Module 2:

- Arrays
- Strings
- Inheritance
- Polymorphism

Arrays in Java

Escape Sequences

- A character preceded by a backslash (\) is an escape sequence and has a special meaning to the compiler.

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

Introduction

- Consider the following declarations:

```
int a, b, c, d, e, f, g, h, i, j;
```

Issues:

1. Variables of the same **data**_type, i.e. ***int***.

2. Individual operation (e.g., input) is needed for each variable

Java Array

- Java provides a data structure, the **array**, which stores a **fixed-size sequential** collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a **collection of variables** of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

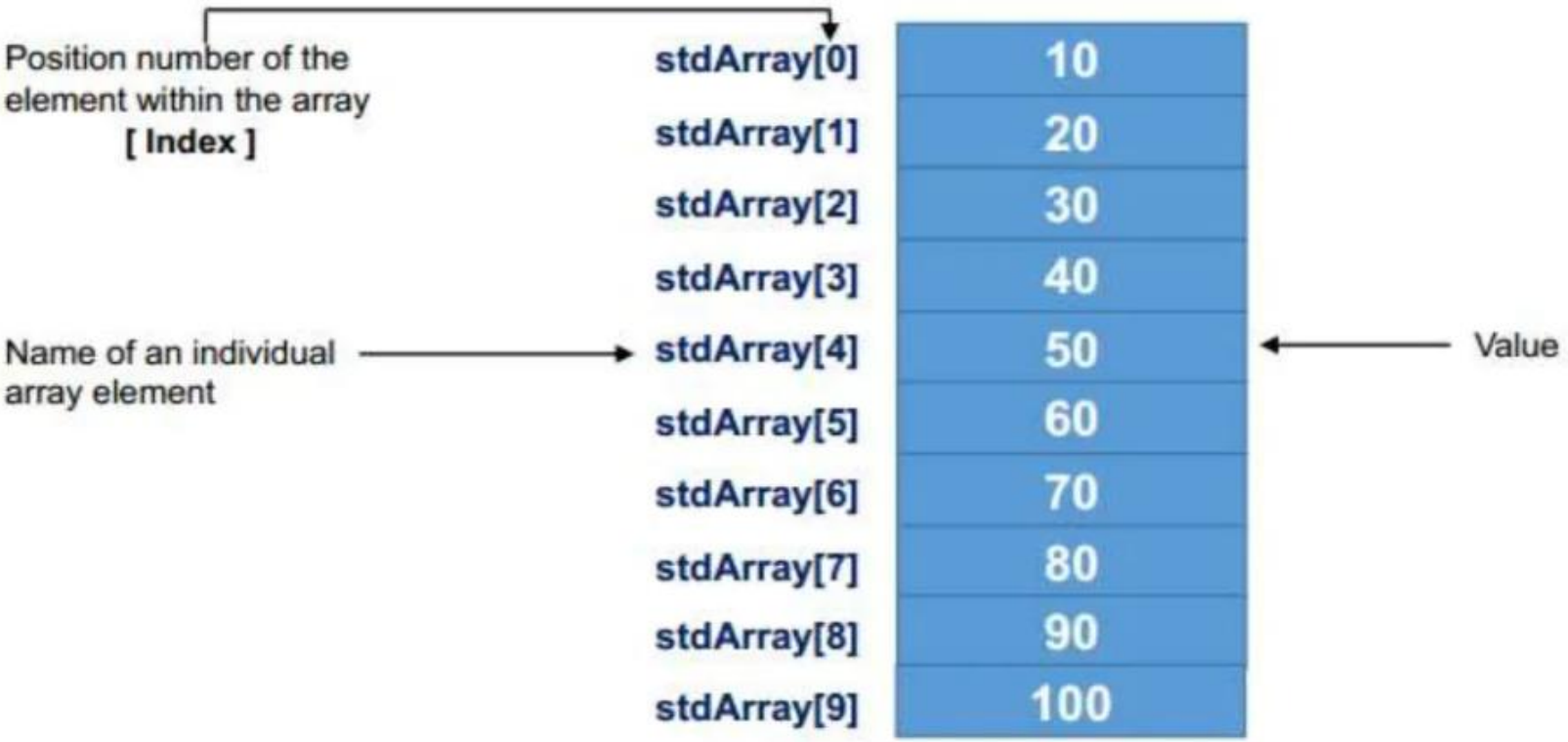
Java Array

- Arrays are **data structures** consisting of **related data** items of the **same type**.
- An array is a **consecutive** group of **memory locations**
- The memory locations in the array are known as **elements** of array.
- **Total number** of elements in the array is call the **length** of the array.

Java Array

- To refer to **a particular location** or element in the array, we **specify the name** of the array and the **position number** of the particular element in the array in square brackets ([]).
- The **position number** is more formally called a subscript or **index**.
- A **subscript** must be an **integer** or integer expression.

Java Array


Position number of the element within the array
[Index]
Name of an individual array element
stdArray[0]
stdArray[1]
stdArray[2]
stdArray[3]
stdArray[4]
stdArray[5]
stdArray[6]
stdArray[7]
stdArray[8]
stdArray[9]
10
20
30
40
50
60
70
80
90
100
Value

Advantages of Arrays

- Array can **store a large** number of **values** with **single name**.
- The **values** store in the array can **sorted easily**.
- The **search process** can be **applied** on array **easily**.
- It can be used to **implement** other **data structures** like linked lists, stacks, queues, trees, graphs etc.
- **2D arrays** are used to represent **matrices**

Type of Arrays

- There are two types of array.
  - Single Dimensional Array
  - Multidimensional Array

Single Dimensional Array in java

- **Syntax to Declare an Array in java**

```
dataType[] identifier; (or)     // preferred way.
dataType [] identifier; (or) // works but not preferred way.
dataType identifier [];      // works but not preferred way.
```

- **Examples**

```
int[] arr;        //Array of Integer type
char[] arr;     //Array of Character
short[] arr;  //Array of Short type
long[] arr;  //Array of long type
```

Creating Arrays

- **new** operator is used to initialize an array.
- **Syntax**

```
arrayRefVar = new dataType[arraySize];
```

Initialization of Array

- Assigning the values into Array, This process is known as initialization.
- This is done using the Array subscripts as show below:

