



OOP's Using JAVA

Subject Code :BT21ITE33

Module 2

Topics Covered

- Data Types
- Operators
- Control Statements
- Variables & Arrays
- String Handling
 - The String Constructors
 - String Length
 - Special String Operations
 - Character Extraction
 - String Comparison
 - Modifying a String
 - String Buffer

Data Types

- Data types specify the different sizes and values that can be stored in the variable. There are **two** types of data types in Java
 - 1. Primitive data types:** The primitive data types include Boolean, char, byte, short, int, long, float and double.
 - 2. Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays, Strings.
- In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Operators in Java

- **Operator** in Java is a symbol that is used to perform operations.
- Java divides the operators into the following groups:
 - ✓ Arithmetic operators
 - ✓ Assignment operators
 - ✓ Comparison operators
 - ✓ Logical operators
 - ✓ Conditional operators

Arithmetic Operator

- Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	$x \% y$

Comparison operators

- comparison operators are used to compare two values

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Assignment operators

- Assignment operators are used to assign values to variables
- For example, we use the assignment operator (=) to assign the value to variable x:
 - in `x=10`

Operator	Example	Same As
=	<code>x = 5</code>	<code>x = 5</code>
+=	<code>x += 3</code>	<code>x = x + 3</code>
-=	<code>x -= 3</code>	<code>x = x - 3</code>
*=	<code>x *= 3</code>	<code>x = x * 3</code>
/=	<code>x /= 3</code>	<code>x = x / 3</code>
%=	<code>x %= 3</code>	<code>x = x % 3</code>

Logical operators

- Logical operators are used to determine the logic between variables or values

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	<code>x < 5 && x < 10</code>
	Logical or	Returns true if one of the statements is true	<code>x < 5 x < 4</code>

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        System.out.println(x > 3 && x < 10);
        // returns true because 5 is greater than 3 AND 5 is less than 10
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        System.out.println(x > 3 || x < 4);
        // returns true because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)
    }
}
```

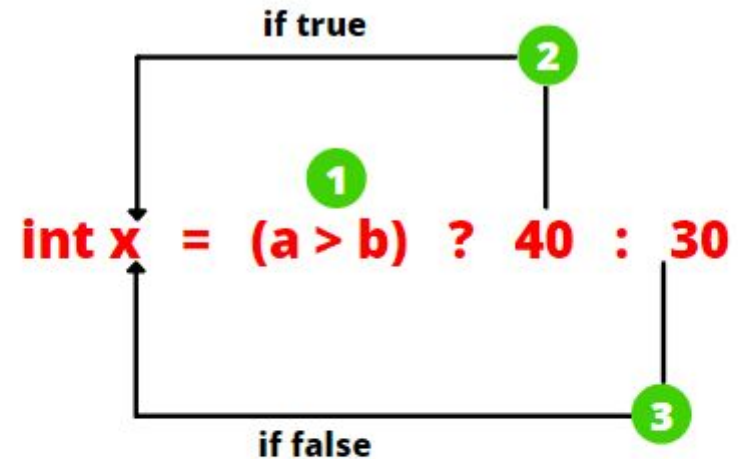
Conditional operators

- Consider this example

```
Int a=30;
```

```
Int b=40;
```

```
In x=(a>b)?a:b;
```



The ternary or conditional operator lets you write one line conditional statements. In this example, 1 is the condition. If it is true, the code immediately following the ? is returned (2); if the condition is false, the code following the : is returned (3).

Fig: Example of conditional operator in Java

Control Statements

- There are three types of control flow statements.
 1. Conditional statements: based on particular condition the selected block will be executed
 - if
 - if –else
 - else-if
 - Switch
 2. Loop statements: The same code will be Repeated multiple times
 - for
 - while
 - do-while
 3. Transfer statements: The control is transfer from one location to another location is called transfer statements
 - break
 - continue

Control Statement

Conditional statements

If Statement

- To take only one option
- Syntax of if statement is given below.

