Laboration - Raining Calculator

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.
Du kan bli ombed att redovisa din lösning muntligen.

Bakgrund

Företaget Raining-INC har utvecklat en ny programvara som är tänkt att hjälpa användaren att räkna fram hur stor chans det är för regn under de kommande dagarna.

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

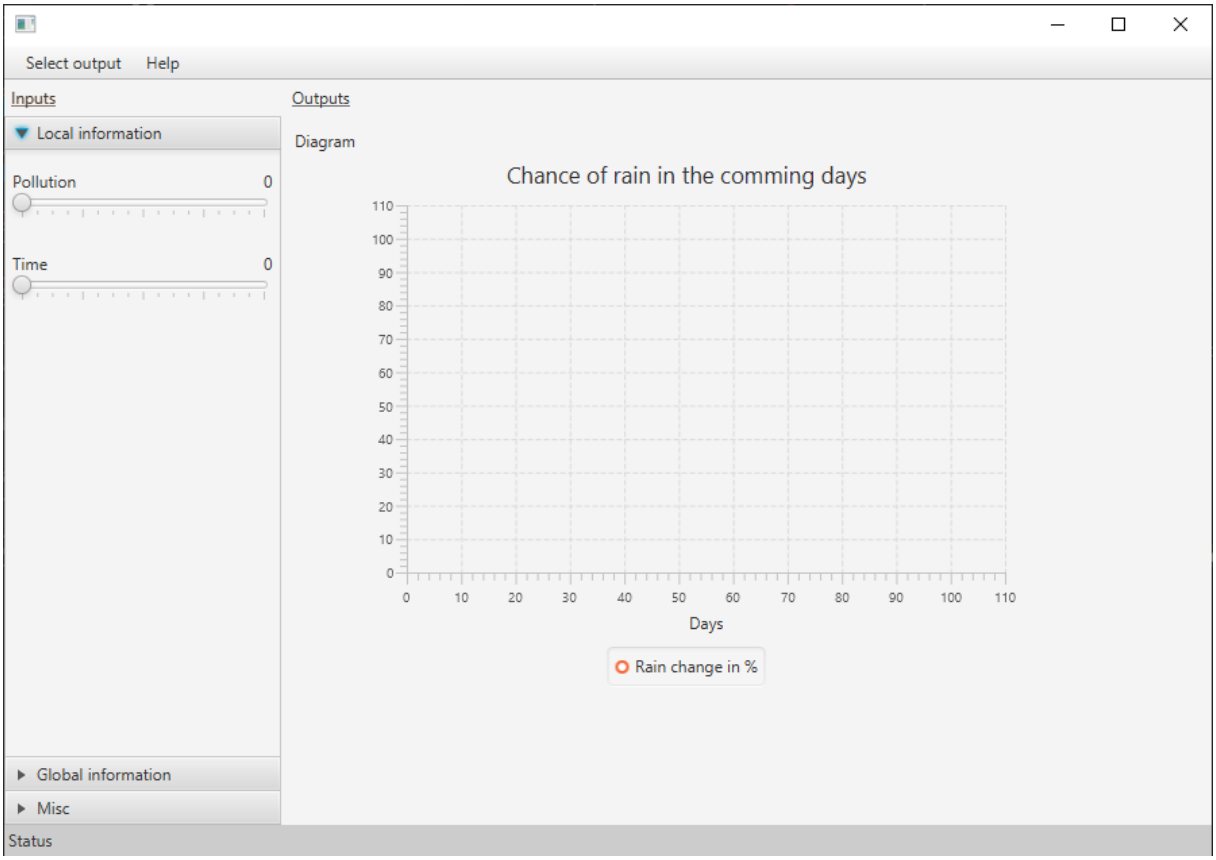
Raining-INC behöver nu din hjälp för att få sista pusselbiten på plats innan publicering av applikationen kan ske.

Projektet

Kod att utgå ifrån hittas i anslutning till denna fil.

