

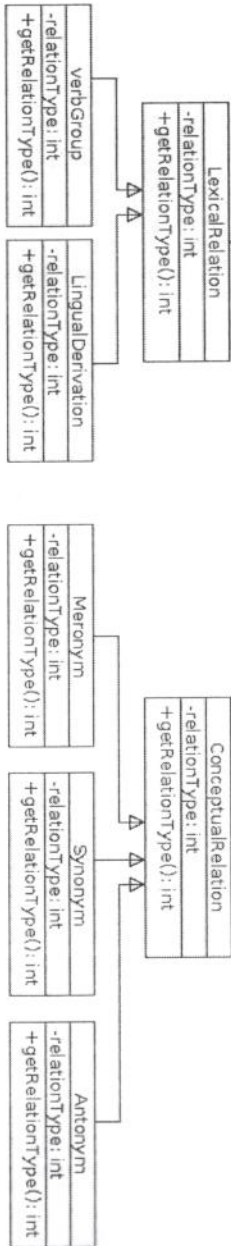
2013

Word can have more than one POS
here you consume word have one only
This is wrong and you need to fix
Mehmed Said

Word
-lemma: String
-internalIdentifier: int
-parentSynset: Synset
-posMap: Map<String, String>
+getLemma(): String
+getInternalIdentifier(): int
+getParentSynset(): Synset
+getPosOfpos: String: String

Synset
-internalIdentifier: int
-gloss: String
-words: Word[]
-relationMap: Map<Relation, Synset>
+getGloss(): String
+getInternalIdentifier(): int
+getWords(): Word[]
+containsWord: Word: Boolean
+getWordsWithRelation(word: Word, relation: Word): Word[]

«Interface» Relation
-relationType: int
+getRelationType(): int



WordNet
-instance: WordNet
-synsets: Synset[]
-settings: Settings
+getInstance(): WordNet
-WordNet(): void
+initialize(file: File): void
+getSynonyms(word: Word, partOfSpeech: String): Word[]
+getAntonyms(word: Word, partOfSpeech: String): Word[]
+getMeronyms(word: Word, partOfSpeech: String): Word[]

Settings
-settingsFile: File
-fileLocations: String[]
+read(): void
+setFile(file: File): void

Relation between Two Synsets
not one
Mehmed Said

Q1, 4

A

Q2

```

class Synset {

    private int internalIdentifier;

    private String gloss;

    private Word[] words;

    private Map<Relation, Synset> replationMap;

    public Synset(Word[] words) {

    }

    public Word[] getWords() {

    }

    public boolean contains(Word word) {

    }

    public Word[] getWordsWithRelation(Word word, Relation relation) {

    }

}

```

=====

```

class Word{

    private String lemma;

    private int inernalIdentifier;

    private Synset parentSynset;

    private Map<String, String> posMap;

    public Word (Map<String, String> posMap){

    }

    public String getLemma(){

    }

    public Synset getParentSynset(){

```

```

    }

    public String getPos(String pos){

    }

}

=====

interface Relation{

    public getrelationType();

}

=====

class ConeptualRelation implements Relation{

    private relationType;

    public getRelationType(){

    }

}

=====

class ConeptualRelation implements Relation{

    private relationType;

    public getRelationType(){

    }

}

=====

class LexicalRelation implements Relation{

    private relationType;

    public getRelationType(){

    }

}

```

```
}
```

```
=====
```

```
class Synonym extends ConceptualRelation{
```

```
    private relationType;
```

```
    public getRelationType(){
```

```
    }
```

```
}
```

```
=====
```

```
class Antonym extends ConceptualRelation{
```

```
    private relationType;
```

```
    public getRelationType(){
```

```
    }
```

```
}
```

```
=====
```

```
class verbGroup extends LexicalRelation{
```

```
    private relationType;
```

```
    public getRelationType(){
```

```
    }
```

```
}
```

```
=====
```

```
class WordNet {
```

```
    private static WordNet instance;
```

```
    private Synset[] synsets;
```

```
    private Settings settings;
```

```
    private WordNet() {
```

```

    }

    public static getInstance() {
    }

    public void initialize (File file) {
    }

    public Word[] getSynonyms ( Word word, String partOfSpeech ){
    }

    public Word[] getAntonyms(Word word, String partOfSpeech){\
    }

    public Word[] getMeronyms(Word word, String partOfSpeech){\
    }
}

```

=====

```

class Settings {

    private File settingsfile;

    private String[] filelocations;

    public read() {
    }

    public setFile ( File file ){
    }

}

```

Q3)

Delegation Pattern, to extend and reuse the functionality of WordNet.

Q4)

- 1) Facade :- The WordNet Class is a Facade for other underlying wordNet classes which hides implementation details from client
- 2) singleton :- to ensure that only one instance of WordNet exists.

Q5)

The Properties File would look like
an XML File.

