Chapter 02 Arrays, Characters and Strings

1. Arrays

1.1 Array Basics

Once an array is created, its size is fixed. An array reference variable is used to access the elements in an array using an index.

1.1.1 Declaring Array Variables

Syntax for declaring array variables:

```
type[] arrayVar;
```

or

```
type arrayVar[]; // Allowed, but not preferred
```

Note: You can use `type arrayVar[];` to declare an array variable. This style comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers. The style `type arrayVar[];` is preferred.

The `type` can be any data type, and all elements in the array will have the same data type. For example, the following code declares a variable `myArray` that references an array of double elements.

```
double[] myArray;
```

or

```
double myArray[]; // Allowed, but not preferred
```

1.1.2 Creating Arrays

Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array. If a variable does not contain a reference to an array, the value of the variable is `null`. **Note**: Java uses `null` instead of NULL.

You cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the `new` operator and assign its reference to the variable with the following syntax:

```
arrayVar = new type[size];
```

This statement does two things:

- (1) it creates an array using `new type[size]` and
- (2) it assigns the reference of the newly created array to the variable `arrayVar`.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement as:

```
datType[] arrayVar = new datType[size];
```

or

```
datType arrayVar[] = new datType[size];
```

Here is an example of such a statement:

```
double[] myArray = new double[10];
```

This statement declares an array variable, `myArray`, creates an array of 10 elements of `double` type, and assigns its reference to `myArray`. To assign values to the elements, use the syntax:

```
arrayVar[index] = value;
```

1.1.3 Array Size and Default Values

The size of an array cannot be changed after the array is created. Size can be obtained using `arrayVar.length`.

Example 01: Get the length of an array (i.e., the number of elements in the array).

```
class Example {
    public static void main(String[] args){
        double[] myArray = new double[10];

        System.out.print(myArray.length);
    }
}
```

Output

```
10
```

When an array is created, its elements are assigned the default value of `0` for the numeric primitive data types, `\u0000` (`null` character. The shortcut for `null` is `'\0'`) for `char` types, and `false` for `boolean` types.

1.1.4 Initializing Arrays

To initialize an array, we combine the declaration, creation, and initialization of the array in one statement, for example:

```
double[] myArray = {1.9, 2.9, 3.4, 3.5};  // note that the new operator is not used here
```

which is equivalent to the following statements:

```
double[] myArray = new double[4];
myArray[0] = 1.9;
myArray[1] = 2.9;
myArray[2] = 3.4;
myArray[3] = 3.5;
```

Splitting declaration, creation, and initialization would cause a syntax error. For example:

```
double[] myArray;
myArray = {1.9, 2.9, 3.4, 3.5}; // Error
```

1.1.5 Accessing and Processing Arrays

The array elements are accessed through the index. Array indices are 0 based; that is, they range from 0 to `arrayVar.length - 1`.

Each element in the array is represented using the following syntax:

```
arrayVar[index];
```

When processing array elements, you will often use a for loop for one of two reasons:

- 1. All of the elements in an array are of the same type. They are evenly processed in the same fashion repeatedly using a loop.
- 2. Since the size of the array is known, it is natural to use a for loop.

Example 02: Find the maximum element of the array.

```
class Example {
    public static void main(String[] args){
        int[] myArray = {7, 2, 9, 5, 4, 6};
        int max = myArray[0];

        for(int i = 1; i < myArray.length; i++) {
            if(myArray[i] > max) {
                max = myArray[i];
            }
        }
        System.out.print("Max = " + max);
    }
}
```

Output

```
Max = 9
```

1.1.6 Foreach Loops

Foreach loop enables you to traverse the array sequentially without using an index variable.

The syntax of the Java for-each loop or range-based loop is:

```
for(dataType item : arrayVar) {
    //...
}
```

Here,

- `arrayVar`: an array or a collection
- `item`: each item of array/collection is assigned to this variable
- `dataType`: the data type of the array/collection

Example 03: Using foreach.

```
class Example {
    public static void main(String[] args) {
        String[] names = {"John", "Lucy", "Jackson"};

        for(String item : names){
            System.out.print(item + "\t");
        }
    }
}
```

Output

```
John    Lucy    Jackson
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Accessing an array out of bounds is a common programming error that throws a runtime. To avoid it, make sure you do not use an index beyond `arrayVar.length - 1` or simply using a foreach loop if possible.

1.2 Anonymous Arrays

Usually, you create a reference variable for the array, and later access its elements through the variable name. Occasionally, you may create an array and use it only once. In this case, you do not have to name it.

Syntax for creating anonymous array is:

```
new elementType[]{value0, value1, ..., valuek};
```

Example 04: Using anonymous arrays.