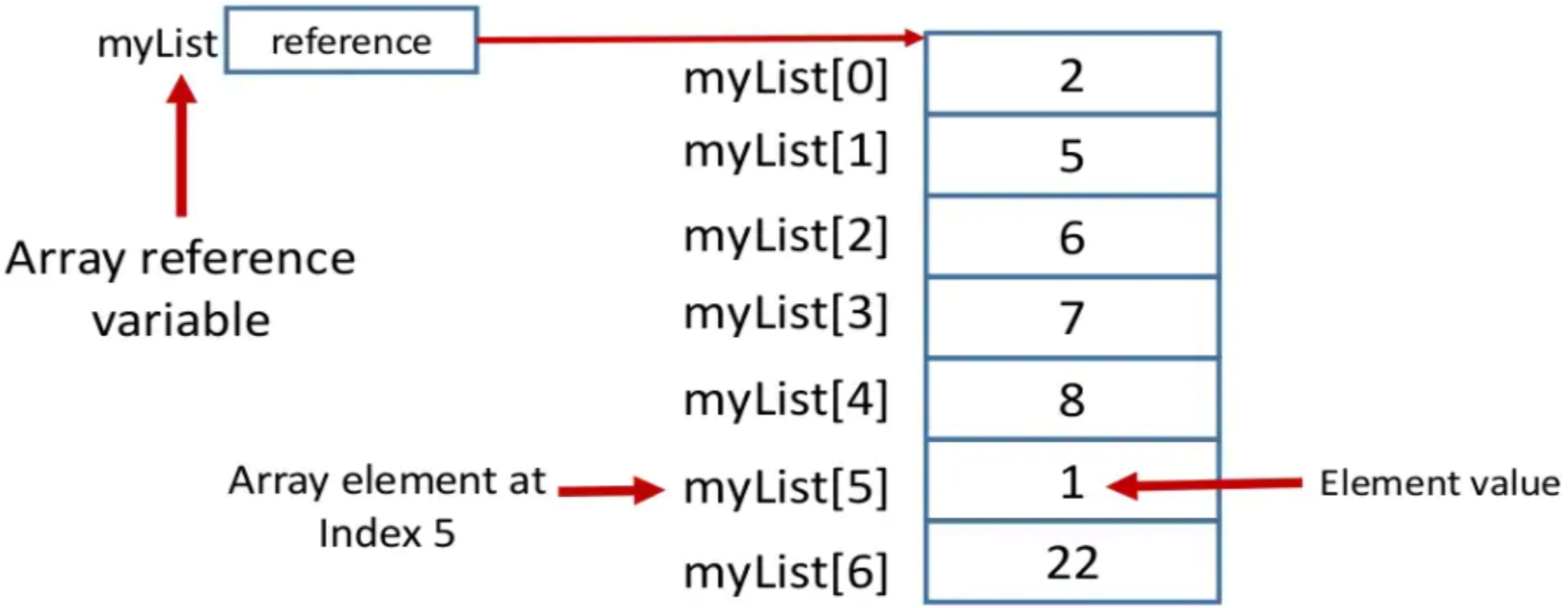
```
arrayname[subscript] = value;
```

- **Example:**

```
Number[0]=31;
Number[1]=34;
.....
Number[5]=20;
```

Initialization of Array

```
double[] myList = new double[]{2,5,6,7,8,1,22};
```



Declaration, Instantiation and Initialization of Java Array

- Syntax for default values:

```
int[] num = new int[5];
Or (less preferred)
int num[] = new int[5];
```

- Syntax with values initialization:

```
int[] num = {1,2,3,4,5};
Or (less preferred)
int num[] = {1, 2, 3, 4, 5};
```

Example of Single Dimensional ava Array

```
public class Test {
    public static void main(String[] args) {
        int a[]=new int[]{10,12,13,14};
        for(int i=0;i<a.length;i++)
        System.out.println(a[i]);
    }
}
```

Output is:

10
12
13
14

Both Generate
Same Output

```
System.out.println(a[0])
System.out.println(a[1])
System.out.println(a[2])
System.out.println(a[3])
```

Taking input from User Using Array

```
import java.util.Scanner;
public class Test {
   public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   double[] numbers = new double[3];
   System.out.println("Please Enter Three Numbers: ");
   for (int i = 0; i < numbers.length; i++){
      numbers[i] = input.nextDouble();
      }
    System.out.println("You Enter the Following Numbers: ");
   for (int i = 0; i < numbers.length; i++){
      System.out.println(i);
      }
   }
}
```

Creating an Array of size 3

Output of Previous Program

```
Please Enter Three Numbers:
22
33
44
You Enter the Following Numbers:
22.0
33.0
44.0
```

Problem Statement

- Write a Java program which calculates the sum of squares of 3 numbers and stored in an array.
- Note Talking input of three numbers from user.

Passing Arrays to Methods

- Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an **int** array –

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

- You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2 –
- Example
- printArray(new int[]{3, 1, 2, 6, 4, 2});

Passing Arrays to Methods

```
public class Example {
   public static void main(String[] args) {
      int[] a = {1,2,3};
      add(a);
      for(int x:a)
      System.out.println(x);
      }
   public static void add(int[] a)
   {
      for(int i=0;i<a.length;i++)
         a[i]=a[i]+4;
   }
}
```

Output is:

5
6
7

String Arrays

- **What are Strings:**
- The **collection** of the **alphabets** is called **string**.
- **For Example:-**
  - 1. “London”.
  - 2. “Austria”.
  - 3. “Football”.
  - 4. “Adil”
  - 5. “Tooth Paste”

String Arrays

- Strings may be declared and created as follows:

```
String[] stringName;
stringName = new String[] {“string”};
```

- **Example**

```
Sting[] firstName;
firstName = new String[] {“Adil”};
```

- These two Statements may be combined as follows:

```
String[] firstName = new String[] {“Adil”};
```

Declare and initialized String Array

```
String str[] = {"Adil"};
```

Equivalent

```
String str[] = new String[4];
    str[0]="A";
    str[1]="d";
    str[2]="i";
    str[3]="l";
```

Calling String Methods from an Array Element

```
public class StringArrayMethods
 {
 public static void main(String[] args)
 {
   // Create an array of Strings.
   String[] names = { "Adil", "Aslam", "Hina", "Ameen" };

   // Display each string in the names array
   // in uppercase.
   for (int index = 0; index < names.length; index++)
      System.out.println(names[index].toUpperCase());
  }
}
```

Output is:

