

MÔN : LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Hướng dẫn Đồ Án : Xây dựng trình quản lý hệ thống file mini

I. Mục tiêu :

- Giúp SV thực hành để hiểu biết hầu hết các tính chất và khả năng lập trình hướng đối tượng của VC# để xây dựng ứng dụng thực tế.

II. Nội dung :

- Thiết kế trực quan các cửa sổ ứng dụng FileManager theo đặc tả chi tiết dưới đây.
- Viết code cho các hàm xử lý sự kiện thực hiện các chức năng quản lý hệ thống file của chương trình

III. Chuẩn đầu ra :

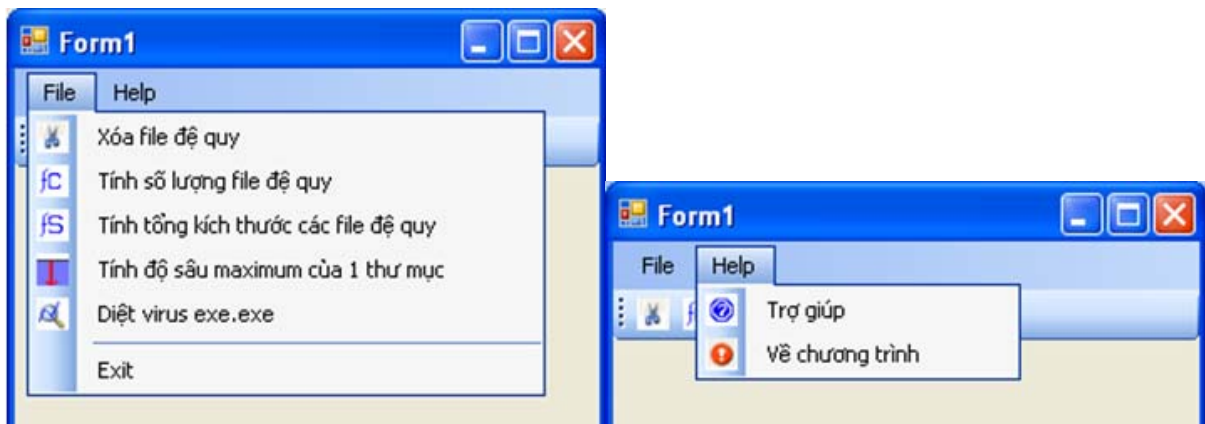
- Sinh viên nắm vững và dùng thành thạo qui trình kỹ thuật để thiết kế trực quan các cửa sổ giao diện của chương trình, thiết lập giá trị các thuộc tính cho từng phần tử giao diện, khai báo hàm xử lý sự kiện cho sự kiện quan tâm của đối tượng giao diện.
- Sinh viên nắm vững và sử dụng thành thạo các tính chất lập trình hướng đối tượng như tính thừa kế, bao đóng, đa xạ để xây dựng các đoạn code tổng quát hóa.

IV. Đặc tả chương trình FileManager :

- Chương trình phải cung cấp được 5 chức năng quản lý hệ thống file sau đây :
 1. duyệt tìm và xóa các file thỏa mãn pattern qui định từ 1 thư mục bắt đầu do người dùng qui định.
 2. duyệt tìm và xóa các file *.exe do virus exe.exe tạo ra từ 1 thư mục bắt đầu do người dùng qui định.
 3. duyệt tìm phần tử có độ sâu sâu nhất trong 1 thư mục do người dùng qui định.
 4. duyệt tìm và tính số lượng các file và các thư mục thỏa mãn pattern qui định từ 1 thư mục bắt đầu do người dùng qui định.
 5. duyệt tìm và tính tổng kích thước các file thỏa mãn pattern qui định từ 1 thư mục bắt đầu do người dùng qui định.

V. Phân tích :

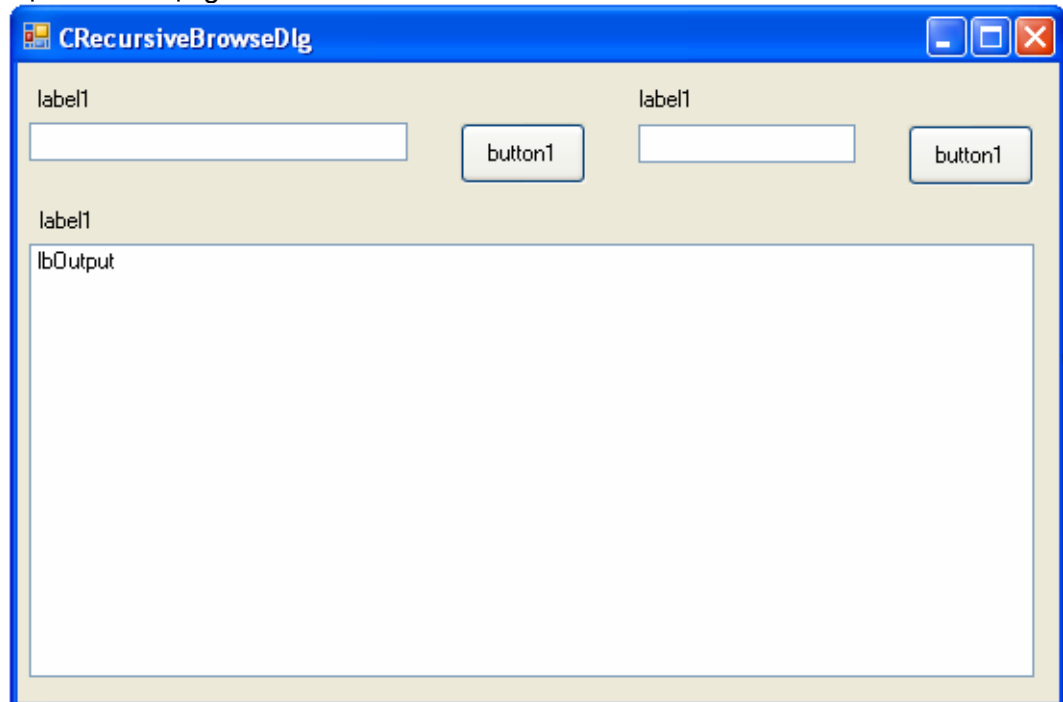
- Để giúp người dùng thực hiện 5 chức năng trên, chương trình nên có menubar như sau :



