algorithmica
re-engineering it talent

Practical Object oriented design
(File Explorer Application)

Overview

In this project, you are required to develop a software application that manages file systems, called MyUFO (My Universal File Organizer). MyUFO is an applpcation which provides functionalities for browsing and organizing files. We will use incremental development process to develop MyUFO. The development of MyUFO is divided into several iterations. Each iteration has a number of requirements to be implemented. In general, the requirements include implementing a number of system features, applying one or two design patterns in your design, writing and performing unit testing in your program, and drawing class diagrams for your design. You have to perform Unit and Integration testing as well. Automate the testing of every method of every class by using unit test framework(e.g.,jUnit/CppUnit). But, trivial methods (e.g., getter and setter functions) can be neglected. After finished unit testing, you are required to do integration testing. Integration testing is the phase in software testing in which individual software modules (classes) are combined and tested as a group.

algorithmica
re-engineering it talent

Practical Object oriented design (File Explorer Application)

MileStone1: Text-mode interface

In the first homework, you need to implement some basic methods and a text-mode interface for MyUFO. The text-mode interface is showed in Table 1.

Table 1 Text-mode interface

1. Select directory path
2. List directory
3. Create a file/directory
4. Remove a file/directory
5. Rename a file/directory
6. Count total size of files under the directory
7. Count number of files and directories under the directory
8. Exit

Command:

There are 8 functionalities in the command menu. Before a user executes any command, the user can use **"Select directory path"** option to select a directory path to manage. If the user doesn't select the directory path, the default value is the application's current directory. The user can use **"List directory"** option to display the entities under the directory which is selected in "Select directory path" option. The list of directory **must be sorted in increasing order**, and all directory entities must appear before file entities. In order to identify the directories and files, all of directory entities must have a "<D>" prefix and all of file entities must have a "<F>" prefix in the list. (See Table 2)

Table 2 List directory

Type	File Name	Extension Name	File Size	Modify Time
<D>	.		0Bits	16-09-2010 12:50:32
<D>	..		0Bits	15-09-2010 14:16:31
<D>	4567		0Bits	15-09-2010 18:34:36
<D>	Debug		0Bits	16-09-2010 13:39:53
<D>	src		0Bits	16-09-2010 10:58:27
<F>		cproject	54.2715KB	15-09-2010 14:16:34
<F>		project	2.09668KB	15-09-2010 14:16:34
<F>	123	txt	0Bits	15-09-2010 18:34:26

In addition, a user can also use **"Create a file/directory," "Remove a file/directory"**, and **"Rename a file/directory"** commands to manage the entities under the directory. If

Practical Object oriented design (File Explorer Application)

a user wants to understand the total size of this directory and the number of files and directories under the directory, he can use **"Count total size of files under the directory"** and **"Count the number of files and directories under the directory"** option to display the information. Finally, the **"Exit"** option will close the application.

Table 3 is a user scenario of MyUFO. A user selects "c:\data\" as default directory, uses the "List directory" option to list the directory, creates a new file called "newfile.txt", renames the file name to "myfile.txt", and then deletes "myfile.txt." He also wants to know the information about this directory, so he uses the "Count total size of files under the directory" and "Count the number of files and directories under the directory" to get the folder information. Finally, he uses the "Exit" option to exit the system.

Practical Object oriented design (File Explorer Application)

Table 3 Use scenario

```
1.    Select directory path

2.    List directory

3.    Create a file/directory

4.    Remove a file/directory

5.    Rename a file/directory

6.    Count total size of files under the directory

7.    Count number of files and directories under the directory

8.    Exit

Command:1

Please key in a directory path:C:\Data

1.    Select directory path

2.    List directory

3.    Create a file/directory

4.    Remove a file/directory

5.    Rename a file/directory

6.    Count total size of files under the directory

7.    Count number of files and directories under the directory

8.    Exit

Command:2

Type File Name                Extension Name            File Size                Modify Time

=============================================================================================================

<D>  .                                                  0Bytes                   16-09-2010 12:50:32

<D>  ..                                                 0Bytes                   15-09-2010 14:16:31

<D>  4567                                               0Bytes                   15-09-2010 18:34:36

<D>  Debug                                              0Bytes                   16-09-2010 13:45:59
```

algorithmica
re-engineering it talent

Practical Object oriented design
(File Explorer Application)

```
<D>  src                                                0Bytes                  16-09-2010 10:58:27
<F>                          cproject                   54.2715KB                15-09-2010 14:16:34
<F>                          project                    2.09668KB                15-09-2010 14:16:34
<F>  123                      txt                        0Bytes                  15-09-2010 18:34:26
1.     Select directory path
2.     List directory
3.     Create a file/directory
4.     Remove a file/directory
5.     Rename a file/directory
6.     Count total size of files under the directory
7.     Count number of files and directories under the directory
8.     Exit
Command:3
Please key in Entity name:newfile.txt
Please choose Entity type[1:file/2:folder]:1
1.     Select directory path
2.     List directory
3.     Create a file/directory
4.     Remove a file/directory
5.     Rename a file/directory
6.     Count total size of files under the directory
7.     Count number of files and directories under the directory
8.     Exit
Command:5
Please key in Entity name:newfile.txt
Please key in the new name:posdhw1.txt
1.     Select directory path
2.     List directory
3.     Create a file/directory
4.     Remove a file/directory
5.     Rename a file/directory
6.     Count total size of files under the directory
7.     Count number of files and directories under the directory
8.     Exit
Command:4
Please key in Entity name:posdhw1.txt
1.     Select directory path
2.     List directory
3.     Create a file/directory
```

Practical Object oriented design
(File Explorer Application)

```
4.     Remove a file/directory

5.     Rename a file/directory

6.     Count total size of files under the directory

7.     Count number of files and directories under the directory

8.     Exit

Command:6

Total size of files: 18.7875MB

1.     Select directory path

2.     List directory

3.     Create a file/directory

4.     Remove a file/directory

5.     Rename a file/directory

6.     Count total size of files under the directory

7.     Count number of files and directories under the directory

8.     Exit

Command:7

Files: 18 Directories:4

1.     Select directory path

2.     List directory

3.     Create a file/directory

4.     Remove a file/directory

5.     Rename a file/directory

6.     Count total size of files under the directory

7.     Count number of files and directories under the directory

8.     Exit

Command:8

Goodbye
```

Practical Object oriented design (File Explorer Application)

MileStone2: Implement file-system using composite pattern

A file system is a tree structure that contains *branches* (folders) as well as leaf nodes (files). Note that a folder object usually contains one or more files or folder objects and thus is a composite object, while a file is a simple object. Note also that files and folders have many operations and attributes in common, such as moving and copying a file or a folder, listing file or folder attributes such as file name and size. Therefore, it would be easier and more convenient to treat both file and folder objects uniformly by defining a file system resource interface.

In this homework, you **must** implement composite pattern to present the file system structure. The following figure represents the core class diagram of MyUFO - a typical composite pattern. File and Directory are two subclasses of Entity. File class doesn't contain any entities, so it plays the leaf role in composite pattern. The Directory object contains either file or directory objects and thus is a composite role.

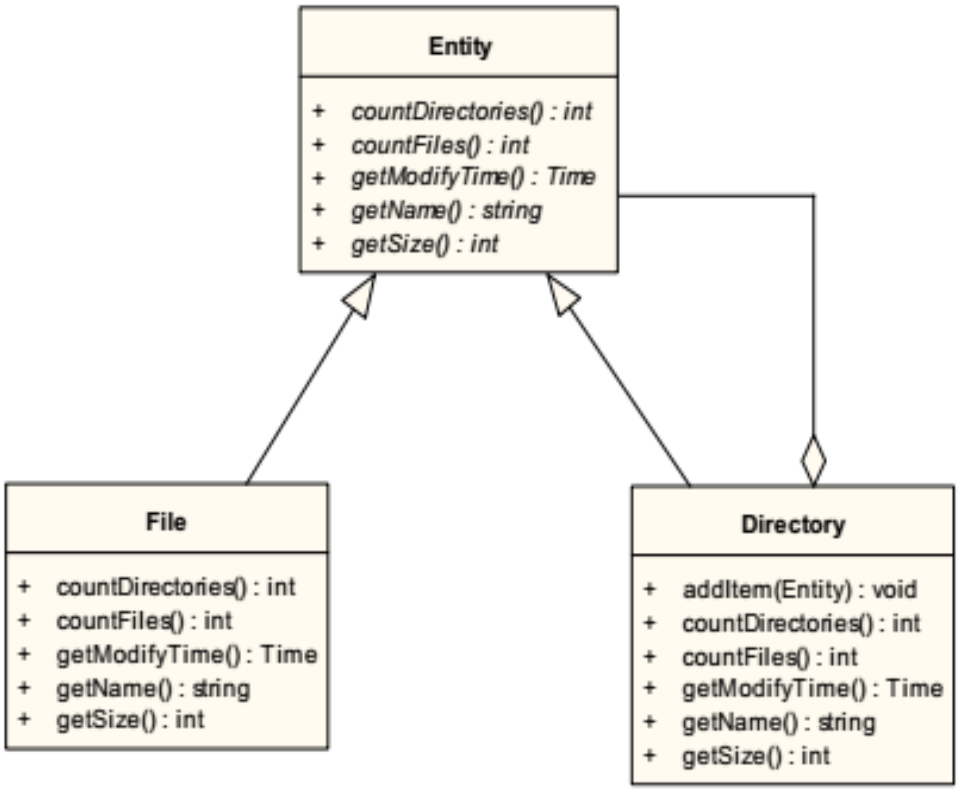


Figure 2 Composite pattern

algorithmica
re-engineering it talent

Practical Object oriented design (File Explorer Application)

MileStone3: Implement directory listing using strategy pattern

In Milestone#1, a list directory command was implemented, but this command always lists the content of the directory by their names. We want to extend the command to support another display strategy by applying strategy pattern. Following figure is the core part of MyUFO class diagram when strategy pattern is applied.

