Labb5_H20

Laboration 5 - Livsträdet

Mål

- Att kunna tolka given programkod och använda den i en ny tillämpning.
- Att kunna bygga upp en trädstruktur från en textfil med XML-kod.
  Det är tillåtet att använda någon XML-parser (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/prutt16/laborationer/parser.txt) men instruktionerna är anpassade till att ni själva programmerar avkodning av filen.
- Att kunna använda den avancerade grafiska bibliotekskomponenten **JTree** för att visualisera ett träd.
- Att förstå **JTree**:s hjälpklasser och att kunna hantera utvidgad information i trädets noder.

Boken *Object-oriented programming in Java* av *Martin Kalin* har använts som kursbok på DD2385 (tidigare version av DD1385). I boken finns ett program som grafiskt presenterar filkatalogträd med hjälp av en avancerad swingkomponent, **JTree**.

Uppgiften i denna labb är att grafiskt presentera en annan trädstruktur. **JTree** ska användas men också den inramning till **JTree** som Kalins program har.

Kalins kod har modifierats en del och delats upp på två klasser. Först har vi `TreeFrame` som definierar trädet och dess inramning: En `TreeFrame` *är* en `JFrame` samt innehåller en knapp, en liten klickruta och det klickbara trädet. `TreeFrame` är ett körbart program men det träd som skapas hara bara ett rotobjekt, inga grenar.

Den andra klassen, `DirTree2`, ärver `TreeFrame` och definierar om de metoder i `TreeFrame` som skapar själva trädstrukturen så att det är ett filkatalogträd som visas. Några nya hjälpmetoder och någon variabel definieras också i `DirTree2`.

TreeFrame.java (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/TreeFrame.java).
DirTree2.java (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/DirTree2.java)

Spara båda dessa klasser. Provkör dem om ni vill. Se till att ni förstår hur de är konstruerade, åtminstone i stora drag.

`java TreeFrame` visar ett träd med bara en enda nod, roten.

`java DirTree2` visar trädet för den filkatalog du står i.

`java DirTree2 ~` visar trädet för din hemkatalog under Linux/Unix (kan ta tid om den är stor).

`java DirTree2 ../../recept` visar katalogen recept som förutsätts ligga två nivåer uppåt i katalogträdet.

Genom klick på katalognoder visas/döljs innehållet. Om `Show details` är markerad ger enkelklick detaljinfo om vald fil eller katalog.

Labbuppgift: Ny subklass till TreeFrame

Labbuppgiftens program ska ha en struktur som liknar `DirTree2`, dvs det ska ärva från `TreeFrame` och definiera om de metoder som behövs för att beskriva ett nytt träd. De metoder som rör filkatalogträd ska förstås ***inte*** vara med i labbuppgiften. Alltså, kopiera DirTree2, ge den ett lämpligare namn för labben och sudda det som har med filkataloger att göra.

Förberedande uppgift: Ett minimalt livsträd

Läs gärna om JTree i API:n, t.ex. här: **JTree (https://docs.oracle.com/javase/9/docs/api/javax/swing/JTree.html)** Dock går det nog att klara uppgiften genom att bara studera exemplen som ges i labbtexten och de givna programfilerna. JTree har många konstruktorer. Vi använder den konstruktor som kräver en trädmodell som parameter. Trädmodellen är en **DefaultTreeModel, (https://docs.oracle.com/javase/9/docs/api/javax/swing/tree/DefaultTreeModel.html)** ett träd som byggs upp av noder av klassen **DefaultMutableTreeNode (https://docs.oracle.com/javase/9/docs/api/javax/swing/tree/DefaultMutableTreeNode.html)** .

