CL309
Object Oriented Analysis and Design Lab
Lab # 02

Conditional operators, loops, and arrays

Objectives:

- Introducing the logic of if and if-else statements.
- Reading user input.
- Making decisions based on user input.
- Using switch-case statement to make decisions based on some value.
- Understand the concept of arrays.
- Defining and using arrays in different types.
- Multidimensional arrays.
- Strings as arrays.

1. if and if-else Statements

Sometimes we need to make a decision between two choices or two actions based on some information or conditions. The statements in by which we make this decision are called conditional statements.

Choices are made by one or more conditions being satisfied or not, we express those conditions by boolean expressions, that is, if the result of the boolean condition is true then the condition is met and the appropriate action(s) is made.

We use if statement to determine whether the condition is met or not. The format of if statement is as follows:

```
if(condition){
//statement(s) to be executed if the condition is true
}
For example, we can check whether some value is positive or negative:
int x = 5;
if(x > 0){
System.out.println("x is positive");
}
else{
System.out.println("x is negative");
}
```

Notice that we use else block to enclose statements that we want to perform if the condition is NOT met. The problem of the above statement is that 0 is considered negative. To solve this problem, we can use >= operator instead of >, but if we wish to detect 0 itself, we can use nested if statement as follows:

```
int x = 5;
if(x > 0){
        System.out.println("x is positive");
}
else if(x < 0){
        System.out.println("x is negative");
}
else if(x == 0){
        System.out.println("x is zero");
}
```

Using else before each if statement means that if one of the conditions is met no other condition will be checked.

2. Reading User Input

You can read input from the user using BufferedReader class in this way:

```
BufferedReader in = new BufferedReader(
new InputStreamReader(System.in));
String input = in.readLine();
```

Before using BufferedReader, be sure to import the library which contain this class, this library is called java.io, in order to import it, we use import statement as follows.

```
import java.io.*;
```

By using character * here we identify that we import all classes in this library. Import statements are used before class definition.
To use BufferedReader class, you need to deal with the exception it throws through readLine() method which is IOException, for instance, we will do this by changing the definition of main() method to throw that exception.

```
public static void main(String[] args) throws IOException{
```

Butting it all together, a simple program that reads user name and prints it on screen.

```
import java.io.*;
class ReadFromUser{
	public static void main(String[] args) throws IOException{
		String userName;
		BufferedReader in;
		in = new BufferedReader(
		new InputStreamReader(System.in));

		System.out.println("Enter your name: ");
		userName = in.readLine();
		System.out.println("Welcome " + userName);
	}
}
```

The above code asks the user to enter his name, reads the input using readLine() method which returns a String value that represents user input from keyboard, then concatenates it with "Welcome" and prints it on the screen.

3. Decision Making Based on User Input

It is clear now that you can write a program that responds to user actions depending on the input provided by user. For instance, we will write a program that uses if-else to determine that the number provided by user is positive, negative or zero, as well as determining whether the number is odd or even.

```
import java.io.*;

class Numbers{
	public static void main(String[] args) throws IOException{
		BufferedReader in;
		in = new BufferedReader(
		new InputStreamReader(System.in));

		int	x;
		String	userInput;
		System.out.print("Enter a number: ");
		userInput = in.readLine();
		x = Integer.parseInt(userInput);

		if(x == 0)
			System.out.println("zero");
```

```
                //x is positive
                else if(x > 0){
                        //x is even
                        if(x % 2 == 0)
                                System.out.println("positive and even");

                        //x is odd
                        else
                                System.out.println("positive and odd");

                }

                //x is negative
                else{
                        //x is even
                        if(x % 2 == 0)
                                System.out.println("negative and even");

                        //x is odd
                        else
                                System.out.println("negative and odd");

                        }
        }//end of main()
}//end of class
```

As you can see, we converted the String we read from user to integer because it is easier to compare, another reason may lead us to this conversion, is that switch-case statement accepts only integers and numerical values or characters, and it cannot accept strings. The following section covers switch-case statement.

4. switch-case Statement

This statement is a special type of decision making techniques, it makes choice among several options based on a numerical value. The format of switch-case statement is the following:

```
switch(variable){
        case value_1:
                //Statement(s) to be executed if
                //variable = value_1
                break;
        case value_2:
```

```
                //Statement(s) to be executed if
                //variable = value_2
                break;
        ... case value_n:
                //Statement(s) to be executed if
                //variable = value_n
                break;
    }
```

