Week 7: Graphical User Interface
An Object Oriented Approach

CS285 Usability Engineering
Interacting with the User

• The User is of most important concern when developing software for interactive use
  – Companies spend billions each year on beta-testing programmes
  – Most recent advances in software development have been to improve user-friendliness

• The User-Interface is the parts of a program the user interacts with
  – Forms, controls, menus
User Interface Requirements

- Identify actors and items from requirements descriptions
- Compile list of objects
- Attach responsibilities to each object (tasks that they must be able to perform)
- Demonstrate how objects with responsibilities interact in system (use-cases)
- Compile complete list of objects + responsibilities → system specification
Designing the User Interface

• The UI should be designed to...
  – Simplify the use of software by providing a simple control model
  – Clarify the output from software, using graphics, tabular formatting etc. where appropriate
  – Reflect the logical structure of the underlying software, regardless of the actual physical implementation

• The ideal User-Interface is invisible
  – The user will only notice it when it is difficult
Using Object-Oriented Design

System is developed as an Object Model
Forms are Viewers and Controllers of Object Model
Forms operate on *domains* in model
O-O System Development

1. Create Model Specification
2. Create System Design
   - Determine Classes to develop, Attributes & Behaviours
   - Determine Object Instances required
3. Detailed design of Class Interfaces
4. Design Viewer/Controllers for Objects
5. Implement Object Model
   - Write Code
   - Test Object Model
6. Implement Viewer/Controllers (Forms)
7. Integrate components to create system
User-Interface of Object Model

• In general, it is good practice to develop an Object Model in code, and then attach a user-interface to it
  – Allows for changes in Model
  – Allows for changes in UI
  – Creates a more easily testable system

• Easiest route is to develop object model and forms as separate classes
  – Form design can include new properties and methods
    • A Form is a controller
    • Information collected on Forms can be passed to Objects
Object Interactions

- **Internal structure of an object**: (this can be modified while the interface remains unchanged)

- **Object interface**
  - Property Definitions
  - Methods Definitions

- **Object User** (a program) (only ever sees the interface)
Forms, Views and Controllers

• A Form is a Window (also an object)
  – Should act as ‘controller’ or ‘view’ for a part of the object model (OM)
    • Controller sends command **messages** to OM
    • View is updated by OM
    • An interactive form can be both View and Controller

• MVC (Model-View-Controller) is an established method for organizing UI
Standard MVC Design

• A single Object Model
• Possible multiple views
  – e.g. Digital/Analogue clock
• Possible multiple controllers
  – e.g. Front panel + remote
Modified MVC

• In GUI software, often the View and Controller are a single unit
  – e.g. List Box with scroll bar
• This simplified model is closer to that type of system
Component User-Interfaces

• Sometimes, UI components are incorporated into the object model
• Only sensible if UI component is specific to a particular object
  – Must merit the additional development time
  – Only worthwhile if reuse is certain
• VB Provides for the development of Controls, components that can be added to the toolbox
User Interface Tools

• Most modern development systems (including VB) include facilities for developing a UI
  – ‘Visual’ model is common (not universal)
  – Ideal is to develop both software and UI
    • as independently as possible
    • to minimize unnecessary interactions between them

• Stress should be placed on
  – Developing object model
  – Developing controls to connect to this model as simply as possible
Event-Driven User Interfaces

- Use of Forms containing input and command controls
  - Input - Text Boxes, Check Boxes, Lists etc.
  - Command - buttons, menus
- A form should allow user-input in any order
  - Processing triggered when a command is issued
  - Ideally, commands are unavailable until all required input information is provided
- Users should be led through the appropriate actions
Form-Based U/I Elements

Text Boxes - note labels to make purpose clear

Options - note use of frame to group options

OK Button, defines a command as a positive action

Cancel Button, allows user to back-off

Form caption shows overall purpose

Check box - should initially show default choice

Form caption shows overall purpose

Check box - should initially show default choice

Form caption shows overall purpose

Check box - should initially show default choice

Form caption shows overall purpose

Check box - should initially show default choice
Standard Control Use

• Text Boxes
  – Free-form text input (single or multi-line)
  – Numeric input (use Changed event to filter out non-numeric characters)

• Check Boxes
  – True/False, Yes/No etc input.

• Option Buttons
  – Arranged in groups of possible options
Standard Control Use - 2

• List and Combo boxes
  – List - selection from a (possibly large) set - all on view at all times
  – Combo - as above but only selected option on view

• Scroll Bars
  – Graphical interaction - shows analogue of input value - sets a numeric
U/I Guidelines

• Simple is best
  – Standard windows controls are well known
  – Windows colour scheme is normally set so that all controls are optimally visible

• Where possible, let user Select from a number of possible options
  – List Boxes, Combo Boxes, Option Groups etc.
U/I Guidelines - 2

• Where possible, provide a default action
  – Option, List selection, input text value etc.
  – chosen/set if user does nothing

• Guide the user by making controls available as necessary
  – Use Enabled property to disable controls if…
    • Other controls must be set first
    • Current settings make their use inappropriate
Typical Use of Enabled Property

