Problema 3.3: El diccionari (Proposta de solució)

Josep M. Ribó

19 de desembre de 2008

1 Objectius

- Dissenyar estructures de dades complexes

Per fer aquest problema, cal que us llegiu: • Els apunts del tema 3

2 `Diccionari` (Febrer-07)

Volem dissenyar una estructura de dades per contenir un diccionari. Al diccionari s'hi emmagatzemaran paraules (que poden ser noms, adjectius o verbs). Cada tipus de paraula tindrà els seus atributs específics:

- Totes tenen un significat
- Els verbs tenen una conjugació (1a, 2a o 3a)
- Els noms tenen un gènere (masc., fem.)
- Els adjectius poden ser qualificatius o no-qualificatius.
- Una paraula es pot relacionar amb altres com s'explica més endavant.

Aquesta estructura de dades la modelitzarem mitjançant una classe que anomenarem `Dictionary` i que oferirà, entre altres, les operacions següents:

- `void putWord(...)`

  Afegeix una paraula al diccionari. Aquesta paraula pot ser de qualsevol dels tipus indicats més amunt.

- `void relatePairOfWords(...)`

  Aquesta operació relaciona dues paraules que ja estan inserides al diccionari. Aquesta relació, en general, és de camp semàntic (per exemple, podem relacionar paraules com *gat, gos...* o bé, d'altres com *empassar, engolir...*). El motiu pel qual dues paraules queden relacionades no importa en la resolució d'aquest problema. Per suposat, una paraula pot estar relacionada amb moltes altres.

  En cas que alguna de les paraules que es volen relacionar no formi part del diccionari, caldrà llençar una excepció.

- `Word* getInfoWord(...)`

  Retorna la informació relativa a una paraula emmagatzemada al diccionari (literal, significat i elements específics de cada tipus de paraula: conjugació, gènere...).

- `void getListOfRelatedWords(...)`

  Obté com a paràmetre de sortida una llista amb les paraules relacionades amb una determinada (només obté les paraula, no la seva informació associada).

  Nota: El client de la classe `Dictionary` no ha de fer cap hipòtesi sobre quin tipus específic de classe `List` s'ha usat per inserir les paraules relacionades.

- `void getListOfWords(..)`

  Obté com a paràmetre de sortida una llista amb totes les paraules del diccionari ordenades alfabèticament.

Es demana:

1. (1 punt) Situa les classes `Word, Verb, Adjective, Noun` i `Dictionary` a la jerarquia de classes de l'assignatura

   - `Dictionary` és subclasse de `Object`.
   - `Word` és subclasse de `Object`
   - `Adjective, Verb` i `Noun` són subclasses de `Word`

2. (2.5 punts) Implementa les classes `Word, Verb`. En particular, es demana:

   - Representar els atributs que han de tenir totes dues classes (podeu fer-los públics, si voleu)
   - Les capceleres d'operacions que han d'oferir (sense implementar-les)

   Classe `Word`:

```
class Word :public Object{
protected:
  MyString literal;
  MyString meaning;

public:

  Word();
  Word(const MyString& lit, const MyString& mean);
  virtual ~Word(){}

  void setLiteral(const MyString& lit);
  void setMeaning(const MyString& mean);
  void getLiteral(MyString& lit) const;
  void getMeaning(MyString& mean) const;

  MyString toString() const;

};


Word::Word()
:literal(""),meaning("")
{}

Word::Word(const MyString& lit, const MyString& mean)
{
```

```
 literal.copy(lit);
 meaning.copy(mean);
}

 void Word::setLiteral(const MyString& lit)
{

  literal.copy(lit);
}


void Word::getLiteral(MyString& lit) const
{
  lit.copy(literal);
}

MyString Word::toString() const
{

 return literal+meaning;

}

.....
```

Comentaris:

- L'operació `getInfoWord` no demana la llista de paraules relacionades (vegeu enunciat del problema, més amunt). Sembla més natural que la classe `Word` no tingui associada la llista de paraules relacionades a una determinada. Quan es desitgi aquesta llista es cridarà a l'operació `getListOfRelatedWords`.
- `Word` és una classe abstracta doncs no implementa les operacions heretades de `Object`: `copy`, `clone` i `operator==`. Per tant, heretarà aquestes operacions com a virtuals pures.
- L'enunciat indicava que es podia implementar la classe *fent públics els atributs*. Aquesta consideració era per facilitar-ne la implementació a l'examen (ja que s'evitava la necessitat de les operacions get-set), però el més correcte i adient al paradigma de la POO és implementar-la tal com està fet en aquesta proposta de solució.