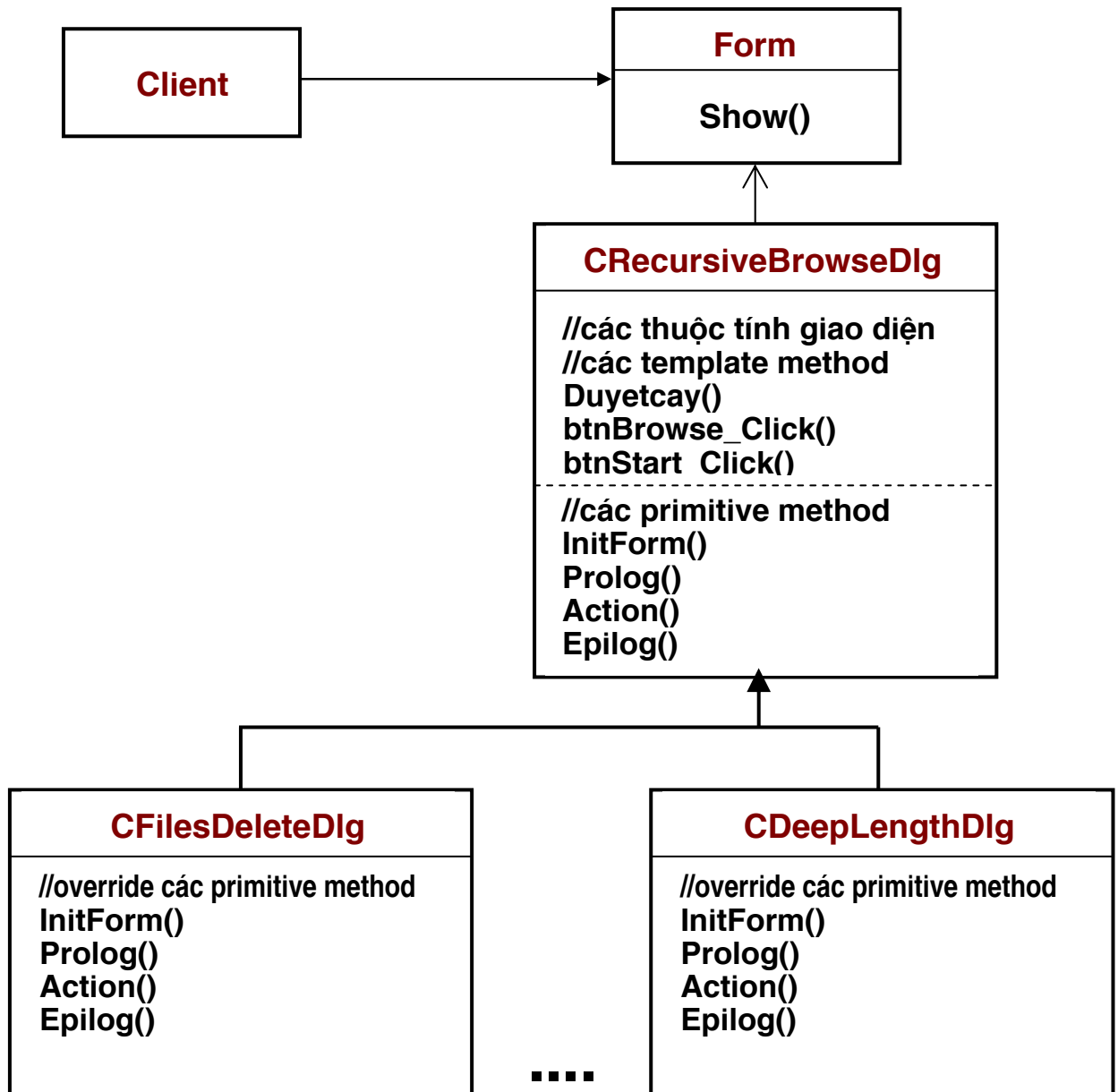
- Để giúp người dùng thực hiện 5 chức năng trên dễ dàng và nhanh chóng hơn, chương trình nên có toolbar như sau (mỗi icon trong toolbar sẽ giúp thực hiện nhanh 1 chức năng tương ứng của chương trình) :



- Phân tích 5 chức năng cần thực hiện của chương trình, ta thấy qui trình thực hiện các chức năng này đều có những công việc giống nhau như sau :
 - ✓ cần 1 form giao diện để người dùng xác định thư mục bắt đầu xử lý, chuỗi pattern nhận dạng các phần tử cần xử lý, hiển thị các thông tin xử lý theo thời gian... Thí dụ form có dạng sau :



- ✓ cần 1 đoạn code thực hiện thuật giải duyệt cây phân cấp từ thư mục xác định bởi người dùng để tìm tất cả phần tử thỏa mãn pattern qui định để xử lý. Lưu ý mỗi chức năng qui định việc xử lý phần tử tìm được hoàn toàn khác nhau : chức năng xóa file thì sẽ xóa file, chức năng đếm số lượng thì sẽ tăng count đếm, ...
- Sau khi phân tích các chức năng của chương trình và nắm vững kiến thức về thiết kế phần mềm hướng đối tượng, ta thấy để giải quyết tốt nhất các chức năng của chương trình là dùng mẫu thiết kế phổ dụng có tên là “Template method” với lược đồ class như sau :



- ✓ thiết kế trực quan 1 lần để tạo giao diện cho form giao diện tổng quát để người dùng xác định thư mục bắt đầu xử lý, chuỗi pattern nhận dạng các phần tử cần xử lý, hiển thị các thông tin xử lý theo thời gian... Đặt tên class cho form này là CRecursiveBrowseDlg, các thuộc tính dùng chung, hàm xử lý button Browse, button Start được viết 1 lần ở class CRecursiveBrowseDlg, ta gọi các hàm này là các template function, thí dụ hàm DuyetCay() sẽ miêu tả thuật giải duyệt cây phân cấp được dùng chung cho mọi chức năng xử lý hệ thống file.
- ✓ Để thực hiện từng chức năng, ta định nghĩa 1 class con của CrecursiveBrowseDlg rồi chỉ cần override các hàm primitive như InitForm, Prolog, Action, Epilog.

VI. Qui trình xây dựng chương trình


VI.1 Qui trình điển hình để tạo các icon đồ họa trong toolbar thể hiện các chức năng :

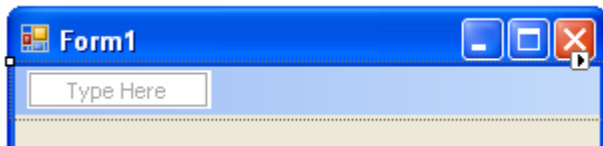
Toolbar là 1 cửa sổ chứa nhiều button (icon), mỗi button cho phép thực hiện 1 chức năng của ứng dụng. Các button có kích thước đều nhau, nên kết hợp 1 ảnh bitmap với từng button, nội dung ảnh làm sao gợi ý cho người dùng về chức năng tương ứng (thí dụ ảnh dạng cái kéo gợi ý chức năng Cut,...).

