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

I. Objective

After completing this tutorial, you can:

- Understand how to program with **Arrays**.
- Understand how to program with **String**.

II. *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.

To simplify the presentations of these methods, ***ptype*** will be used as a placeholder for a primitive type.

1. 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. 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` muse lies between zero and `original.length`, inclusive. 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. 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. sort(*ptype*[ ] a)

Sorts the array into ascending order. For floating point values, the method uses the total order imposed by the appropriate `compareTo()` method and all `NaN` values are considered equivalent and equal. This method is not defined for `boolean` and `short`.

Example:

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

5. 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 one 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` or `short`.

Example:

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

III. The class *String*

Java provides a class **String** in the package `java.lang` to support nonmutable strings. A nonmutable string is one that 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.

Create a variable of type String and assign it a value:

```
String greeting = "Hello";
```

1. length()

A String in Java is actually 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. 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. compareTo(String str)

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 table.

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

4. 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. 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, 9)); // Perfect
```

6. indexOf(String str, int fromIndex)

Returns the index of the first substring equal to `str` starting from the index `fromIndex`.

7. replace(char oldChar, char newChar)

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

8. equals(String str)

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 and not the content of the string.

9. split(String regex)

Method splits this string against given regular expression and returns a char array.

10. trim()

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

IV. Exercises

1. Write a Java program **(Do not use *Arrays* class, just use the primitive array)**:
   - Write function to find the index of an element in an array.
   - Write function to remove a specific element from an array.
   - Write function to insert an element (specific position) into an array.
   - Write function to find the maximum value of an array.
   - Write function to find the minimum value of an array.
   - Write function to find the duplicate values of an array of integer values.
2. Write a Java program:
   - Write function add two matrices of the same size.
   - Write function multiply a matrix with a number.

- Write function print a matrix to screen in matrix format.
- Write main function and run all above functions.

3. Write a Java program:
   - Enter the full name.
   - Split the full name into first name and last name.
   - Capitalize the full name.
4. Write a Java program:
   - Find the length of the string.
   - Count number of words in string.
   - Concatenate one string contents to another.
   - Check a string is palindrome or not.
5. Write a Java program:
   - Remove leading white spaces from a string.
   - Remove trailing white spaces from a string.
   - Remove extra spaces from a string.