הנחיות כלליottes:

- קראו בעיון את קובצי הניהול הנגישים באתר הקורס (http://virtual2002.tau.ac.il/).
- הגישו את התוז_bcז usando מערכת ה-VirtualTAU בלבד (http://virtual2002.tau.ac.il/).
- יש להגיש קובץ zip יחודי הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש zvainer יقرأ הקובץ zvainer_hw7.zip). קובץ ה-zip יכיל את קבצי התוכננים usて致します וממשים התוז_bcז על המחשבים למשתמשים המבקרים usてպון לממשים.

:**תוז_bcז 7**

----

**תוז_bcז 7**

**הנחיות כלליות:**

- קראו בעיון את קובצי ניהול הנגישים באתר הקורס (http://virtual2002.tau.ac.il/).
- הגישו את התוז_bcז باستخدام מערכת ה-VirtualTAU.
- יש להגיש קובץ zip יחידי הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש zvainer יقرأ הקובץ zvainer_hw7.zip). קובץ ה-zip יכיל את קבצי התוכננים usてוז_bcז וממשים התוז_bcז על המחשבים למשתמשים המבקרים usてופון לממשים.

**תוז_bcז 7**

**הלק א’:** התוז_bcז של מקדונלד הזקן

Old MacDonald had a farm, E-I-E-I-O
And on his farm he had some chicks, E-I-E-I-O
With a cluck-cluck here and a cluck-cluck there
   Here a cluck there a cluck
   Everywhere a cluck-cluck
Old MacDonald had a farm, E-I-E-I-O

Old MacDonald had a farm, E-I-E-I-O
And on his farm he had some cows, E-I-E-I-O
With a moo-moo here and a moo-moo there
   Here a moo there a moo
   Everywhere a moo-moo
With a cluck-cluck here and a cluck-cluck there
   Here a cluck there a cluck
   Everywhere a cluck-cluck
Old MacDonald had a farm, E-I-E-I-O

Old MacDonald had a farm, E-I-E-I-O
And on his farm he had some dogs, E-I-E-I-O
With a woof-woof here and a woof-woof there
   Here a woof there a woof
   Everywhere a woof-woof
With a moo-moo here and a moo-moo there
   Here a moo there a moo
   Everywhere a moo-moo
With a cluck-cluck here and a cluck-cluck there
   Here a cluck there a cluck
   Everywhere a cluck-cluck
Old MacDonald had a farm, E-I-E-I-O
Requirements:

In Old MacDonald's farm you can find: dogs, cows, pigs, chicks and horses. In this exercise you will write an application that receives as input a list of animals in old MacDonald's farm (with possible repetitions). The application prints:

1. **The list of animals in old MacDonald's farm with their sounds.** The order of the animals in this list is exactly the order in the input list.
   For example: for the input "cow pig chick chick cow" the output is
   
   ```
   cow: moo
   pig: oink
   chick: cluck
   chick: cluck
   cow: moo
   ```

2. **The status of old MacDonald's farm:** a two column table where the first column contains animal names (no repetitions!) in alphabetical order and the second column contains the number of animals of this type in old MacDonald's farm.
   For example: for the input "cow pig chick chick cow" the output is:
   
   ```
   Animal  Count
   ------  ----
   chick   2
   cow     2
   pig     1
   ```

3. **The "old MacDonald's had a farm" song for the animals in the farm.** Revise the song to describe only the animals currently in the farm. For every animal type the line "And on his farm he had some ..." appears exactly once, the song then continues repeating previous types. The order of appearance of the animal types in the song is the order of appearance in the input list.
   For example: for the input "cow pig chick chick cow" the output is:
   
   ```
   Old MacDonald had a farm, E-I-E-I-O
   And on his farm he had some cows, E-I-E-I-O
   With a moo-moo here and a moo-moo there
   Here a moo there a moo
   Everywhere a moo-moo
   Old MacDonald had a farm, E-I-E-I-O
   
   Old MacDonald had a farm, E-I-E-I-O
   And on his farm he had some pigs, E-I-E-I-O
   With an oink-oink here and an oink-oink there
   Here an oink there an oink
   Everywhere an oink-oink
   With a moo-moo here and a moo-moo there
   ```
Here a moo there a moo
Everywhere a moo-moo
Old MacDonald had a farm, E-I-E-I-O

Old MacDonald had a farm, E-I-E-I-O
And on his farm he had some chicks, E-I-E-I-O
With a cluck-cluck here and a cluck-cluck there
Here a cluck there a cluck
Everywhere a cluck-cluck
With an oink-oink here and an oink-oink there
Here an oink there an oink
Everywhere an oink-oink
With a moo-moo here and a moo-moo there
Here a moo there a moo
Everywhere a moo-moo
Old MacDonald had a farm, E-I-E-I-O

Design:

A schematic description of the interfaces, classes and methods (details might differ slightly from the code).
**Resources:**

A skeleton for the application was implemented for you and you can download the files from the web site. Some of the classes have a complete implementation and should not be altered. Others are missing some implementation details and it is up to you to add those.

You should not change the signature of public methods, but you may add private methods and fields as you see fit.