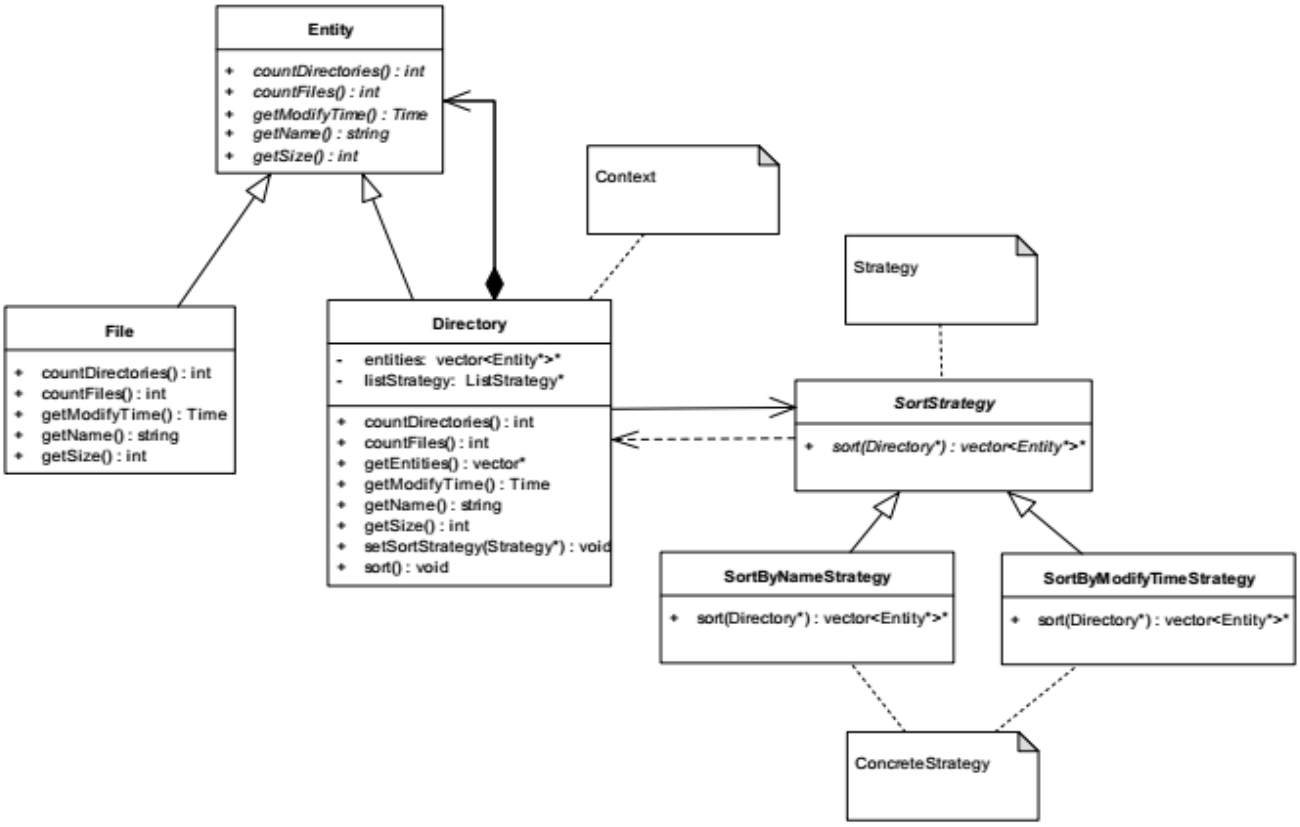


Figure 1 Strategy pattern for listing directory

There are two phases when a list command is issued. The first phase of the program sorts the entities in the directory, and the second phase lists the sorted entities. In this case, the sorting strategy is a variable part in the program. Strategy pattern helps us decouple the sorting strategy from the Directory class. This allows us to change the sorting strategy at runtime. In this homework, **two directory sorting strategies must be implemented**, namely **SortByNameStrategy** and **SortByModifyTimeStrategy**. The **SortByNameStrategy** is the same as the list command which we have already implemented in Milestone#1. The entities are sorted in increasing order according to their names. The **SortByModifyTime Strategy** is a new strategy to be implemented in this homework. Following Table is a sample result of listing by modification time. A user

Practical Object oriented design (File Explorer Application)

selects the **SortByModifyTime Strategy**, and uses the list command to display the list of entities. The entities **are sorted by their modification time in descending order.** If the user did not select the directory sorting strategy, the entities under the directory are listed using **SortByNameStrategy** by default.

```
0.    Select directory listing strategy

1.    Select directory path

2.    List directory

3.    Create a file/directory

4.    Remove a file/directory

5.    Rename a file/directory

6.    Copy single file

7.    Move single file

8.    Count total size of files under the directory

9.    Count number of files and directories under the directory

10.   Exit

Command:0

[1]List directory by name [2]List directory by last modification time:2

Command:2

Type File Name                  Extension Name              File Size                 Modify Time

-------------------------------------------------------------------------------------------------------------------

<D>  .                                                      0Bytes                    16-09-2010 12:50:32

<D>  ..                                                     0Bytes                    15-09-2010 14:16:31

<D>  Debug                                                  0Bytes                    16-09-2010 13:39:53

<D>  src                                                    0Bytes                    16-09-2010 10:58:27

<D>  4567                                                   0Bytes                    15-09-2010 18:34:36

<F>  123                        txt                         0Bytes                    15-09-2010 18:34:26

<F>                             cproject                    54.2715KB                 15-09-2010 14:16:34

<F>                             project                     2.09668KB                 15-09-2010 14:16:34
```

Practical Object oriented design (File Explorer Application)

MileStone4: Adding more file management functionalities

In order to offer more file management functionalities, we will implement two new commands in this homework. The first one is "**Copy single file**" command. This command allows user to copy a file from one directory to another. The following table shows an example of executing "Copy single file" command. A user copies a "POSDHW2.txt" file from the "doc" directory. After executing the command, a copy in the "c:\Data" directory is created.

Table 2 User scenario of "Copy single file"

```
0.    Select directory listing strategy

1.    Select directory path

2.    List directory

3.    Create a file/directory

4.    Remove a file/directory

5.    Rename a file/directory

6.    Copy single file

7.    Move single file

8.    Count total size of files under the directory

9.    Count number of files and directories under the directory

10.   Exit

Command:1

Please key in a directory path:C:\Data

Command:6

Please key in source location:doc/POSDHW2.txt

Please key in target location:POSDHW2.txt

Command:2

Type File Name                Extension Name          File Size              Modify Time
-------------------------------------------------------------------------------------------------------------

<D>  .                                                0Bytes                 16-09-2010 12:50:32

<D>  ..                                               0Bytes                 15-09-2010 14:16:31

<D>  doc                                              0Bytes                 15-09-2010 18:34:36

<F>  POSDHW2                      txt                 54.2715KB                 15-09-2010 14:16:34
```

Practical Object oriented design (File Explorer Application)

The second command is "**Move single file**" command. This command allows users to move a file from one directory to another. The following table shows an example of executing "Move single file" command. After executing the command, the source file will be deleted and a copy of the source file will be created in the target directory. If there is a file with the same name in the target directory, the application will ask the user whether to overwrite the file or cancel the operation.

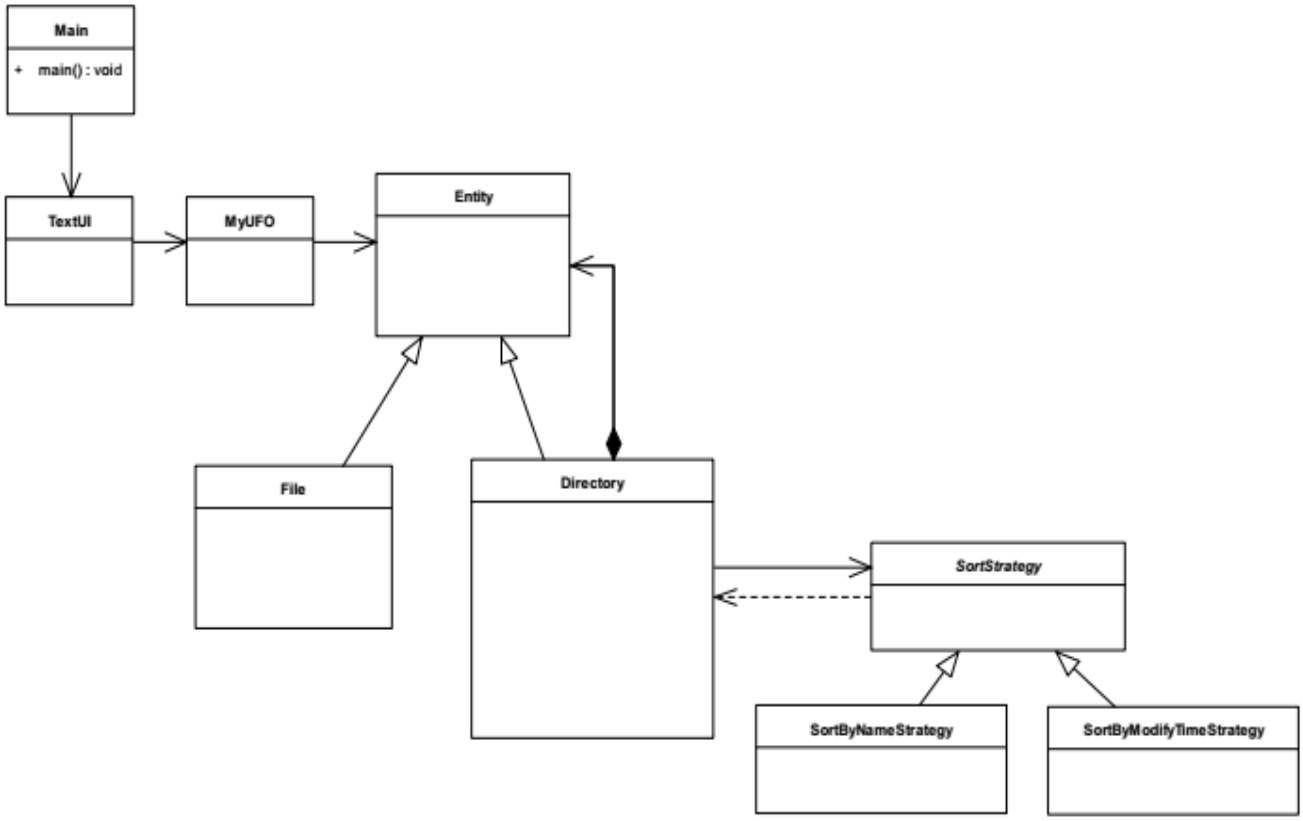
Table 3 User scenario of "Move single file"

```
0.    Select directory listing strategy

1.    Select directory path

2.    List directory

3.    Create a file/directory

4.    Remove a file/directory

5.    Rename a file/directory

6.    Copy single file

7.    Move single file

8.    Count total size of files under the directory

9.    Count number of files and directories under the directory

10.   Exit

Command:1

Please key in a directory path:C:\Data

Command:7

Please key in source location:POSDHW2.txt

Please key in target location:doc/POSDHW2.txt

Overwrite the existing file? [Y/n]y

Command:2

Type File Name                Extension Name          File Size              Modify Time

-------------------------------------------------------------------------------------------------------------------

<D>  .                                                0Bytes                 16-09-2010 12:50:32

<D>  ..                                               0Bytes                 15-09-2010 14:16:31

<D>  doc                                              0Bytes                 15-09-2010 18:34:36
```

