Projektbeskrivning

Bibliotekshanteringssystem

2024-03-22

Projektmedlemmar:
Enzo Visser - enzcu445@student.liu.se

Handledare:
Thea Antonson - thean981@student.liu.se

Innehåll

Projektplan

1. Introduktion till projektet

Jag planerar att utveckla ett bibliotekshanteringssystem som syftar till att vara ett vertyg som effektiviser hanteringen av biblioteksresurser. Systemet kommer att möjliggöra registrering, utlåning, återlämning och katalogisering av böcker samt hantering av användarkonton, inklusive bibliotekariers och låntagares. Det finns även utrymme för ytterliggare implimentationer såsom bokrekomendationer och liknande. Målet är att skapa en lättanvänd, intuitiv plattform som både förbättrar tillgängligheten till bibliotekets samlingar och förenklar administrativa uppgifter.

2. Ytterligare bakgrundsinformation

Jag kommer att behöva integrera systemet med en befintlig databas av verk för att simulera ett bibliotek som redan har digitala samlingar. Alternativt kan jag generera påhittade verk eller använda mig av en mindre databas på 100 kända verk såsom: https://github.com/benoitvallon/100-best-books/blob/master/books.json.

3. Milstolpar

#	Beskrivning
1	Ladda ned/upprätta databas av verk samt kunna öppna ett fönster med alla boktitlar.
2	Möjliggör registrering och inloggning av användare.
3	Skapa en mer strukturerad lista på samtliga verk.
4	Implementera funktioner för att lägga till, visa, och ta bort böcker från katalogen. (baserat på vem som är inloggad).
5	Implementera en sökfunktion för att hitta böcker baserat på titel, författare eller ISBN.
6	Möjligöra för lånetagare att låna böcker
7	Implementera funktionalitet för att återlämna böcker och uppdatera bokens status.
8	Utveckla ett enkelt användargränssnitt för de grundläggande funktionerna.
9	Införa olika användarroller (bibliotekarier och låntagare) och deras specifika funktioner.
10	Implementera möjligheten att reservera böcker som är utlånade.
11	Utveckla ett system för att hantera förseningsavgifter och skicka påminnelser till användare med försenade återlämningar.
12	Rapportgenerering för bibliotekarier Skapa funktioner för att generera rapporter om lånade böcker, användaraktiviteter, och förseningsavgifter.

13	Bokrekomendationssystem Utveckla en enkel algoritm för att rekommendera böcker baserat på användarens tidigare lån och intressen.
14	Uppdatera användargränssnittet för att inkludera de nya funktionerna och förbättra användarupplevelsen.
15	Säkerhetsfunktioner Implementera säkerhetsfunktioner som datakryptering och säkra inloggningsförfaranden.
16	Möjliggör för användare att lämna recensioner och betyg på böcker.
17	Sista GUI finslipning.
18	
19	
20	
21	
22	
23	
...	

4. Övriga implementationsförberedelser

Grundläggande Arkitektur

Bok: Representativ klass för en bok, med egenskaper som titel, författare, ISBN, status (tillgänglig, utlånad, reserverad), kategori, genre, antal sidor, utgivningsår. Eventuellt även bild och länk (om jag använder mig av databasen jag hittat).

Användare: Bas klass för en användare med gemensamma egenskaper som namn0 (e-post) och lösenord. Den här klassen kan utökas till subklasser såsom Bibliotekarie och Låntagare för att representera olika roller och deras specifika beteenden och tillåtelser.

Lån: Hanterar information om lånade böcker, inklusive vilken bok som är utlånad, till vem, lånedatum och förfallodatum.

Katalog: En klass som ansvarar för att hantera bokkatalogen, inklusive att lägga till nya böcker, ta bort böcker, och söka efter böcker baserat på olika kriterier.

Gränssnitt

Användarinteraktion: Ett gränssnitt för att hantera all interaktion med användaren.

Databashantering: Definierar de operationer som kan utföras på databasen, såsom att lägga till, ta bort, uppdatera eller hämta data.

Designöverväganden och Förberedelser

Ärvning och polymorfism: Till exempel, genom att låta Bibliotekarie och Låntagare ärva från Användare, kan man enkelt utöka systemet med nya användartyper om jag vill.

Inkapsling: Se till att jag skyddar data och exponerar endast nödvändiga metoder för interaktion med objekten.

5. Utveckling och samarbete

Arbetar själv men kommer försöka att vara i så god tid som möjligt med detta projekt.

(Resten av dokumentet ska inte lämnas in förrän projektet är klart, men titta ändå genom allt för att se vilka delar ni behöver arbeta med och fylla i *kontinuerligt* under projektets gång!)

Projektrapport

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).

Tänk på att ligga på lagom ambitionsnivå! En välskriven implementationsbeskrivning (avsnitt 6) hamnar normalt på **3-6 sidor** i det givna formatet och radavståndet, med ett par mindre UML-diagram och kanske ett par andra små illustrerande bilder vid behov. Hela projektrapporten (denna sista halva av dokumentet) behöver sällan mer än 10-12 sidor.

6. Implementationsbeskrivning

I det här avsnittet, och dess underavsnitt (6.x), beskriver ni olika aspekter av själva *implementationen*, under förutsättning att läsaren redan förstår vad *syftet* med projektet är (det har ju beskrivits tidigare).

Tänk er att någon ska vidareutveckla projektet, kanske genom att fixa eventuella buggar eller skapa utökningar. Då finns det en hel del som den personen kan behöva förstå så att man vet *var* funktionaliteten finns, *hur* den är uppdelad, och så vidare. Algoritmer och övergripande design passar också in i det här kapitlet.

Bilder, flödesdiagram, osv. är starkt rekommenderat!

Skapa gärna egna delkapitel för enskilda delar, om det underlättar. **Ta inte bort några rubriker!**

Även detta är en del av examinationen som visar att ni förstår vad ni gör!

6.1. 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, och inte heller på om man ”hann med” milstolparna eller inte!

6.2. Dokumentation för programstruktur, med 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 saknar en naturlig plats i koden. Då kan den placeras 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.
    - Notera att det sällan är särskilt användbart att lägga in hela projektet i ett enda gigantiskt diagram (vad är det då man fokuserar på?). Hitta intressanta delstrukturer och visa dem. Ni behöver normalt inte ha med fält eller metoder i diagrammen.
    - **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.

I det här avsnittet har ni också en möjlighet att visa upp era kunskapen genom att diskutera koden i objektorienterade termer. Ni kan till exempel diskutera hur ni använder och har nytta av (åtminstone en del av) objekt/klasser, konstruktorer, typhierarkier, interface, ärvning, overriding, abstrakta klasser, subtypspolymorfism, och inkapsling (accessnivåer).

Labbhandledaren och examinatorn kommer bland annat att använda dokumentationen i det här avsnittet 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?

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/intressanta lösningar** eller annat som handledaren borde titta på vid den senare genomgången av programkoden.

Vi räknar med att de flesta projekt behöver runt **3-6 sidor** för det här avsnittet.

7. 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 används*.

Inkludera flera (**minst 3**) **skärmdumpar** som visar hur programmet ser ut! Dessa ska vara ”inline” i detta dokument, inte i separata filer. Sikta på att visa de relevanta delarna av programmet för någon som *inte* startar det själv, utan bara läser manualen!

(Glöm inte att ta bort våra instruktioner, och exportera till PDF-format med korrekt

namn enligt websidorna, innan ni skickar in!)