Laboration – Stock Watcher

DENNA LABORATION SKALL LÖSAS INDIVIDUELLT OCH DU SKALL HA FÖRSTÅELSE FÖR DE OLIKA DELARNA I DIN IMPLEMENTATION VID INLÄMNING.
DET ÄR EJ TILLÅTET ATT KOPIERA EN ANNANS STUDENTS LÖSNING FÖR AKTUELLT PROJEKT.
DET ÄR EJ TILLÅTET ATT KOPIERA EN LÖSNING IFRÅN INTERNET.
DET ÄR EJ TILLÅTET ATT LÅTA ANNAN PERSON ELLER AI LÖSA UPPGIFTEN ÅT DIG.
DU KAN BLI OMBED ATT REDOVISA DIN LÖSNING MUNTLIGT. BAKGRUND

Företaget LAN-INC har utvecklat en programvara som kan ge information om priset på de aktier som företaget bevakar i realtid.

Implementationen är nästan helt klar, men i slutskedet av utvecklingen så hoppade företages enda utvecklare *BumliDev54* av projektet.

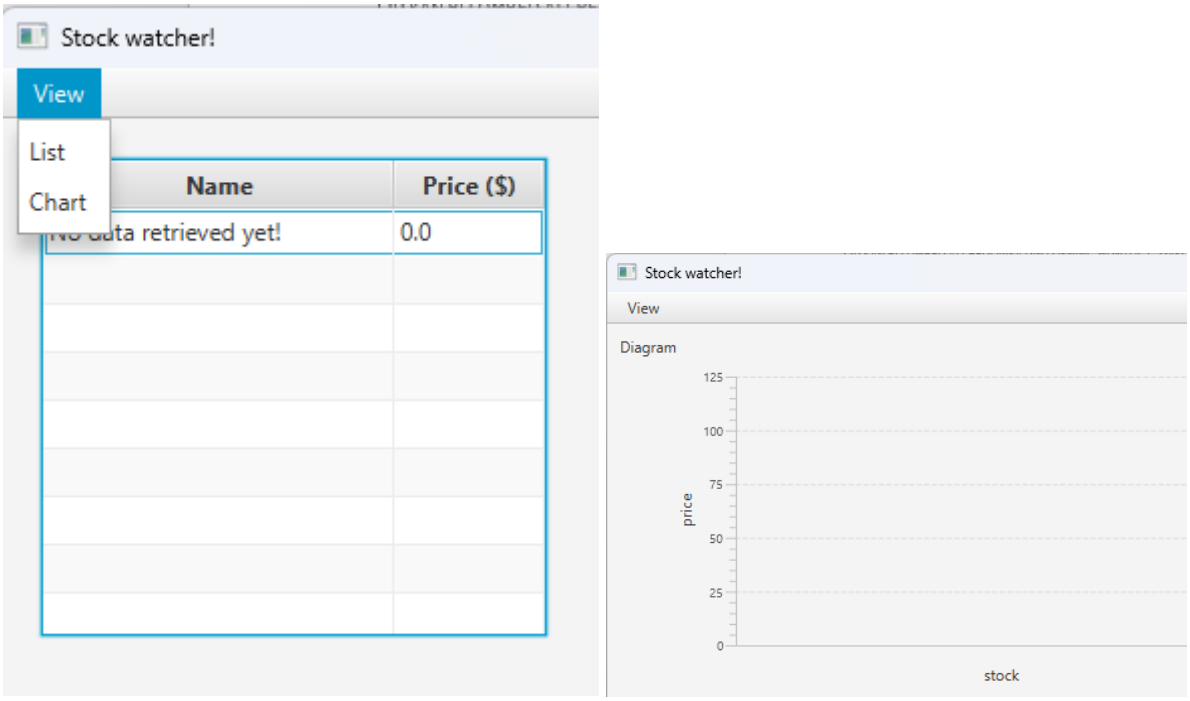
LAN-INC behöver nu din hjälp för att få sista pusselbiten på plats innan publicering av applikationen kan ske.

