**Exercise 1: The factory**

**What is a factory?**

A factory is a creational design pattern that provides an interface for creating objects without specifying the exact class of the object that will be created. It encapsulates the object creation logic within a separate method or class, allowing the client code to create objects without being aware of the specific implementation details.

Here, the factory method is responsible for creating instances of parser objects based on the parser types defined in the "parserTypes.py" file. It abstracts away the creation logic and allows the client code to obtain a parser instance without knowing the specific implementation details of each parser.

**Does the implementation of the factory method follow the Interface Segregation Principle?**

Interface Segregation Principle: Clients should not be forced to depend on methods or interfaces they do not use.

Regarding the example:

HsmrPdfParser.py and HsmrTextParser.py inherit from the parent class HsmrParser.py, which defines a set of methods that the parsers should implement. It is a form of interface segregation at the class level, as the specific parsers implement the required methods.

But overall, the factory method does not directly depend on interfaces but rather returns instances of concrete HsmrPdfParser.py or HsmrTextParser.py.

**How to solve:**

In order to solve this problem, we can define an interface class for the parsers, and then the factory class would be depends on the interface directly, not depends on the txt or pdf classes.

**Exercise 2: Single responsibility:**

**Review the Python files starting with Ccs. Are those files adhering to the single-responsibility principle: "Every class should have only one responsibility”?**

CcsClassification.py: The class is responsible for handling the CCS classification information from a CSV file, including reading the file, managing the data frame, adding a PDF key column, and providing access to the classification data. This class has responsible for classification. **So it follows SRP.**

CcsHospitalDataExtracter.py: The class is responsible for fetching HSMR reports, reading the HSMR files, and creating plots based on the CCS data from hospitals. It has multiple responsibilities. **So It does NOT follow SRP**.

CcsHospitalInfo.py: The class is responsible for storing the CCS information of a single hospital, including the hospital type, code, name, place, and a data frame with CCS information. The responsibility of this class is to store information. **So, it follows the SRP**.

**Exercise 3: The base classes:**

**In the code, several base classes are used. Can you find examples of the**[**Liskov substitution principle**](https://en.wikipedia.org/wiki/Liskov_substitution_principle)**: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it." Explain your answer.**

The Liskov Substitution Principle states that objects of a child class should be able to replace objects of the parent class without affecting the correctness of the program. In other words, if we have a function that accepts an object of the parent class, it should also be able to accept objects of any child class without any issues.

Parent class: HsmrParser.py

Child classes: HsmrTextParser.py and HsmrPdfParser.py

both HsmrTextParser.py and HsmrPdfParser.py inherit from HsmrParser.py and provide their own implementation of the read\_file method, they can be substituted for objects of the base class. So, It follows the LSP.

**Exercise 4. The local settings object:**

**singleton object**: A singleton object is a design pattern that restricts the instantiation of a class to a single object. It ensures that only one instance of the class exists throughout the application, providing a global point of access to that instance. In other words, a singleton class allows you to create one and only one instance of the class.

Regarding the setting class: It ensures that only one instance of the class is created and provides a global point of access to that instance. The implementation of the singleton pattern can be observed in the class methods \_\_new\_\_ and \_\_init\_\_. The \_\_new\_\_ method is responsible for creating a single instance of the \_\_LocalSettings class, and the \_\_init\_\_ method ensures that the instance is assigned to the LocalSettings.instance attribute.

**Search for the Settings class. What makes this class a singleton object and is a singleton object SOLID?**

About the Settings class, it primarily focuses on the Single Responsibility Principle, which states that a class should have only one reason to change. The responsibility of the Settings class is to load and provide access to the settings defined in the local\_settings.json file. It encapsulates the logic related to reading the file, parsing the settings, and providing the values when requested. This separation of concerns aligns with the Single Responsibility Principle.

**The hospital types codes are stored in a python module hospital\_types.py. Is this a logical solution?**

Yes, it is. It promotes code organization and separation of concerns. It allows for easy access and management of hospital-type codes throughout the application. It helps avoid hard-coding the codes in multiple places and allows for modularity and maintainability.

## Next question is on the next page

**Exercise 5. UML Class diagram**

**\_\_LocalSettings**

Setting

Load\_setting()

Load\_setting()

Is\_setting\_loaded()

Get\_setting(setting\_key)

Get\_all\_settings()

\_\_str\_\_()

**LocalSetting**

c\_\_LocalSettings:class

\_\_new\_\_(cls)

\_\_init\_\_()

\_\_getattr\_\_()

**HsmrParser**

localSettings :class

file\_path:str

hospital\_type:str   hospital\_code:str

hospital\_name:str

hospital\_city:str

has\_file\_path()

get\_hospital\_name()

is\_umc()

has\_file\_content()

write\_content\_to\_text\_file()

extract\_ccs\_numbers()

read\_file

**ParserTypes**

HSMR\_REPORT\_PDF

HSMR\_REPORT\_TEXT

**HsmrParserFactory**

Get\_instance(parser\_type)

**HsmrTextParser**

read\_file()

**HsmrPdfParser**

read\_file()