BIRLA INSTITIUTE OF TECHNOLOGY AND SCIENCE, PILANI
CS F213
LAB-5

AGENDA	**DATE: 09/09/2014**	**TIME: 02 Hours**

1. Understand the concept of Arrays, passing arrays to a method
2. Multi Dimensional Arrays
3. Strings, StringBuffer and StringTokenizer

1 Single dimensional array and passing an array as parameter

Array concept is similar in java and c in many aspects. Array is a collection of homogeneous items. An array object contains a number of variables. The number of variables may be zero, in which case the array is said to be *empty*. The variables contained in an array have no names; instead they are referenced by array access expressions that use non-negative integer index values. These variables are called the *components* of the array. If an array has *N* components, we say *N* is the length of the array; the components of the array are referenced using integer indices from 0 to *N* - 1, inclusive.

All the components of an array have the same type, called the *component type* of the array. If the component type of an array is T, then the type of the array itself is written T[].

Let us consider an example where in a retail shop we have various items which are identified by their item IDs and each of them have an amount associated with it. You have to write a program to print the amount to purchase an item based on the item index.

RetailStore
- itemId : int[]
- price :double[]
+ computePrice(value: int): double

The Class has two array as variables and one method to compute the price.

Example 1

RetailStore.java

```
Public class RetailStore {
     private int[] itemId = {1001,1002,1003,1004,1005};
     private double[] price = {13.50, 18.00, 19.50, 25.50};

     private double computePrice(int value) {
         // method to compute the price of the item.
        // it returns the price
        for (int i = 0; i < price.length; ++i) {
         // note the use of price.length.
        // it gives the length of the array
          if (itemId[i] == value) {
               return price[i];
          }
        }

        // method which takes in the index and
         // returns the price of the item
        return price[value];
     }

     public static void main(String[] args) {
          //main method. Execution begins here
          RetailStore retailOne = new RetailStore();
          System.out.println("price of item id 1002 is "
                                    +retailOne.computePrice(1003));
          System.out.println("price of item id 1004 is "
                                     +retailOne.computePrice(1004));
           /* System.out.println("price of item id 1009 is "
                                    +retailOne.computePrice(1007));*/
          /* un-comment the above line and see the output.
           * Why there is no compilation error?*/
     }
}
```

1.1 Exercise -

- ✓ Uncomment the last lines in the above code and see the output
- ✓ Is there any error or runtime exception? [
- ✓ Why compiler is not showing the error?. (hint:- you can check If the value passed in the computePrice() method exists in the itemId array. In java there is no array bound checking. So you have to be very careful while accessing the elements in an array. This runtime exception can be avoided by ensuring that the array element to be accessed is already initialized)

2. Strings, StringBufffer and StringTokenizer.

class String

java.lang.String

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. [**You can refer to lecture slides or java API docs for more details on String class, its constructors and methods**]

class StringBufffer

java.lang.StringBuffer

A string buffer is like a String, **but can be modified (mutable).** At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls. [**Refer to lecture notes or java API docs for more details on StringBuffer class, its constructors, and methods**]

StringTokenizer

java.util.StringTokenizer

To use this class you have to import **java.util.StringTokenizer**. The string tokenizer class allows an application to break a string into tokens. The StringTokenizer methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments. The set of delimiters (the characters that separate tokens) can be specified at the time of instantiation. A token is returned by taking a substring of the string that was used to create the StringTokenizer object. StringTokenizer is a legacy class that **is retained for compatibility reasons although its use is discouraged in new code**. [**Refer to lecture notes or java API docs for more details on StringBuffer class, its constructors, and methods**]

2.1 Exercise -

You are given the skeleton code for four incomplete classes named ***Name, Student, StudentList*** and a driver class named ***Test***. You have to complete the code for all the classes as per the following specifications:

(a) class Name: Name class encapsulates three attributes of a person. **name** such as **first name**, **middle name** and **last name**. This class supplies only one constructor which receives a string value and this string value contains the values for all the three attributes either in comma (,) or semicolon (;) separated format.

If the values are comma separated then the three attribute values are in the following order:
<First name>,<Middle Name>,<Last Name>
If the values are semicolon separated then the three attribute values are in the following order:
<Last name>;<Middle Name>;<First Name>

For example: If the value supplied for constructor parameter is **"Rajesh,Kumar,Khanna"** the First name is "Rajesh", Middle Name is "Kumar" and Last name is "Khanna" . If the value

supplied for constructor parameter is **"Khanna;Kumar;Rajesh"** then Last name is "Khanna", Middle name is "Kumar" and First name is "Rajesh". ***[Assume string parameter for constructor either contains comma or semicolon in its value but not both. There is no need to check for validity of the string parameter.]*** The Name class supplies accessor methods for every instance field. The class also supplies a method **getName()** for retrieving the full name of a person. The **getName()** method returns the full name after concatenating and adding spaces between first name, middle name and last name fields in order. The class also supplies **toString()** method which returns value after simply concatenating the values of first name, middle name and last name fields.

