Trådar och XML/JSON - OU3

Applikationsutveckling (JAVA), ht2022

Olivia Harlin, olha0073, id19ohn

Innehållsförteckning

1. Inledning

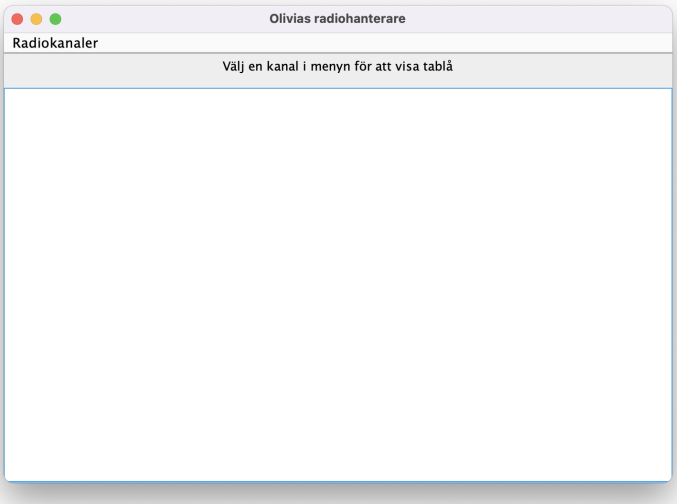
Denna rapport beskriver den tredje och sista laborationen i kursen Applikationutveckling (Java). Uppgiften gick ut på att skapa ett program som visar information om olika radiokanaler och deras program. Syftet var att få insikt i hur XML/JSON kan implementeras och användas, träna på att skapa egna GUIn, skriva trådsäkra program och följa ett givet dataformat samt öva färdigheter i rapportskrivning. Programmet skulle byggas enligt en MVC-struktur, vara användarvänlig samt hantera de undantag som kan uppstå.

2. Användarhandledning

1. Öppna programmet i IntelliJ IDEA.
2. Kontrollera att rätt version av java är vald genom *file > project structure.* Under SDK bör JDK 17 visas.
3. Kompilera programmet genom att klicka på hammar-ikonen (build project) högst upp i fönstret.
4. Kör sedan programmet så att en out-mapp skapas om en sådan inte redan finns. Detta gör du genom att klicka på run-knappen höst upp i fönstret.
5. Nu ska en körbar JAR-fil skapas. Börja med att navigera till *file > project structure.*
6. Klicka på *Artifacts* i menyn till vänster.
7. Klicka på + och välj sedan *JAR > From modules with dependencies.*
8. Välj sedan programmets main fil genom att klicka på mapp-ikonen och sedan Main.java. Klicka på *Apply*.
9. Nu borde en JAR-fil existera under *out > artifacts.* Om inte, gå till *build > build artifacts* så bör filen genereras.
10. Öppna terminalen och stega dit jar-filen finns. Starta nu programmet genom föjande kommando:

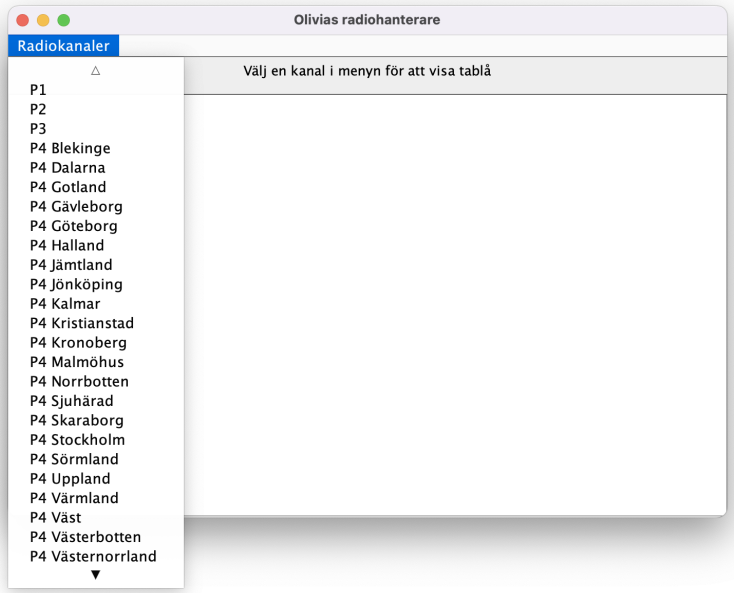
```
java -jar RadioProject.jar
```

När användaren startar programmet så blir vyn i figur 1 synlig.