```
class Example {
    public static void main(String[] args) {
        for(String item : new String[]{"John", "Lucy", "Jackson"}){
            System.out.print(item + "\t");
        }
    }
}
```

Output

```
John    Lucy    Jackson
```

1.3 Copying Arrays

To copy the contents of one array into another, you have to copy the array's individual elements into the other array.

Often, in a program, you need to duplicate an array. In such cases, you could attempt to use the assignment statement (=), as follows:

```
array2 = array1; // will not work
```

However, this statement does not copy the contents of the array referenced by `array1` to `array2`, but instead merely copies the reference value from `array1` to `array2`. After this statement, `array1` and `array2` reference the same array, as shown in Figure 7.4. The array previously referenced by `array2` is no longer referenced; it becomes garbage, which will be automatically collected by the Java Virtual Machine. This process is called garbage collection.

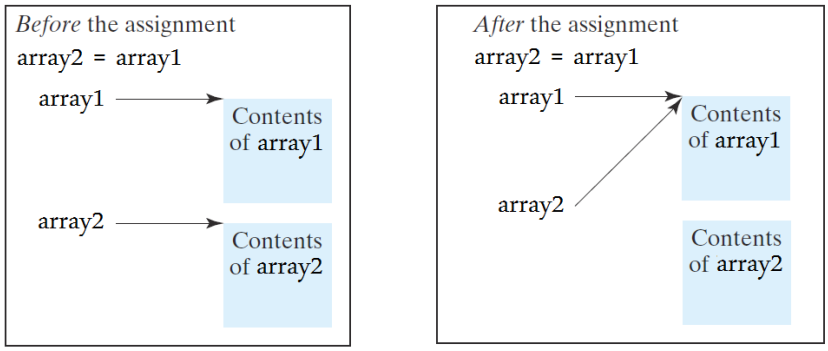


FIGURE 7.4 Before the assignment statement, array1 and array2 point to separate memory locations. After the assignment, the reference of the array1 array is passed to array2.

You can write a loop to copy every element from the source array to the corresponding element in the target array.

Example 05: Copy arrays by using a loop to copy individual elements one by one.

```
class Example {
    public static void main(String[] args) {
        int[] sourceArray = {2, 3, 1, 5, 10};
        int[] targetArray = new int[sourceArray.length];

        // Copy the arrays
        for (int i = 0; i < sourceArray.length; i++) {
            targetArray[i] = sourceArray[i];
        }

        //Display the array after copied
        for(int item : targetArray){
            System.out.print(item + " ");
        }
    }
}
```

Output

```
2 3 1 5 10
```

We can also copy arrays by using the `static arraycopy`() method in the `System` class instead of using a loop.

The syntax for `arraycopy` is:

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, length);
```

Here,

- `srcPos` indicate the starting positions in `sourceArray`.
- `tarPos` indicate the starting positions in `targetArray`.
- `length` indicate the number of elements copied from `sourceArray` to `targetArray`.

Example 06: Copy arrays by using the `static arraycopy()` method in the `System` class.

```
class Example {
    public static void main(String[] args) {
        int[] sourceArray = {2, 3, 1, 5, 10};
        int[] targetArray = new int[sourceArray.length];

        // Copy the arrays
        System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);

        //Display the array after copied
        for(int item : targetArray){
            System.out.print(item + " ");
        }
    }
}
```

Output

```
2 3 1 5 10
```

Note: The `arraycopy` method does not allocate memory space for the target array. The target array must have already been created with its memory space allocated. After the copying takes place, `targetArray` and `sourceArray` have the same content but independent memory locations.

The `arraycopy` method violates the Java naming convention. By convention, this method should be named arrayCopy (i.e., with an uppercase C).

1.4 The `Arrays` Class

The `java.util.Arrays` class contains various static methods for sorting and searching arrays, comparing arrays, filling array elements, and returning a string representation of the array. These methods are overloaded for all primitive types.

You can also use the `toString()` method to return a string that represents all elements in the array. This is a quick and simple way to display all elements in the array.

