CORE JAVA

INTRODUCTION TO JAVA:

Java is an object oriented programming language. It was developed by “James gosling”,”Patrik Naughtan”,”Chris worth”,”Ed.Frank” and Mike Sheridan” at sun Microsystem in 1991. It took 18 months to develop the first working version. This version was initially called “OAK” but it was named “JAVA” in 1995.

Initially JAVA was not developed for internet application. It was platform independent language that could be used on variety of CPUs under different environment. Java derives much of from c and c++.

Java can be used to create two types of program i.e. application program and applets. An application program is a program that runs on your computer whereas an applet is very tiny program that will execute on the web browser. It is used to dynamically download the pictures, video clips and sound etc over the network.

Java provides more method. If you are using java compatible web browser then we can download data on the network without fearing of viral infection.

JAVA BUZZ WORD

1. **Simple**: java inherits most of features from c++ and also inherits its object oriented concept. Java eliminates most of the confusing concept from c++.
2. **Robust**: java provides facility to the user to develop the program under multiple environments means to say that you can execute java application on different architecture machine. Since java application is a strictly typed language so there is very less chance to create programming error.
3. **Object oriented**: java is completely object oriented programming language. In the oops concept it provides facility to create class, object, inheritance, polymorphism, package, interface etc.
4. **Architectural neutral**: java program provides the facility to compile and run your program any where anytime because java has its own environment to execute the program.
5. **Dynamic**: in the dynamic concept java create an object at run time of program.

Notes: **java supports both interpreter and compiler.**

Compiler: compiler is a system software .it is used to check system error of the program. If any syntax error founds, it reports to the user about the error. It checks all the program at a time to find the syntax error. Compiler displays all the errors of program at a time. “Just in time” is the compiler name of java. It is called “JIT” compiler.

Interpreter: it is also system software which checks the program line by line. It is basically used to check runtime error.

JVM (java virtual machine) in an interpreter of your byte code and it converts the byte code into machine understandable form.
Byte code: byte code is the set of instructions which will be understood by only java. Byte code is not machine code.

Points:
When you compile the program java program, java follows the following steps as per described below:

1. Using the JIT compiler the class converts into java understandable form ie byte code.
2. After successfully compilation of the program, the byte code should be convert into machine understandable form.
3. We use java virtual machine to covert byte code into machine understandable form.
4. **Javac** command uses file name which has an extension **.java** to convert into byte code.
5. **Java** command uses class file to convert byte code into executable form to run the program.
6. javac command uses JIT compiler and java command uses JVM (java virtual machine) interpreter.
7. You have not to use extension of class file with java command.

Tokens: - it defines about the operator, variable, constants, identifier, keyword, literals and comment.

Identifiers: - the java compiler wants to know the name of items in the program. As the name suggests identifier identify the name of variables, methods and elements of classes. Identifiers are either reserved word or variable name, constant name, method and classes.

Reserved words are name which is provided by the Java .If the programmer uses the reserve word for own purpose then java will generate the error because reserve words have special task that is defined by the java. Identifier can be begins with alphabet character, underscore (_) or dollar sign ($);

Keywords: - keywords are reserve word which is given by the java and it has special task which will be performed by the java only means to say that you can not use keywords in the program for other purpose.It is a special tokens and they are always represent in lowercase.

Keywords are used in flow controls, identifiers, class and expression.

Data declaration keywords

Boolean,byte,char,float,int, long,short

Loop key words

Break,continue,do,for,while.

Conditional keyword

Case ,if,else,switch.

Exception keyword

Catch,finally,through,try.

Structure keyword

Abstract class,default extends implements,instance of,interface.

Modify and access keywords.

Final,nativem new,private,protected,public,static,synchronized,threadsafe,void.

Miscellaneous keyword

False,import,null,package,return,super,true.

Literals

Data is represented by literals in Java. Similar to literals in other programming languages, Java literals are based on character and number representations. The types of literals are integer, floating-point, boolean, character, and string.
Every variable consists of a literal and a data type. The difference between the two is that literals are entered explicitly into the code. Data types are information about the literals, such as how much room will be reserved in memory for that variable, as well as the possible value ranges for the variable.

Integer Literals

Integers are whole numbers, such as 1 and 5280. Integer literals can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
Decimals can be positive, zero, or negative. Decimal literals cannot start with 0, as in 01234. After the beginning number, a decimal literal can consist of the numbers 0-9. Numbers beginning with 0 are reserved for octal and hexadecimal literals. Therefore, when using decimals, do not right-justify with leading zeros. The upper limit of a positive decimal integer is 231-1, or 2,147,483,647. The lower limit is -2,147,483,648 or -231.
Octal literals start with 0 and can be followed by any number 0-7. They can be positive, zero, or negative. The maximum value of an octal literal is 017777777777, which is equivalent to 231-1.
Hexadecimal integer literals start with 0x or 0X, followed by one or more hexadecimal digits. Letters A-F used in a hexadecimal integer can be uppercase or lowercase. Hexadecimal integers can be positive, zero, or negative. The upper limit of a positive hexadecimal literal is 0x7fffffff, which is, once again, equivalent to 231-1. The values available are 1-9, A-F, and a-f.

A compile-time error will occur if any of these values is exceeded.

Floating-Point Literals

A floating-point literal represents a number that has a decimal point in it, such as 3.7. Java standards specify that floating-point numbers must follow the industry standard specification as written in IEEE-754.
Single-precision floating-point numbers consist of a 32-bit space and are designated by uppercase or lowercase f. Double-precision numbers are allotted a 64-bit space and are designated by uppercase or lowercase d. Double-precision floating-point numbers are the default. Therefore, 3.7 is a double-precision floating-point number, and 3.7f is a single-precision floating-point number.
How do you know whether to use a single- or double-precision floating-point number? It depends on the size of the number. If there is any possibility that a number could grow out of range or need to be of greater precision than single, declare it a double.
Compile-time errors will occur if a nonzero floating-point literal is too large or small.
The largest magnitude single-precision floating-point literal is ±3.40282347e+38f, and the smallest is ±1.40239846e-45f. The largest double-precision floating-point number is 1.79769313486231570e+308, and 4.94065645841246544e-324 is the smallest floating-point number.

Note
Floating-point literals also can be expressed using exponents or scientific notation, as shown in the preceding paragraph. Uppercase or lowercase e is used to denote the exponent portion of a floating-point number. An example of this is 2.1e3f; this is a single-precision floating-point number representing 2100.

Boolean Literals

A boolean literal is either of the words true or false. Unlike other programming languages, no numeric value such as 0 or 1 is assigned. Therefore, the value of a boolean literal is, literally, true or false. Booleans are used extensively in program control flow logic.
Character Literals

Programmers often use a single character as a value. In Java, this is represented by character literals. The value of a character literal is enclosed by single quotes. An example is 'y'.

Assigning a value to a character literal gets more interesting if the value must be a single quote, a backslash, or other nonprintable characters. A backslash (\) is used to designate certain nonprintable characters or characters that are properly part of the command. For example, assign the value '\'' to characterize the single quote. Table 6.2 shows some examples of assigning values to character literals.

Table 6.2. Specifying character literals.

Description or Escape Sequence	Sequence	Output
Any character	'y'	Y
Backspace (BS)	'\b'	Backspace
Horizontal tab (HT)	'\t'	Tab
Linefeed (LF)	'\n'	Linefeed
Formfeed (FF)	'\f'	Form feed
Carriage return (CR)	'\r'	Carriage return
Double quote	'\"'	"
Single quote	'\''	'
Backslash	'\\'	\
Octal bit pattern	'\ddd'	Octal value of ddd
Hex bit pattern	'\xdd'	Hex value of dd
Unicode character	'\udddd'	Actual Unicode character of dddd

Compile-time errors occur if anything other than a single quote follows the value designator or the character after '\' is anything but b, t, n, f, r, ", ', \, 0, 1, 2, 4, 5, 6, or 7.

String Literals

String literals are a sequence of characters enclosed in double quotes, such as "This is a string literal". This could also be "Hello World" or even "" for a null character string. The javac compiler does not strip out whitespace from within string literals.

String literals can be concatenated. For example, if one string contained "This is the beginning" and another string contained "of a beautiful relationship", they could be concatenated together. The representation would be "This is the beginning" + "of a beautiful relationship". (It is not necessary to have spaces on either side of the plus sign.)

A string literal cannot span more than one line; those that do so can be broken up at declaration time and then concatenated at use time. This also makes programs more readable and strings less likely to be mistaken for comments.

As with the character literal, the backslash is used to denote symbols that otherwise would not work. The double quote is represented by the string "\"".

Operators

Operators are the symbols used for arithmetic and logical operations. Arithmetic symbols define operations that apply to numbers (for example, 2 + 3). Operators, except the plus sign (+), are used only for arithmetic calculations. The + operator can be used with strings as well, as shown in the previous example of string literal concatenation. Tables 6.3 through 6.7 list all the Java operators.

Table 6.3. The Java arithmetic operators.

Operator	Operation	Example
+	Addition	g + h
-	Subtraction	g - h
*	Multiplication	g * h
/	Division	g / h
%	Modulus	g % h

Table 6.4. The Java assignment operators.

Operator	Operation	Example	Meaning
=	Assign value	a = 7	a = 7
+=	Add to current variable	g += h	g = g + h
-=	Subtract from current	g -= h	g = g - h

*=	Multiply current	g *= h	g = g * h
/=	Divide current	g /= h	g = g / h
%=	Modulus current	g %= h	g = g % h

Table 6.5. The Java increment and decrement operators.