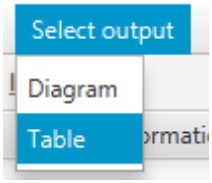
Att göra

När du startar applikationen så möts du av följande:



I vänstra delen av applikationen finns det möjlighet att ange input till programmet (*ex: Pollution och Time*).

I högra delen av programmet visas chansen för regn de kommande dagarna. När du drar i inputs-sliders till vänster så uppdateras inte grafen till höger. Du kan dock se uppdateringen av värden ifall du går upp i huvudmenyn och byter output-vy:



Du kommer då se ett värde i table-vyn, och om du byter tillbaka till Diagram-vyn så är även denna uppdaterad. Så de får värden men endast vid instansering av vyn, och inte när värden uppdateras när vyn är synlig.

Det är här du kommer in, din uppgift är att lösa så vi kan se direkt när output-värden ändras. Detta skall göras med hjälp av designmönstret Observer (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?

** Det är endast tre klasser som du behöva göra ändringar i för att lösa uppgiften. 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. Om du vill får du självklart göra ändringar i andra klasser.

OBS Javas native lösning för observer 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

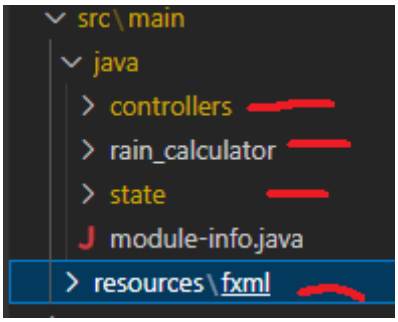
När vi betygsätter inlämningen tittar vi på om:

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

Hjälpmedel

Genomgång av klasserna i projektet:

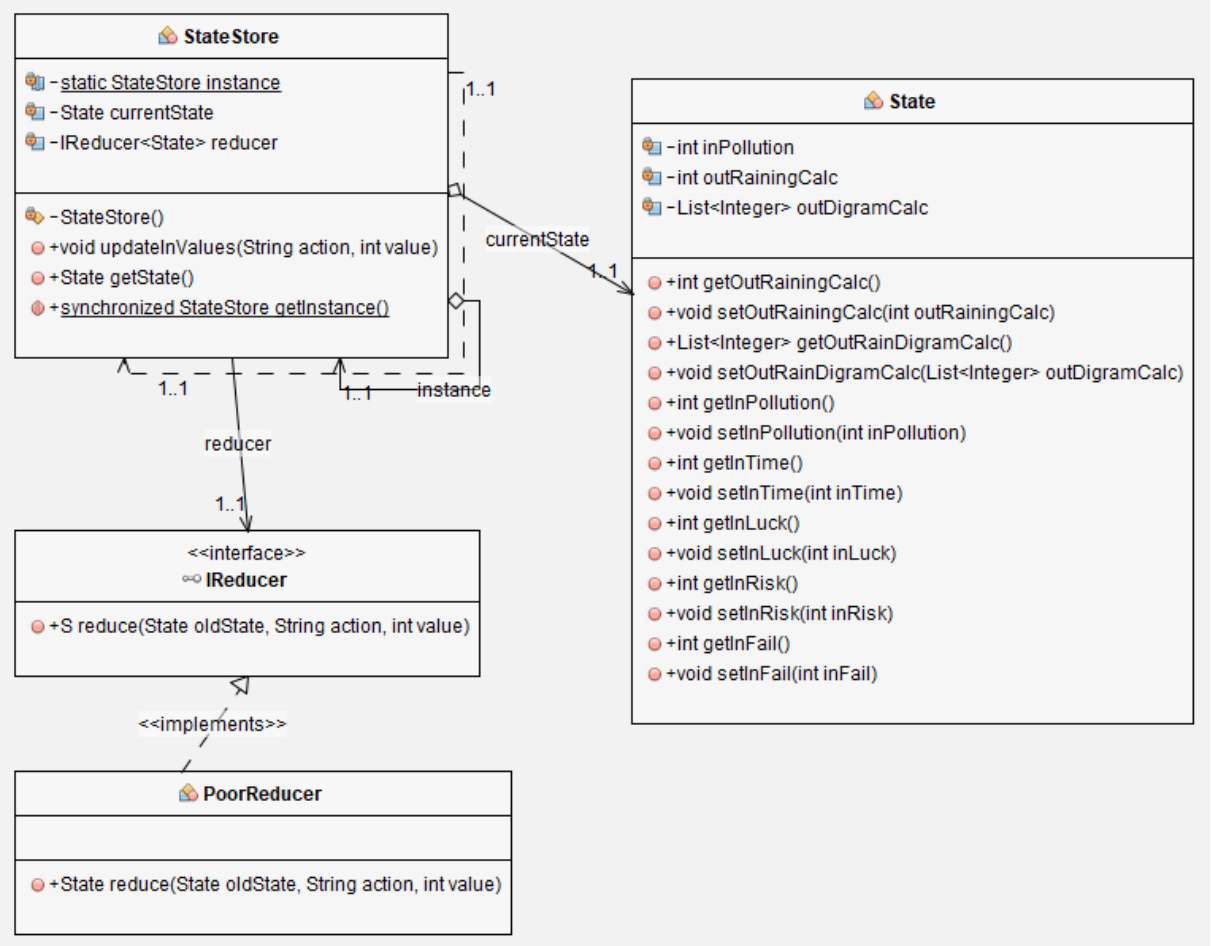
I applikationen återfinns fyra moduler (paket/packages):