```
if(condition) {  
  True body; //executes when condition is true  
}
```

Example of "if" statement

Program 1

```
1. class Student {  
2.     public static void main(String[] args) {  
3.         int x = 20;  
4.         int y = 10;  
5.         if(x>y) { //if Condition  
6.             System.out.println("x is greater than y");  
7.         }  
8.     }  
9. }
```

Control Statement
Decision-Making statements

“if-else” statement

- Two option available.
- The [if-else statement](#) is an extension to the if-statement, which uses another block of code, i.e., else block.
- The else block is executed if the condition of the if-block is evaluated as false.

- **Syntax:**

```
if(condition) {
```

```
True body; //executes when condition is true
```

```
}
```

```
else{
```

```
False body; //executes when condition is false
```

```
}
```

Example of "if else" statement

Program 2

```
1.  class Student {  
2.  public static void main(String[] args) {  
3.  int x = 20;  
4.  if(x > 18) {  
5.  System.out.println("Eligible for marriage....");  
6.  } else {  
7.  System.out.println("Not eligible for marriage try after some time....");  
8.  }  
9.  }  
10. }
```

“if-else-if” statement

- The if-else-if statement contains the if-statement followed by multiple else-if statements.
- In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true.

- **Syntax**

```
if(condition 1) {  
statement 1; //executes when condition 1 is true  
}  
else if(condition 2) {  
statement 2; //executes when condition 2 is true  
}  
else {  
statement 2; //executes when all the conditions are false  
}
```


Example of "if else if" statement

Program 3

```
class Positive {  
public static void main(String[] args) {  
    int number=-13;  
    if(number>0){  
        System.out.println("POSITIVE");  
    }else if(number<0){  
        System.out.println("NEGATIVE");  
    }else{  
        System.out.println("ZERO");  
    }  
}  
}
```

Example of "if else if" statement

Program 3

```
class Sleep {  
public static void main(String[] args) {  
    int a=10;  
    if(a==10){  
        System.out.println("Aslam don't sleep");  
    }  
    else if(a==20){  
        System.out.println(" Amar don't sleep ");  
    }  
    else if(a==30){  
        System.out.println("Umar don't sleep");  
    }  
    else{  
        System.out.println(" Bharath don't sleep ");  
    }  
}  
}
```

Control Statement
Decision-Making statements

switch statement

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int.
- Points to be noted about switch statement:
 - The case variables can be int, short, byte, char. String type is also supported since version 7 of Java
 - Cases cannot be duplicate
 - Default statement is executed when any of the case doesn't match the value of expression. It is optional.

- The syntax to use the switch

```
switch(argument){  
case value1: Statements;  
  break; //optional  
case value2: Statements;  
  break; //optional  
.....  
default:  
  code to be executed if all cases are not matched;  
}
```

Example of "Switch" statement Program 5

```
int a=20;
//Switch expression
switch(a){
//Case statements
case 10: System.out.println("10");
break;
case 20: System.out.println("20");
break;
case 30: System.out.println("30");
break;
//Default case statement
default: System.out.println("Not in 10, 20 or 30");
}
}
}
```

EX:

