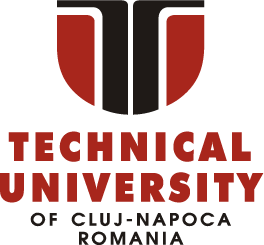
Technical University of Cluj-Napoca May, 2016

Programming Techniques

Laboratory - **HOMEWORK 5**

Romanian Synonyms Dictionary

**

*Teacher*: Ioan SALOMIE *Student*: Iulia Maria ṢOMFELEAN

*Teacher Assistant*: Delia BALAJ Group 30425

1. Objective

***TP Lab – Homework 5***

***1. Study the Java Collection Framework Map***[*https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html*](https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html)

***2. Consider the implementation of one of the following:***

*a) A dictionary of Romanian language or a dictionary of English language or*

*b) A dictionary of synonyms (thesaurus) for Romanian or English language.*

*It is required to use Java Collection Framework Map for the implementation.*

*Define and implement a domain specific interface (populate / add / remove / copy / save / search, etc.).*

*Consider the implementation of specific utility programs for dictionary processing. For example:*

*- Implement a method for checking dictionary consistency. A dictionary is consistent, if all words that are used for defining a certain word are also defined by the dictionary.*

*- Implement dictionary searching using \* (any string, including null) and ? (one character). For example, you can search for a?t\*.*

*Use the above examples to warm up your imagination.*

***Note****. The good things acquired as a result Homework 4 (i.e. contracts, invariants, assert, separating the interface from implementation, javadoc, etc.) will be also used for this homework.*

* 1. Analyzing and modelling the problem

In general usage, a **thesaurus** is a reference work that lists words grouped together according to similarity of meaning (containing synonyms and sometimes antonyms), in contrast to a dictionary, which provides definitions for words, and generally lists them in alphabetical order.

The main purpose of such reference works is to help the user "to find the word, or words, by which [an] idea may be most fitly and aptly expressed" – to quote Peter Mark Roget, architect of the best known thesaurus in the English language.

Although including synonyms, a thesaurus should not be taken as a complete list of all the synonyms for a particular word. The entries are also designed for drawing distinctions between similar words and assisting in choosing exactly the right word. Unlike a dictionary, a thesaurus entry does not give the definition of words.

* 1. Scenarios and use cases

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

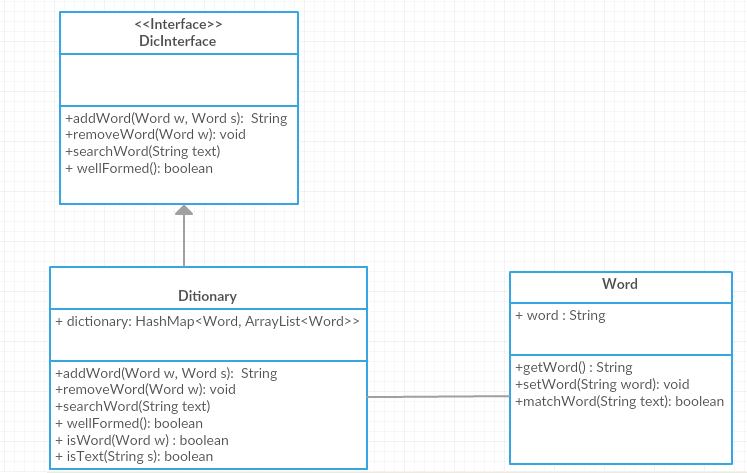
The use cases are strongly related to the user steps. I tried to design my interface in a user friendly mode, and that’s the result:

1. Implementation
   1. Diagrams
2. Use case diagrams



1. Class diagram

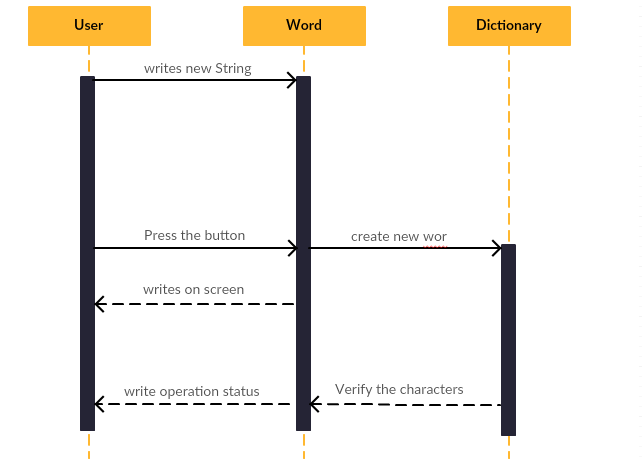
**Class diagram for the  *model* part**



1. Sequence diagram

A **Sequence diagram** is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams** or **event scenarios**.