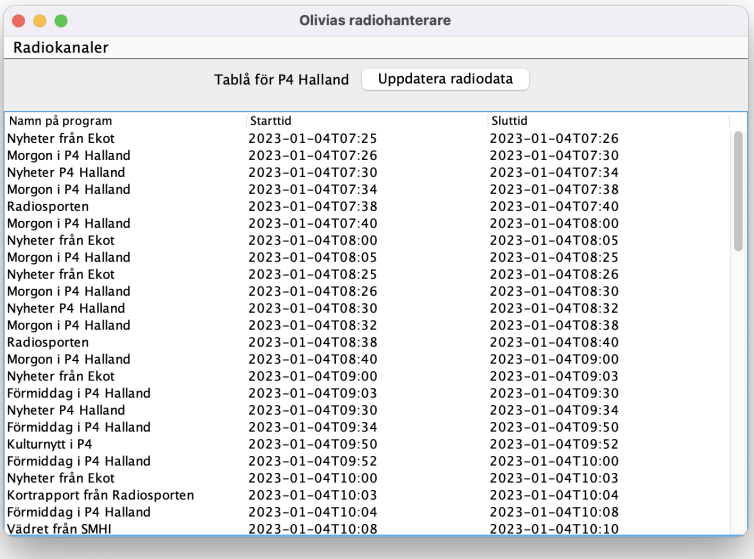
Figur 1 - Startvy

Texten i vyn instruerar användaren att välja en radiokanal i menyn. Menyn är skrollbar och ger användaren möjlighet att välja en av radiokanalerna att se motsvarande tablå för. Se figur 2.



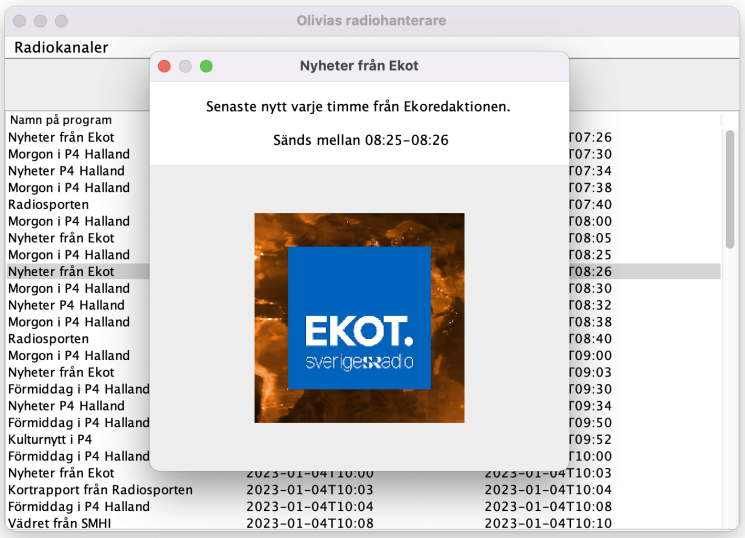
Figur 2 - Vy med menyval

När användaren valt en radiokanal i menyn så kommer tablån för kanalen att visas som en tabell i vyn. I vyn finns även en knapp med alternativ att uppdatera tablån. Se figur 3.



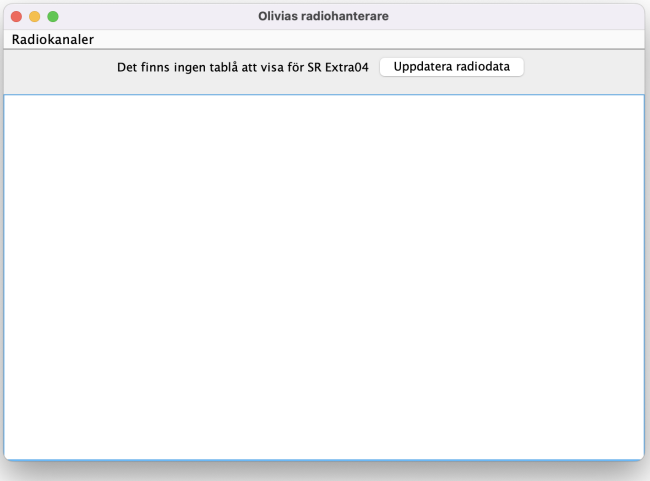
Figur 3 - Radiotablå

Nu har användaren möjlighet att klicka på en rad i tabellen för att se mer information om det valda programmet. Detta visas genom dialogrutan i figur 4.