Example 07: Using `toString()` method.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        int[] arr = {2, 4, 7, 6, 3, 10};

        System.out.println(Arrays.toString(arr));
    }
}
```

Output:

```
[2, 4, 7, 6, 3, 10]
```

You can use the `sort()` or `parallelSort()` method to sort an array. `parallelSort()` method is more efficient if your computer has multiple processors.

Example 08: Sorts the whole array of characters partially using `sort()` and `parallelSort()` methods.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        char[] arr = {'E', 'A', 'C', 'F', 'D', 'B'};

        // Sort the array
        Arrays.sort(arr);  // Sort the whole array
        //Arrays.parallelSort(arr);

        System.out.println(Arrays.toString(arr));
    }
}
```

Output

```
[A, B, C, D, E, F]
```

To sort an array in descending order, we use `Arrays.sort()` and `Collections.reverseOrder()` methods. However, it is not possible to sort an array of primitives in descending order using the `Arrays.sort()` and `Collections.reverseOrder()` methods, since the comparator work on the Wrapper classes and objects instead of the primitives.

Example 09: Sort an array in descending order.

```
import java.util.Arrays;
import java.util.Collections;

class Example {
    public static void main(String[] args) {
        Integer[] arr = {4, 2, 3, 1, 5};

        Arrays.sort(arr, Collections.reverseOrder());
        //Arrays.parallelSort(arr, Collections.reverseOrder());

        System.out.println(Arrays.toString(arr));
    }
}
```

Output

```
[5, 4, 3, 2, 1]
```

Example 10: Sort an array of Strings.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        String[] countries = {"Canada", "japan", "Cambodia", "america","korea",};

        Arrays.parallelSort(countries, String.CASE_INSENSITIVE_ORDER);
        //Arrays.sort(countries, String.CASE_INSENSITIVE_ORDER);

        System.out.println(Arrays.toString(countries));
    }
}
```

Output

```
[america, Cambodia, Canada, japan, korea]
```

You can use the `binarySearch`() method to search for a key in an array. The array must be **pre-sorted in increasing order**. If the key is not in the array, the method returns `-(insertionIndex + 1)`

Example 11: Search the indexes in an array of integers.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        int[] arr = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66};
        System.out.println("11 is at index " + Arrays.binarySearch(arr, 11));
        System.out.println("60 is at index " + Arrays.binarySearch(arr, 60));
        System.out.println("9 is at index " + Arrays.binarySearch(arr, 9));
        System.out.println("3 is at index " + Arrays.binarySearch(arr, 3));
        System.out.println("0 is at index " + Arrays.binarySearch(arr, 0));
        System.out.println("70 is at index " + Arrays.binarySearch(arr, 70));
    }
}
```

Output:

```
11 is at index 4
60 is at index 8
9 is at index -4
3 is at index -2
0 is at index -1
70 is at index -11
```

Example 12: Search the indexes in an array of characters.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        char[] arr = {'a', 'c', 'g', 'x', 'y'};
        System.out.println("a is at index " + Arrays.binarySearch(arr, 'a'));
        System.out.println("z is at index " + Arrays.binarySearch(arr, 'z'));
    }
}
```

Output:

```
a is at index 0
z is at index -6
```

You can use the `equals`() method to check whether two arrays are strictly equal. Two arrays are strictly equal if their corresponding elements are the same.

Example 13: In the following code, `arr1` and array are equal, but `arr2` and `arr3` are not.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        int[] arr1 = {2, 4, 7, 10};
        int[] arr2 = {2, 4, 7, 10};
        int[] arr3 = {4, 2, 7, 10};

        System.out.println(Arrays.equals(arr1, arr2)); // true
        System.out.println(Arrays.equals(arr2, arr3)); // false
    }
}
```

Output:

```
true
false
```

You can use the `fill()` method to fill in all or part of the array. For example, the following code:

Example 14: Fill an array `arr` with `5`.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        int[] arr = {2, 4, 7, 6, 3, 10};

        Arrays.fill(arr, 5); // Fill 5 to the whole array

        for (int item : arr)
            System.out.print(item + " ");
    }
}
```

Output:

```
5 5 5 5 5 5
```

Example 15: Fill `8` into elements `arr[1]` through `arr[4 - 1]`.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        int[] arr = {2, 4, 7, 6, 3, 10};

        Arrays.fill(arr, 1, 4, 8); // Fill 8 to a partial array
                                   // from arr[1] to arr[4 - 1]
        for (int item : arr)
            System.out.print(item + " ");
    }
}
```

Output:

```
2 8 8 8 3 10
```

2. Character Data Type and Operations

`char` is used to represent a single character. A character literal is enclosed in single quotation marks.