```
class SwitchExample2 {
public static void main(String[] args) {
int a=20;
//switch expression with int value
switch(a){
//switch cases without break statements
case 10: System.out.println("10");
case 20: System.out.println("20");
```

Loop Statements

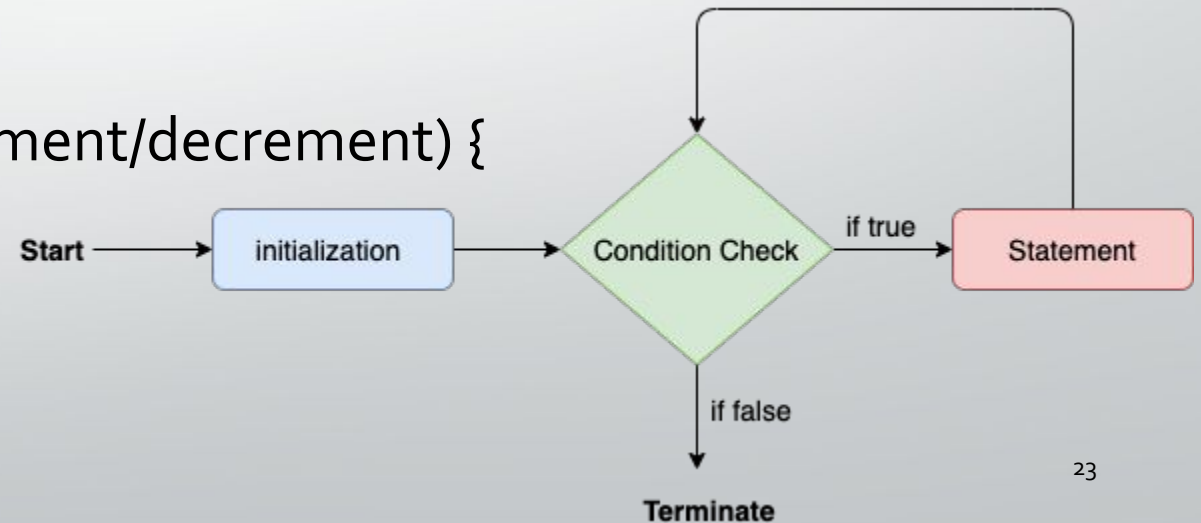
- Loop statements: The same code will be Repeated multiple times
 - for
 - while
 - do-while

Loop Statements

for

- In Java, for loop is similar to C and C++
- It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
- We use the for loop only when we exactly know the number of times, we want to execute the block of code.
- Syntax

```
for(initialization, condition, increment/decrement) {  
//block of statements  
}
```



Example of "for loop"

Program 5

```
1. class Test {  
2.   public static void main(String[] args) {  
3.     for(int i= 0; i<=10; i++) {  
4.       System.out.println("Good Morning...." );  
5.     }  
6.   }  
7. }
```


Loop Statements

While

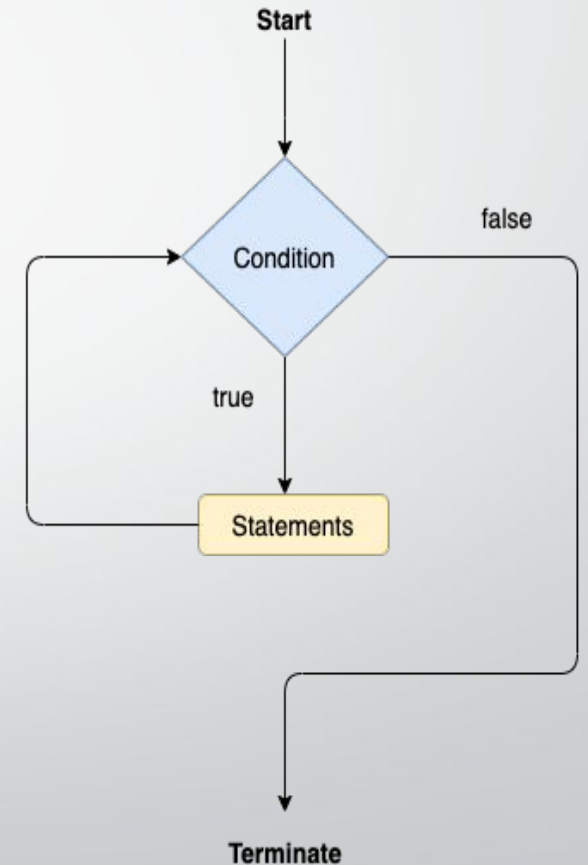
- The while loop is also used to iterate over the number of statements multiple times.
- For loop is recommended when we have :(start;end_cond;incre_decre)
- while loop is recommended when we have the only condition check.
- **syntax :**

