DESIGN PATTERNS

- Creational:

1.FACTORY – By using the Factory Method pattern, we replace direct object construction calls using the new operator with calls to a special *factory* method. In our project, we have used factory design pattern in the case of creating different types of objects of classes which have as interface Person. This type of pattern is used as we can replace for example the class dermatologist with a doctor of another specialization. So one factory method is createPerson(). Moreover, we also can use a factory method prepareOrder() as they implement this method differently. The client knows that all person objects are supposed to have the prepareOrder() method, but for the client is not important how it works.

2.ABSTRACT FACTORY – By using the abstract factory design pattern, we can create Prescription interface, where the PDFPrescription and Eprescription implement this interface. As they are related to each other with slight differences, this design pattern is applicable.

3.PROTOTYPE – As the prescription contains the same information for both PDF and the virtual form, it is needed to make an interface class called prescription. In this interface, it is found the clone() method that will be used by the inherited classes to make the prototypes.

4.SINGLETON – It is a creational design pattern that lets you make sure that a class has only one instance, and at the same time, we are providing a global access point to this instance. In this program, we need to create the diary for a patient only once. So, the design pattern singleton is used in the case of the diary class. Once a patient is created, the diary object is created only once. In the diary class, we declare the constructor as private and we declare a method createDiary() which creates the object diary only once.

5.BUILDER – It is a pattern that is used to create an object step by step. In this software, the order is created step by step because it is firstly proposed by the dermatologist, then the pharmacist selects the medications and ask the patient if he/she wants to make the order. Then, they need to send the contact and address, make the payment and then the order is finished.

- Behavioral:

6.OBSERVER –Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing. In this software, the dermatologist subscribes the diary of the patient. He/she doesn't need to make continuous checking if the patient hasn't made changes.

7.STATE: In the case of state pattern, there are present the different behaviors that user shows when his/her states changes. For instance, if the normal user becomes a patient, he/she will have different behaviors from a normal user, or from a previous patient who is healed.

- Structural:

8.ADAPTER – This design pattern improves the quality of this application. As the images scanned should be of high quality to have a proper diagnosis from the dermatologist, we have to use an adapter which make the conversion of the image from any type to png as it is a lossless format with the highest quality. After it is converted, it is used in the form.

9.BRIDGE – By using this design pattern, we have split the image class into subclasses (child classes) which are the types of images: jpg, png, tiff. This design pattern improves the functionalities of the image class.

10.DECORATOR - It is a structural design pattern gives the opportunity to new behaviors to objects by placing these objects inside the wrapper objects that contain the behaviors. In this project, it can be used in the case where the order needs to be transform into a bill and the main difference that they have is the fact that bill is printable, so to solve this we create bill decorator which has the methods to format the order in bill format and print().