ADIL
ASLAM
HINA
AMEEN

toUpperCase is a Method of String Class

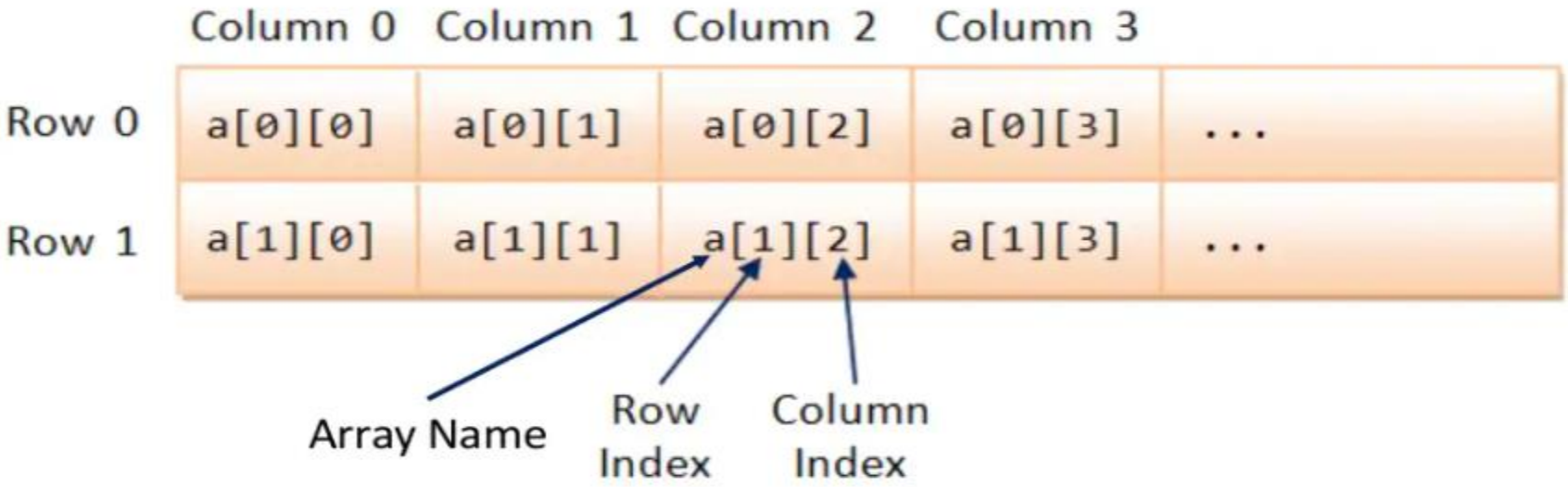
Arrays class methods

Method name	Description
`Arrays.binarySearch(`***array***`, `***value***`)`	returns index of value in a sorted array (< 0 if not found)
`Arrays.copyOf(`***array***`, `***length***`)`	returns a new copy of an array
`Arrays.equals(`***array1***`, `***array2***`)`	returns true if the two arrays contain same elements
`Arrays.fill(`***array***`, `***value***`)`	sets every element to the given value
`Arrays.sort(`***array***`)`	arranges the elements into sorted order
`Arrays.toString(`***array***`)`	returns a string for the array, such as "[10, 30, -25, 17]"

Multidimensional Array

- Arrays with two dimensions (i.e. subscripts) often represent tables of values consisting of information arranged in rows and columns.
- By convention, the first identifies the element's row and the second identifies the element's column
- Arrays that require two subscripts to identify a particular element are called two-dimensional arrays or 2-D arrays
- Arrays with two or more dimensions are known as multidimensional arrays

Two Dimensions Arrays



Syntax of two Dimensional Arrays

To declare a two-dimensional array, we simply list two sets of empty brackets, like this:

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

Example to Create two Dimensional Array in java

Create two Dimensional Array as:

```
//3 row and 3 column
int[][] arr = new int[3][3];

//3 row and 3 column
String[][] arr = new String[3]3];
```

Please Note that

- Like the one dimensional arrays, two dimensional arrays may me initialized by the following their declaration with list of initial values enclose in braces. For Example,

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

- Initializes the elements of the first row to zero and second row to one. The initialization is done row by row. The above statement can be equivalently written as :

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

- By surrounding the elements of each row by braces.
- We can also initialize a two dimensional array in the form of a matrix as shown below :

```
int arr[2][3] = {
                    {0,0,0} ,  //show one row
                    {1,1,1}
                },
```

Example of two Dimensional Array in java

- `int[][] arr = new int[2][3];`
- Above statement will create 2*3 element array of an integer values that can by accessed by arr.

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

- **Initializing Values**
- `int arr = {{1,2,3},{4,5,6}};`

```
int[][] wrong = new int[][]; // not OK, you must specify 1st dimension
 int[][] right = new int[2][];   // OK
```

Example of two Dimensional Array in java

```
public class Example {
   public static void main(String[] args) {
            int arr[][]= {{1,2},{3,4}};
      for (int i = 0; i < arr.length; i = i + 1)
      {
         for(int j=0; j < arr[i].length; j = j + 1)
         {
          System.out.print(arr[i][j]+"   ");
         }
         System.out.println();
      }

   }
}
```

```
System.out.println(arr[0][0]);
System.out.println(arr[0][1]);
System.out.println(arr[1][0]);
System.out.println(arr[1][1]);
```

Note the Output in Matrix form

Output is:

1 2
3 4

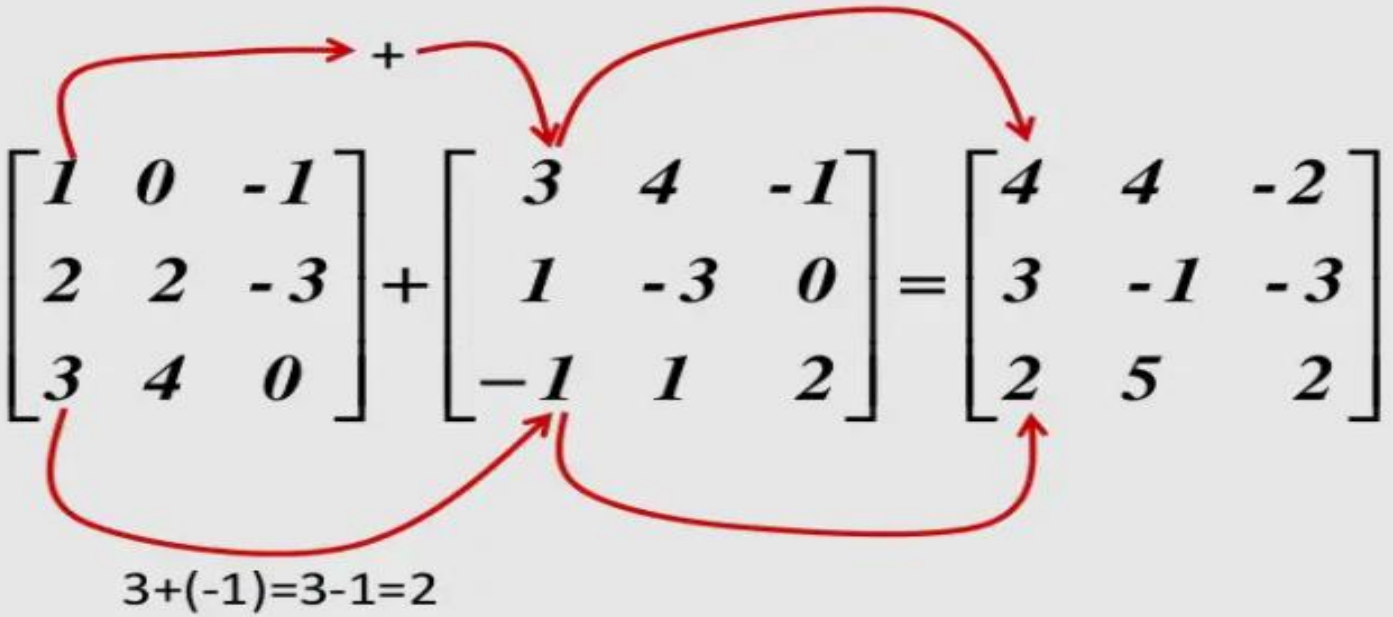
Another Example of 2D Array

```
public class Test {
   public static void main(String[] args) {
      String[][] salutation = {
         {"Mr. ", "Mrs. ", "Ms. "},
         {"Adil"}
      };
      // Mr. Adil
      System.out.println(salutation[0][0] + salutation[1][0]);
      // Mrs. Adil
      System.out.println(salutation[0][1] + salutation[1][0]);
   }
}
```

