TDDD78 / 729A85 – Objektorienterad programmering och Java

**Projektbeskrivning**

**<Omega – Tv show tracker>**

**2015-02-12**

**Projektmedlemmar:**

Fredrik Iselius <[freis685@student.liu.se](mailto:freis685@student.liu.se)>

Pontus Iversen <[poniv820@student.liu.se](mailto:poniv820@student.liu.se)>

**Handledare:**

Mariusz Wzorek <mariusz.wzorek@liu.se>

Table of Contents

1. Introduktion till projektet 2

2. Ytterligare bakgrundsinformation 2

3. Milstolpar 2

3. Övriga implementationsförberedelser 3

4. Utveckling och samarbete 4

5. Implementationsbeskrivning 5

5.1. Milstolpar 5

5.2. Dokumentation för programkod 5

5.3. Användning av fritt material 6

5.4. Användning av objektorientering 6

5.5. Motiverade designbeslut med alternativ 6

5.6. Bredd inom Java och objektorientering 6

6. Användarmanual 7

7. Slutgiltiga betygsambitioner 7

8. Utvärdering och erfarenheter 7

|  |
| --- |
| Planering |

Den här delen av projektbeskrivningen skriver ni i samband med första inlämningen, gärna under tiden ni arbetar på sista labben för att göra det möjligt att få kommentarer innan projektstart. Den lan kompletteras senare, men försök få med så mycket som möjligt redan från början.

**Läs först genom** [**http://www.ida.liu.se/~TDDD78/labs/2015/select.shtml**](http://www.ida.liu.se/~TDDD78/labs/2014/select.shtml) **om att välja projekt**!

**Instruktionerna nedan tar ni så klart bort när ni skrivit klart projektbeskrivningen.**

1. Introduktion till projektet

Omegas huvudsakliga uppgift är att “tracka” tv-serier. Tanken är att man ska kunna få veta vilket avsnitt man ska se härnäst, när det kommer och välja vilka tv-serier man vill följa. Det kommer även finnas möjlighet att se hur lång tid man spenderat på att titta på tv-serie totalt och en av våra bästa funktioner kommer vara ”rewatch”. ”Rewatch” kommer skapa en separat instans för vald serie. Detta kommer ge möjlighet att se vilket avsnitt man ska se härnäst i omstarten och samtidigt veta vilket man ska se när det kommer nya. Användaren kommer även ha möjlighet att upptäcka nya serier med vår discovery-funktion.

BILD ?

2. Ytterligare bakgrundsinformation

För att hålla koll på alla olika tv-serier använder vi som utav en databas importerad från internet. Hemsidan vi använder oss av heter thetvdb.com. Databasen förser oss med all information vi behöver så som avsnittens visningsdatum, information om varje serie, speltid osv.

Trakt är en webbaserad applikation som huvudsakligt har samma syfte som vår produkt kommer att ha. De har en API som vi kommer använda oss utav för att synkronisera vårt data mellan Trakt och vår produkt. VI kommer även använda deras API för att få serierekommendationer till vår discovery-funktion.

API står för Application Programming Interface och är ett gränssnitt mellan en applikation och en viss specifik programvara. APIt fungerar som en mjukvarukomponent och låter applikationen kommunicera och använda denna programvara som till exempel kan vara ett dynamiskt bibliotek. (Läs mer här: <http://sv.wikipedia.org/wiki/Application_Programming_Interface>)

Information vi kommer att hämta från thetvdb och Trakt är i formaten .xml och .json. För att använda denna data kommer vi att ”parsa” den hjälp av olika parser.

En parser är ett program som analyserar en dataström, i detta fall datan från thetvdb och Trakt. Att ”parsa” innbär att man lyfter ut den information som behövs för att sedan kunna lagra det i vår databas.

3. Milstolpar

1. Sätta upp en SQL-databas där vi kan lagra alla information.
2. Få en fungerande anslutning mellan thetvdb.com och vårt program med hjälp av API.
3. Hämta och ”parsa” information. Första steget blir att hämta ID:et för varje serie. Sedan hämta resterande information med hjälp av ID:et.
4. Få in den ”parsade” informationen i databasen.
5. Göra ett test för att se så all information hanteras korrekt.
6. Skapa funktion som ger möjlighet att lägga till vilka serier man vill följa.
7. Implementera simpelt GUI för att visa nästkommande avsnitt. Någon typ av simpel veckokalender för visa avsnitt i korrekt ordning. (Mer utbyggnad på GUI kommer senare)
8. Läs in information från thetvdb från start. Databasuppdatering vid uppstart. (Progressmeter (?) som laddar medans informationen laddar.)
9. Markera som sedd / Uppdatera playcount, antalet gånger man sett ett avsnitt. (Lägga till timestamp när man kollat på ettt avsnitt för bättre synkronisering med Trakt senare.)
10. Göra så man ser total tid man kollat på serier.
11. Status på serier. Välja om man har börjat kolla på dem eller vill se dem senare osv.
12. ”Rewatch”. Se ”Introduktion” för ytterligare förklaring.
13. Synka vår data (playcounts osv) med Trakt.
14. Discovery-funktion. (Baserad på Trakt-rekommendationen.)
15. Trakt. Check in-funktion som ger information att man börjat kolla på ett avsnitt. Det visas då på Trakt att man håller på att kolla på avsnittet.
16. Förfina GUI, fixa snyggare struktur m.m.
17. Implementera ”popular/trending” från Trakt.
18. Implementera samma sak fast för filmer.