1. Công việc đầu tiên cần thực hiện là dùng 1 trình soạn thảo đồ họa (Paint, CorelDraw,...) để thiết kế (vẽ) từng ảnh bitmap gợi ý cho chức năng của từng button trong Toolbar. Bạn có thể vẽ mới hình bitmap hay dùng trình "Screen Capture" cắt các icon có sẵn của ứng dụng nào đang chạy và dán vào vùng soạn thảo ảnh của trình soạn thảo đồ họa. Sau khi soạn xong 1 ảnh, ta cất ảnh lên file dạng *.bmp. Lưu ý rằng các ảnh phải có cùng kích thước (thí dụ ta chọn 20*20) :

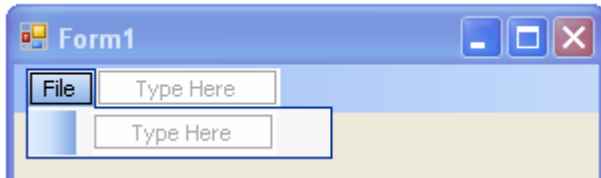
- file FilesDelete.bmp chứa ảnh của button  (xóa file)
- file FilesCount.bmp chứa ảnh của button  (đếm số lượng)
- file FilesSize.bmp chứa ảnh của button  (tính tổng kích thước)
- file DeepLen.bmp chứa ảnh của button  (tính độ sâu max)
- file ExeVirusDel.bmp chứa ảnh của button  (tìm và diệt virus exe.exe)
- file Help.bmp chứa ảnh của button  (trợ giúp của chương trình)
- file About.bmp chứa ảnh của button  (giới thiệu thông tin về chương trình)

VI.2 Qui trình điển hình để xây dựng MenuBar cho cửa sổ chương trình :

2. Chạy VS .Net, chọn menu File.New.Project để hiển thị cửa sổ New Project.
3. Mở rộng mục Visual C# trong TreeView "Project Types", chọn mục Window, chọn icon "Windows Application" trong listbox "Templates" bên phải, thiết lập thư mục chứa Project trong listbox "Location", nhập tên Project vào textbox "Name:" (td. FileManager), click button OK để tạo Project theo các thông số đã khai báo.
4. Form đầu tiên của ứng dụng đã hiển thị trong cửa sổ thiết kế, việc thiết kế form là quá trình lặp 4 thao tác tạo mới/xóa/hiệu chỉnh thuộc tính/tạo hàm xử lý sự kiện cho từng đối tượng cần dùng trong form.
5. Nếu cửa sổ ToolBox chưa hiển thị chi tiết, chọn menu View.Toolbox để hiển thị nó (thường nằm ở bên trái màn hình). Click chuột vào button  (Auto Hide) nằm ở góc trên phải của cửa sổ ToolBox để chuyển nó về chế độ hiển thị thường trực.
6. Duyệt tìm phần tử MenuStrip (trong nhóm Menu & Toolbars), chọn nó, drag nó về vị trí bất kỳ trong form để tạo menubar cho cửa sổ chương trình. Menubar lập tức được tạo ra ở trên cửa sổ. Menubar mới chỉ có 1 menu trống có caption là "Type Here".



7. Click chuột vào chuỗi "Type Here" để thiết lập cursor ở đây rồi nhập vào caption của menu đầu tiên của chương trình "File". Sau khi nhập xong caption cho menu, click chuột để chọn nó, cửa sổ thuộc tính của menu này được hiển thị trong cửa sổ thuộc tính (thường nằm ở dưới phải màn hình). Duyệt tìm và hiệu chỉnh nội dung của thuộc tính (Name) = mnuFile.
8. Chọn lại menu File, hiện giờ nó chỉ chứa 1 MenuItem trống có caption là "Type Here" :



Click chuột vào chuỗi “Type Here” trong menu pop-up “File” để thiết lập cursor ở đây rồi nhập vào caption của MenuItem đầu tiên là “Xóa file đệ quy”. Sau khi nhập xong caption cho MenuItem, click chuột trên nó để chọn nó, máy sẽ hiển thị cửa sổ thuộc tính của nó. Ấn phải chuột trên MenuItem để hiển thị menu lệnh, chọn chức năng “Set Image” để hiển thị cửa sổ “Set Resource”, đánh dấu chọn vào checkbox “Local resource”, click button Import, duyệt tìm và xác định file bitmap được dùng làm icon cho MenuItem này. Xem cửa sổ thuộc tính của MenuItem “Xóa file đệ quy” vừa tạo, duyệt tìm và hiệu chỉnh nội dung của thuộc tính (Name) = mnuFilesDelete.

9. Lặp lại bước 8 để tạo các MenuItem chức năng còn lại :

- “Tính số lượng file đệ quy” có (Name) = mnuFilesCount
- “Tính tổng kích thước các file đệ quy” có (Name) = mnuFilesSize
- “Tính độ sâu maximum của 1 thư mục” có (Name) = mnuFileDeepLen
- “Diệt virus exe.exe” có (Name) = mnuFileVirusDel
- “-“ có (Name) = tên mặc định. Phần từ này tự biến thành làn phân cách (để tạo nhóm chức năng trong 1 menu pop-up).
- “Exit” có (Name) = mnuFileExit

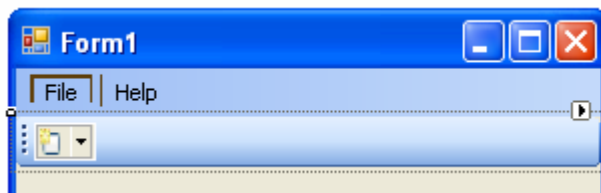
10. Lặp lại các bước 8 & 9 để tạo menu Help với (Name) = mnuHelp gồm 2 MenuItem chức năng sau đây :



- “Trợ giúp” có (Name) = mnuHelpHelp
- “Về chương trình...” có (Name) = mnuHelpAbout

Sau khi thiết kế xong, cửa sổ có dạng y như hình vẽ ở mục V trên đây.