```
while(condition){
```

```
body
```

```
}
```

The body will be executed until the condition is fail



Example of "while" Program 6

```
1. class Test {  
2. public static void main(String[] args) {  
3. // for loop to printing the data 10 times  
4. for(int j=0;j<10;j++)  
5. {  
6. System.out.println("Good Morning..");  
7. }  
8. // while loop to printing the data 10 times  
9. int i=0;  
10. while(i<10) {  
11. System.out.println("Good Evening..");  
12. i++;  
13. }  
14. }  
15. }
```

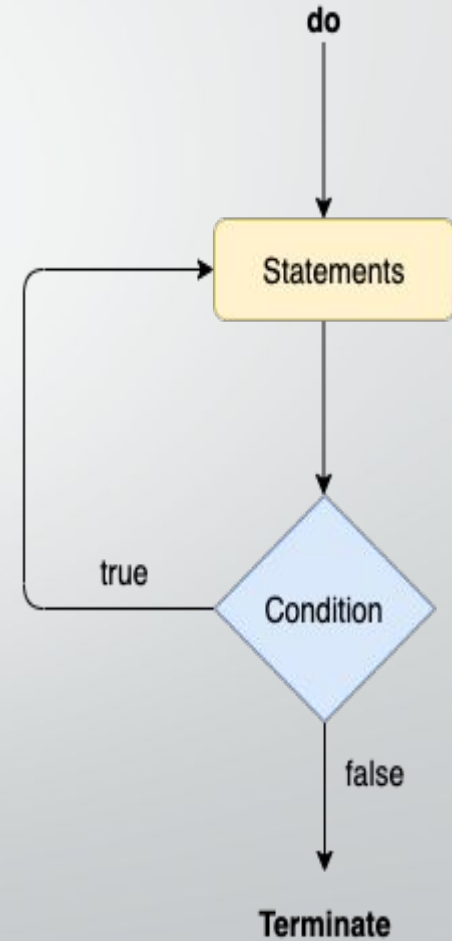
- //To print the data 10-time it is recommended to use for loop.

Loop Statements

do While

- If you want execute the body first without condition check then use do while.
- The syntax

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



```
do {  
    System.out.println("Good Morning");  
}while(20<10);  
}  
}
```

do While

EX:

```
class Test {  
    public static void main(String[] args) {  
        Int i=0;  
        do {  
            System.out.println("Good Morning");  
            i++;  
        }while(i<10);  
    }  
}
```

Jump Statement

- Jump statements are used to transfer the control of the program to the specific statements.
- There are two types of jump statements in Java, i.e., break and continue.
 - break statement- As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement
 - continue statement –The continue statement break one iteration(in the loop),if a specified condition occurs and continues next iteration in the loop.

Jump Statement

```
class Test
{
    public static void main(String[] args)
    {
        for (int i=0;i<=10 ;i++)
        {
            if (i==5)
            {
                break;
            }
            System.out.println(i);
        }

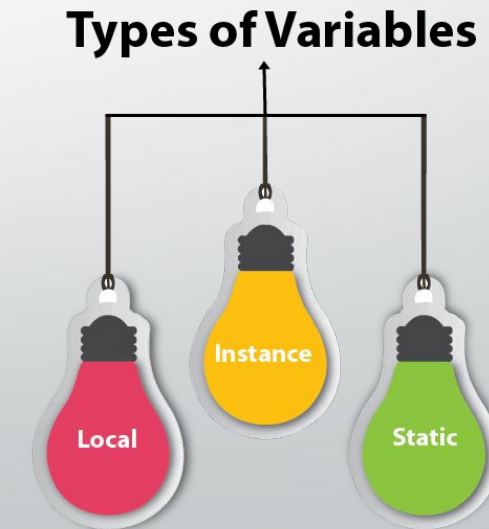
        System.out.println("*****");

        for (int i=0;i<=10 ;i++)
        {
            if (i==5)
            {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

```
E:\>java Test
0
1
2
3
4
*****
0
1
2
3
4
6
7
8
9
10
E:\>
```

