Gruppuppgift - Bibliotek

1 Inledning

1.1 Genomförande

Uppgiften ska utföras I grupp bestående av 4-6 medlemmar. Tiden för utförande är 4/3-17/3.

Om gruppen är i behov av handledning kontakta Rolf Axelsson, rolf.axelsson@mah.se eller Jonas Wahlfridh, jonas.vahlfrid@mah.se.

Gruppen anmäls på It's Learning senast 8/3. Var noga med att inkludera alla gruppmedlemmar i anmälan av grupp. Grupper kan komma att justeras av lärare om för få eller för många medlemmar finns.

1.2 Redovisning

Inlämning av uppgiften ska ske **senast torsdagen 17/3-2015** på It's Learning. Redovisning äger rum måndag respektive onsdag under vecka 12. Var noga med att alla gruppmedlemmar är aktiva vid redovisningen. Redovisningsschema och granskande grupp publiceras den 18/3.

Inlämningen sker genom att en i gruppen lämnar in filen **DA353A_GUaa.zip** på It´s learning. **aa** ersätts med gruppens nummer. Filen ska innehålla följande:

1. Dokumentet ***DA353A_GUaa.pdf*** vilket ska följa mallen som finns i filen DA353AGU_Rapportmall.doc. Denna mall har utförligare instruktioner än nedan som ska följas. Dokumentet i stort:
   a. Innehåller en uppräkning av gruppens medlemmar
   b. Uttrycker vad gruppens olika medlemmar har bidragit med i lösningen.
   c. En instruktion så att lärare och andra grupper kan kompilera och köra applikationen (se punkt 2).
   d. Alla källkodsfiler som ingår i projektet. Källkodsfilerna ska vara kommenterade så att det rimligt lätt går att förstå olika delar av koden. Källkodsfilerna ska vara ordnade i bokstavsordning och varje klass ska börja på ny sida. Källkoden ska vara formaterad med tydliga indrag för att underlätta läsning.
2. Katalogen ***diagram_CRC*** ska innehålla inskannade handritade diagram och CRC kort. Använd t.ex. kopieringsmaskinerna på Mah som har en bra scanna till mejlfunktion. Anledningen till detta är att gruppen ska visa att man har gjort en bra designansats, innan det datorbaserade verktyget används.
3. Katalogen ***projekt*** vilken ska innehålla Together projektet, med källkodsfiler, klassdiagram och sekvensdiagram och allt som ingår i projektet.

Redovisningen sker gruppvis och under redovisningen deltar normalt ytterligare ett par grupper som åhörare och granskare. En grupp kan räkna med ca 10 minuter för att presentera sitt arbete. Alla gruppmedlemmar måste vara beredda på att svara på frågor om systemets design och implementation. Efter presentationen framför den granskande gruppen sina resultat av granskningen.

2 Beskrivning av uppgiften

Att designa och implementera en applikation som hanterar vissa funktioner i ett bibliotek. Biblioteket har böcker och dvd-filmer för utlåning till låntagare. I programmet ska böcker och dvd-filmer representeras av subklasser till klassen Media. I bilaga 1 finner ni klassen Media.

2.1 Funktioner som ska implementeras i systemet

1. Ett Media-objekt ska kunna lånas ut. En låntagare med visst id (personnummer) lånar ett Media-objekt med visst id. Utlåningen ska endast genomföras om:
   - Media-objektet inte redan är utlånat.
   - Media-objektet finns på biblioteket.
   - Låntagaren är känd på biblioteket.
2. Ett Media-objekt ska kunna lämnas tillbaka. Återlämning sker genom att Media-objektets id anges.
3. En låntagare ska kunna få se en lista över de Media-objekt som just nu är utlånade till henne. Använd en JTextArea för att visa de utlånade objekten.
4. Punkterna 1-3 ovan ska skötas med hjälp av ett grafiskt användargränssnitt.
5. När programmet startas ska följande filer läsas in:
   - Media.txt
   - Lantagare.txt
     Se bilaga 2 för specifikation av filernas innehåll

