Vorgehensweise und Entscheidungsfindung

Zunächst hatten wir ein Problem mit dem Verständnis und der Umsetzung der Aufgabenstellung. Wir haben versucht möglichst nahe an unserem alten Modell zu bleiben.

Wir haben uns dann überlegt, die Klasse Elevator in ElevatorJobs zu refactoren. Vorteil wäre gewesen, wir hätten nicht z.B. drei Threads die in einer Endlosschleife (while(true)) als Fahrstuhl durch die Gegend fahren, sondern mittels der Queue im Executor die Möglichkeit, den Executor die Jobs auf die Fahrstühle zu verteilen. Wir würden also immer wenn eine neue Aktion kommt (FS wird gerufen nach…, FS soll fahren nach…, Türen offen halten), diese als Runable in den Executor werfen.

Der Nachteil ist aber, dass wir nicht mehr wissen dann, auf welchem Fahrstuhl der Job letztlich ausgeführt wird. Wollen wir also die Türen beispielsweise offen halten, wissen wir nicht mehr, welcher Thread das ist. Wir können uns zwar die Queue des Exectors holen, aber nützlich war dies nicht.

Die nächste Überlegung war dann, doch wieder den Executor nur die Fahrstühle in der Endlosschleife fahren zu lassen. Damit ist das Konzept des Executors und der Queue zwar nicht wirklich brauchbar, aber wir haben nun wieder die Möglichkeit, auf unsere Threads brauchbar zuzugreifen.

Für diesen Weg haben wir uns dann auch entschieden. Weiterhin sind wir durch die Aufgabenstellung a04 in AD auf Deques aufmerksam geworden ☺ Hier gibt es ja das LinkedBlockingDeque. Durch den Einsatz dieser Struktur konnten wir uns einiges an Arbeit sparen und ein paar Codeschnipsel aus unserem Entwurf in a06 vereinfachen. Wir brauchen beispielsweise die goTo() Methode nicht mehr synchronisieren. Praktischerweise nimmt uns das Deque diese ganze Arbeit bereits ab. Das locked unsere Queue beim hinzufügen von Jobs und beim abarbeiten. Sehr praktische Sache ☺

Für die Vorrangfahrt ist der erste Fahrstuhl reserviert, ähnlich wie in der Uni. Wird in einem Stock der Vorrangknopf gedrückt, wird die Queue des Fahrstuhls gecleared und der Vorrang Job hinzugefügt.

Eine Taste zum Offenhalten der Türen gibt es auch einmal für jeden Fahrstuhl. Die maximale BlockingSize ist als globale Variable einstellbar, steht derzeit auf 5 Sekunden. Wenn man den Knopf gedrückt hält, wird die Tür damit offen gehalten, lässt man los, fährt der Fahrstuhl los. Hält man den Knopf länger als 5 Sekunden gedrückt, fährt er dennoch los.

Für den letzten Aufgabenteil haben wir einen Knopf erstellt, über den Passagiere in den Fahrstuhl einsteigen können. Hat man diesen Knopf gedrückt, erscheint ein JDialog zur Eingabe des Wunschstockwerkes. Ja wir geben zu, wir haben keine so „schönen“ Animationen gebaut, wir haben uns wieder mehr auf eine auf technischer Basis komplette und möglichst genaue Umsetzung gestürzt ☹

Für unsere block-Methoden zum Offenhalten der Türen hatten wir ursprünglich einen Semaphor, bzw. eher einen blockMutex verwendet, ein synchronized auf sich selbst mit this reichte aber aus.