Projektet

Kod att utgå ifrån hittas i anslutning till denna fil.

Att göra

När du startar applikationen så möts du av följande:



En enkel applikation med två stycken vyer, ”List” och ”Chart”, du kan växla mellan dessa vyer i menyn under valet ”View”. Det hittas dock ingen data i vyerna (*namn och pris på aktier*). **Det är här du kommer in, din uppgift är att lösa så vi ser aktierna och deras pris i vyerna. Detta skall göras med hjälp av designmönstret Observer (Observer pattern)*.**

Observera att all logik för att hämta information om aktier och dylikt redan finns på plats. Det som saknas är kommunikation emellan klasserna (Observer Pattern).

Konkret:

- Få önskat resultat i applikationen genom att implementera Observer pattern*.
- Bifoga i din inlämning en .pdf som besvarar följande fråga:
  1. Följer din implementation av Observer pattern **pull** eller **push** metodiken för mönstret?

** Börja med att identifiera vilken klass som bör vara Subjekt-klassen och vilka klasser som blir observers (subscribers).*

Du kommer troligen behöva skapa och lägga till klasser och/eller interfaces för att lösa uppgiften. Du är fri att göra vilka ändringar du vill i projektet för att lösa uppgiften.

OBS Javas ”native” lösning för observer (på klassnivå eller annan färdig lösning som till exempel ObservableList) får ej nyttjas, du behöver alltså skapa samtliga pusselbitar i observer mönstret själv.

Förberedelser

- Föreläsningar och laborationer.
- Kursbok, kap 1 och 2 *(kap 5 förekommer i lösningen sedan tidigare)*.

Inlämning

Du laddar upp en .zip innehållandes:

1. Koden för en fungerande lösning av applikationen där Observer pattern är implementerat.
2. En .pdf med efterfrågat innehåll.

Betyg

Inlämningen kan ge något av betygen underkänd (U)* eller godkänd (G).

**Vid betyget U (underkänd) kan du antingen ges möjlighet att komplettera din lösning upp till betyget G (godkänd) alternativt så behöver du genomföra en helt ny uppgift.*

Betygskriterier

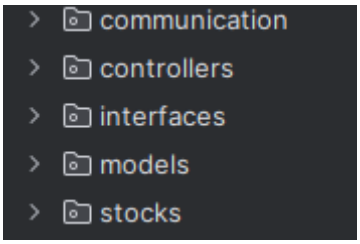
När vi betygsätter inlämningen tittar vi på om:

- Efterfrågat mönster finns implementerat.
- Frågan besvaras korrekt givet din/er implementation.
- Kvalitet på lösningen.
- Funktionaliteten i applikationen.

Nuvarande projekt

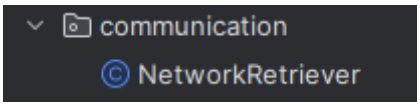
Genomgång av klasserna i projektet:

I applikationen återfinns fem moduler (paket/packages):

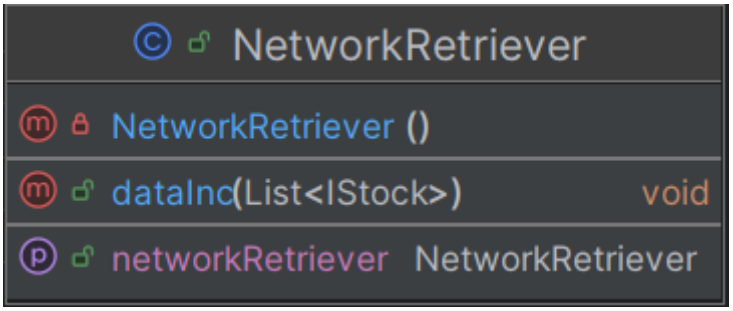


I detta dokument gås valda delar av controllers samt communication paketen igenom. Övriga paket bör vara självförklarande men har du några frågor så kontakta aktuell lärare.