Output is:
Mr. Adil
Mrs. Adil

Addition of Two Matrices

- In mathematics addition of two matrices is:



Addition of 2 Matrices in java

```
public class Test {
   public static void main(String[] args) {
   int a[][]={{1,2,3},{4,5,6}};
   int b[][]={{1,2,3},{4,5,6}};
   //creating another array (matrix) to store the sum of two matrices
   int c[][]=new int[2][3];
   //adding and printing addition of 2 matrices
   for(int i=0;i<2;i++){
      for(int j=0;j<3;j++){
       c[i][j]=a[i][j]+b[i][j];
         System.out.print(c[i][j]+" ");
      }
      System.out.println();      //new line
      }
   }
}
```

Creating and Initializing two Array (Matrices)

Addition of two Arrays done here

Output is:

2 4 6

8 10 12

Matrix Multiplication Program-5

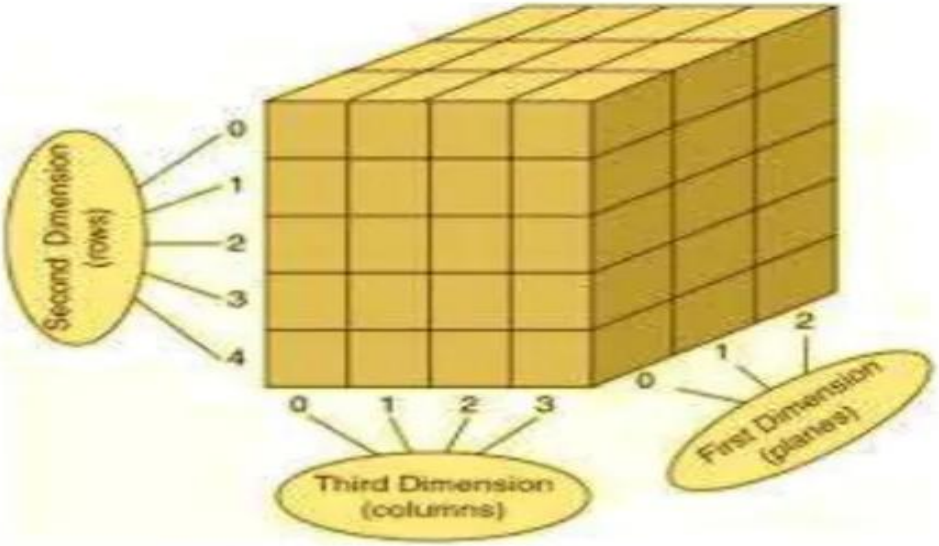
```
/* Matrix Multiplication Starts Here */

int sum = 0;
for(int i=0; i<r1; i++)
{
   for(int j=0; j<c2; j++)
   {
      for(int k=0; k<c1; k++)
      {
         sum = sum + A[i][k]*B[k][j];
      }
      C[i][j]=sum;
      sum=0;
   }
}
```

```
**************************

        Output

**************************

The 1st Matrix is
1       2
3       4
**************************

The 2nd Matrix is
1       2
3       4
**************************

The Result of Multiplication is
7       10
15      22
```

Three Dimensional Arrays in Java

- Declare Three dimensional Array

```
int arr[][][] = new int[2][4][3];
```



Three Dimensional Arrays in Java

- Initialization of three dimensional arrays

```
int[][][] threeDArray =
      { { {1,  2,  3}, { 4,  5,  6}, { 7,  8,  9} },
      { {10, 11, 12}, {13, 14, 15}, {16, 17, 18} },
      { {19, 20, 21}, {22, 23, 24}, {25, 26, 27} } };
```

String Class

Java – String Class and its Methods

- String is probably the most commonly used class in java library. String class is encapsulated under java.lang package. In java, every string that you create is actually an object of type String. One important thing to notice about string object is that string objects are immutable that means once a string object is created it cannot be altered.
- **What is an Immutable Object?**
  - An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper class's objects are immutable.

Java – String Class and its Methods

- **Creating a String**
- There are two ways to create a String in Java
  - String literal
  - Using new keyword

- **String literal**
- In java, Strings can be created like this: Assigning a String literal to a String instance:

```
String str1 = "Welcome";
String str2 = "Welcome";
```

Java – String Class and its Methods

- **Using New Keyword**
- As we saw above that when we tried to assign the same string object to two different literals, compiler only created one object and made both of the literals to point the same object. To overcome that approach we can create strings like this:

```
String str1 = new String("Welcome");
String str2 = new String("Welcome");
```

- In this case compiler would create two different object in memory having the same text.

• Concatenating String

- There are 2 Ways to concatenate two or more string.
  - Using concat() method
  - Using + operator

• 1) Using concat() method

```
String s = "Hello";
String str = "Java";
String str2 = s.concat(str);
String str1 = "Hello".concat("Java");    //works with string literals too.
```

• Using + Operator (Concatenation)

```
String str = "Adil";
String str1 = "Aslam";
String str2 = str + str1;   // or
String str3 = "Adil"+"Aslam";
```

- **String Comparison**
- String comparison can be done in 3 ways.
  - Using equals() method
  - Using == operator
  - By CompareTo() method
- **Using equals() method**
- equals() method compares two strings for equality. Its general syntax is.

```
String s = "Help";
String s1 = "Hello";
String s2 = "Hello";
s1.equals(s2);   //true
s.equals(s1) ;   //false
```

- **Using == operator**
- == operator compares two object references to check whether they refer to same instance. This also, will return true on successful match.

```
String s1 = "Java";
String s2 = "Java";
String s3 = new string ("Java");
test(s1 == s2)    //true
test(s1 == s3)    //false
```

- **By compareTo() method**
- compareTo() method compares values and returns an int which tells if the string compared is less than, equal to or greater than other string. Its general syntax is,

```
String s1 = "Adil";
String s2 = "Kashif";
String s3 = "Adil";
s1.compareTo(S2);    //return -1 because s1 < s2
s1.compareTo(S3);    //return 0 because s1 == s3
s2.compareTo(s1);    //return 1 because s2 > s1
```

Java – String Class and its Methods

- **String length() Method**
- This method returns the length of the string. The length is equal to the number of 16-bit Unicode characters in the string.
- **Syntax**

```
int length()
```

- **Parameters**
  - **NA**
- **Return Value**
  - This method returns the length of the sequence of characters represented by this object.

