

DictionaryManagement

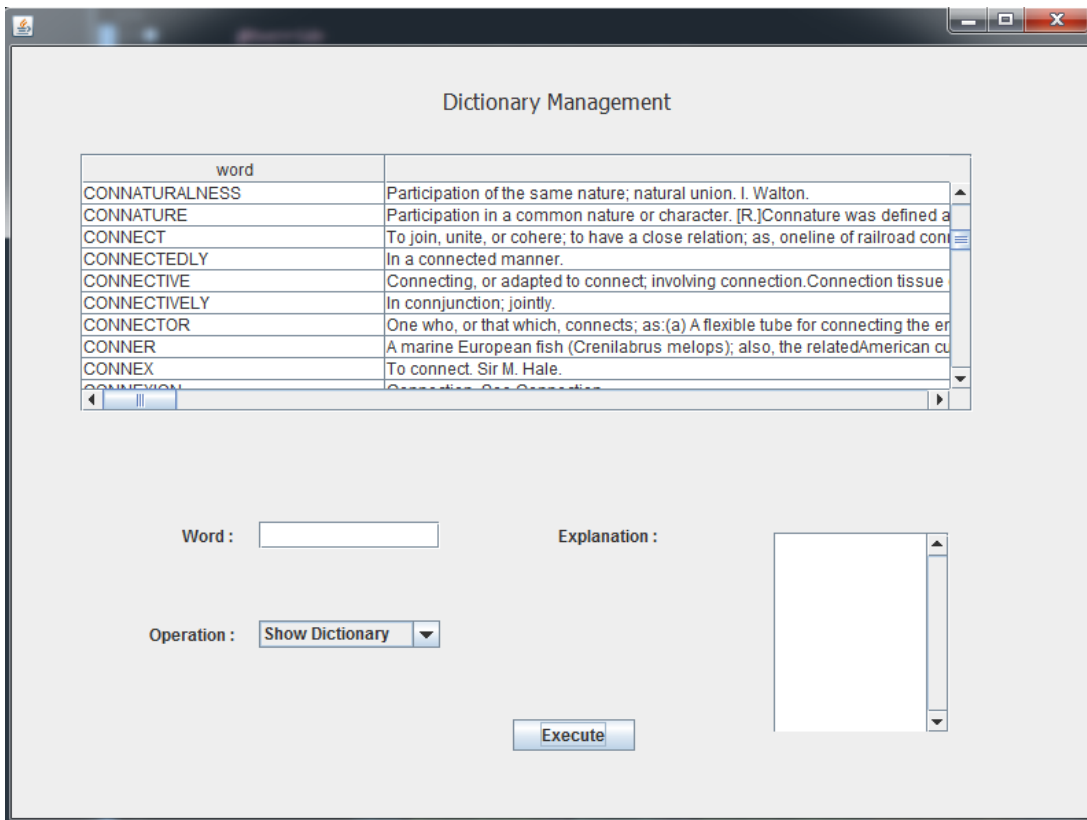
Student: Maxim Bogdan-Gheorghe

Cuprins

1. Obiectivul temei.....	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare	4
3. Proiectare.....	5
4. Implementare și testare.....	16
5. Rezultate	17
6. Concluzii și dezvoltării ulterioare	17
7. Bibliografie.....	1818

1. Obiectivul temei

Obiectivul temei este proiectarea, implementarea și realizarea unei aplicații menite pentru a gestiona un dicționar al limbii engleze. Aplicația poate fi folosită atât de un specialist, cât și de o persoană obișnuită care dorește să se documenteze asupra unor definiții generale al unor cuvinte din limba engleză. De asemenea, utilizatorul poate adăuga cuvinte noi, șterge cuvinte vechi, precum și salva și încărca starea dicționarului.



The screenshot shows a window titled "Dictionary Management". It contains a table with two columns: "word" and "Explanation". The table lists several words related to "connect", such as "CONNATURALNESS", "CONNATURE", "CONNECT", "CONNECTEDLY", "CONNECTIVE", "CONNECTIVELY", "CONNECTOR", "CONNER", "CONNEX", and "CONNECTION". Below the table, there are input fields for "Word :" and "Explanation :", a dropdown menu for "Operation :" with "Show Dictionary" selected, and an "Execute" button.

word	Explanation
CONNATURALNESS	Participation of the same nature; natural union. I. Walton.
CONNATURE	Participation in a common nature or character. [R.]Connature was defined a
CONNECT	To join, unite, or cohere; to have a close relation; as, oneline of railroad con
CONNECTEDLY	In a connected manner.
CONNECTIVE	Connecting, or adapted to connect; involving connection.Connection tissue
CONNECTIVELY	In conjunction; jointly.
CONNECTOR	One who, or that which, connects; as:(a) A flexible tube for connecting the er
CONNER	A marine European fish (Crenilabrus melops); also, the relatedAmerican cu
CONNEX	To connect. Sir M. Hale.
CONNECTION	Connecting. See Connection.

