OBJECT-ORIENTED PROGRAMMING
LAB 2: ARRAY, ARRAYS CLASS, STRING CLASS

I. Objective

In this second tutorial, you will:

- Practice with an **array** in Java.
- Understand how to program with **Arrays** class.
- Have basic knowledge about the object, how to create the object in Java, and practice it with the sample class defined by Java.
- Understand how to program with **String** class.

II. Array

Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.
To ***declare*** an array, you can use this statement:

```
dataType[] arrayName;
```

You can also place the brackets after the array's name:

```
dataType arrayName[];
```

But we encourage you should use the first form to declare an array.
There are two basic ways to **initialize** an array. You can use the **new** operator or use the braces to list the values:

```
dataType[] arrayName = new dataType[arraySize];
dataType[] arrayName = {value0, value1, ..., valueK};
```

Example:

```
public class MyProgram {
    public static void main(String[] args) {
        int[] a = {1, 2, 3, 4, 5};
        int[] b = new int[3];
        b[0] = 1;
        b[1] = 2;
        b[2] = 3;

        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println();
        int sum = 0;
        for (int i = 0; i < b.length; i++) {
            sum = sum + b[i];
        }
        System.out.println("sum = " + sum);
    }
}
```

Another way, you can use this statement to create an anonymous array:

```
new dataType[] {value0, value1, ..., valueK};
```

Example:

```
public class MyProgram {
    public static int sum(int[] arr) {
        int result = 0;
        for(int i = 0; i < arr.length; i++) {
            result += arr[i];
        }
        return result;
    }
    public static void main(String[] args) {
        System.out.println("sum = " + sum(new int[] {2,4,6,8}));
    }
}
```

We have a special for loop in the above sample called *enhanced for* loop. The *enhanced for* loop is mainly used to traverse a collection of elements including arrays. The syntax is as follows.

Example:

```
public class MyProgram {
    public static void main(String[] args) {
        int[] a = {1, 3, 5, 7, 9};
        for (int x : a) {
            System.out.println(x);
        }
        int sum = 0;
        for (int x : a) {
            sum = sum + x;
        }
        System.out.println("sum = " + sum);
    }
}
```

III. *Arrays* Class

This class contains various static methods for manipulating arrays. Many of the methods have unique specifications for each of the primitive types (boolean, byte, char, short, int, long, float, double). Though only the methods for the primitive types are specifically discussed, many of the methods also support an array of elements of type *Object* and generic types. You can read more about this class here.

To simplify the presentations of these methods, ***ptype*** will be used as a placeholder for a primitive type. The following, we will introduce some typical methods of this

1. static *ptype*[ ] copyOf(*ptype*[ ] original, int newLength)

Copies the specified array of primitive types, truncating or padding (if needed) so the copy has the specified length. If padding is necessary, the numeric types will pad with `zero`, char will pad with `null`, and boolean will pad with `false`.

Example:

```
int[] org = new int[] {1, 2, 3, 4, 5};
int[] copy = Arrays.copyOf(org, 3);   //copy = [1, 2, 3];
```

2. static *ptype*[ ] copyOfRange(*ptype*[ ] original, int beginIndex, int endIndex)

Copies the range `beginIndex` to `endIndex-1` of the specified array into a new array. The index `beginIndex` must lie between zero and `original.length` inclusively. As long as there are values to copy, the value at `original[beginIndex]` is placed into the first element of the new array, with subsequent elements in the original array placed into subsequent elements in the new array. Note that `beginIndex` must be less than or equal to `endIndex`. The length of the returned array will be `endIndex - beginIndex`.

Example:

```
int[] org = new int[] {1, 2, 3, 4, 5};
int[] copy = Arrays.copyOfRange(org, 1, 6); //copy = [2, 3, 4, 5, 0];
```

3. String toString(*ptype*[ ] a)

Return the string representation of the contents of the specified array, the resulting string consists of a list of the array's elements, separated by a comma and a space, enclosed in square brackets ("[ ]"). It returns "null" if the array is `null`.

Example:

```
int[] org = new int[] {1, 2, 3, 4, 5};
String copy = Arrays.toString(org, 3);
System.out.println(copy); // [1, 2, 3]
```

4. static void sort(*ptype*[ ] a)

Sorts the specified array into ascending numerical order. This method is not defined for `boolean`.

