OBJECT-ORIENTED PROGRAMMING
LAB 8: COLLECTIONS OF DATA

I. Objective

After completing this lab tutorial, you can:

- Understand *Generics, ArrayList, Vector, and HashMap* in OOP.

II. Generics

There are programming solutions applicable to a wide range of data types. The code is the same other than the data type declarations.

In Java, you can make use of generic programming: A mechanism to specify a solution without tying it down to a specific data type.

Generic types naming conventions

In this lab tutorial, we have a list of naming conventions:

E – Element

K – Key

V – Value

T – Type

Generic Methods

You can write a single generic method declaration that can be called with arguments of different types. Based on the types of arguments passed to the generic method, the compiler handles each method call appropriately. Following are the rules to define Generic Methods.

- All generic method declarations have a type parameter section delimited by angle brackets (`<` and `>`) that precede the method's return type (`<E>` in the next example).
- Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
- The type parameters can be used to declare the return type and act as placeholders for the types of arguments passed to the generic method, which are known as actual type arguments.

- A generic method's body is declared like any other method's. Note that type parameters can represent only *reference types*, not primitive types (like int, double, and char…).

Example

```
// GenericMethod.java
public class GenericMethod {
    public static <E> void printArray( E[] inputArray ) {
        for(E element : inputArray) {
            System.out.printf("%s ", element);
        }
        System.out.println();
    }
}
```

```
// Test.java
public class Test {
    public static void main(String args[]) {
        Integer[] intArray = {1, 2, 3, 4, 5};
        Double[] doubleArray = {1.1, 2.2, 3.3, 4.4};
        Character[] charArray = { 'T', 'D', 'T', 'U' };

        System.out.println("Array integerArray contains:");
        printArray(intArray);

        System.out.println("\nArray doubleArray contains:");
        printArray(doubleArray);

        System.out.println("\nArray characterArray contains:");
        printArray(charArray);
    }
}
```

Generic Classes

A generic class declaration looks like a *non-generic class* declaration, except that the class name is followed by a type parameter section.

As with generic methods, the type parameter section of a generic class can have *one or more type* parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Example

```
// Box.java
public class Box<T> {
    private T t;

    public void set(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
}
```

```java
// Test.java
public class Test {
    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        Box<String> stringBox = new Box<String>();

        integerBox.set(new Integer(10));
        stringBox.set(new String("Hello World"));

        System.out.printf("Integer Value :%d\n\n", integerBox.get());
        System.out.printf("String Value :%s\n", stringBox.get());
    }
}
```

III. ArrayList

The *ArrayList* class is a resizable array, which can be found in `java.util` package.

The difference between a built-in array and an *ArrayList* in Java is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an *ArrayList* whenever you want. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

- *ArrayList* inherits *AbstractList* class and implements *List* interface.
- *ArrayList* is initialized by size, however, the size can increase if the collection grows or shrinks if objects are removed from the collection.
- Java *ArrayList* allows us to randomly access the list.
- *ArrayList* **can not be used for primitive types**, like int, char, etc. We need a wrapper class for such cases.

Constructors

- `ArrayList()`: This constructor is used to build an empty array list
- `ArrayList(Collection coll)`: This constructor is used to build an array list initialized with the elements from collection c
- `ArrayList(int capacity)`: This constructor is used to build an array list with the initial capacity being specified

Methods

- `add(Object obj)`: This method is used to append a specific element to the end of a list.
- `add(int index, Object obj)`: This method is used to insert a specific element at a specific position index in a list.
- `get(int index)`: Returns the element at the specified position in this list.

- `set(int index, E element)`: Replaces the element with the specified element at the specified position in this list.
- `remove(int index)`: Removes the element at the specified position in this list.
- `remove(Object obj)`: Removes the first occurrence of the specified element from this list, if it is present.
- `clear()`: This method is used to remove all the elements from any list.
- `size()`: Returns the number of elements in this list.

Example

```
import java.util.ArrayList;

public class Test2 {
    public static void main(String[] args) {

        ArrayList mylist = new ArrayList();

        mylist.add(10);
        mylist.add("Hello");
        mylist.add(true);
        mylist.add(15.75);

        int i = (Integer)mylist.get(0);
        String s = (String)mylist.get(1);
        boolean b = (boolean)mylist.get(2);
        double d = (double)mylist.get(3);

        System.out.println("1st element: " + i);
        System.out.println("2nd element: " + s);
        System.out.println("3rd element: " + b);
        System.out.println("4th element: " + d);
    }
}
```

```
import java.util.*;

class Test {
    public static void main(String[] args) {
        int n = 5;
        ArrayList<Integer> arrlist = new ArrayList<Integer>(n);
        for (int i=1; i<=n; i++)
            arrlist.add(i);
        System.out.println(arrlist);
        arrlist.remove(3);
        System.out.println(arrlist);
        for (int i=0; i<arrlist.size(); i++)
            System.out.print(arrlist.get(i)+" ");
    }
}
```

IV. Vector

The *Vector* class implements a growable array of objects. Vectors fall in legacy classes but now it is fully compatible with collections.

- *Vector* implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.
- They are very similar to *ArrayList* but *Vector* is synchronized and has some legacy methods that the collection framework does not contain.
- It extends *AbstractList* and implements *List* interfaces.

Constructors

- `Vector()`: Creates a default vector of an initial capacity is 10.
- `Vector(int size)`: Creates a vector whose initial capacity is specified by size.
- `Vector(int size, int incr)`: Creates a vector whose initial capacity is specified by size and increment is specified by *incr*. It specifies the number of elements to allocate each time that a vector is resized upward.
- `Vector(Collection coll)`: Creates a vector that contains the elements of collection *coll*.