Practical Object oriented design (File Explorer Application)

MileStone5: Separating Model from text-mode UI

One of the most important things to do is to separate model code from view code (text-mode UI code). The model classes (e.g. **MyUFO**, **Entity**, **File** and **Directory**) should not depend on UI. In addition, you are required to do some refactoring to separate text-mode UI code from model classes. The separation implies that the model classes do not use any standard I/O methods. For example, "**printf**", "**scanf**", "**cin**", "**sysout**" and "**cout**" are not allowed to appear in the model classes. A reference class diagram is shown in following figure. The TextUI class is the text-mode UI. Note that a new class, MyUFO, is used to wrap (and control) all model classes.



Practical Object oriented design
(File Explorer Application)

MileStone6: Implement GUI front end

The following figure is the GUI front-end to be implemented in this homework.

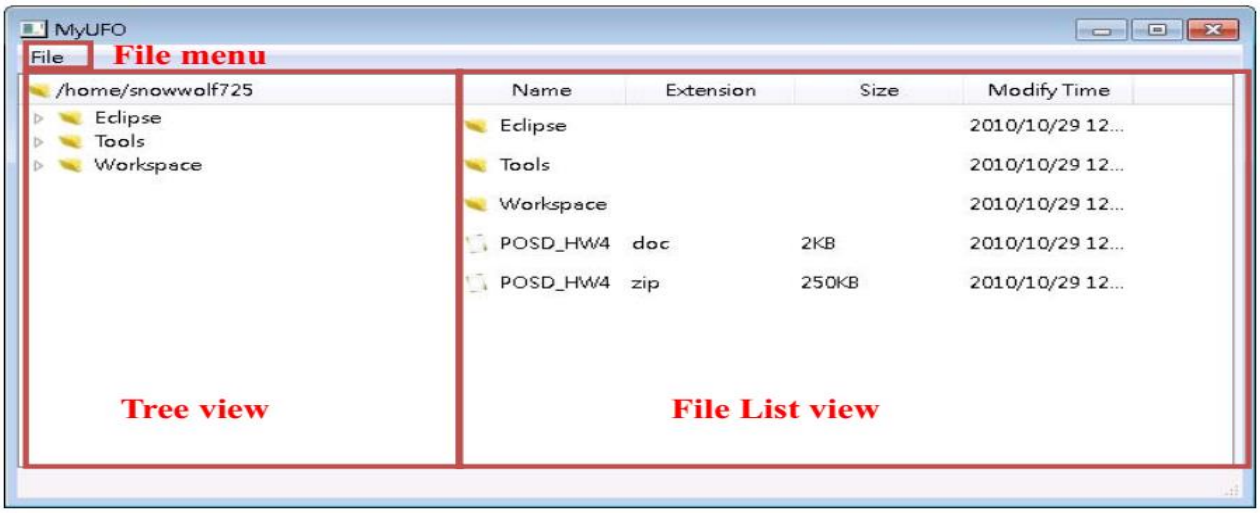


Figure 1 MyUFO GUI

MyUFO's window has a File menu, a Tree view and a File list view. MyUFO displays the file system tree in the left part. The tree view only displays directory items, and does not display file items. The right part of MyUFO's window is a file list view, which displays the entities under the directory selected in the Tree view. The file list view contains the information of the entities, including "Name", "Extension", "Size", and "Modify Time." When MyUFO is started, MyUFO's current working directory is displayed in the Tree view and File list view. The File menu contains only two menu items, "**Select Path**" and "**Exit**."

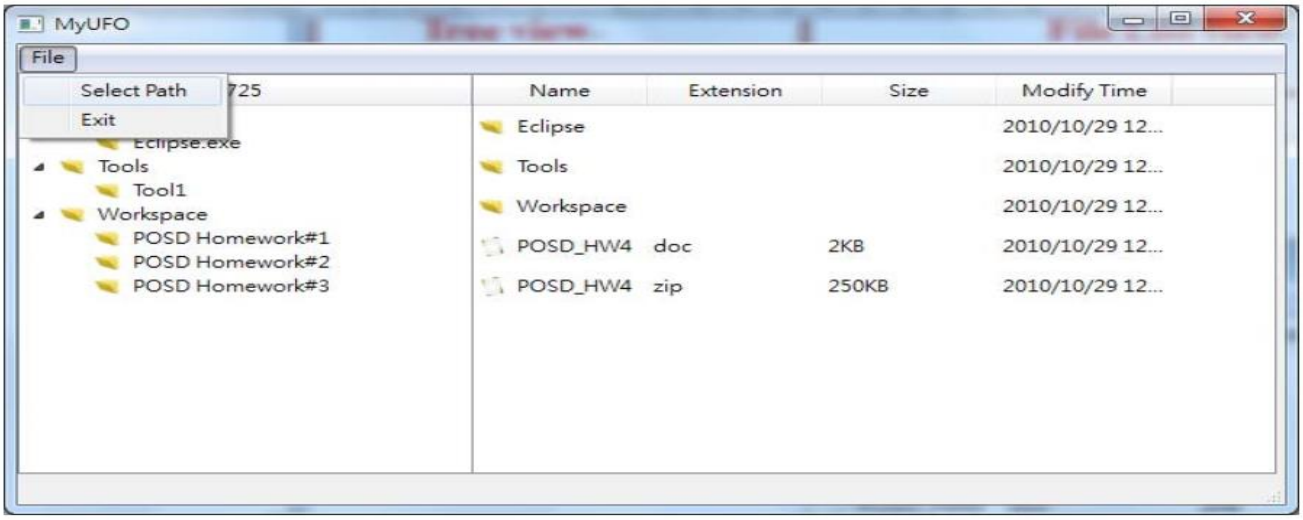


Figure 2 MyUFO Menu

Practical Object oriented design (File Explorer Application)

When a user clicks the "Select Path" menu item, a dialog should be opened (see Figure 3).

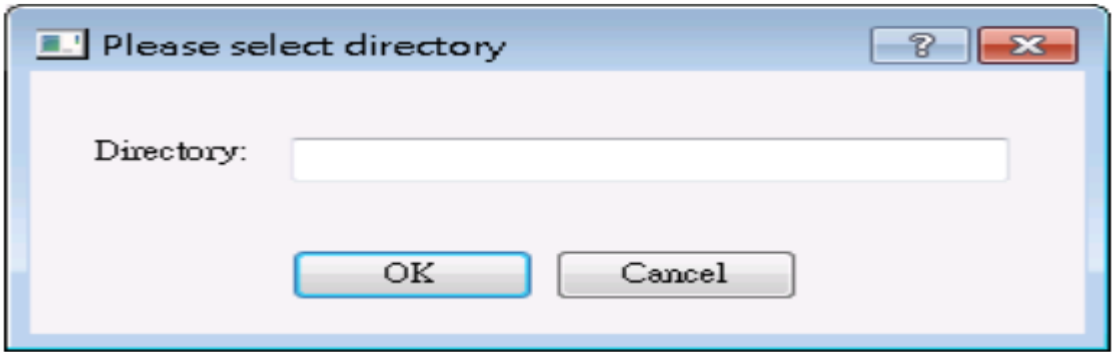


Figure 3 Dialog of path selection

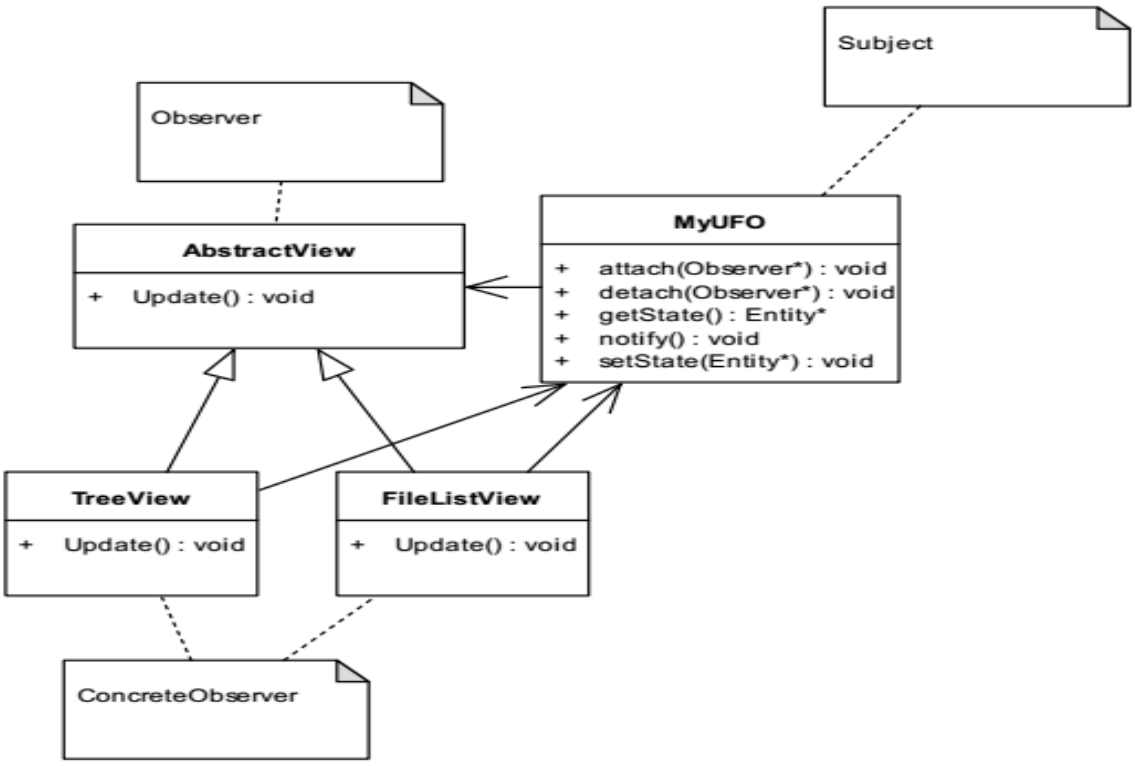
After the user enters the path in the text field and clicks "OK" button, the Tree view should immediately refresh its display to show the directories under the selected path, and the File list view should show all the entities under the selected path. A user can exit the application by clicking the Exit menu item. The inclusion of the new GUI front-end should not break the existing text mode front-end. You are required to make MyUFO to support both GUI and text UI. However, to simply the problem, you may choose one of the following three implementations: (1) enable TextUI and GUI simultaneously, (2) use command line option "**textmode**" to indicate that TextUI is used (the default is GUI), (3) create two projects, one for Text mode UI and another for GUI.

