**DECISION MAKING AND BRANCHING**

A Java program is a set of statements, which are normally executed sequentially in the order in which they appear. This happens when options or repetitions of certain calculations are not necessary. We have a number of situations, where we may have to change the order of execution of statements based on certain conditions, or repeat of group of statements until certain specified conditions are met.

When a program breaks the sequential flow and jumps to another part of the code, it is called branching. When the branching is based on particular condition, it is known as conditional branching. If branching takes place without any decision, it is known as unconditional branching.

**Control or Decision making statements:**

1. If statement
2. If..else statement
3. Nesting of if.. else
4. Else if ladder
5. Switch statement
6. Conditional operator

**Simple IF:** The general form of the simple if statement is

The ‘statement - block’ may be a single statement or a group of statements. If the test expression is true, the ‘statement – block’ will be executed, otherwise the ‘statement – block’ will be skipped and the execution will jump to ‘statement –x’.

|  |  |
| --- | --- |
|  | **Example:**  import java.util.Scanner;  class Example\_If  {  public static void main(String args[])  {  int x;  Scanner sc=new Scanner(System.in);  System.out.print(" Enter the value of x less than 10 ");  x=sc.nextInt();  if(x<10)  {  System.out.print(" Entered value less than 10 ");  }  System.out.println("ok");  } } |

**If-else statement:** The general form is

If the test expression is true, then the true-block statement(s) immediately following the if statement are execute. Otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both.

|  |  |
| --- | --- |
|  | **Example:**  import java.util.Scanner;  class Example\_If\_Else  {  public static void main(String args[])  {  int x;  Scanner sc=new Scanner(System.in);  System.out.print(" Enter the value of x less than 10 ");  x=sc.nextInt();  if(x<10)  {  System.out.print(" Entered value less than 10 ");  }  else  {  System.out.println(" Entered value greater than 10 ");  } } } |

**Nesting of if....else statement:** The general form of the nested if-else statement is

If the test condition1 is false statement -3 will be executed, otherwise it continues to perform the second test. If the condition – 2 true, the statement -1 will be evaluated, otherwise the statement -2 will be evaluated and the control is transferred to the statement – x.

|  |  |
| --- | --- |
|  | **Example:**  class IfElseNesting  {  public static void main(String args[])  {  int a=325, b=712, c=478;  System.out.println(" largest value is : ");  if(a>b)  {  if(a>c) {  System.out.println(a);  }  else {  System.out.println(c);  }  }  else {  if(c>b)  {  System.out.println(c);  }  else  {  System.out.println(b);  } } } } |

**Else if ladder:** The general form is

This construct is known as the else if ladder. The conditions are evaluated from top downwards. As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the ‘statement-x’ (skipping the rest of the ladder). When all the n conditions become false, then the final else containing the default-statement will be executed.

|  |  |
| --- | --- |
|  | **Example:**  import java.util.Scanner;  class Else\_If\_Ladder  {  public static void main(String args[])  {  int mark;  Scanner sc=new Scanner(System.in);  System.out.println(" Enter the mark ");  mark=sc.nextInt();  if(mark >= 75)  {  System.out.println(" Distinction ");  }  else if(mark >=60 && mark <75)  {  System.out.println(" First Class ");  }  else if (mark >=50 && mark <60)  {  System.out.println(" Second Class ");  }  else if(mark >=35 && mark <50)  {  System.out.println( " Third class");  }  else  {  System.out.println(" Fail ");  } } } |

**Switch Statement:**

The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.

The expression is an integer expression or characters. Value-1, value-2... are constants or constant expressions and are known as case labels.

Block-1, block-2 ... are statement lists and may contain zero or more statements. It is important to note that case labels end with a colon (:).

 The break statement at the end of each block signal the end of a particular case and causes an exit from the switch statement. The default is an optional case.