2.1 ASCII Code

Most computers use ASCII (American Standard Code for Information Interchange), an 8-bit encoding scheme, for representing all uppercase and lowercase letters, digits, punctuation marks, and control characters. Table 4.4 shows the ASCII code for some commonly used characters.

TABLE 4.4 ASCII Code for Commonly Used Characters

Characters	*Code Value in Decimal*
`'0'` to `'9'`	48 to 57
`'A'` to `'Z'`	65 to 90
`'a'` to `'z'`	97 to 122

2.2 Casting Between `char` and Numeric Types

A `char` can be cast into any numeric type, and vice versa.

2.2.1 Implicit Casting

Implicit casting can be used if the result of a casting fits into the target variable. Otherwise, explicit casting must be used.

A data type of lower size (occupying less memory) is assigned to a data type of higher size. This is done implicitly by the JVM. This is also named as automatic type conversion.

2.2.2 Explicit Casting

A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size. This is not done implicitly by the JVM and requires explicit casting; a casting operation to be performed by the programmer.

Syntax:

(dataType) expression

When a floating-point value is cast into a `char`, the floating-point value is first cast into an `int`, which is then cast into a `char`.

```
char chr = (char)65.25; // Decimal 65 is assigned to chr
```

When a character is cast into a numeric type, the character is cast into the specified numeric type.

```
int i = (int)'A'; // Decimal 65 is assigned is assigned to i
```

`byte` → `short` → `int` → `long` → `float` → `double`

In the above statement, left to right can be assigned implicitly and right to left requires explicit casting. That is, byte can be assigned to short implicitly but short to byte requires explicit casting.

2.2.3 Boolean Casting

A `boolean` value cannot be assigned to any other data type. Except `boolean`, all the remaining 7 data types can be assigned to one another either implicitly or explicitly; but `boolean` cannot. We say, `boolean` is incompatible for conversion.

2.3 `char` Methods

Java provides the following methods in the `Character` class for testing characters as listed in Table 4.6. The `Character` class is defined in the `java.lang` package.

TABLE 4.6 Methods in the Character Class

Method	*Description*
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

Example 16: Using Character methods

```
class Example {
    public static void main(String[] args) {
        char chr1 = 'D', chr2 = '2', chr3 = '?';

        System.out.println(Character.isLetter(chr1));
        System.out.println(Character.isDigit(chr2));
        System.out.println(Character.isLetterOrDigit(chr3));
    }
}
```

Output:

```
true
true
false
```

3. String

`String` (note the capital S) is a predefined class in the Java library, just like the classes `System` and `Scanner`. For example:

```
String message = "Welcome to Java";
```

The `String` type is not a **primitive type**. It is known as a **reference type**. Any Java class can be used as a reference type for a variable. The variable declared by a reference type is known as a **reference variable** that references an object. Here, `message` is a reference variable that references a string object with contents `"Welcome to Java"`.

Note: When a variable holds an object, it actually holds a reference to that object. A reference to an object is `null` when the variable no longer refers to any object.

3.1 String Methods

Strings are objects in Java. The methods listed in Table 4.7 below can only be invoked from a specific string instance.

For this reason, these methods are called **instance methods**. A non-instance method is called a **static method**. A static method can be invoked without using an object. All the methods defined in the `Math` class are `static` methods. They are not tied to a specific object instance. For example, the `pow`() method in the `Math` class can be invoked using `Math`.`pow`(2, 2.5).

TABLE 4.7 Simple Methods for `String` Objects

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

Example 17: Using String methods.

```
class Example {
    public static void main(String[] args) {
        String str = "  Welcome to Java  ";

        System.out.println("Length of the str is " + str.length());
        System.out.println("Charecter at index 0 is " + str.charAt(5));
        System.out.println(str.toUpperCase());
        System.out.println(str.toLowerCase());
        System.out.println("After trimmed length is " + str.trim().length());
    }
}
```

Output:

```
Length of the str is 19
Charecter at index 0 is c
  WELCOME TO JAVA
  welcome to java
After trimmed length is 15
```

Note: You must use the method `charAt`(index) instead of the [ ] operator. [ ] operator is not supported for String type in Java because strings are objects.