Operator	Operation	Example	Meaning
++	Increment by 1	g++ or ++g	g = g + 1
--	Decrement by 1	g-- or --g	g = g -1

Table 6.6. The Java comparison operators (which return true or false).

Operator	Operation	Example	Meaning
==	Equal	g == h	Is g equal to h?
!=	Not equal	g != h	Is g not equal to h?
<	Less than	g < h	Is g less than h?
>	Greater than	g > h	Is g greater than h?
<=	Less than or equal	g <= h	Is g less than or equal to h?
>=	Greater than or equal	g >= h	Is g greater than or equal to h?

Table 6.7. The Java bitwise operators.

Operator	Operation
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
>>>	Zero fill right shift
~	Bitwise complement

<<=	Left shift assignment
>>=	Right shift assignment
>>>=	Zero fill right shift assignment
x&=y	AND assignment
x|=y	OR assignment
x^=y	NOT assignment

Comments

Some programmers believe that if something is difficult to code, it should be difficult to maintain. There is a special place in the cosmos for them; let's hope that it's not on your project! Of course, comments should be used throughout code to explain the programmer's rationale. Comments also can be used to block out certain code sections for testing purposes.

Comments have an initial indicator (usually text) and an end indicator. Sometimes comments are used to separate logic, in which case there may be no text.

Table 6.8 illustrates three ways to indicate comments in Java.

Table 6.8. Comment indicators.

Start	Text	End Comment
/*	text	*/
/**	text	*/
//	text	(everything to the end of the line is ignored by the compiler)

The /* comment is familiar to C programmers. The /** comment is typically used for machine-generated comments. The third style is familiar to C++ programmers.

The following are examples of comments:

```
/* this is an example of a comment */

/** this is another example of a comment. I can
    go to multiple lines and the compiler will
```

```
   look at this as a comment until I do the proper
   end comment as in */

// with this method I can't go to multiple lines unless

// I start each line with the comment marker.
```

Comments can be used anywhere in code without affecting the code's execution or logic. Be sure to terminate the comment, or the compiler will get confused.

Q. ***Write a program to accept two numbers from user and find their summation.***

```
import java.io.*;
class sum
{
            public static void main(String args[]) throws IOException
            {
                  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
             int x,y,z;
            System.out.println("Enter the first number=");
            x=Integer.parseInt(br.readLine());
            System.out.println("Enter the second number=");
            y=Integer.parseInt(br.readLine());
            z=x+y;
            System.out.println("Summation is="+z);
            }
}
```

Points to be remember:

public : main() must be declared as public , because it will be called automatically and starts the program from here.

static : it determines main method can be called without creating an object of the class.

void() :- main() does return any value to its caller.

String arg[] := here arg[] is an array of String class.

System.in :- “System” is class and “in” is the field of System class.”System.in” creates an object that will be connected to the input device such as keyboard.

BufferedReader; it is also a class to create a Buffer object in the memory and it is used to accept information from its keyboard in the form of text .it holds the information in the buffered temporary .

readLine() is the method available in BufferedReader class and it is responsible to read the line of text from the Buffer.

System.out:- System is the class name and out is a static variable in System class.Out is called a field in System class. When we call this field ,an object of PrintStream class will be created internally.System.out gives an object of PrintStream class and this object by default represents the standard output device i.e the monitor.

println or **print** method belongs to PrintStream class and it is used to display information on the screen.

The Simple Assignment Operator

One of the most common operators that you'll encounter is the simple assignment operator "=". You saw this operator in the Bicycle class; it assigns the value on its right to the operand on its left:

```
int cadence = 0;
int speed = 0;
int gear = 1;
```

The Arithmetic Operators

The Java programming language provides operators that perform addition, subtraction, multiplication, and division The only symbol that might look new to you is "%", which divides one operand by another and returns the remainder as its result.

+	additive operator (also used for String concatenation)
-	subtraction operator
*	multiplication operator
/	division operator
%	remainder operator

```
class ArithmeticDemo
        {

                public static void main (String[] args)
                {

                // result is now 3
                  int result = 1 + 2;
                    System.out.println(result);

                 // result is now 2
                  result = result - 1;
                 System.out.println(result);

                  // result is now 4
                   result = result * 2;
                  System.out.println(result);

                  // result is now 2
                  result = result / 2;
```

```
                    System.out.println(result);

                    // result is now 10
                    result = result + 8;
                     // result is now 3
                     result = result % 7;
                     System.out.println(result);
                    }
            }
```

You can also combine the arithmetic operators with the simple assignment operator to create *compound assignments*. For example, `x+=1;` and `x=x+1;` both increment the value of `x` by 1.

The + operator can also be used for concatenating (joining) two strings together, as shown in the following `ConcatDemo` program:

```
class ConcatDemo {
   public static void main(String[] args){
      String firstString = "This is";
      String secondString =
         " a concatenated string.";
      String thirdString =
         firstString+secondString;
      System.out.println(thirdString);
   }
}
```

By the end of this program, the variable `thirdString` contains "This is a concatenated string.", which gets printed to standard output.

The Unary Operators

The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

The following program, `UnaryDemo`, tests the unary operators:

```
class UnaryDemo
        {

                public static void main(String[] args)
                {
                 // result is now 1
                   int result = +1;
                   System.out.println(result);
                   // result is now 0
                   result--;
                   System.out.println(result);
                   // result is now 1
                  result++;
                System.out.println(result);
                 // result is now -1
                 result = -result;
                  System.out.println(result);
                  boolean success = false;
                   // false
```

```
                     System.out.println(success);
                    // true
                     System.out.println(!success);
                    }
            }
```

The increment/decrement operators can be applied before (prefix) or after (postfix) the operand. The code `result++;` and `++result;` will both end in `result` being incremented by one. The only difference is that the prefix version (`++result`) evaluates to the incremented value, whereas the postfix version (`result++`) evaluates to the original value. If you are just performing a simple increment/decrement, it doesn't really matter which version you choose. But if you use this operator in part of a larger expression, the one that you choose may make a significant difference.

```
class PrePostDemo
        {
                public static void main(String[] args)
                {
        int i = 3;
                 i++;
                // prints 4
                System.out.println(i);
                ++i;
                // prints 5
                System.out.println(i);
                // prints 6
                 System.out.println(++i);
                 // prints 6
                System.out.println(i++);
                // prints 7
                System.out.println(i);
                }
        }
```

The Equality and Relational Operators

The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand. The majority of these operators will probably look familiar to you as well. Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

```
class ComparisonDemo
        {

                public static void main(String[] args)
                {
                 int value1 = 1;
                  int value2 = 2;
                  if(value1 == value2)
                   System.out.println("value1 == value2");
                   if(value1 != value2)
                  System.out.println("value1 != value2");
                  if(value1 > value2)
                 System.out.println("value1 > value2");
                   if(value1 < value2)
                    System.out.println("value1 < value2");
                 if(value1 <= value2)
                System.out.println("value1 <= value2");
                 }
        }
```

Output:

value1 != value2

value1 < value2

value1 <= value2

The Conditional Operators

The `&&` and `||` operators perform *Conditional-AND* and *Conditional-OR* operations on two boolean expressions. These operators exhibit "short-circuiting" behavior, which means that the second operand is evaluated only if needed.

&& Conditional-AND
|| Conditional-OR

The following program, `ConditionalDemo1`, tests these operators:

```
class ConditionalDemo1
        {

                public static void main(String[] args)
                {
                 int value1 = 1;
                 int value2 = 2;
                if((value1 == 1) && (value2 == 2))
                  System.out.println("value1 is 1 AND value2 is 2");
                 if((value1 == 1) || (value2 == 1))
                 System.out.println("value1 is 1 OR value2 is 1");
                }
        }
```

Another conditional operator is `?:`, which can be thought of as shorthand for an `if-then-else` statement (discussed in the Control Flow Statements section of this lesson). This operator is also known as the *ternary operator* because it uses three operands. In the following example, this operator should be read as: "If `someCondition` is `true`, assign the value of `value1` to `result`. Otherwise, assign the value of `value2` to `result`."

The following program, ConditionalDemo2, tests the `?:` operator:

```
class ConditionalDemo2
        {

                public static void main(String[] args)
                {
                  int value1 = 1;
                 int value2 = 2;
                 int result;
                 boolean someCondition = true;
                 result = someCondition ? value1 : value2;
                  System.out.println(result);
                }
        }
```

Because `someCondition` is true, this program prints "1" to the screen. Use the `?:` operator instead of an `if-then-else` statement if it makes your code more readable; for example, when the expressions are compact and without side-effects (such as assignments).

The Type Comparison Operator instanceof

The `instanceof` operator compares an object to a specified type. You can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.

The following program, InstanceofDemo, defines a parent class (named `Parent`), a simple interface (named `MyInterface`), and a child class (named `Child`) that inherits from the parent and implements the interface.

```
class InstanceofDemo
        {
                public static void main(String[] args)
                {
                        Parent obj1 = new Parent();
                          Parent obj2 = new Child();


                         System.out.println("obj1 instanceof Parent: "
                         + (obj1 instanceof Parent));
                          System.out.println("obj1 instanceof Child: "
                           + (obj1 instanceof Child));
                        System.out.println("obj1 instanceof MyInterface: "
                           + (obj1 instanceof MyInterface));
                        System.out.println("obj2 instanceof Parent: "
                          + (obj2 instanceof Parent));
                        System.out.println("obj2 instanceof Child: "
                         + (obj2 instanceof Child));
                         System.out.println("obj2 instanceof MyInterface: "
                           + (obj2 instanceof MyInterface));
                        }
            }

class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}
```

Output:

```
obj1 instanceof Parent: true
obj1 instanceof Child: false
obj1 instanceof MyInterface: false
obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof MyInterface: true
```

When using the `instanceof` operator, keep in mind that `null` is not an instance of anything.