Your implementation should rely on classes from the Collections framework (Set, List, Map, etc.). Read the documentation for the various classes and choose the ones you find most suitable for your need. (hint: you can choose several collections to support multiple requirements)

**Fully implemented classes:** The interface Animal and the classes implementing it (Pig, Cow, Horse, Chick and Dog) all belong to the package

```
il.ac.tau.sw1.oldmac.animals.
```

The class Main (not shown in the diagram) is the entry point to the application (i.e. its main method should be used). Main and all the classes in il.ac.tau.sw1.oldmac.animals are implemented and should not be altered.

**What you should implement:**

Complete the implementation of the classes **Farm**, **FarmBuilder** and **Song** in the package il.ac.tau.sw1.oldmac as described below.

**FarmBuilder class:**

Builds a Farm object out of a list of animals. Implements a single method:

- public static Farm buildFarm(String[] animalNames)

The method receives a list of animal types then returns a new Farm populated with those animals.

**Farm class:**

Represents a farm. implements the following methods:

- public void addAnimal(Animal animal)

  Adds a new animal to the farm.
• public Iterator<Animal> iterator()

  Returns an iterator over all the animals in the farm. The order is the same as the order in which the animals were inserted to the farm.

• public Iterator<Animal> iteratorUnique()

  Returns an iterator over all the animals in the farm without repetitions. The iterator iterates over the animals in the farm by the order of their addition to the farm. For example, if the animals added to the farm were: cow, pig, chick, chick, cow (in this order), then the order of iteration is cow, pig, chick.

• public void printStatus()

  Prints the status of the farm as described in the second requirement.

**Song class:**

• public static void printSong(Farm farm)

  Prints the "Old MacDonald had a farm" song as described in the third requirement.

You may add any methods and fields as you deem necessary to those three classes. In your implementation you should use classes (and interfaces) from the Java Collection Framework.

You may assume that:

• The list of arguments to the application is not empty and that every argument is one of the following: "cow", "chick", "horse", "dog" or "pig".
• The method Iterator.remove() is never called for the two iterators of class Farm.

**Singletone animals**

In order to avoid holding unnecessary instances of animals, animal implementations follow the singleton pattern (http://en.wikipedia.org/wiki/Singleton_pattern). We will briefly go over it in tigrul.
In this section, you are required to implement a simple search engine.

Some of the code has already been written for you and is available for download on the course website. Our search engine will handle a small number of HTML pages that we will scan from the web.

These pages are fixed in advance.

You can find the list in the class `SearchEngine`. If you want, you can add or modify the pages you are using.

After downloading an HTML page from the web, we only look at the text part of the page and break that part into individual words.

This code is already implemented for you in the class `HTMLTokenizer`. Moreover, the class `SearchEngine` in which the code that powers the system interacts with the user already exists.

What to do:

We want to create an index of all the words that appeared in all the pages we downloaded from the web.

This index will allow us to perform searches for a specific word at a later time.

You need to implement the class `MyWordIndex` that implements the `WordIndex` interface.

The class will keep the index, enable adding words to the index and searching for words.

```java
package il.ac.tau.cs.sw1.simplesearch;

import java.util.Collection;
import java.util.List;

public interface WordIndex {

    /**
     * Add the words originating in the specified URL.
     * @param words - collection of words to add to the index
     * @param strURL - the location of the page containing the words
     */
    void index(Collection<String> words, String strURL);

    /**
     * Search for a given word in the index
     * @param word - the word to search
     * @return A list of pages containing the word. The pages are ordered according to the relative importance of the word within them.
     */
    List<String> search(String word);
}
```
The methods:

- **index** method

  The method takes in a collection of words and the URL from which they came. It is responsible for filling your database structure with data. It takes a set of words (possibly repeated) and the URLs from which they came.

  You need to choose the structure you want to use and ensure the following:
  - how many times a word appears in each page
  - how many words are in each page
  - how many words a word is used in each page (with repetitions).

  Your database query should be a list of words as they appear in the original page, but you should keep the lowercase version of the words.

- **search** method

  This method takes a word to search for and returns a sorted list of URLs where the word appears.

  We will sort the list such that the higher the relative weight of a word in a page, the higher it will be in the list. The relative weight of a word in a page will be the number of times it appears divided by the total number of words on the same page.

  Hint:
  - Use the Java Collections framework, especially the Map and HashMap classes, which may be helpful.
  - Use the `Collections.sort(...)` method to sort the list.
  - Note that the "natural sorting" of the URLs (String) is alphabetical and not as required.
  - To change the sorting method, you need to implement a Comparator.

  Note:
  - The words received from the HTML are "unclean" (containing punctuation, etc.). There is no need to clean them every time, just use them as they are.

```java
public class Main {
    public static void main(String[] args) {
        SimpleSearchEngine searchEngine =
            new SimpleSearchEngine(new MyWordIndex());
        searchEngine.run();
    }
}
```

Conclusions:

Using the `Main` class, you can use the `MyWordIndex` implementation to search for words in HTML documents. For example:

```
> java
```
הערות כלליות:
בשני חלקי התרגיל חלק períוכד פה גנובב עיגולום עיגולום טספ מיםוח לחלקים החסרים. חוכל
ליגוב את הקוד הקים עיון שחרירימ את كبצי הים המופיעים בחתר אליכם להחש. חוכל חוכל
ליגוב את הפרוביקט בעיז הפורית הייבואון (File-import) של אקליפט.
ביחרו בייבוא של פרויקט קים (Existing Projects into Workspace) אחר מכ FRAME ואת קובי
הויפ בוגר ישתלט בחירה הקובמ המתחים.