String length() Method Example

```
public class Example {

   public static void main(String[] args) {
     String str1 = " ";
     String str2 = "Welcome";
     String str3 = "My Name is Adil Aslam";

   System.out.println("Length of First String is: "+str1.length());
   System.out.println("Length of Second String is: "+str2.length());
   System.out.println("Length of Third String is: "+str2.length());
   }
}
```

Output is:

Length of First String is: 1
Length of Second String is: 7
Length of Third String is: 7

Java – String Class and its Methods

- **String toLowerCase() Method**
- toLowerCase() method returns string with all uppercase characters converted to lowercase
- **Syntax**

```
String toLowerCase())
```

```
public class Example {

   public static void main(String[] args) {
   String str = "My Name is Adil Aslam";
   System.out.println(str.toLowerCase());

   }
}
```

Output is:
my name is adil aslam

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013

- **String toUpperCase() Method**
- This method returns string with all lowercase character changed to uppercase.
- **Syntax**

```
String toUpperCase())
```

```
public class Example {

   public static void main(String[] args) {
   String str = "My Name is Adil Aslam";
   System.out.println(str.toUpperCase());

   }
}
```

Output is:
MY NAME IS ADIL ASLAM

- **String replace() Method**
- This method returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
- **Syntax**

```
String replace(char oldChar, char newChar)
```

String replace() Method Example

```
public class Example {

   public static void main(String[] args) {
   String str = "My Name is Adil Aslam";
   System.out.println(str.replace('A', 'a'));
   System.out.println(str.replace('N', 'I'));

   }
}
```

Note: Replace all A with a

Output is:

My Name is adil aslam
My Iame is Adil Aslam

- **String compareTo(String anotherString)**
- This method compares two strings lexicographically.
- **Syntax**

```
int compareTo(String anotherString)
```

String compareTo Method Example

```
public class Example {

   public static void main(String[] args) {
     String str1 = "Strings are immutable";
     String str2 = "Strings are immutable";
     String str3 = "Integers are not immutable";
     int result = str1.compareTo( str2 );
     System.out.println(result);
     result = str2.compareTo( str3 );
     System.out.println(result);
     result = str3.compareTo( str1 );
     System.out.println(result);
   }
}
```

Output is:

```
0
10
-10
```

- **String split() Method**
- This method has two variants and splits this string around matches of the given regular expression.
- **Syntax**

```
String split(String regex)
```

String split() Method Example

```
public class Example {

   public static void main(String[] args) {
    String str = "My _Name is _Adil Aslam";
    //splits the string based on "_"
    String[] words=str.split("_");
    //using java foreach loop to print elements of string array
    for(String w:words){
    System.out.println(w);
    }
   }
}
```

Output is:

```
My
Name is
Adil Aslam
```

- **String join() Method**
  - The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.
  - In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.
- **Syntax**

```
String join(CharSequence delimiter, CharSequence elements)
```

String join() Method Example

```
public class Example {

   public static void main(String[] args) {
    String joinString1=String.join
    ("-","Welcome","to","String");
    System.out.println(joinString1);
    String joinString2=String.join
    ("_","MY","Name","Adil","Aslam");
    System.out.println(joinString2);
   }
}
```

Output is:
Welcome-to-String
MY_Name_Adil_Aslam

Java StringBuffer class

StringBuffer class

- Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
- Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.
- String is a peer class of String. While String create strings of fixed length, StringBuffer create Strings of flexible length that can be modified in terms of both length and content. We can insert characters and substrings in the middle of strings, or appended another string to the end, etc.
- **What is Mutable String**
  - A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

StringBuffer class

- Important Constructors of StringBuffer class:

- **StringBuffer()**
  - creates an empty string buffer with the initial capacity of 16.
- **StringBuffer(String str)**
  - creates a string buffer with the specified string.
- **StringBuffer(int capacity)**
  - creates an empty string buffer with the specified capacity as length.
- **StringBuffer(CharSequence seq)**
  - This constructs a string buffer that contains the same characters as the specified CharSequence.

Difference Between String and StringBuffer

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

StringBuffer class-Methods

- **StringBuffer append() Method**
  - The **java.lang.StringBuffer.append(String str)** method appends the specified string to this character sequence.The characters of the String argument are appended, in order, increasing the length of this sequence by the length of the argument. If **str** is null, then the four characters "null" are appended.
- **Declaration**

```
StringBuffer append(String str)
```

StringBuffer append(String str) Method Example

```
public class Example {
   public static void main(String[] args) {
    StringBuffer sb=new StringBuffer("Hello ");
    //now original string is changed
    sb.append("Java ");
    System.out.println(sb);
    //now original string is changed
    sb.append("World");
    System.out.println(sb);
   }
}
```

Pass String Here

Output is:

Hello Java

Hello Java World

Java StringBuilder class

StringBuilder class

- StringBuilder is identical to StringBuffer except for one important difference it is not synchronized, which means it is not thread safe. Its because StringBuilder methods are not synchronized.
- Important Constructors of StringBuilder class:
- **StringBuilder()**
  - creates an empty string Builder with the initial capacity of 16.
- **StringBuilder(String str)**
  - creates a string Builder with the specified string.
- **StringBuilder(int length)**
  - creates an empty string Builder with the specified capacity as length.

Difference Between StringBuffer and StringBuilder

There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:

No.	StringBuffer	StringBuilder
1)	StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.

StringBuffer Example

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

      StringBuffer buffer=new
      StringBuffer("hello");
      buffer.append("java");
      System.out.println(buffer);
   }
}
```

Output is:
hellojava

StringBuilder Example

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

      StringBuilder builder=new
      StringBuilder("hello");
      builder.append("java");
      System.out.println(builder);
   }
}
```