Word :

Explanation :

Operation : Show Dictionary

Execute

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Primul pas efectuat în analiza și modelarea problemei este cel de a identifica substantivele principale din cerință pentru a putea stabili clasele utilizate. Dicționarul este cel mai general substantiv care poate fi reprezentat drept clasă, prin urmare acesta va fi transpus primul. Cuvintele vor fi păstrate într-o asociere cu definițiile acestora, iar structura de date utilizată pentru a memora aceste asocieri va fi principala structură de date modificabilă. Din clasa Dictionary se va facilita accesul direct asupra acesteia. Pentru a permite extinderea rapidă a operațiilor se va folosi șablonul de proiectare : “decorator”. Astfel, creăm clasa abstractă OperationDecorator care are ca rol menținerea unei referințe către dicționarul propriu-zis și efectuarea operațiilor asupra acestuia.

În continuare, proiectăm clase pentru restul operațiilor :

-AddWord

-RemoveWord

-WordMatcher

-PopulateDictionary

-SaveDictionary

-LoadDictionary

Clasa AddWord are rolul de a adăuga un cuvânt în dicționar. Clasa RemoveWord are rolul de a șterge un cuvânt din dicționar. Clasa WordMatcher va găsi un cuvânt în dicționarul dat după un anumit șablon. PopulateDictionary va analiza un fișier JSON din care va extrage un șir de caractere ce reprezintă un dicționar deja format din cuvinte și definiții. Aceasta va încărca apoi clasa Dictionary cu șirul nou format. Clasa SaveDictionary va salva dicționarul curent în fișierul dicstate.ser, iar clasa LoadDictionary va încărca din fișierul respectiv. Astfel, operațiile de bază efectuate pe dicționar sunt îmbuătățite prin adăugarea de operații noi. De asemenea, creerea și implementarea altor operații necesită un timp suplimentar minim, precum și o putere de muncă limitată. Ca și tehnici, am folosit design by contract pentru a realiza un anumit contract cu utilizatorul unei metode, serializare pentru a salva și încărca starea unui obiect și analizare și retragere de informații dintr-un fișier JSON pentru a popula dicționarul. Scenariile de utilizare pot varia, atât pentru oameni de rând, cât și pentru specialiști. De asemenea, programul poate fi transformat foarte ușor în orice tip de dicționar prin ștergerea dicționarului curent și introducerea



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

termenilor specifici. De asemenea softul se poate utiliza și în școli, pentru a ajuta elevii/studentii atât în căutarea lor de cuvinte noi, cât și în personalizarea unui vocabular propriu care poate fi implementat cu ușurință datorită flexibilității șablonului de proiectare ales.

3.Proiectare

Pentru prezentarea proiectării aplicației, continuăm prin a prezenta tehnicile de programare utilizate :

Design by contract :

- reprezinta un "contract" care specifică restricțiile la care trebuie să se supună datele de intrare ale unei metode, valorile posibile de ieșire și stările în care se poate afla programul - aceste restricții sunt date sub forma unor:

a) precondiții: reprezintă obligațiile pe care datele de intrare ale unei metode trebuie să le respecte pentru ca metoda să funcționeze corect

b) postcondiții: reprezintă garanțiile pe care datele de ieșire ale unei metode le oferă

c) invariante: reprezintă condiții impuse stărilor în care programul se poate afla la un moment dat

Cel care a implementat clasa sau metoda respectivă îi spune utilizatorului ce date sunt considerate valide ca date de intrare. Aceasta îl scuteste să folosească teste de validare a datelor, care în unele cazuri doar ar încetini algoritmul, ele fiind redundante cu teste care deja sunt făcute pentru validarea datelor, de exemplu imediat ce au fost introduse de către utilizator.

În același timp utilizatorul știe din contract la ce date posibile de ieșire sau stări intermediare să se aștepte și atunci poate să-și optimizeze propriul cod în funcție de acestea

Contractul unei clase sau al unei metode se specifică literar, printr-un text, inclus în program ca un comentariu sau scris în documentația aferentă - în faza de dezvoltare a unui program se pot folosi instrucțiuni care să testeze îndeplinirea contractului, instrucțiuni care să fie scoase pe urmă (manual sau automat) din codul final.

O aserțiune generează o excepție specială cu textul dat după ":", în cazul în care condiția ei este falsă. Deoarece aserțiunile au fost introduse mai târziu în limbajul Java trebuie specificat la opțiunile de compilare: -source 1.4 - implicit aserțiunile nu fac nimic (sunt dezactivate), ca și cum codul este pregătit pentru livrare. Dacă dorim să activăm aserțiunile trebuie să specificăm la opțiuni: -enableassertions (sau -ea) .

Serializare

Serializarea este o metoda prin care se pot salva, într-o maniera unitara, datele împreuna cu semnatura unui obiect. Folosind aceasta operatie se poate salva într-un fisier, ca sir de octeti, o instanta a unei clase, în orice moment al executiei. De asemenea, obiectul poate fi restaurat din fisierul în care a fost salvat în urma unei operatii de serializare.

Utilitatea serializarii constă în următoarele aspecte:

Asigură un mecanism *simplu de utilizat* pentru salvarea și restaurarea a datelor.

Permite *persistența obiectelor*, ceea ce înseamnă că durata de viața a unui obiect nu este determinată de execuția unui program în care acesta este definit - obiectul poate exista și între apelurile programelor care îl folosesc. Acest lucru se realizează prin serializarea obiectului și scrierea lui pe disc înainte de terminarea unui program, apoi, la relansarea programului, obiectul va fi citit de pe disc și starea lui refăcută. Acest tip de persistență a obiectelor se numește *persistența ușoară*, întrucât ea trebuie efectuată explicit de către programator și nu este realizată automat de către sistem.

