Key Word In Context – Factory

You might have noticed in A3 (and if you did notice applaud yourself): we violated Single Responsibility Principle in MasterControl. We added logic. If this, then create that. Else, create that. Hey, it turns out there is a design pattern for this!

Design requirement:

1. Create an InputFactory responsible for creating the appropriate strategy, InputFromFile or InputFromConsole.
   Remember that since InputFactory must be able to create InputFromConsole, and since InputFromConsole requires the singletons, you need to pass them in through to the constructor of InputFactory.
2. Create an OutputFactory responsible for creating the appropriate strategy, OutputToFile or OutputToConsole.
   Remember that since OutputFactory must be able to create OutputToConsole, and since OutputToConsole requires a singleton, you need to pass it in through to the constructor of OutputFactory.
3. MasterControl should create both factories and simply pass in a parameter based on user input. There should be ***no*** logic/branches in MasterControl.
4. I want to see 2 different forms of factories, to demonstrate knowledge of both. The easiest form is to just put a switch/case or if/else into the create() method of the factory. It should not use an enum. InputFactory should use this paradigm.
   The more complicated form is to create an enum that is responsible for the creation. OutputFactory should use this paradigm. OutputType should be the enum used, with FILE and CONSOLE being the names of the values, so you can easily use OutputType.valueOf(string) to get the enum. The method should still accept a String, to demonstrate knowledge of decoupling the enum from the client code.

Guidelines:

MasterControl:

No more logic
Create one of each type of factory and call the create() method with the String choice.

InputFactory:

Constructor needs both singletons to be passed in
Throw an IllegalArgumentException if the choice is not FILE or CONSOLE.
Method:

```
public Input create(String choice)
```

OutputFactory:

Constructor needs system singleton to be passed in
Throw an IllegalArgumentException if the choice is not FILE or CONSOLE.
Method:

```
public Output create(String choice)
```

OutputType:

Method:

```
public abstract Output getInstance(SystemWrapper systemWrapper);
```

The only functional change to the output of the system is the exception thrown for incorrect user input.

Submission

Same as A1.