  Recordeu que està indicat fer públics els atributs d'una classe quan es compleixen simultàniament dues condicions:

  (a) La classe és molt senzilla
  (b) La classe es dissenya per donar servei a la implementació d'una altra classe i no està previst mostrar-la als clients

  Aquestes dues condicions les compleixen les classes `Node` de `LinkedList` o de `HashMap`. La classe `Word` només compleix la primera.

Classe `Verb`:

```
class Verb :public Word{

  int conj;
```

```
public:

  Verb();
  Verb(const MyString& lit, const MyString& mean, int c);
  virtual ~Verb(){}

  void setConj(int c);
  int getConj() const;

  Object* clone() const;
  void copy(const Object& obj);
  bool operator==(const Object& obj) const;
  MyString toString() const;

};


Verb::Verb()
{conj=0;}

Verb::Verb(const MyString& lit, const MyString& mean, int c)
:Word(lit,mean)
{
 conj = c;
}

Object* Verb::clone() const{

  Verb* aux = new Verb();
  aux->copy(*this);
  return aux;
}

void Verb::copy(const Object& obj){


  const Verb& vaux = dynamic_cast<const Verb&>(obj);

  literal.copy(vaux.literal);
  meaning.copy(vaux.menaning);
  conj = vaux.conj;
}


MyString Verb::toString() const{

  return Word::toString()+MyString(conj);
}


.....
```

3. (1.5 punts) Proposa una interfície per la classe `Dictionary` (o sigui: proposa una capcelera per les operacions enumerades més amunt). Especifica també les excepcions a la/les operacions que les necessitin.

```
class Dictionary :public Object{

//Representacio.....

public:
 //...Constructores i altres ops......

 void putWord(const Word& w);
 void relatePairOfWords(const MyString& lit1,const MyString& lit2)
                              throw (NonExistingWordException);
 Word* getInfoWord(const MyString& lit, bool& found) const;
 List<MyString>* getListOfRelatedWords(const MyString& lit) const;
 List<MyString>* getListOfWords() const;

};
```

Comentaris:

- Per relacionar dues paraules (op. `relatePairOfWords`) és millor usar els literals que les pròpies paraules amb tota la seva informació associada. S'evita una redundància inútil.

  En tot cas, les dues paraules que es volen relacionar han d'haver estat introduïdes prèviament al diccionari amb l'operació `putWord`. En cas contrari es llençarà l'excepció `NonExistingWordException`.

- A les operacions `getListOfRelatedWords` i `getListOfWOrds` el client de la classe espera una llista. Aquesta llista pot ser `ArrayList, LinkedList...` (en realitat, al client li és igual quina).

  D'igual manera, a l'operació `getInfoWord`, el client espera una paraula. Però aquesta paraula pot ser de classe `Verb, Adjective` o `Noun`. El client no sap, a priori, de quin tipus serà la paraula retornada (per exemple: `d.getInfoWord(MyString("menjar"))` retornarà un `Verb` , mentre que `d.getInfoWord(MyString("finestra"))` retornarà un `Noun`.

  En aquestes situacions en que una acció/operació ha de retornar un objecte però el tipus d'aquell objecte només es coneixerà en temps d'execució, el més adient és **retornar un apuntador a la classe més general que pugui ser retornada**, fins i tot si aquella classe és abstracta.

  No hi ha cap problema en crear un apuntador a una classe abstracta. El que no es pot fer és crear un objecte d'una classe abstracta.

4. (2 punts) Proposa una representació per la classe `Dictionary`. Indica molt clarament el significat de la/les estructures de dades que intervinguin en aquesta representació. Ajuda't d'un dibuix.

   Procediment per determinar la representació d'una classe complexa:

   (a) *Determinar les operacions consultores de la classe.*

   Les operacions consultores han d'executar-se eficientment. Proposarem una representació per a la classe que atorgui la màxima eficiència possible a les operacions consultores. Podem dir que les operacions consultores guien la representació.