VI.3 Qui trình diễn hình để xây dựng ToolBar :

11. Duyệt tìm phần tử ToolStrip (trong nhóm Menu & Toolbars), chọn nó, drag nó về vị trí bất kỳ trong form để tạo Toolbar cho cửa sổ chương trình. Toolbar lập tức được tạo ra ở trên cửa sổ. Toolbar mới chỉ có 1 Button trống như hình dưới :



12. Click chuột vào mũi tên chỉ xuống của Button trống để hiển thị menu lệnh, chọn lệnh Button để tạo Button mới. Button mới có hình đồ họa mặc định là . Trong cửa sổ thuộc tính của button mới, duyệt tìm thuộc tính Image, click chuột vào button  bên phải thuộc tính để hiển thị cửa sổ “Select Resource”, đánh dấu chọn vào checkbox “Local resource”, click chuột vào button “Import”, duyệt tìm và xác định file bitmap được dùng làm icon cho Button này (). Xem cửa sổ thuộc tính của Button vừa tạo, duyệt tìm và hiệu chỉnh thuộc tính (Name) = tbFileDelete.

13. Lặp lại bước 12 nhiều lần để tạo các button còn lại :

- Button  có (Name) = tbFilesCount
- Button  có (Name) = tbFilesSize
- Button  có (Name) = tbFileDeepLen
- Button  có (Name) = tbFileVirusDel
- Button  có (Name) = tbHelpHelp
- Button  có (Name) = tbHelpAbout

Sau khi thiết kế xong Toolbar, ta thấy Toolbar có dạng sau :



VI.4 Qui trình diễn hình để định nghĩa interface sử dụng :

Trong chương trình, ta có 5 class miêu tả 5 chức năng mà chương trình cung cấp. Chi tiết hiện thực từng class chức năng cần được che dấu đối với phần còn lại của chương trình. Để giải quyết vấn đề này tốt nhất, ta sẽ định nghĩa interface sử dụng chung cho 5 class chức năng, interface này có tên là Icommand và chỉ chứa đúng 1 hàm dịch vụ :

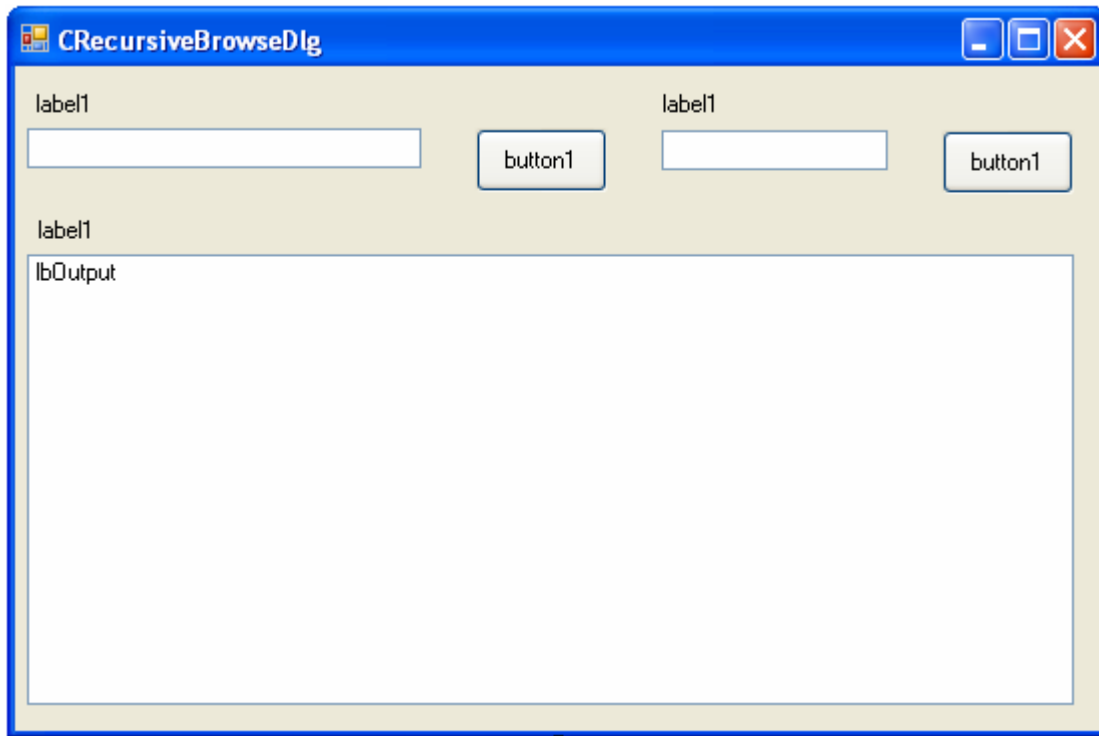
`void Show(); //hiển thị form chức năng để người dùng làm việc`

14. Để định nghĩa interface, ta dời chuột về phần tử gốc của cây Project trong cửa sổ "Solution Explorer", ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.New Item để hiển thị cửa sổ "Add New Item", chọn mục "Interface", hiệu chỉnh tên interface là ICommand.cs, chọn button Add để máy tạo 1 interface mới.
15. Cửa sổ soạn code cho interface ICommand được hiển thị, ta định nghĩa interface đơn giản như sau :


```
namespace FileManager {
    interface ICommand {
        void Show(); //hiển thị form chức năng để người dùng làm việc với nó
    }
}
```

VI.5 Qui trình diễn hình để xây dựng trực quan 1 Dialog Box :

Giả sử ta cần 1 form tổng quát chứa các đối tượng giao diện sau đây để phục vụ chung cho tất cả các chức năng của chương trình FileManager :



Qui trình điển hình để xây dựng trực quan Form (Dialog Box) trên gồm các bước thao tác sau :