Compensarea diferențelor între sisteme de operare - transmiterea unor informații între platforme de lucru diferite se realizează unitar, independent de formatul de reprezentare a datelor, ordinea octeților sau alte detalii specifice sistemelor repective.

Transmiterea datelor în rețea - Aplicațiile ce rulează în rețea pot comunica între ele folosind fluxuri pe care sunt trimise, respectiv recepționate, obiecte serializate

RMI (Remote Method Invocation) - este o modalitate prin care metodele unor obiecte de pe o altă mașină pot fi apelate ca și cum acestea ar exista local pe mașina pe care rulează aplicația. Atunci când este trimis un mesaj către un obiect "remote"

(de pe altă mașină), serializarea este utilizată pentru transportul argumentelor prin rețea și pentru returnarea valorilor.

Java Beans - sunt componente reutilizabile, de sine stătătoare ce pot fi utilizate în medii vizuale de dezvoltare a aplicațiilor. Orice componentă Bean are o stare definită de valorile implicite ale proprietăților sale, stare care este specificată în etapa de design a aplicației. Mediile vizuale folosesc mecanismul serializării pentru asigurarea persistenței componentelor Bean.

Un aspect important al serializării este că nu salvează doar imaginea unui obiect ci și toate referințele la alte obiecte pe care acesta le conține. Acesta este un proces recursiv de salvare a datelor, întrucât celelalte obiecte referite de obiectul care se serializează pot referi la rândul lor alte obiecte, și așa mai departe. Așadar referințele care construiesc starea unui obiect formează o întreagă rețea, ceea ce înseamnă că un algoritm general de salvare a stării unui obiect nu este



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

tocmai facil. În cazul în care starea unui obiect este formată doar din valori ale unor variabile de tip primitiv, atunci salvarea informațiilor încapsulate în acel obiect se poate face și prin salvarea pe rând a datelor, folosind clasa `DataOutputStream`, pentru ca apoi să fie restaurate prin metode ale clasei `DataInputStream`, dar, așa cum am văzut, o asemenea abordare nu este în general suficientă, deoarece pot apărea probleme cum ar fi: variabilele membre ale obiectului pot fi instanțe ale altor obiecte, unele câmpuri pot face referință la același obiect, etc.

Salvarea datelor încapsulate într-un obiect se poate face și prin salvarea pe rând a datelor, folosind clasa `DataOutputStream`, pentru ca apoi să fie restaurate prin metode ale clasei `DataInputStream`, dar o asemenea abordare nu este în general suficientă, deoarece pot apărea probleme cum ar fi :

- datele obiectului pot fi instanțe ale altor obiecte
 - în unele cazuri, este necesară și salvarea tipului datei
 - unele câmpuri fac referință la același obiect
- Asadar, prin serializare sunt surprinse atât datele, semnatura clasei (numele metodelor și definiția lor - nu și implementarea) precum și starea obiectului.

Pentru a putea fi serializat un obiect trebuie să fie instanța a unei clase care implementează una din interfețele :

- `java.io.Serializable` sau
- `java.io.Externalizable` (care extinde clasa `Serializable`)

Interfața `Serialize` nu are nici o metodă, ea da doar posibilitatea de a specifica faptul că se dorește ca o anumită clasă să poată fi serializată. Declarația unei astfel de clase ar fi :

```
class ClasaSerializabila implements Serializable {...}
```

În urma serializării obiectele sunt pot fi salvate într-un fișier, în același fișier putând fi salvate și mai multe obiecte. Operațiile de intrare-ieșire la nivelul obiectelor se realizează prin intermediul unor fluxuri de obiecte, implementate de clasele `ObjectInputStream` și `ObjectOutputStream`. Salvarea unui obiect într-un fișier se realizează astfel :

```
MyObject o = new MyObject();  
FileOutputStream fout = new FileOutputStream("fisier");  
ObjectOutputStream sout = new ObjectOutputStream(fout);  
sout.writeObject(o);
```

Restaurarea unui obiect salvat într-un fișier se face într-o manieră asemănătoare:

```
FileInputStream fin = new FileInputStream("fisier");  
ObjectInputStream sin = new ObjectInputStream(fin);  
o = (MyObject) sin.readObject();
```

Pe lângă metodele de scriere/citire a obiectelor cele două clase pun la dispoziție și metode pentru scrierea tipurilor de date primare, astfel încât apeluri ca cele de mai jos sunt permise :

```
FileOutputStream ostream = new FileOutputStream("t.tmp");  
    ObjectOutputStream p = new ObjectOutputStream(ostream);  
    p.writeInt(12345);  
    p.writeObject("Today");  
    p.writeObject(new Date());  
    p.flush();  
    ostream.close();
```

```
FileInputStream istream = new FileInputStream("t.tmp");  
    ObjectInputStream p = new ObjectInputStream(istream);  
    int i = p.readInt();  
    String today = (String)p.readObject();  
    Date date = (Date)p.readObject();  
    istream.close();
```

ObjectInputStream și ObjectOutputStream implementează indirect interfețele DataInput, respectiv DataOutput, interfețe ce declară metode atât pentru scrierea/citirea datelor primitive, cât și pentru scrierea/citirea obiectelor. Pentru transferul obiectelor sunt folosite metodele:



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

```
final void writeObject( java.lang.Object obj )  
    throws java.io.IOException  
  
final java.lang.Object readObject( )  
    throws java.io.OptionalDataException,  
           java.lang.ClassNotFoundException, java.io.IOException
```

Acestea apeleaza la rândul lor metodele implicite de transfer `defaultWriteObject` și `defaultReadObject` (având aceleasi semnături ca mai sus).