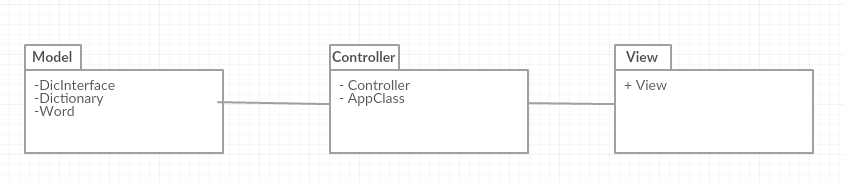
A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



1. Package diagram

Package diagram shows the arrangement and organization of model elements in middle to large scale project. Package diagram can show both structure and dependencies between sub-systems or modules.

Package diagram shows the arrangement and organization of model elements in middle to large scale project. Package diagram can show both structure and dependencies between sub-systems or modules.



* 1. Data Structures
     1. ArrayList

It’s a resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null.

In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

* + 1. HashMap

HashMap maintains key and value pairs and often denoted as HashMap<Key, Value> or HashMap<K, V>. HashMap implements Map interface. HashMap is similar to Hashtable with two exceptions – HashMap methods are unsynchornized and it allows null key and null values unlike Hashtable. It is used for maintaining key and value mapping.

It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It neither does any kind of sorting to the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

Packages

Java packages help in organizing multiple modules and group together related classes and interfaces. Packages avoid name conflicts.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

* A *Model* , which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
* A *View* , which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
* A *Controller* , which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

Given the fact that I have chosen a Model-View-Controller Pattern, I splitted my classes into the corresponding packages and, alongside them, some useful classes :

* **model**: contains the “brain” of the project, the classes that model the problem.
* **view**: represents the GUI
* **controller**: the controller part interconnects the model and the view

. . . . . . . . … . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

* 1. Class Design

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: you divide a problem into smaller problems and then you solve these little, simple and well-known problems .  
Dividing your program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.

I tried to design my project in the MVC architecture, that’s why I have 3 principal parts:

* + 1. The model – contains the logic of the application
* **Dictionary**: contains a collection with the purpose of preserving the word from our dictionary in a <key, value> manner.
* **Word:** it’s the main unit of our app, and defines an entry from a dictionary. Has various methods to work with, such as isWord(), searchWord(), etc.
* **WordContext:** it’s a class that defines a contextual synonym.
* **WordTotal:** it’s a class that defines a total synonym.
* **WordFactory:** it’s a class that belongs to the Factory DP and it is used to create different types of words.
* **IWord:** it’s an interface that is used in the Factory DP, with just one method for the moment, toString().
* **DicInterface:** it’s the interface implemented by the dictionary, contains operations such as addWord, removeWord, and it’s created from the perspective of Design By Contract.
* **WordContext:** it’s a class that defines a contextual synonym.
  + 1. The controller – contains the linking between the model and the view of the application.

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

* **Controller:** it’s the class that deals with the serialization and deserialization of the Dictionary, also contains initialization methods.
* **App:** has the task to create instances of View and Controller and to link them in order to launch the application.
  + 1. The view – View represents the visualization of the data that model contains.
* **View:** contains all graphic elements of our dictionary, a search field, adding new word field, removing words field, also a button to check consistency and to previsualize the word contained in the dictionary.

I used JTable for various operations:

The JTable is used to display and edit regular two-dimensional tables of cells.

The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.

JTables are typically placed inside of a JScrollPane. By default, a JTable will adjust its width such that a horizontal scrollbar is unnecessary. To allow for a horizontal scrollbar, invoke [setAutoResizeMode(int)](https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html" \l "setAutoResizeMode(int)) with AUTO\_RESIZE\_OFF. Note that if you wish to use a JTable in a standalone view (outside of a JScrollPane) and want the header displayed, you can get it using [getTableHeader()](https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html" \l "getTableHeader()) and display it separately.

* 1. Algorithms
     1. JSON Serialization and Deserialization

To understand object serialization in detail, let us serialize a Java object to a JSON file and then read that JSON file to get the object back.

## Object Serialization Example

In the following example, we will create a Student class. Thereafter we will create a student.json file which will have a JSON representation of Student object.

First of all, create a Java class file named JacksonTester in**C:\>Jackson\_WORKSPACE**.