   Operacions consultores de la classe `Dictionary`:

   - getInfoWord
   - getListOfRelatedWords
   - getListOfWords()

   (b) *Per cadascuna de les operacions consultores obtingudes a 1, determinar la clau mitjançant la qual aquestes operacions accedeixen a la informació requerida*

   Habitualment, la clau serà de classe `MyString`. Però cal determinar el significat conceptual d'aquella clau (nif, matrícula, paraula...).

- getInfoWord ⟶ literal de la paraula que es vol cercar al diccionari (MyString)
- getListOfRelatedWords ⟶ literal de la paraula de la que es vol cercar les paraules relacionades (MyString)
- getListOfWords() ⟶ En aquest cas no hi ha clau

(c) *Dissenyar una taula per cadascuna de les claus trobades a 2*

- `tinfoword`

Clau	Valor	Implementació
Literal de la paraula (`MyString`)	Apuntador a la informació guardada guardada sobre la paraula (`Word*`)	BSTMap

Comentaris:

- No posem com a valor directament `Word` perquè aleshores no podríem encabir-hi adjectius, verbs o noms. És preferible posar `Word*` i d'aquesta manera podrem posar a cada element de la taula, indistintament un apuntador a `Adjective`, a `Noun` o a `Verb`.
- Optem per la implementació de la taula en forma de `BSTMap` per tal de facilitar l'obtenció de les paraules ordenades alfabèticament.

- `trelatedwords`

Clau	Valor	Implementació
Literal de la paraula (`MyString`)	Llista de tots els literals relacionats amb la clau (`LinkedList<MyString>`)	BSTMap o THashMap

(d) *Si és possible, fusionar algunes de les taules del pas 3 per tal de minimitzar el nombre de taules de què constarà la representació de la classe*

Una condició necessària per tal que dues taules es puguin fusionar en una de sola és que la clau per totes dues tingui el mateix significat conceptual (**no que sigui de la mateixa classe sinó que tingui el mateix significat**). Exemple: 2 claus nif, 2 claus nom de ciutat, 2 claus literal de paraula (com és aquest cas).

Per fusionar dues taules crearem una nova classe auxiliar que contingui com a atributs els valors de les dues taules que volem fusionar.

En aquest cas podem fusionar les dues taules dissenyades prèviament creant una classe `WordInfoRel` que emmagatzemi per cada paraula:

- La seva informació associada (significat, literal i info. específica segons el tipus de paraula)
- La llista dels literals relacionats amb aquesta paraula.

```
class WordInfoRel{
public:

 Word* pword;
 LinkedList<MyString> lrel;
};
```

Després d'aquesta fusió quedaria una sola taula:

- `tword`

Clau	Valor	Implementació
Literal de la paraula (`MyString`)	Info paraula i llista lit. relacionats (`WordInfoRel`)	BSTMap

(e) *Implementar la classe complexa en termes de les taules que han resultat d'aquest procés i de la resta d'atributs que siguin necessaris*

```
class Dictionary :public Object{

 BSTMap<WordInfoRel> tword;

public:

//....

};
```

5. (1.5 punts) Implementa l'operació `getListOfRelatedWords`

   Implementarem aproximadament algunes operacions de la classe `Dictionary`

```
void Dictionary::putWord(const Word& w){
  MyString lit;
  WordInfoRel wir;
  BSTMapIterator<WordInfoRel> it;

  w.getLiteral(lit);
  tword.get(lit,wir,found);

  if (found){
    delete wir.pword;
    wir.pword = w.clone();
  }
  else{
    wir.pword = w.clone();
  }
  tword.put(lit,wir,it);
}

Word* Dictionary::getInfoWord(const MyString& lit, bool& found) const
{
  WordInfoRel wir;

  tword.get(lit,wir,found);

  if (found){ return (wir.pword)->clone();}
  else return NULL;
}

List<MyString>* Dictionary::getListOfRelatedWords(const MyString& lit)
const{
   WordInfoRel wir;
   bool found;

   tword.get(lit,wir,found);

   if (found) return wir.lrel.clone();
   else return new LinkedList<MyString>;
}
```

6. (1.5 punt) Si la llista de paraules relacionades hagués de retornar una llista amb tota la informació de cada paraula i vulguéssim usar la representació més eficient possible, quins canvis hauríem de fer? Dóna la idea d'aques