Bitwise and Bit Shift Operators

The Java programming language also provides operators that perform bitwise and bit shift operations on integral types. The operators discussed in this section are less commonly used. Therefore, their coverage is brief; the intent is to simply make you aware that these operators exist.

The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a `byte` contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

The signed left shift operator "<<" shifts a bit pattern to the left, and the signed right shift operator ">>" shifts a bit pattern to the right. The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator ">>>" shifts a zero into the leftmost position, while the leftmost position after `">>"` depends on sign extension.

The bitwise `&` operator performs a bitwise AND operation.

The bitwise `^` operator performs a bitwise exclusive OR operation.

The bitwise `|` operator performs a bitwise inclusive OR operation.

The following program, `BitDemo`, uses the bitwise AND operator to print the number "2" to standard output.

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

```
                    {
                    int bitmask = 0x000F;
                     int val = 0x2222;
                      // prints "2"
                     System.out.println(val & bitmask);
                    }
          }
```

Expressions, Statements, and Blocks

Now that you understand variables and operators, it's time to learn about *expressions*, *statements*, and *blocks*. Operators may be used in building expressions, which compute values; expressions are the core components of statements; statements may be grouped into blocks.

Expressions

An *expression* is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language that evaluates to a single value. You've already seen examples of expressions, illustrated in bold below:

int **cadence = 0**;
anArray[0] = 100;
System.out.println(**"Element 1 at index 0: " + anArray[0]**);

int **result = 1 + 2**; // result is now 3
if (**value1 == value2**)
    System.out.println(**"value1 == value2"**);

The data type of the value returned by an expression depends on the elements used in the expression. The expression `cadence = 0` returns an `int` because the assignment operator returns a value of the same data type as its left-hand operand; in this case, `cadence` is an `int`. As you can see from the other expressions, an expression can return other types of values as well, such as `boolean` or `String`.

The Java programming language allows you to construct compound expressions from various smaller expressions as long as the data type required by one part of the expression matches the data type of the other. Here's an example of a compound expression:

```
1 * 2 * 3
```

In this particular example, the order in which the expression is evaluated is unimportant because the result of multiplication is independent of order; the outcome is always the same, no matter in which order you apply the multiplications. However, this is not true of all expressions. For example, the following expression gives different results, depending on whether you perform the addition or the division operation first:

```
x + y / 100    // ambiguous
```

You can specify exactly how an expression will be evaluated using balanced parenthesis: (and). For example, to make the previous expression unambiguous, you could write the following:

```
(x + y) / 100  // unambiguous, recommended
```

If you don't explicitly indicate the order for the operations to be performed, the order is determined by the precedence assigned to the operators in use within the expression. Operators that have a higher precedence get evaluated first. For example, the division operator has a higher precedence than does the addition operator. Therefore, the following two statements are equivalent:

```
x + y / 100
```

```
x + (y / 100) // unambiguous, recommended
```

When writing compound expressions, be explicit and indicate with parentheses which operators should be evaluated first. This practice makes code easier to read and to maintain.

Statements

Statements are roughly equivalent to sentences in natural languages. A *statement* forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

- Assignment expressions
- Any use of ++ or --
- Method
- Object creation expressions

Such statements are called *expression statements*. Here are some examples of expression statements.

```
// assignment statement
aValue = 8933.234;
// increment statement
aValue++;
// method invocation statement
System.out.println("Hello World!");
// object creation statement
Bicycle myBike = new Bicycle();
```

In addition to expression statements, there are two other kinds of statements: *declaration statements* and *control flow statements*. A *declaration statement* declares a variable. You've seen many examples of declaration statements already:

```
// declaration statement
double aValue = 8933.234;
```

Finally, *control flow statements* regulate the order in which statements get executed.

Blocks

A *block* is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following example, illustrates the use of blocks:

```
class BlockDemo
        {
                public static void main(String[] args)
                {
                 boolean condition = true;
                 if (condition)
                        { // begin block 1
                         System.out.println("Condition is true.");
                        } // end block one
                else
                        { // begin block 2
                         System.out.println("Condition is false.");
                        } // end block 2
                }
        }
```

CONDITIONAL STATEMENT

if……..else

Working principle of if……..else statement:-

1. if the condition is true then the statement of if part will execute otherwise java looks for the if part or else part(if it is available) to execute the statement.

Note:

2) java is a case sensitive language.

3) you will have to write "if" and "else" in lower case to avoid the error.

4) Else is an optional part of the condition statement.

5) Braces are optional,if there is a single line statement or code after if condition and else part.

6) it works on boolean value.

SYNTAX:-

1. **if(condition)**

```
            {
                        Statement;


            }
```

2. **if (condition)**

```
            {
                        Statement;
            }
 Else
                        Statement;
```

3. **if(condition)**

```
  {
                        Statement;


  }
```

4. **if(condition)**

```
  {
            Statement;


  }
if(condition)
 {
           Statement;
 }
```

5. **if(condition)**

```
 {
            if(condition)
                Statement
            If(condition)
```

```
                **Statement**
        **Else**
                **Statement**
    **}**
    **Else**
    **{**
        **Statement**
    **}**
```

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

        int testscore = 76;
          char grade;
          if (testscore >= 90)
              {
              grade = 'A';
              } else if (testscore >= 80)
                  {
                  grade = 'B';
                  } else if (testscore >= 70)
                      {
                      grade = 'C';
                       } else if (testscore >= 60)
                          {
                          grade = 'D';
                          } else
                              {
                              grade = 'F';
                              }
        System.out.println("Grade = " + grade);
```

```
                    }
          }
```

Switch case

This structure is used to execute sequence of statement on the basis of case specified

The working principle of switch case is as follows

a. At first the value will dropped within the parenthesis of switch case.
b. Switch case is matched the value with the case constant value.
c. If matching is found then the associate statement will execute otherwise it looks to default statement (if is present).

Notes

1. break statement with case is used to execute sequence of statements associated with particular case and transfer the control to the statement just after case.
2. if break is not used with case statement, then control will be transfer to the next case statement unless if finds break.
3. only constant value is allowed with the case statement.

Syntax:-

```
switch(variable)
{
        case constant1:
                sequence of statements;
                break;
        case constant2:
                sequence of statements;
                break;
        ………………………………..
        ………………………………..
        ………………………………..
```

```
                    default
                            sequence of statements
              }
```

Example:-

```
public class SwitchDemo
          {
                    public static void main(String[] args)
                    {
                    int month = 8;
                     String monthString;
                     switch (month)
                              {
                              case 1:  monthString = "January";
                                         break;
                              case 2:  monthString = "February";
                                         break;
                              case 3:  monthString = "March";
                                         break;
                                case 4:  monthString = "April";
                                         break;
                              case 5:  monthString = "May";
                                         break;
                              case 6:  monthString = "June";
                                          break;
                               case 7:  monthString = "July";
                                         break;
                               case 8:  monthString = "August";
                                          break;
                               case 9:  monthString = "September";
                                         break;
                                case 10: monthString = "October";
```

```
                                    break;
                        case 11: monthString = "November";
                                    break;
                         case 12: monthString = "December";
                                    break;
                        default: monthString = "Invalid month";
                                    break;
                        }
                System.out.println(monthString);
                }
        }
```

□Looping statement:

Looping statement is the way to execute sequence of statement again and again till the given condition is over means to say that if you execute or run the common task again and again then you will have to use loop.
The working principle of loop:

1. at first it check initialization portion.
2. after initialization it check the condition.
3. if the condition is true then the statement will execute otherwise the control will transfer to the statement just after loop.

Syntax:-

```
        <Initialization>;
    while (condition)
    {
            statement(s);
            increment/decrement;
    }
```

Note:

1. Condition must enclose within parenthesis “()”.
2. Braces are optional, if there is a single statement is being used.
3. Braces are required, if there is multiline statement.

```
class WhileDemo
        {
          public static void main(String[] args)
                {
                 int count = 1;
                  while (count < 11)
                        {
                          System.out.println("Count is: "+ count);
                        count++;
                        }
                }
        }
```

ARRAY

1. What is an array and why do we use an array?
2. What are the properties of an array?
3. What are demerit of an array?
4. Describe the types of an array.

Ans.1,2,3,4

Let us suppose you are using multiple variables having same data type. For example x1, x2, x3, x4… have integer type and we know that all these variables store in different location of memory and also it is very complex task to remember the name of variables in the program . Java also takes enough time to bind the variables from different location of memory and due to this reason it makes the program very slow. To solve this problem, Java provides a very popular concept called array. An array is a container that holds collection of value or data having same data type.

There are following properties of an array to make it very useful.

1. It contains collection of data having same data type.
2. It arranges all the data in the consecutive area of memory.
3. Java can easily find all the data of an array because all the data are store in consecutive area of memory.
4. It is very easy to access or maintain all the data of an array.
5. It makes the program very small.
6. The main feature of an array is that if you pass an array to the function then it means you are passing the reference of an array to the function such that any changes made by function will effect to the actual array.
7. In an array all the variables will have same name and to access each variables we use index and index always starts from zero (0).

The main demerit of an array is wastage of space, Let us suppose you have acquired 100 spaces of integer type and the size of 100 spaces is 100*4 where 4 byte is the length of integer type and total size is 400 byte.In the program if you use only 10 spaces for the operation then you are wasting 90 spaces (90*4=360 byte) that is the demerit of an array.

To solve the problem it is the program /user responsibility to use the space on the requirement basis.

There are two types of an array as single dimensional array and multidimensional array.

How to declare single dimensional array :-

Single dimensional array stores the data into multiple columns of single row means it uses only one dimension to store or access the data for an array

Figure

4	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7

Index

Declaration of single dimesional array :-

Syntax:-

Datatype [] variable name ;

Datatype variable name[] ;

e.g.