Personalizarea serializării se realizeaza prin supradefinirea (într-o clasa serializabila!) a metodelor `writeObject` și `readObject`, modificând astfel acțiunea lor implicită.

Metoda `writeObject` controleaza ce date sunt salvate și este uzual folosita pentru a adauga informatii suplimentare la cele scrise implicit de metoda `defaultWriteObject`.

Metoda `readObject` controleaza modul în care sunt restaurate obiectele, citind informațiile salvate și, eventual, modificând starea obiectelor citite astfel încât ele să corespundă anumitor cerințe.

Aceste metode trebuie obligatoriu să aibă urmatorul format:

```
private void writeObject(ObjectOutputStream stream)  
throws IOException  
private void readObject(ObjectInputStream stream)  
throws IOException, ClassNotFoundException
```

De asemenea, uzual, primul lucru pe care trebuie să îl facă aceste metode este apelul la metodele standard de serializare a obiectelor `defaultWriteObject`, respectiv `defaultReadObject` și abia apoi să execute diverse operațiuni suplimentare. Forma lor generală este:

```
private void writeObject(ObjectOutputStream s)  
throws IOException {  
    s.defaultWriteObject();  
    // personalizarea serializării  
}  
private void readObject(ObjectInputStream s)  
throws IOException, ClassNotFoundException {  
    s.defaultReadObject();  
    // personalizarea deserializării  
    ...  
    // actualizarea stării obiectului (dacă e necesar)  
}
```

Metodele `writeObject` și `readObject` sunt responsabile cu serializarea clasei în care sunt definite, serializarea superclasei sale fiind făcută automat (și implicit). Dacă însă o clasa trebuie să-și coordoneze serializarea proprie cu serializarea superclasei sale, atunci trebuie să implementeze interfața `Externalizable`.



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Exista cazuri când dorim ca unele variabile membre sau sub-obiecte ale unui obiect sa nu fie salvate automat în procesul de serializare. Acestea sunt cazuri comune atunci când respectivele câmpuri reprezinta informatii confidentiale, cum ar fi parole, sau variabile auxiliare pe care nu are rost sa le salvam. Chiar declarate ca private în cadrul clasei aceste câmpuri participa la serializare. O modalitate de a controla serializare este implementarea interfetei Externalizable, asa cum am vazut anterior. Aceasta metoda este însa incomoda atunci când clasele sunt greu de serializat iar multimea câmpurilor care nu trebuie salvate este redusa.

Pentru ca un câmp sa nu fie salvat în procesul de serializare atunci el trebuie declarat cu modificatorul **transient** si trebuie sa fie ne-static. De exemplu, declararea unei parole ar trebui facuta astfel:

```
transient private String parola; //ignorat la serializare
```

<

Atentie

Modificatorul static anuleaza efectul modificatorului transient;

```
static transient private String parola; //participa la serializare
```

De asemenea, nu participa la serializare sub-obiectele neserializabile ale unui obiect, adica cele ale caror clase nu au fost declarate ca implementând interfata Serializable (sau Externalizable).

Exemplu: (câmpurile marcate 'DA' participa la serializare, cele marcate 'NU', nu participa)

```
class A { ... }  
class B implements Serializable { ... }  
public class Test implements Serializable {  
    private int x; // DA  
    transient public int y; // NU  
    static int var1; // DA  
    transient static var2; // DA  
    A a; // NU  
    B b1; // DA  
    transient B b2; // NU  
}
```

Atunci când o clasa serializabila deriva dintr-o alta clasa, salvarea câmpurilor clasei parinte se va face doar daca si aceasta este serializabila. In caz contrar, subclasa trebuie sa salveze explicit si câmpurile mostenite.

Șabloane de proiectare



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Șabloanele de proiectare reprezintă soluții generice la probleme des întâlnite în elaborarea aplicațiilor software. Utilizarea șabloanelor de proiectare este benefică din următoarele motive: . Soluțiile prezentate în cadrul șabloanelor de proiectare au fost analizate de întreaga comunitate a specialiștilor în programarea orientată-obiect pentru o perioadă lungă de timp. Astfel, au fost studiate toate implicațiile utilizării unui anumit șablon, iar soluțiile preconizate au un nivel ridicat de calitate. . Șabloanele de proiectare reprezintă soluții care trebuie doar adaptate la necesitățile problemei pe care dorim să o rezolvăm. Timpul de dezvoltare al aplicațiilor care folosesc șabloane de proiectare este în general mai redus față de aplicațiile care sunt dezvoltate pornind de la zero. Utilizarea șabloanelor de proiectare nu implică numai avantaje, în anumite situații pot apărea, de asemenea, și dezavantaje. De exemplu, pentru anumite aplicații, soluțiile oferite de șabloanele de proiectare pot să fie mai complexe decât unele concepute special pentru aplicația respectivă. Aceasta complexitate este datorată faptului că șabloanele de proiectare reprezintă soluții generale care trebuie să poată fi aplicate la o clasă largă de probleme. De obicei, dezavantajele sunt relativ minore față de beneficiile aduse de utilizarea șabloanelor de proiectare și pot fi minimizate prin utilizarea diferitelor specializări ale șabloanelor. Pentru descrierea șabloanelor de proiectare se folosesc diagrame UML însoțite de explicații suplimentare referitoare la aplicabilitatea șablonului, rolul componentelor șablonului, avantajele și dezavantajele utilizării acestuia etc.

