

Laboration 2 - XML och Trådar

Namn	Emil Söderlind
CS	id15esd
CAS	emsa0120
Inmälningsdatum	25 januari 2019
Kurs	Applikationsutveckling i Java HT18
Kurskod	5DV135
Kursansvarig	Johan Eliasson
E-brev	Soderlindemil@gmail.com
Version	2

Innehåll

1	Användarhandledning	2
1.1	Leveransbeskrivning	2
1.2	Exempelkörning av programmet	2
2	Systembeskrivning	4
2.1	Klassers ansvar	4
2.2	XML-parsning APIManager	7
2.3	Swing GUI View	7
2.4	Actionlisteners Controller	8
2.5	Affärslogik Model	8
2.6	Designmönster	9
2.6.1	MVC	9
2.6.2	Observer	9
2.6.3	DAO objekt	10
3	Trådsäkerhet	11
3.1	Kommunikation mellan trådar	11

1 Användarhandledning

Laborationen har gått ut på att implementera ett java program som tillhandahåller Sveriges Radios radiokanalers tablå via ett GUI implementerat i javas Swing bibliotek.

Nedan beskrivs vilka filer som levereras samt en exempelkörning av programmet.

1.1 Leveransbeskrivning

Denna laboration består av följande filer:

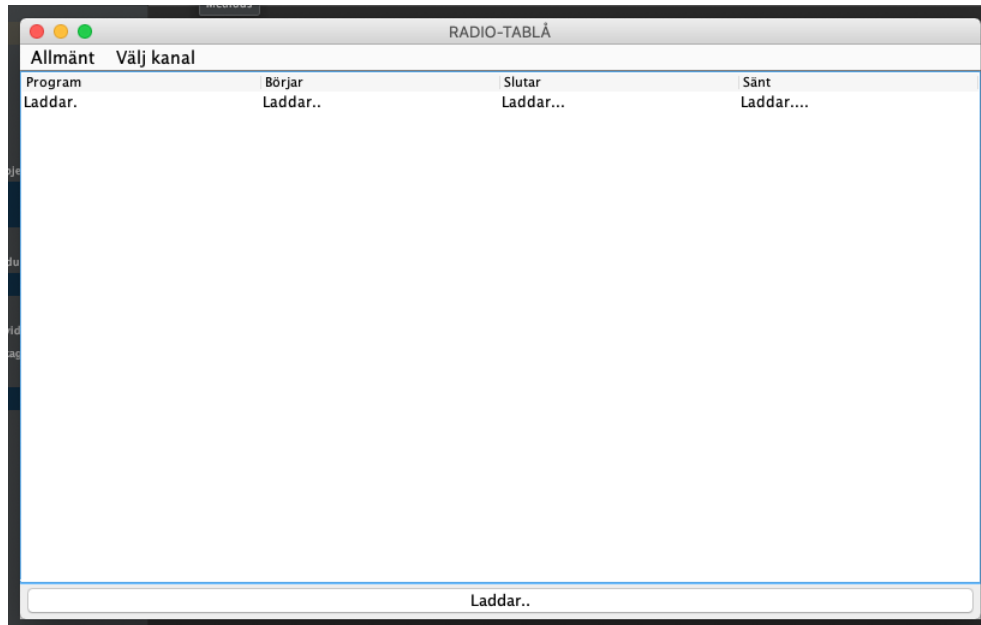
- **OU2_rapport.pdf**
Laborationsrapport som du läser just nu.
- **source.zip**
 - Main.java
 - Model.java
 - View.java
 - Controller.java
 - APIManager.java
 - SchedulePackage.java
 - ChannelSchedule.java
 - Program.java
 - ResourceProvider.java
 - Observer.java
 - Subject.java
- **SR_Schedule.jar**
En kompilerad och körbar jar-fil

1.2 Exempelkörning av programmet

För att starta programmet körs följande i samma katalog som SR_Schedule.jar:

Listing 1: Körning av program

```
$ java -jar SR_Schedule.jar
```



Figur 1: Senaste tablå hämtas

1. Inledningsvis när programmet startas hämtas ny data från Sveriges Radio API, se figur 1. Det tar strax under en minut .
2. När tablå är hämtad visas den valda kanalens logotype till vänster och dess tablå (+/- 12 timmar) till vänster, se figur 2. Programmens titel, starttid, sluttid och huruvida programmet redan sänts visas i tabellen.
3. Under Välj kanal fliken kan användaren välja kanal att presentera tablå för, se figur 4.
4. Ifall användaren klickar på ett program i tabellen presenteras en popup-ruta med mer information om programmet, se figur 4.

