Laboration 1a: En Trie-modul

1 Syfte

Programmering med referenser, avlusning, testning och korrekthet.

2 Bakgrund

Vi skall under laboration 1a-c stegvis bygga en översättarapplikation. Slutresultatet kan t.ex. se ut som;

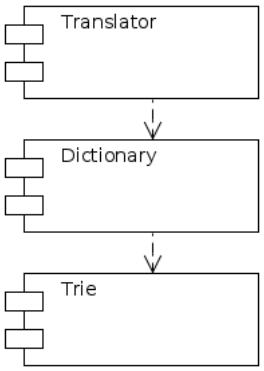
Figur 1: Den slutgiltiga applikationen.

Applikationen kommer att bestå av tre delar (moduler) se Figur 2.

3 Trie

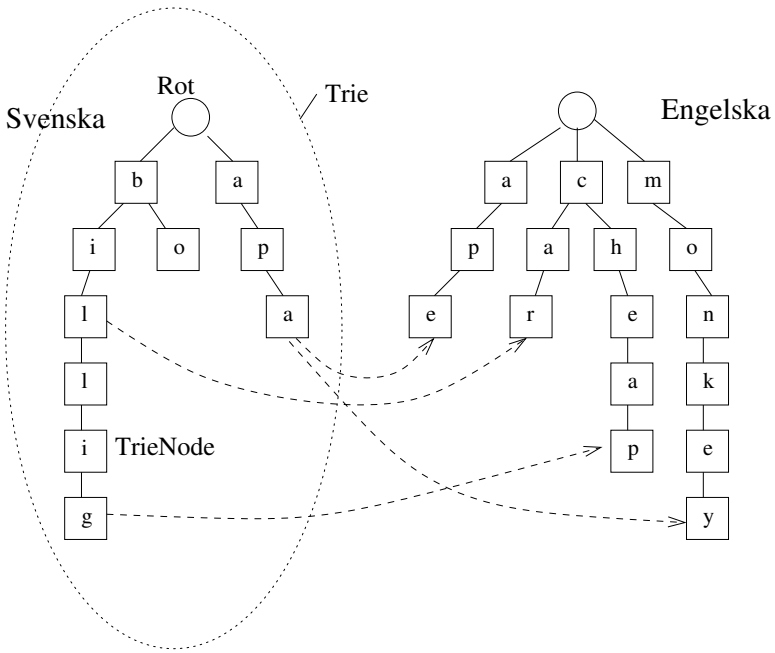
Applikationen arbetar som synes med strängar. Vi behöver något som givet en inledning (ett prefix) av ett ord ger oss en lista med alla ord som inleds med prefixet (och senare en lista med alla översättningar).

I denna laboration skall vi konstruera en klass som kan hjälpa oss med detta, en s.k. Trie (uttalas “tree” eller “try” av eng. retrieval).



Figur 2: Den färdiga applikationens delar (UML komponenter). Trie och Dictionary är moduler som används av applikationen Translator. Pilarna visar på beroenden d.v.s. Translator använder Dictionary som använder Trie.

En Trie är uppbyggt som ett upp och nedvänt "träd". I trädet sparar man strängarna tecken för tecken i "grenar". Med denna struktur blir det lätt att hitta alla ord som inleds med ett visst prefix. De finns på samma gren. Dessutom är det ett effektivt sätt att lagra orden på.