### File: JacksonTester.java

import java.io.File;

import java.io.IOException;

import org.codehaus.jackson.JsonGenerationException;

import org.codehaus.jackson.JsonParseException;

import org.codehaus.jackson.map.JsonMappingException;

import org.codehaus.jackson.map.ObjectMapper;

public class JacksonTester {

public static void main(String args[]){

JacksonTester tester = new JacksonTester();

try {

Student student = new Student();

student.setAge(10);

student.setName("Mahesh");

tester.writeJSON(student);

Student student1 = tester.readJSON();

System.out.println(student1);

}

catch (JsonParseException e) { e.printStackTrace(); }

catch (JsonMappingException e) { e.printStackTrace(); }

catch (IOException e) { e.printStackTrace(); }

}

private void writeJSON(Student student) throws JsonGenerationException, JsonMappingException, IOException{

ObjectMapper mapper = new ObjectMapper();

mapper.writeValue(new File("student.json"), student);

}

private Student readJSON() throws JsonParseException, JsonMappingException, IOException{

ObjectMapper mapper = new ObjectMapper();

Student student = mapper.readValue(new File("student.json"), Student.class);

return student;

}

}

class Student {

private String name;

private int age;

public Student(){}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String toString(){

return "Student [ name: "+name+", age: "+ age+ " ]";

}

}

* + 1. Search words

The user can search after a different synonym , writing it into the search field.

As a result, the word contained in the dictionary fulfilling the search keyword are returned.

For the search I used JAVA Regex: Java provides the java.util.regex package for pattern matching with regular expressions. Java regular expressions are very similar to the Perl programming language and very easy to learn.

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. They can be used to search, edit, or manipulate text and data.

The java.util.regex package primarily consists of the following three classes:

* **Pattern Class:** A Pattern object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, you must first invoke one of its public static**compile()** methods, which will then return a Pattern object. These methods accept a regular expression as the first argument.
* **Matcher Class:** A Matcher object is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. You obtain a Matcher object by invoking the **matcher()** method on a Pattern object.
* **PatternSyntaxException:** A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.
  + **public** **boolean** matchWord(String toSearch) {
  + String input = toSearch;
  + String patternStr = input.replaceAll("\\\*", "\\.\*");
  + Pattern p = Pattern.*compile*(patternStr);
  + Matcher m = p.matcher(**this**.word);
  + **boolean** matchess = **false**;
  + **if** (m.find())
  + matchess = **true**;
  + **return** matchess;
  + }
  1. Design Patterns
     1. Singleton Design Pattern

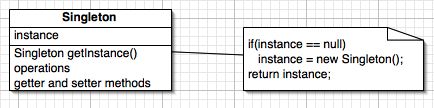
*The Singleton pattern is deceptively simple, even and especially for Java developers.*

Sometimes it's appropriate to have exactly one instance of a class: window managers, print spoolers, and filesystems are prototypical examples. Typically, those types of objects—known as singletons—are accessed by disparate objects throughout a software system, and therefore require a global point of access. Of course, just when you're certain you will never need more than one instance, it's a good bet you'll change your mind.

The Singleton design pattern addresses all of these concerns. With the Singleton design pattern you can:

* Ensure that only one instance of a class is created
* Provide a global point of access to the object
* Allow multiple instances in the future without affecting a singleton class's clients

Although the Singleton design pattern—as evidenced below by the figure below—is one of the simplest design patterns, it presents a number of pitfalls for the unwary Java developer. This article discusses the Singleton design pattern and addresses those pitfalls.



* + 1. Factory Design Pattern

### Intent

* Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
* Defining a "virtual" constructor.
* The new operator considered harmful.

### Problem

A framework needs to standardize the architectural model for a range of applications, but allow for individual applications to define their own domain objects and provide for their instantiation.

### Discussion

Factory Method is to creating objects as Template Method is to implementing an algorithm. A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps), and then delegates the creation details to subclasses that are supplied by the client.

Factory Method makes a design more customizable and only a little more complicated. Other design patterns require new classes, whereas Factory Method only requires a new operation.