**Flowchart**

The general form is

**Example:**

**Java program to print week day using the switch statement.**

**import** java.util.Scanner;

**class** switchExample {

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

**int** dayno;

Scanner sc=**new** Scanner(System.*in*);

System.*out*.println(" Enter day number in a week \n");

dayno=sc.nextInt();

**switch**(dayno)

{

**case** 1: System.*out*.println( " sunday \n");

**break**;

**case** 2: System.*out*.println( " monday \n");

**break**;

**case** 3: System.*out*.println( " tuesday \n");

**break**;

**case** 4: System.*out*.println( " wednesday \n");

**break**;

Output:

Enter day number in a week

5

thrusday

**case** 5: System.*out*.println( " thrusday \n");

**break**;

**case** 6: System.*out*.println( " friday \n");

**break**;

**case** 7: System.*out*.println( " saturday \n");

**break**;

**default**:

System.*out*.println(" invalud day number");

}

}

}

**DECISION MAKING AND LOOPING:**

The process of repeatedly executing a block of statements is known as Looping.

The programme uses a loop construct to instruct the computer to perform repetitive tasks for a finite number of times based on conditions. Each loop construct contains two parts namely condition part and body part. The body part of the loop will be executed repeatedly as long as the given condition evaluates to true.

**Looping operations:**

1. while Loop
2. do ..while Loop
3. for Loop



**The while statement:** The general form is

The while is an entry-controlled loop statement. The simplest of all the looping structure in java is the while statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process is repeated until the test condition becomes false and the control is transferred out of the loop.

**Example:**

class While\_Example

{

public static void main(String args[])

{

int i=1;

while(i<=10)

{

System.out.println( i);

i++;

}

}

}

****

**The do..while Statement:** The general form is

do..while loop is an exit-controlled loop statement as it allows the body of the loop to be executed for the first time without any condition.

At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

**Example:**

class Do\_While

{

public static void main(String args[])

{

int i=1;

do

{

System.out.println(i);

i++;

}

while(i<=10);

}

}

**The for statement:** The general form is



It is a entry-controlled loop.

Execution of the **for** statement is as follows:

1. Initialization of the control variable is done first.
2. The value of the control variable is tested using the test condition. The test condition is a relational expression. When the loop is true, the body of the loop is executed. Otherwise the loop is terminated.
3. When the body of the loop is executed, the control is transferred back to the **for** statement after evaluating the last statement in the loop. Now the control variable is incremented or decremented. The new value of the control variable is tested again to see whether it satisfies the loop condition. If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

**Example:**

class for\_Example

{

public static void main(String args[])

{

int i;

for(i=1;i<=10;i++)

{

System.out.println( i);

}

}

}

**Difference b/w while and do-while:**

|  |  |
| --- | --- |
| **While** | **Do -while** |
| It is an entry-control loop | it is an exit-controlled loop |
| Test condition is tested first | Test condition is tested after executing the body of the loop |
| It is pre-testing loop | It is post-testing loop |
| It will not give guarantee to execute the body of the loop minimum once | It will give the guarantee to execute the body of the loop minimum once |
| Syntax: while (condition)  Simple or compound statement; | Syntax: do  Simple (or) compound statement;  while(condition); |

**Break Statement:**

The break statement is used to terminate the loop. When the **break** statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is, the break will exit only a single loop.

**Syntax:** **break;**

Terminating the loop with break statement

**Example:**

class Break

{

public static void main(String args[])

{

int i;

for(i=1;i<10;i++)

{

if(i==7)

break;

System.out.print(i+" ");

}

}

}

**Continue statement:**

Like break statement, java supports another similar statement called the continue statement. However, unlike the break which caused the loop to be terminated, the continue, as the name implies, causes the loop to be continued with the next iteration after skipping any statements in between. The continue statement tells the compiler. “SKIP THE FOLLOWING STATEMENTS AND CONTIUNE WITH THE NEXT ITERATION”. The format of the continue statement is simply

**Continue;**

**Example:**

import java.util.Scanner;

class positive

{

public static void main(String args[])

**output:**

enter any number 5

enter any number -7

enter any number 6

sum is 11

{

int i,n,sum=0;

Scanner sc=new Scanner(System.in);

for(i=1;i<=3;i++)

{

System.out.print("\n enter any number");

n=sc.nextInt();

if(n<0)

continue;

else

sum=sum+n;

}

System.out.print("\n sum is "+sum);

}

}

**Difference b/w break and continue:**

|  |  |
| --- | --- |
| **Break** | **Continue** |
| Takes the control to the outside the loop | Takes the control to the beginning of the loop |
| Used in switch statement | Used only in loop statements |
| Associated with if condition in loops | Associated with if condition |
| Terminates the loop when break is encountered | Doesn’t terminates the loop when continue is encountered. |

**ARRAYS**

An array is a homogeneous collection of elements of same type that share a common name. The elements of the array are stored in consecutive memory locations and are referred by an index (also known as subscript). In java, arrays are created on dynamic memory, i.e., allocated at runtime by JVM.

