



Object Oriented Programming

(CSE 2202)

Chapter One and Two

Part-II

Control Statements, Arrays and Strings

Objectives

At the end of this session students will be able to learn:

- Overview of Java statements
- If-else statements
- Switch statement
- For loop
- while and do-while loops

Control Flow Statements

- Control flow statements govern the flow of control in a program during execution, that is, the order in which statements are executed in a running program.
- There are three main categories of control flow statements :
- Selection statements: **if, if-else and switch.**
- Iteration statements: **while, do-while and for.**
- Transfer statements: **break, continue, return, try-catch and finally.**

- **If Statement:-** This is a control statement to execute a single statement or a block of code, when the given condition is true and if it is false then it skips **if** block and rest code of program will execute.

Syntax:

```
if(conditional_expression){  
    <statements>;  
    ...;  
    ...;  
}
```

```
import java.util.*;

public class Demoif
{
    public static void main(String k[ ]){
        Scanner t=new Scanner (System.in);
        System.out.println("pls enetr the integer
value");

        int x = t.nextInt();
        char grade ;
        if(x>= 85){
            grade = 'A';
        }
        if(x >= 70 && x < 85){
            grade = 'B';
        }
        System.out.println("x = " + x + " and grade =
"+ grade);
    }
}
```

- **If-else Statement:-**The "if-else" statement is an extension of if statement that provides another option when 'if' statement evaluates to "false" i.e. else block is executed if "if" statement is false.

Syntax:

```
if(conditional_expression)
{
    <statements>;
    ....;
}
else
{
    <statements>;
    ....;
}
```

■ **Example:**

```
int n = 11;
if(n%2 == 0)
{
    System.out.println("This is even number");
}
else
{
    System.out.println("This is not even number");
}
```

- **Switch Statement:-** This is an easier implementation to the if-else statements.
- The keyword "**switch**" is followed by an *expression* that should evaluate to byte, short, char or int primitive data types only.
- In a **switch** block there can be one or more labeled cases.
- The expression that creates labels for the case must be unique.

- **Syntax:**

```
switch(control_expression)    {  
    case expression 1:  
        <statement>;  
        break;  
    case expression 2:  
        <statement>;  
        break;  
    ...  
    default:  
        <statement>;  
        break;
```

```
int day = 2;
switch (day) {
case 1:
    System.out.println("Mon");
    break;
case 2:
    System.out.println("Tues");
    break;
case 3:
    System.out.println("Wed");
    break;
case 4:
    System.out.println("Thurs");
    break;
case 5:
    System.out.println("Friday");
    break;
case 6:
    System.out.println("Saturday");
    break;
case 7:
    System.out.println("Sunday");
    break;
default:
    System.out.println("Invalid day");
    break;
}
```

Iteration statements: while, do-while and for



- The process of repeatedly executing a block of statements is known as looping.
- In looping, sequence of statements are executed until some certain condition for the termination of loop is satisfied.
- Java provides three types of loops
 - I. **for loop**
 - II. **while loop**
 - III. **do..while loop**

- **while loop-** is a control structure that allows you to repeat a task a certain number of times.
The syntax for a while loop is:

```
initialization;  
while (test condition)  
{  
    Body of loop;  
}
```

```
int x = 1;  
while(x <= 5)  
{  
    System.out.println(x);  
    x++;  
}
```

- **do/while-** loop is similar to a while loop, except that a do/while loop is guaranteed to execute at least one time.

- The syntax of a do/while loop is:

```
do
{
Body of loop;
}while( text condition);
```

```
int y = 1;
do
{
System.out.println(y);
y += 1;
}while(y <= 8);
```

- **for loop-** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

- The syntax of a for loop is:

***for(initialization; test condition;
increment/decrement)***

***{
Body of loop;
}***

eg.-

```
for(int i = 1; i <= 5; i++)  
    {  
        System.out.println(i);  
    }
```

Jumping Statements

- **break** :-the break keyword can be used in any of the loop control structures to cause the loop to terminate immediately.
- When a break occurs, the flow of control will jump to the next statement past the loop.

```
public class Break1 {  
    public static void main(String H[]) {  
        int k = 1;  
        while(k <= 10)  
        {  
            System.out.println(k);  
            if(k == 6)  
            {  
                break;  
            }  
            k++;  
        }  
        System.out.println("The final value of k is " + k);  
    }  
}
```

