Sit-in Lab #2

CS1020 AY2010/11 Semester 2
Date: 10 March 2011, Thursday
School of Computing, National University of Singapore

Instruction

There are **two exercises** in this sit-in lab. The second exercise is related to the first exercise. You can more or less reuse the coding to reduce the coding effort. You are restricted to use topics covered in the take home lab two, namely: Generic classes, Vector class, String class and simple class relationship. Please note that proper use of object oriented design (e.g. visibility of object attributes, methods etc) are part of the programming style criteria.

Time Limit: **1 hour 40 minutes**

1. Exercise One (Dictionary)

The idea of **lookup table** is very useful in computing. A lookup table basically keep tracks of tuples (pairs) of 2 components **{ key, value }**. The main operation of a lookup table is to retrieve the value associated with a given key.

For example, the **English dictionary** (or any language dictionary) is an example of a lookup table, where both the key and value are strings. The **key** in this case is a single English word, and the **value** is the associated explanation (meaning) of the word. So, given a single English word (the key), we will look up for the associated meaning (the value) in the dictionary.

Your task is to provide an implementation of a **Dictionary class** that provides the following operations:

1. **`void update( String key, String value )`**

   Behavior: If the pair **{ key, value }** does not exist in the dictionary, add a new pair. If there is already a pair of **{ key, oldValue }, replace the old pair with the new information given as {key, value}.** The key is **case sensitive**.

2. **`String getValue( String key ) throws NoSuchElementException`**

   Behavior: Return the value associated with the specified key. If the key cannot be found, an exception **NoSuchElementException** is thrown.

3. **`int size()`**

   Behavior: Return the number of pairs stored.

To test the dictionary class, write **a separate driver main method to** processing the user input as follows:

User Input	Meaning	Sample Output
1 word : explanation	Add **{ word, explanation}** to the Dictionary object. Note that the word is a single word, where explanation is a sentence of multiple words. There is a " **:** " (space colon space) separating the word from the explanation. Print the size of the Dictionary as result. Example: **1 apple : A juicy red fruit**	**`size: 1`**
2 word	Retrieve the meaning of the "word". If the word is not in the dictionary, print error message. Example: **2 apple**	If word can be found: **`apple = A juicy red fruit`** Otherwise: **`apple not found!`**
3	Stop the program	Program ended

All input and output should be handled in the driver main method only. You only need to provide implementations in the following two files:

`Dictionary.java` : The implementation of the **`Dictionary`** class
`TestDictionary.java`: The driver class

Other files **should not be modified in any way**. The files are:

`Pair.java` : Implemented Generic Pair class. You are free to use it if needed.

1.1 Sample Run

Please note that user input is in **bold font**.

```
$ java TestDictionary

1 Apple : A juicy red fruit
size: 1
1 Coding : A very fun activity to turn brain cells into code
size: 2
2 Coding
Coding = A very fun activity to turn brain cells into code
1 Coding : Not a fun activity
size: 2          //Notice that the size is not changed. The meaning of "Coding" is updated.
2 Coding
Coding = Not a fun activity
2 Happiness
Happiness not found!
3
Program ended
```

1.2 Hints and Suggestions:

- You are given the Pair< S, T > generic class to help with the implementation. You are free **to choose whether to utilize it**.
- You are allowed to use Java Vector to store the information in the Dictionary class.
- When comparing keys, you should use the ***equals*()** method instead of the ***compareTo***() method. Using ***equals*()** will make the code more portable to second exercise later.

2. Exercise Two (Generic Dictionary)

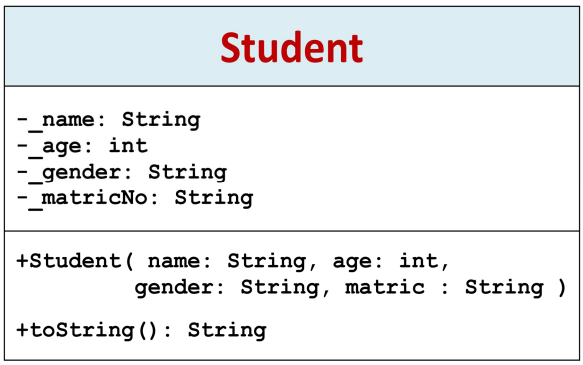
Since the idea of lookup table is applicable for **any data type**, we can generalize **Dictionary** as a generic class. Write a **`GenDictionary`** class along the line of the following declaration:

```
class GenDictionary< KeyType, ValueType >
//KeyType and ValueType are generic type variable
{
    //support the same 3 methods:
     // update( key, value), getValue( key )  and size()
}
```

This class should support the exact same 3 methods given in the 1st exercise: **update(key, value), getValue(key)** and **size()**. You need to figure out what are the corresponding changes in the parameter and return type for these methods. The coding should remain largely unchanged.

Let us use this generic dictionary as a ***student lookup table***. Each student contains the following information: **name (a string with multiple words), age (an integer)** and **gender (a single word)**. In addition, each student is uniquely identified by **matriculation number (a single word)**.

The student class is already implemented for you. The class diagram is as follows:



Write a driver main method to test the generic dictionary class by processing the user input as follows:

User Input	Meaning	Sample Output
1 **Student Name** **Age** **Gender** **Matric Number**	Add a student to the student lookup table. The student's information is given in the next 4 lines of input. Print the size of the lookup table as result. Example: **1** **Alan Turing** **18** **Male** **G3N1U5**	**`size: 1`**
2 MatricNumber	Retrieve the student with the matric number given. If the student is not in the look up table, print error message. Example: **2 G3N1U5**	If student can be found: **`Student Information:`** **`Name: Alan Turing`** **`Age: 18`** **`Gender: Male`** **`Matric: G3N1U5`** Otherwise: **`G3N1U5 not found!`**
3	Stop the program	Program ended

All input and output should be handled in the driver main method only. You only need to provide implementations in the following two files:

`GenDictionary.java` : The generic implementation of the **`GenDictionary`** class
`TestGenDictionary.java`: The driver class

Other files **should not be modified in any way**. The files are:

`Student.java` : Implemented Student class
`Pair.java` : Implemented Generic Pair class

2.1 Sample Run

Please note that user input is in **bold font**.

$ **java TestGenDictionary**

```
1
Alan Turing
18
Male
G3N1U5
size: 1
1
John Von Nuemann
19
Female
C001010
size: 2
2 G3N1U5
Student Information:
Name: Alan Turing
Age: 18
Gender: Male
Matric: G3N1U5
1
John Von Nuemann
20
Male
C001010
size: 2                    //Updated existing student, not adding new one
2 C001010
Student Information:
Name: John Von Nuemann
Age: 20
Gender: Male
Matric: C001010
2 U0102345
U0102345 not found!
3
Program ended
```

2.2 Hints and Suggestions:

- The code is very similar to exercise 1. Try to reuse the code whenever possible.

- If you need to read an integer (nextInt()), followed by reading a sentence (nextLine()). Please remember that nextLine() will read any left over characters from previous input (including new line characters!). Use an additional nextLine() to get rid of left over before you read the next sentence.
- When comparing keys, you should use the **equals()** method instead of the **compareTo()** method.

~~~~~ END OF SIT-IN LAB 2 ~~~~~
~~~~~