**Array types:**

1. One Dimensional Arrays
2. Two dimensional Arrays
3. Multi Dimensional Arrays

**ONE DIMENSIONAL ARRAY:**

When array is declared with only one dimension (subscript) then it is called one dimensional array or single dimensional array.

**Declaration of one-dimensional array:**

**Syntax:**

**Type array\_name[size];**

**Type[size] array\_name;**

1. The type specifies the type of element that will be contained in the array, such as int, float, or char.
2. Size indicates the maximum number of elements that can be stored inside the array.
3. Array\_name is an identifier that specifies the name of the array.

Datatype[ ] array\_name=new datatype[size]

new operator is used to initialize an array.

**Ex**:

**int arr[5];**

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

a[0] a[1] a[2] a[3] a[4]

**Initialization of arrays:**

After an array is declared, its elements must be initialized. Otherwise, they will contain garbage value.

Arrays can be initialized as follows:

**Syntax:**

Datatype[ ] array\_name={ elements separated by commas};

**Ex:**

int[] arr=new int[5];

int[] arr={1,2,3,4,5};

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |

arr[0] arr[1] arr[2] arr[3] arr[4]

**Array Length**

To know the length or size of an array we use a property named **length**.

Example: int x=arr.length;

**Program to display the elements of an array**

class array1

{

public static void main(String args[])

{

int arr[]={1,2,3,4,5,6};

int x=arr.length;

System.out.println(" The array elements are ");

for(int i=0;i<x;i++)

System.out.println(arr[i]);

}

}

**Sorting an array**

class sortarray1

{

public static void main(String args[])

{

int arr[]={5,8,62,42,85,1};

int x=arr.length;

int i,j;

System.out.println(" The array elements are ");

for(i=0;i<x;i++)

{

for(j=i+1;j<x;j++)

{

if(arr[i]>arr[j])

{

int temp;

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

} } }

System.out.println(" After sorting elements are ");

for(i=0;i<x;i++)

System.out.println(arr[i]);

} }

**TWO-DIMENSIONAL ARRAY**

When an array uses only two subscripts then it is called “Two dimensional array”. It can be viewed as table of elements, which contains rows and columns. A Two dimensional array is useful for matrix operations.

**Example:**

int table[2][3];

In the above example the table contains 2 rows and 3 columns, which mean, the table, can store 6 values.

0 1 2

|  |  |  |
| --- | --- | --- |
| 1  (0,0) | 2  (0,1) | 3  (0,2) |
| 6  (1,0) | 7  (1,1) | 8  (1,2) |

0

1

**Declaration of two-dimensional array:**

The general form is

**Syntax:**

Datatype arrayname[ ] [ ]= new datatype[row\_size ] [column\_size ];

**Example:**

int arr[][]=new int[3][4];

**Initialization Of Two-Dimensional Array**

The general form is

**Syntax:**

Datatype arrayname[][]={list of values};

**Example:**

int arr[3][3]={1,2,3,4,5,6,7,8,9};

int arr[][]={{1,2},{3,4},{5,6}};