Denna förberedande uppgift behöver inte redovisas men bra att göra för att komma igång. Skapa ett träd med en rot och tre barn som alla är löv. I trädets rot ska det stå `Liv` och de tre barnen ska vara `Växter`, `Djur` och `Svampar`. Dessa ska senare i sin tur kunna innehålla ordningar, underordningar, familjer, släkten och arter. Gör en ny subklass till `TreeFrame`.
Definiera `initTree()` att göra följande:

- Skapa en rotnod med ordet "Liv".
- Skapa en trädmodell.
- Tillverka en nod `child` på samma sätt som rotnoden fast med texten "Växter".
- Addera den till trädmodellen med `root` som förälder, `root.add(child);`
  Titta eventuellt i `DirTree2` för att se hur man gör.
- Gör likadant med "Djur" och "Svampar", alltså lägg dem som barn till "Liv".
- Skapa trädet, **JTree**-objektet (kan göras direkt efter modellen skapas).
- Kompilera, kör och kolla att det ser bra ut.

Det går bra att skapa modellobjekt och trädobjekt från en enda rotnod och därefter bygga ut med fler noder.

Ett rekursivt livsträd

I stället för att tillverka noderna "för hand", läs livsträdet från filen **Liv.xml** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/Liv.xml) (finns även som **Liv.txt** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/Liv.txt)) och bygg upp ett träd. Programmet måste klara godtyckligt antal barn på varje nivå. En liten testversion av filen ser ut som visas här nedanför. Filen finns också i **LillaLiv.txt** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/LillaLiv.txt). Om du har problem med svenska bokstäverna, använd följande versioner som är utan åää: **Life.txt** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/Life.txt), **Life.xml** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/Life.xml) och **TinyLife.txt** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/Material/TinyLife.txt)

```
<Biosfär namn="Liv"> är allt som fortplantar sej
<Rike namn="Växter"> kan inte förflytta sej
</Rike>
<Rike namn="Djur"> kan förflytta sej
</Rike>
<Rike namn="Svampar"> är varken djur eller växter
</Rike>
</Biosfär>
```

Förutom nodens namn finns i filerna också en nivå (t.ex. Biosfär, Rike) och en förklarande text. I `DefaultMutableTreeNode` kan bara ett enda attribut (som dock har typen Object) om noden lagras. I uppgiften ska nivå, namn och text lagras för varje nod. Lös detta t.ex. genom att skriva en egen subklass till `DefaultMutableTreeNode`, där subklassen har fler attribut och metoder.

Trädstrukturer är rekursiva. Ett träd kan mycket väl skapas med en rekursiv metod men det går också bra att använda en stack och bygga trädet utan rekursion, i en enkel repetition. Båda är utmärkta programmeringsövningar. Här bredvid ska visas hur labbens program kan se ut vid körning. A*nvänd den här **länken** (http://www.csc.kth.se/utbildning/kth/kurser/DD2385/prutt16/laborationer/Life.png) om bilden inte visas*.

Liksom i DirTree2 ska man kunna ange filnamn i exekveringskommandot

`java LifeTree livfil`

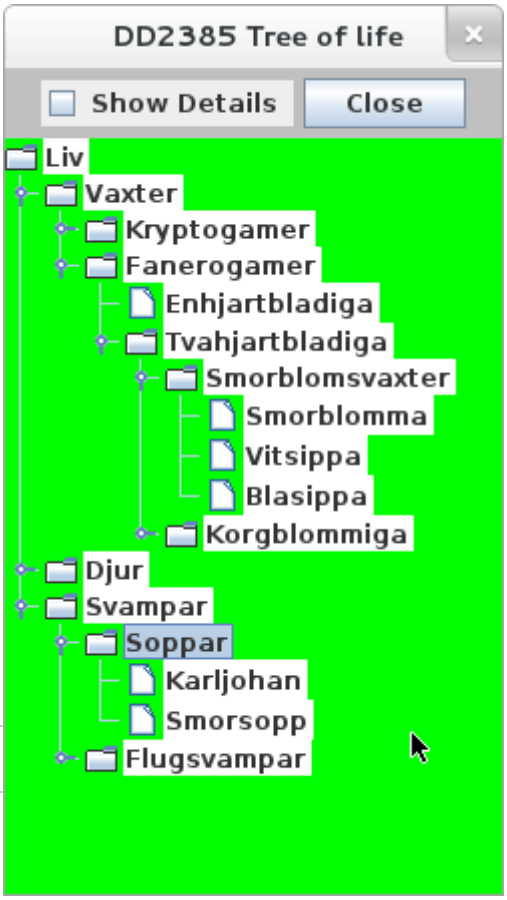
och om inget namn anges ska programmet ta en fil med hårdkodat namn, t.ex. `Liv.xml`.