2.2 Krav på implementationen av systemet

- Vid lagring av objekt ska objektsamlingar konstruerade under kursens gång användas, t.ex. ArrayStack, LinkedStack, ArrayQueue, LinkedQueue, ArrayList, LinkedList, PriorityQueue, BinarySearchTree, AVLTree, HashtableCH och HashtableOH.
- Media-objekten ska lagras i en struktur vilken medger snabb sökning, t.ex. i en hashtabell. Ett bibliotek innehåller nämligen stora mängder objekt.
- Låntagarna ska lagras i en struktur som medger både snabb sökning och att objekten är ordnade (efter personnummer), t.ex. ett (balanserat) binärt sökträd.
- Utlåningen ska lagras i tidsordning, d.v.s. det objekt som lånades först ska vara först i datastrukturen. Det finns dock inga krav på att programmet hålla reda på utlåningsdag, tid m.m eller att det ska gå att söka effektivt bland de utlånade objekten. Till dessa objekt kan du använda en eller flera listor.

2.3 Krav på design av systemet

Designmetodiken och principer som finns beskriven i dokumentet: OOAD – Objektorienterad analys och design, LTH ska följas, se filen OOAD_CRC_LTH.pdf. Model-View-Controller designmönstret ska klart framgå av klassdiagrammet, se https://sv.wikipedia.org/wiki/Model-View-Controller. Det är inte nödvändigt att till punkt och pricka följa direktförbindelse och förbindelse via en observatör, men uppdelningen mellan klasserna ska framgå. Det är t.ex. inte tillåtet att lägga all funktionalitet i användargränssnittet. Det är inte heller tillåtet att skapa för stora klasser, utan varje klass ska ha ett ansvarsområde inte flera.

Tänk på att era klassdiagram och sekvensdiagram måste vara konsistenta på så vis att det inte får finnas några motsägelser eller konflikter mellan diagrammen. Det får exv. inte finnas objekt eller meddelanden i ett sekvensdiagram som inte har någon motsvarighet i klassdiagrammet.

Handritade diagram och CRC kort

Innan övergång till datorbaserade verktyg, ska projektmedlemmarna skissa på klassdiagram, objektdiagram och sekvensdiagram förhand med penna och papper.

Klassdiagram med CRC kort

Skapa registerkort till systemet enligt registerkortsmetoden som finns beskriven i dokumentet ”OOAD – Objektorienterad analys och design” under rubriken ”2. Registerkort – en enkel metod för OOAD”. Använd filen RegisterkortsmallV4.doc som mall. CRC korten ska användas lite modifierat än vad som beskrivs i OOAD dokumentet. Relationer ska inte anges på CRC kortet, utan istället ska relationerna ritas på ett klassdiagram. UML syntaxen för attribut och operationer ska användas på CRC korten. Ansvarsområde, operationer och attribut ska skrivas i på CRC kortet.

Attributens och operationer synligheten ska anges. Inkapsling av operationer och attribut ska tillämpas. Metodparametrar och deras typer/klass samt metodernas returvärde ska framgå.

Attribut/instansvariabler som är associationer/referenser till andra objekt visas bara med en relation dvs. inte som ett attribut i på CRC kortet. Dock ska t.ex. referenser till ”String” och andra objekt som inte tillhör domänen, visas som ett attribut på CRC kortet.

Klasser är ritningar till objekt, därför behöver parameter och returtyp anges när metoder deklareras, enligt exempel nedan.

getUserName(UserNo:int):String
setUserName(userName:String):void

Förutom CRC kort ska även ett klassdiagram som beskriver systemet ritas. Klassdiagrammet ska visa relationer, multiplicitet, navigerbarhet/riktning, associationsnamn och/eller rollnamn. Attribut och operationer behöver inte visas eftersom dessa finns på registerkortet. Rita klassdiagrammet för hand med blyertspenna. Aggregation och komposition anges bara om denna extra information är viktig för implementationen. Om det inte är viktigt att ange detta för implementationen använd vanliga associationer.

Objektdiagram

Ett pågående användning av mjukvaran ska exemplifieras med ett handritat objektdiagram. Namnet och klassen som objektet har instansieras från ska framgå t.ex. playerOne:HumanPlayer. Relationer mellan objekt kallas länk och ritas med ett streck. Länkar kan dekoreras på liknande sätt som associationer. Navigerbarhet/riktning ska visas. Tillstånd på instansvariabler kan anges för att förtydliga, försök att hitta och ange några.