```
int [] num;
int num[] ;
```

Declaration of an array does not ensures that array has been created rather than it tells the Java compiler that the array is going to be allocate consecutive area of memory of given size.

There are two ways to initialize an array :-

At first you will declare an array then after using “new” reserve word you can allocate an area of memory.

Syntax :-

<Datatype> [] variable Name ;

< Datatype > variable Name [] ;

Variable name = new <Datatype> [size] ;

Allocation of area of memory.

int[] num;

Or

int num [] ;

num =new int [10]

Above example shows that ‘num’ is a variable of single dimensional array. At first it is declared with data type and after that with the help of given size it allocates an area of memory.

```
import java.io.*;

   class array1

        {

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

                {

                 BufferedReader br=new BufferedReader (new InputStreamReader(System.in));

                 int i;

                 int num[];
```

```
                    num=new int[10];
                     for(i=0;i<10;i++)
                            {
                                    System.out.println("enter a number    ");
                                    num[i]=Integer.parseInt(br.readLine());
                            }
                     for(i=0;i<10;i++)
                            {
                                    System.out.println(num[i]);
                            }
                    }
            }
```

In the second process you will allocate an area of memory through an array in the time
of declaration. Simply you can say that array can allocate an area of memory at the time of declaration.

Syntax:

1. <Data Type>[] Variable Name=new <Data Type>[Size];
2. <Data Type Variable Name[]=new <Data Type>[Size];

1. int[] num=new int[10];
2. int num[]=new int[10];

```
class array1
        {
                public static void main(String []args)throws IOException
                {
                 BufferedReader br=new BufferedReader (new InputStreamReader(System.in));
                 int i;
                 int num[]=new int[10];
                for(i=0;i<10;i++)
                        {
                                System.out.println("enter a number    ");
                                num[i]=Integer.parseInt(br.readLine());
                        }
                 for(i=0;i<10;i++)
                        {
                                System.out.println(num[i]);
                        }
                }
         }
```

HOW TO INITIALIZE AN ARRAY AT THE TIME OF DECLARATION

In this process you can input the data into an array at the time of declaration.

1. <Data type>[] Variable Name={Value1,Value2,Value3,…..Value N};
2. <Data type> Variable Name[]={Value1,Value2,Value3,…..Value N};

```
e.g int[] num={4,5,6,7,9};
    String s[]={"Amit","Sumit","Dilip"};
```

```
class array
{
        public static void main(String[] a)
        {
                int n[]={1,2,3,4,5,6,7,8,9};
                int i,sum=0;
         for(i=0;i<num.length;i++)
                {
```

```
                                System.out.println(num[i]);
          Sum+=num[i];
                    }
                    System.out.println("Sum of all numbers="+sum);


                    }
          }
```

Points

1. "length" is the most important property of an array that is used to find the length of an array.
2. Index of an array always starts from "0";
3. "new" is a reserve word that is used to dynamically or run time allocate an array of memory.
4. Array can be initialize at the time of declaration.

TWO DIMENSIONAL ARRAY:-

Two dimensional array is the part of multidimensional array and it stores or holds the data into multiple rows and columns.In other words you can say that if there are number of rows called m and number of columns are called n, then the two dimensional array stores m*n data.

Two dimensional array uses two subscript or index variable to access each elements of an array.

Syntax:-

1. data type[][] variable name=new data type[row size][column size];

2. data type variable name[][]=new data type[row size][column size];

3.data type variable name[]={{value1,value2},{value3,value4},……,{valueN-1,valueN}};

e.g.

1. int[][] num=new int[4][5];

2. int num[][]=new int[4][5];

3. String[][] str={{“Mr”,”Mrs”},{“Amit”,”Jones”};

The following program numbers each element in the array from left to right, top to bottom, and then displays these values:

```
 // demonstrate a two-dimensional array.

class TwoDArray
          {
           public static void main(String args[])
                    {
                                   int twoD[][]= new int[4][5];
                                   int i, j, k = 0;

                                   for(i=0; i<4; i++)
                    for(j=0; j<5; j++)
                        {
                             twoD[i][j] = k;
                              k++;


                        }


                        for(i=0; i<4; i++)
                        {
                        for(j=0; j<5; j++)
                        System.out.print(twoD[i][j] + " ");
                        System.out.println();
                        }
                    }
          }
```

This program generates the following output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

COMMAND LINE ARGUMENT

Command line argument provides facility to the user to input the value or pass the value at the run time of program. The data type of passing value String. The value or data which is passed by the user will be arranged in an array and on the basis of subscript or index you can perform the operation on an array.

There is a method of array called "***length***" which is responsible to find how many number of elements are available in the array.

Example:-

```
class employeeidstring
        {
                public static void main(String a[])
                        {
                                String eid;
                                double bsalary=0, ta=0, da=0, hra=0, gross_sal=0;

                          if(a.length!=2)
                                System.out.println("Only two characters are allowed....");
                          else
                           {
                                bsalary=Double.parseDouble(a[1]);
                                ta=bsalary*0.02;
                                da=bsalary*0.02;
                                hra=bsalary*0.02;
                                gross_sal=ta+da+hra+bsalary;
                                System.out.println("gross salary =" +gross_sal);
                                eid=a[0];
                                System.out.println("employee id =" +eid);

                           }
                        }
        }
```

CLASS AND OBJECT

***Discuss the object and object oriented programming*:-**

The fundamental idea behind the object oriented language is to combine into a single unit both data(member data) and the method (member method) where member method operates on the data such unit is called an object.

An object provides the only way to access the member of class. The data is hidden and is save from accidental alteration. Data and method are encapsulated into a single entity. Data encapsulation and data hiding are the key terms used for describing an object oriented programming language.

OOPS is a software development technique that has gained popularity because of the potential gains in program productivity over conventional software development methodology. To support the principle of object oriented programming, all oops language must support following three common features.

- Encapsulation
- Polymorphism
- Inheritance

Encapsulation:

Encapsulation is a mechanism that binds together code and the data; it manipulates and keeps both safe from outside interference and misuse.

For an object oriented language code and data may be combines in such a way that a self contained black box is created when code and data are linked together in the fashion to create an object.

In other words an object is the device that supports encapsulation.

Polymorphism:

All object oriented programming language support polymorphism, i.e one interface multiple method. In simple term polymorphism is an attribute that has one interface control access to a general class of action.

<u>Inheritance</u>:

Inheritance is a process by which object can acquire the properties of another object. This is important because it supports the concept of classification

For e.g. - a red delicious apple is the part of the classification apple which in term is the part of fruit class, which is under the larger class food without the use of classification. Each object would have to define explicitly all of its characteristics .However through use of classification an object need only to define those qualities that makes it unique within the class.

<u>Following are the advantage of oops:</u>

- **Simple Approach**: The objects are real world model which result in simple programming structures.
- **Reusability:** Objects once made can be reused in more than one program.
- **Flexibility**: Application built on Object Oriented Programming can be flexible in adopting to different situation because interaction between object does not effect the internal working of object.
- **Maintainability:** Objects are separate entity, which can be main separately align fixing of bugs or any other changes easily.

Syntax:-

```
Class <Class name>
{
        [Access specifier] <Data Type> instance-variable1;
        [Access specifier] <Data Type> instance-variable2;
        …………………………………………………….
        …………………………………………………….
        [Access specifier] <Return type> method-name1([Parameter list])
        {
                //Body of method.
```

```
                        }
                        [Access specifier] <Return type> method-name2([Parameter list])
                        {
                                //Body of method.
                        }
                }
```

Points:

- Collectively, the methods and variables defined within a class are called member of the class.
- Variables and methods defined within a class are called instance variable and instance method of the class because each instance of the class i.e each object of the class contains its own copy of variables and method.

Declaring objects

There are two steps process to declare object of class:-

- First, you must declare a variable of the class type and variable does not define an object.
- Using the "new" operator you must create an object and passes the reference of object to the variable.

The "***new***" operator dynamically allocates memory for an object and return the reference of an object to the variable.

- In java objects must be dynamically created.

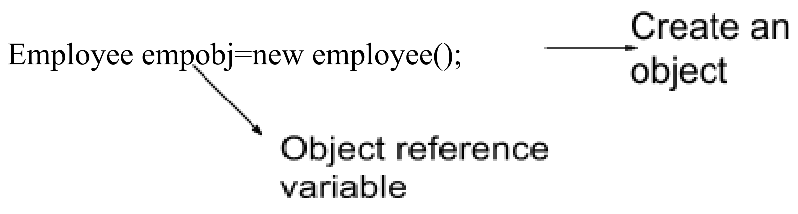
ASSINGNING OBJECT REFERENCE TO THE VARIABLE:

When you assign one object reference variable to another object reference variable ,you are not creating a copy of the object rather than you are only making a copy of the reference.