16. Dời chuột về phần tử gốc của cây Project trong cửa sổ “Solution Explorer”, ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.Windows Form để hiển thị cửa sổ “Add New Item”, chọn mục “Windows Form”, hiệu chỉnh tên Form là CRecursiveBrowseDlg.cs, chọn button Add để máy tạo 1 form mới.
17. Form mới tạo đã hiển thị trong cửa sổ thiết kế, việc thiết kế form là quá trình lặp 4 thao tác tạo mới/xóa/hiệu chỉnh thuộc tính/tạo hàm xử lý sự kiện cho từng đối tượng cần dùng trong form.
18. Thay đổi phông chữ kích thước form cho đủ lớn hầu chứa đủ các đối tượng giao diện trong form như hình vẽ trên.
19. Nếu cửa sổ Toolbox chưa hiển thị chi tiết, chọn menu View.Toolbox để hiển thị nó (thường nằm ở bên trái màn hình). Click chuột vào button  (Auto Hide) nằm ở góc trên phải cửa sổ Toolbox để chuyển nó về chế độ hiển thị thường trực.
20. Duyệt tìm phần tử Label (trong nhóm Common Controls), chọn nó, dời chuột về vị trí trên trái trong form và vẽ nó với kích thước mong muốn. Vì đây là phần tử tổng quát để các class con override nên ta không cần thiết lập nội dung cụ thể cho thuộc tính Text của nó, tuy nhiên để class con truy xuất dễ dàng, ta cần hiệu chỉnh thuộc tính (Name) = lblStartDir, Modifiers = protected.
21. Duyệt tìm phần tử TextBox (trong nhóm Common Controls), chọn nó, dời chuột về vị trí ngay dưới Label vừa vẽ và vẽ nó với kích thước mong muốn. Hiệu chỉnh thuộc tính (Name) = txtStartDir, Modifiers = protected.
22. Duyệt tìm phần tử Button (trong nhóm Common Controls), chọn nó, dời chuột về vị trí ngay bên phải TextBox vừa vẽ và vẽ nó với kích thước mong muốn. Hiệu chỉnh thuộc tính (Name) = btnBrowse, Modifiers = protected.
23. Lặp lại 3 bước 20, 21, 22 để vẽ 1 Label có (Name) = lblPattern, Modifiers = protected, 1 TextBox có (Name) = txtPattern, Modifiers = protected, 1 button có (Name) = btnStart,

Modifiers = protected. Bạn cũng có thể dùng phương pháp nhân bản vô tính để tạo các phần tử mới giống như những phần tử đã có, thí dụ bạn chọn 3 đối tượng đã vẽ (Label, TextBox, Button), copy chúng rồi paste vào vị trí mới, dời vị trí và thay đổi kích thước các phần tử mới theo yêu cầu.

24. Duyệt tìm phần tử Label (trong nhóm Common Controls), chọn nó, dời chuột về vị trí ngay dưới TextBox txtStartDir và vẽ nó với kích thước mong muốn. Hiệu chỉnh thuộc tính (Name) = lblOutput, Modifiers = protected.
25. Duyệt tìm phần tử ListBox (trong nhóm Common Controls), chọn nó, dời chuột về vị trí ngay dưới Label vừa vẽ và vẽ nó với kích thước mong muốn. Hiệu chỉnh thuộc tính (Name) = lbOutput, Modifiers = protected.

Sau khi thiết kế form xong, Form có dạng đúng theo yêu cầu ở trên.

VI.6 Định nghĩa các hàm xử lý sự kiện cần thiết trên các đối tượng giao diện :

26. Dời chuột về button btnBrowse, ấn kép chuột vào nó để tạo hàm xử lý sự kiện Click chuột cho button, cửa sổ mã nguồn sẽ hiển thị để ta bắt đầu viết code cho hàm. Cách tổng quát để tạo hàm xử lý sự kiện là chọn đối tượng btnBrowse, cửa sổ thuộc tính của nó sẽ hiển thị, click icon  để hiển thị danh sách các sự kiện của đối tượng, duyệt tìm sự kiện quan tâm (Click), ấn kép chuột vào comboBox bên phải sự kiện Click để máy tạo tự động hàm xử lý cho sự kiện này. Cửa sổ mã nguồn sẽ hiển thị khung sườn của hàm vừa được tạo với thân rỗng, nhiệm vụ của người lập trình là viết code miêu tả thuật giải thực hiện đúng chức năng mong muốn :

```
//hàm xử lý sự kiện Click trên button btnBrowse
private void btnBrowse_Click(object sender, EventArgs e) {
    //tạo form duyệt chọn thư mục
    FolderBrowserDialog dlg = new FolderBrowserDialog();
    //hiển thị form duyệt chọn thư mục để người dùng duyệt chọn thư mục làm việc
    dlg.ShowDialog();
    //hiển thị đường dẫn thư mục vào textbox txtStartDir
    txtStartDir.Text = dlg.SelectedPath;
}
```

27. Lặp lại bước 24 để tạo hàm xử lý sự kiện Click chuột trên button btnStart, khung sườn của hàm vừa được tạo với thân rỗng, viết code cho hàm này như sau :

```
//hàm xử lý sự kiện Click trên button btnStart
//hàm này là 1 template method điển hình,
//nó chỉ chứa 3 bước trừu tượng để giải quyết bất kỳ chức năng nào.
private void btnStart_Click(object sender, EventArgs e) {
    //1. thiết lập các giá trị đầu để thực hiện chức năng
    Prolog();
    //2. duyệt cây và thực hiện chức năng trên từng phần tử tìm được
    DuyệtCay(txtStartDir.Text, txtPattern.Text);
    //3. thực hiện các công việc kết thúc chức năng
    Epilog();
}
```

28. Hiện thực tối thiểu các hàm primitive (có thân rỗng) để máy không báo lỗi (vì các hàm này sẽ được dùng trong các hàm template của class hiện hành). Ý tưởng là để cho các class con override các hàm này theo yêu cầu riêng :

```
//thiết lập các chuỗi caption cho các đối tượng giao diện
virtual public void InitForm() { }
```



```
//thiết lập các giá trị đầu để thực hiện chức năng
virtual public void Prolog() { }
//thực hiện chức năng trên phần tử tìm được
virtual public void Action(String fname, byte fop) { }
//thực hiện các công việc kết thúc chức năng
virtual public void Epilog() { }
```

29. Hiện thực hàm template quan trọng nhất của class hiện hành, hàm này sẽ duyệt cây phân cấp từ thư mục xác định để tìm tất cả các phần tử con thỏa mãn tiêu chuẩn được xác định trong chuỗi pattern :

```
//duyet cây và thực hiện chức năng trên từng phần tử tìm được
public void DuyetCay(String sdir, String spattern) {
    // tìm các file thỏa mãn pattern và xử lý
    string[] flist = Directory.GetFiles(sdir, spattern);
    foreach (string fname in flist)
        Action(fname,0);

    //xác định tất cả thư mục con
    string[] sdlistw = Directory.GetDirectories(sdir);
    //xác định tất cả thư mục con thỏa pattern
    string[] sdlist = Directory.GetDirectories(sdir,spattern);
    //duyet xử lý từng thư mục con
    foreach (string subdir in sdlistw)
        if (thuocve(subdir,sdlist)) { //thư mục thỏa pattern
            DuyetCay(subdir, "");
            Action(subdir,1);
        }
        else //thư mục không thỏa pattern
            DuyetCay(subdir, spattern);
    }
    //kiểm tra chuỗi s có nằm trong danh sách các chuỗi sl
    private bool thuocve(String s, String[] sl) {
        int max = sl.Length-1;
        for (int i = 0; i <= max; i++)
            if (s == sl[i]) return true;
        return false; //trả về false nếu s không nằm trong danh sách
    }
}
```