Șablonul decorator

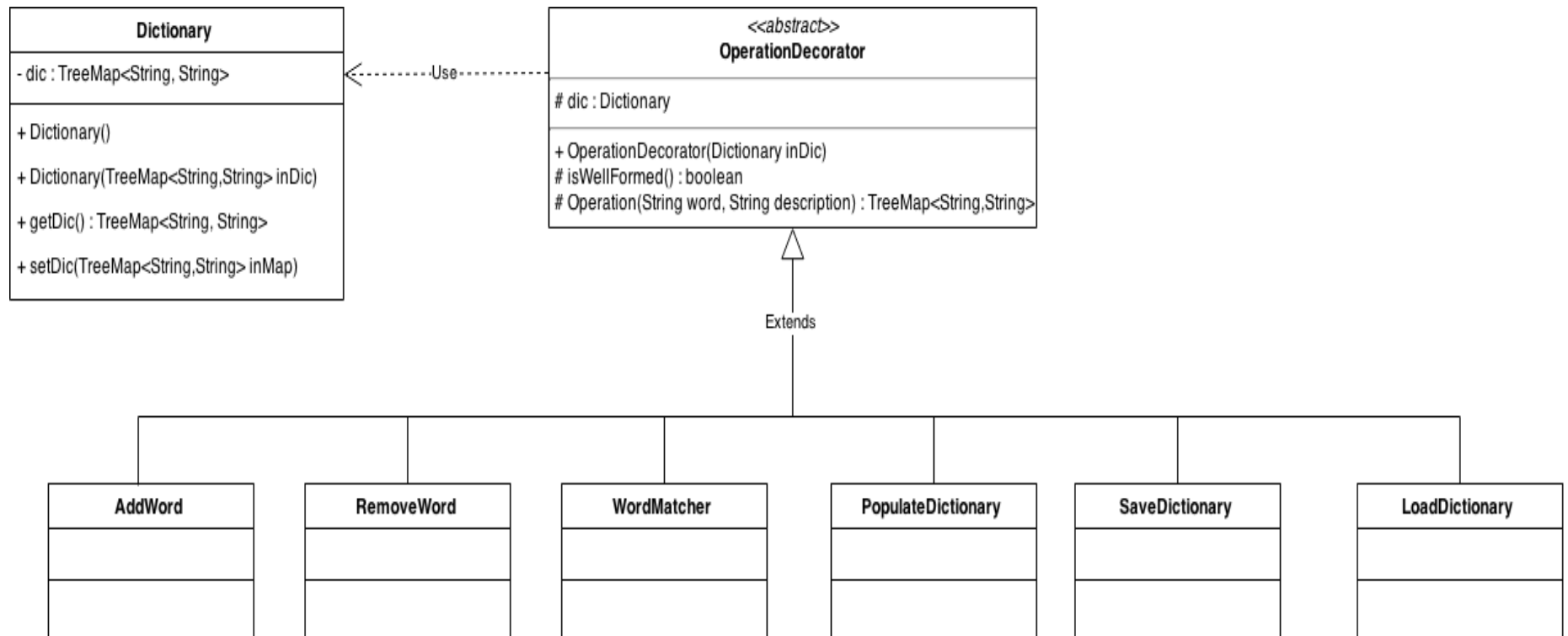
Utilizare: adaugă caracteristici și elemente de comportament noi unui obiect în mod dinamic și transparent, fără a afecta alte obiecte. Poate fi utilizat pentru situații în care anumite responsabilități ale obiectului pot fi acordate/ retrase. Reprezintă o alternativă la utilizarea subclaselor, mai ales atunci când folosirea acestora ar determina crearea unei structuri arborescente de clase și subclase foarte stufoasă.

5 În esență, șablonul Decorator reprezintă o soluție atunci când ne dorim să cream un obiect, să modificăm comportamentul metodelor obiectului după ce l-am creat, să revenim la vechiul comportament, totul fără a afecta codul client care interacționează cu obiectul.

Sablonul Decorator (152) se ocupă de relațiile între clase și obiecte, care suportă înfrumusețare prin închidere transparentă. Termenul „înfrumusețare” are de fapt un înțeles mai larg decât cel prezentat de noi aici. În șablonul Decorator, înfrumusețarea referă „orice lucru care adaugi responsabilități la un obiect. Ca exemple de înfrumusețare, ne putem gândi la un arbore abstract de sintaxă, cu acțiuni semantice, un automat cu stare finită cu tranziții noi sau o rețea de obiecte persistente cu etichete atribut. Șablonul Decorator generalizează abordarea utilizării în programul DictionaryManagement pentru a o face aplicabilă pe scări mai largi.