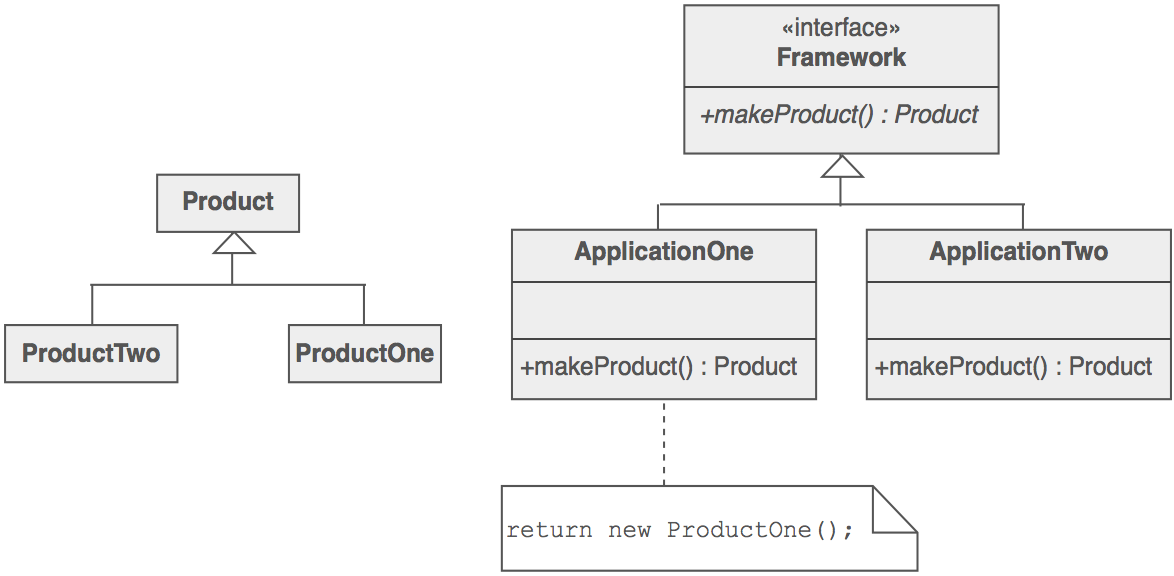
People often use Factory Method as the standard way to create objects; but it isn't necessary if: the class that's instantiated never changes, or instantiation takes place in an operation that subclasses can easily override (such as an initialization operation).

Factory Method is similar to Abstract Factory but without the emphasis on families.

Factory Methods are routinely specified by an architectural framework, and then implemented by the user of the framework.

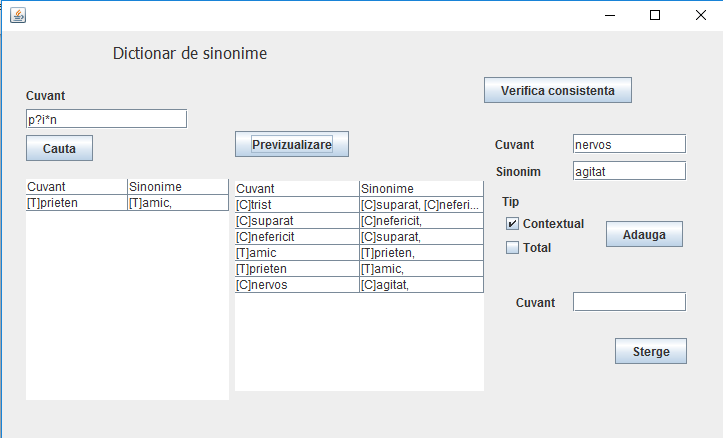
### Structure

The implementation of Factory Method discussed in the Gang of Four (below) largely overlaps with that of Abstract Factory. For that reason, the presentation in this chapter focuses on the approach that has become popular since.



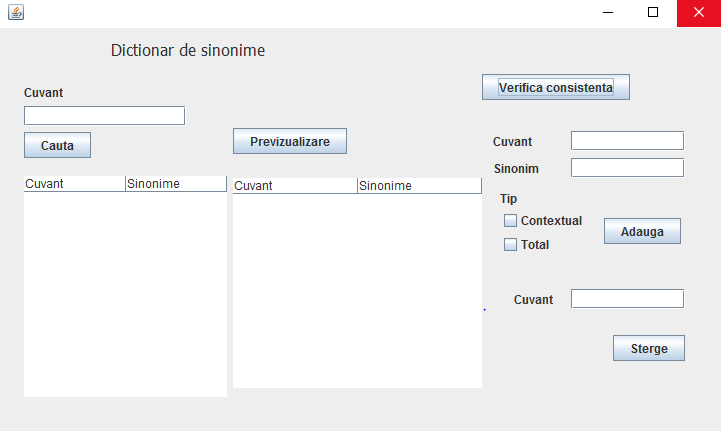
* 1. User Interface

The user interface has the role of connecting the user with our application. He can perform some dictionary based operations: view words, search for a specific one, add words, remove words, check consistency of the dictionary, etc.