3. Övriga implementationsförberedelser

Man kan ibland spara mycket tid på att först ta en ordentlig funderare på t.ex. vad programmet ska göra, ungefär hur funktionaliteten kan delas upp mellan olika klasser, osv.

Tänk därför gärna genom den grundläggande strukturen på programmet och skriv ner de tankar ni har om detta. Det går inte nödvändigtvis att bestämma hela strukturen i förväg, men vissa delar kanske kan planeras i detalj och vissa delar kan beskrivas i stora drag. Detta avsnitt är inte del av examinationen, men ju mer ni kan beskriva här, desto större chans att ni får användbara kommentarer från handledaren.

Många av tidigare års studenter har rapporterat att de hade stor nytta av det här. En del har istället rapporterat att projektet visade sig bli helt annorlunda än de hade tänkt.

4. Utveckling och samarbete

Vi siktar först och främst på att jobba så mycket som möjligt i skolan. Dels på alla schemalagda pass men även vid andra tillfället då det passar bra. Vid behov kommer vi även att programmera på helgerna. Vi kommer att sitta med varandra så mycket som möjligt för att kunna diskutera och lättare komma fram till lösningar. Någon gång i veckan kommer vi även gå igenom vad vi gjort så båda är med på hur koden fungerar och även uppdatera projektbeskrivningen.

Vi har båda fixat GitHub också så vi kan versionshantera koden och även lägga till ”issues” vi behöver lösa. Uppdelningen av koden kommer ske under tiden av arbetet. Vi anpassar oss efter våra behov att samarbeta och lösa problem tillsammans alternativt skriva självständigt.

Vi siktar på så högt betyg som möjligt. Vi jobbar på och gör så gott vi kan så får vi se hur långt det räcker. Vi har svårt nu på förhand att veta hur svåra vissa delar kommer vara att implementera.

|  |
| --- |
| Slutinlämning |

Även om denna del inte ska lämnas in förrän projektet är klart, är det **viktigt att arbeta med den kontinuerligt** under projektets gång! Speciellt finns det några avsnitt där ni ska beskriva information som ni lätt kan glömma av när veckorna går (vilket flera tidigare studenter också har kommenterat).

5. Implementationsbeskrivning

Den här sektionen använder ni för att beskriva hur projektet är strukturerat och implementerat. Algoritmer och övergripande design passar också in i det här kapitlet (bilder, flödesdiagram, osv. kan vara användbart). Skapa gärna egna delkapitel för enskilda delar, om det underlättar.

5.1. Milstolpar

Ange för varje milstolpe om ni har genomfört den helt, delvis eller inte alls. (Detta är till för att labbhandledaren ska veta vilken funktionalitet man kan ”leta efter” i koden. Själva bedömningen beror inte på antalet milstolpar i sig!)

5.2. Dokumentation för programkod, inklusive UML-diagram

Programkod behöver dokumenteras för att man ska förstå hur den fungerar och hur allt hänger ihop. Vissa typer av dokumentation är direkt relaterad till ett enda fält, en enda metod eller en enda klass och placeras då lämpligast vid fältet, metoden eller klassen i en Javadoc-kommentar, *inte här*. Då är det både enklare att hitta dokumentationen och större chans att den faktiskt uppdateras när det sker ändringar. Annan dokumentation är mer övergripande och placeras då här. Det kan gälla till exempel:

* Övergripande programstruktur, t.ex. att man har implementerat ett spel som styrs av timer-tick n gånger per sekund där man vid varje sådant tick först tar hand om input och gör eventuella förflyttningar för objekt av typ X, Y och Z, därefter kontrollerar kollisioner vilket sker med hjälp av klass W, och till slut uppdaterar skärmen.