I detta dokument gås controllers samt state paketen igenom, huvudansvar samt delar som kan vara viktiga för dig för att lösa uppgiften. Övriga paket bör vara självförklarande men har du några frågor så kontakta aktuell lärare.

State paketet

State paketet innehåller logik rörande hantering av applikationens tillstånd (den data som den för närvarande använder). Paketet innehåller 3 klasser och ett interface:



Klassen State: Ansvarar för att hålla reda på data, alltså en ren modellklass där de värden som behövs i applikationen sparas.

Klassen StateStore: Ansvarar för att hålla reda på applikationens nuvarande tillstånd (State-klassen). Det är även denna klassen utomstående klasser kan anropa för att göra ändringar på ”tillståndet”. Detta görs då med anrop till metoden **updateInValues.** updateInValues tar två parametrar, action och value.

Denna metoden anropas i de olika kontroller klasserna som börjar med Input, exempel: updateInValues("pollution", 50);

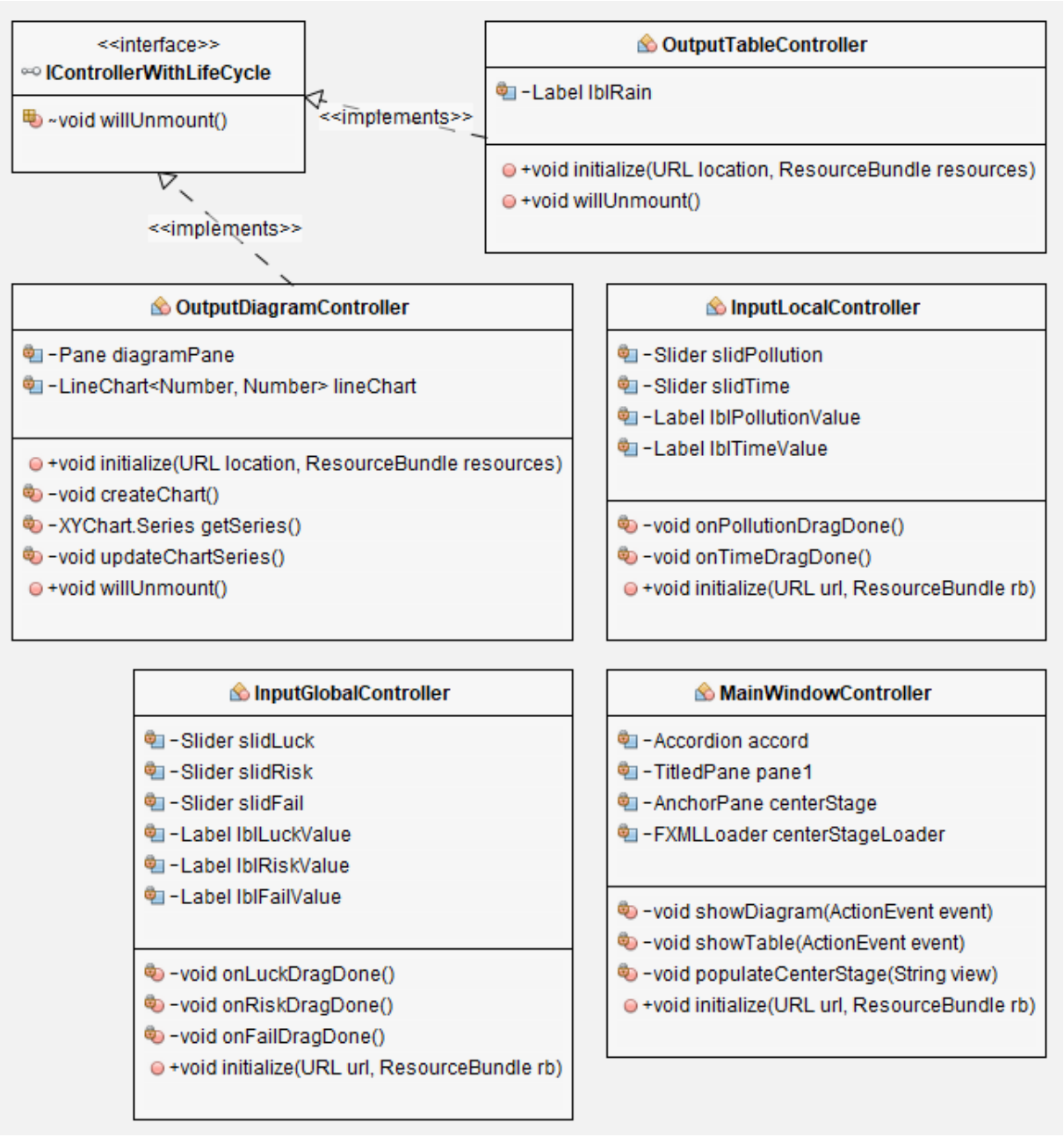
För att säga att vi vill uppdatera värdet av pollution till 50.

Interface Reducer samt klassen PoorReducer:

Idén med en reducer är kunna ta en behållare med data (State) och utföra en modifikation (action) på den. När det är gjort ges ett nytt tillstånd tillbaka där modifikationen nu återspeglas.