Communication paketet:

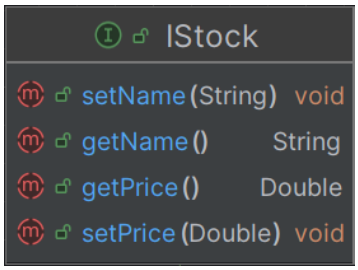


Communication paketet innehåller endast en klass, **NetworkRetriver**:



Tre aspekter att beakta gällande NetworkRetriver:

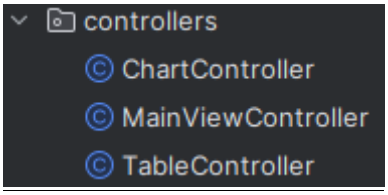
1. I klassen så hittas metoden dataInc. Denna metod anropas varje gång ”StockWatcher” har fått information om aktier och pris. Denna information anländer som en ArrayList innehållandes objekt som följer IStock interfacet.



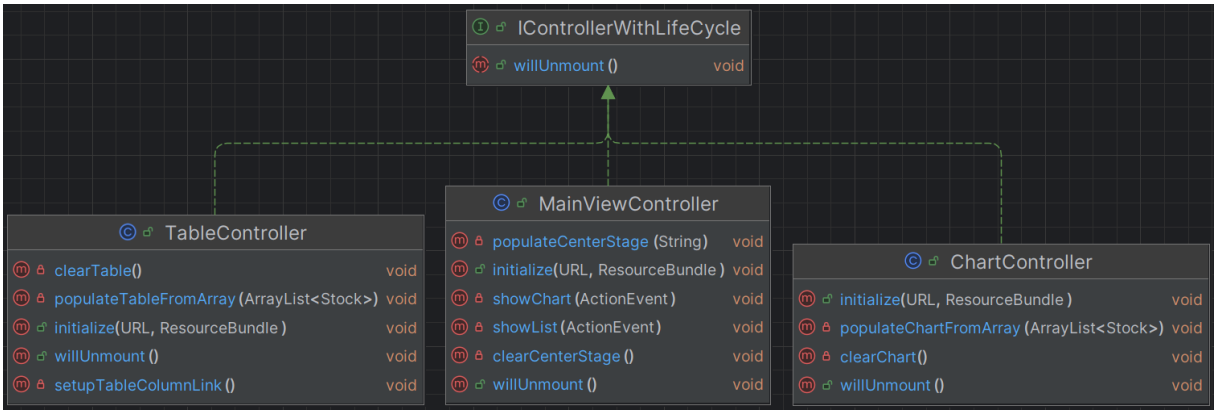
2. Klassen är för närvarande byggd som ett Singleton Pattern (SP). Du är fri att nyttja denna implementation eller bygga om det till något annat för att passa din observer lösning. *Denna examination handlar om att implementera Observer, inte nödvändigtvis att klasser skapas / nås på sätt som helt förhåller sig till SOLID (något vi återkommer till senare i kursen).*

3. I konstruktorn så instansieras StockWatcher klassen, detta behöver vara kvar för att möjliggöra hämtning av aktier och priser.

Controllers paketet:



Innehåller tre klasser:



Controllers paketet innehållet klasser med logik kopplat till varje vy (det användaren ser). Vyerna är indelade i avsnitt där varje avsnitt har sin egen kontrollerklass.

MainWindowController:
Huvudfönstret som innehåller de andra vyerna. Metoden populateCenterStage utför det centrala arbetet för klassen, nämligen att skifta vilken vy som syns (*List eller Chart*). Metoden får in en sträng som representerar vyn som skall visas, JavaFX matchar denna sträng emot en FXML fil som laddas tillsammans med respektive kontroller klass (*Table- eller ChartController*).

Generellt:
Controller-klasserna exekverar metoden **initialize** när den blir synlig samt metoden **willUnregister** innan den försvinner ifrån användarens synfält.

På kutssidan så hittas även en inspelning som presenterar hur JavaFX fungerar (vyer och kontroller klasser) som kan vara bra att se innan du arbetar med uppgiften. Observera att fokuset är inte på ”korrekta” JavaFX lösningar utan att implementera efterfrågat mönster i projektet.