|  |  |
| --- | --- |
| **2-D program to display matrix**  class two\_dim  {  public static void main(String args[])  {  int arr[][]={{1,2,3},{4,5,6}};  int i,j,m,n;  for(i=0;i<2;i++)  {  for(j=0;j<3;j++)  {  System.out.print(arr[i][j]+ " ");  }  System.out.println();  } } }  **Program to multiply two matrices**  class array\_mul  {  public static void main(String args[])  {  int a[][]={{1,2},{4,5}};  int b[][]={{1,2},{4,5}};  int c[][]=new int[2][2];  int i,j,k;  System.out.println(" The multiplication of two matrices are Shown below ");  System.out.println("---------------------------");  for(i=0;i<2;i++)  {  for(j=0;j<2;j++)  {  c[i][j]=0;  for(k=0;k<2;k++)  {  c[i][j]=c[i][j]+(a[i][k]\*b[k][j]);  }  System.out.print(c[i][j]+"\t");  } System.out.println(); } } } | **Program on Transpose matrix**  class transpose  {  public static void main(String args[])  {  int a[][]= {{1,2,3},{4,5,6},{7,8,9}};  int i,j;  System.out.println (" After Transpose ");  for(i=0;i<3;i++)  {  for(j=0;j<3;j++)  {  System.out.print(a[j][i]+" ");  }  System.out.println(" ");  }  }}  **Program to add two matrices**  class array\_addition  {  public static void main(String args[])  {  int a[][]={{1,2,3,4},{5,6,7,8},{6,9,2,5}};  int b[][]={{2,5,2,6},{6,5,7,2},{8,5,7,9}};  int c[][]=new int[3][4];  int i,j;  System.out.println(" The addition of two matrices are Shown below ");  System.out.println("---------------------------");  for(i=0;i<3;i++)  {  for(j=0;j<4;j++)  {  c[i][j]=a[i][j]+b[i][j];  System.out.print(c[i][j]+ "\t");  }  System.out.println();  } }} |

**3D array:**

Three-dimensional array is the collection

of [two-dimensional arrays](https://code4coding.com/two-dimension-array-in-java-language/) in Java programming language.

**Sum of a diagonal elements in a matrix**

class sum\_diagonal

{

public static void main(String args[])

{

int a[][]={{1,2,3},{5,6,7},{9,10,11};

int i,j,sum=0;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

{

if(i==j)

{

sum=sum+a[i][j];

} } }

System.out.print(" sum of diagonal "+sum); }

}

Three-dimensional array is also called the multidimensional

array.

**Syntax for 3D:**

datatype arrayname[ ][ ][ ]=new datatype[size1][size2][size3];

**Initializing a 3d array in Java**

**Method 1**

**Syntax**

array\_name[index\_1][index\_2][index\_3]=value;

**Example**

arr[0][0][0]=45;       //initialize first elements of 3 d array

we can initialize every index, like this

**Method 2**

int[][][] arr{ { {34,67,43}, {576,697,423},{576,697,423} },

{ {39,47,33},{376,987,453},{57,69,42}}

}

**Example:**

class thread

{

public static void main (String args[])

{

int arr[][][]={{{1,2,3},{4,5,6}},{{7,8,9},{2,4,6}}};

for(int i=0;i<2;i++)

{

for( int j=0;j<2;j++)

{

for(int k=0;k<3;k++)

{

System.out.printf("arr[%d][%d][%d]= %d ",i,j,k,arr[i][j][k]);

}

System.out.println();

}

System.out.println();

}}}

**STRINGS**

Strings represent a sequence of characters and these are enclosed with double quotes. In java, strings are class objects and implemented using three classes, namely String, StringBuffer and StringBuilder. String is a class in java.lang (language) package. But in java, all classes are also considered as data types. So we can take String as a datatype.

**The most direct way to create a string is to write:**

String greeting=”Hello World”;

Strings may be declared and created as follows:

**String stringname;**

**stringname=new String(“String”);**

**Example:**

String First\_name;

First\_name=new String(“ cnu “);

These two statements are combined as follows:

String First\_name=new String(“ cnu “);

The String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.