30. Vì các đoạn code trên có dùng 1 số class thư viện trong namespace System.IO, nên ta phải khai báo namespace này bằng cách dời về đầu file mã nguồn của class CRecursiveBrowseDlg rồi thêm lệnh sau đây vào :

```
using System.IO;
```

31. Ta hiệu chỉnh lại lệnh định nghĩa class CRecursiveBrowseDlg như sau để nó hiện thực interface ICommand :

```
//class CRecursiveBrowseDlg thừa kế class Form và hiện thực interface ICommand
public partial class CRecursiveBrowseDlg : Form, ICommand
```

- 31b. Ta hiệu chỉnh lại thân của hàm khởi tạo class CRecursiveBrowseDlg như sau :

```
//hàm khởi tạo class CRecursiveBrowseDlg
public CRecursiveBrowseDlg() {
    InitializeComponent();
}
```

```

    InitForm();
}

```

VI.7 Định nghĩa class con phục vụ xóa file đệ quy :

32. Dời chuột về phần tử gốc của cây Project trong cửa sổ “Solution Explorer”, ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.Class để hiển thị cửa sổ “Add New Item”, chọn mục “Class”, hiệu chỉnh tên class là CFilesDeleteDlg.cs, chọn button Add để máy tạo 1 class mới.

33. Khi cửa sổ soạn code cho class CFilesDeleteDlg hiển thị, hiệu chỉnh nội dung của class như sau :

```

using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace FileManager {
    class CFilesDeleteDlg : CRecursiveBrowseDlg {

        //override hàm InitForm
        public override void InitForm() {
            //hiệu chỉnh các chuỗi caption cho các phần tử giao diện
            this.Text = "Chức năng xóa file và thư mục đệ quy";
            this.lblStartDir.Text = "Thư mục bắt đầu xóa :";
            this.lblPattern.Text = "Nhập pattern :";
            this.lblOutput.Text = "Các file & thư mục bị xóa :";
            this.btnBrowse.Text = "Browse";
            this.btnStart.Text = "Start";
        }

        //override hàm Prolog
        public override void Prolog() { }

        //override hàm Action để thực hiện xóa thư mục hay file tùy trường hợp
        public override void Action(String fname, byte fop) {
            if (fop == 1) { //thư mục
                Directory.Delete(fname);
                lbOutput.Items.Add("Remove " + fname);
            } else { //file
                File.Delete(fname);
                lbOutput.Items.Add("Delete " + fname);
            }
        }

        //override hàm Epilog
        public override void Epilog() { }
    } //hết class
} //hết namespace

```

VI.8 Định nghĩa class con phục vụ đếm số lượng file đệ quy :

34. Dời chuột về phần tử gốc của cây Project trong cửa sổ “Solution Explorer”, ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.Class để hiển thị cửa sổ “Add New Item”, chọn mục “Class”, hiệu chỉnh tên class là CFilesCountDlg.cs, chọn button Add để máy tạo 1 class mới.
35. Khi cửa sổ soạn code cho class CFilesCountDlg hiển thị, hiệu chỉnh nội dung của class như sau :

```
using System;
using System.Collections.Generic;
using System.Text;

namespace FileManager {
    class CFilesCountDlg : CRecursiveBrowseDlg {
        //định nghĩa các thuộc tính dữ liệu cần dùng
        private int dcount;
        private int fcount;

        //override hàm InitForm
        public override void InitForm() {
            //hiệu chỉnh các chuỗi caption cho các phần tử giao diện
            this.Text = "Chức năng đếm file & thư mục đệ quy";
            this.lblStartDir.Text = "Thư mục bắt đầu đếm :";
            this.lblPattern.Text = "Nhập pattern :";
            this.lblOutput.Text = "Kết quả đếm :";
            this.btnBrowse.Text = "Browse";
            this.btnStart.Text = "Start";
        }

        //override hàm Prolog
        public override void Prolog() {
            fcount = dcount = 0;
        }

        //override hàm Action để thực hiện đếm số thư mục và số file
        public override void Action(String fname, byte fop) {
            if (fop == 1) { //thư mục
                dcount++;
            } else { //file
                fcount++;
            }
        }

        //override hàm Epilog
        public override void Epilog() {
            lbOutput.Items.Clear();
            lbOutput.Items.Add("So file la : " + fcount);
            lbOutput.Items.Add("So thu muc la : " + dcount);
        }
    } //hết class
} //hết namespace
```

VI.9 Định nghĩa class con phục vụ tính tổng kích thước các file đệ quy :

36. Dời chuột về phần tử gốc của cây Project trong cửa sổ “Solution Explorer”, ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.Class để hiển thị cửa sổ “Add New Item”, chọn mục “Class”, hiệu chỉnh tên class là CFileSizeDlg.cs, chọn button Add để máy tạo 1 class mới.

37. Khi cửa sổ soạn code cho class CFileSizeDlg hiển thị, hiệu chỉnh nội dung của class như sau :

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Runtime.InteropServices;
namespace FileManager {
    class CFileSizeDlg : CrecursiveBrowseDlg {
        //định nghĩa các thuộc tính dữ liệu cần dùng
        ulong ClusterSize;
        ulong SumSize;
        ulong SumWaste;

        //khai báo hàm API Windows cần dùng
        [DllImport("kernel32")]
        public static extern int GetDiskFreeSpace(
            string lpRootPathName,
            out int lpSectorsPerCluster,
            out int lpBytesPerSector,
            out int lpNumberOfFreeClusters,
            out int lpTotalNumberOfClusters
        );

        //override hàm InitForm
        public override void InitForm() {
            //hiệu chỉnh các chuỗi caption cho các phần tử giao diện
            this.Text = "Chức năng tính tổng kích thước các file đệ quy";
            this.lblStartDir.Text = "Thư mục bắt đầu tính :";
            this.lblPattern.Text = "Nhập pattern :";
            this.lblOutput.Text = "Kết quả tính :";
            this.btnBrowse.Text = "Browse";
            this.btnStart.Text = "Start";
        }

        //override hàm Prolog
        public override void Prolog() {
            SumSize = SumWaste = ClusterSize = 0;
        }

        //override hàm Action để thực hiện tổng kích thước các file
        public override void Action(String fname, byte fop) {
            if (ClusterSize == 0) { //nếu chưa có kích thước cluster đĩa
                Int32 cs, ss, fs, ts;
```

