OBJECT-ORIENTED PROGRAMMING
LAB 3: ARRAYS CLASS, STRING CLASS

I. Objective

After completing this tutorial, you can:

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

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

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. 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:

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` 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. 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. The method uses the total order imposed by the appropriate compareTo() method for floating point values, 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 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.

III. *String* class

Java provides a class **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:

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

1. 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:

```java
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.

```java
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.

```java
"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.

```java
"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)

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. split(String regex)

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

10. 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

1. Write a Java program **(Do not use the *Arrays* class, use the primitive array like Lab 2)**:

   - 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*.

- 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])

- 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])

- 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])

2. To create an integer 2-dimensional array (matrix), you can use the following statements:

```java
int[][] arr = new int[2][3] // [[0 0 0]
                            //  [0 0 0]]
int[][] arr1 = {{1, 2, 3}, {4, 5, 6}} // [[1 2 3]
                                       //  [4 5 6]]
```

Write a Java program:

- Write a function to add two matrices of the same size.

- Write a function to multiply a matrix with a number.

- Write a function to print a matrix to screen in matrix format.

- Write a main function and run all the above functions.

3. 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)

4. Write a Java program:

- Find the length of the string.

- Count the number of words in the string.

- Concatenate one string's contents to another.

- Check if a string is a palindrome or not.

5. Write a program to count how many times one word appears in the paragraph. Expect result is the 2-dimension array in which the first column contains only one word, and the second column contains its frequency. Words are not case-sensitive.

   For example, given the string:

   "You are living on a Plane. What you style Flatland is the vast level surface of what I may call a fluid, on, or in, the top of which you and your countrymen move about, without rising above it or falling below it."

   The example output may be like this figure (the order of the words is not obligated to follow this example):

```
C:\Users\Admin\Desktop\Java Exercise\Solve\Lab3>java WordsFrequency
'You': 3,
'are': 1,
'living': 1,
'on': 2,
'a': 2,
'Plane': 1,
'What': 2,
'style': 1,
'Flatland': 1,
'is': 1,
'the': 2,
'vast': 1,
'level': 1,
'surface': 1,
'of': 2,
'I': 1,
'may': 1,
'call': 1,
'fluid': 1,
'or': 2,
'in': 1,
'top': 1,
'which': 1,
'and': 1,
'your': 1,
'countrymen': 1,
'move': 1,
'about': 1,
'without': 1,
'rising': 1,
'above': 1,
'it': 2,
'falling': 1,
'below': 1
```