- **Continue-** keyword can be used in any of the loop control structures.
- It causes the loop to immediately jump to the next iteration of the loop means **causes the early iteration of a loop**.
 - In a for loop, the continue keyword causes flow of control to immediately jump to the update(increment/decrement) statement.
 - In a while loop or do/while loop, flow of control immediately jumps to the boolean expression.

```
public class Continue1{
    public static void main(String [] args){
        System.out.println("The for loop");
        for(int i = 1; i <= 10; i++) {
            if(i % 2 == 0) {
                continue;
            }
            System.out.println(i);
        }
        System.out.println("The while loop");
        int j = 20;
        do {
            if(j % 3 != 0) {
                continue;
            }
            System.out.println(j);
        }while(j-- > 0);
    } }
```

Arrays

- An array is **a group of contiguous or related data** items that share a common name.
- is a container object that holds a fixed number of values of a single type.
- Unlike C++, in Java arrays are created dynamically.
- ***An array can hold only one type of data!***

Example:

int[] can hold only integers

char[] can hold only characters

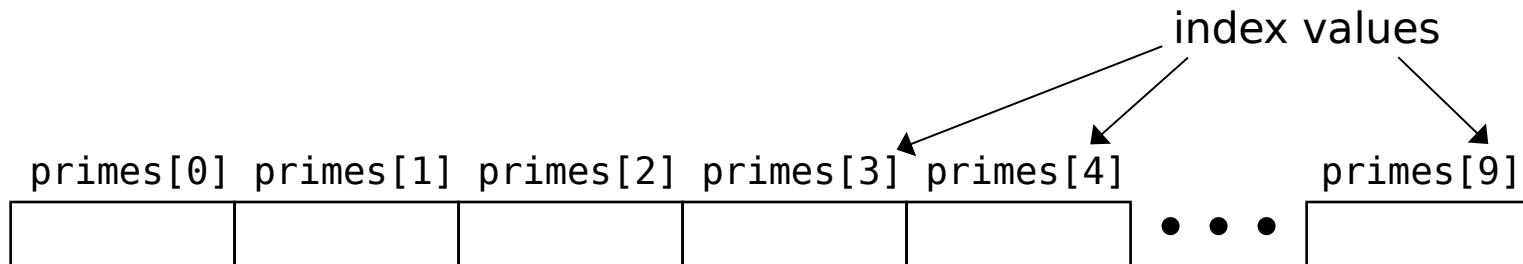
- A particular values in an array is indicated by writing a number called *index* number or *subscript* in brackets after the array name.

Example:

slarray[10] represents salary of the 10th employee.

Contd...

- The length of an array is established when the array is created. After creation, its length is fixed.
- Each item in an array is called an *element*, and each element is accessed by its numerical *index*.
- Array indexing *starts from 0 and ends at n-1*, where n is the size of the array.



One-Dimensional Arrays

- A list of items can be given one variable name **using only one subscript** and such a variable is called a **single-subscripted variable** or a **one-dimensional array**.

Creating an Array

- Like any other variables, arrays must be declared and created in the computer memory before they are used.
- Array creation involves three steps:
 1. Declare an array Variable
 2. Create Memory Locations
 3. Put values into the memory locations.

Declaration of Arrays

- Arrays in java can be declared in two ways:
 - i. ***type arrayname [];***
 - ii. ***type[]arrayname;***

Example:

```
int number[];  
float slaray[];  
float[] marks;
```

- when creating an array, each element of the array receives a default value zero (for numeric types) ,false for boolean and null for references (any non primitive types).

Creation of Arrays

- After declaring an array, we need to create it in the memory.
- Because an array is an object, you create it by using the **new** keyword as follows:

arrayname =new type[size];

Example:

```
int []number;  
float marks[];  
number=new int[5];  
float marks=new float[7];
```

- It is also possible to combine the above two steps , declaration and creation, into one statement as follows:

type arrayname =new type[size];

Initialization of Arrays

- Each element of an array needs to be assigned a value; this process is known as **initialization**.
- Initialization of an array is done using the array subscripts as follows:
 - ***arrayname [subscript] = Value;***

Example:

```
number [0]=23;  
number[2]=40;
```

- Unlike C, java protects arrays from **overruns** and **underruns**.
- Trying to access an array beyond its boundaries will generate an error.