Practical Object oriented design
(File Explorer Application)

MileStone7: Implement View observers

In **MyUFO**, the user interface has two different views: **Tree view** and **File List view**. Both views show different aspects of the model. When a user clicks a directory in the Tree view, the directory should be highlighted to indicate that the directory is selected, and the File List view should immediately show the entities under the selected directory. When a user double clicks a directory in the File List view, the highlighted (selected) directory in the Tree view should be automatically changed to the newly selected directory.

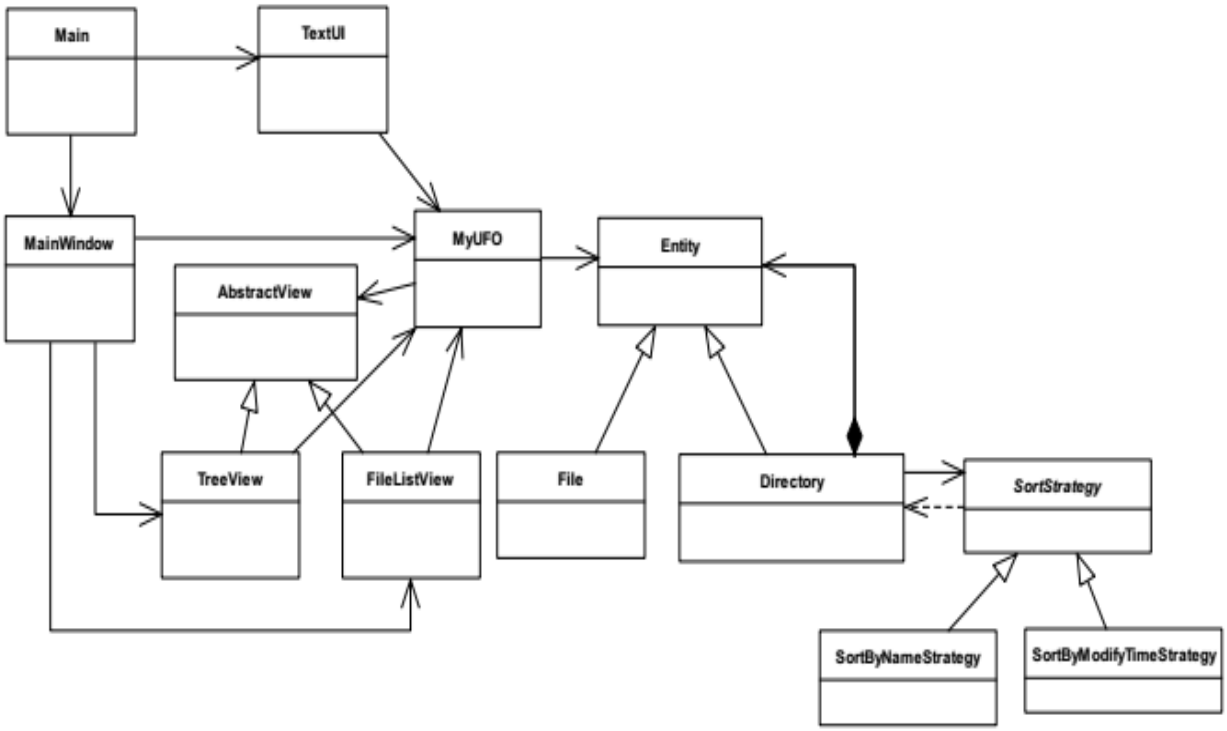
The Observer pattern describes how to establish these relationships. The key objects in this pattern are Subject and Observer. Several observers may depend on the same subject. If the subject's state is changed, all observers will be notified. The following figure shows the class diagram when Observer pattern is applied. An abstract class called **AbstractView** is used to represent the base class of **Observer**, and it has two subclasses (TreeView and FileListView) representing the two different views in the application.



The following figure shows the reference design class diagram. The MainWindow class is GUI front-end for MyUFO; it cotains a TreeView and a FileListView. Both views show different aspects of the MyUFO. The Main class can choose to execute TextUI to open a text-mode front-end, or executes MainWindow to open a GUI front-end.

algorithmica
re-engineering it talent

Practical Object oriented design
(File Explorer Application)



Practical Object oriented design
(File Explorer Application)

MileStone8: Menu Implementation

In MileStone#6, a simple File menu was implemented. In this milestone, we will add new functionalities to the menu bar to improve MyUFO. Table 1 shows the menu items which you need to implement in this milestone.

Table 1 MyUFO menu list

File	
Select Path	Select the Current Directory
Exit	Exit MyUFO program
List Strategy	
List By Name	Change the default strategy of FileListView to ListByName Strategy
List By Modify Time	Change the default strategy of File list view to ListByModifyTime Strategy
Edit	
Undo	Undo the last command
Redo	Redo the last undone command
Add file	MyUFO pops a dialog and asks user to give a file name for the new file. After user enters a valid file name, an empty file was added to the directory selected in the Tree view.
Add directory	MyUFO pops a dialog and asks user to give a directory name for the new directory.

Practical Object oriented design
(File Explorer Application)

	After user enters a valid directory name, an empty directory was added to the directory selected in the Tree view.
Copy file	MyUFO pops a dialog and asks user to give target directory. After user enters a valid directory path, the selected file was copied to the target directory.
Delete	MyUFO pops a dialog (Figure 1) and asks user to confirm. If user selects the yes option in the dialog, all the selected files will be deleted.
Move file	MyUFO pops a dialog and asks user to give target directory. After user enters a valid directory path, the selected file was moved to the target directory.
Rename	MyUFO pops a dialog and asks user to give a new name for the entity. After user enters a valid name, the name of entity will be changed to new file name.
Statistics	
Count total size of files under the directory	MyUFO pops a dialog (Figure 2) and shows the total size of files under the directory
Count number of files and directories under the directory	MyUFO pops a dialog (Figure 3) and shows the number of files and directories under the directory

Practical Object oriented design
(File Explorer Application)

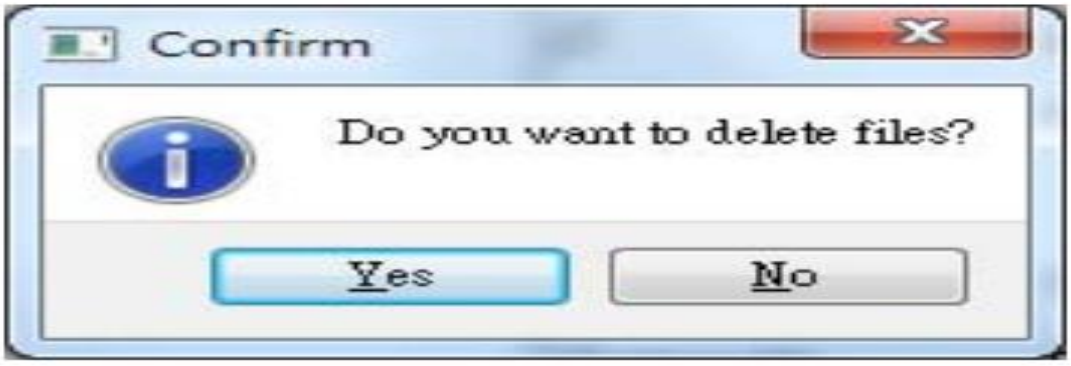


Figure 1 Delete file

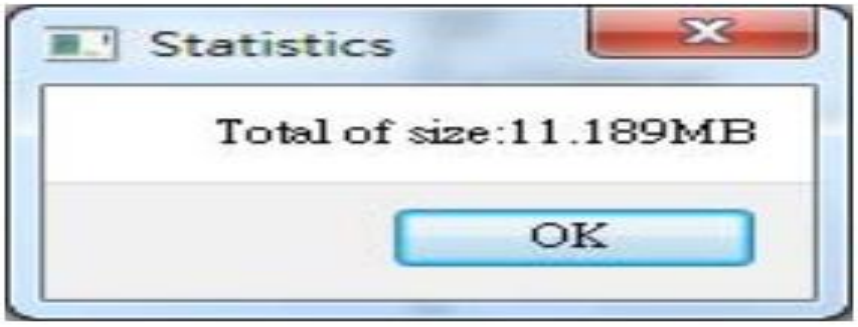


Figure 2 Count total size

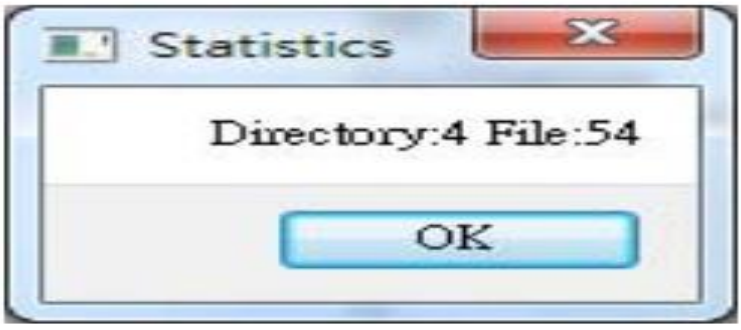


Figure 3 Count number of entities

Practical Object oriented design
(File Explorer Application)