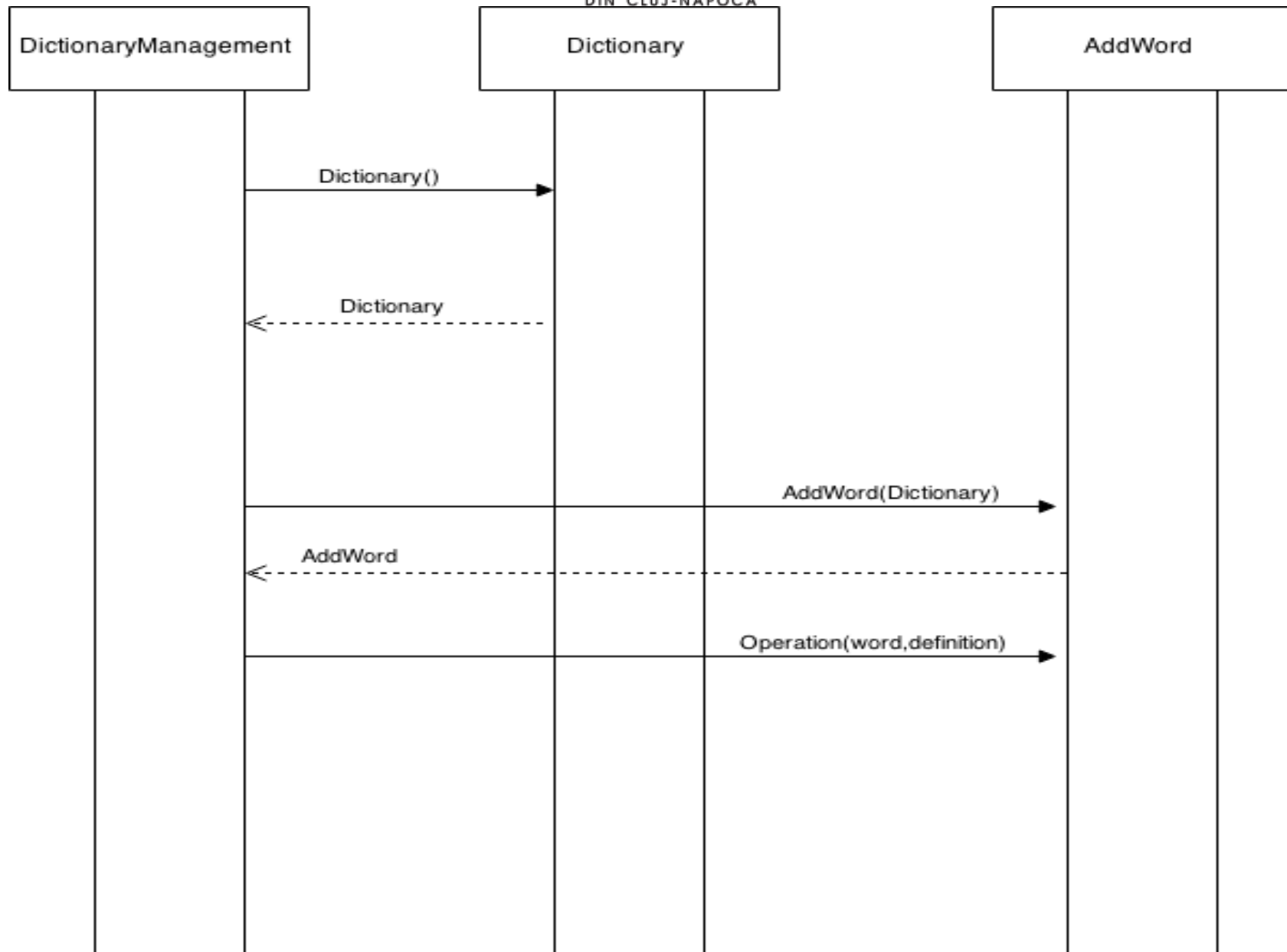


UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA



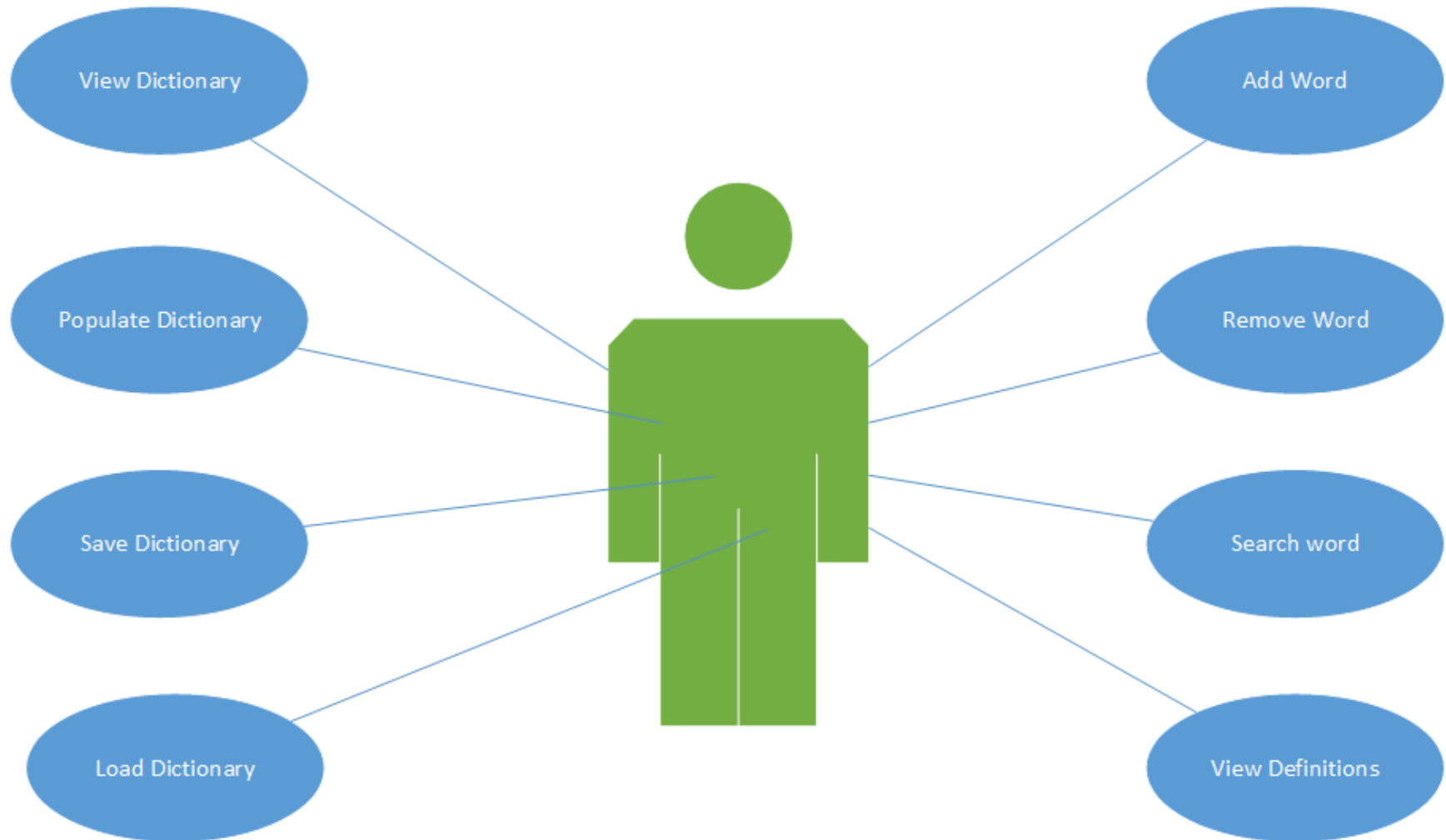


UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA



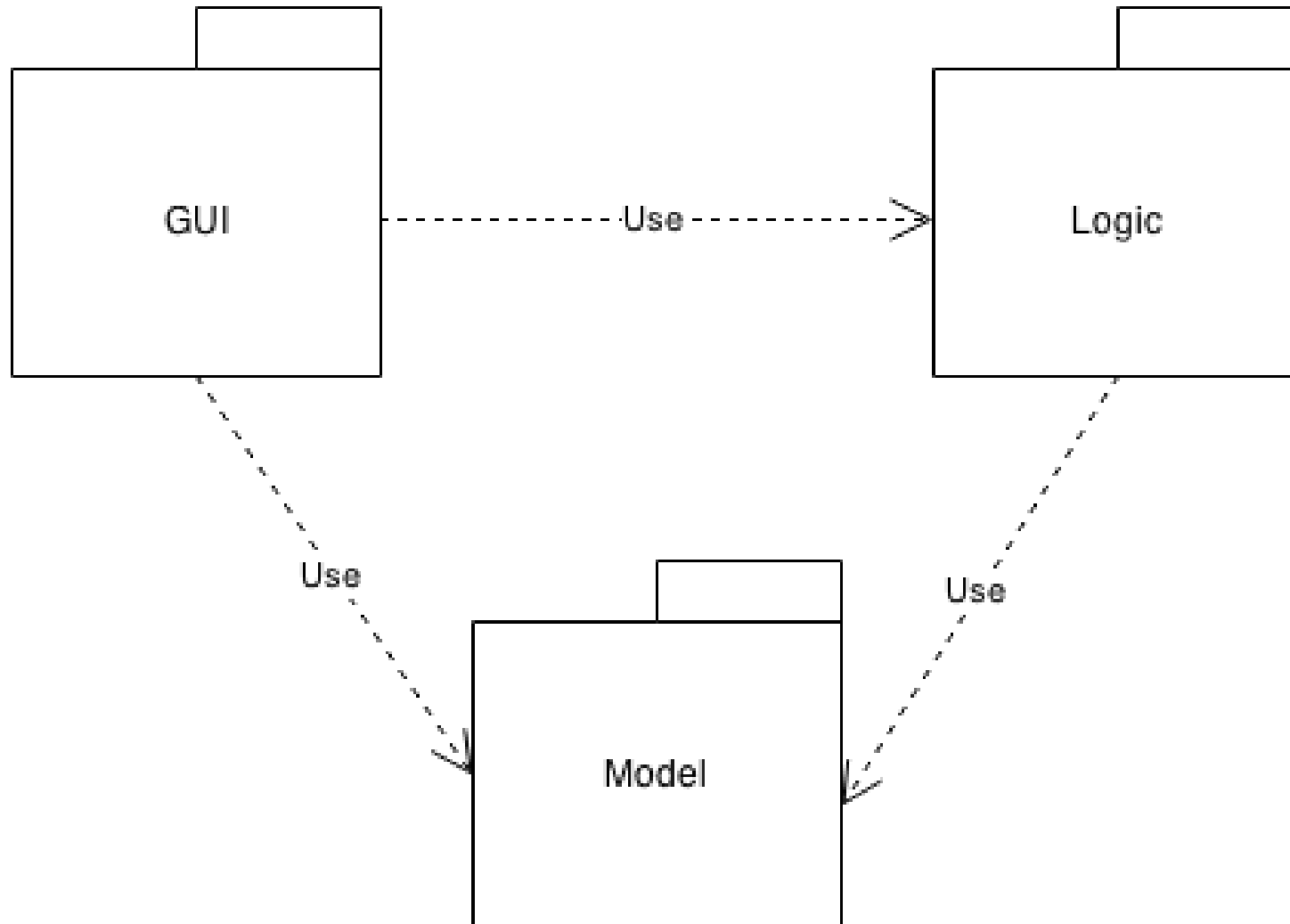


UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA





UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA



4. Implementare și testare

Clasa Dictionary conține :

```
private TreeMap<String,String> dic;
```

Acest TreeMap<String, String> conține asocierea făcută între cuvânt și definiție. Menționăm că dicționarul este într-o stare consistentă în momentul în care fiecare cuvânt are o definiție complet descrisă(nu șirul vid etc.). Acest lucru se face în metoda isWellFormed() din clasa OperationDecorator. Pe lângă această asociere, clasa Dictionary mai conține setterele și getterele necesare pentru a putea fi efectuate operații mai complexe asupra acesteia. Aceste settere și gettere urmează a fi decorate.

Clasa OperationDecorator conține :

```
protected Dictionary dic;
```

Aici se realizează legătura între dicționar și interfața care îl decorează.

Metoda isWellFormed() :

```
protected boolean isWellFormed()  
{  
    for(Entry<String,String> entry : dic.getDic().entrySet())  
    {  
        if((entry.getValue().equals(""))||(entry.getValue()==null))  
            return false;  
    }  
    return true;  
}
```

Aceasta parcurge TreeMap-ul și dacă găsește inconsistențe, returnează false.

```