Example:

```
int intArr[] = {10, 20, 15, 22, 35};
Arrays.sort(intArr);
System.out.println(intArr); // [10, 15, 20, 22, 35]
```

5. static *ptype* binarySearch(*ptype*[ ]a, *ptype* key)

Searches the array for the `key` value using the binary search algorithm. The array must be sorted before making this call. If it is not sorted, the results are undefined. If the array contains duplicate elements with the key value, there is no guarantee which element will be found. For floating point types, this method considers all NaN values to be equivalent and equal. The method is not defined for boolean.

Example:

```
int intArr[] = {10, 20, 15, 22, 35};
Arrays.sort(intArr);
int index = Arrays.binarySearch(intArr, 22);
System.out.println(index); //3
```

Students can use the functionality provided by the Arrays class for array processing after the midterm exam. During the midterm exam, students are not permitted to use these methods. All array-related problems must be solved manually.

IV. Classes and objects in Java

Java is an *Object-Oriented Programming* (OOP) language. Everything you work in Java is through classes and objects. In this lab, we only learn how to create an object from the available class in Java, we will learn about this topic more carefully in Lab 4.

BigDecimal is a class, which is defined in **java.math** package. This class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion. Besides that, the *BigDecimal* class gives its user complete control over rounding behaviour. You can read more about this class here.

Object is an instance of a Class.

Example:

```
import java.math.BigDecimal;

public class Test {
    public static void main(String[] args) {
        BigDecimal num = new BigDecimal(1);
        BigDecimal num1 = new BigDecimal(4);
        BigDecimal x = num;
        // BigDecimal y;
        System.out.println(num);
        System.out.println(num1);
        System.out.println(x);
        // System.out.println(y);
    }
}
```

Output:

```
1
4
1
```

With the above example, the variables: *num, num1, x,* and *y* were created from the BigDecimal class. In other words, the variables: *num, num1, x,* and *y* are the pointers to the objects. However, variable y hasn't been initialized. Therefore, the value of that variable will be undermined until an object is created and assigned to it. If you try using an uninitialized variable, you will get a compiler error.

The format of code when you want to create an object from the class is:

```
ClassName instanceName = new ClassConstructor([paramater1, parameter2, …])
```

Let's observe another example:

```
import java.math.BigDecimal;

public class Test1 {
    public static void main(String[] args) {
        BigDecimal x = new BigDecimal(4);
        BigDecimal y = new BigDecimal(4);

        System.out.println(x == y);
        System.out.println(x.equals(y));
    }
}
```

Output:

```
false
true
```

With objects, we don't use the operator "==" to compare values, because this operator will compare the addresses and there are the pointers so they have different addresses. *BigDecimal* supports the method called **equals** to compare the value of two *BigDecimal* objects.
Next, we will learn how to invoke the **non-static method** and **static method** from a class. To invoke a static method, you do not need to create an object. Instead, you can invoke directly from the class name. Contrarily, with the non-static method, you need to create an object to invoke it.
Example:

```
import java.math.BigDecimal;

class Test2 {
    public static void main(String[] args) {
        BigDecimal x = new BigDecimal(-4);

        //abs() is a non-static method of the BigDecimal class
        BigDecimal y = x.abs();

        //valueOf() is a static method of the BigDecimal class
        BigDecimal z = BigDecimal.valueOf(20.22);

        System.out.println(y);
        System.out.println(z);
    }
}
```

Output:

```
4
20.22
```

V. *String* class

Java provides a class named **String** in the package `java.lang` to support non-mutable strings. A non-mutable string cannot be changed once it has been created. Instances of the `String` class can be combined to form new strings, and numerous methods are provided for examining **String** objects. A **String** variable contains a collection of characters surrounded by double quotes. You can read more about this class here.

Create a String object by string literal and by ***new*** keyword:

```
String greeting = "Hello"; // by string literal
String greeting1 = new String("Hello"); // by new keyword
```

1. int length()

A String in Java is an object, which contains methods that can perform certain operations on strings. For example, the length of a string can be found with the `length()` method:

```
String greeting = "Hello";
System.out.println("The length of the greeting string is: " + greeting.length());
// The length of the greeting string is: 5
```

2. char charAt(int index)

You can reference the individual characters in a string by using the method `charAt()` with the same index that you would use for an array.

```
String greeting = "Hello";
System.out.println(greeting.charAt(1)); // e
```

3. int compareTo(String anotherString)

