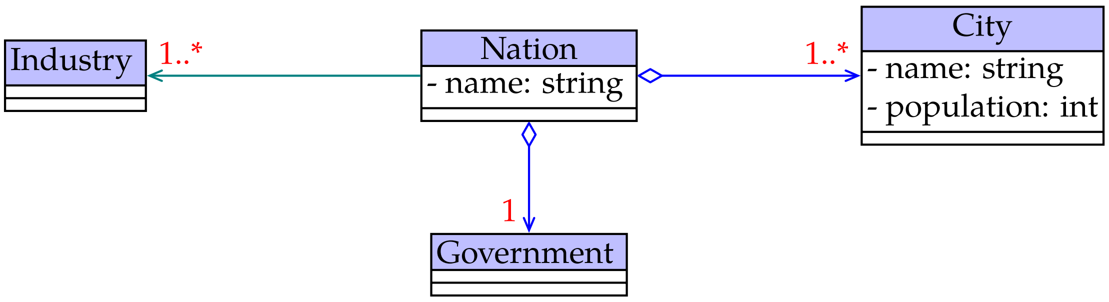
Worksheet 1: Containers

Updated: 5th March, 2018

1. Discussion: Multiplicity

Consider this UML classs diagram:



Focusing on the `Nation` class specifically:

(a) Would it make sense to have `getPopulation()` and `setPopulation()` methods?

(b) What other accessors and mutators could you usefully have? Assume that multiplicity is implemented using a List.

(c) How would this differ if you used Sets instead of Lists?

2. Discussion: Maps and Multiplicity

Consider this class:

```
public class RealEstate
{
    private Map<Address,Property> properties;
    ...
}
```

Here, `properties` is a map that stores pairs of `Address` objects (keys) and `Property` objects (values).

Note: Java maps contain these methods (among others):

```
public void put(K key, V value); // Adds or overwrites a value.
public V get(K key);             // Retrieves a value.
```

`K` and `V` are the datatypes of the keys and values.

(a) What accessors and mutators could you usefully have in the `RealEstate` class?

(b) How would you update the address of a property?

3. Working with Containers #1: AddressBook

Obtain a copy of AddressBook.zip from Blackboard.

Note: Notice the file "build.xml"? See section on Ant and Other Things below.

This contains incomplete Java code for a very simplistic address book application. Conceptually, an address book contains a collection of entries, where each entry can be retrieved by name. Each entry contains a name and a collection of unique email addresses.

The application reads names and email addresses from a text file. Each line of the file contains one address book entry, containing strings separated by ":". The first string is the person's name. Each subsequent string is an email address for that person. Once read, the user is then allowed to search for entries, either by name or by email.

You must complete the code. You will need to write the `AddressBook` and `Entry` classes, using standard Java containers, and fill in the gaps in the `AddressBookApp` class. Use the *most appropriate* containers.

Note: *Don't use arrays!* The point of the exercise is to apply concepts from the lecture. Arrays have their place, but here you need to practice using Java's lists, sets and maps. (These are some of the best things about modern programming languages!)

4. Working with Containers #2: ImageViewer

Obtain a copy of one of the ImageViewer-xyz.zip files Blackboard. There are several, for different languages/GUIs, but they all share the same (incomplete) design.

Note: It's your choice which version you select. Once you finish with one version, you can also (if you like) try out your design in a different language.

The image viewer reads an album file containing image filenames, one per line, each followed by a colon and then some caption text. Once read, a simple GUI is displayed, allowing the user to view one image at a time (and its caption), and select the next or previous one as desired.

You must complete the code. You will need to write the `Album` class, using a standard container, and fill in the gaps in:

- One place in the `readAlbumFile()` method/function (in `MainApplication.java`, `main.cpp`, or `main.py`); and
- Several places in the `MainWindow` class.

Ant and Other Things

AddressBook and ImageViewer (the Java and C++ versions anyway) are set up to use a tool called Ant, which works a bit like Make. (The build.xml file is quite like a makefile.)

To build the project, simply type the following in the main directory:

```
[user@pc]$ ant
```

This will generate a compiled program in the `dist` directory. In the case of Java projects, it generates a .jar file, which you can run like this:

```
[user@pc]$ java -jar AddressBook.jar
```

(A .jar file is just a .zip file containing Java .class files.)

End of Worksheet