Pedro Inácio 59184

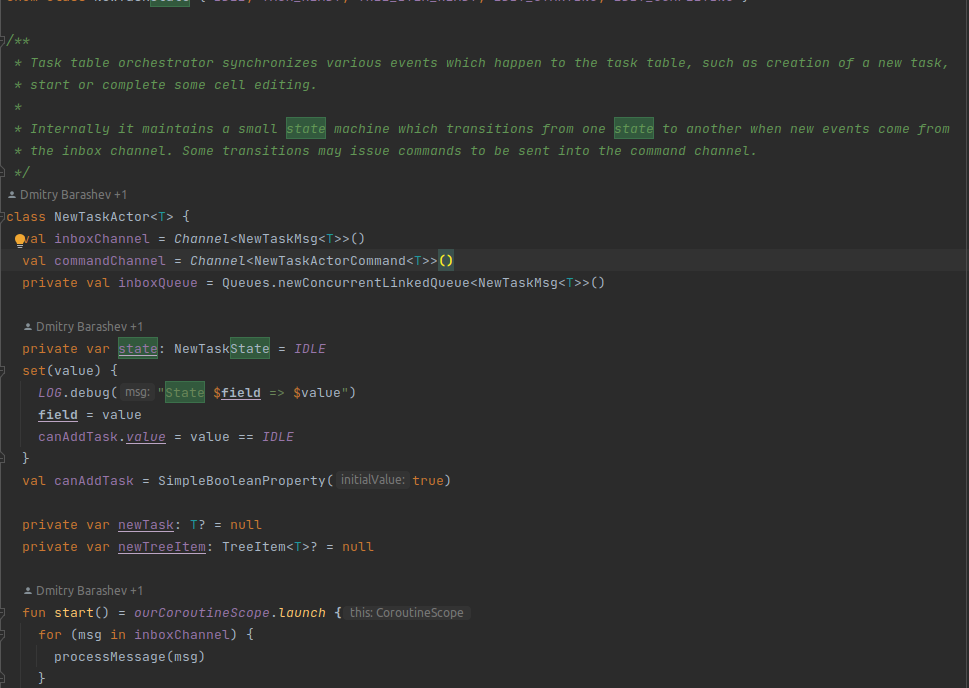
Design patterns

# State pattern

A state pattern is present when the class depends on a series of states to decide how to act facing a situation.

This design pattern is used in package ganttview, with the goal of maintaining “context” in the actor of its current state. In fact this pattern here is so clear that the comment on top of the class NewTaskActor says it all:

*/\*\**  
 *\* Task table orchestrator synchronizes various events which happen to the task table, such as creation of a new task,*  
 *\* start or complete some cell editing.*  
 *\**  
 *\* Internally it maintains a small state machine which transitions from one state to another when new events come from*  
 *\* the inbox channel. Some transitions may issue commands to be sent into the command channel.*  
 *\*/*

