Laboration 1: En Lexikonmodul

1 Syfte

- Programmeringsmetodik och testdriven utveckling.
- Att arbeta med OO-design och därmed komma i kontakt med viktiga begrepp som modularitet, programmering mot gränssnitt, beroenden, m.m.

2 Uppgift

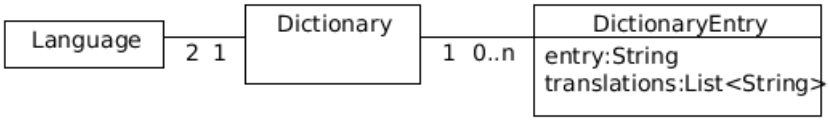
Vi skall implementera en lexikon-modul för vidare användning i laboration 2 där vi lägger på ett GUI. Slutprodukten kommer att se ut ungefär som nedan. I denna laboration koncentrerar vi oss på innandömet.

3 Funktionalitet

Modulen skall klara att översätta ett antal (ca 10 st) ord mellan Svenska och Engelska. För *varje bokstav* användaren anger visar lexikonet ett antal tänkbara ord i utgångsspråket och ett antal möjliga översättningar av dessa, se bild ovan.

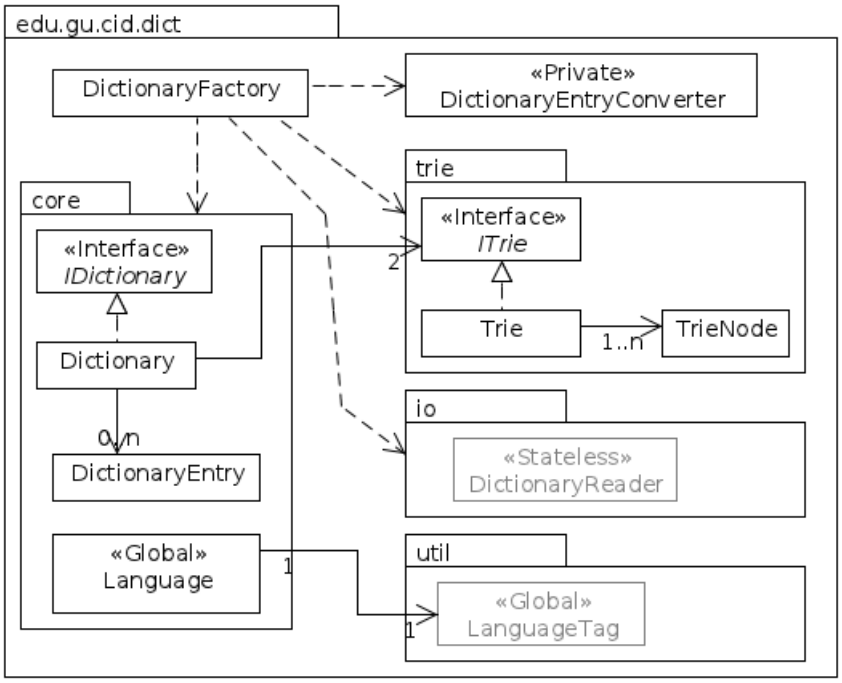
4 Domänmodell

Modellen är mycket enkel. Lexikonet består av ett antal DictionaryEntries vilka håller ordet i utgångsspråket och en lista med översättningar för målspråket. Dessutom behövs från och tillspråk (Language).



5 Designmodell

Utöver modellens klasser behövs ett antal "tekniska" designklasser. Modulen är uppbyggd av ett huvudpaket (i "edu.gu.cid.dict" ersätter du cid med ditt login) och ett antal underpaket. Pilar visar vad som använder/är beroende av vad (alla pilar visas inte).



- I huvudpaketet ligger fabriken som producerar lexikon-objekt och en paketprivat hjälpklass som omvandlar mellan sträng och DictionaryEntry.
- Paketet core innehåller själva domänmodellen. IDictionary är gränssnittet till modellen. Dictionary är implementationen av IDictionary (själva lexikonet). Dictionary använder två s.k. Trie:er (Traj:er) för att lagra ord.

- Trie-paketet innehåller Trie-klassen, gränssnitt till denna samt en hjälpklass. Används bara intern i modulen och skulle behöva vara privat, men något sådant finns inte i Java[1].
- io innehåller klassen DictionaryReader som sköter inläsning av lexikonet från fil eller över nätet till Trie:erna.
- util-paketet innehåller "allmänna" eventuellt återvinningsbara klasser. LanguagTag är en enum med konstanter för olika språknamn (t.ex. us_EN för engelska från USA)[2]. Används internt av bl.a. av Language. Kommer att dyka upp "överallt" (även i kod som använder modulen) därför markerad som global (på samma sätt som Language)

Användaren av modulen använder bara interfacet IDictionary och fabriken.