* Översikter över relaterade klasser och hur de hänger ihop.
  + Här kan det ofta vara bra att använda **UML-diagram** för att illustrera – det finns även i betygskraven. Fundera då först på vilka grupper av klasser det är ni vill beskriva, och skapa sedan ett UML-diagram för varje grupp av klasser (det är sällan särskilt användbart att lägga in hela projektet i ett enda gigantiskt diagram).
  + Skriv sedan en textbeskrivning av vad det är ni illustrerar med UML-diagrammet. Texten är den huvudsakliga dokumentationen medan UML-diagrammet hjälper läsaren att förstå texten och få en översikt.
  + IDEA kan hjälpa till att göra klassdiagram som ni sedan kan klippa och klistra in i dokumentet. Högerklicka i en editor och välj Diagrams / Show Diagram. Ni kan sedan lägga till och ta bort klasser med högerklicksmenyn. Exportera till bildfil med högerklick / Export to File.

Labbhandledaren och examinatorn kommer bland annat att använda den här dokumentationen för att förstå programmet vid bedömningen. Ni kan också tänka er att ni själva ska vidareutveckla projektet efter att en annan grupp har utvecklat grunden. Vad skulle ni själva vilja veta i det läget? Om viktig information saknas kan ni få komplettera.

När ni pratar om klasser och metoder **ska deras namn anges tydligt** (inte bara ”vår timerklass” eller ”utritningsmetoden”).

Framhäv gärna det ni själva tycker är bra lösningar eller annat som handledaren borde titta på vid bedömningen.

Se även sidan om kvalitetskriterier på webben.

Vi räknar med att de flesta projekt behöver runt 2-5 sidor för det här avsnittet.

5.3. Användning av fritt material

Skriv ner vilka klassbibliotek och vilket annat fritt material ni har använt utöver det som ingår i Java 8, eller ange uttryckligen att ni inte har använt något.

5.4. Användning av objektorientering

För varje betygsnivå måste ni peka ut hur ni har använt ett visst antal olika **typiska egenskaper hos just objektorienterade språk.** Se projektsidorna på webben för detaljerade instruktioner!

1. …
2. …
3. ...

5.5. Motiverade designbeslut med alternativ

För varje betygsnivå måste ni beskriva ett visst antal **designbeslut** ni har tagit. Se projektsidorna på webben för detaljerade instruktioner!

1. …
2. …
3. …
4. …
5. …
6. …
7. …
8. ...

5.6. Bredd inom Java och objektorientering

För varje betygsnivå måste ni använda ett visst antal specifika OO- eller Java-finesser. Här ska ni beskriva vilka ni har använt och var. Var tydliga – ange klassnamn och metodnamn där det är lämpligt. Se projektsidorna på webben för detaljerade instruktioner!

1. …
2. …
3. …
4. …
5. …
6. …
7. …
8. ...

6. Användarmanual

När ni har implementerat ett program krävs det också en manual som förklarar hur programmet fungerar. Ni ska beskriva programmet tillräckligt mycket för att en labbhandledare själv ska kunna starta det, testa det och förstå hur det fungerar.

Inkludera flera (minst 3) skärmdumpar som visar hur programmet ser ut! Dessa ska vara ”inline” i detta dokument.

7. Slutgiltiga betygsambitioner

Ange här vilket betyg ni slutgiltigt siktar på i ert projekt. Se sedan till att ni har följt de kvantitativa kraven för detta betyg och att detta finns dokumenterat i projektbeskrivningen ovan.

8. Utvärdering och erfarenheter

Detta avsnitt är en väldigt viktig del av projektspecifikationen. Här ska ni tänka tillbaka och utvärdera projektet (något som man alltid bör göra efter ett projekt). Som en hjälp på vägen kan ni utgå från följande frågeställningar (som ni gärna får lämna kvar i texten så att vi lättare kan sammanställa information om specifika frågor):

* *Vad gick bra? Mindre bra?*
* *Lade ni ned för mycket/lite tid?*
* *Var arbetsfördelningen jämn? Om inte: Vad hade ni kunnat göra för att förbättra den?*
* *Har ni haft någon nytta av projektbeskrivningen? Vad har varit mest användbart med den? Minst?*
* *Har arbetet fungerat som ni tänkt er? Har ni följt "arbetsmetodiken"? Något som skiljer sig? Till det bättre? Till det sämre?*
* *Vad har varit mest problematiskt, om man utesluter den programmeringstekniska delen? Alltså saker runt omkring, som att hitta ledig tid eller plats att vara på.*
* *Vad har ni lärt er så här långt som kan vara bra att ta med till era egna kommande kurser/projekt?*
* *Vilka tips skulle ni vilja ge till studenter i nästa års kurs?*
* *Har ni saknat något i kursen som hade underlättat projektet?*

Vi använder detta för att utveckla och förbättra kursen till nästa år. Vissa delar som är användbara som tips till andra studenter kan komma att citeras (givetvis helt anonymt!) under föreläsningar eller på websidor.

**(Glöm inte att exportera till PDF-format innan ni skickar in!)**