Figur 4 - Dialogruta med programinformation

Om användaren istället väljer en kanal som inte har någon tablå att visa så kommer vyn i figur 5 att bli synlig för användaren. Texten i vyn informerar om att det inte finns



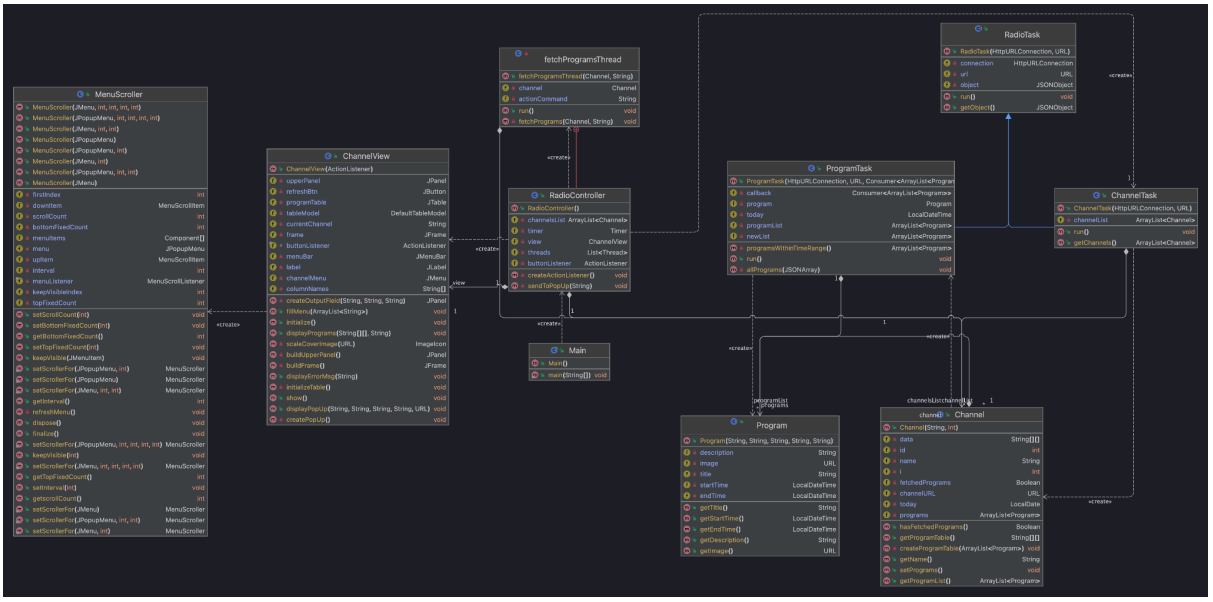
Figur 5 - Ingen tablå att visa

3. Systembeskrivning

Följande avsnitt beskriver systemets uppbyggnad och funktionallitet.

3.1 UML

I figur 6 visas systemet beskrivet av ett klassdiagram.



Figur 5 - Klassdiagram över systemet

3.2 MVC-struktur och algoritm

MVC står för Model-View-Controller och är en programstruktur som ofta används för sammansättningen av applikationer. Modellen är tänkt att inehålla data, regler och instruktioner som applikationen följer. Vyn representerar de tillstånd som visas för användaren. Styrningen (controller) sköter kommunikationen mellan applikationens olika delar, ex. genom att ta in användarens instruktioner och omvandla dessa så att programmet kan tolka dem.[1]

Programmet följer en MVC-struktur bestående av fem modeller, ett styrningsobjekt (controller) och två klasser tillhörande vyn. Dessa klasser och deras funktion beskrivs nedan.