Output is:
hellojava

Inheritance in Java

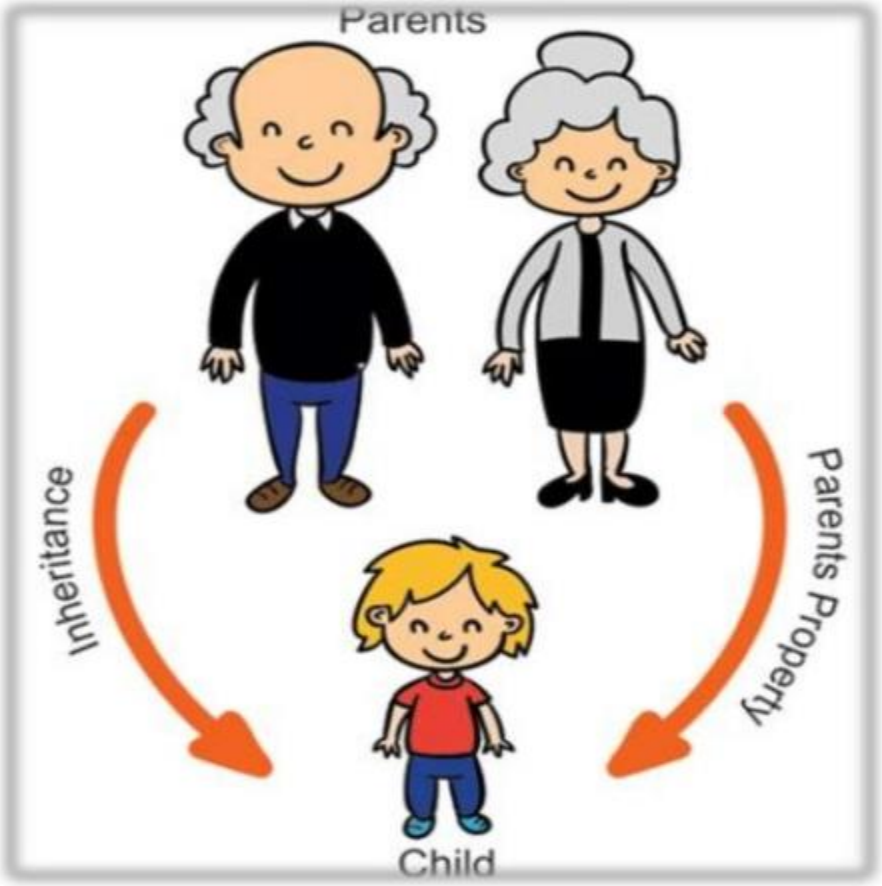
Inheritance in Java

- **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.
- When a Class extends another class it inherits all non-private members including fields and methods. Inheritance in Java can be best understood in terms of Parent and Child relationship, also known as Super class(Parent) and Sub class(child) in Java language.
- Inheritance defines is-a relationship between a Super class and its Sub class. extends and implements keywords are used to describe inheritance in Java.

Inheritance in Java

- Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.
- **Why use Inheritance ?**
- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usability

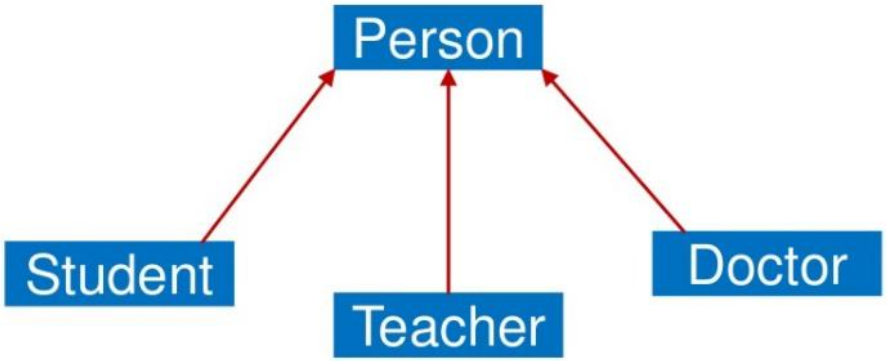
Inheritance in Java

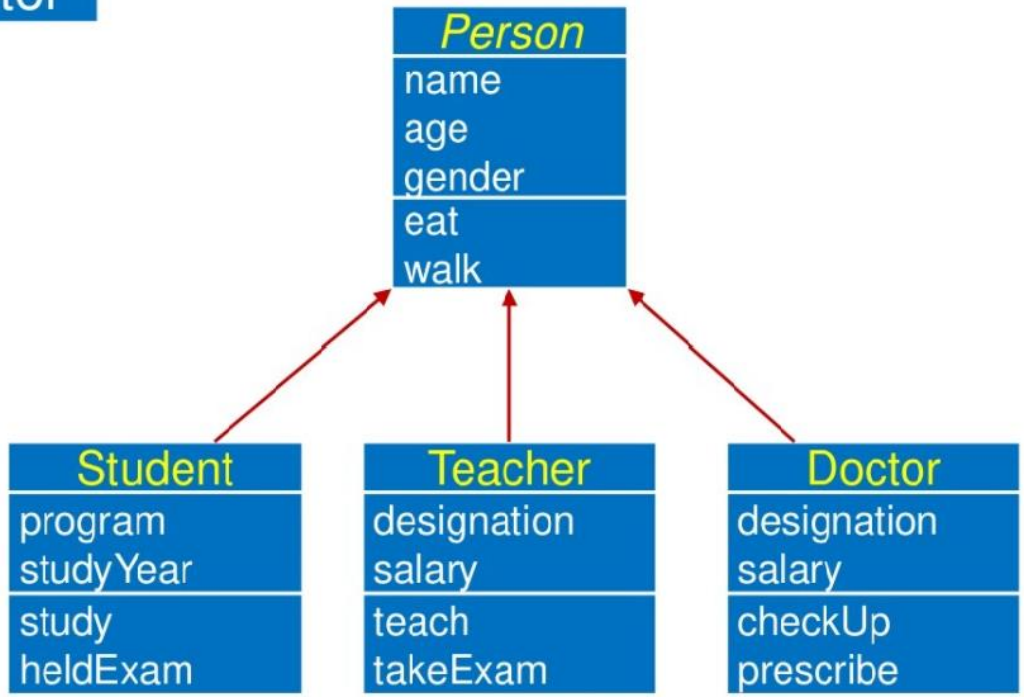


Inheritance in Java

No	Term	Definition
1	Inheritance	**Inheritance** is a process where one object acquires the properties of another object
2	Subclass	Class which inherits the properties of another object is called as **subclass**
3	Superclass	Class whose properties are inherited by subclass is called as **superclass**
4	Keywords Used	extends and implements