Contd...

- Java generates an *ArrayIndexOutOfBoundsException* when there is under run or over run.
- The Java interpreter checks array indices to ensure that they are valid during execution.
- Arrays can also be initialized automatically in the same way as the ordinary variables when they are declared, as shown below:

type arrayname [] = {list of Values};

Example:

```
int number[] = {35,40,23,67,49};
```

Contd...

- It is also possible to assign an array object to another array object.

Example:

```
int array1[] = {35, 40, 23, 67, 49};  
int array2[];  
array2 = array1;
```

Array Length

- In Java, all arrays store the allocated size in a variable named **length**.
- We can access the length of the array **array1** using **array1.length**. **Example:**

```
int size = array1.length;
```

```
//sorting of a list of Numbers
class Sorting
{
    public static void
        main(String [ ]args)
    {
        int num[ ]= {55, 40, 80,
12, 65, 77};
        int size = num.length;
        System.out.print("Given
List: ");
        for (int i=0; i<size; i++)
        {
            System.out.print(" " +
num[ i ] );
        }
        System.out.print("\n");
        //Sorting Begins
        for (int i=0; i<size; i++)
        {
            for (int j=i+1; j<size;
j++)
            {
```

```
                if (num[i] < num [j])
                {
                    //Interchange Values
                    int temp = num[i];
                    num [i] = num [j];
                    num [j] = temp;
                }
            }
        }
        System.out.print("SORTED
LIST" );
        for (int i=0; i<size; i++)
        {
            System.out.print(" " " + num
[i]);
        }
        System.out.println(" ");
    }
}
```

Two-Dimensional Arrays

- A 2-dimensional array can be thought of as a grid (or matrix) of values.
- Each element of the 2-D array is accessed by providing two indexes: *a row index and a column index*
- A 2-D array is actually just an array of arrays.
- A multidimensional array *with the same number of columns* in every row can be created with an array creation expression:

Example:

```
int myarray[][]= new int [3][4];
```

- Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. For example,

```
int myarray[2][3]= {0,0,0,1,1,1};
```

or

```
int myarray[][]= {{0,0,0},{1,1,1}};
```

- We can refer to a value stored in a two-dimensional array by using indexes for both the column and row of the corresponding element. For

Contd...

```
//Application of two-dimensional Array
class MulTable{
    final static int ROWS=12;
    final static int COLUMNS=12;
    public static void main(String [ ]args) {
        int pro [ ] [ ]= new int [ROWS][COLUMNS];
        int i=0,j=0;
        System.out.print("MULTIPLICATION TABLE");
        System.out.println("  ");
        for (i=1; i<=ROWS; i++)
        {
            for (j=1; j<=COLUMNS; j++)
            {
                pro [i][j]= i * j;
                System.out.print("  "+ pro [i][j]);
            }
            System.out.println("  ");
        }
    }
}
```

Strings

- Strings represent a sequence of characters.
- The easiest way to represent a sequence of characters in Java is by using a character:

```
char ch[ ]= new char[4];  
ch[0] = 'D';  
ch[1] = 'a';  
ch[2] = 't';  
ch[3] = 'a';
```

- This is equivalent to ***String ch="Hello";***
- Character arrays have the advantage of being able to query their length.
- But they are not good enough to support the range of operations we may like to perform on strings.

Contd...

- In Java, strings are declared and created as follows:

```
String stringName;
```

```
stringName= new String("String");
```

Example:

```
String firstName;
```

```
firstName = new String("Jhon");
```

- The above two statements can be combined as follows:

```
String firstName= new String("Jhon");
```

- The *length()* method returns the length of a

String Arrays

- It is possible to create and use arrays that contain strings as follows:

String arrayname[] = new String[size];

Example:

String item[] = new String[3];

item[0] = "Orange";

item[1] = "Banana";

item[2] = "Apple";

- It is also possible to assign a string array object to another string array object.

String Methods

1. The **length();** method returns the length of the string.

Eg: `System.out.println("Hello".length());` // prints 5

- The + operator is used to concatenate two or more strings.