Sekvensdiagram

Visa interaktionen mellan objekten i ett sekvensdiagram. Tänk på att klasser är ritningar till objekt och att objekten lagras i minnesområdet Heap i mjukvarulandet. Sekvensdiagrammet ska visa på realisering av centrala användningsfall för systemets funktionalitet. I dessa diagram visas hur scenarierna realiseras genom att visa hur olika objekt skickar meddelanden (metodanrop) till varandra. Objekten skapas vid uppstart, skapandet av objekten behöver inte visas. Delete/destroy av objekten behöver inte visas. Objekten ska visa vilka klasser de är instanser av samt vara namngivna.

Argument och operationernas/metodernas returvariabelnamn samt variabelnamnet där eventuellt returvärde lagras ska framgå. Till varje objekt behövs det en klass som är en ritning till objektet.

På ett sekvensdiagram är det objekt som utbyter meddelande, enligt exempel nedan.

myUserName = getUserName(UserNo):userName
setUserName(userName):void

På sekvensdiagrammet ska det även vid behov finnas villkorad option, alternativt beteende och iteration.

Design och implementation med UML verktyget Together

Together är en plugin till Eclipse vilket gör att det både går att skriva Java kod och skapa UML diagram i samma verktyg. Installera Together genom att följa instruktionerna i dokumentet Together installation, se filen TogetherInstallationIssue1DraftF.pdf. Together finnas installerat i dom vanliga datorsalarna och spellabbet OR:A332, OR:A333.

Arbetsgång

Dom handritade diagrammen och CRC kort används som indata, för att skapa Java klasser i Together. Starta Together genom att köra Together.exe. Skapa ett Java projekt. Skapa Java klasserna och skriv in deras ansvarsområde i beskrivningen av klassen i en Javadoc kommentar. Skapa metoder och instansvariabler. Skriv stubb kod för t.ex. returvärden så att koden hela tiden kompilerar, exempel return null. Skrivkod för relationerna d.v.s. instansvariabler för referenser till andra domänklasser klasser. När design via kod har kommit en bit på väg adderas ett Together modelleringsprojekt till Java projektet.

Klassdiagram

Klassdiagrammet ska visa relationerna, navigerbarhet, multiplicitet och associationsnamn eller rollnamn. Synligheten ska vara privat på attribut och metoder som inte behöver synas utåt.

Skapa ett klassdiagram och dra in klasser som ska visas på diagrammet. Navigerbarheten ska framgå av riktnings pilen på alla associationen. Höger klicka och välj Properties, markera directed till true. Multiplicitet och associationsnamn eller rollnamn anges på motsvarandesätt genom att högerklicka. Det går att via ett klassdiagram och Java kod fönster samtidigt och göra ändringar, i tex. koden så att klassdiagrammet uppdateras direkt och vise versa. Projektmedlemmarna väljer själva om forward eller reverse engineering ska användas för att skapa klassdiagrammet. Även undantagsklasser ska visas på klassdiagrammen.

Sekvensdiagram

En fördel med Together är att det går att generera sekvensdiagram från Java kod, men det går även att generera kod från sekvensdiagram.
Sekvensdiagram används för att visa hur ett användningsfall kan realiseras. På sekvensdiagrammet visas ett antal objekt som utbyter meddelanden (metodanrop). Objekt behöver klasser för att kunna instansieras och kräver därför att det finns klasser. Under modellering hittas ibland klassen först och andra gånger objektet först. För att ett objekt ska kunna anropa ett annat, krävs att det finns en länk till det andra objektet. Länken instansieras av en association som implementeras med en referens i Java. För att ett objekt ska kunna erbjuda funktionalitet måste det finnas metoder definierade i klassen. Ett sätt att hitta metoder är att gå igenom ett användningsfall och säkerställa att det realiseras på ett sekvensdiagram.

I Together går det som ett första steget skriva ett meddelande i form av en text t.ex. ”add” från ett objektet aController till objektet aCalculator. I steg två kopplas objekten till sina klasser. I steg tre skapas en metod med parametrar och retur typ IntegerNumber Calculator::add(IntegerNumber integerA, IntegerNumber integerB). I steg fyra genereras kod i det anropande objektet med argument integerAnswer = aCalculator->add(integerA, integerB); (höger klicka för att generera kod). Alternativt går det att samtidigt skriva metoden i en editor, och sen välja metoden i sekvensdiagrammet. Tyvärr är det lite omständligt att skapa sekvensdiagram genom forward engineering.