```

        string[] parts = fname.Split("\\");
        GetDiskFreeSpace(parts[0] + "\\", out cs, out ss, out fs, out ts);
        ClusterSize = (ulong)cs * (ulong)ss;
    }
    if (fop == 1) { //thư mục
        DirectoryInfo fi = new DirectoryInfo(fname);
    } else { //file
        FileInfo fi = new FileInfo(fname);
        SumSize += (ulong) fi.Length;
        SumWaste += (ClusterSize - (ulong)fi.Length % ClusterSize);
    }
}

//override hàm Epilog
public override void Epilog() {
    lbOutput.Items.Clear();
    lbOutput.Items.Add("Kích thước cluster disk : " + ClusterSize);
    lbOutput.Items.Add("Tổng kích thước các file là : " + SumSize);
    lbOutput.Items.Add("Tổng kích thước lãng phí : " + SumWaste);
    lbOutput.Items.Add("Không gian chiếm trên disk : " + (SumSize + SumWaste));
}
} //hết class
} //hết namespace

```

VI.10 Định nghĩa class con phục vụ tính tổng độ sâu max của cây thư mục :

38. Dời chuột về phần tử gốc của cây Project trong cửa sổ “Solution Explorer”, ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.Class để hiển thị cửa sổ “Add New Item”, chọn mục “Class”, hiệu chỉnh tên class là CDeepLengthDlg.cs, chọn button Add để máy tạo 1 class mới.

39. Khi cửa sổ soạn code cho class CDeepLengthDlg hiển thị, hiệu chỉnh nội dung của class như sau :

```

using System;
using System.Collections.Generic;
using System.Text;

namespace FileManager
{
    class CDeepLengthDlg : CRecursiveBrowseDlg {
        //định nghĩa các thuộc tính dữ liệu cần dùng
        int DeepLength;
        String path;

        //override hàm InitForm
        public override void InitForm() {
            //hiệu chỉnh các chuỗi caption cho các phần tử giao diện
            this.Text = "Chức năng tính độ sâu sâu nhất của 1 nhánh thư mục";
            this.lblStartDir.Text = "Thư mục bắt đầu tính :";
            this.lblPattern.Text = "Nhập pattern :";
            this.lblOutput.Text = "Kết quả tính :";
            this.btnBrowse.Text = "Browse";
        }
    }
}

```

```

        this.btnStart.Text = "Start";
    }

    //override hàm Prolog
    public override void Prolog() {
        DeepLength = 0;
    }

    //override hàm Action để thực hiện tính độ sâu max của cây thư mục
    public override void Action(String fname, byte fop) {
        //tách đường dẫn của phần tử thành các thành phần rời rạc
        string[] parts = fname.Split("\");
        if (parts.Length > DeepLength) { //nếu độ sâu của thành phần hiện hành lớn nhất
            DeepLength = parts.Length;
            path = fname;
        }
    }

    //override hàm Epilog
    public override void Epilog() {
        lbOutput.Items.Clear();
        lbOutput.Items.Add("Do sau max la : " + DeepLength);
        lbOutput.Items.Add("Phan tu co do sau max la : ");
        lbOutput.Items.Add(path);
    }
} //hết class
} //hết namespace

```

VI.11 Định nghĩa class con phục vụ xóa các file bị nhiễm virus exe.exe :

40. Dời chuột về phần tử gốc của cây Project trong cửa sổ “Solution Explorer”, ấn phải chuột vào nó để hiển thị menu lệnh, chọn chức năng Add.Class để hiển thị cửa sổ “Add New Item”, chọn mục “Class”, hiệu chỉnh tên class là CExeVirusDelDlg.cs, chọn button Add để máy tạo 1 class mới.

41. Khi cửa sổ soạn code cho class CExeVirusDelDlg hiển thị, hiệu chỉnh nội dung của class như sau :

```

using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace FileManager {
    class CExeVirusDelDlg : CrecursiveBrowseDlg {

        //override hàm InitForm
        public override void InitForm() {
            //hiệu chỉnh các chuỗi caption cho các phần tử giao diện
            this.Text = "Chức năng xóa các file nhiễm exe.exe virus";
            this.lblStartDir.Text = "Thư mục bắt đầu diệt :";
            this.lblPattern.Text = "Nhập pattern :";
            this.lblOutput.Text = "Các file & thư mục bị xóa :";
        }
    }
}

```

```

        this.btnBrowse.Text = "Browse";
        this.btnStart.Text = "Start";
    }

    //override hàm Prolog
    public override void Prolog() {
        lbOutput.Items.Clear();
    }

    //override hàm Action để thực hiện xóa file bị nhiễm virus exe.exe
    public override void Action(String fname, byte fop) {
        if (fop == 1) { //thư mục
        } else { //file
            FileInfo fi = new FileInfo(fname);
            if (fi.Length == 61440) { //nếu kích thước file = 61440
                File.Delete(fname);
                lbOutput.Items.Add("Delete " + fname);
            }
        }
    }

    //override hàm Epilog
    public override void Epilog() { }
} //hết class
} //hết namespace

```

VI.12 Viết code cho chương trình chính

42. Duyệt tìm mục Form1.cs trong cửa sổ “Solution Explorer” miêu tả cửa sổ chính của chương trình, ấn phải chuột vào nó để hiển thị menu lệnh, chọn lệnh “View Code” để hiển thị cửa sổ soạn mã nguồn của class Form tương ứng. Thêm lệnh định nghĩa thuộc tính dữ liệu sau vào đầu class Form1 :

```

    ICommand dlg; //tham khảo đến form chức năng mà người dùng muốn dùng

```

43. Duyệt tìm mục Form1.cs trong cửa sổ “Solution Explorer” miêu tả cửa sổ chính của chương trình, ấn kép chuột vào nó để hiển thị lại cửa sổ chính của chương trình.
44. Click chuột vào menu File để hiển thị các MenuItem của nó. Dời chuột về MenuItem “Xóa file đệ quy” rồi ấn kép chuột trên nó để tạo hàm xử lý sự kiện Click chuột trên nó. Khi cửa sổ code hiển thị, viết code đơn giản sau đây cho hàm :

```

    //hàm xóa file theo pattern
    private void mnuFilesDelete_Click(object sender, EventArgs e)
    {
        //tạo Form xóa file theo pattern và hiển thị nó để người dùng làm việc
        dlg = new CFilesDeleteDlg();
        dlg.Show();
    }

```

45. Lập lại các bước 40 và 41 nhiều lần để tạo và viết các hàm xử lý sự kiện Click chuột cho các MenuItem chức năng còn lại. Code của các hàm như sau :

