CSSE 371: Lab 1-2

Background

Assume that you work for a company that specializes in the next generation Application Launcher. To launch an application, a user has to drag and drop a file to a dedicated folder monitored by the launcher and it will pick it up. Currently, your next generation launcher supports text editing (txt) and local web page browsing (html and htm).

The executives of your company decided that it's time to make the application more universal and extensible. They want you to re-design and implement the application to support the following features:

F1. The universal launcher should easily allow adding support for new kinds of App on top of the text and web page support.

F2. In addition to launching different apps, it should also allow addition of new handlers at runtime that can receive directory change notifications and perform custom actions, e.g., running background services, setting up application security rules, and so on. For our purposes, as a proof of concept, printing the name of the newly added files, or reading the content of the recently modified text file and printing all of its characters backwards are just fine to test this feature.

Your executives will assume that you have not forgotten any of the concepts discussed in CSSE 374 so far when you are re-designing and implementing the universal launcher.

Design

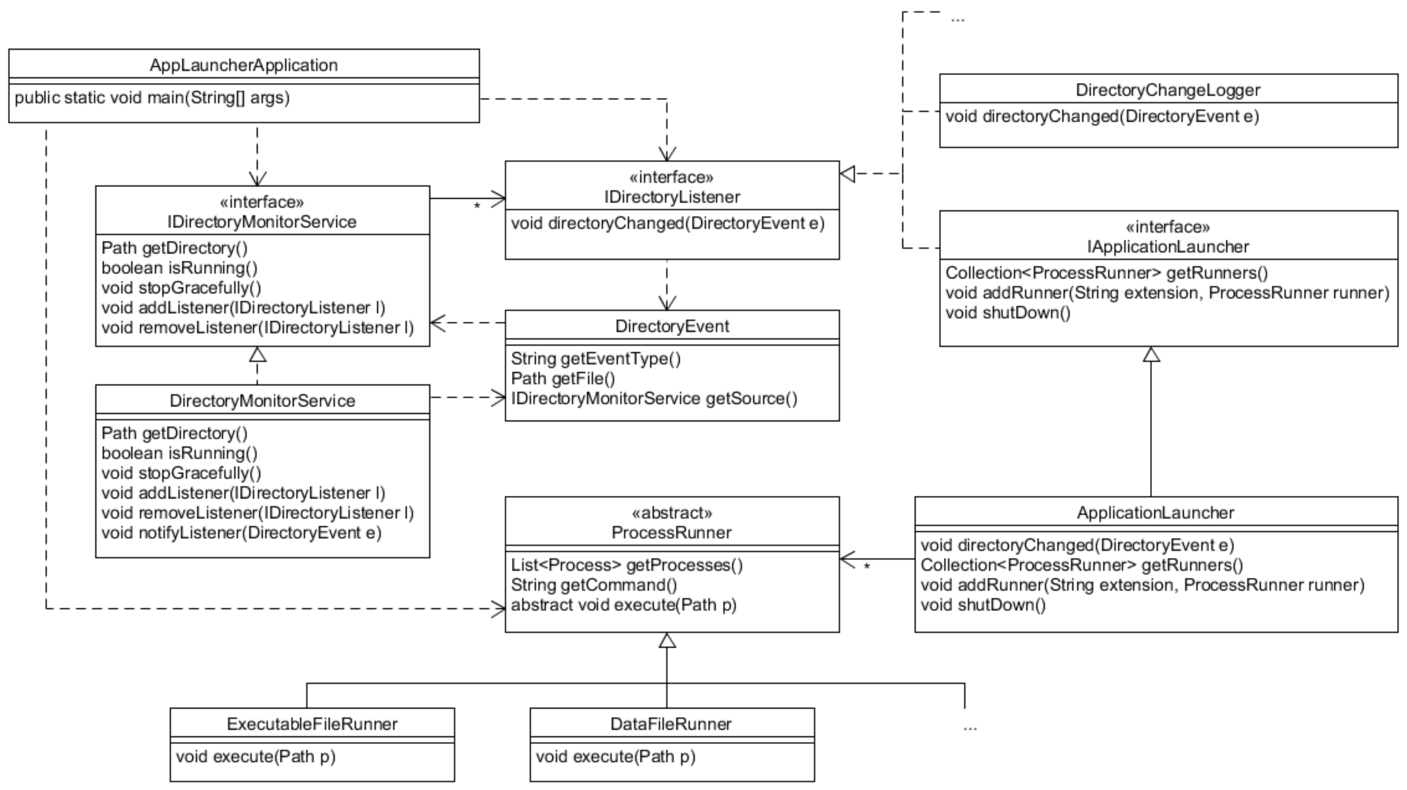
Create **Lab1-2/docs/Answer.pdf** with answers to the following problems:

Q1. Identify and explain problems prevalent in the existing design. [**5 points**]

The core logic has to be modified every time we add support for a new kind of application.

The code to handle directory notifications (*handleDirectoryEvent()*) is tightly coupled with the rest of the code and does not allow extension for new kind of handlers without modification of the core logic.

Q2. Create a UML Class Diagram to present your new design idea and explain it in a few lines. [**15 points**]


AppLauncherApplication
public static void main(String[] args)
«interface»
IDirectoryMonitorService
Path getDirectory()
boolean isRunning()
void stopGracefully()
void addListener(IDirectoryListener l)
void removeListener(IDirectoryListener l)
DirectoryMonitorService
Path getDirectory()
boolean isRunning()
void stopGracefully()
void addListener(IDirectoryListener l)
void removeListener(IDirectoryListener l)
void notifyListener(DirectoryEvent e)
«interface»
IDirectoryListener
void directoryChanged(DirectoryEvent e)
DirectoryEvent
String getEventType()
Path getFile()
IDirectoryMonitorService getSource()
DirectoryChangeLogger
void directoryChanged(DirectoryEvent e)
«interface»
IApplicationLauncher
Collection<ProcessRunner> getRunners()
void addRunner(String extension, ProcessRunner runner)
void shutDown()
«abstract»
ProcessRunner
List<Process> getProcesses()
String getCommand()
abstract void execute(Path p)
ApplicationLauncher
void directoryChanged(DirectoryEvent e)
Collection<ProcessRunner> getRunners()
void addRunner(String extension, ProcessRunner runner)
void shutDown()
ExecutableFileRunner
void execute(Path p)
DataFileRunner
void execute(Path p)
*
*
...
...

We use the Observer pattern for allowing new kind of handlers for directory change (*IDirectoryListener* is the Observer and *IDirectoryMonitorService* is the Subject). The *IApplicationLauncher* is itself one such observer that implements the Strategy pattern to allow running different application through the *ProcessRunner* hierarchy.

Implementation

Q4. Implement your new solution in the **Lab1-2/src/problem** package. For the proof of concept, make sure you have one extra example (on top of txt and html) for **F1** and at least two example handlers for **F2**. [**F1 – 10, F2 – 15 points**]

Testing

Q5: Implement necessary test cases in the **Lab1-2/test/problem** package that tests both **F1** and **F2**. [**10 points**]

[**Note**: The general unit testing convention is that you have at least one test class per concrete implementation class. A test class may have several unit testing methods that check boundary conditions of the methods in the concrete implementation. Example: If you have classes **A** and **B** that inherit interface **I**, then you should create **TestA** and **TestB** class in your test suite.]

Deliverable

Bundle you project in the **zip** format [**not rar**] and turn it in on Moodle.