Ett alternativ är reverse engineering, skissa först på papper, Together text meddelanden, ritprogram, löpande text eller enbart i huvudet. OBS det är tillåtet att använda forward engineering i Together, ni kanske gillar det bättre än författaren av detta dokument. Skissen används som indata för att skriva stubbkod. När ett avsnitt stubbkod är skriven används Together för att generera ett sekvensdiagram, höger klicka på en metod och välj ”Generate sequence diagram”. Sekvensdiagrammet används för att visualisera hur användningsfallet realiseras, så att lösningen kan granskas.

2.4 Granskning

Varje grupp som redovisar ska också granska en annan grupp. Vilken grupp som ska granskas och tillhörande filer publiceras på It’s Learning 18/3. Vid redovisning så har den granskande gruppen 5 minuter att muntligt presentera en sammanfattning av sin granskning efter den granskade gruppens presentation.

Vid granskningen bör följande frågeställningar undersökas:

- Kunde koden köras enligt instruktionerna i dokumentationsmaterielet?
- Fungerar systemet som avsett? Detta innebär att ni ska kunna utföra den enligt uppgiften specificerade funktionaliteten.
- Hur fungerar koden vid undantag? Testa att mata in felaktiga värden, trycka på knappar i fel logiskt ordning och liknande.
- Är koden bra strukturerad? Är exempelvis indelningen i klasser logisk och följer Model-View-Controller mönstret, är metoder och variabler tydligt namngivna, med mera.
- Innehåller dokumentationen det den ska enligt specifikationen ovan?
- Stämmer klassdiagram och kod överens?
- Stämmer klassdiagram, sekvensdiagram överens?
- Stämmer interaktionen i sekvensdiagrammen överens med koden?
- Är de valda scenarierna på sekvensdiagrammen centrala för systemets funktionalitet?

3 Bilagor

3.1 Bilaga 1

```
// Klassen Media är abstrakt och måste därför ärvas. Böcker och dvs-
filmer
// ska representeras av klasser vilka ärver Media-klassen.
public abstract class Media {
    private String id;

    public Media( String id ) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public boolean equals( Object obj ) {
        Media media = (Media)obj;
        return id.equals( media.getId() );
    }
}
```

3.2 Bilaga 2

Media.txt

Innehållet i Media.txt beskriver Media-objekt av typen Bok eller Dvd. En rad i textfilen är formaterad på lite olika sätt beroende på om raden beskriver en bok eller en dvd. En rad inleds alltid med:
Typ av media;id, …

Exempel på rad som beskriver en bok:

Bok;427769;Deitel;Java how to program;2005

Raden består alltså av: typ av media, id, författare, titel och utgivningsår. Lägg märke till att semikolon skiljer på olika delar av informationen.

Exempel på en rad som beskriver en dvd:

Dvd;635492;Nile City 105,6;1994;Robert Gustavsson;Johan Rheborg;Henrik Schyffert

Raden består alltså av: typ av media, dvd-nummer, titel, utgivningsår och skådespelare. Antalet skådespelare kan vara från noll och uppåt.

Lantagare.txt

Innehållet i Lantagare.txt beskriver låntagare. Varje rad i filen är en låntagare och är formaterad så här:
personnummer;namn;hemtelefon

Exempel på rad i filen

891216-1111;Harald Svensson;040-471024

Inläsning från textfil

Läsningen från textfilerna ovan kan ni göra på samma sätt som gjorts på vissa laborationer på kursen. När ni delar upp den inlästa strängen med split-metoden så ska ni använda argumentet ”;”, t.ex inläsning av Lantagare.txt:

```
: // Här skapas bl.a. lämplig datastruktur
String str;
String[] values;
Lantagare lantagare;
while( ( str = reader.readLine() ) != null ) {
    values = str.split(";");
    lantagare = new Lantagare( values[ 0 ], values[ 1 ], values[ 2 ] );
    map.put( values[ 0 ], lantagare );   // nyckel = personnummer,
                                         // värde = Lantagare-objekt
}
: // Här returneras bl.a. strukturen med Lantagare-objekt
```