RadioController

Denna klass är programmets styrning och sköter därför all kommunikation mellan vy och modell. Programmet inleds med att en instans av RadioController skapas. Det första som sker i konstruktorn är att en lista skapas för att hålla reda på de olika trådar som ska komma att

[1] https://it-ord.idg.se/ord/model-view-controller/

skapas i programmet. Därefter etableras en koppling till API:et för att på så sätt kunna hämta alla kanaler som ska visas i vyn. I denna lösning så hämtas kanalerna endast en gång.

Efter att kanalerna hämtats så skapas en lyssnare via metoden createActionListener. Denna metod ser till att hantera de olika ActionCommands som vyn returnerar vid ex. ett knapptryck och därefter utföra matchande handlingar. Först kontrolleras det så att kommandot från vyn inte är null. Därefter kontrolleras det huruvida kommandot inleds med “show” eller inte. Om så är fallet så indikerar det att data ska hämtas för att visas upp i en dialogruta. Om kommandot inte inleds med “show” så kontrolleras det istället huruvida kommandot inleds med “refresh” eller inte. Om så är fallet så indikerar det att en befintligt inläst kanal ska läsas in på nytt från API:et. Oavsett om kanaler ska hämtas på nytt från API:et eller läsas in från cachad data så görs detta genom att skapa en tråd som i sin run-metod avgör hur hämtningen ska ske. Därefter startas tråden. Se pseudo-kod nedan för beskrivning av logiken i createActionListener.

```
if actionCommand is not null

   if actionCommand starts with “show”
      send programInfo to popUp

   else if actionCommand starts with “refresh”
      splitString <- split actionCommand
      channel <- splitString[1]
      findChannelandStartThread()

   else
      channel <- actionCommand
      findChannelandStartThread()
```

Efter att ActionListenern skapats så kan vyn skapas. Kanalerna skickas in till vyn via metoden fillMenu som ser till att menyn i vyn fylls med de olika kanalerna som menyobjekt.

RadioTask (abstrakt klass)

RadioTask är en abstrakt klass som används för att standardisera de “tasks” som används i programmet för att hämta kanaler och program. Klassen består av en run-metod vars uppgift är att etablera en koppling till API:et baserat på den URL som konstruktorn tar in. Därefter hämtas XML-objektet och konverteras till ett JSON-objekt med hjälp av följande kodrad.

```
object = XML.toJSONObject(String.valueOf(stringBuilder));
```

Channel

Denna klass representerar en kanal. Förutom att kanalen har ett namn så har den även en lista med alla tillhörande program. Första gången en kanals program ska läsas in så anropas metoden setPrograms(). Denna har i uppgift att göra själva API:anropet en gång i timmen. Detta sker genom en timer som kör programTask en gång varje timme. ProgramTask returnerar programmen i en lista via en callback. När listan tagits emot så kan en tvådemensionell String-array skapas utifrån programmet som lästs in. Det är denna som senare ska komma att visas i vyn.

ChannelTask

Denna klass har i uppgift att hämta alla kanaler från API:et. Den ärver därför klassen RadioTask vars run-metod anropas först i klassens egna run-metoden. Därefter plockas relevanta delar ut ur JSON-objektet för att skapa olika kanalobjekt som läggs till i en lista channelList.

Program

Denna klass representerar ett radioprogram. Ett program består av en titel, en beskrivning, start- och sluttid samt en omslagsbild. Alla dessa tilldelas via konstruktorn där allt tas in som strängar och sedan konverteras till rätt datatyp. Tiderna måste konverteras i två steg. Först till ZonedDateTime och sedan till LocalDateTime som är den form som används i övrigt i programmet.

ProgramTask

ProgramTask har i uppgift att hämta program från API:et och ärver därför RadioTask vars run-metod anropas först i den egna run-metoden. Därefter hämtas alla program från API:et i en JSON-array. Ur denna sorteras sedan alla program med sändningstid som börjar 6 timmar innan och 12 timmar efter den aktuella tidpunkten. Det är dessa som senare ska komma att visas i vyn. Lisan av program inom tidsspannet returneras via en callback. Om ett fel uppstår, ex. om det inte finns några program att hämta så skapas ett temporärt program-objekt som indikerar att det inte finns några program att hämta. Detta returneras sedan via en callback.