The String class contains the methods, as listed in Table 4.8, for comparing two strings.

TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if `s1` is a substring in this string.

To compare two strings, you might want to use == operator.

```
if (str1 == str2){ // this does not compare strings
    //code
}
```

However, the == operator checks only whether `str1` and `str2` refer to the same object; it does not tell you whether they have the same contents. Therefore, you cannot use the == operator to find out whether two string variables have the same contents. Instead, you should use the `equals`() method.

You also cannot use the comparison operators (<, <=, >, >=) to compare strings. Instead, you can use `compareTo`() method.

Example 18: Compare two strings.

```
public class Example {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "Hi";

        if(str1.equals(str2)){
            System.out.println("str1 is equal to str2");
        }
        else {
            if (str1.compareTo(str2) > 0)
                System.out.println("str1 is greater than str2");
            else
                System.out.println("str2 is greater than str1");
        }
    }
}
```

Output:

```
str2 is greater than str1
```

You can obtain a single character from a string using the `charAt()` method. You can also obtain a substring from a string using the substring method (see Figure 4.2) in the `String` class.

TABLE 4.9 The `String` Class Contains the Methods for Obtaining Substrings

Method	*Description*
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 4.2. Note the character at `endIndex` is not part of the substring.

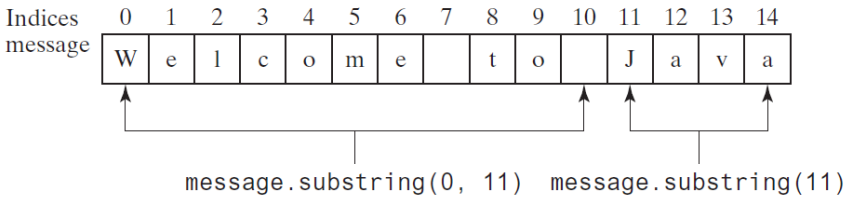


FIGURE 4.2 The `substring` method obtains a substring from a string.

The `String` class provides several versions of `indexOf()` and `lastIndexOf()` methods to find a character or a substring in a string.

TABLE 4.10 The `String` Class Contains the Methods for Finding Substrings

Method	*Description*
`indexOf(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

More String methods here: https://www.programiz.com/java-programming/library/string

3.2 Concatenating Strings

Using the + operator is the most common way to concatenate two strings in Java. You can provide either a variable, a number, or a String literal.

Example 19: Concatenate strings.

```
class Example {
    public static void main(String[] args) {
        String str1 = "Welcome";
        String str2 = "Java";
        String str3 = str1 + " to " + str2;

        System.out.println(str3);
    }
}
```

Output:

```
Welcome to Java
```

Example 20: Concatenate strings.

```
class Example {
    public static void main(String[] args) {
        int num1 = 10;
        String str1 = "Java is " + "awesome.";
        String str2 = "num1 is " + num1;
        String str3 = "The first alphabet is " + 'a';
        String str4 = "2 * 5 is " + 2 * 5;

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
    }
}
```

Output:

```
Java is awesome.
num1 is 10
The first alphabet is a
2 * 5 is 10
```

3.3 Immutable Strings and Interned Strings

`String` in Java is a very special class, as it is used almost in every Java program. That is why it is Immutable to enhance performance and security.

`String` class is immutable. That means we cannot modify a string object once it is created. The only feasible solution is to create a new string object with the new characters.

Example 21:

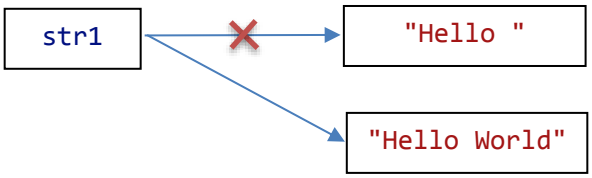
```
class Example {
    public static void main(String[] args) {
        String str1 = "Hello ";
        str1 = str1 + "World";

        System.out.println(str1);
    }
}
```

Output:

```
Hello World
```

We can understand the above example with the help of the following diagram:



The string object that contains `"Hello "` remains unchanged, and a new string object is created with `"Hello World"`. It shows that the strings are immutable. You explicitly have changed the reference variable to point to the `"Hello World"`.