```
                Box b1=new Box();
                Box b2=b1;
```

As we know that new operator is responsible to create an object and it passes the reference of an object to the object reference variable.

Employee empobj=new employee();



So we can say that empobj is object reference variable. When you are passing one object reference variable to another object reference variable, you are not creating a copy of the object, and you are only making the reference.

A CLOSER LOOK AT ARGUMENT PASSING

There are two ways that java can pass argument to the method

1 call by value.
2 call by reference.

1) **call by value**: in call by value process the copy of actual parameter will be pass to the formal parameter of the method. In this process any changes made in formal parameter will not effect the value of actual parameter.

2) **call by reference**: when you are passing object of particular class to the formal parameter of called method then in this situation you are passing reference of the object to the formal parameter.Remember that in call by reference process you always pass the reference of the object not the copy of object, any changes made in the formal parameter will effect to the actual parameter.

ACCESS SPECIFIER

Access specifier specifies the accessing limitation of object member. It provides some reserve word,on the basis of these reserved word the program specifies what is the limitation or access capacity of the member of an object.

Basically access specifier is used for preventing from mistake. There are following access specifier are available in java:-

i) **private:** if you specify the member of the class as private, then that member will be only access by the other member of its class means these type of member cannot be accessed by the outside of its class.

ii) **Public:** if you specify the member of class as public, then member can be accessed by the outside of its class and also it can be accessed by other class which is located in another location.

iii) **Protected:** if you specify the member of class as protected then it will be accessible by inherit class.

iv) **Default access specifier:** java also supports default access specifier where you don't use private,public or protected and the member of default access specifier will only be accessable to its current file or current package(folder).

METHOD

A method in `JAVA language` is a block of code that performs a specific task. It has a name and it is reusable i.e. it can be executed from as many different parts in a `JAVA Program` as required. It also optionally returns a value to the calling program

So method in a JAVA program has some properties discussed below.

- Every method has a unique name. A method can be called from within another method.
- A method is independent and it can perform its task without intervention from or interfering with other parts of the program.
- A method performs a specific task, Such as adding two or more integer, sorting an array into numerical order, or calculating a cube root etc.
- A method returns a value to the calling program. This is optional and depends upon the task your method is going to accomplish.

Advantages of using methods:

There are many advantages in using method in a program they are:

- The length of the source program can be reduced by using methods
- A method may be used later by many other programs this means that a JAVA programmer can use method written by others.
- A method can be used to keep away from rewriting the same block of codes which we are going use two or more locations in a program. This is especially useful if the code involved is long or complicated.

Method Components:

Method declaration or prototype:-
It determines the model or blue print for the method which tells the compiler about the name of method, its return type and types of arguments.

Method definition:-

Actual method itself is referred to as method definition. The first line of the method definition is known as method declarator.

Method call:-

Method call is specified by the method name followed by arguments, enclosed in parenthesis and terminated by a semicolon. It calls actual method or called method to be executed.

Parameter:-

Variable that holds the value being passed to the method is called parameter. When parameter passed to method call are called actual parameter and parameters passed to called method or actual method is called formal parameter.

Return:-

Method can return the value to its caller (method call) using return reserve keyword. If a method does not return anything then it will be indicated by the keyword void.

Note:-

1. If a method returns values to its caller then return value, return type and the variable which is going to hold return value must be same.
2. After executing the method then control is transferred back to its caller.

Method Overloading:

Method overloading is a technique by which more than one method can have same name as far as their signature differs:

- More than one method can have same name if these number of parameter differs.
- More than one method can have same name if the data type of the parameter differs.
- More than one method can have same name if the sequence of the parameters differs

Note: No two methods can be overloaded on the basis of return type.

```
class Bicycle
{

    private int cadence = 0;
    private int speed = 0;
    private int gear = 1;

    public void changeCadence(int newValue)
           {
                     cadence = newValue;
           }

    public void changeGear(int newValue)
            {
                      gear = newValue;
           }

    public void speedUp(int increment)
            {
                     speed = speed + increment;
            }

    public void applyBrakes(int decrement)
           {
                      speed = speed - decrement;
           }

    public void printStates()
           {
                      System.out.println("cadence:" +  cadence + " speed:" +speed + " gear:" + gear);
           }
}
class BicycleDemo
```

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

                         // Create two different
                        // Bicycle objects
                         Bicycle bike1 = new Bicycle();
                         Bicycle bike2 = new Bicycle();

                          // Invoke methods on
                          // those objects
                           bike1.changeCadence(50);
                           bike1.speedUp(10);
                           bike1.changeGear(2);
                           bike1.printStates();

                           bike2.changeCadence(50);
                           bike2.speedUp(10);
                           bike2.changeGear(2);
                          bike2.changeCadence(40);
                          bike2.speedUp(10);
                          bike2.changeGear(3);
                          bike2.printStates();
                         }
        }
```

CONSTRUCTOR

Constructor is used to initialize the member data of class before creating an object of the class. It is a special method which has same name as that of the class and which follow the given properties-

- A constructor has no return type such as void and other datatype but by mistake if you use return type then constructor will be behave as method of class and it can be access by object of the class using dot(.) operator.

- When you don’t explicitly defines the constructor for a class, then java creates a default constructor for the class and also it initializes all the variables / member data of the class.

- A constructor can accept arguments we can say that the constructor can be overloaded.

- Constructor should have same name as that of class .

- If you specify of your own constructor for a class, the default constructor of java will no longer used.
- When you are going to create an object,object depends upon the description of constructor as you define in the class. Let us suppose if you define an overloaded constructor in the class “xyz” and there is no default constructor of class “xyz” , at that time you can create an object of parameterized constructor but you can not create an object of default constructor.

- Constructors always call or execute before creating an object.

- The access specifier of constructor should be public.

- An object of particular class can execute only one constructor at a time means to say that one object will execute only one constructor of particular class whether there may be more than one constructor in the class.

```
class another
{
 private int x,y;
 public another(int a, int b)
            {
                  x = a;
                  y = b;
            }
 public another()
```

```
                 {
                  x=0;
                 y=0;
                 }
  public int area()
                 {
                 int ar = x*y;
                 return(ar);
                  }
}
class construct
{
   public static void main(String[] args)
                 {
                          another a = new another(12,12);
                            System.out.println("Area of rectangle : " + a.area());
                            another b = new another();
                           System.out.println("Area of rectangle : " + b.area());
                 }
}
```

STATIC METHOD AND STATIC VARIABLE:

Using static keyword member data and member method can be accessed by the class without creating an object of the current class, Means to say that when the member data or member method is declared with static, it can be accessed before creating an object of its class.

Member of the class that is declare with static keyword does not create an instance of or copy of a static member means to say that objects of the class share the same member at a time.

There are following restrictions with static method are described below:-

- It can only call other static method.
- Static method can access only static data.

- They cannot refer to this or super reserve keyword in any way.
- Non static member method can access the static data.

Note:

Outside of the class in which static method or static data are defined ,can be used independently of any object. To do so you have to specify the class name with dot(.) operator.

```
public class HowToAccessStaticMethod
{
  int i;
  static int j;
  public static void staticMethod()
                      {
                                 System.out.println("you can access a static method this way");
                      }
  public  void nonStaticMethod()
                      {
                                 //i=100;
                                 //j=1000;
                                 System.out.println("Don't try to access a non static method");
                      }
  public static void main(String[] args)
                      {
                                  //i=100;
                                 //j=1000;
                                  //nonStaticMethod();
                                  staticMethod();
                       }
}
```

ABOUT “this “RESERVE KEYWORD:-

“this” reserve keyword points or holds the reference of current object of class. It is also used to call the constructor of current class.

If the name of member data of current class and the name of formal parameter of the method having the same name and that time you are going to assign formal parameter value to the member data. Then in this situation this reserved keyword helps to JVM to decide which one is member data and formal parameter.

ABOUT “super” RESERVE KEYWORD: - super is a reserve keyword provided by java which holds the reference of parent class or base class. ”super” reserve keyword can be use for the following purpose:-

 With the help of super reserve keyword you can access member data or member method of base class by its inherited class.

“super” reserve keyword always refers to the base class of the child class.

Syntax:-

i) super.<base class member data>;

ii) super.<base class member method>([Formal parameter list]);

 Another use of super reserve keyword is that ,you can access the default constructor or overloaded constructor of the base class. It should always be the first statements in the child class constructor to call the base class constructor.

Q.WHAT ARE THE DIFFERENCE BETWEEN “this” AND “super” RESERVED KEYWORD?

this	**super**
1) “this” holds the reference of current Object.	1) “super” holds the reference of the base class.
2) with the help of “this” you can call the current class constructor.	2) with the help of “super” reserved keyword you can call base class constructor.