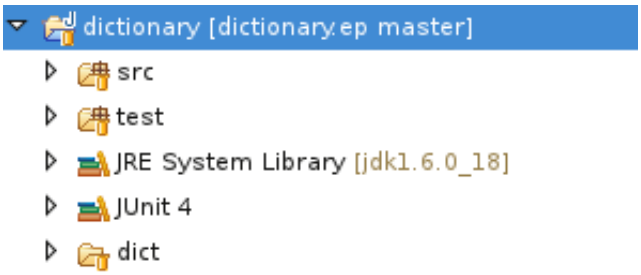
6 Metod

Övergripande kommer vi att göra följande.

1. Inspektion av DictionaryReader och skapa "converter"-klassen.
2. Implementera strukturen för att lagra ord, klasserna Trie och TrieNode. Testa.
3. Implementera IDictionary och därefter Dictionary. Testa Dictionary.
4. Implementera fabriken (trivial inga tester behövs).

7 Utvecklingsmiljön

Skapa ett Eclipseprojekt med följande struktur[3].



- src är en s.k. source folder d.v.s. där ligger *.java filer (skapas då projektet skapas)

[1] Kan ev. lösas med inre klasser men sådana har vi inte gått igenom)
[2] Se vidare http://docs.oracle.com/javase/6/docs/api/java/util/Locale.html#toString%28%29
[3] De små gula cylindrarna kommer inte att finnas, de beror på att jag använder ett versionshanteringssystem)

- test är också en source folder. I denna lägger vi all kod för tester. Skapa genom att högerklicka > New > Source Folder. Testfoldern skall ha samma paketstruktur som src (annars kommer vi inte åt att testa paketprivata klasser).
- dict är en mapp som innehåller textfiler med ord och översättningar (finns färdig).
- JUnit4 är ett bibliotek för enhetstestning. Om det saknas Högerklicka > Build Path > Add Libraries...

För att exakt se vad projektet innehålle t.ex. var *.class-filerna finns kan man använda fliken Navigator.

Uppgift 1 Översätt designmodellen ovan till paket, "tomma" klasser och gränssnitt. Lägg till allt i projektet (finns vissa givna klasser att hämta på kursidan se src_out.zip, test_out.zip och dict.zip). Målet är att projektet skall gå att kompilera. Alla klasser som inleds med Test.., är JUnit-tester och skall ligga i source-foldern test.

8 DictionaryReader

Inspektera DictionaryReader. Klassen får en URI till lexikon-datan (eftersom vi vill kunna hämta data över nätet).

Uppgift 1 Det finns en färdig testklass för DictionaryReader, TestDictionaryReader. Inspektera denna och kör testerna (läs dokumentationen). Försök att lägga till en egen testmetod (behöver inte vara speciellt genomtänkt, bara för att prova hur man gör, läs in och kolla om något ord finns t.ex.).

Uppgift 2 Man kan avlusa (debugga) tester. Provat detta!

Vi låter inte modulen hantera några fel utan skickar dessa vidare till användaren.