As you can see, we put some variable in switch then follow it by several case blocks. Execution goes through cases one by one comparing each one with the value of the variable specified in switch statement, then executes it and all cases that follows, unless we use break keyword to avoid the execution from expanding to unwanted statements. The following example shows how to use switch-case statement.

```
    import java.io.*;
    class SwitchCaseDemo{
        public static void main(String[] args) throws IOException{
            BufferedReader in;
            in = new BufferedReader(
            new InputStreamReader(System.in));
                System.out.println("1. Print your name");
                System.out.println("2. Print your age");
                System.out.println("3. Exit");

        int choice;
        choice = Integer.parseInt(in.readLine());

        switch(choice){
        case 1:
            System.out.print("Enter your name: ");
            String name = in.readLine();
            System.out.println("Welcome " + name");
            break;
        case 2:
            System.out.print("Enter your age: ");
            String age = in.readLine();
            System.out.println("You are " + age);
            break;
        case 3:
            System.exit(0);
        }//End switch
    }//End main
}//End class
```

A special case that is executed without checking the value of the variable is called default case, we use default keyword instead of case keyword to specify the default case. Because we usually use break after each case, default block is executed when the value of the variable did not match any of specified cases value. To see how this works, add a default case to our previous SwitchCaseDemo class (shown in boldface) and try to enter any value other than 1, 2 and 3.

```
switch(choice){
case 1:
        System.out.print("Enter your name: ");
        String name = in.readLine();
        System.out.println("Welcome " + name");
        break;
case 2:
        System.out.print("Enter your age: ");
        String age = in.readLine();
        System.out.println("You are " + age);
        break;
case 3:
        System.exit(0);
default:
        //User did not enter 1, 2 or 3
        System.out.println("Invalid choice");
}//End switch
```

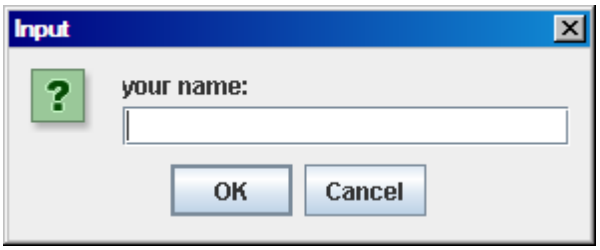
Lab Extras

There is an easier way to read from user, which depends on showing a dialog box for the user and let him enter his input in a text box inside that dialog. To use this input dialog we import the library called javax.swing, this library contains classes that deal with GUI. What we interested in among the classes of this library is a class called JOptionPane and a method inside it called showInputDialog(). The following example shows how to use this technique to read from user.

```
import javax.swing.*;

class EasyRead{
public static void main(String[] args){
        String userInput;
        userInput = JOptionPane.showInputDialog("your name: ");
        System.out.println("Hello " + userInput + " !");
        }
}
```

You can notice here that we do not declare any objects and no exceptions to throw. An easy and straight forward method. The output of the above code is a dialog box like the image shown below. If user clicks OK the input is read and returned as String, if he click Cancel dialog box closes without returning any input entered.



5. What Are Loops?

We use loops to make one or more statements repeat for specified number of times. Clearly this have an importance when we wish to take more than one input from user or do some calculations. For example, to calculate the sum of numbers from 1 to 10, we define a variable named sum, initialize it with 0, then use another variable i to have values from 1 to 10. The following sections discuss loops in more details.

5.1. while and do-while Loops

This is the simplest type of loops, it consists of a condition (boolean expression) and a body which contain one or more statements; as long as condition is true, statement(s) in the body keep to execute, after each execution, the condition is checked again, this implies that if the condition was initially false, statements in loop body will never execute. The format of while loop is the following.

```
while(condition){
//Statement(s) to execute as long as condition is true.
//Also known as loop body.
}
```

Here is an example of while loop:

```
class Loops{
public static void main(String[] args){
                //Prints numbers from 1 to 10
                int x = 1;
                while(x <= 10){
                        System.out.println(x);
                        x++;
                }
        }
}
```

It is clear that the value of x affected the execution of the loop, as you can derive from the output, the condition is true until x reaches 11. We can also use break to exit the loop.

```
int x = 1;
while(true){ //infinite loop
        System.out.println(x);
        if(x == 10){
        break;
}
}
```

using break in the above code exits the loop when x = 10.
The do-while loop like while loop uses boolean expression to determine whether or not go to next iteration. The difference between it and while loop is the sequence of execution, do-while loop body appears before the condition, this implies that even if the condition is false, the loop will execute at least one time. The format of the do-while loop is the following:

```
do{
        //Loop body
}while(condition);
```

if we repeat the last example using do-while, the code will look like this:

```
int x = 1;
do{
        System.out.println(x++);
}while(x <= 10);
```

The output will be the same of previous example. So how can we clearly see the difference? Consider the following two loops.

```
//while
int x = 0;

while(x != 0){
     System.out.println(x);
}
```

Output:

```
//do-while
int x = 0;

do{
     System.out.println(x);
}while(x != 0);
```

Output:

```
0
```

As you can see, although the condition was initially false, do-while loop executed one time, but while loop was not executed.