RADIO-TABLÅ

Välj kanal (1) | **Välj kanal (2)** | Välj kanal (3)

P4 Sörmland
P4 Uppland
P4 Värmland
P4 Väst
P4 Västerbotten
P4 Västernorrland
P4 Västmanland
P4 Örebro
P4 Östergötland
SR Söpmi
Sisuradio
P6
P3 Din gata
P2 Musik
P3 Star
Radioapans knattekanal
P2 Klassiskt
P2 Världen

örebro
sverigesradio

Program	Börjar	Slutar	Sänt
Radiosporten	12:02	12:04	x
P4 Extra	12:04	12:30	x
Nyheter från SR Örebro	12:30	12:32	x
P4 Extra	12:32	13:00	x
Nyheter från Ekot	13:00	13:03	x
P4 Extra	13:03	13:30	x
Nyheter från SR Örebro	13:30	13:32	x
P4 Extra	13:32	13:57	x
Kulturnytt i P4	13:57	13:59	x
Eftermiddag i P4 Örebro	13:59	14:00	x
Nyheter från Ekot	14:00	14:03	x
Eftermiddag i P4 Örebro	14:03	14:30	
Nyheter från SR Örebro	14:30	14:32	
Eftermiddag i P4 Örebro	14:32	14:38	
Radiosporten	14:38	14:42	
Eftermiddag i P4 Örebro	14:42	15:00	
Nyheter från Ekot	15:00	15:02	
Eftermiddag i P4 Örebro	15:02	15:30	
Nyheter från SR Örebro	15:30	15:34	
Eftermiddag i P4 Örebro	15:34	15:39	
Eftermiddag i P4 Örebro	15:39	15:45	
Dagens Eko: sammanfattning av dagens n...	15:45	16:00	
Eftermiddag i P4 Örebro	16:00	16:25	
Nyheter från Ekot	16:25	16:26	
Eftermiddag i P4 Örebro	16:26	16:30	
Nyheter från SR Örebro	16:30	16:34	
Eftermiddag i P4 Örebro	16:34	16:35	
Kvällspasset i P4	16:35	17:01	
Nyheter från Ekot	17:01	17:02	
Radiosporten	17:02	17:03	
Kvällspasset i P4	17:03	17:43	

Uppdatera tablå

Figur 2: Tablån hämtad

2 Systembeskrivning

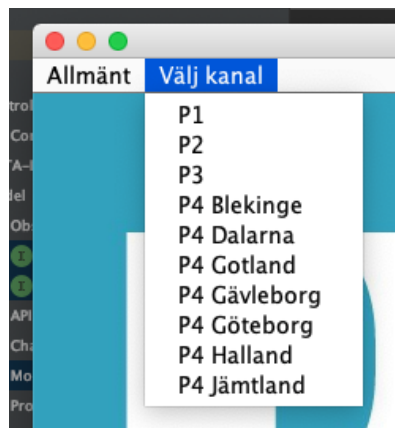
Programmet utgår ifrån en MVC struktur för kunna sära på affärslogik Model och presentationen View med en mellanhand Controller. I figur 5 återfinns ett UML klassdiagram över MVC-strukturen samt APIManager som ansvarar för parsning av Sveriges Radios XML API, läs mer om det i kapitel 2.2.

Nedan beskrivs samtliga klassers ansvar översiktligt, centrala klasser mer utförligt samt hur olika designmönster är applicerade.

2.1 Klassers ansvar

Nedan beskrivs samtliga klassers ansvar översiktligt.

- **Model**
Affärslogiken i programmet. Läs mer i kapitel 2.5.
- **APIManager**
Parsar API och skapar SchedulePackage objekt. Läs mer kapitel 2.2.
- **Main**
Startar programmet, skapar instanser av Model, View, Controller kopplar ihop dessa samt kör huvudloopen i Model.