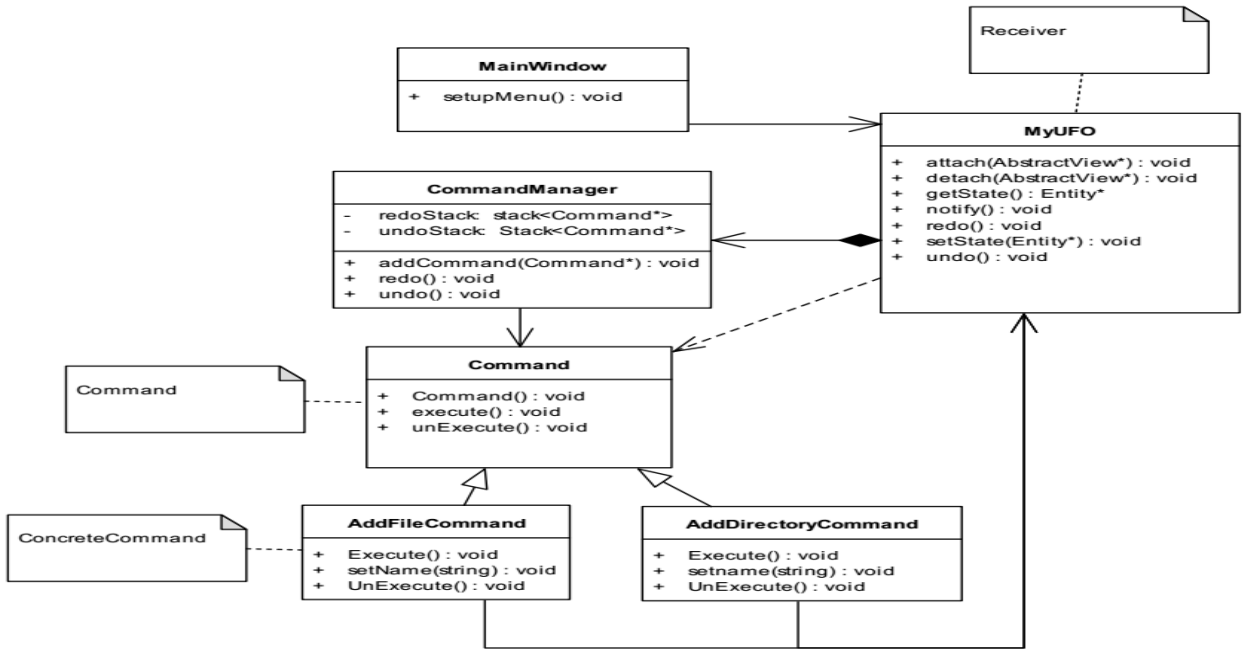
MileStone9: Redo and Undo Implementation

Redo and **undo** operations are common features supported by most modern GUI applications. Undo command erases the last change done to the application, and reverts to an old state. The opposite of undo is Redo which lets a user re-executes the last undone action. You are required to support redo/undo for the operations listed in following table.

Table 2 Operations that must support Undo/Redo

Commands
Add file
Add directory
Copy file
Move file
Rename

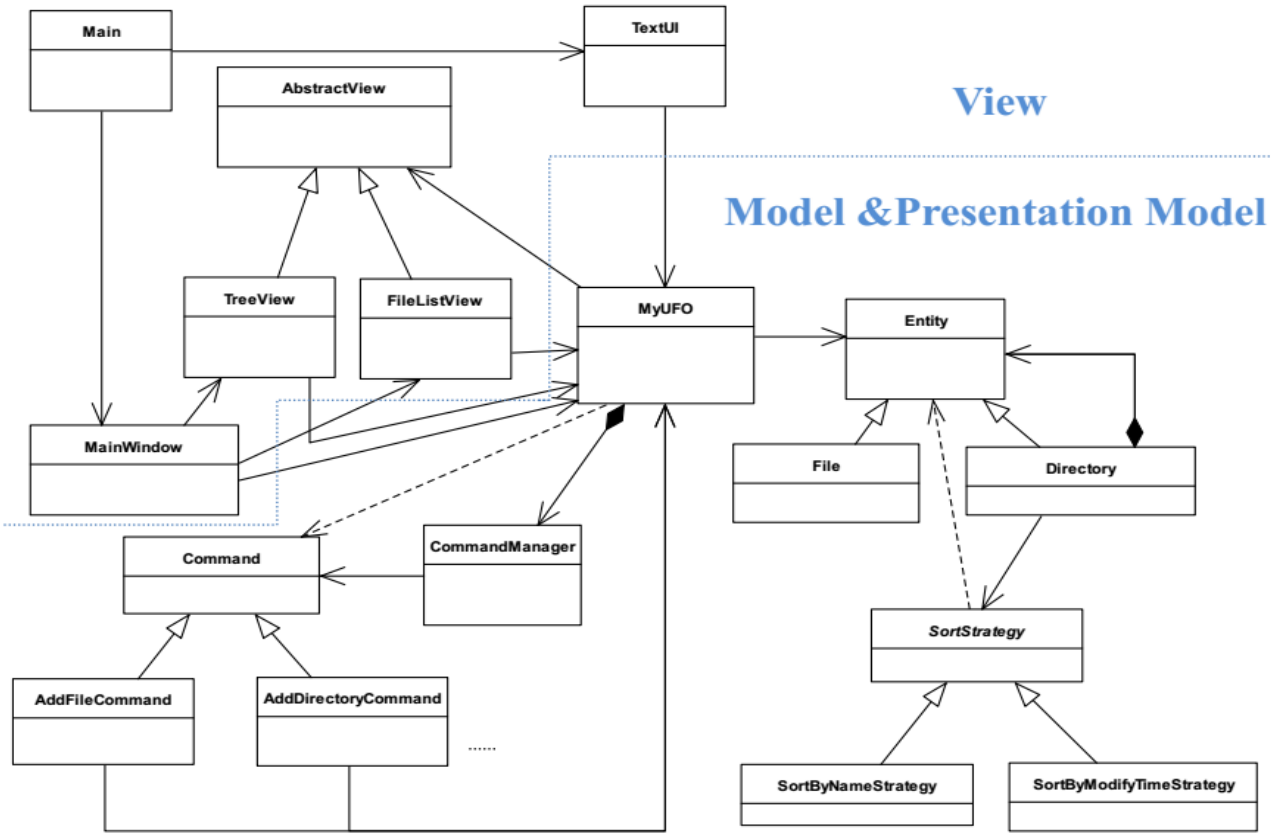
In this milestone, you need to apply command pattern to implement redo/undo operations. In the following figure, a Command class encapsulates a request. For example, the AddFileCommand encapsulates the request of adding a file. MyUFO is the receiver of the command. In order to implement the redo/undo, two stacks are needed to store ***commandhistory.*** Each time when undo command is performed, application gets the most recent operation from the undo stack (i.e., the top element in undo stack), executes the UnExecute() method to replace application state with the previous state, and moves the command from the undo stack to the redo stack. When a user executes the redo command, the application retrieves the top element in redo stack, and automatically executes the Execute() method.



algorithmica
re-engineering it talent

Practical Object oriented design (File Explorer Application)

The following figure shows the reference design class diagram:



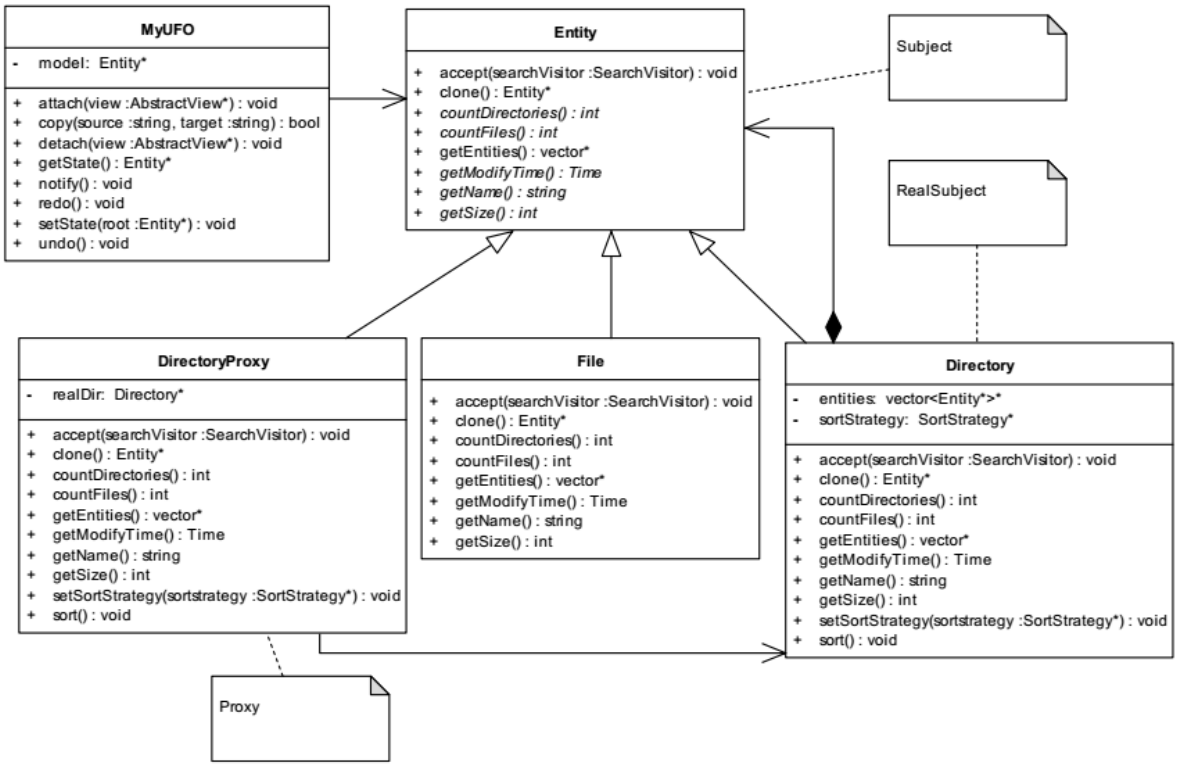
algorithmica
re-engineering it talent

Practical Object oriented design (File Explorer Application)

MileStone10: Applying Proxy Pattern for Directory Creation

When a user selects a path, MyUFO must build a model (instances of the composite pattern) for the selected directory. This is time-consuming if the selected directory contains a large number of subdirectories and files. In order to enhance the performance, we will use Proxy pattern to implement lazy loading Mechanism. **Lazy Loading** is a programming practice in which you only load or initialize an object when you first need it. This can potentially give you a performance boost, especially if you have a lot of entities under the directory.