3) "this" reserve keyword can be the part of Inherited class or non inherited class.	3) it always used in inherited class.s

Examples:-

```
class another
            {
                  private int a,b;
                  public another(int a, int b)
                   {
                       this.a = a;
                       this.b = b;
                   }
                 public another()
                   {
                       a=0;
                       b=0;
                   }
                  public int area()
                  {
                       int ar = a*b;
                        return(ar);
                 }
            }
class construct
            {
                  public static void main(String[] args)
                 {
                   another a = new another(12,12);
                   System.out.println("Area of rectangle : " + a.area());
                  another b = new another();
                   System.out.println("Area of rectangle : " + b.area());
```

```
                        }
                }
```

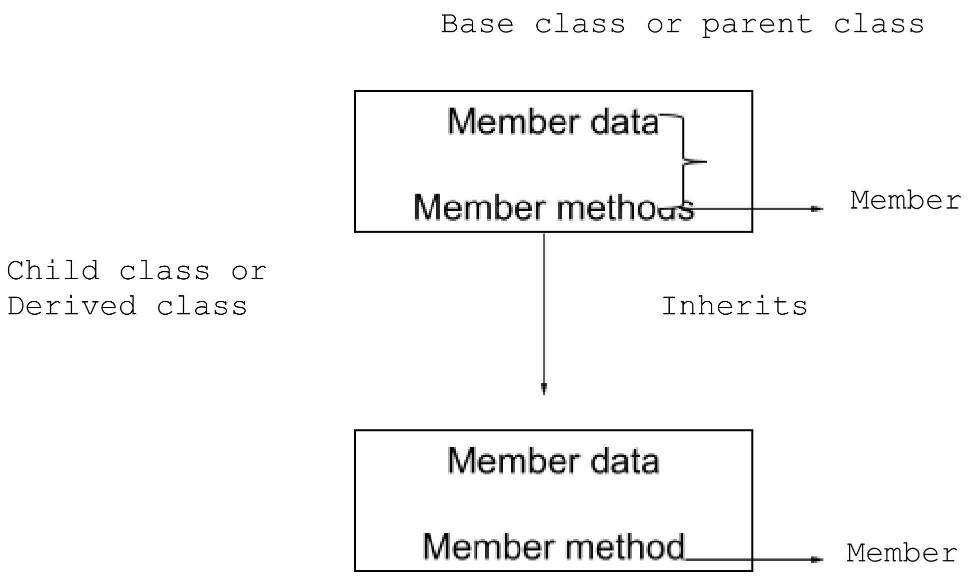
□INHERITANCE

ANS: Inheritance is the process to create a new class from existence class means to say that in Inheritance there should be at least one class available for the inheritance.

Here, the first class is called base class or parent class that may contain collection of methods and member data. The second class is called child class or derived class that will inherit all the properties such as method, member data, and constructor of the base class, child class or derived class can also add some extra properties to its class, so we can say that child class or derived class has more properties in compare to the base class.

In inheritance child class or derived class can be the base class or parent class of new class. Shortly we can say that with the help of an inheritance we can create a new class has more properties then the old class or an existing class.

Diagram of Inheritance:

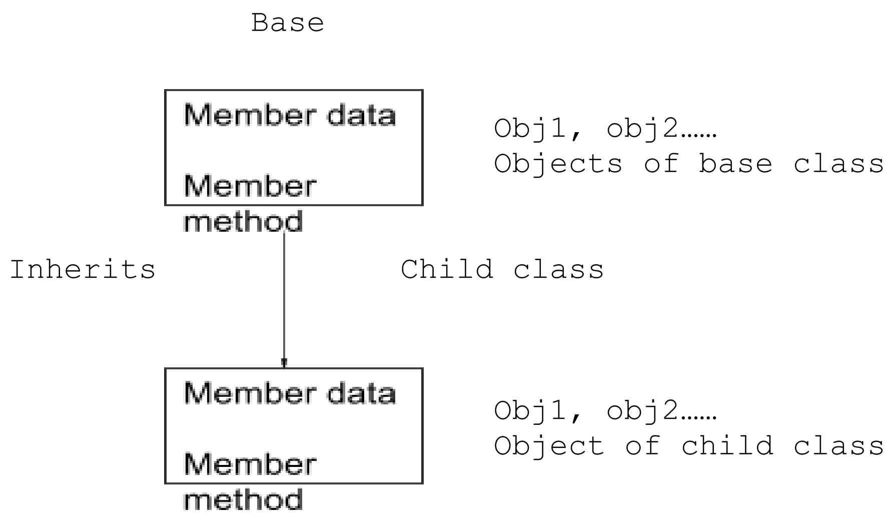


POINTS:

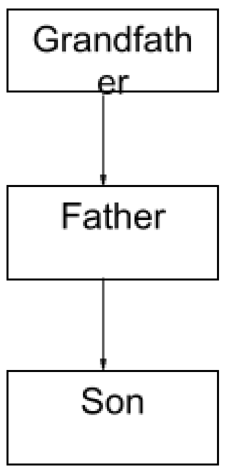
1. By the derived class you can access all the member of its parent class.
2. Always remember that the base class will provide all the member to its child class or derived class without compromise with security.

Features of an inheritance:

a. Derived class or child class does not disturbed to the functionality of its base class means to say that you can create an object of both the class i.e. derive class and base class.



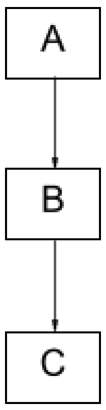
b. Derived class or inherited class has some extra features to its base class or patent class.
c. You can also create a new class that will inherit all the members of derived class means to say that this concept is based upon the following diagram as given below:



By the above diagram we can say that son has the properties of his father and grandfather and also son has own property.

DIFFERENT TYPES OF INHERITENCE

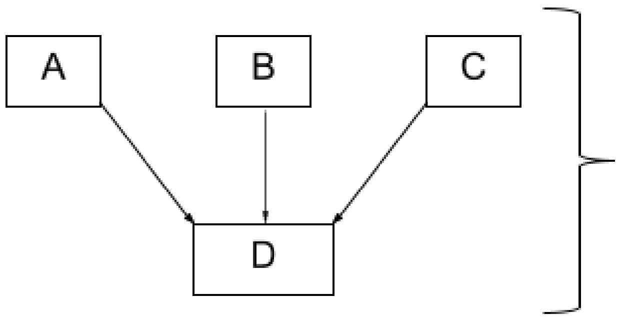
1. Single level inheritance: In this inheritance there will be only two classes, one for parent class or base class and another will be its derived class or child class.
2. Multilevel inheritance: In this inheritance concept more than two classes will arrange hierarchical model.



Here we can say that 'C' inherits the properties of 'B' and also 'B' inherits the properties of 'A'.

1) Multiple inheritances: This inheritance is something complex as compared to single level and multilevel inheritance because the child class or derived class will access the properties of more than one base class. So, we can say that in the multiple inheritances there will be more than one base class for child class.

JAVA does not support multiple inheritances but with the help of an interface it can be possible.



Multiple inheritances

ADVANTAGES OF INHERITENCE

≈ As we know that derived class has more properties as compare to its parent class so we can say that with the help of inheritance we can create a new product or object which has some extra properties including own properties.

≈ Since, inherited class has pre-build methods and member data of its base class it means it saves the extra time which could be taken by the pre-built methods and data.

≈ Inheritance provides facility to make the product easily and fastly and also the newly created product has more properties as compared to an existing product.

NOTE:

1. "Protected" access specifier used in inheritance. It is the combination of private and public access specifier. It is called private because the member data cannot access by the outside of class (parent class and child class) and it is called public because only derived class can access the member data of its base class of parent class.
2. "extends" reserve keyword takes the responsibility to inherit all the properties of its base class means to say that it is used for inheritance.

object.java

```
class base
        {
         protected int i,j;
         public void input(int i,int j)
                    {
                                this.i=i;
                                this.j=j;
                     }
         public void sum()
                    {
                                System.out.println("Summation is"+(i+j));
                    }
        }

class child extends base
        {
                public void mul()
                    {
                                System.out.println("Multiplication is="+(i*j));
                    }
        }

class object
        {
                public static void main(String args[])
                    {
                                child obj1=new child();
                                obj1.input(10,89);
                               obj1.mul();
```

```
                                      obj1.sum();
                              }
                    }
```

CALLING BASE CLASS CONSTRUCTOR THROUGH INHERITED OR DERIVED CLASS USING “super” RESERVE KEYWORD.

```
import javax.swing .*;
class employee
          {
                    protected String eid,ename;
                    protected double ta,da,hra,bsal,tot_sal;
                    public employee()
                         {
                                   eid=ename=null;
                                   ta=da=hra=bsal=0.0;
                         }
                    public employee(String eid,String ename)
                         {
                                   this.eid=eid;
                                   this.ename=ename;
                         }
                    public void calculate()
                         {
                                   ta=bsal*.05;
                                   hra=ta;
                                   da=ta;
                                   tot_sal=ta+da+hra+bsal;
                         }

          }
class emp_child extends  employee
          {
```

```
                    public emp_child()
                        {
                                super();
                        }
                    public emp_child(String id,String name,double bsal)
                        {
                                super(id,name);
                                this.bsal=bsal;
                        }
                    public void display()
                        {
                            JOptionPane.showMessageDialog(null,"Employee id is:::"+eid);
                            JOptionPane.showMessageDialog(null,"Employee name is:::"+ename);
                            JOptionPane.showMessageDialog(null,"Employee ibasic salary is:::"+bsal);
                            JOptionPane.showMessageDialog(null,"TA is:::"+ta);
                            JOptionPane.showMessageDialog(null,"DA is:::"+da);
                            JOptionPane.showMessageDialog(null,"HRA is:::"+hra);
                            JOptionPane.showMessageDialog(null,"Total payment is:::"+tot_sal);
                        }
               }
class call_employee
               {
                    public static void main(String args[])
                        {
                            emp_child eobj[]=new emp_child[2];
                            for(int i=0;i<2;i++)
                              {
                                  String id=JOptionPane.showInputDialog("Enter the employee id for object
                                                                                 "+(i+1));
                                   String nm=JOptionPane.showInputDialog("Enter the employee name for
                                                                                 object "+(i+1));
                                 double bsal=Double.parseDouble(JOptionPane.showInputDialog("Enter the
                                                              employee  basic salary for  object "+(i+1)));
                                eobj[i]=new emp_child(id,nm,bsal);
                                eobj[i].calculate();
                               }
                             for(int i=0;i<2;i++)
                              {
                                eobj[i].display();
                              }
                          }}
```