3.4 Splitting Strings

Example 22: Split a string by whitespace and store its parts in an array using the `split()` method.

```
import java.util.Arrays;

class Program {
    public static void main(String[] args) {
        String str = "Welcome to Java";
        String[] array = str.split(" ");

        System.out.println(Arrays.toString(array));
    }
}
```

Output:

```
[Welcome, to, Java]
```

3.5 Conversion Between Strings and Numbers

To convert a string into an int value, use the `Integer.parseInt()` method, for example:

```
int intValue = Integer.parseInt("123");
```

To convert a string into a double value, use the `Double.parseDouble()` method, for example:

```
double doubleValue = Double.parseDouble("123.45");
```

If the string is not a numeric string, the conversion would cause a runtime error. The `Integer` and `Double` classes are both included in the `java.lang` package, and thus they are automatically imported.

You can convert a number into a string; simply use the string concatenating operator, for example:

```
String s = 125 + "";
String s = 32.56 + "";
```

4. Generating Random Numbers

Many times, we require using random numbers in our application to produce simulations or games and other applications that require random events. For example, in a game of dice, without having random events, we will have the same side popping up every time we throw the dice. In the physical environment, we can have random events generated but it is not possible when it comes to computers.

`Math` class provides a `random()` method that generates a random `double` value (`0 <= Math.random() < 1.0`). You can use it to write a simple expression to generate random numbers in any range. For example,

Example 23: Generate random numbers.

```
class Example {
    public static void main(String[] args) {

        // Random integer between between 0 and 9
        int num1 = (int)(Math.random() * 10);

        // Random integer between between 50 and 99
        int num2 = 50 + (int)(Math.random() * 50);

        // Random integer between a and a + b, excluding a + b
        int a = 5, b = 12;
        int num3 = (int)(a + Math.random() * b);

        System.out.println(num1 + "\t" + num2 + "\t" + num3);
    }
}
```

Output

```
5    91    10
```

In some cases, you might need to shuffle arrays or strings. Let's see how we can shuffle an array using `random()` method.

Example 24: A program that shuffles an array.

```
import java.util.Arrays;

class Example {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        int random_index, temp;

        // Shuffle the array
        for (int i = 0; i < array.length; i++){
            random_index = (int)(Math.random() * 5);  // generate a number
                                                      // between 0 and 4
            if(i == random_index)
                continue;

            // swap the elements
            temp = array[i];
            array[i] = array[random_index];
            array[random_index] = temp;
        }

        // Display the array
        System.out.print(Arrays.toString(array));
    }
}
```

Output

```
[2, 4, 1, 5, 3]
```

Exercises

Write the following programs:

1. Ask the user to enter a sentence, then displays the longest word within the sentence.
2. Ask the user to enter a sentence, then capitalizes the first character of each word in the sentence.
3. Create an array of countries names, then sorts the array alphabetically. Note: use case insensitive.
4. Ask the user to enter an integer number in a range of [0, 999], then display the number in English. Example: 123456 is one hundred twenty-three thousand, four hundred fifty-six.
5. (Game: locker puzzle) There are 456 lockers and 456 players. Each player has to pick one locker number, then enters a builder. When the game starts, all the lockers are closed. As the players enter the building, the first player (P1) has to open every locker. Then the second player (P2) begins with the second locker (L2), and has to close every other locker (L2 to L456). Player P3 begins with the third locker (L3) and has to change (closes the locker if it was open, and open it if it was closed) every third locker (L3, L6, …). Player P4 begins with locker L4 and changes every fourth locker (L4,

L8, …). Player P5 starts with L5 and changes every fifth locker (L5, L10, …), and so on, until Player P456 changes L456.

After all the players have passed through the building and changed the lockers, the game ends. The players whose locker closed will be killed.

Write a program to display all open locker numbers separated by exactly one space and ten per line. How many open lockers in total? (Hint: Use an array of 456 bool elements, each of which indicates whether a locker is open (true) or closed (false).)

