Tutorial – Week 07 5COSC019W – Object Oriented Programming – Java

Arrays and Collections

07 November - 08 November

Array and ArrayList

Array :

- Arrays are simple data structure that store a set of values of the same type.
- Arrays are part of the Java language.
- Each value is accessed through an index
- Array index always starts at 0, not 1
- Array lengths cannot be changed once they are declared
- Arrays can be initialized when declared

ArrayList :

- It s a more flexible way to store data
- ArrayList can grow automatically as needed
- Has capacity that is increased when needed
- Has size() method that returns actual number of elements in the ArrayList
- ArrayList class is not in the core Java language – It is in package java.util, which you must import:
  `import java.util.*;  // At top of program`

Exercises

1) Consider the following method implementation:

```
public static void loadArray(int[] list) {
   for (int i = 1; i < list.length; i++)
   {
         list[i] = list[i] + list[i - 1];
   }
}
```

Indicate which values will be store in the array after the method `loadArray` is executed if the following array of int is passed as a parameter to `loadArray`.

a) {7}
b) {3, 6}
c) {4, 6, 8}
d) {1, 2, 3, 4}
e) {8, 4, 2, 0, 4}

2) Consider the following method implementation:

```
public static void listDemo(ArrayList<Integer> list) {
   for (int i = 0; i < list.size(); i++)
   {
         int element = list.get(i);
         list.remove(i);
         list.add(0, element + 1);
   }
System.out.println(list);
}
```

Indicate which will be the output produced after the method `listDemo` is executed if the following ArrayLis of int is passed as a parameter to `loadArray`.

f) [5, 10, 15]
g) [8, 2, 8, 7,4]
h) [-1, 3, 28, 17, 9, 30]

3) Write a class `Book`. The class should have:

- Four *instance variables*. They represent the title (String), the price(double), the year was publiashed (int) and the name of the author (String).
- One *constructor* to inizialise title, year, name of the author.
- Write "setter" and "getter" methods for all of the instance variables.
- Write the toString() method for the book class that return a string with the information about the book.

4) Write a program that saves in an arrayList all the books that the user inserts from the keyboard. Implement a menu console from where the user can insert a book with the relative information (title, price, year, author's name).

Sorting an ArrayList<Object> with Comparable

5) Print the list of book sorted by year.

First implement `Comparable` interface and then Override the compareTo method.

```
public  class Book implements Comparable<Book>{
…
…
public int compareto(Book b){
    // compare here the book based on the year
}
```

Now, in your main, you can call `Collections.sort` on ArrayList in order to sort your list of books.

HashMap

`HashMap` maintains key and value pairs and often denoted as `HashMap<Key, Value>`. HashMap implements Map interface. 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. You must need to `import java.util.HashMap` in order to use the HashMap class and methods.

6) Consider now that each book will be placed in a numbered shelf in a book shop. Use Java map collection to map each book with the shelf. For each book the user will enter the number of the shelf where it will be placed. Note that in one shelf you can placed different books.

```
public static void main(String args[]) {
…
…
      /* This is how to declare HashMap */
      HashMap<Book, Integer> hmap = new HashMap<Book, Integer>();

   // insert the key for each book
```

Note that if you need to store primitive data types, you will need to use their corresponding wrapper class objects

```
for (int i=0; i<bookList.size(); i++){

      System.out.printl ("Please, enter the number of the shelf where is
      placed the book" + bookList.get(i).getTitle());

      int numShelf = s.nextInt();

      /*Adding elements to HashMap*/
      hmap.put(bookList.get(i), numShelf);
}
```

Key: the key is unique! In this case is the book entity

Value: the number of the shelf. In one shelf can be placed several books.

7) Write some code that lets the user insert a shelf number from the keyboard and then the list of book in that shelf will be printed.

   Hint: you can check all the values in a HashMap using an Iterator. Iterator is an object that enables to cycle through a collection, obtaining or removing elements. Read carefully the following code on how to use an iterator:

```
System.out.println("Insert the number of the shelf");
int selectedShelf = s.nextInt();
System.out.println("The book in shelf " + selectedShelf + " are:");

/* Display content using Iterator*/
      Set set = hmap.entrySet();
      Iterator iterator = set.iterator();
      while(iterator.hasNext()) {
         Map.Entry entry = (Map.Entry)iterator.next();

         if(selectedShelf == (int) entry.getValue()){
              System.out.println(((Book)entry.getKey()).getTitle());
         }

}
```

entrySet(): Returns a set view of the mappings contained in this map. Each element in the returned set is a Map.Entry.

Cast entry to be int and Book respectively

Extra Exercises that you can do as indipendent study

8) Implement a queue in Java using Array or ArrayList. Write the push and pop functionality.

   You should implement a Generic Class! In this way your queue will work for any type (String, int, instance of specific objects, etc.).
   In order to implement a generic class you have to declare your Queue class in the following way:

```
public class Queue<T> {

    private int maxSize;
    private ArrayList<T> queue;

    public Queue(int size) {…}
    public void enQueue(T value) {…}
```

T is the generic representation of the type!

```
    public T deQueue() {…}
    public String toString() {…}
}
```

In the method where Queue is instanciated you need to include, before the method's return type, the type parameter, inside angle brackets:

```
public static <T> void main(String[] args) {
        Queue newQueue = new Queue(5);
        newQueue.enQueue("a");
        newQueue.enQueue("b");
        newQueue.enQueue("c");
        newQueue.enQueue("e");
        newQueue.enQueue("f");

        System.out.println((T) newQueue.toString());
        System.out.println((T) newQueue.deQueue().toString());
        System.out.println((T) newQueue.deQueue().toString());
        System.out.println((T) newQueue.toString());
    }
```

9) Singly Linked Lists are a type of data structure. In a singly linked list each node in the list stores the contents of the node and a pointer or reference to the next node in the list. Watch the following video about the linked list: https://westminster.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=1573494f-a6a7-47ac-98a7-af4100c16a5c

According to what you learnt in the video and in the lecture (Lecture Week 05 from slide 43 to slide 49) implement a generic Singly-Linked List. Watch the following video about the linked list

- Implement a class Node that will have the contents of the node and a pointer or reference to the next node in the list. It does not store any pointer or reference to the previous node. Write the get and set methods and a CompareTo method to compare if two Nodes are the same.

- Implement a SinglyLinked List. Remember that to store a single linked list, you only need to store a reference or pointer to the first node in that list. The last node has a pointer to nothingness to indicate that it is the last node. Implement the following methods:

```
public void add(T newElement);
public void addAfter(T newElement, T after);
public void deleteFront();
public void deleteAfter(T after);
public void traverse();
```

- Implement a main class where you can instanciate the Linked list and try the operations you implemented.