public abstract TreeMap<String, String> Operation(String word, String definition);
```

Metoda abstractă Operation este metoda prin care sunt decorate operațiile. Fiecare clasă care implementează o operație atât la nivel de dicționar, cât și la nivel de cuvânt va realiza concret această metodă. Ea primește ca și parametrii String word și String definition care sunt fie ambii șiruri vide, fie ambii șiruri diferite de cel vid. În primul caz, se vor realiza operații pe dicționar, ceea ce nu au nevoie de anumite cuvinte introduse de utilizator. Al doilea caz realizează operații



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

concrete pe cuvânt(adăugare, ștergere, căutare și orice alta operație gândită de cel care dorește să extindă clasa dată) de aceea cuvântul din asociere trebuie să conțină definiția sa. Corectitudinea operațiilor sunt asigurate de metoda `isWellFormed()` care dacă aceasta returnează o valoare falsă, înseamnă ca este violată consistența dicționarului și acesta este adus în starea anterioară care era consistentă. Dacă valoarea returnată este true, atunci înseamnă că totul este în regulă și se execută operația cu succes. Pentru partea de dezvoltare efectivă, toate testele se vor executa în JVM folosind opțiunea `-ea` care activează assert-urile, pentru a asigura corectitudine în parametrii dați.

5.Rezultate