METHOD OVERLOADING THROUGH INHERITANCE:

```
class base_func_over
            {
                  protected int i,j;
                   public void accept(int i,int j)
                  {
                        this.i=i;
                        this.j=j;
                   }
                   public void display()
                  {
                       System.out.println("Value at i is="+i);
                      System.out.println("Value at j is="+j);
                  }
            }
class child_func_over extends base_func_over
            {
                  public void accept()
                   {
                        this.i=90;
                       this.j=100;
                  }
```

```
                }


class call_method

                {

                public static void main(String args[])

                        {

                             child_func_over obj1=new child_func_over();

                             obj1.accept();

                             obj1.display();

                             obj1.accept(50,50);

                             obj1.display();

                        }

                }
```

CALLING CURRENT CLASS CONSTRUCTOR THROUGH “this” RESERVE KEYWORD:-

```
class base_func_over

                {

                        protected int i,j;

                        public void accept(int i,int j)

                             {

                                        this.i=i;

                                        this.j=j;

                             }

                        public void display()

                            {

                                       System.out.println("Value at i is="+i);

                                       System.out.println("Value at j is="+j);

                             }

                        public base_func_over()

                             {

                                          this(79,89);

                             }
```

```
        public base_func_over(int i,int j)
            {
                        this.i=i;
                       this.j=j;
            }
    public static void main(String args[])
            {
                       base_func_over obj1=new base_func_over();
                       obj1.display();
            }

    }
```

METHOD OVERRIDING:

As we know that java supports multilevel inheritance. in this approach when a method of derived class has same name and same number of parameters and its data type and also return type as that of method described in its super class then the method of the derived class is said to method override the method of super class.

In the multilevel inheritance method override approach is existing and you are going to create an object of derived class then you can access the overrided method of derived class.

Points

- If the method overriding is existing in inheritance then method will be called by the object of current class.
- If two methods have the same name but their signature are differs then it is called method overloading .

DIFFERENCE BETWEEN METHOD OVERLOADING AND METHOD OVERRIDING

METHOD OVERLOADING	**METHOD OVERRIDING**
1) in method overloading two or more than two methods have same name but different signature.	in method overriding two or more than two method have the same name and same signature.
2) It can use same class and also can be use with inherited class.	2) it will always use in base class and sub class
3) the return type of method overloading can be differ.	3) the return type of method must be same.
4)JVM decides which method will be call depending Upon the signature of the method	4) JVM decides which method is call depending upon the class of the object which is use to call the method.

methodOverride.java

```
class Animal
      {
                int height=10;
                 int weight=20;

                void talk()
                          {
                              System.out.println("\"Animal talking\"");
                           }

                void food()
                           {
                               System.out.println("\"Animal Eating\"");
                          }
       }

class Cat extends Animal
      {
                 void talk()
                           {
                              System.out.println("\"meo... meo\"");
                           }

                 void food()
                            {
                              System.out.println("\"Drink MILK\"");
                           }

       }

class methodOverride
      {
```

```
                public static void main(String[] args)
                        {
                           Animal a = new Animal();
                            Cat c = new Cat();

                           /* c.height=20 */;


                            c.talk();
                            c.food();


                             System.out.println("\"Height \""+c.height);


                        }
        }
```

POLYMORPHISM

Polymorphism is made up of two Greek words poly means “many” and morphos means “form”. In java if two or more than two methods have same name ,same return type and same access specifier but their tasks are different are called method overriding. If you are using method overriding concept in inheritance then inherited class will only access its own method and it will not call the method of the base class. To solve this problem polymorphism provides a solution that is given below:-

- At first you have to create an object of base class.
- After calling the overrided method of the base class, create an object of inherited class and pass the reference of the object of inherited class to base class object.
- Then call the overrided method of child class.

Points:-

1. In polymorphism base class holds the reference of child class or inherited class object.
2. It provides the solution of method overriding where as method overriding exist in inheritance.

```
import javax.swing.*;
class poly1
          {
                 public void display()
                    {
                              JOptionPane.showMessageDialog(null,"Base class is called...");
                    }
          }
class poly2 extends poly1
          {
                 public void display()
                    {
                              JOptionPane.showMessageDialog(null,"Child class is called...");
                    }
          }
class call_poly
          {
                public static void main(String args[])
                    {
                              poly1 bobj=new poly1();
                              bobj.display();
                              bobj=new poly2();
                              bobj.display();
                    }
          }
```

POLYMORPHISM WITH VARIABLES

```
          float x=7.25;
          int y=int(x);
```

the actual data type of variable x is changed by using cast operator. Here x is a float type variable ,it is converted into integer type to assign the value to another integer variable, this means ‘x’ exists in two forms, the first is float and second Is integer. This process also comes under the polymorphism.

STATIC BINDING/STATIC POLYMORPHISM

In static polymorphism, the compiler knows and binds the method call to the method body/method code at the time of compilation. Static method or final method is the example of static polymorphism.

POLYMORPHISM WITH STATIC METHOD

A static method is a method where single copy is available is the memory and that is shared by all the objects of the class. static method related to the class, not to the object.

There are following properties of static method:-

- Static method can be overloaded or overrided.
- Static method,static overloaded method and static overrided method donot depend upon the object of class.

```
import javax.swing.*;
class poly1
        {
                public static void display()
                  {
                          JOptionPane.showMessageDialog(null,"Base class is called...");
                   }
        }
class poly2 extends poly1
        {
                public static void display()
                   {
                          JOptionPane.showMessageDialog(null,"Child class is called...");
```

```
                        }
          }
class call_poly
          {
                    public static void main(String args[])
                        {
                              poly1 bobj=new poly1();
                              bobj.display();
                              poly2 cobj=new poly2();
                              cobj.display();
                              bobj=cobj;
                              bobj.display();
                        }
          }
```

Note:-

- Static method belongs to the class not to the object
- In the polymorphism if you are passing the reference of child class or derived class to the base class then because of static method, base class of object will always call its own static method not to the method of defined class.
- You can access the static method through an object of class or directly through the class name.

DYNAMIC BINDING /DYNAMIC POLYMORPHISM

When a method is called, the method call is bound to the method body at the time of running the program dynamically. Java compiler does not know which method body will be call at the time of compilation only JVM knows which method will be call at runtime of program. Dynamic polymorphism is also called runtime polymorphism.

```
import javax.swing.*;
class base
          {
                    public void display()
                        {
```

```
                                        JOptionPane.showMessageDialog(null,"Base class is called...");
                        }
            }
class child extends base
            {
                        public void display()
                        {
                                        JOptionPane.showMessageDialog(null,"Child class is called...");
                        }
            }
class call_poly
            {
                        public static void main(String args[])
                        {
                                        base bobj=new base();
                                        bobj.display();
                                        child cobj=new child();
                                        cobj.display();
                                        bobj=cobj;
                                        bobj.display();
                        }
            }
```

POLYMORPHISM WITH FINAL METHOD

Method which is declared with final reserved keyword is called final method. If you are using final reserve keyword with the method then the method will not override by the inherited class.Due to this reason only method overloading is allowed.

```
class base
            {
                        public final void display()
                        {
```

```
                                        System.out.println(“Base class is called…”);
                              }
                    }
class child extends base
                    {
                         public void display()
                              {
                                        System.out.println(“Child class is called.....”);
                              }
                    }
class call
                    {
                         public static void main(String args[])
                              {
                                        base bobj=new base();
                                        bobj.display();
                                        bobj=new child();
                                        bobj.display();
                              }
                    }
```

***//Above program will not execute and display compile time error*.**

ABSTRACT CLASS AND ABSTRACT METHOD

An abstract method is a method which has no any body has been defined. It is basically used, when the same method has to perform different task. i.e depends upon the object calling it.

An abstract class is a class that contains abstract method. There are following properties of the abstract class:-

1. Since it is incomplete class, so we cannot create an object of abstract class.
2. Abstract method of the abstract class must be implemented in the sub class or derived class.
3. Abstract class can contain both concrete method and abstract method.
4. Abstract class must be declared by the reserved keyword abstract and abstract method must be declared by the reserved keyword abstract.
5. Abstract class can be inherited by another class.
6. It can contain instance variable.
7. An abstract class can inherit another abstract class.
8. Donot use private access specifier with an abstract method.

INTERFACE

A class that contains only abstract method and there is no any concrete method is called an interface.

There are following feature of interface:-

1. It is not possible to create an object of interface.
2. All the member of interface should be public
3. An interface can inherit another interface.
4. But an interface cannot implement another interface.

5. If a class implements the interface then interface forces that class to implement all the methods of an interface compulsory.
6. An interface can have variables which are public and final by default. This means all the variables in the interface are public.
7. Multiple inheritances are supported by inheritance.