6. Ask the user to enter a hexadecimal number (which can be an integer number or floating-point number). If the value entered is a hexadecimal number, display a message "Thank you", otherwise, display a message "This is not a hexadecimal".
7. (*Decimal to binary*) Ask the user to enter a number (which can be an integer number or floating-point number) and displays its corresponding binary value. Note: you are not allowed to use any library method to do the conversion.
8. (*Binary to decimal*) Ask the user to enter a binary number (which can be an integer number or floating-point number) and displays its corresponding decimal value. Note: you are not allowed to use any library method to do the conversion. Also, the input must be case insensitive.
9. (*Decimal to hex*) Ask the user to enter a number (which can be an integer number or floating-point number) and displays its corresponding hexadecimal value. Note: you are not allowed to use any library method to do the conversion.
10. (*Hex to decimal*) Ask the user to enter a hexadecimal number (which can be an integer number or floating-point number) and displays its corresponding decimal value. Note: you are not allowed to use any library method to do the conversion. Also, the input must be case insensitive.
11. (*Hex to binary*) Ask the user to enter a hexadecimal number (which can be an integer number or floating-point number) and displays its corresponding binary value. Note: you are not allowed to use any library method to do the conversion. Also, the input must be case insensitive.
12. (*Binary to hex*) Ask the user to enter a binary number (which can be an integer number or floating-point number) and displays its corresponding hexadecimal value. Note: you are not allowed to use any library method to do the conversion. Also, the input must be case insensitive.
13. (Game: scissor, rock, paper) Write a program that plays the popular scissor-rock-paper game. (A scissor can cut a paper, a rock can knock a scissor, and a paper can wrap a rock.) The program randomly generates a number 0, 1, or 2 representing scissor, rock, and paper. The program asks the user to enter a number 0, 1, or 2 and displays a message indicating whether the user or the computer wins, loses, or draws. The program will allow a user to play until they win. Here are sample runs:

```
scissor (0), rock (1), paper (2): 2 Enter
The computer is paper. You are paper too. It is a draw.

scissor (0), rock (1), paper (2): 1 Enter
The computer is scissor. You are rock. You won.
```

14. (Game: hangman) Write a hangman game that randomly generates a word and prompts the user to guess one letter at a time, as presented in the sample run. Each letter in the word is displayed as an asterisk. When the user makes a correct guess, the actual letter is then displayed. When the user finishes a word, display the number of misses and ask the user whether to continue to play with another word. Declare an array to store words, as follows:

```
// Add any words you wish in this array
String[] words = {"write", "that",...};
```

```
(Guess) Enter a letter in word ******* > p Enter
(Guess) Enter a letter in word p****** > r Enter
(Guess) Enter a letter in word pr**r** > p Enter
     p is already in the word
(Guess) Enter a letter in word pr**r** > o Enter
(Guess) Enter a letter in word pro*r** > g Enter
(Guess) Enter a letter in word progr** > n Enter
     n is not in the word
(Guess) Enter a letter in word progr** > m Enter
(Guess) Enter a letter in word progr*m > a Enter
The word is program. You missed 1 time
Do you want to guess another word? Enter y or n>
```

15. (Simulation picking a pearl) A screen flashes in front of you and explains you the game. There are six pearls kept in a bowl in front of you and an empty bowl. Among the six pearls, three are white and three are black. You can divide all pearls as you like into the two bowls as long as each bowl has at least one pearl. Once you are done, you will pull a lever, which will turn the room pitch black. The bowls will move and shuffle around. In the dark, you have to pick up one pearl from any bowl. If the pearl you have in your hand is white, you will be allowed to live, but if the pearl you picked is black, the room will be filled with poisonous gas and you will die.

    How would you divide the pearls to increase your chances of survival? Write a program that simulates you picking a pearl one hundred times for each of every possible choice you can make. Display the number of your survival in each of your choice.

16. (Simulation firing a gun) For this game, six players will play together. Five bullets are put into a gun's round barrel which can hold up to six bullets.

    A staff from the game will start pointing the gun at the first player and firing the gun. If no bullet is fired, the player survives, otherwise, the player gets shot and die. The staff will do the same to all players one by one. Note that the staff will roll the barrel before each shooting.

    Which turn (1st to 6th) would you choose to increase your chance of survival? Write a program to simulate this gun shooting game one thousand rounds, and display the number of each player getting shot based on the turn they take to get shot.

Reference

[1] Y. Daniel Liang. 'Introduction to Java Programming', 11e – 2019

[2] https://www.programiz.com/java-programming