În urma implementării unei aplicații menite pentru a stoca un dicționar de cuvinte al limbii engleze, s-a obținut un soft flexibil, ușor de extins și maleabil pentru nevoi atât personale, cât și muncitorești. Astfel, softul dat poate fi folosit atât în cadrul unei persoane fără specialitate în domeniul lingvistic, cât și de un specialist(ex :translator) și chiar și de un elev/student. Posibilitatea de a modifica dicționarul în funcție de nevoie permite schimbarea mediului de utilizare în domeniul științific prin realizarea unui dicționar științific, la nevoie.

6.Concluzii și dezvoltări ulterioare

În urma realizării softului dat, s-au repetat tehnicile de :

-serializare

- design by contract

și s-a învățat lucrul cu șabloane de proiectare.

Prin design by contract se stabilește un contract cu utilizatorul software-ului, astfel încât atât cel care a propus software-ul, cât și cel care îl utilizează trebuie să îl respecte. Dacă oricare dintre cele două părți nu îl respectă, atunci contractul este încălcat și apare un bug în software.

Prin serializare se pot salva date într-un fișier sub formă binară, urmând ca mai apoi să fie deserializate pentru a fi citite(atât pe aceeași mașină, cât și pe alta)

Prin utilizarea șabloanelor de proiectare se construiește un soft flexibil, ușor de extins, performant ce îndeplinește condițiile tuturor producătorilor și ofertanților de pe piață.

7. Bibliografie

1. Cristian Frăsinaru – Curs practic de Java
2. Laborator Design după contract
3. Proiectarea orientată pe obiect din perspectiva ingineriei software - Luca Adrian
4. Curs serializare
5. Șabloane de proiectare : Elementele reutilizării software-ului orientat obiect - Banda celor 4