Eg: `String myname = "Harry"`

`String str = "My name is" + myname + ".";`

2. The **charAt();** method returns the character at the specified index.

Syntax : `public char charAt(int index)`

Ex: `char ch;`

`ch = "abc".charAt(1);` // ch = "b"

3. The **equals()**; method returns 'true' if two strings are equal.

Syntax : *public boolean equals(Object anObject)*

Ex: `String str1="Hello",str2="hello";`

`(str1.equals(str2))? System.out.println("Equal"); :
System.out.println("Not Equal"); // prints Not Equal`

4. The **equalsIgnoreCase()**; method returns 'true' if two strings are equal, ignoring case consideration.

Syntax : *public boolean equalsIgnoreCase(String str)*

Ex: `String str1="Hello",str2="hello";`

`if (str1.equalsIgnoreCase(str2))
System.out.println("Equal");`

5. The **toLowerCase();** method converts all of the characters in a String to lower case.

Syntax : *public String toLowerCase();*

Ex: String str1="HELLO THERE";

System.out.println(str1.toLowerCase()); // prints **hello there**

6. The **toUpperCase();** method converts all of the characters in a String to upper case.

Syntax : : *public String toUpperCase();*

Ex: System.out.println("wel-come".toUpperCase()); // prints **WEL-COME**

7. The **trim();** method removes white spaces at the beginning and end of a string.

Syntax : *public String trim();*

Ex: System.out.println(" wel-come ").trim()); //prints

8. The **replace();** method replaces all appearances of a given character with another character.

Syntax : *public String replace('ch1', 'ch2');*

Ex: `String str1="Hello";`

`System.out.println(str1.replace('l', 'm'));` // prints **Hemmo**

9. **compareTo();** method Compares two strings lexicographically.

- The result is a negative integer if the first String is less than the second string.
- It returns a positive integer if the first String is greater than the second string. Otherwise the result is zero.

Syntax : *public int compareTo(String anotherString);*

public int compareToIgnoreCase(String str);

Ex: `("hello".compareTo("Hello")==0)` ?

`System.out.println("Equal");`

10. The **concat();** method concatenates the specified string to the end of this string.

Syntax : *public String concat(String str)*

Ex: `System.out.println("to".concat("get").concat("her")); //`
returns together

11. **compareTo();** method Compares two strings lexicographically.

- The result is a negative integer if the first String is less than the second string.
- It returns a positive integer if the first String is greater than the second string. Otherwise the result is zero.

Syntax : *public int compareTo(String anotherString);*

public int compareToIgnoreCase(String str);

Ex: `("hello".compareTo("Hello")==0) ?`
`System.out.println("Equal");`

12. The **substring();** method creates a substring starting from the specified index (n^{th} character) until the end of the string or until the specified end index.

Syntax : *public String **substring**(int beginIndex);*
*public String **substring**(int beginIndex, int endIndex);*

Ex: "smiles".substring(2); //returns "ile"
"smiles".substring(1, 5); //returns "mile"

13. The **startsWith();** Tests if this string starts with the specified prefix.

Syntax: *public boolean **startsWith**(String prefix);*

Ex: "Figure".startsWith("Fig"); // returns **true**

14. The **indexOf();** method returns the position of the first occurrence of a character in a string either starting from the beginning of the string or starting

- *public int **indexOf**(int ch);* Returns the index of the first occurrence of the character within this string starting from the first position.
- *public int **indexOf**(String str);* - Returns the index of the first occurrence of the specified substring within this string.
- *public int **indexOf**(char ch, int n);* - Returns the index of the first occurrence of the character within this string starting from the nth position.

Ex: String str = "How was your day today?";

```
    str.indexOf('t'); // prints 17
    str.indexOf('y', 17); // prints 21
    str.indexOf("was"); // Prints 4
    str.indexOf("day", 10)); //Prints 13
```

15. The **lastIndexOf();** method Searches for the last occurrence of a character or substring.

▪ The methods are similar to indexOf() method.

16. **valueOf();** creates a string object if the parameter or converts the parameter value to string representation if the parameter is a variable.

Syntax: *public String valueOf(variable);*
public String valueOf(variable);

Ex: `char x[]={'H','e',' ','l',' ','o'};`

`System.out.println(String.valueOf(x));`//prints

Hello

`System.out.println(String.valueOf(48.958));` //

prints 48.958

17. **endsWith();** Tests if this string ends with the specified suffix.

Syntax: *public boolean endsWith(String*

Thank You ...