MenuScroller

Denna klass är tagen från kod skriven av Darryl Burke hämtad från följande sida https://tips4java.wordpress.com/2009/02/01/menu-scroller/?fbclid=IwAR113Z3xdvZEg-W2TMkG0sYXLU-Mp-TcyS-xpodSJfTRfXi9N5Bf-QBMMvU. Klassen används enbart för att göra menyn i vyn skrollbar.

ChannelView

Denna klass representerar vyn. Först byggs en frame och sedan panelen som innehåller den text som visas till användaren samt knappen för att uppdatera informationen i vyn. Därefter initieras tabellen som senare ska komma att visa all radioinformation. Detta sker genom att en DefaultTabelModel först skapas som sedan används för att skapa själva tabellen.

Efter att tabellen har initierats skapas dialogrutan som ska visas då användaren klickar på en programrad. Detta sker genom att en mouseListener skapas som lyssnar på eventuella klick. Då användaren klickar någonstans så hämtas den markerade tabellraden. Därefter simuleras ett klick så att ett action command kan skickas till Controllern. Controllern kan utifrån det inkommande kommandot svara med den information som ska visas i dialogrutan.

Efter att dialogrutan skapats så kan menyn byggas. Först skapas en JMenuBar, sedan en JMenu som används för att skapa ett MenuScroller-objekt. Menyn kan sedan läggas till i vyn. JMenuItems skapas då metoden fillMenu anropas vars uppgift är att fylla menyn med alla radiokanaler.

Själva tabellen fylls via en metod displayPrograms. Här sätts refresh-knappens actionCommand till “refresh” samt kanalnamnet, alltså ex. “refresh P4”. Metoden tar bland annat in en tvådimensionell string-array som används för att skapa själva tabellen. Om denna är null så går det att anta att inga program finns att visa i tabellen, därav skrivs istället ett meddelande ut till användaren i vyn. Om det finns data att visa så fylls tabellen och namnet på kanalen visas som rubrik. Tabellen “skrivs över” med hjälp av repaint().

Main

Main-klassen har endast i uppgift att skapa en instans av RadioController-klassen och därmed starta programmet.

3.3 Trådsäkerhet

För att se till att programmet är trådsäkert så har flera åtgärder vidtagits. Eftersom Java Swing inte är trådsäkert för uppdatering av komponenter så används invokeLater(). Denna ser till att komponenterna endast kan uppdateras från Event Dispatcher tråden. Se kod nedan.

```
SwingUtilities.invokeLater(() -> {
            view = new ChannelView(buttonListener);
            view.show();
            view.fillMenu(channelsList);
        });
```

Utöver att se till att swing och vykomponenterna är trådsäkra så är det viktigt att metoderna som används för hämtning av data från API:et också uppfyller kraven för trådsäkerhet. Det skulle potentiellt kunna uppstå problem om användaren lyckas initera en hämtning då en annan redan pågår. Därför startas en ny tråd varje gång metoden fetchPrograms() anropas. Se kod nedan.

```
threads.add(new fetchProgramsThread(channelsList.get(i),
e.getActionCommand()));
                        threads.get(threads.size()-1).start();
```

De metoder som potentiellt skulle kunna orsaka ett så kallat Race Condition har deklarerats Synchronized. På så sätt ser programmet till att endast en tråd ges åtkomst till metoden åt gången. De metoder som deklarerats som Synchronized är framförallt sådana metoder som tilldelar värden till varabler som sedan används för att visa rätt data i vyn. Listan av program är en sådan variabel som endast får modifieras av en tråd åt gången därav har metoden setPrograms() deklarerats som Synchronized.

3.4 Designmönster

För att uppnå en god programstruktur så har flertalet designmönster används vid skapandet av programmet. Eftersom Observer-mönstret är grunden till den MVC-struktur som programmet följer så går det att säga att detta mönster implementerats.

Mönstret Template Method har används för att konstruera Task-klasserna. ChannelTask och ProgramTask ärver den abstrakta superklassen RadioTask där en stor den av run-metoden finns definierad. Denna används i båda subklassernas egna run-metoder.

Ett annat mönster som går att återfinna bland annat i Task-klasserna är Command-mönstret. Dessa klasser är objekt där något som ska utföras kapslats in i klassen. Samma gäller för den interna klassen fetchProgramsThread vars enda uppgift är att utföra trådarnas uppgifter.