**STRING METHODS:**

The string class defines a number of methods that allows us to accomplish a variety of string manipulation tasks.

|  |  |
| --- | --- |
| **METHOD CALL** | **TASK PERFORMED** |
| s2=s1.toLowerCase(); | Converts the string s1 to all lowercase. |
| s2=s1.toUpperCase(); | Converts the string s1 to all uppercase. |
| s2=s1.replace(‘x’,’y’); | Replace all appearances of x with y |
| s2=s1.trim(); | Remove white spaces at the beginning and end of the string s1. |
| s1.equals(s2) | Returns true if s1 is equal to s2 |
| s1.equalsIgnoreCase(s2) | Returns true if s1=s2 ignoring the case of character |
| s1.length() | Gives the length of s1 |
| s1.charAt(n) | Gives nth character of s1 |
| s1.compareTo(s2) | Returns negative if s1<s2, positive if s1>s2, and zero if s1 is equal to s2 |
| s1.compareToIgnoreCase(s2) | Equals compareTo(), ignore the case characters |
| s1.concat(s2) | Concatenates s1 and s2 |
| s1.substring(n) | Gives substring starting from nth  character |
| s1.substring(n,m) | Gives substring starting from nth character up to mth (not including mth) |
| s1.indexof(‘x’) | Gives the position of the first occurrence of ‘x’ in the string s1 |
| s1.lastIndexOf('x') | Gives the last occurrence of 'x' in the string s1. |
| s1.startWith("sub") | Returns true, if a string beginning with the "sub " |
| s1.endsWith("sub ") | Returns true, if a string ending with the "sub" |

**Example:**

**Output:**

F:\java\_programs>javac str\_example.java

F:\java\_programs>java str\_example

The uppercase string ASHWIKA

The lowercase string nitiksha

Concating two stings NITIKSHAreddy

The length of a string 7

comparison of strings 19

string equals false

string equalignorecase false

charat w

substring TIKSHA

class str\_example

{

public static void main(String args[])

{

String s1="ashwika";

String s2="NITIKSHA";

String s3="reddy";

System.out.println(" The uppercase string " +s1.toUpperCase());

System.out.println(" The lowercase string " +s2.toLowerCase());

System.out.println(" Concating two stings " +s2.concat(s3));

System.out.println(" The length of a string " +s1.length());

System.out.println(" comparison of strings "+ s1.compareTo(s2));

System.out.println(" string equals "+ s1.equals(s2));

System.out.println(" string equalignorecase "+ s1.equalsIgnoreCase(s2));

System.out.println(" charat "+ s1.charAt(3));

System.out.println(" substring "+ s2.substring(2));

}

}

**STRING COMPARISON**

Different ways to compare String in Java:

1. **By Using equals() Method:** In java, String equals( ) method compares the two given strings based on the data / content. If all the contents of both the string are same then it returns true. If all characters are not matched then it returns false.

**Example:**

class StringEquals

{

public static void main(String args[])

{

String s1="Hello";

String s2="Hello";

String s3=new String("hello");

System.out.println(s1.equals(s2));

System.out.println(s1.equals(s3));

} }

1. **By using equalsIgnoreCase( ) method:**

The string equalsIgnoreCase( ) method compares two string irrespective of the case of the string. This method returns true if the argument is not null and the contents of both the strings are same ignoring case, else false.

class StringEqualsIgnoreCase

{

public static void main(String args[])

{

String s1="Hello";

String s2="Hello";

String s3=new String("hello");

System.out.println(s1.equalsIgnoreCase(s2));

System.out.println(s1.equalsIgnoreCase(s3));

} }

1. **By Using == Operator:**

The == operator compares references not values.

class DoubleEquals

{

public static void main(String args[])

{

String s1="Hello";

String s2="Hello";

String s3=new String("Hello");

System.out.println(s1==s2);

System.out.println(s1==s3);

} }

1. **By compareTo() Method**:

The java compares two strings based on the Unicode value of each character in the strings. If the compareTo() method returns negative if s1s2, and zero if s1=s2.

**Example:**

class StringCompareTo

{

public static void main(String args[])

{

String s1="Hello";

String s2="Hello";

String s3=new String("hello");

System.out.println(s1.compareTo(s2));

System.out.println(s1.compareTo(s3));

}

}

**IMMUTABILITY OF STRINGS IN JAVA**

Mutable objects are those objects whose contents can be modified. Immutable objects are those objects, once created cannot be modified. String class objects are immutable.