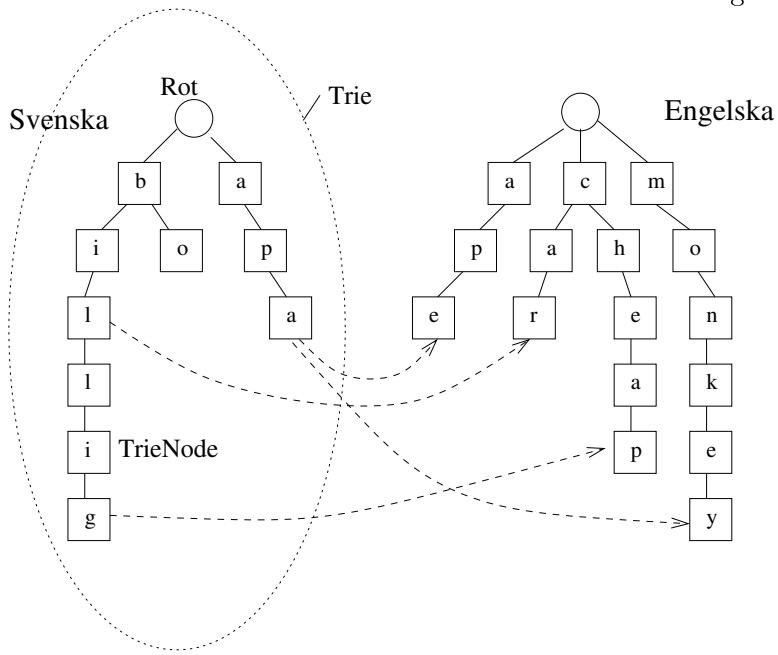
9 Trie

Lexikondata finns i enkla textfiler t.ex sv_SV2en_US.dict (svenska till amerikansk engelska). Då man väljer språk läser programmet in orden i filen (allt) och sparar ord och översättningar i två Trie:er (upp- och nedvända träd), se t.ex. http://en.wikipedia.org/wiki/Trie. Varje TrieNode innehåller ett tecken (bokstav) samt referenser uppåt och nedåt till andra noder i Trien, se figur nedan (heldragna linjer)[4].

Om ett ord i utgångsspråkets ingår i lexikonet finns det en referens till ett ord i målspråket (referenser mellan Trie:erna). T.ex, bil → car (streckade pilar). Ord som inte ingår saknar referens, t.ex. bill (ingen streckad linje). Ett ord kan ha många översättningar (apa →ape, monkey) och tvärt om (synonymer). [5]

[4] Noden ovanför en nod brukar kallas parent och de nedanför children.
[5] Frivilligt: Hur hantera synonymer i utgångsspråket?

Med denna representation blir det lätt att hitta alla ord som inleds med en viss teckensekvens. Dessutom är det ett effektivt sätt att lagra orden på.



Uppgift 1 Studera bilden ovan. Vilken data måste finnas i TrieNoderna för att dessa skall kunna kopplas ihop till en Trie och för att man skall kunna manövrera i Trie:n? Klassen skall vara enkel och innehåller i princip bara set och get metoder. Eftersom klassen är så enkel behövs inga tester. Gör klart klassen.

Uppgift 2 Försök hitta några representationsinvarianter för Trie:n. Skriv ner dessa i en klasskommentar när ni skapar Trie-klassen (om vi inte hunnit gå igenom detta så avvakta...).

Uppgift 3 Skapa klassen Trie (låt gränssnittet vara så länge). Trie:n skall bara ha en referens till "rot"-noden d.v.s. den översta TrieNoden. Denna nod innehåller ingen data utan är bara själva ingången till trädstrukturen. Vi behöver metoder för att;

- Lägga till ord (skall heta add och ta en sträng som parameter)
- Läsa ut ett ord (returtyp string).
- Skapa en lista med alla ord som inleds med en viss sekvens.

Implementera en metod i taget och testa denna bl.a. m.h.a. representationsinvarianterna. För metoden add finns en påbörjad test (TestTrie.testAdd()).

TIPS: Låt metoden som lägger till ett ord i Trie:en returnera en referens till den sist insatta noden, underlättar i fortsättningen.

Uppgift 4 Gör klart gränssnittet till Trie (ITrie som skall användas av Dictionary) och lägg till en factory-method i Trie.

10 Dictionary

Vi fortsätter med själva lexikonklassen, Dictionary.

Uppgift 1 Fundera på vilka metoder som kan behövas i gränssnittet? Vad vore bekvämnt för en användare av modulen? Implementera gränssnittet IDictionary.

Uppgift 2 Gör klart klassen Dictionary. Parametrisera konstruktorn med all data som behövs för att bygga de två Trie:erna. Implementera en metod i taget och testa. Vid behov bryt ner metoder i flera små metoder (och testa).

Uppgit 3 Tänk igenom testning. Har vi verkligen täckt alla fall?? Försök hitta alla specialfall. Kör ett kodtäckningsverktyg, täcker testerna all kod?

11 Fabriken

Uppgift 1 Implementera fabriken som levererar objekt vilka implementerar IDictionary. Hur skall man få tag i fabriken (fabrikobjektet)? Vilken indata behöver fabriken? Användaren skall som sagt bara kunna använda fabriken för att skapa nya lexikon.

Uppgit 2 Clean up. Jämför din kod med designmodellen ovan. Stämmer allt, paket, pilar? Kodstil, gå igenom! Tänk till på allt vi gått igenom i designväg, vilken kvalité har din kod? Kanske den utdelade koden har något problem...? Kör Findbugs.

12 Redovisning

Körningsgodkännande samt kodgenomgång. Görs under laborationspassen. Se till att bli avprickad.

Inlämningsdatum Se kurssidan.