Q.WHAT ARE DIFFERENC BETWEEN ABSTRACT CLASS AND INTERFACE?

ABSTRACT CLASS	***INTERFACE***
1. An abstract class defines with abstract word, reserve keyword	1. An interface will be defined by the reserve interface.
2. A class will access all the abstract method and avoid concrete method. .	2. A class will access all the abstract class and abstract method of interface
3. In abstract class, concrete method or abstract method will be used.	3. In interface there will be only abstract method and no concrete method will be used.
4. Abstract class can contain instance variable.	4. It contains only constant value.

EXAMPLES OF INTERFACE

inter.java

```
package p1;

interface inter
   {
           public void accept(String name,int ino,int qty,int rate);
            public void display();
   }
```

dep_rec_tab.java

```
package p1;
import java.io.*;

public  class dep_rec_tab implements inter
{
       protected int ino,qty,rate,stock;
       protected String name;
       public void accept(String name,int ino,int qty,int rate)
              {
                 this.ino=ino;
                 this.qty=qty;
                 this.rate=rate;
                this.name=name;
              }
       public void display()
              {
                 System.out.println("Item name          :: "+name);

                System.out.println("Item Code          :: "+ino);

                System.out.println("Rate               :: "+rate);
```

```
                    System.out.println("Quantity            :: "+qty);
                }
     }
```

dep_tran_rec.java

```
package p1;
import p1.dep_rec_tab;

public class dep_tran_rec extends dep_rec_tab
     {
                protected int qty_sold,tot_price;

               public dep_tran_rec()
                    {
                            qty_sold=0;
                            tot_price=0;
                    }

               public void accept(String nme,int ino,int qty,int rate)
                     {
                            super.accept(nme,ino,qty,rate);
                             this.qty_sold=456;

                             tot_price=qty_sold*rate;
                      }
               public void display()
                    {
                             super.display();
                             System.out.println("Quantity sold       :: "+qty_sold);
```

```
                              System.out.println("Total Price          :: "+tot_price);
                    }


     }
```

call.java

```
import p1.dep_tran_rec;

import java.io.*;

class call
{
       public static void main(String arg[])throws IOException
                {
                    dep_tran_rec obj=new dep_tran_rec();

                     obj.accept("aaa",500,111,25);

                    obj.display();
                }
}
```

PACKAGE

A package is a directory which is used to hold collection of classes. With the help of package we can arrange all the class files into single location. There is very easy process to send all the class files into the package and you can easily implement the package in your program.

e.g.:- **java.io.*,** here java is the package or directory which contains another

Sub package or subdirectory i.e. “io”.

There are following properties of the package:-

1. Package is used to arrange collection of classes.
2. Since package store all the classes in a separate directory so accidental deletion will not take place.
3. a package can contains more than one package and group of package are called library and all the classes of package are called books of library. Books of library can be used several times and we know that classes of package are called books of library so we can also use classes of package several times.
4. Package makes the program very easy and with the help of package, you can create jar file easily.

THERE ARE TWO TYPES OF PACKAGE ARE AVAILBALE IN JAVA:-

i) **USER DEFINED PACKAGE**:-

As the name suggests user will create own package and perform operation on the package with the help of following steps:-

A) At first you have to use a reserved keyword i.e. “package” with package name. This package will be declared at the top of class file.

B) Now you have to create a class:-

addition.java

```
package p1;
public class addition
{
        private int x,y;
        public void accept(int x,int y)
              {
                    this.x=x;
                    this.y=y;
              }
```

```
                public void display()
                        {
                                System.out.println("Summation ::=="+(x+y));
                        }
        }
```

C) Now compile the program by using the following command **"java –d . addition.java"**.

"-d":- this option is responsible to find the package name and create the package with the help of package name. Here p1 is the package name that will be created by this option. After creating the package it will transfer the class file such as addition.class to the package p1.

".":- this option describes the current path of java file where package is going to create.

"c:\admin\addition.java"

Here current path of java file is **"c:\admin"** where package p1 will be created.

D) Now you have to create an object of class which is available in the package. There are two Ways to create an object of the class.

FIRST WAY:-

```
import p1.additon;
class call_additon
{
        public static void main(String args[])
                {
                        addition objA=new addition();
                        objA.accept(10,20);
                        objA.display();
                }
}
```

SECOND WAY:-

```
class call_additon
{
        public static void main(String args[])
                {
                        addition objA=new p1.addtion();
                        objA.accept(10,20);
                        objA.display();
                }
}
```

E) Now compile and run the program:-

```
javac call_addition.java
java call_additon
```

NOTE:-

i) All the classes of package must be declared with public access specifier because only public class will be accessible from outside of the package.

ii) If the class file is declare with public the file name must be same as that of class name.

iii) Because of public class, we cannot create more than one public class in a single java file.

ii) PRE BUILD PACKAGE IN JAVA:-

There are collections of pre build package in java, which helps the programmers to create graphical user interface environments, making connectivity with database and perform keyboard operation or mouse operation on the control.

There are following pre build packages are available in java:-

1. java.io.* :-

Here "io" stands for input and output whenever you use the input and output operation in java then you need to take help the classes of pre build package i.e.; "java.io.*".

2. java.awt.* :-

here "awt" stands for abstract window toolkit. This package helps to user to create GUI environments with image. It has collection of classes and also package that helps to user to create GUI structure.

There is a very important that is "java.awt.event.*" with the help of this package the programmer takes the action on the component such as clicking on the button,focusing on textbox and performing action on radio button,checkbox etc.

3. javax.swing.* : -

here "x" stands for extended and it extends the class from another package that is "java.awt" not only it extends the classes of another package but also it has own set of classes which provides facility to user to create GUI environments.

4. java.sql.* :-

sql stands for structural query language. This is very important package to create connectivity with database like oracle or MS-Access. Using this packge you insert the record into the table,update the record or delete the record from table.

5. java.applet.* :-

applets are very tiny program which runs from server to clint and get executed on client machine through network.

6. java.lang.* :-

“lang” stands for language. It contain wrapper classes and string method to perform operation on the string. Wrapper classes are used for convert primitive data type into class.

Basics about Strings in Java

1.You can create Strings in various ways:-

a) ***By Creating a String Object***

```
String s=new String("abcdef");
```

b) ***By just creating object and then referring to string***

```
String a=new String();
a="abcdef";
```

c) ***By simply creating a reference variable***

```
String a="abcdef";
```

2.All the strings gets collected in a special memory are for Strings called " String constant pool".

3. JVM does all string related tasks to avoid the memory wastage for more info on this refer "How JVM Handles strings".

4. Every String is considered as a string literal.

5.Strings are immutable only reference changes string never changes. New string literals are referenced when there is any manipulation. The old string gets lost in the preceding. look below

```
String s="abcd";

s=s+"efgh";
```

Important String Methods

Following are most commonly used methods in the String class.

1. public String concat(String s)

This method returns a string with the value of string passed in to the method appended to the end of String which used to invoke the method.

```
String s="abcdefg";
System.out.println(s.concat("hijlk"));
```

Note:-
Always use assignment operator in case of concat operator otherwise concat will be unreferenced and you will get old String. example

```
s.concat("hijkl");
System.out.println(s);
```

It will present output as " abcdefg " different than what we have expected. So always be careful in using the assignment operator in String method calls.
2.public charAt(int index)

This method returns a specific character located at the String's specific index.Remember,String indexes are zero based. Example:-

```
String s="Alfanso Mango";
System.out.println(s.charAt(0));
```

The output is 'A'

3.public int length()

This method returns the length of the String used to invoke the method. example:-

```
String s="name";
System.out.println(s.length());
```

The output is 4

4.public String replace(char old,char new)

This method return a String whose value is that of the String to invoke the method ,updated so that any occurrence of the char in the first argument is replaced by the char in the second argument. Example:-

```
String s="VaVavavav";
System.out.println(s.replace('v','V'));
```

The output is VaVaVaVaV

5.public boolean equalsIgnoreCase(String s)

This method returns a boolean value depending on whether the value of the string in the argument is the same as the String used to invoke the method.This method will return true even when character in the string object being compared have different cases. Example:-

```
String s="Vaibhav";
System.out.println(s.equalsIgnoreCase("VAIBHAV"));
```

The output is true

6.public String substring(int begin) / public String substring(int begin index,int end index)

substring method is used to return a part or substring of the String used to invoke the method. The first argument represents the starting location of the substring. Remember the indexes are zero based. example:-

```
String s="abcdefghi";
System.out.println(s.substring(5));
System.out.println(s.substring(5,8));
```

The output would be

```
" fghi "
" fg "
```

7.public String toLowerCase()

This method returns a string whose value is the String used to invoke the method, but with any uppercase converted to lowercase. Example:-

```
String s="AbcdefghiJ";
System.out.println(s.toLowerCase());
```

Output is " abcdefghij "

8.public String trim()

This method returns a String whose value is the String used to invoke the method ,but with any leading or trailing blank spaces removed. Example:-

```
String s="hey here is the blank space ";
System.out.println(s.trim())
```

The output is " heyhereistheblankspace"

9.public String toUpperCase()

This method returns a String whose value is String used to invoke the method ,but with any lowercase character converted to uppercase. Example:-

```
String s="AAAAbbbbb";
System.out.println(s.to UpperCase());
```

The output is " AAAABBBBB "

Above mentioned String methods are most commonly used in String class. Do remember them or otherwise I am always up for you and of course you can refer this site every time you need something.