Detta finns implementerat genom ett interface som definierar hur Reducer's får se ut (API). Vi har en PoorReducer som utför själva jobbet (konkret klass). Namnet Poor föreslår att detta är en fattig implementation av mönstret (state-reducer-pattern).

Controllers paketet:



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. Input-vyer sätts i pane1 respektive pane2 (de utgör vänstra delen av applikationen). I metoden populateCenterStage så sätts vyn som skall synas till höger i applikationen (output data). Vid start av applikationen anges diagramvyn som output. I menyn går det sedan att ändra så man i stället ser resultatet som en tabell.

InputGlobalController + InputLocalController:
Kontrollerar främst när någon drar i dem sliders som finns i inputvyerna, när det sker så kommuniceras det vidare till State-paketet i applikationen att användaren vill uppdatera applikationens tillstånd.

OutputDiagramController:
Kontrollerar Output data för vyn ”Diagram”. Metoden updateChartSeries kan anropas för att säga till klassen att hämta och skapta data till diagrammet. Denna data hämtas då ifrån State-paketet. Ingen ny kod behövs för själva diagrammet, men det som efterfrågas i uppgiften är när denna updateChartSeries skall köras.

OutputTableController:
Bör när du gjort din lösning uppdatera sin Label så det för användaren redovisas det som återfinns när man anropar getOutRainingCalc() metoden som återfinns i State-klassen.

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