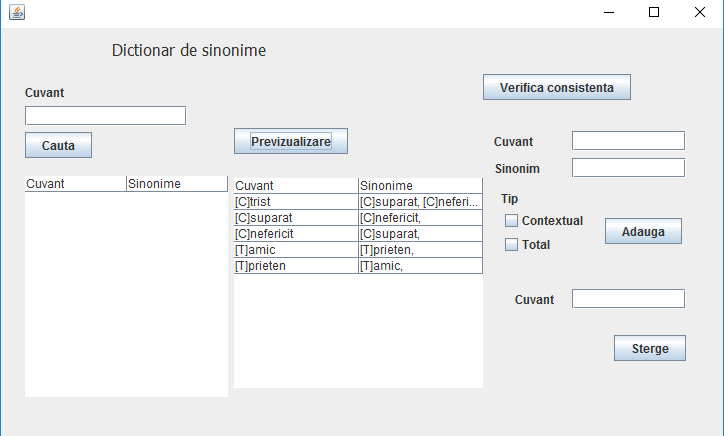


1. Implementation and testing

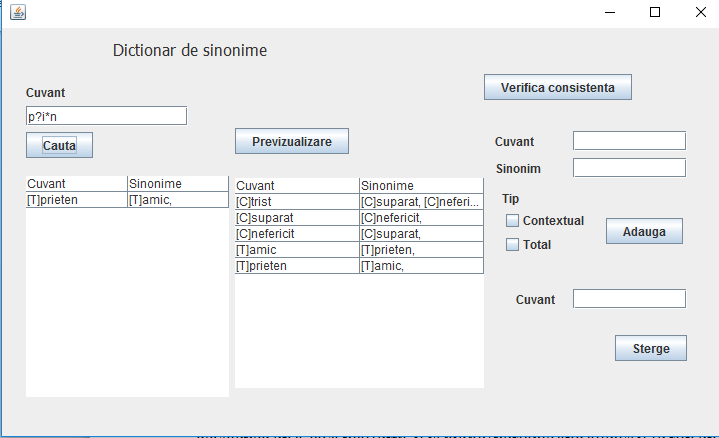
This application was developed and tested only in Eclipse, but this thing should not affect it’s portability.



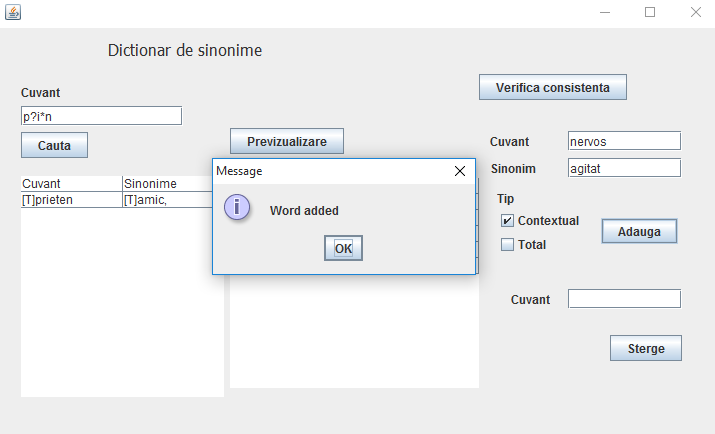
* Previsualizing the words already existing into the dictionary of synonyms



* Searching for a word, you can use \* if you want to replace more than one character, and ? to replace just one.



* Add new words, choose what type of synonym is, contextual or total



1. Results and improvements

The application is an user friendly and helpful application to perform As the application is developed on a Java platform, it is highly portable and allows it to run on several operating systems (as long as they have the Java SDK installed).

As improvements, I would enumerate:

* Adding more words
* Making some changes to the interface to improve aspect
* Copy dictionary contents externally
* Using a database to store and retrieve the information

1. Conclusions

By developing this application, I have enriched my knowledge of OOP and Java, I’ve learned more about Map Collections, Design Patters, especially Factory and Singleton. Also, I’ve worked and gained some experience with JSON files and REGEX class.

1. Bibliography
   1. Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS
   2. Programming Techniques – Lectures of prof. Ioan SALOMIE
   3. Head First Java 2nd Edition, Kathy SIERRA
   4. [www.stackoverflow.com](http://www.stackoverflow.com)
   5. <https://en.wikipedia.org/wiki/Thesaurus>
   6. <http://whatis.techtarget.com/definition/model-view-controller-MVC>
   7. wikipedia.org
   8. <https://docs.oracle.com/javase/7/docs/api/javax/swing/JTable.html>
   9. <http://www.tutorialspoint.com/java/java_regular_expressions.htm>
   10. <http://beginnersbook.com/2013/12/hashmap-in-java-with-example/>
   11. https://sourcemaking.com/design\_patterns/factory\_method