5.2. for Loop

The base of for loop is similar to while and do-while loops; but there is slight different in format that makes for loop more powerful, specially with numerical values.

Let's first have a look at for loop format, after that we will discuss its contents.

```
for(initialization; test-condition; post-statements){
//Loop body
}
```

We can notice that for loop consists of three major parts:

- **Initialization:** a statement to be executed before the loop begins, this statement is usually used to define a variable or set of variables that control the flow of the loop.
- **Test-condition:** This is the boolean expression that is used to determine whether or not the loop should execute the next iteration, similar to conditions previously discussed in while and do-while loops.
- **Post-statements:** A set of statements (separated by commas ',') to be executed after each iteration. These statements are usually used to change (increment or decrement) the value of controlling variable(s) declared in initialization, sometimes we call it post-increment.

To better understand the concept of for loop, write and execute the following program.

```
class ForLoopDemo{
public static void main(String[] args){
        for(int x = 1; x <= 10; x++){
                System.out.println(x);
                }
        }
}
```

The flow of execution of for loop is the following:
1. Initialization is executed once.
2. Test-condition is checked to determine whether or not loop body should be executed.
3. Loop body execution.
4. Post-statements are executed.
5. Back to step 2 through 4 until the stop condition is false, then loop exits.

Another example of for loop is the program that prints out the results of multiplications from 1 to 12.

```
class MultiplicationTable{
public static void main(String[] args){
        for(int i = 1; i <= 12; i++){
                for(int j = 1; j <= 12; j++){
                        int k = i * j;
                        System.out.println(i + " * " + j + " = " + k);
                        }
                }
        }
}
```

In the above example, we used nested loops, that is, a loop inside the body of another loop, we can use as many levels of nesting as we need. Beware that when we use nested loops, the inner loop executes until the end in one iteration of the outer loop, because the whole inner loop is one statement in the body of outer loop.

Lab Extras

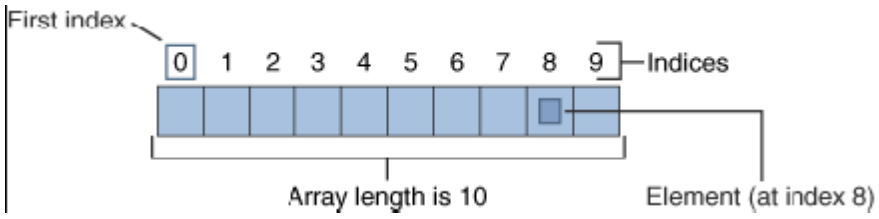
We use continue keyword to skip some statements inside loop body and start new iteration, consider the following program that prints out factors of 30.

```
for(int i = 1; i <= 30; i++){
if(30 % i != 0){ continue; }
        System.out.println(i);
}
```

As you can see, when 30 is not divisible by i, we use continue to jump to the next iteration without executing the next line that prints the output.

6. What Are Arrays?

Arrays are special data types that let us store specified number of variables from the same type using one variable name. Imagine the situation that you need to store 20 names of students as strings and 20 integers as marks, then you need to define 40 variables, and this is clearly very hard and not practical, in such case you need to use arrays. Arrays are indexed data types, that means they are storing different elements discriminating between them using unique index for each one.



We will refer to array maximum size as array length. It is also clear that indices of an array are zero-based, that is, they start from 0 to length – 1; for example, the array shown in the figure has a length of 10 (stores up to 10 elements), and the last index is 9.

6.1. Using Arrays in Java

Declaring and using an array in Java is similar to declaration of any variable; you have to specify data type and name, in addition to this, you have also to specify the length of the array before using it. Array definition format is as follows:

```
DataType[] arrayName;
```

To initialize the array, use new keyword and specify the length.

```
arrayName = new DataType[length];
```

Here are some examples of array declaration:

```
int[] a = new int[6]; //Array of integers of length 6
String[] b = new String[10]; //Array of strings of length 10
```

To access array members, use array name with desired index specified between square brackets [ ].

```
int[] x = new int[3];
x[0] = 5; //Store 5 in the first index
x[1] = 10; //Store 10 in the second index
x[2] = 15; //Store 15 in the third index
System.out.println(x[2] + x[1]); //Prints 25
```

If you try to access an index out of the range of array (greater or equals array length) JVM will throw an IndexOutOfRangeException.

```
int[] x = new int[3]; //Valid indices are 0, 1 and 2
x[3] = 7; //IndexOutOfRangeException
```