**Example:**

class Test

{

public static void main(String args[])

{

String s1="Data";

String s2="Base";

s1=s1+s2;

System.out.println(s1);

}

}

Here, JVM creates two objects, s1 and s2 separately. When s1+s2 is done, JVM creates a new object and stores the string DataBase in that object. But it does not modify the contents of the string s1. After creating the new object, the reference s1 is adjusted to refer to that new object. Observer that the contents of the string s1 are not modified, so the Strings are called immutable.

**CLASS, OBJECTS AND METHODS**

**DEFINING A CLASS:**

A class is a user-defined data type with a template that serves to define its properties. A class encloses both the data and functions that operate on the data, into a single unit. The variables are called data members and functions are called member functions in a class. The basic form of a class definition is:

***class <class name> extends <Super class> implements <interface name>***

***{***

***[fields declaration]***

***[methods declaration]***

*}*

The class name followed by the keyword ***class*** is the actual class which we want to define.

The keyword ***extends*** is used to extend the features of an existing class to the current class. We can use only one super class to extend at a time.

The keyword ***implements*** is used to implement the interfaces if any. We can use more than one interface to implement in the current class.

**FIELDS DECLARATION:**

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. We can declare the instance variables exactly the same way as we declare local variables. **Example**

class Rectangle

{

int length;

int width;

}

**METHODS DECLARATION:**

A class with only data fields has no life. The objects created by such a class cannot respond to any messages. We must therefore add methods that are necessary for manipulating the data contained in the class. Methods are declared inside the body of the class but immediately after the declaration of instance variables. The general form of a method declaration is

class Rectangle

{

int length;

int width;

void getdata (int x,int y)

{

length=x;

width=y;

}

int rectarea()

{

int area=length\*width;

return (area);

}

}

**type methodname [parameter list]**

**{**

**Method-body;**

**}**

Method declarations have four basic parts:

1. The name of the method (method name)
2. The type of the value the method returns (type)
3. A list of parameters (parameter-list)
4. The body of the method

**Examples:**

class Rectangle

{

int length;

int width;

void getdata (int x,int y)

{

length=x;

width=y;

}

}

**CREATING OBJECT:**

An instance of a class is called object. (Or)

An object in java is essentially a block of memory that contains space to store all the instance variables.

Objects in java are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object. **Example**

**Rectangle rect1; //declare the object**

**rect1=new Rectangle(); //instantiate the object**

Both statements are combined into one as shown below

**Rectangle rect1=new Rectanlge();**

**ACCESSING CLASS MEMBERS:**

We can access the class members i.e. instance variables and methods in the following way

**Objectname.variablename=value;**

**Objectname.methodname(parameterlist);**

From the above statements, objectname is the name of the object, variablename is the name of the instance variable. Methodname is the method that we wish to call, and parameter-list is a comma separated list of actual values.

**Example:**

class Rectangle

{

int length; //DECLARATION OF VARIABLES

int width;

void getData(int x,int y) // DEFINITION OF METHOD

{

length=x;

width=y;

}

int rectarea()

{

int area=length\*width;

return(area);

}

}

class rectarea // CLASS WITH MAIN METHOD

{

public static void main(String args[])

{

int area1,area2;

Rectangle r=new Rectangle(); //CREATING OBJECTS

Rectangle r1=new Rectangle();

r.length=10; //ACCESSING VARIABLES

r.width=25;

area1=r.length\*r.width;

r1.getData(12,10); //ACCESSING METHODS

area2=r1.rectarea();

System.out.println("Area1 = " +area1);

System.out.println("Area2= "+area2);

}

}

**VISIBILITY CONTROL (ACCESS MODIFIERS OR ACCESS SPECIFIERS)**

**PRIVATE:** The data members and member functions that are declared as **private** can be accessed only within the class and not from the outside the class.

**PUBLIC:** The data members and member function that are declared as **public** can be accessed within the class as well as from outside the class.

**PROTECTED:** The data members and member functions declared as **protected** cannot be accessed outside the class, but can be accessed from a derived class. The protected keyword is frequently used in inheritance of class.