Proxy pattern provides a surrogate or placeholder for another object to control access to it. A reference class diagram is given in following figure. DirectoryProxy plays the proxy role for a Directory (RealSubject). DirectoryProxy contains a Directory object, and DirectoryProxy can be treated as a Directory object. DirectoryProxy always delegates its operations to the real subject (Directory). The only difference between DirectoryProxy and Directory is the management of sub-directories. In order to implement lazy loading, the sub-directories under the DirectoryProxy are also DirectoryProxy objects. In fact, all of directories in MyUFO will be presented as DirectoryProxy objects. The Directory object in the DirectoryProxy will be initialized only when the user first accesses it.



Practical Object oriented design
(File Explorer Application)

In order to clarify lazy loading mechanism, we will use an example to describe how it works. When a user select root ("/") as the current directory, MyUFO will build a Directory Proxy object for the root directory. The select event triggers the lazy loading Mechanism. The actual instance of root directory will be created, and all sub-directories under the root directory will be added into the real root directory object. However, all sub-directories will be created as Directory Proxy objects. In addition, if the user selects a directory (e.g. "/home") which contains directories and files, the files (ex. "file1.txt" and "file2.txt") will also be added into the Directory object (see Figure 3).

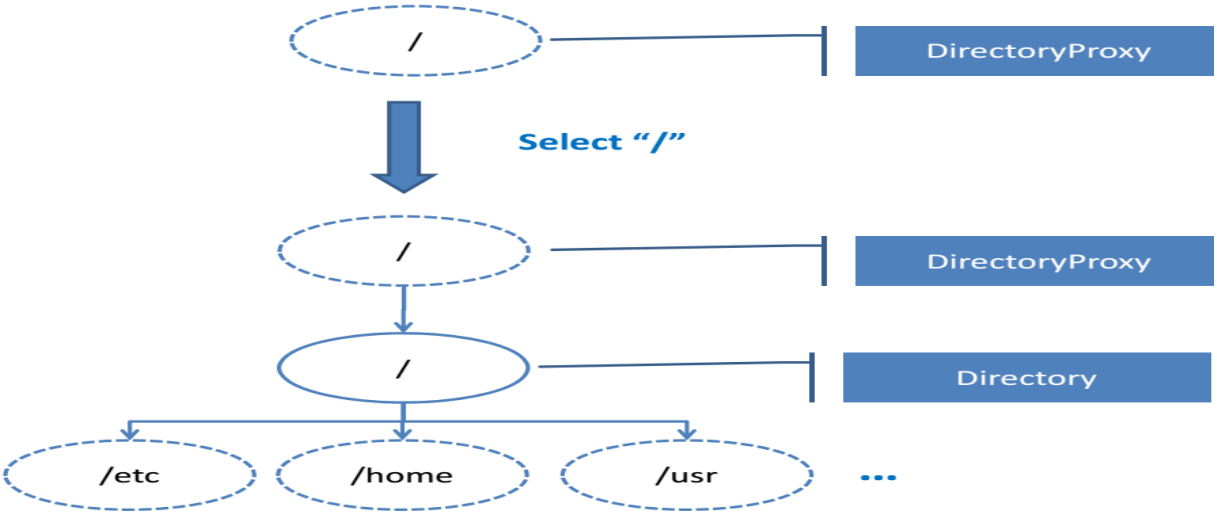


Figure 2 Example for proxy pattern

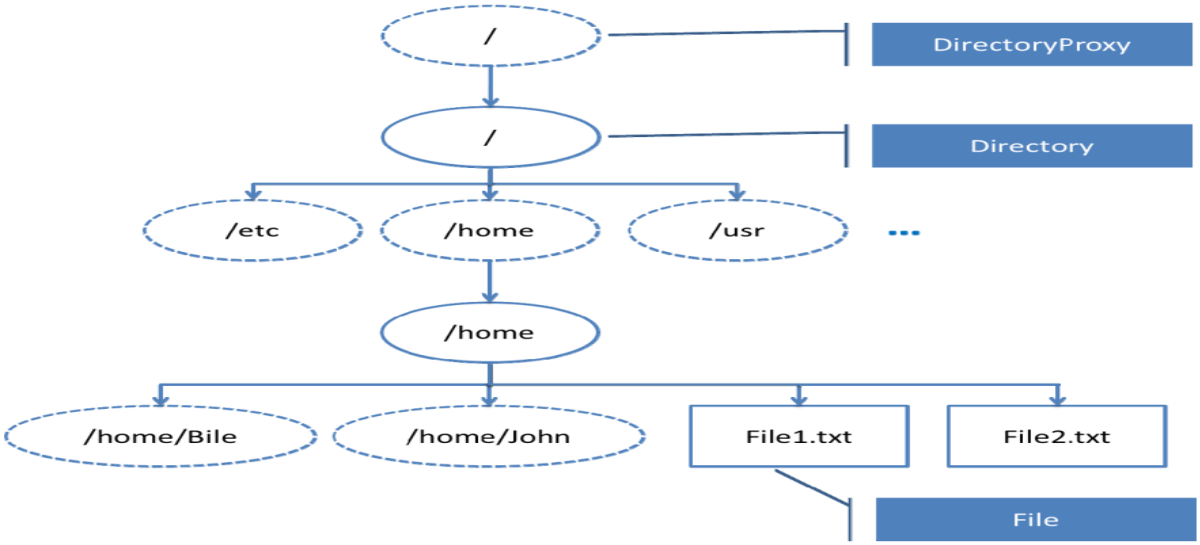


Figure 3 Example for proxy pattern

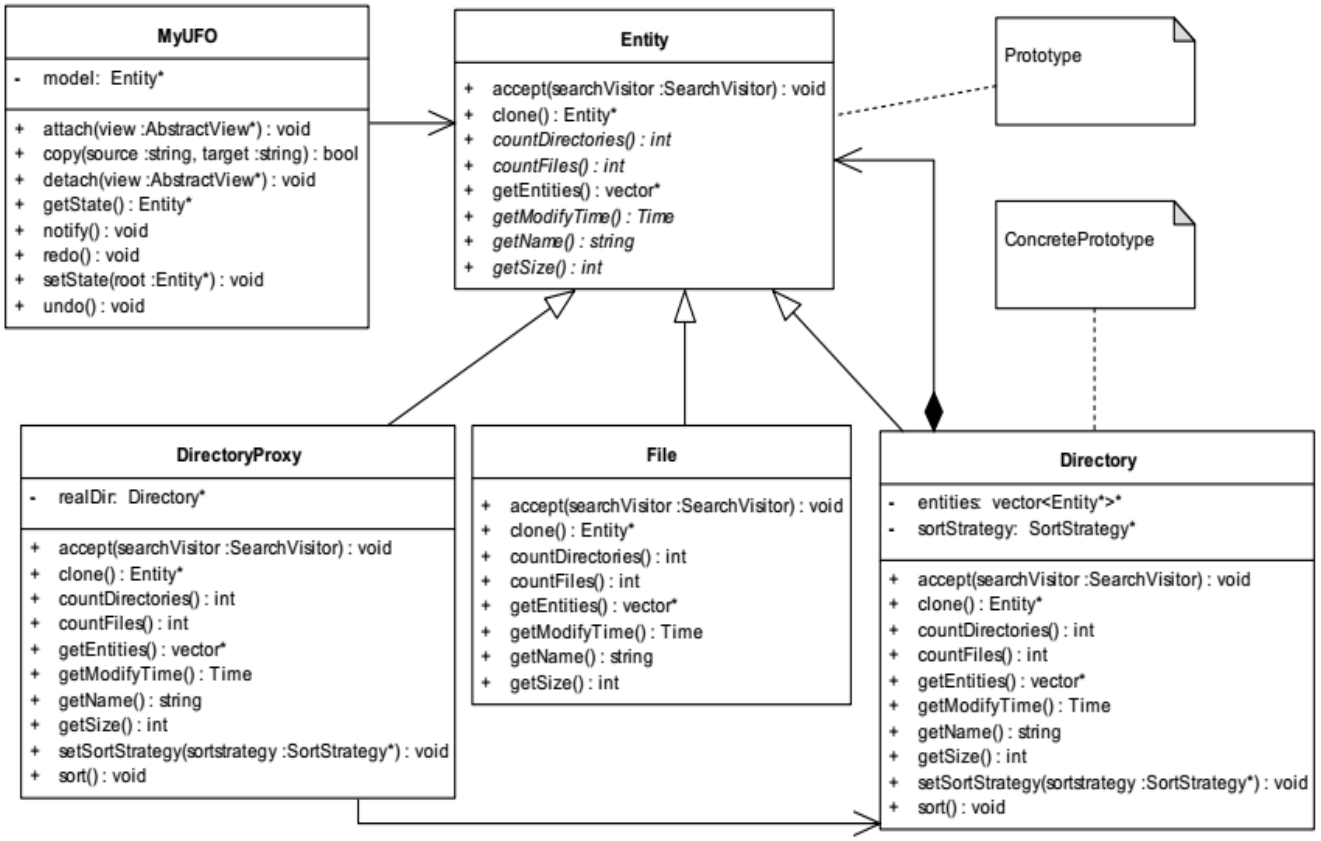
algorithmica
re-engineering it talent

Practical Object oriented design (File Explorer Application)

MileStone11: Enhancing copy and move commands

In this homework, we will enhance the copy and move commands to allow copying and moving directories. Processing a directory is more complicated than a file. MyUFO not only needs to copy the selected folder, but also copy the files and sub-folders under the selected folder. In addition, if the folder has a lot of different type entities, the status will become more complex than copy of file.