Another technique to declare an array is to directly provide its members between brackets { and }, separated using commas ','.

```
int[] x = {10, 15, 20, 25, 100}; //Length = 5
System.out.println(x[3]); //Prints 25
String[] days = {"Sat", "Sun", "Mon", "Tue", "Wed", "Thu", "Fri"};
//We are leaving in Monday
System.out.println("We are leaving in " + days[2]);
```

It is common to deal with arrays using for loop, we can use length property to determine the length of the array. The following code stores multiples of 3 in an array then prints them from greater to smaller.

```
int[] a = new int[20];
for(int i = 0; i < a.length; i++){
        a[i] = i * 3;
}
for(int i = a.length – 1; i >= 0; i—){
        System.out.print(a[i] + " ");
}
```

The following example stores names and numbers of n students and performs search.

```
import javax.swing.*;
class Students{
public static void main(String[] args){
        int[] numbers;
        String[] names;
        String input = JOptionPane.showInputDialog("Enter number of students");
        int size = Integer.parseInt(input);
        numbers = new int[size];
        names = new String[size];

        for(int i = 0; i < numbers.length; i++){
                names[i] = JOptionPane.showInputDialog("Enter Student name");
                input = JOptionPane.showInputDialog("Enter Student number");
        numbers[i] = Integer.parseInt(input);
        }
}
```

Now we have all numbers and names, index will bind two arrays together, that is, student in index 5 has his name in names[5] and his number stored in numbers[5].

The following example shows a definition of a 5 by 4 array of integers.

```
int[][] x = new int[5][4];
//i indicates row (row by row)
for(int i = 0; i < 5; i++){
	//j indicates column (column by column)
	for(int j = 0; j < 4; j++){
			x[i][j] = i * j;
		}
	}

//Print array
for(int i = 0; i < x.length; i++){
	for(int j = 0; j < x[i].length; j++){
		System.out.print(x[i][j] + " ");
	}
	System.out.println("");
}
```

You can see from the previous example that we deal with each element in the array as array itself, for example, we call x[i].length, this means that x[i] is an array of integers that may have its own length, because it is not necessary that all rows have the same number of columns. The following example shows that:

```
int[][] a = {
{0}, //a[0]
{0, 1}, //a[1]
{0, 1, 2}, //a[2]
{0, 1, 2, 3} //a[3]
};
for(int i = 0; i < a.length; i++){
	for(int j = 0; j < a[i].length; j++){
		System.out.print(a[i][j] + " ");
	}
	System.out.println("");
}
```

The output of the above code is the following:

```
0
0 1
0 1 2
0 1 2 3
```

As you can see, we filled the array above with sub-arrays. Two dimensional arrays are common in representing matrices, for example, we can write a program that adds two matrices (remember

that matrices must have same number of rows and same number of columns when it comes to addition).

6.2. String as Arrays

We can deal with strings as array of characters, this could be accomplished through two methods in String class called length() and charAt(), the following example prints a string putting each character in separate line.

```
String s = "FAST NU Peshawar";
for(int i = 0; i < s.length(); i++){
        System.out.println(s.charAt(i));
}
```

Beware of the difference between strings and arrays when accessing length; length in arrays is a variable so we call it without brackets, but in string, length() is a method so we call it with brackets. charAt() method simply returns the character contained in the provided zero-based index within the string, for example, calling charAt(3) in the previous example returns 'T'.

Home Exercises (to be shown in the next lab)

1. Write a program that asks the user to enter two numerical values (integers) and then select an operation (addition, subtraction, multiplication and division) then prints the result based on operation selected. The code below shows examples of the output (text shown in boldface is supposed to be user input).

   Enter first number: 4
   Enter second number: 2
   1. Addition (+).
   2. Subtraction (-).
   3. Multiplication (*).
   4. Division (/).
   Enter operation number: 3
   The result is 8

2. Modify calculation program in session 4 exercise 3 by adding the following question at the end of the program:

   Do you want to make another calculation?

1. Yes
2. No
Enter your option:

If user selects yes (by entering 1), program will ask him again to enter new two numbers and select operation, if user selects no (by entering 2), program exits. Use appropriate loop to accomplish this.

3. Write a program that reads 16 integers from the user, stores them in 5 by 4 matrix, the last row of the matrix should contain the sums of the columns (see figure) calculated by your program. Then the whole matrix is printed.

User input	4	7	1	8
	3	8	9	5
	1	3	4	5
	8	1	5	6
Sum	16	19	19	24

4. Write a program that asks the user to enter certain number, after that asks him to enter another 20 numbers, after entering them all, it prints out the number of occurrences of the first number. See the below example (text shown in boldface is supposed to be user input).

Enter number to search for: **2**
Enter a number: (20 times)
2 3 56 7 9 2 4 5 5 6 2 21 33 19 32 88 0 32 100 20
The number (2) occurred 3 times in your input.