Laboration 1 del 1: En översättare

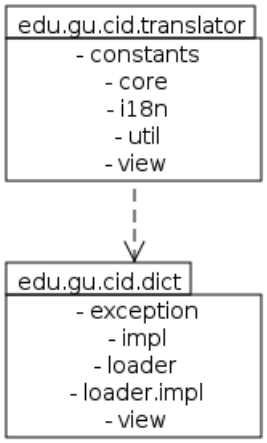
1 Syfte

- Att introducera utvecklingsmiljön och något större program.
- Testdriven utveckling.
- Att arbeta med OO-design och därmed komma i kontakt med viktiga begrepp som modularitet, programmering mot gränssnitt, beroenden, m.m.
- Att ge en kort introduktion till specifikation och verifikation.

2 Uppgift

Ni skall implementera en prototyp till en översättare tänkt för mobila plattformar ungefär som i Figur 1. I denna del koncentrerar vi oss på innandömet (GUI m.m. kommer i del 2).

Figur 1: Engelska till Svenska



Figur 2: Övergripande design (underpaket listade).

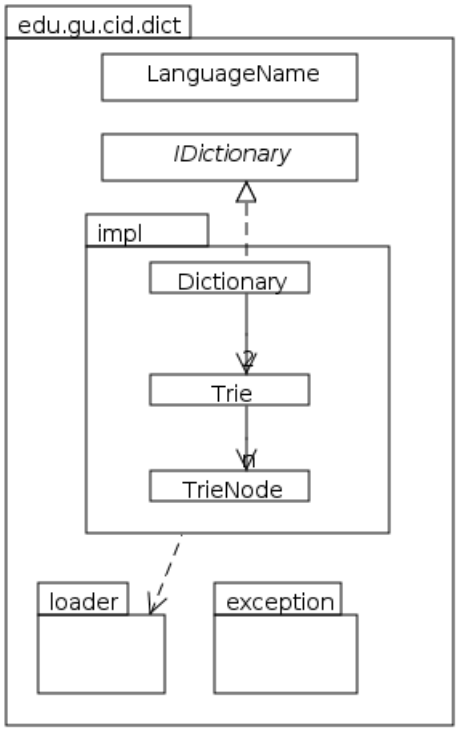
3 Övergripande design

Applikationen kan delas upp i följande delar (som alltså kommer att utvecklas helt separat);

Paketet edu.chl.cid.dict innehåller själva ordboken (dictionary) och

edu.gu.cid.translator innehåller översättaren (som använder ordboken).

4 Design ordboken



Figur 3: Design ordboken.

- IDictionary är gränssnittet som translator kommer att använda.
- LanguageName är en enum med konstanter för olika språknamn. Internt i programmet använder vi språkoberoende språknamn(!) skrivna som land-understreck-region t.ex en_US, engelska från region USA.
- Paketet impl innehåller implementationen av IDictionary samt två hjälpklasser båda paket-privata. Trie är en struktur som lagrar ord i ett språk. En Trie byggs upp med hjälp av TrieNode-objekt. Dictionary använder två Trie:er en för utgångsspråket och en för målspråket.
- loader sköter inläsning av ordboken från fil till Trie:erna.
- exception innehåller en applikationsspecifikt undantagsklass.

Lägg märke till pilarna (alla visas inte)! Vem använder vad och vilka beroenden finns?

4.1 Design mönster

IDictionary och Dictionary skall byggas som en Facade (Facade mönstret).

5 Kravspecifikation

Oklara saker får ni fråga om.

5.1 Funktionalitet

Applikationen 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 Figur 1.

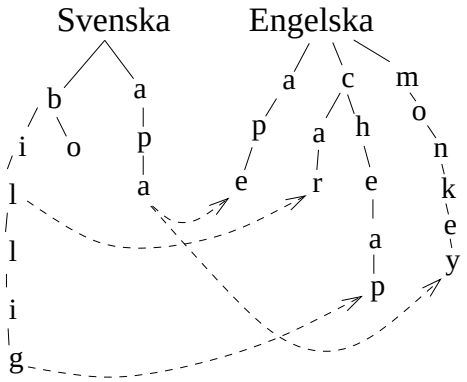
5.2 Datahantering

Ordböcker finns i enkla textfiler t.ex sv_SV2en_US.dict (svenska till amerikansk engelska). Då man väljer språk läser programmet in ordboken (allt) och sparar ord och översättningar i två Trie:er, 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)[1].

Om ett ord i utgångsspråkets ingår i ordboken 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. [2]

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.



Figur 4: De två Trie:erna (delar av) för Svenska till Engelska.

5.3 Felhantering

Se vidare kommentarer i DictionaryException.

5.4 Specifikation och Verifikation

Se utvecklingsprocess.

[1] Noden ovanför en nod brukar kallas parent och de nedanför children. Hela strukturen kallas ett träd.

[2] Frivilligt: Hur hantera synonymer i utgångsspråket?

6 Utvecklingsprocess

OBS! Detta blir inte ett helt program (main-metod saknas, vi kör koden i JUnit tester).

Följande process rekommenderas;

1. Hämta hem startkod för labben från kurssidan. Packa upp och importera till Eclipse (File > Import > General > Existing Project into Workspace > Browse ... > Finish). Inspektera projektet.

2. Börja med att köra testen för filhanteringen, se kommentarer i fil. Prova att skapa någon egen enkel testmetod.

3. Implementera och testa Trie-och TrieNode-klassen (fundera över ansvarsfördelningen mellan dessa, responsibilities?). Innan du börjar koda ge minst ett svar på följande;

   a) Vilka operationer kommer Dictionary att behöva (hur använder Dictionary Trie?)? In och utdata?

   b) Vilka representationsinvarianter finns för Trie? Skriv ner dessa som en klasskommentar. Notera för varje muterande metod hur invarianterna upprätthålls.

4. Det finns en påbörjad test för Trieklassen. Skapa en test för någon representationsinvariant (att användas av andra tester senare).

5. Skapa en test för att kontrollera att ett Trie-objekt uppfyller (alla) RI direkt efter instansiering.

   TIPS Man kan debugga testerna.

6. Fortsätt tills Trie fungerar och är genomtestad (code coverage)

7. Vilka metoder skall IDictionary innehålla? Tänk igenom (hög abstraktionsnivå)! Implementera därefter klassen Dictionary. Dictionary **skall** ha en "factory metod" som returnerar typen IDictionary (Factory/Facade-mönstren). Se t.ex http://en.wikipedia.org/wiki/Factory_method.

8. Skapa en ny test för Dictionary (File > New > JUnit Test Case). Implementera och testa.

9. När Dictionary är klar kör den färdiga testsviten (kontrollera namn på ingående klasser). Har du testat alla extremfall??

10. Höj kodkvalitén. Kör Findbugs och JDepend!

7 Redovisning

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

Inlämningsdatum Se kurssidan.