```

    //hàm đếm số file theo pattern
    private void mnuFilesCount_Click(object sender, EventArgs e)
    {

```

```
//tạo Form đếm số file theo pattern và hiển thị nó để người dùng làm việc
dlg = new CFilesCountDlg();
dlg.Show();
}
```

```
//hàm xóa các file bị nhiễm virus exe.exe
private void mnuFileVirusDel_Click(object sender, EventArgs e)
{
    //tạo Form xóa virus và hiển thị nó để người dùng làm việc
    dlg = new CExeVirusDelDlg();
    dlg.Show();
}
```

```
//hàm tính độ sâu max
private void mnuFileDeepLen_Click(object sender, EventArgs e)
{
    //tạo Form đếm số file theo pattern và hiển thị nó để người dùng làm việc
    dlg = new CDeepLengthDlg();
    dlg.Show();
}
```

```
//hàm tính tổng kích thước các file
private void mnuFilesSize_Click(object sender, EventArgs e)
{
    //tạo Form tính tổng kích thước các file theo pattern
    //và hiển thị nó để người dùng làm việc
    dlg = new CFilesSizeDlg();
    dlg.Show();
}
```

```
//hàm hiển thị thông tin trợ giúp
private void mnuHelpHelp_Click(object sender, EventArgs e)
{
    MessageBox.Show("Xin lỗi, chức năng này chưa được hiện thực!");
}
```

```
//hàm hiển thị thông tin về chương trình
private void mnuHelpAbout_Click(object sender, EventArgs e)
{
    MessageBox.Show("Xin lỗi, chức năng này chưa được hiện thực!");
}
```

46. Duyệt tìm mục Form1.cs trong cửa sổ “Solution Explorer” miêu tả cửa sổ chính của chương trình, ấn kép chuột vào nó để hiển thị lại cửa sổ chính của chương trình.

47. Dời chuột về Button “Xóa file đệ quy” trong Toolbar rồi ấn kép chuột trên nó để tạo hàm xử lý sự kiện Click chuột trên nó. Khi cửa sổ code hiển thị, viết code đơn giản sau đây cho hàm :

```
//hàm xóa file theo pattern
private void tbFilesDelete_Click(object sender, EventArgs e)
{

```



```
//gọi hàm xử lý MenuItem có chức năng tương ứng
mnuFilesDelete_Click(sender, e);
}
```

48. Lập lại các bước 40 và 41 nhiều lần để tạo và viết các hàm xử lý sự kiện Click chuột cho các Button chức năng còn lại trong Toolbar. Code của các hàm như sau :

```
//hàm đếm số file theo pattern
private void tbFilesCount_Click(object sender, EventArgs e)
{
    //gọi hàm xử lý MenuItem có chức năng tương ứng
    mnuFilesCount_Click(sender, e);
}
```

```
//hàm xóa các file bị nhiễm virus exe.exe
private void tbFileVirusDel_Click(object sender, EventArgs e)
{
    //gọi hàm xử lý MenuItem có chức năng tương ứng
    mnuFileVirusDel_Click(sender, e);
}
```

```
//hàm tính độ sâu max
private void tbFileDeepLen_Click(object sender, EventArgs e)
{
    //gọi hàm xử lý MenuItem có chức năng tương ứng
    mnuFileDeepLen_Click(sender, e);
}
```

```
//hàm tính tổng kích thước các file
private void tbFilesSize_Click(object sender, EventArgs e)
{
    //gọi hàm xử lý MenuItem có chức năng tương ứng
    mnuFilesSize_Click(sender, e);
}
```

```
//hàm hiển thị thông tin trợ giúp
private void tbHelpHelp_Click(object sender, EventArgs e)
{
    //gọi hàm xử lý MenuItem có chức năng tương ứng
    mnuHelpHelp_Click(sender, e);
}
```

```
//hàm hiển thị thông tin về chương trình
private void tbHelpAbout_Click(object sender, EventArgs e)
{
    //gọi hàm xử lý MenuItem có chức năng tương ứng
    mnuHelpAbout_Click(sender, e);
}
```

49. Chọn menu Debug.Start Debugging để dịch và chạy thử ứng dụng, nếu máy báo lỗi thì tìm hiểu và sửa lỗi cho đến khi hết lỗi. Khi cửa sổ chính chương trình hiển thị, hãy thử từng lệnh trong MenuBar và trong Toolbar để kiểm tra kết quả.

VI.13 Xem xét code cho 5 class con và rút ra các kết luận hữu ích :

- **Tính thừa kế** trong hướng đối tượng cho phép ta tập trung viết code dùng chung cho nhiều chức năng trong 1 class (cha), chứ không cần lập lại trong từng class (con) miêu tả từng chức năng.
- Để miêu tả từng chức năng, ta chỉ cần định nghĩa 1 class con thừa kế class cha đã có rồi override các hàm primitive, cụ thể ta chỉ cần override các hàm sau :
 - ✓ Hàm InitForm để thiết lập chuỗi caption cho các đối tượng giao diện.
 - ✓ Hàm Prolog (nếu cần) để thiết lập giá trị đầu cho các thuộc tính dữ liệu.
 - ✓ Hàm Action để miêu tả công việc xử lý trên phần tử tìm được.
 - ✓ Hàm Epilog (nếu cần) để thực hiện 1 số việc kết thúc chức năng.
- Mặc dù các hàm template được viết trong class cha và gọi các hàm primitive được định nghĩa trong class cha (lúc này các class con chưa có), nhưng khi chạy trên đối tượng thuộc class con nào, nó sẽ kích hoạt được đúng hàm primitive được override trong class con đó (nhờ **tính đa xạ**).
- Mặc dù các class chức năng có nhiều thành phần có tầm vực public, protected,... nhưng chương trình chính (Client) dùng các đối tượng thông qua interface ICommand, interface này chỉ có 1 hàm chức năng Show() nên chương trình chính chỉ có thể gọi được hàm Show chứ không thể truy xuất được dịch vụ gì khác của các đối tượng mà mình đang tham khảo tới. Điều này thể hiện rất rõ nét **tính bao đóng** và lợi ích của nó.
- Dùng macro **[DllImport]** trong thư viện **System.Runtime.InteropServices** để chuyển 1 hàm API có sẵn trong thư viện lập trình cổ điển thành 1 tác vụ của 1 class VC#, nhờ đó ứng dụng hướng đối tượng VC# có thể gọi các hàm API dễ dàng (nhưng theo cơ chế gọi thông điệp của hướng đối tượng).