Gruppuppgift - Bibliotek

1 Inledning

1.1 Genomförande

Uppgiften ska utföras I grupp bestående av 3-4 medlemmar. Tiden för utförande är 17/2-16/3.

Om gruppen är i behov av handledning kontakta Rolf Axelsson, rolf.axelsson@mah.se eller Kristina Allder, kristina.allder@mah.se.

Gruppen skapas på It's Learning senast 24/2. 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 16/3-2017** på It's Learning. Redovisning äger rum tisdag vecka 12. Var noga med att alla gruppmedlemmar är aktiva vid redovisningen. Redovisningsschema och granskande grupp publiceras den 17/3.

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. Dokumentation i form av klassdiagram och sekvensdiagram ritade med något verktyg.
   e. 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.

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 håller 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.
- Applikationen ska ha ett grafiskt användargränssnitt.

Designen av systemet ska följa tanken om boundary, control och entity-klasser.

- Boundary-klasser har ingen eller mycket lite av logiken i systemet. Boundary-klasser känner endast till andra boundary-klasser och/eller control-klasser.
- Control-klasser ansvarar för den övergripande logiken i systemet och behöver känna till både boundary- och entity-klasser. Control-klasser behöver dock inte känna till alla boundary- och entity-klasser.
- Entity-klasser lagrar data i systemet och ansvarar för beräkningar och transformationer av denna data. Entity-klasser känner endast till andra entity- och/eller control-klasser.

2.3 Krav på designdokumentation av systemet

Systemets design ska dokumenteras med klassdiagram och sekvensdiagram.

För klassdiagrammet gäller att:

- Klassdiagrammet visar alla klasser i systemet som ni själva skrivit och hur dessa är associerade till varandra. Är där vissa bibliotheksklasser eller andra hjälpklasser som ni anser förtydligar diagrammet så bör även dessa visas, detta kan exempelvis vara klasser som hanterar datastruktuer eller kommunikation.
- Associationer mellan klasser ska visas med så lämplig associationstyp som möjligt. Associationer ska vara namngivna där detta rimligtvis kan ske och för alla associationer där multiplicitet kan tillämpas så skall detta visas.
- Alla klasser ska vara angivna som boundary, control eller entity-klass på något sätt (kan visas på olika sätt med UML – välj ett av dessa).
- Klassdiagrammet behöver inte visa attribut och operationer om ni inte anser att detta behövs av tydlighetsskäl.
- Klassdiagrammet behöver inte visa alla klasser som används för att bygga upp GUIt. Det räcker om den klass som representerar själva fönstret visas i klassdiagrammet. Används flera fönster i GUIt ska alla dessa klasser synas i diagrammet.

Gruppen ska också skapa ett visst antal sekvensdiagram som visar interaktionen mellan objekt vid exekvering. Varje person i gruppen är ansvarig för och ritar minst ett sekvensdiagram. För sekvensdiagrammen så gäller att:

- Sekvensdiagrammen behöver endast visa de objekt för GUIt som finns i klassdiagrammet.
- Sekvensdiagrammen ska visa alla meddelanden som skickas till något objekt av de klasser som finns i klassdiagrammet.
- Alla selektioner och iterationer i kod ni själva skrivit ska visas i sekvensdiagrammen.
- De anrop eller strukturer som sker internt i hjälpklasser för exempelvis datastrukturer, objektsamlingar eller kommunikation behöver inte visas. Anrop från något objekt till ett objekt av en hjälpklass måste visas men det behöver inte visas vad som sker inuti den metod som anropas i hjälpklassen.

Varje person i gruppen ska vara ansvarig för minst ett sekvensdiagram. Oavsett gruppensstorlek så ska alltid sekvensdiagram 1-4 nedan göras. Sekvensdiagrammen ska visa kodens struktur för följande fall:

1. Uppstart av programmet inklusive inläsning av textfiler.
2. Utlåning av ett Media-objekt till en låntagare.
3. Listning av utlånade Media-objekt för en viss låntagare.
4. Återlämning av media-objekt.

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 17/3. 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, är metoder och variabler tydligt namngivna, med mera.
- Innehåller dokumentationen det den ska enligt specifikationen ovan?
- Stämmer klassdiagram och kod överens?
- Stämmer klassdiagram och sekvensdiagram överens?
- Stämmer interaktionen i sekvensdiagrammen överens med koden?

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 ) {
        if(obj instanceof Media) {
            Media media = (Media)obj;
            return id.equals( media.getId() );
        }
        return false;
    }
}
```

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
```