The skeleton code for the class "Name" is given below:

```
class Name {
    private String fname; // First Name
    private String mname; // Middle Name
    private String lname; // Last Name
    // provide accessor methods as per the given specification
    // provide implementation for getName() method as per the given
    // specification
    Name(String name) {
       /* Complete the constructor by extracting the values of three name
          fields. Note that name value may be either comma separated or
          semicolon separated */
          // Write Your Code Here
    }
} // End of class Name.
```

(b) class Student: Student class has two attributes: ***name*** of type ***Name*** [note that Name is class as mentioned above] and ***age*** of type ***int***. The class supplies only one parameterized constructor which receives the values for all instance fields of the class as parameters. First parameter is of ***Name*** type and second is of type ***int***. The class supplies accessor method for every instance field and **toString()** method which returns a string after concatenating and adding spaces between values of first name , middle name , last name and age attributes for this instance. ***Provide the implementation for the class "Student" as mentioned below as per the specification given above.***

```
class Student {
    private Name name; // name of student
    private int age;// age of student
    /* Complete the Student class by adding proper constructor,
       accessor methods and by adding any other method which
       are required as per specification */
    // Write Your Code From Here
}// End of Student class
```

(c) class StudentList: This class encapsulates the list of size 10 of type Student. This class contains only static fields and methods. The list of students is maintained as an array of type Student[ ]. The skeleton code is given as follows:

```
class StudentList {

    public static Student[] list = new Student[10]; // list of students
    public static int count =0; // count of students added in the list

    public static void addStudent(Student std) {
       if(count >= 20) return; // if count is already 20 or more then return
       list[count] = std;
       count++;
    }

    public static Student[] getStudentsWithAge(int age) {
       /* This method returns all the students whose age is equal to age
          parameter of this method. If no such student is found then it
          returns null. */
       // Write Your Code From Here
    }

    public static Student[] getStudentsWithLastName(String lastName) {
       /* This method returns all the students whose last name attribute
          value matches with lastName parameter of this method. If no such
          students is found then it returns null. */
          // Write Your Code From Here
    }

    public static Student[] getStudentsInRange(int minAge, int maxAge) {
       /* This method returns all the students whose age falls between minAge
          and maxAge (both parameters inclusive) */
          // Write Your Code From Here
    }
}// End of class StudentList
```

(d) class Test: This class is the driver class. The incomplete code for the class is given below. *You have to complete methods readStudent() and main() of this class as per commented specification.*

```
class Test {
   public static Student readStudent() throws IOException {
   /* This method reads the student details and returns the Student instance.
      Values to be read from System.in are:
      1. First name of Student, 2. Middle name of student, 3. Last name of
      Student, 4. Name format (1 for comma(,) separated and 2 for semicolon
      separated), 5. age of student
   */
   } // End of readStudent() Method
   public static void main(String args[]) throws IOException {
   /* 1. Write java code for reading details of 10 students and add them
      in the static list of StudentList class.*/
   /* 2. Write java code for displaying the all the students with age 20 from
      static list field of StudentList class */
   /* 3. Write java code for displaying the student details for all students
      having last name “Sharma” from static list of StudentList class*/
   /* 4. Write java code for displaying all the students whose age falls in
      the range minAge = 16 and maxAge = 20 from static list of StudentList
      class*/
   }// End of main() Method
}// End of Test class
```

3. Two dimensional Array

For declaring 2 dimensional arrays there are many types of syntax available. Will show a set of examples and you can try this code. Please note the various types of initialization syntax used below.

Example 2

TwoDExample.java

```
public class TwoDExample {
	public static void main(String[] args) {
		// main method
		int[][] multi = new int[5][10];
		// most commonly used way of initializing
		int[][] multi1 = new int[][] {
				{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
				{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
				{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
				{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
				{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

		// above way shows the detailed initialization of array
		int[][] mult2 = new int[5][];
		for (int i = 0; i < 5; i++) {
			mult2[i] = new int[10];
		}

		// above method shows initializing elements with for loop
		int[][] multi3 = new int[5][];

		multi3[0] = new int[10];
		multi3[1] = new int[10];
		multi3[2] = new int[10];
		multi3[3] = new int[10];
		multi3[4] = new int[10];

		/* using this above method the arrays are initialized only by mentioning
		   the row numbers first */
		// note you can use nested for loop for printing the array
	}
}
```

Example 3

The source code of above Example1 (RetailStore) is modified to include a new variable called ItemName. Also note the use of constructor in the new code which is more suiting real world scenario. Compare the codes and identify the difference in both.

Now create a new class RetailStoreExample which extends the RetailStoreClass. Remember that the child class will be able to access the parent class's members which are public and protected. The detailed code is available below. Now run the RetailStoreExample class (Remember that execution always starts from main method)

Also this new example covers the use of various Methods belonging to the classes String, StringBuffer and StringTokenizer.