Example – Inheritance

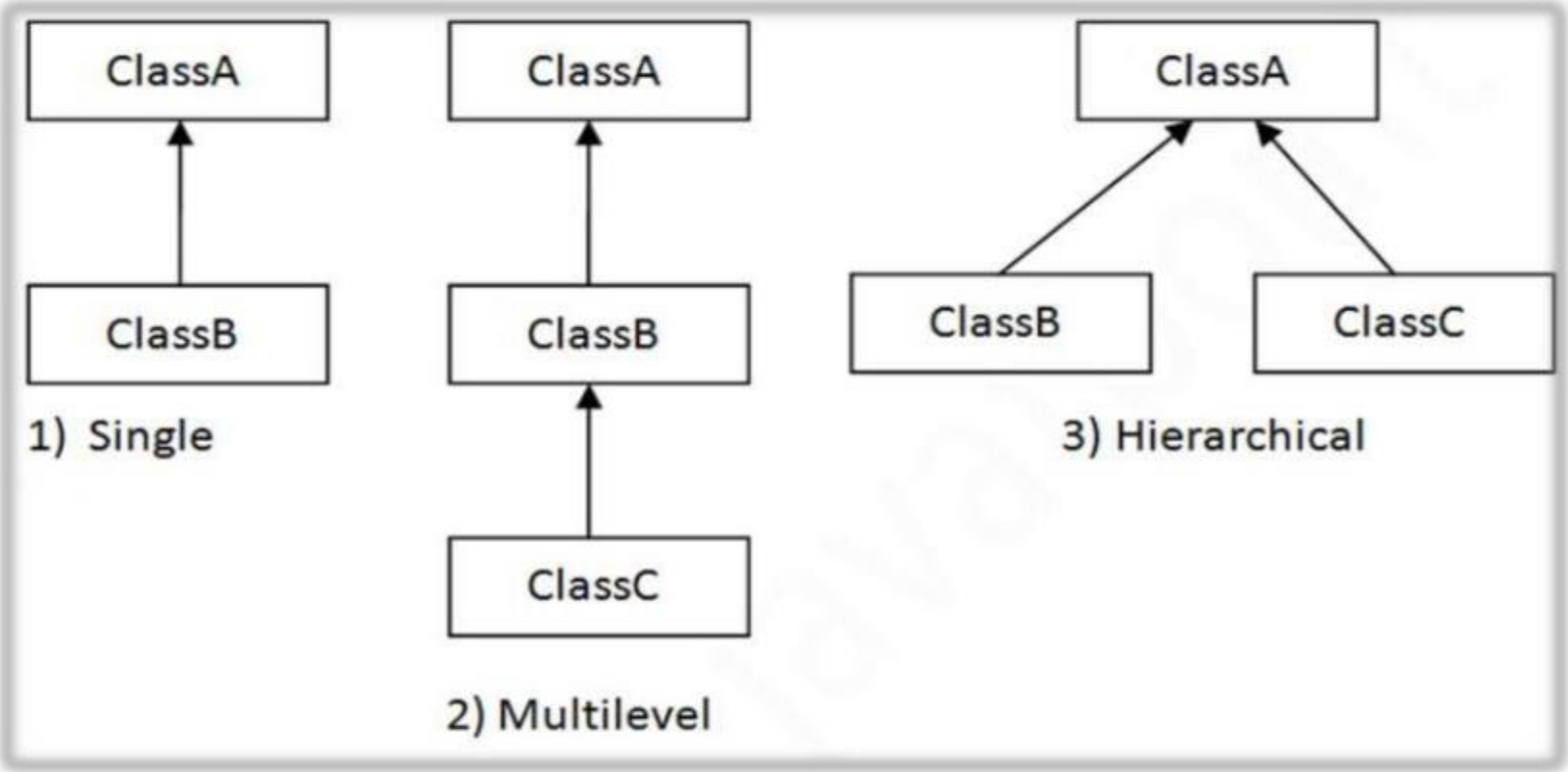




Inheritance in Java

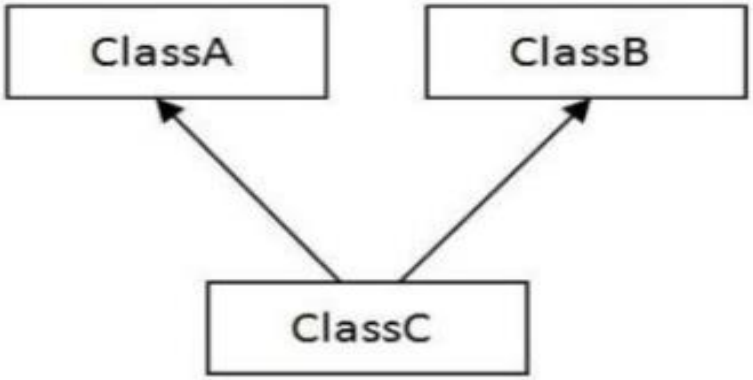
- **Inheritance in Java**
- Inheritance in Java is done using
  - **extends** – In case of Java class and abstract class
  - **implements** – In case of Java interface.
- **What is inherited**
  - In Java when a class is extended, sub-class inherits all the **public**, **protected** and **default (Only if the sub-class is located in the same package as the super class)** methods and fields of the super class.
- **What is not inherited**
  - **Private** fields and methods of the super class are not inherited by the sub-class and can't be accessed directly by the subclass.
  - Constructors of the super-class are not inherited. There is a concept of constructor chaining in Java which determines in what order constructors are called in case of inheritance.