Variables in JAVA

- A variables are used to store the values by using that value we are achieving the project requirements.
- A variable is assigned with a data type.
- There are three types of variables in java:
 - local
 - instance
 - static



Variables in JAVA

1) Local Variable

- A variable declared inside the method is called local variable.
- You can use this variable only within that method
- other methods in the class aren't even aware that the variable exists.

```
Void m1()
```

```
{
```

```
Int a=10; // Local variables
```

```
System.out.println(a); //Possible
```

```
}
```

```
Void m2()
```

```
{
```

```
System.out.println(a); //Not Possible
```

```
}
```


2) Instance Variable

- A variable declared inside the class but outside the method.
- Accessing done through object .
- Direct access.

EX:

2.

3. **class** variable

3.

4.

{

5.

int a=10;//instance variable

6.

public static void main(String args[])

7.

{

8.

variable obj=new variable(); /* object creation*/

9.

System.out.println(obj.a); -----//10

10.

}

11.

}

3) Static variable

- A variable that is declared as static is called a static variable.
- Memory allocation for static variables happens only once.
- Direct access.

2. **EX:**

3. **class** variable

4. {

5. **static int** a=10;//static variable

6. **public static void** main(String args[])

7. {

8. System.out.println(a); -----//10

9. }

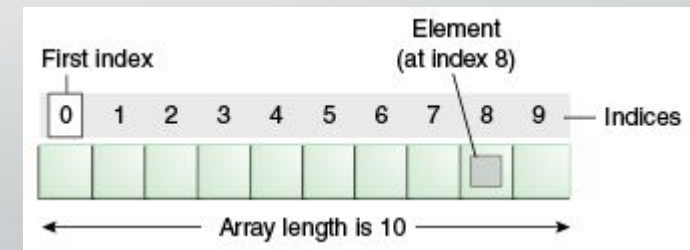
10. }

Example of "Variables" Program 9

```
1.  EX:
2.  class variable
3.  {
4.      int a=10;//instance variable
5.      static int b=10;//static variable
6.      public static void main(String args[])
7.      {
8.          int c=10;//Local variable
9.          System.out.println(b);
10.         System.out.println(c);
11.         variable obj=new variable(); /* object creation*/
12.         System.out.println(obj.a); -----//10
13.     }
    }
```

ARRAY in JAVA

- Arrays are used to store the group of elements and these elements are homogenous & fixed number
- `int[] a={10,20,30,40};`
- `double [] d={10.2,20.6,2.5};`
- Arrays are indexed based, index start from zero.



```
1.  EX:
2.  class Test
3.  {
4.      int[] a={10,20,30,40};
5.      public static void main(String args[])
6.      {
7.          System.out.println(a[0]);
8.          System.out.println(b[1]);
9.          System.out.println(c[2]);
10.         System.out.println(d[3]);

        for(int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
1.     }
2. }
```

ARRAY in JAVA

- **Advantages**

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

- **Disadvantages**

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

- There are **two** types of array.

- Single Dimensional Array
- Multidimensional Array

String in JAVA

- Generally, String is a sequence of characters.
- But in Java, string is an object that represents a sequence of characters.
- The `java.lang.String` class is used to create a string object.
- There are two ways to create String object:
 - By string literal (`String s="welcome";`)
 - By new keyword (`String s=new String("Welcome");`

Example of "String"

Program 10

```
1. public class StringExample{
2.     public static void main(String args[]){
3.         String s1="java";//creating string by Java string literal
4.         char ch[]={'s','t','r','i','n','g','s'};
5.         String s2=new String(ch);//converting char array to string
6.         String s3=new String("example");//creating Java string by new keyword
7.         System.out.println(s1);
8.         System.out.println(s2);
9.         System.out.println(s3);
10.    }
```


String Handling

- The **Java String class length()** method finds the length of a string. The length of the Java string is the same as the Unicode code units of the string.