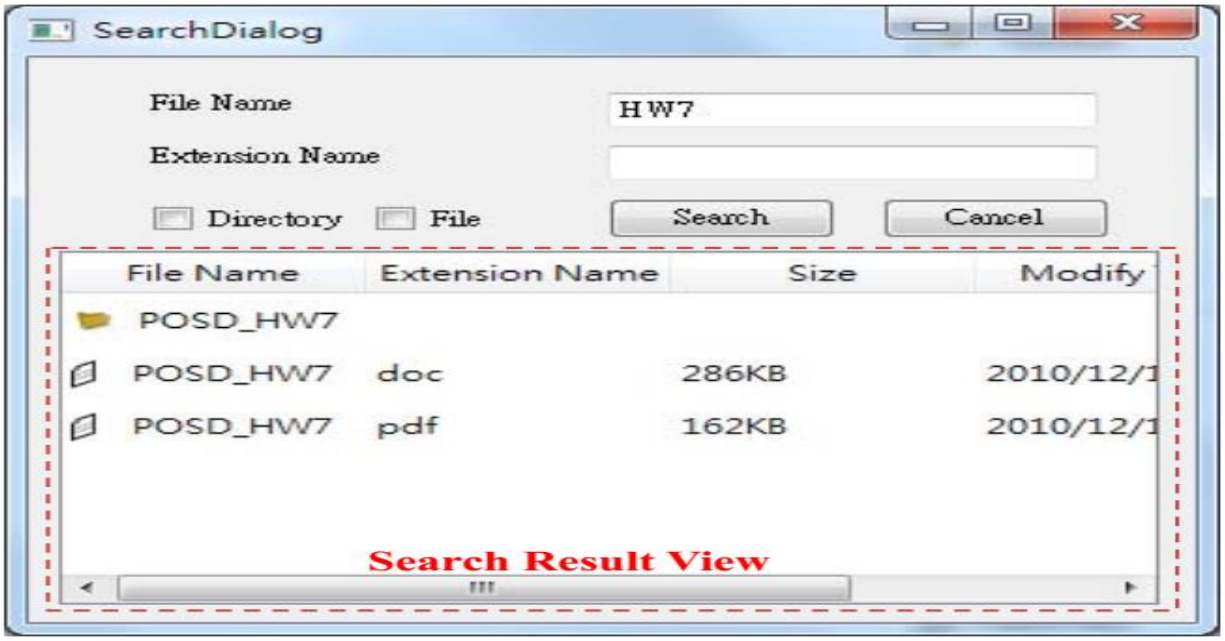
In this case, creating an instance of a directory (along with the children of the directory) is very complicated. Thus, we will apply **Prototype pattern** to resolve this problem. The following figure shows the class diagram after applying the Prototype pattern. Entity's subclasses implement the clone method so that they can be cloned. When a copy operation is requested, MyUFO uses the pointer of the selected entity to call its clone method to get a cloned instance of the entity, and then puts the cloned object to the target directory.



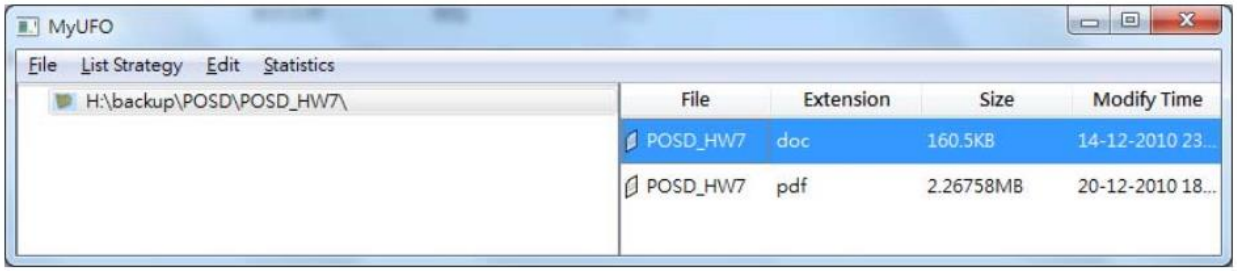
Practical Object oriented design
(File Explorer Application)

MileStone12: Enhancing copy and move commands

We would like to provide a search feature for MyUFO. The following figure shows the user interface of the search dialog, which allows the user to search for files and directories in the current directory.



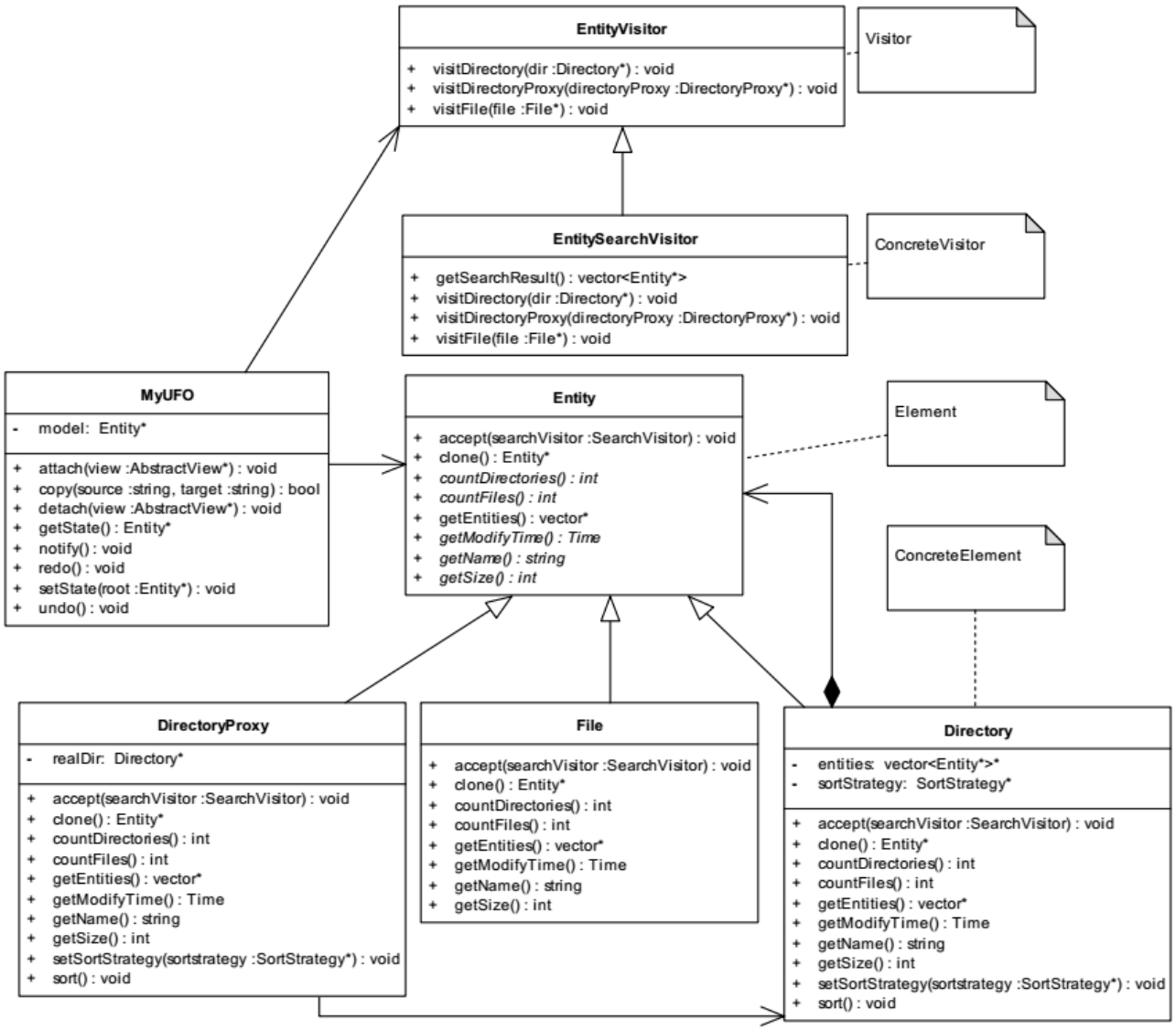
If a user wants to do a search, he/she can bring up the search dialog by clicking on the search menu item. Then, the user can type the entities information in the dialog, and selects the types (Directory or File) which he/she wishes to search. After the user clicks the search button, the search result will be displayed in the search result view. In addition, when a user double-clicks an entity in the search view, the corresponding entity in the File List View should be highlight. For example, if a user double-clicks the POSD_HW7.doc in above figure, MyUFO should automatically select and highlight the POSD_HW7.doc in the File List View.



You are required to apply **Visitor pattern** to implement the search feature. The following

Practical Object oriented design (File Explorer Application)

figure shows the class diagram after applying the Visitor pattern. Entity and its subclasses (Element and Concrete Element) add an accept method that takes a visitor as an argument. Two new classes EntityVisitor and EntitySearchVisitor must be implemented. When the user performs a search, MyUFO instantiates anEntitySearchVisitor, and calls Entity's accept method (taking the visitor as an argument). EntitySearchVisitor visits all Entities, and stores the search results. Finally, MyUFO gets the search results from EntitySearchVisitor, and displays the search result in the search result view.