< File Name >

< path > File Path < path >

</ File Name >

a- Public class DynamicLoading {

Public List<< class ? >> getClasses() {

SettingsHolder holder = new SettingsHolder();

List< class <?> > classes = new

List< String > Paths = holder.getPaths();

for (String path : Paths) {
make JarEntry entry

String className = getClass Name(entry.getJarEntryName());

Class<?> cl = Class.forName(className);

classes.add(cl);

}

}

b-

```
Public Class RightClickPopupMenu {
```

```
WordProcessor P = WordProcessor.getInstance();
```

```
Public void actionPerformed(ActionEvent e) {
```

```
String word = e.getHighlightedWord();
```

```
if (e.getSource() == "synonyms")
```

```
P.getSynonyms(word);
```

```
else if (e.getSource() == "meronyms")
```

```
P.getMeronyms(word);
```

```
else if (e.getSource() == "antonyms")
```

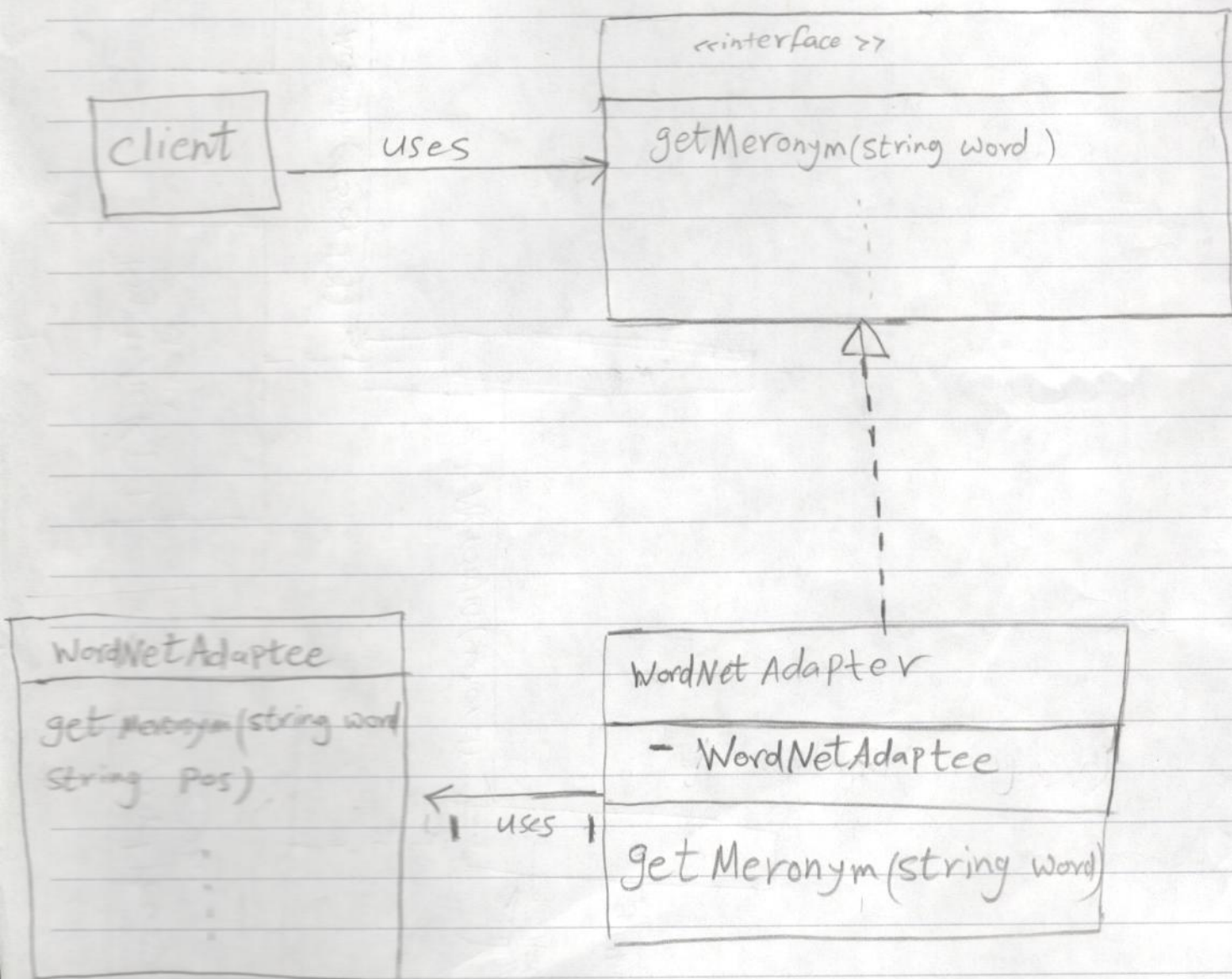
```
P.getAntonyms(word);
```

```
}
```

```
}
```


Q6) a) Adapter design Pattern

b)



(Q1)

- باہمی مدحت
- ① 37 عبد الرحمن یسری علی
- ② 39 علی محمد شاہد
- ③ 50 محمد احمد ابوالفتح