**CONSTRUCTOR:**

Java supports a special type of method, called a constructor that enables an object to initialize itself when it is created. Constructors have the same name as the class itself. Secondly, they do not specify any return type, not even void. Types of constructors are **default constructor and parameterized constructor.**

**Default constructor** is a constructor with no arguments or not parameter list.

**Parameterized constructor** is a constructor with arguments.

The below example show both Default Constructor and Parameterized Constructor

**Example:**

class Rectangle

{

int length;

int width;

Rectangle() // constructor with no arguments

{

System.out.println( “ Default constructor”);

}

Rectangle(int x, int y) //constructor with arguments

{

length=x;

width=y;

}

int rectarea()

{

return (length\*width);

}

}

class ex\_a

{

public static void main(String args[])

{

Rectangle r=new Rectangle(); //calling default constructor

Rectangle rect=new Rectangle(14,10); // calling parameterized constructor

int area1=rect.rectarea();

System.out.println(area1);

//System.out.println(rect.rectarea());

}

}

**INHERITANCE**

Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called **Inheritance**.

The existing class is known as the **base class** or **super class** or **parent class** and the new class is called the **subclass** or **derived class** or **child class**.

We use inheritance for 1) code reusability 2) method overriding

**Syntax of Java Inheritance**

**class subclass\_name extends superclass\_name**

**{**

**//methods and fields**

**}**

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

class Employee

{

float salary=40000;

void disp()

{

System.out.println(" hello world");

}

}

class Programmer extends Employee

{

int bonus=10000;

public static void main(String args[])

{

Programmer p=new Programmer();

p.disp();

System.out.println("Programmer salary is:"+p.salary);

System.out.println("Bonus of Programmer is:"+p.bonus);

}

}

**TYPES OF INHERITANCE**

**Single Inheritance:** Derivation of a class from only one base class is called Single inheritance.

class A

{

........

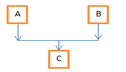
}

class B extends A

{

........

}

**Multiple Inheritance:** Derivation of a class from one or more base classes is called multiple inheritance. Java does not support multiple inheritance i.e. classes cannot have more than one super class. Java provides this concept implemented using interfaces.

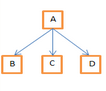
**Hierarchical Inheritance:** Derivation of several classes from one base class i.e. one class may be inherited by more than one class is called hierarchical inheritance.

class A

{

........

}

class B extends A

{

........

}

class C extends A

{

........

}

class D extends A

{

........

}

**Multilevel Inheritance:** Derivation of a class from another derived class is called Multilevel Inheritance.

class A

{

.......

}

class B extends A

{

.......

}

class C extends B

{

.......

}

**POLYMORPHISM**

**METHOD OVERLOADING**

Method overloading means, that we can create multiple methods with same name but with different parameters lists.

**Example:**

class overloading

{

void dis()

{

class room

{

int len,bred;

room(int x, int y)

{

len=x;

bred=y;

}

room(int x)

{

len=bred=x;

}

int area()

{

return(len\*bred);

}

}

class overloa

{

public static void main(String args[])

{

int a1,a2;

room r=new room(2,3);

room r1=new room(4);

System.out.println(r.area());

System.out.println(r1.area()); } }

System.out.println(" method overloading");

}

void dis(int x)

{

System.out.println(" x value is " +x);

}

}

class overload

{

public static void main(String args[])

{

overloading ovr=new overloading();

ovr.dis();

ovr.dis(2);

}

}

**ABSTRACT METHOD AND ABSTRACT CLASS**

An abstract method does not contain any body. It contains only the method header. So we can say it is an incomplete method. An abstract method must be redefined in a subclass, thus making overriding compulsory.

A class which is declared with the **abstract** keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

Since, abstract class contains incomplete methods. It is not possible to estimate the total memory required to create the objects. So, JVM cannot create objects to an abstract class. We should create sub classes and all the abstract methods should be implemented in the sub classes.

**Example:**

abstract class Bank{

abstract int getRateOfInterest();

}

class SBI extends Bank{

int getRateOfInterest(){return 7;}

}

class PNB extends Bank{

int getRateOfInterest(){return 8;}

}

class TestBank{

public static void main(String args[]){

Bank b=new SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

Bank b1=new PNB();

System.out.println("Rate of Interest is: "+b1.getRateOfInterest()+" %");

}}