Assignment 5: Printers

Suggestions and guidelines:

Some more suggestions on how you can structure the solution. The solution is expected to be on the following lines, though other (hopefully better!) solutions are welcome.

Long message, please read carefully.

1. We have already looked at the File hierarchy. The main aspect here is to define a File base class and appropriate derived classes for each kind of file (Audio, Text etc) , and define virtual methods for functions where we want sub-type specific behaviour. As described earlier, getting the print representation should be done this way.

2. The only place in the program (apart from the implementation of the derived classes of File) where you look at the specific type of file, should be when creating file objects based on reading the input. Here, obviously, you will be calling the appropriate constructor of derived class of File based on the string read from the input. Note that 1 and 2 ensure extensibility - we can keep adding new File types (ie. derived classes) by adding their definition and implementation, and will not have to make changes to the rest of the program, apart from the place where we construct derived types based on the input.

3. Folder's have a fairly simple function. They act as a container of files - so you can add any object of type File (ie. any of its derived types) to the Folder. Should not be aware of derived types of File. In addition, it would be useful to have a method that can get the File handle (or pointer) given the string name of the file.

4. Server: try to keep the server functionality simple - it should focus on getting print requests and dispatching it to the correct printer. And, if the printer is not available, it keeps it in a pending list until the printer is free. Decisions on whether a particular printer can print a particular file in a specified mode should be decided between the printer and file. That is, there should not be any code in the Server that checks for the type of file or printer or the print mode (BW/Colour). Note that this provides the right kind of encapsulation and extensibility. For example, we can add new types of printer or change the capabilities of a printer without affecting any other part of the solution.

5. Printer: Ideally, you should define a class hierarchy here, where the base class Printer handles the general requests and lets its derived classes **decide** if and how they will handle a file. Thus you can think of a method in the base class that confirms if a printer is available, and another that handles the print request.

a. The mechanism for managing availability of printers would be common to all printers, and this logic can be kept in the base class. The only thing that changes from printer to printer is the amount of time it takes to print a particular file (which would be a function of the printer and the file) and this information should be provided by the derived class to the base class. What would be the best design for this?

b. Handling a printer request (that is, a call to a printer to print a given file in a given mode) again has parts common to all printers, and parts that are printer+file instance specific. So, again, it is probably best to have some of the logic implemented in the base class, and have the derived class provide additional information if needed. See if virtual protected methods can be used here and in the earlier part.

6. Time and pending task management by the Server. This is the main task of the server, apart from maintaining the list of printers and the default printer. So, the Server needs to

maintain the "clock time" of the program. For this, you can use any mechanism that guarantees monotonicity of time. Simplest might be to use the <ctime> functions provided:

As an example, the following will give you the elapsed time since the "start" in secs.

```
time_t start;
start = time();

.... and after some computations

time_t now = time();
double seconds = difftime(now, start);
```

Since we are simulating time here and don't need exact time, you could return any multiple or fraction of the above whenever the server get a request for time.

As discussed, we want the server to manage pending tasks and the printers to focus on printing. So, when a server gets a print request, it should check if the printer is available, and if so, pass the request to the printer. The printer can decide if it can handle the request or now, and return a message appropriately. If the printer is not available, the Server should keep it in a list of pending tasks.
For simplicity, we will assume the Server uses a “polling” mechanism – it constantly scans the list of pending tasks, and checks if the appropriate printer is free, in which case it dispatches the job to the printer. This happens until all pending tasks are done.

7. To check how well your design handles polymorphism (and does not use explicit type checks), analyse the following for your code (**don’t actually do these changes in the submitted code). This is just for your experimentation**:

Assume we have a new kind of file to be added – say a Video file, and assume this file cannot be printed. To add this file type to the system, how many changes do you have to do to your code:

- How many of the existing files get modified?
- Did any of the existing Printer or File or Server code get modified?
- In each of these files, how many lines get modified?
- How many new files get created?
- How large are the new files?

Similarly, if we add a new printer type to the system (say a thermal printer), what changes would be needed, and which of the existing class implementations would get impacted, and to what extent?