Sub txtName_Change() ‘ Do same for txtAddress
    CheckEnabled
End Sub

Sub CheckEnabled()
    If txtName.Text<>”” And txtAddress.Text<>””
    Then
        cmdOK.Enabled = True
    Else
        cmdOK.Enabled = False
    End If
End Sub

Note: Cancel button is always valid → can invoke cancel at any time

OK Button
disabled until its use is valid
Event Driven Programming

• Procedural language to support a user interface
• Processes are initiated by an event – something that happens, e.g., user clicks on a button
• Your role - anticipate the events and to write the code that will be executed when the event occurs
• Large code split into smaller blocks - subroutines that are associated with events
• A VB application is *interactive* in the sense that the user is constantly interacting with the program
Event Handlers

• A special form of Sub that Visual Basic associates with an event
  – Mouse operation
  – Key-press
  – Signal from another application

• An event handler is called automatically when the event happens
  – Program can respond to external stimuli

```vbnet
Private Sub Button1_Click()
    MsgBox “Hello World”
End Sub
```
Visual Basic Structure

- Forms: containing interface objects
- Form Objects: contain VB Code in event driven subroutines (Sub)
- General: area for each Form to hold declarations for the Form and definitions of Procedures:
  - Sub Procedures
  - Function Procedures
- Modules: contains Global Visual Basic Code in the form of Procedures
- Class Modules (Objects): provide a Object Oriented programming environment
Visual Basic Lab

• Integrated Development Environment - IDE provides for all the facilities to build, test and execute a program
• Exploration of the IDE → Lab 1 exercise
• Supplementary Lab 1 exercise deals with forms and controls

Recommended Texts:

Visual Basic - an Object Oriented Approach  by A McMonnies
An Introduction to Visual Basic  by D Schneider
Programming Constructs

• Data stored in variables that must be declared
• Assignment statement =
• Operators:
  – simple arithmetic (+, -, *, /...)
  – string (text) manipulation
  – date manipulation
• Event Handlers
Declarations

• Variables need to be declared
  – with the name and the type of data
  – Integer types (Byte, Integer, Long) for whole numbers
  – Floating point types (Single, Double) for numbers with fractional parts
  – Dates and times (one type, Date, for both)
  – Fixed point types (Decimal, Currency) for high precision (many digits)
  – String type for text
Variable Scope

• Where a variable can be accessed depends on how it is declared
  – Local scope (Dim) – only within Procedure (Sub or Function) in which it is declared
  – Module Scope (Private) – available in every Procedure within a module
  – Global Scope (Public) – available throughout application

• Controlling scope makes it possible to work within contexts, so that variable names can be reused with no conflicts
Declarations

<table>
<thead>
<tr>
<th>Dim  index As integer</th>
<th>Can be placed anywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private name As String</td>
<td>Declarations in General section</td>
</tr>
<tr>
<td></td>
<td>Only accessible from same module</td>
</tr>
<tr>
<td>Public today As Date</td>
<td>Declarations in General section</td>
</tr>
<tr>
<td></td>
<td>Accessible anywhere</td>
</tr>
</tbody>
</table>
Local & Static Variables

• Local variables (within Sub or Function)
  – Declared with **Dim** are reset to zero every time Procedure is called
  – Declared with **Static**, retain their value

```vba
Sub Forgettable()
    Dim Number As Integer
    Number = Number + 1
End Sub ' Always 1 at End Sub

Sub Persistent()
    Dim CallCount As Integer
    CallCount = CallCount + 1
End Sub ' Counts up
```
Variant Type

• A Variant is an all purpose variable
  – Type is always Variant
  – Sub-Type matches whatever value is assigned
  – Sub-Type changes to suit

Private Sub SquareRoot_Click()
Dim V As Variant
  V = 2           Assign an integer
  V = Sqr(V)      Assign a double
  V = “Root 2 = “ & V Assign a string
End Sub
User Defined Types

• A UDT is composed of simple types (Integers, strings etc.)
• Each instance contains one of each component variable
• Must be placed in Standard Modules
• Definition is Global (available throughout program)
• Useful for grouping related information
An example UDT

Type PhoneEntry
    Name As String
    Telephone As String
End Type

Dim PE As PhoneEntry
    PE.Name = "Fred Bloggs"
    PE.Telephone = "2468"
Print PE.Name, PE.Telephone
Module Structure

• Every module has
  – General Declarations section
  – Subroutines (Subs) and Functions

• Form modules also have
  – User interface elements (which appear on-screen)
  – Event handlers

• Class modules also have
  – Event receptors (objects that can respond to events)
Subs & Functions

• All executable statements must be part of a Sub or Function in VB
  – Sub – an operation, delineated by Sub and End Sub, containing a sequence of statements
    • a Sub ‘call’ is a statement that invokes the Sub (executes the operation)
  – Function – similar to a Sub, but returning a value as a result
    • a function call is an expression – can be used as a value in another expression or statement
  – Both Subs and Functions can have parameters or arguments – variables used to get information in and/or out of the Sub/Function
Subs

Sub Greeting (Name As String)
    MsgBox "Hello " & Name
End Sub

Greeting ("John")
Greeting "John"
Call Greeting ("John")
Functions

Function Add (X As Integer, Y As Integer)
    Add = x + Y
End Add

MsgBox ( Add(2,3))