Design Patterns

1. Façade Pattern – Structural Design Pattern

Found at at /src/main/java/org/jabref/model/metadata/MetaData.java

The MetaData class holds in itself objects from multiple other classes. Many of those objects qualify as metadata and together they form a highly complex subsystem with many functionalities. A few examples of metadata objects can be seen below:







Access to certain functionalities of these objects, as well as access to the objects themselves is then granted by several methods in the MetaData class. An example of this can be seen in the following methods:

-Access to objects:

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

-Access to certain functionalities of the objects:

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

The MetaData class works as a façade class which wraps metadata representing objects, providing a simple communication link between the rest of the system and the wrapped classes, as well as simplifying the interaction with a subsystem that is comprised of several moving parts. The façade class is then used in multiple places throughout the code, replacing what would have to be a systematic instantiation of all of the “encapsulated” classes.

1. Template Method – Behavioral Design Pattern

Found at /src/main/java/org/jabref/model/groups/AbstractGroup.java

The template method design pattern allows a superclass to define the structure of a method while delegating the specifics of the implementation to subclasses. This results in a public method in the superclass that, although it might include additional logic and even other non-abstract methods, will include at least one abstract method.

The example found in the AbstractGroup class is a rather simplified implementation of this. The isMatch() method uses the abstract method contains() - which is implemented independently by each AbstractGroup subclass - and works as a template method.

Texto

Descrição gerada automaticamente

Other implementations of this design pattern can have more complex template methods, with multiple abstract methods within them as well as some logic.

1. Prototype – Creational Design Pattern

Found at /src/main/java/org/jabref/model/entry/BibEntry.java

The prototype design pattern allows the cloning of specific objects while still making the resulting object independent of its origin. This is implemented through a cloning method inside the class to be cloned, which returns a new object based on its own fields. Classes that allow cloning of their objects should share a common interface for this purpose.

The BibEntry class implements the *Cloneable* interface and overrides the clone() method, thus being a prototype class.

As seen below, the BibEntry class returns an object that is similar to the original one in every field of the class, with the exception of the id, which is generated when the new clone object is created.

Texto

Descrição gerada automaticamente

Gabriela Costa 58625