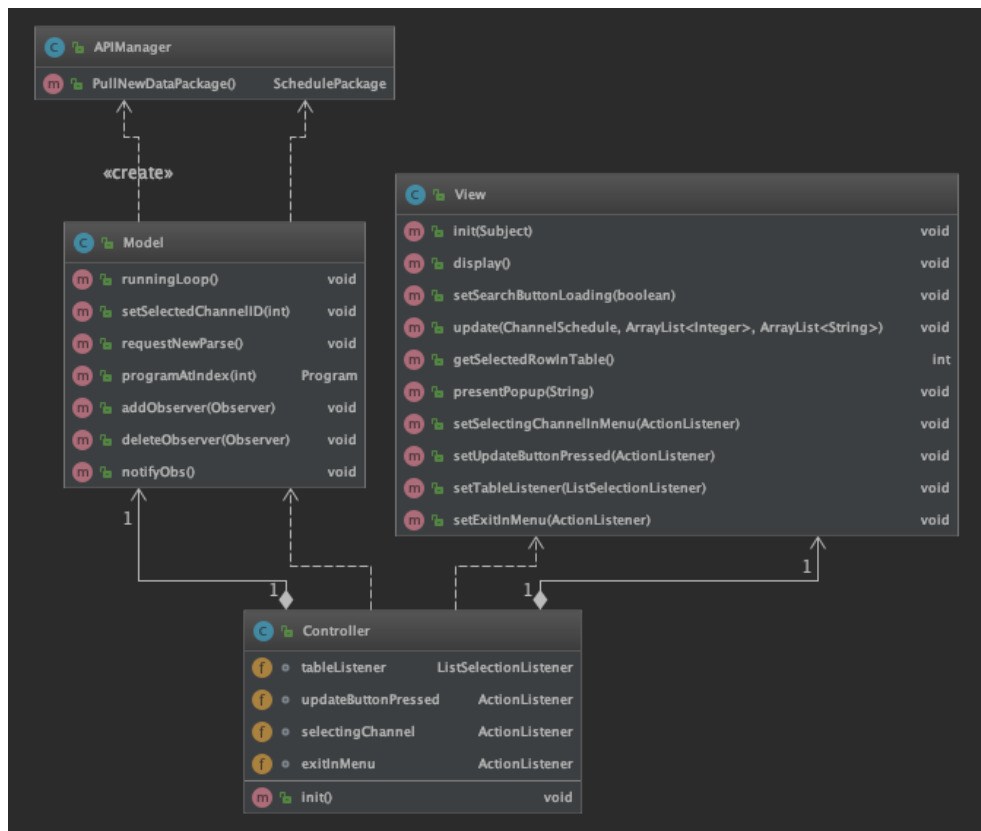


Figur 3: Senaste tablå hämtas



Figur 4: Senaste tablå hämtas

- **View**
Ansvarar för att upprätthålla användargränssnittet. Läs mer i kapitel 2.3.
- **Controller**
Fungerar som mellanhand till Model och View. Ansvarar för lyssnare i GUI. Läs mer i kapitel 2.4.
- **SchedulePackage**
DAO-objekt. Representerar en parsning av samtliga kanaler och deras program. Läs mer i kapitel 2.6.3.
- **ChannelSchedule**
DAO-objekt. Representerar en kanal och dennes program. Läs mer i kapitel 2.6.3.



Figur 5: UML klassdiagram över de mest centrala delarna programmet

- **Program**
DAO-objekt. Representerar ett program. Läs mer i kapitel [2.6.3](#).
- **ResourceProvider**
Interface som deklarerar hur en ResourceProviders gränsyta skall vara utformat. APIManager implementerar detta.
- **Observer**
Interface som deklarerar hur en Observers gränsyta skall vara utformat. View implementerar detta. Läs mer i kapitel [2.6.2](#).
- **Subject**
Interface som deklarerar hur en Subjects gränsyta skall vara utformat. Model implementerar detta. Läs mer i kapitel [2.6.2](#).

2.2 XML-parsning | **APIManager**

För att erhålla Sveriges Radios samtliga kanalers tablå använder `APIManager` sig av deras XML-baserade API. Genom en sekvens av hämtningar skapar `APIManager` ett `SchedulePackage`-objekt av samtliga tillgängliga kanalers program de senaste och kommande 12 timmarna. En parsning går enligt följande:

1. Hämtar tillgängliga kanalers namn, ID och logotype
2. Skapa en instans av `SchedulePackage` R
3. För varje kanals ID
 - (a) Skapa en `ChannelSchedule` instans K av kanalen
 - (b) För varje dag av; igår, idag & imorgon:
 - i. För varje sida i resultatet:
 - A. För varje program i sidan:
 1. Skapa en `Program` instans I av programmet
 2. Adderar I till K .
 - (c) Addera K till R
4. Returnera `SchedulePackage` R

2.3 Swing GUI | **View**

`View` bygger upp och upprätthåller GUI:t som användaren interagerar med. GUI:t använder sig av en `borderlayout`. GUI:t består av en `JFrame` innehållande:

- **Menybar**
Menyfältet består av två delmenyer; Allmänt, där man kan stänga av programmet och Välj kanal, där man kan välja kanal, se figur 3.
- **Logotype**
West i `borderlayout` ligger en `JPanel` som visar upp den valda kanalens logotype, se figur 2.
- **Tablå**
Center i `borderlayout` ligger en `JTable` som visar upp den valda kanalens tablå, se figur 2.
- **Uppdatera knapp**
South i `borderlayout` återfinns en `JButton`, se figur 2.