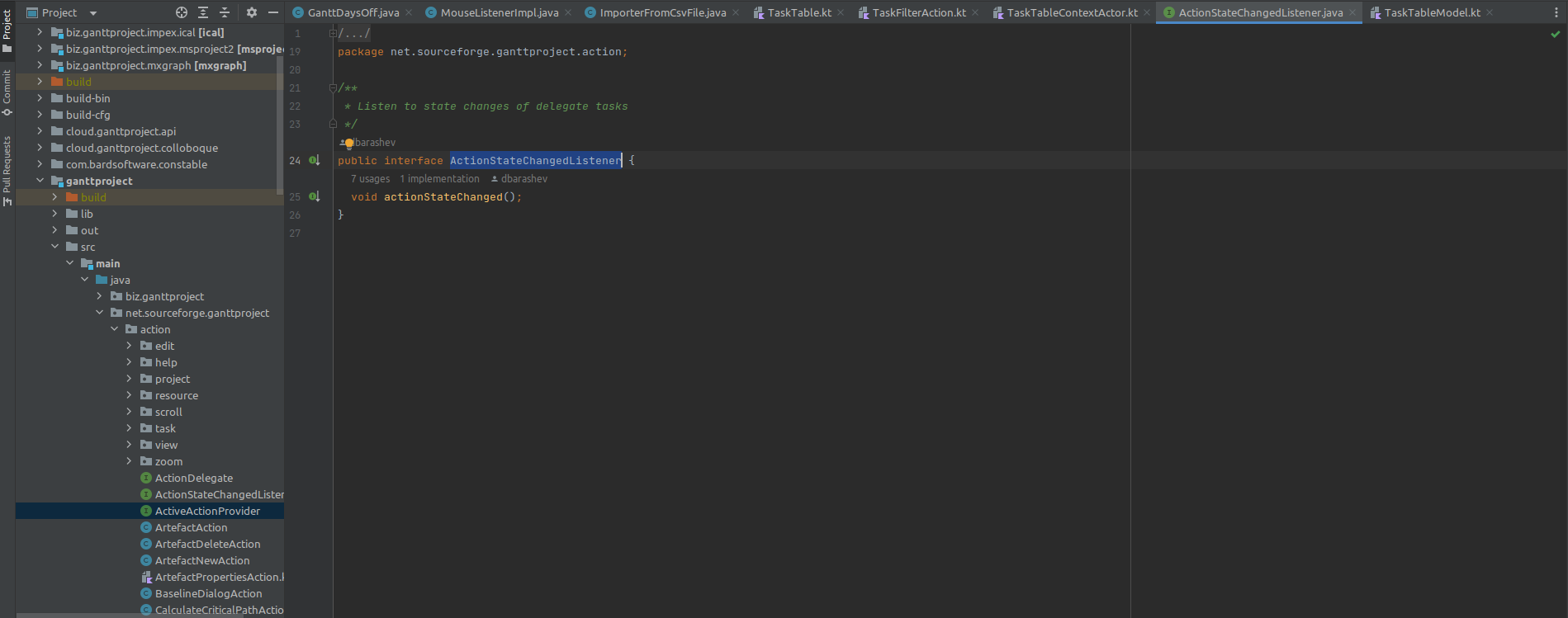


# Observer pattern

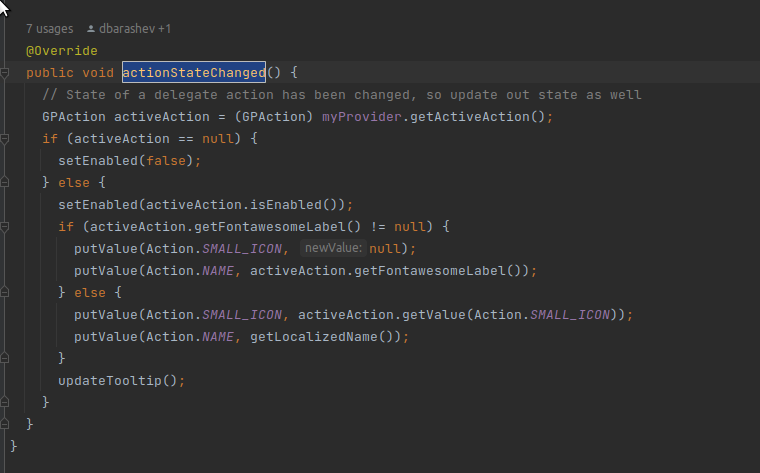
The Observer design pattern is a pattern where a subject keeps a list of observers and notifies them of any state changes, usually by calling one of their methods. The observers rely on the subject to inform them of changes in the subject's state.

In the interface ActionStateChangedListener its clear that any class that implements it is an observer for when a state of something changes.

*/\*\**  
 *\* Listen to state changes of delegate tasks*  
 *\*/*



Usage example in ArtefactAction:



# Command Pattern

The command pattern is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time.

In the class TaskTable.kt the method initNewTaskActor checks for external commands to know what to do.

// Launches a coroutine which is reading command messages from the New Task Actor and  
// starts or commits tree cell edits.