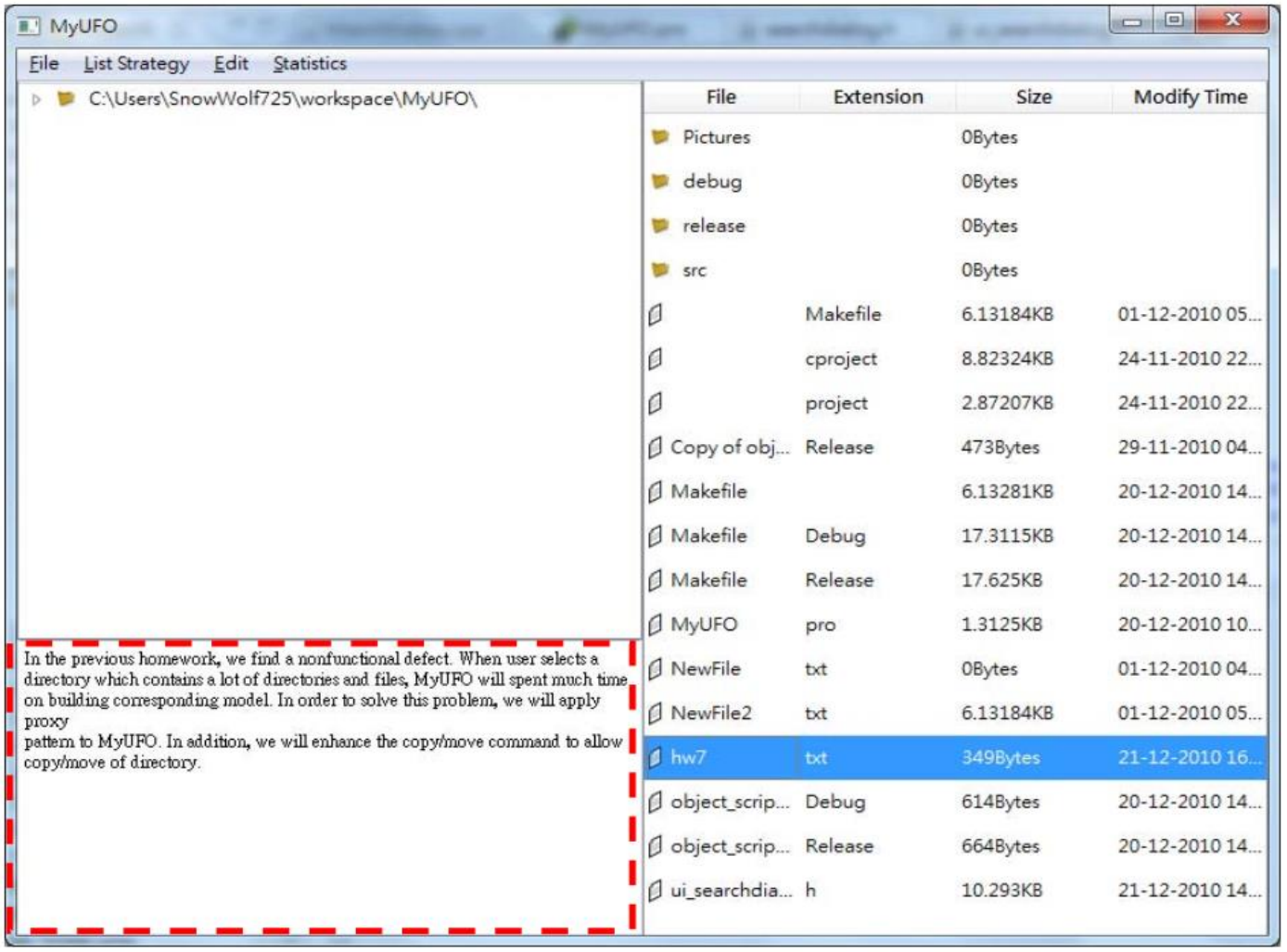


algorithmica
re-engineering it talent

Practical Object oriented design
(File Explorer Application)

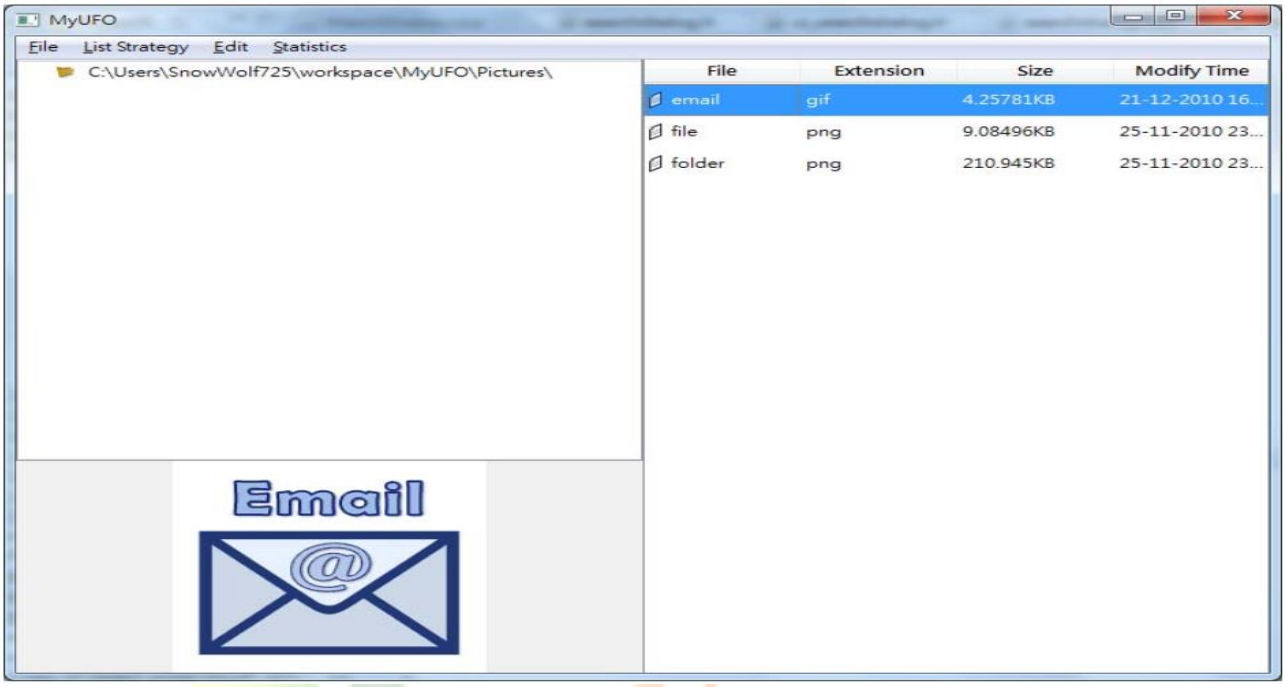
MileStone13: File Preview

File Preview is a common feature frequently seen in file managers. File Preview allows you to view the contents of file before you open the file. In order to make MyUFO easier to use, we will introduce this feature to MyUFO. We will add a preview widget at the bottom-left corner of MyUFO. When the user selects a file with "txt" extension, the contents of file should be show in preview widget like in the figure below.



Practical Object oriented design
(File Explorer Application)

If the user selects an image file (with “bmp”, “jpg”, “png”, or “gif” extension), miniature of the file should be shown in preview widget like in the following figure.



For simplicity, the other type of files should not display anything in the preview widget. If we put the creation process of Preview widget in the MainWindow, MainWindow must know the detail of creating the preview widget. In order to hide the detail of creatiion, we will apply **Simple Factory pattern** to create preview widget. In this case, PreviewCreator is a Factory that creates PreviewWidget (Product). The benefit is that MainWindow doesn’t need to know how the preview widget is created.

Practical Object oriented design
(File Explorer Application)

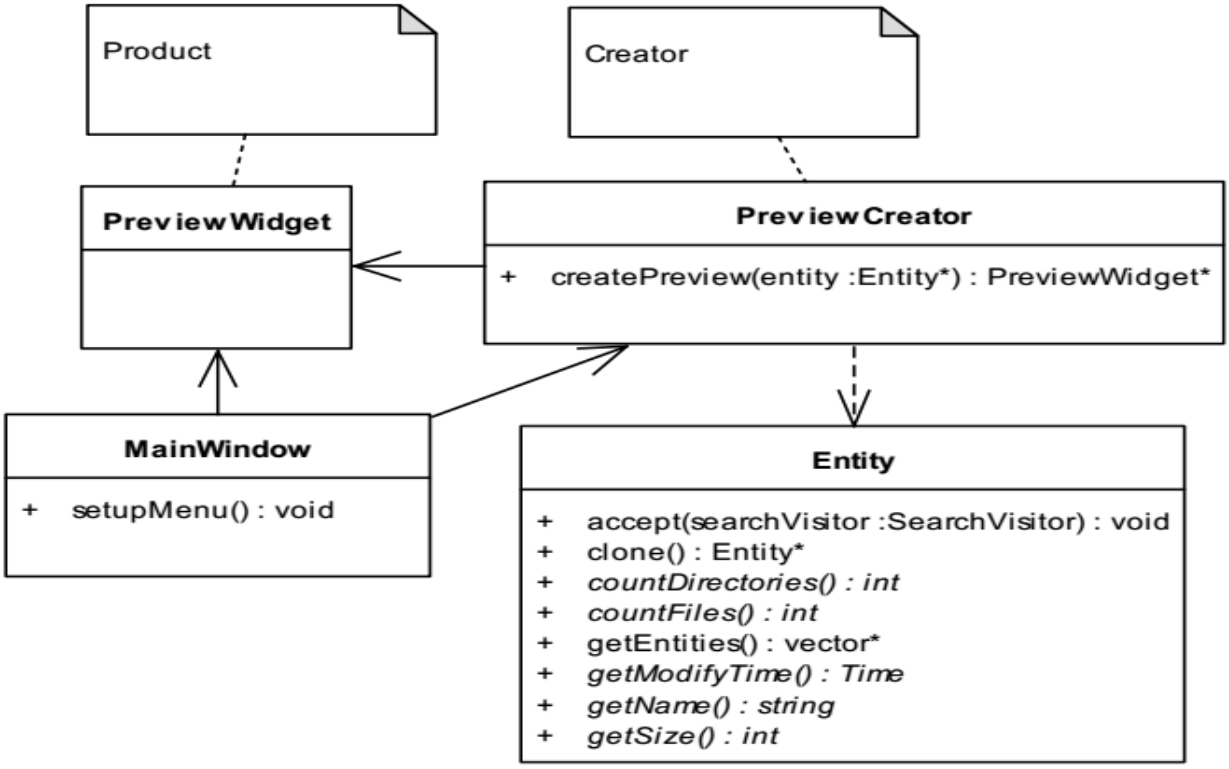


Figure 10 Simple Factory