

(course)

a) Intrinsic controls

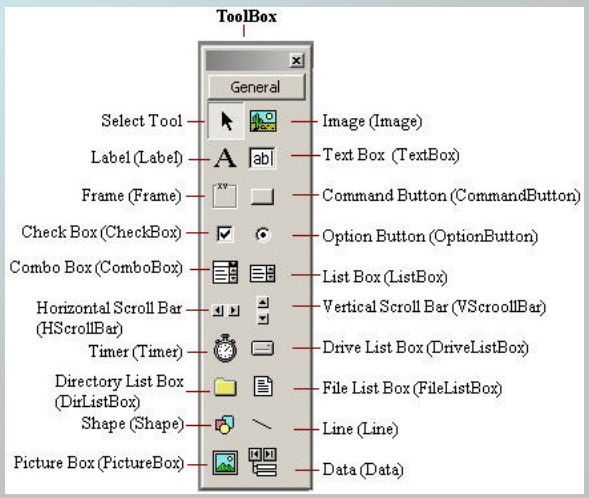


Figure 5.25. The standard toolbox in VB

Contents

Bibliography

1. [AvDg.03]Avram Vasile, Dodescu Gheorghe – *Informatics: Computer Hardware and Programming in Visual Basic*, Editura Economica, Bucuresti, 2003, chapter 5, pages 225-286

2. [AASA.02]V.Avram V, C.G.Apostol, T.Surcel, D.Avram – Birotica Profesionala, Editura Tribuna Economica, 2002, chapter 2 pages 47-92, chapter 3 pages 107-196

3. [SMAA] T.Surcel, R.Marsanu, V.Avram, D.Avram – Medii de programare pentru gestiunea bazelor de date, Editura Tribuna Economica, 2004, chapter 4, pages 213-326