Läs mer om hur View erhåller tablå-data från Model via observer-gränssnittet i kapitel [2.6.2](#).

2.4 Actionlisteners | Controller

Inledningsvis kopplar Controller ihop Model och View genom deras observer designmönster, se kapitel [2.6.2](#). Detta för att Model ska kunna tillhandahålla View med tabell-data utan ett allt för starkt beroende.

Samtliga actionlisteners som körs vid event i GUI är deklarerade och exekverade i Controller. Samtliga actionlisteners finns beskrivna nedan:

- **"Uppdatera tablå-knapp"**
Körs vid klick på "Uppdatera tablå-knappen". Sätter flaggan new-ParseRequested till true i Model. Leder till att Model kör en ny parsning
- **Avsluta**
Körs vid klick "Ävsluta" i menyn under allmänt, stänger av programmet.
- **Väljer ny kanal**
Körs då användaren väljer en kanal under "Välj kanal" i menyn. Leder till att Model ger View den valda kanalens tablå
- **Mer information om program**
Körs då användaren klickar på ett program i tabellen. Leder till att Controller erhåller programmet från Model och ber View presentera ett popup-meddelande med beskrivning av programmet.

2.5 Affärslogik | Model

Model använder en instans av APIManager för att beställa en ny instans av tillgängliga kanalers radio-tablå, i form av en instans av ett DAO objekt SchedulePackage.

Model kör i main-tråden och utfår ifrån följande exekvering:

1. Första hämtning av SchedulePackage instans från en APIManager
2. Kalla på Observer:s (View) Update () -metod med vald kanalen.
3. Var 120 ms (Timer) kör den följande:
 - (a) Ifall det gått en timme sedan sista hämtning

- i. Utför ny hämtning av `SchedulePackage`
 - ii. Kalla på `Observer:s (View) Update ()`-metod med vald kanalen
- (b) Ifall användaren via `Controller` beställt ny hämtning, genom att sätta en flagga `newParseRequested`, i `Model`
- i. Utför ny hämtning av `SchedulePackage`
 - ii. Kalla på `Observer:s (View) Update ()`-metod med vald kanalen
 - iii. Sätter flaggan `newParseRequested` till `false`

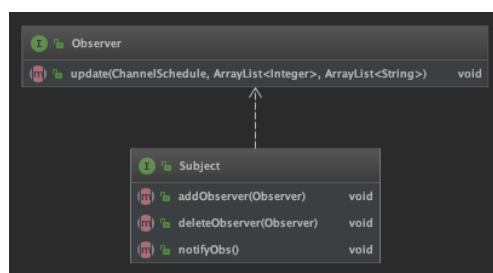
2.6 Designmönster

Nedan beskriver jag ett antal designmönster som jag valt att implementera i implementationen.

2.6.1 MVC

För att sära på affärslogik `Model` och presentationen `View` används MVC-designmönstret, se figur 5.

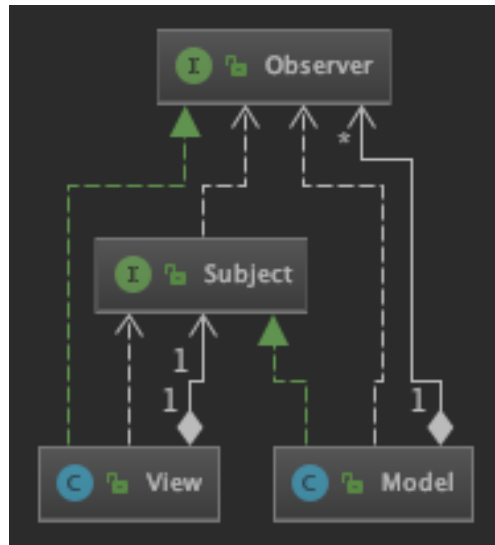
2.6.2 Observer



Figur 6: Observer-subject

För att tillhandahålla vyn `View` med efterfrågad kanals tablådata, från affärslogiken i modellen `Model`, används ett Observer-subject designmönster. I och med att `Vyn` implementerar interfacet `Observer` har denne en `Update ()`-metod som `Model` kan kalla på, i och med att

vederbörande implementerar interfacet `Subject`. Se figur 6 för interface:ns. Detta leder till att det inte finns något direkt beroende mellan `View` och `Model`, se figur 7.