RetailStore.java

```
public class RetailStore {
	private int[]itemId;
	private double[] price;
	private String itemName[];

	/* The constructor is used here for the initialization purpose*/
	public RetailStore() {
		itemId = new int[] { 1001, 1002, 1003, 1004, 1005 };
		price = new double[] { 950.00, 750.00, 450.00, 350.00, 250.00 };
		itemName = new String[] {
				"Yonex Tennis Racket-950", "Yonex Badminton Racket-750",
				"Silvers Badminton Racket-450", "Cosco Badminton shuttle-350",
				"Cosco Tennis Racket-250" };
	}

	protected double computePrice(int value) {
		// method to compute the price of the item. it returns the price
		for (int i = 0; i < price.length; ++i) {
			// note the use of price.length. it gives the length of the array
			if (itemId[i] == value) {
				return price[i];
			}

		}
		return price[value];
	}

	protected String fetchDescription(int value) {
		// method to compute the description of the item. it returns the
		// description
		for (int i = 0; i < price.length; ++i) {
			// note the use of price.length. it gives the length of the array
			if (itemId[i] == value) {
				return itemName[i];
			}

		}
		return null;
	}
}
```

RetailStoreExample.java

```
public class RetailStoreExample extends RetailStore {

	public static void main(String[] args) {
		int index;
		RetailStore retailOne = new RetailStore();

		String description = retailOne.fetchDescription(1004);

		// below line illustrates the use of split function of String class
		String StringArray[];
		// below line split the string and stores the splitted values to an
		// array
		StringArray = description.split("\\s");

		/* this commented code illustrates the use of StringTokenizer to achieve
		 * the same functionality of split method
		 *
		 * StringTokenizer st = new StringTokenizer(Description);
		 * StringArray = new String[st.countTokens()];
		 * for (int i=0; st.hasMoreTokens(); i++) {
		 *       StringArray[i]=st.nextToken();
		 * }
		 */
		String type = StringArray[2];
		System.out.println("the type of the item is " + type);
		System.out.println("the charactor at starting position is "
				+ type.charAt(0));

		// below line will find the location of the symbol "-"
		index = type.indexOf('-');

		String stringFromSubstring = type.substring(index + 1);
		System.out.println("the price computed using the substring method is "
				+ stringFromSubstring);

		String stringFromDouble = Double.toString(
					new RetailStore().computePrice(1004));
		System.out.println("the price of the item computed using double.toString
		method is " + stringFromDouble);
	}
}
```

3.1 Exercise –

Include lines of code in the above program to compare the Strings stringFromDouble and stringFromSubstring . We require to show that both the Strings are representing the same value. Use substring method and String comparison method (String.equals) and show that both the strings represent the same value.

3.2 Exercise –

(A) Consider a class named '***Address***' which encapsulates the address of any particular person having attributes as :

– line1 : String
– line2 : String
– line3 : String
– city : char[]
– state : char[]
– pin : String .

The class supplies only one parameterized constructor which receives only one parameter of type ***String*** which embeds the values of all the attributes in $ separated form as per following format: ***"line1$line2$line3$city$state$pin"***
$ Character is used as a separator to separate the values of ***line1, line2, line3, city, state and pin*** attributes. The class supplies accessor methods for every instance field. All accessor methods return only ***String*** type value. Implement the ***Address*** class in java as per mentioned specification.

(b) Considering the availability of the code of class ***Address*** of (A) in this question, complete the implementation of the following class named '***AddressList'*** as per commented specification given below

```
class AddressList {
   public static int countAddressWithCity(Adress[] addressList, String city) {
   /* This method returns the count of the addresses from addressList which have the
      city attribute equals to city parameter passed for this method. If the length
      of any passed argument is zero or value of any passed argument is null then it
      returns -1. */
   }// End of method countAddressWithCity()
   public static int countAddressWithPin(Adress[] addressList, String strP) {
   /* This method returns the count of the addresses from addressList which have the
      pin attribute starting with strP parameter passed for this method. If the
      length of any passed argument is zero or value of any passed argument is null
      then it returns -1. */
   }// End of method countAddressWithCity()
   public static Address[] getAddressWithCity(Adress[] addressList, String city) {
   /* This method returns all the addresses from addressList by storing them in
      String[] which have the city attribute equals to city parameter passed for this
      method. If the length of any passed argument is zero or value of any passed
      argument is null then it returns null. If addressList does not contain any
      address with city attribute value equal to city parameter passed for this
      method even then it returns null. */
   }// End of method getAddressWithCity()
   public static Address[] getAddressWithPin(Adress[] addressList, String strP){
   /* This method returns all the addresses from addressList by storing them in
      String[] which have their pin attribute starting with strP parameter passed for
      this method. If the length of any passed argument is zero or value of any
      passed argument is null then it returns null. If addressList does not contain
      any address whose pins attribute value starts with strP parameter passed for
      this method even then it returns null. */
   }// End of method getAddressWithCity()
}// End of class AddressList
```

(c) Write a suitable driver class named ***Test*** for a class named 'AddressList' and test the behavior of all the method.