Syntax

```
public int length()
```

Example of "String Length"
Program 11

```
1. public class LengthExample{  
2. public static void main(String args[]){  
3.   String s1="javatpoint";  
4.   String s2="python";  
5.   System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string  
6.   System.out.println("string length is: "+s2.length());//6 is the length of python string  
7.   }}
```

Special String Operations

- There are 4 types of special string Operations

- String Literals
- String Concatenation
- String Concatenation with Other Data Types
- String Conversion and *toString()*

- **String literal** is created by using double quotes.

- For Example:

```
String s="javaguides";
```

- **String Concatenation** can be done using + operator, which concatenates two strings, producing a String object.

- For Example:

```
String age = "9";  
String s = "He is " + age + " years old.";  
System.out.println(s);
```

- You can concatenate strings with **other types of data**.
- For example

```
int age = 9;  
String s = "He is " + age + " years old.";  
System.out.println(s);
```

String Conversion and *toString()* : *valueOf()* method

For Example

valueOf() (The *valueOf()* method converts data from its internal format into a human-readable form.)

```
public static String valueOf(boolean b)
```

```
public static String valueOf(char c)
```

```
public static String valueOf(char[] c)
```

```
public static String valueOf(int i)
```

```
public static String valueOf(long l)
```

Character Extraction

- String is treated as an object in Java so we can't directly access the characters that comprise a string.
- There are several ways by which characters can be extracted from String class object.
- **charAt()** -- method is used to extract a single character at an index.
 - `char charAt(int index)`
- **getChars()** -- used to extract more than one character.
 - `void getChars(int stringStart, int stringEnd, char arr[], int arrStart)`
- **getBytes()** -- extract characters from String object and then convert the characters in a byte array.
 - `byte [] getBytes()`
- **toCharArray()** -- alternative of getChars() method.
 - `char [] toCharArray()`

Example of "Character Extraction"

Program 11

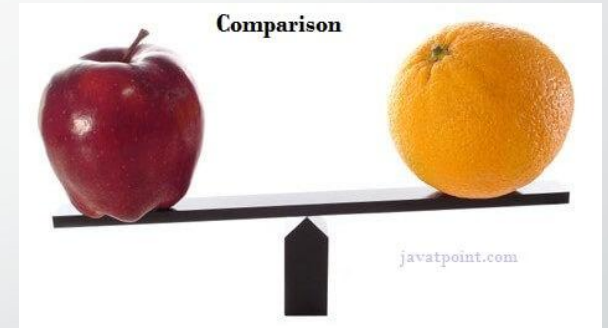
```
1. class temp
2. {
3.     public static void
4.     main(String...s)
5.     {
6.         String str="Hello";
7.         char ch=str.charAt(2);
8.         System.out.println(ch)
9.         ;
10.    }
11. }
```

```
1. class temp
2. {
3.     public static void main(String...s)
4.     {
5.         String str="Hello World";
6.         char ch[]=new char[4];
7.         str.getChars(1,5,ch,0);
8.         System.out.println(ch);
9.     }
10. }
```

```
class temp
{
    public static void main(String...s)
    {
        String str="Hello World";
        char ch[]=str.toCharArray();
        System.out.println(ch);
    }
}
```

String Comparison

- We can compare String in Java on the basis of **content** and **reference**
- It is used in
 - **authentication** (by equals() method),
 - **sorting** (by compareTo() method),
 - **reference matching** (by == operator).
- There are three ways to compare String in Java:
 - By Using equals() Method
 - By Using == Operator
 - By compareTo() Method



Example of "String Comparison"