You should not use the “`==`” operator to test whether two strings are equal. Using the “`==`” operator determines only whether the references to the strings are the same; it does not compare the contents of the **String** instances. You can compare strings by using the `compareTo()` method. Not only can you determine whether two strings are equal, but you can determine which of two strings comes before the other according to the Unicode value of the character in the strings.

```
"Star".compareTo("star"); // returns negative
"abc".compareTo("abc"); // returns zero
"d".compareTo("abc"); // returns positive
```

4. String concat(String str)

You can use the `concat()` method to concatenate two strings. You can also concatenate two strings to form another string by using the "+" operator.

```
"Hello".concat(" Wolrd");

String greeting = "Hello";
greeting += " Wolrd";
```

Besides adding two strings together, you can also concatenate a string and a value of a primitive type together by using the + operator.

```
String string = "PI = " + 3.14;
// PI = 3.14
```

5. String substring(int begin) / String substring(int begin, int end)

Using `substring()` to access part of a string.

```
String title = "J Perfect's Diary";
System.out.println(title.substring(2)); // Perfect's Diary
System.out.println(title.substring(2, 9)); // Perfect
```

6. int indexOf(String str, int fromIndex)

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

7. String replace(char oldChar, char newChar)

Returns a string that is obtained by replacing all characters `oldChar` in the string with `newChar`.

8. boolean equals(String str)

The method compares the two given strings based on the content of the string. If any character is not matched, it returns *false*. If all characters are matched, it returns *true*.

You can't use "==" to compare two String, because with "==" you compare object references, not the content of the string.

9. String[] split(String regex)

The method splits this string against a given regular expression and returns a *String array*.

10. String trim()

Returns a string that has all leading and trailing spaces in the original string removed.

11. char[] toCharArray()

Converts this string to a new character array.

12. String toUpperCase() / String toLowerCase()

Converts all of the characters in this String to upper case / lower case.

IV. Exercises

a) Array (Do not use the Arrays class in the following exercises)

1. Write a function `public static int findMax(int[] arr)` to find the maximum value of an array.
2. Write a function to sum all even numbers of an array.
3. Write a function to count how many prime numbers are in an array.
4. Write a function `public static int find(int[] arr, int k)` to find the index of an element *k* in an array, if the element doesn't exist in an array return -1. (the first element index is 0)
5. Write a function `public static void square(int[] arr)` to square all elements of an array.
6. Write a function `public static int[] divisibleNumbers(int[] arr, int k)` to find the elements divisible by *k* in an array. (*Hint: Count how many possible elements and create a new array with a length equal to the number counted. After that, put all possible elements into the new array.* Ex: arr = [1,2,3,4,5,6,7] with *k* = 2 → [2,4,6]).
7. Write a function to find the third largest element in an array.
8. Write a function to remove the first specific element from an array and return *true*, if the element does not exist in an array return *false*.
9. Write a function to insert an element at a specific position into an array. (After insertion you can replace the last element with the element before)
   (Ex: arr = [1,2,4,3] insert 5 at position 2 returns [1,2,5,4])
10. Write a function to find the duplicate values of an array of integer values.
    (Ex: arr = [1,3,1,3,2,4] returns [1,3])
11. Write a function to remove the duplicate values of an array of integer values.
    (Ex: arr = [1,3,1,3,2,4] returns [1,3,2,4])

b) Class and Object

1. In the *main* method, create an array of five **BigDecimal** objects with arbitrary values.
2. Write a function `public static BigDecimal findMax(BigDecimal[] arr)` to find the maximum value of a BigDecimal object array.
3. Call the `findMax` function with the BigDecimal array above and print the result to the screen.

c) String

1. Write a Java program that receives the full name of a person and writes functions to:
    - Return the first name and last name, except the middle name.
      (Ex: Nguyen Van Chien → Nguyen Chien)
    - Return the middle name. (Ex: Nguyen Thi Thu Thao → Thi Thu)
    - Capitalize the full name. (Ex: nguyen van chien → Nguyen Van Chien)
    - Uppercase all vowels and lowercase all consonants. (Ex: Nguyen Van Chien → ngUyEn vAn chIEn)
2. Write a Java program that receives a string and writes functions to:
    - Find the length of a string.
    - Count the number of words in a string.
    - Find the longest word in a string.
    - Count the number of occurrences of a character in a string.
    - Check if a string is a palindrome or not.

-- END --