Methods

- `isEmpty()`: Tests if this vector has no components.
- `size()`: Returns the number of components in this vector.
- `add(E el)`: Appends the specified element to the end of this Vector.
- `add(int index, E el)`: Inserts the specified element at the specified position in this Vector.
- `remove(int index)`: Removes the element at the specified position in this Vector.
- `remove(Object obj)`: Removes the first occurrence of the specified element in this Vector if the Vector does not contain the element, it is unchanged.
- `get(int index)`: Returns the element at the specified position in this Vector
- `indexOf(Object obj)`: Searches for the first occurrence of the given argument, testing for equality using the equals method.
- `contains(Object obj)`: Tests if the specified object is a component in this vector.

Example

```
import java.util.Vector;
public class TestVector {
    public static void main(String[] args) {
        Vector<String> courses = new Vector<String>();
        courses.add("501043");
        courses.add(0, "501042");
        courses.add("502043");

        System.out.println(courses); // [501042, 501043, 502043]
        System.out.println("At index 0: " + courses.get(0)); // At index 0: 501042

        if (courses.contains("501043"))
            System.out.println("501043 is in courses");
            courses.remove("501043");
        for (String c : courses)
            System.out.println(c); // 501042 // 502043

    }
}
```

V. HashMap

You learned that *Arrays* store items as an ordered collection, and you have to access them with an index number (int type). A *HashMap*, however, stores items in "key: value" pairs, and you can access them by an index of another type (e.g. a *String*).

One object is used as a key (index) to another object (value). It can store different types: *String* keys and *Integer* values, or the same type, like *String* keys and *String* values.

You can learn more about *LinkedHashMap*, and *TreeMap*. In this section, we just focus on *HashMap*.

Constructors

```
HashMap<K,V> map = new HashMap<K,V>();
```

Methods

- `put(Object key, Object value)`: It is used to insert a particular key-value pair mapping into a map.
- `get(Object key)`: It is used to retrieve or fetch the value mapped by a particular key.
- `replace(K key, V value)`: This method replaces the entry for the specified key only if it is currently mapped to some value.
- `remove(Object key)`: It is used to remove the values for any particular key in the Map.
- `clear()`: Used to remove all mappings from a map.
- `size()`: It is used to return the size of a map.

- `keySet()`: It is used to return a set view of the keys.

Example

```
import java.util.HashMap;

public class MyClass {
    public static void main(String[] args) {
        HashMap<String,Integer> people = new HashMap<String,Integer>();

        people.put("John", 32);
        people.put("Steve", 30);
        people.put("Angie", 33);

        for (String i : people.keySet()) {
            System.out.println("key: " + i + " value: " + people.get(i));
        }
        // key: John value: 32
        // key: Steve value: 30
        // key: Angie value: 33
    }
}
```

VI. Exercises

1. Write a Java program:
   - Create a class Person with attributes such as *name* and *birth year*. Write constructor, setter, getter, and toString methods.
   - Create a class Student that inherits from class Person with attributes *id*, *score*. Write constructor, setter, getter, and toString methods.
   - Create a class Employee that inherits from class Person with attributes *id*, *salary*. Write constructor, setter, getter, and toString methods.
   - Create class PersonModel and perform the following tasks:

```
import java.util.ArrayList;

public class PersonModel <T> {
    private ArrayList<T> al = new ArrayList<T>();
    public void add(T obj) {
        al.add(obj);
    }
    public void display() {
        for (T obj : al) {
            System.out.println(obj);
        }
    }
    public static void main(String[] args) {
        // code here
    }
}
```

- Create a ***PersonModel*** object.
- Use the add method to enter 2 students.
- Use the display method to display information of 2 students who have entered.
- Create a ***PersonModel*** object.
- Use the add method to enter 2 employees.
- Use the display method to display information of 2 employees who just entered.
- Create a ***PersonModel*** object.
- Use the add method to enter the names of 2 people.
- Use the display method to display information of 2 people who just entered.

2. Write a Point class with 2 attributes **x** and **y** and some methods like **getter, setter, and toString**. Write a program to create an *ArrayList* with Point data types. Then find all points inside the circle that have center O (0, 0) with a radius of 1. Hint: Write a method for calculating the distance between a point and center O (0, 0).

$$distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

3. Given an abstract class **Student** with 02 properties: **sName**, **gpa** (grade point average), this class has an abstract method **public abstract String getRank()**. Class **ITStudent** and **MathStudent** are two subclasses of **Student**. **ITStudent** and **MathStudent** both have properties **sID**, but the **sID** of **ITStudent** is **int** type, **sID** of **MathStudent** is **String** type. **ITStudent** and **MathStudent** have different criteria to rank.
   - **ITStudent**: gpa <= 5 is rank "C", 8 >= gpa > 5 is rank "B", 10 >= gpa > 8 is rank "A".
   - **MathStudent**: gpa >= 5 is "Passed", gpa < 5 is "Failed".

   Write a method **public ArrayList<Student> findStudent(ArrayList<Student> lstStu)** in the processing class to find the **Student** objects that have rank **"A"** or **"Pass"** in the Student ArrayList. Write a main method in the processing class to create an **ArrayList** of **Student** objects and print the result of the above method.
4. Write a program using *Vector* class with *Integer* data type. Create an **X** vector of length *n*, then enter the elements into the vector. Calculate the vector Y as the value of **f (X)** with **f (X)** as $2x^2 + 1$ and then output the value to the screen.
5. Use *HashMap* to write an English-Vietnamese dictionary program with about 10 words. Each word has 2 attributes: "word" and "meaning".
   - Write a method to check if a word exists in the dictionary or not.
   - Write a method to find the meaning of a word.
   - Write the main method that requires the user to enter words to look up and then search, and print on the screen the meaning of the words you have requested.

-- THE END --