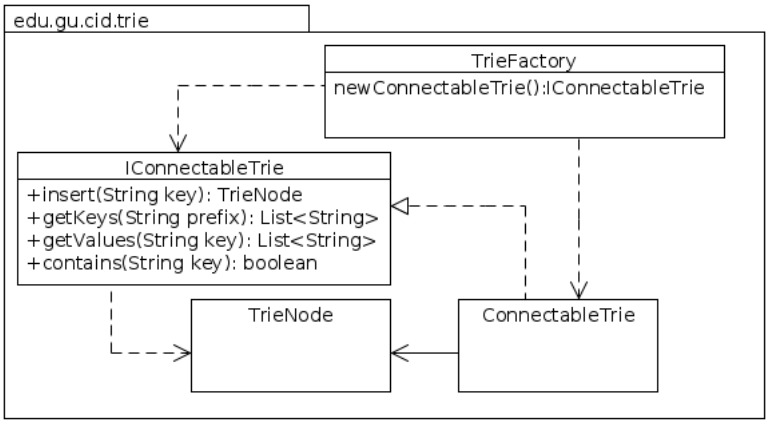


Figur 3: Två hopkopplade Trie:er

Implementationsmässigt är Trie:en uppbyggd av s.k. noder (klassen TrieNode). Varje tecken i ett ord finns i en nod och noderna är ihopkopplade med varann (heldragna linjer). Det som är speciellt med vår Trie är att den kan kop-

plas ihop med en likadan Trie (streckade pilarna). Genom att koppla ihop noder i en Trie med en annan kommer vi att från utgångsspråket (Svenska/Engelska) kunna hitta översättningar i målspråket (Engelska/Svenska)[1].

4 Designmodell



Figur 4: Designmodell för Trie modulen (UML klass diagram)

Se Figur 4. Fabriken är till för att producera Trie:er. Skall senare användas av Dictionary.

5 Modulens gränssnitt

Modulens gränssnittet ges av IConnectableTrie. Följande gäller

- insert, Skall lägga till ett ord i Trie:en. Returnerar en referens till noden med den sista bokstaven i ordet (detta gör att vi kan koppla noden till en nod i en annan Trie).
- contains, svara sant om strängen finns i Trie:en annars falskt.
- getKeys, Ger alla ord i Trie:en som inleds med parametern prefix.
- getValues. Om Trie:en är hopkopplad med en annan Trie ger metoden alla strängar (översättningar) från den andra som är kopplade till parametern key i denna Trie (t.ex. ovan: apa ger ape och monkey). Annars ger metoden en tom lista.

TIPS Ett sätt att markera vart ett ord slutar är att låta slutnoden ha sig själv som "översättning" (peka på sig själv).

[1] Frivilligt: Hur hantera synonymer?

6 Implementation

Allt du behöver finns att ladda ner som ett Eclipse-projekt, hämta från kurssidan. Packa upp och importera till Eclipse. Trie:en skall byggas som en länkad datastruktur bestående av TrieNoder.

1. Fundera vilka referenser (associationer) som behövs i TrieNode. Se till att dessa finns som attribut. Generellt gäller att TrieNode skall ligga på en lägre abstraktionsnivå än ConnectableTrie, TrieNode skall inte ha någon logik bara hantera sin egen data och svara på enkla frågor (set/get/has/is).

2. Vilka referenser behövs i ConnectableTrie?

3. Börja med att implementera metoderna insert och contains i ConnectableTrie. Arbeta parallellt med TrieNode. Testa kontinuerligt!

   Tips Allt är inte rekursion, tänk till!

   Testning och avlusning: Använd den färdiga JUnit testen (vissa saker måste avkommenteras och ev. modifieras beroende på hur ni implementerar saker). Ta bort tester som inte skall köras genom att kommentera bort @Test (eller använd @Ignore). Vid problem kör testen i avlusaren (debug).

4. Fortsätt med getKeys och därefter getValues (som kräver två hopkopplade Trie:er, se JUnit-testen). Identifiera lämpliga hjälpmetoder.

5. För varje metod: Skriv en kommentar före metoden med ett kort resonemang som styrker korrektheten (skriv på naturligt språk). Finns några invarianter? Vad kan med säkerhet sägas efter någon loop?

7 Redovisning

Körningsgodkännande samt kodinspektion. Görs under laborationspassen. Se till att bli avprickad. Följande kommer att kontrolleras:

- Allmän kodstil, indentering, krullparenteser, hårdkodade värden, ...(Eclipse kan hjälpa till)
- Implementationen skall klara alla tester i den bifogade JUnit-testen.
- Inga varningar skall finnas.
- Metodkommentarer med resonemang skall vara av tillräckligt hög kvalitet.

Inlämningsdatum Se kurssidan.