**DESIGN PATTERNS**

**Illustrating code snippet**

**Text

Description automatically generated**

**Exact location on the codebase**

****

**Design Pattern**

The pattern detected in the JabRefPreferences class is the **Singleton Pattern.** This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. This pattern involves a single class that is responsible to create an object while making sure that only a single object gets created.

**An explanation of the rationale for identifying this as a pattern instantiation**

In this method, a new object is created. This class has only one instance and provides a global point to access it. This variable makes an object accessible, but it pollutes the namespace, and it still allows instantiating multiple objects of that class. So, the solution could be to make the class itself responsible for keeping track of its sole instance, which means, this class could ensure no other instance is created, by intercepting requests to create new objects or even the class could provide the sole way to access the instance

**Illustrating code snippet**

**Text

Description automatically generated**

**Exact location on the codebase**

****

**Design Pattern**

This pattern encapsulates a request as an object of its own, so the pattern detected is the **Command Pattern.** In this case, the sender object can create a command object to encapsulate the order. The answer is the invoker object that invokes the command objects to complete whatever task it is supposed to do. The sender in this case is this method inside the class that creates a command (called applicationCommand) in which calls a method which is the receiver (get).

**An explanation of the rationale for identifying this as a pattern instantiation**

The command pattern stores different requests (command objects - applicationCommand) into lists and manipulates them before they are completed. This pattern allows, as well, the command to be undone or redone. This pattern has various Benefits such as allowing commands to be manipulated as objects, its functionalities can be added to the command objects, such as putting them into queues and decoupling the objects of the software program.

**Illustrating code snippet**

**Text

Description automatically generated**

**Exact location on the codebase**

****

**Design Pattern**

This pattern is a behavioural design pattern that lets us define a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing, so the pattern detected is the **Observer Pattern.** The observer pattern is used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under the behavioural pattern category.

**An explanation of the rationale for identifying this as a pattern instantiation**

The StateManager class uses the ObservableList (a list of tasks – which can be assigned to either isRunning or to getProgress), in order to observe objects of the type of the task, creating an array Observable which keeps the tasks in progress and the tasks running. The observer design pattern is also suitable for any scenario that requires push-based notification. The pattern defines a provider (also known as a subject or an observable) and zero, one, or more observers (depending on the length of the array). In short, observers register with the provider, and whenever the task state change occurs, the provider automatically notifies all observers by calling one of their methods.