Figur 7: `Model` och `View` i Observer-subject

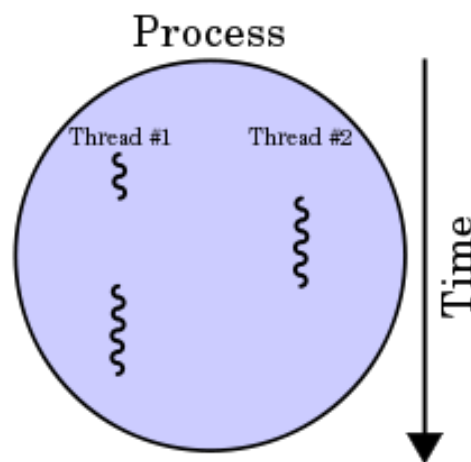
2.6.3 DAO objekt

För att förvara en parsnings data i form av de olika kanalerna och deras program struktureras dessa utifrån tre olika lager av DAO-objekt. Se figure 8. Nedan beskrivs de olika:

- `Program` | Ett program med:
 - Titel
 - Beskrivning
 - Starttid
 - Sluttid
 - Sämt program (Ja/Nej)
- `ChannelSchedule` | En kanal med:
 - Kanals namn
 - Kanals ID
 - Lista med kanalens `Program`

- Kanalens logotype
- `SchedulePackage` | Alla tillgängliga kanalers `ChannelSchedule`:
 - När hämtningen skedde
 - Lista med kanalernas `SchedulePackage`

3 Trådsäkerhet



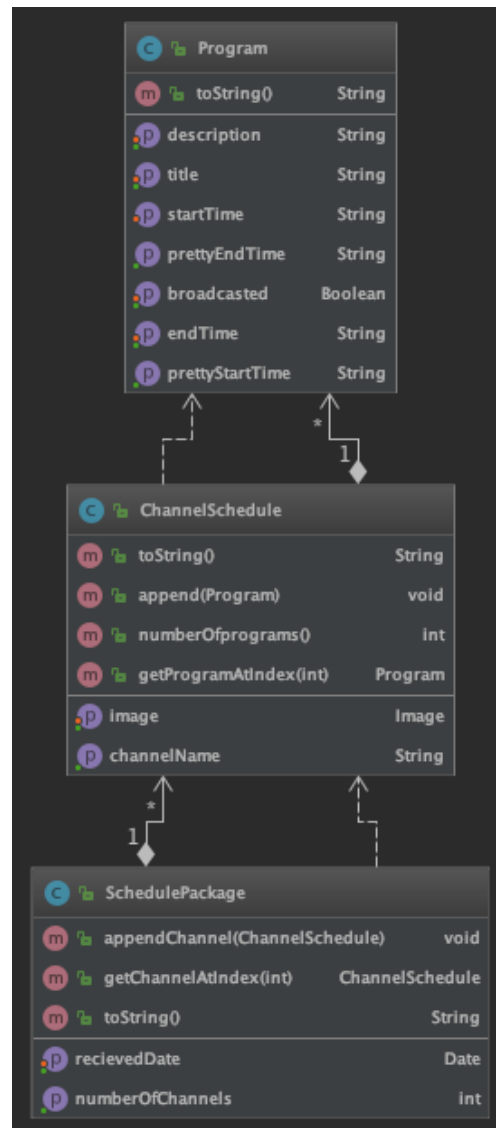
Figur 9: Illustrerad tråd

Programmet använder sig av följande trådar:

- Logik-tråd | `Model`
Uppgift: Kör parsning och affärslogik med main-tråden.
- Swing-tråd | `View`
Uppgift: Ansvarar för EDT och uppdatering av GUI

3.1 Kommunikation mellan trådar

När logik-tråden ska uppdatera GUI vid t.ex. ny parsning, kallas samtliga observers update-metoder med ny data. I update-metoden körs uppdateringen i sin tur med `SwingUtilities.invokeLater()` vilket får swing/EDT tråden att själv exekvera uppdateringen av GUI, vilket därmed säkerställer swing/GUI:ts **trådsäkerhet**.



Figur 8: Lager av DAO-objekt