Program 12

```
class Teststringcomparison1{
public static void main(String args[]){
String s1="Sachin";
String s2="Sachin";
String s3=new String("Sachin");
String s4="Saurav";
System.out.println(s1.equals(s2));//true
System.out.println(s1.equals(s3));//true
System.out.println(s1.equals(s4));//false
}
}
```

```
1.class Teststringcomparison3
{
2. public static void main(String
args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=new String("Sachin");
6. System.out.println(s1==s2);
//true (because both refer to
same instance)
7. System.out.println(s1==s3);
//false(because s3 refers to i
nstance created in nonpool)
8. }
9.}
```

```
1.class Teststringcomparison4{
2. public static void main(String args
[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3="Ratan";
6. System.out.println(s1.compareTo(
s2));//0
7. System.out.println(s1.compareTo(
s3));//1(because s1>s3)
8. System.out.println(s3.compareTo
(s1));//-1(because s3 < s1 )
9. }
10.}
```

Modifying a String

- Methods for modifying a String objects.
 - *substring()*
 - `substring(int beginIndex)`
 - `substring(int beginIndex, int endIndex)`
 - *concat()* -- We can concatenate two strings using `concat()`
 - *replace()*
 - `String replace(char original, char replacement)`
 - `String replace(CharSequence original, CharSequence replacement)`
 - *replaceAll()* – *Replaces each substring of this string that matches the given regular expression with the given replacement.*
 - `replaceAll(String regex, String replacement)`
 - *replaceFirst()* -- Replaces the first substring of this string that matches the given regular expression with the given replacement.
 - `replaceFirst(String regex, String replacement)`
 - *trim()* -- The `trim()` method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.
 - `String trim()`

StringBuffer

- A string buffer is like a String, but can be modified
 - append()
 - insert()
 - replace()
 - delete()
 - reverse()
 - capacity()

Example of "String Buffer" Program 13

- StringBuffer Class append() Method

```
1. class StringBufferExample1{  
2. public static void main(String args[]){  
3.   StringBuffer sb=new StringBuffer("Hello ");  
4.   sb.append("Java");  
5.   System.out.println(sb);  
6.   } }
```

StringBuffer insert() Method

```
1. class StringBufferExample2{  
2. public static void main(String args[]){  
3.   StringBuffer sb=new StringBuffer("Hello ");  
4.   sb.insert(1,"Java");//now original string is changed  
5.   System.out.println(sb);//prints HJavaello  
6. }  
7. }
```

StringBuffer replace() Method

```
1. class StringBufferExample3{  
2. public static void main(String args[]){  
3.   StringBuffer sb=new StringBuffer("Hello");  
4.   sb.replace(1,3,"Java");  
5.   System.out.println(sb);//prints HJavallo  
6. }  
7. }
```

StringBuffer delete() Method

```
1. class StringBufferExample4{  
2. public static void main(String args[]){  
3.   StringBuffer sb=new StringBuffer("Hello");  
4.   sb.delete(1,3);  
5.   System.out.println(sb);//prints Hlo  
6. }  
7. }
```

StringBuffer capacity() Method

```
1. class StringBufferExample5{
2.   public static void main(String args[]){
3.     StringBuffer sb=new StringBuffer();
4.     System.out.println(sb.capacity());//default 16
5.     sb.append("Hello");
6.     System.out.println(sb.capacity());//now 16
7.     sb.append("java is my favourite language");
8.     System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9.   }
10. }
```

StringBuffer reverse() Method

```
1. class StringBufferExample6{  
2. public static void main(String args[]){  
3.   StringBuffer sb=new StringBuffer("Hello");  
4.   sb.reverse();  
5.   System.out.println(sb);//prints olleH  
6. }  
7. }
```

Assignment-2

1. List out 4 primitive and 3 non primitive data types in java
2. What is "if" statement, Explain 4 types of decision making statements with syntax
3. Write a brief note on "Switch Statement", Explain 3 types of switch Statements with syntax
4. What are variables? Write note on 3 types of variables
5. What are the 3 ways to compare the string? Write java program for all the 3 ways to compare the string.