Detaljupplysningar om livet

I det ursprungliga programmet får man upplysningar om vald fil ifall "Show details" är valt. I ditt program ska man i stället få se den förklarande texten, till exempel

`Art: Mås gillar Vaxholmsbåtar`.

Fixa så att `showDetails(TreePath p)` lägger ut denna text i sin `JOptionPane`.



Den gröna färgen erhålls genom att sätta *trädobjektets* bakgrundsfärg till grön, inte bakgrunden på JFramen som man skulle kunna tro.

Läsning från textfil

Görs enklast med klassen `Scanner`:

```
static Scanner sc = new Scanner(new File("infil.txt"));
```

`new File(...)` kan generera ett `FileNotFoundException` som måste hanteras med `try ... catch`. Om det gick bra att hitta filen kan man sedan läsa radvis med `sc.nextLine()`. Gör `Scanner`-variabeln `static` så kan den skapas redan i `main` - metoden! Med `sc.hasNextLine()` känner man av om det finns fler rader att läsa i filen. Det är bara öppnandet av filen som måste stå i `try ... catch`. Filgenomgången kan göras utanför.

Krav på programmet

- Programmet måste klara indatafiler med godtycklig storlek. Programmet *får inte* förutsätta att noder har ett fast antal barn eller att trädet har ett visst djup.
- Programmet ska kontrollera att starttagg och sluttagg stämmer överens och i annat fall skriva ut felmeddelande och avbryta. Andra typer av fel i indatafilen behöver inte identifieras.

Filformat m.m.

- Programmet får förutsätta att indatafilen är formaterad som exemplen, alltså att filen är radindelad med en "tagg" per rad som exempelfilerna och att det är exakt ett blanktecken mellan nivån och ordet namn samt inga blanktecken alls på andra ställen utom i den förklarande texten i slutet av raderna. Här är en exempelrad:
  `<Biosfär namn="Liv">` är allt som fortplantar sej
- Det är tillåtet att läsa filen radvis, dvs att låta programmet hämta in en hel rad från filen och därefter bearbeta hela den raden, läsa in ny rad o.s.v. Alternativet är att läsa ett tecken i taget vilket gör uppgiften svårare.
- Att plocka ut nivå, namn och text från en rad är en lite "pillig" men inte särskilt svår uppgift, och den är väl avgränsad. Se till att Java-koden för detta inte står sammanblandad med koden

för att bygga trädet. Då blir trädbyggarkoden mer överskådlig vilket är bra särskilt om det skulle behövas felsökning i den. Att avkoda raden kan t.ex. göras i en static-metod som gärna kan ligga i subklassen till DefaultMutableTreeNode.

Självklart är det tillåtet och snyggt att klara av indata som inte är så väl formaterade, t.ex. har flera blanktecken mellan orden, blanktecken före och efter tecknen "=", "<" och ">" samt extra radbyten eller utelämnade radbyten. Det är naturligtvis också utmärkt att identifiera andra syntaxfel än felaktig slut-tagg men det krävs inte!

Redovisning

- Kör programmet med den större infilen och demonstrera alla funktioner.
- Visa UML-diagram.

Extrauppgift för högre betyg (liten)

Ordna så att man förutom grundinformationen **Art: Mås gillar Vaxholmsbåtar** också får hela kedjan av namn utskriven på formen

Men allt som är Mås är Fåglar är Djur är Liv.

Användbar metod i `TreePath` är `getLastPathComponent()`. I `DefaultMutableTreeNode` och därmed i `MyNode` finns metoden `getParent()`.