Types of Inheritance

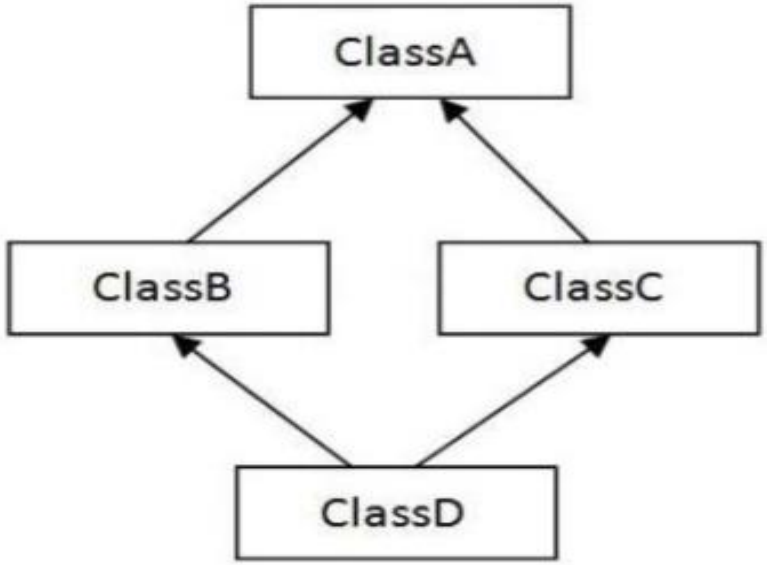


Types of Inheritance

- In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



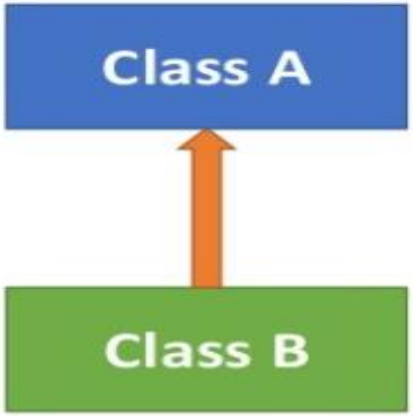
4) Multiple



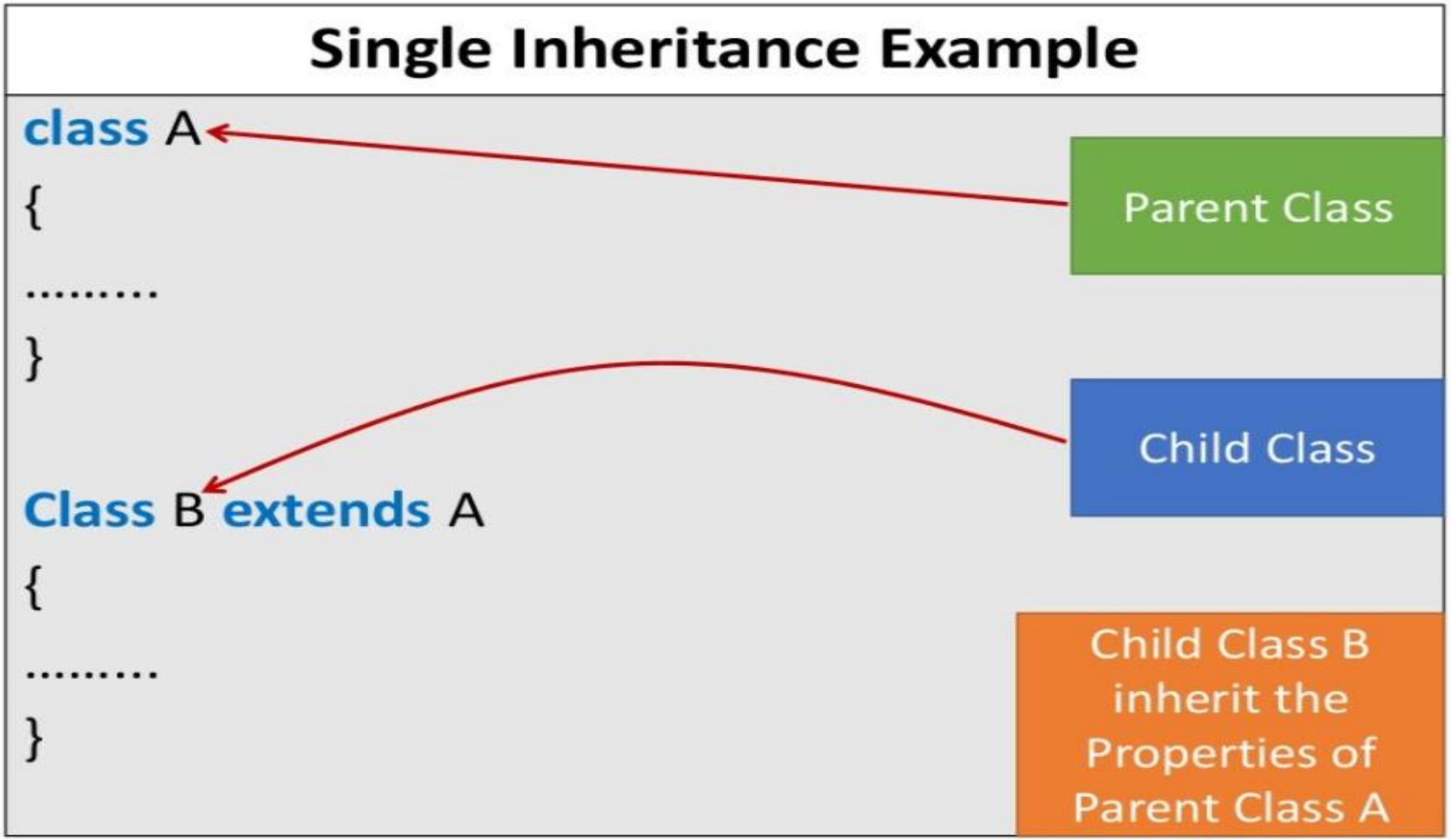
5) Hybrid

Types of Inheritance

- **1) Single Inheritance**
- **Single inheritance** is easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



Single Inheritance Example



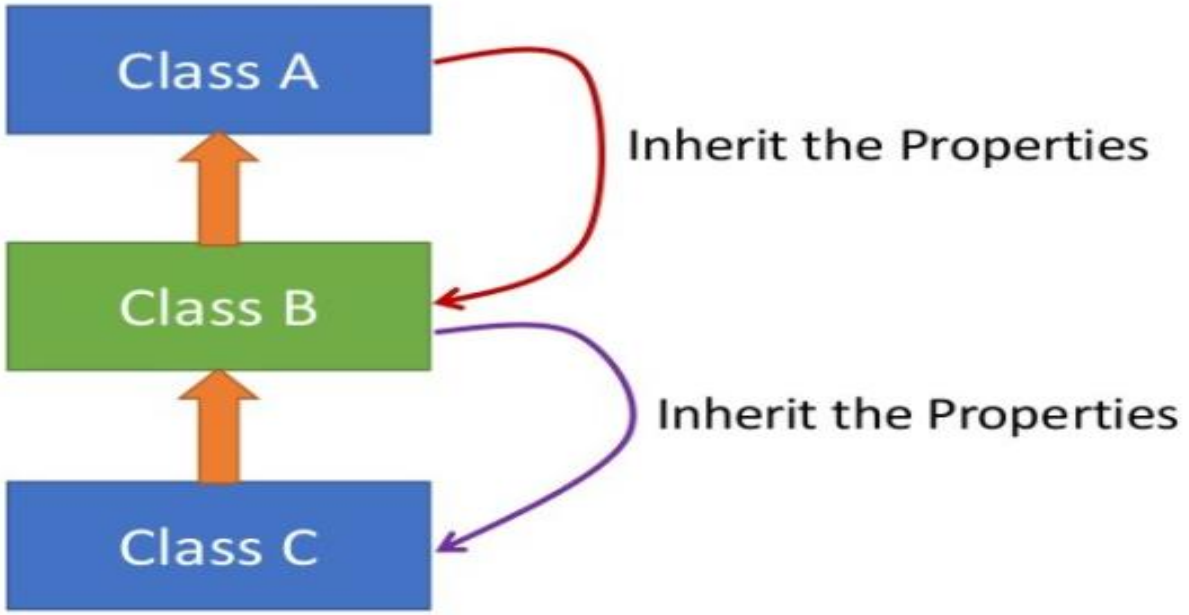
Single Inheritance Example

```
class A {
  int data=10;
}
class B extends A {
   public void display()
   {
     System.out.println("Data is:"+data);
   }
   public static void main(String args[])
   {
     B obj = new B();
     obj.display();
  }
}
```

Child Class B inherit/Access the data field of Parent Class A

Output is:
Data is:10

Multilevel Inheritance



- Here class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and method of class A along with B that's what is called multilevel inheritance.

Multilevel Inheritance

- Class C can inherit the Members of both Class A and B Show below :

C Contains B Which Contains A

Multilevel Inheritance Example

```
class A {
.....
}
Class B extends A {
.....
}
Class C extends B {
.....
}
```

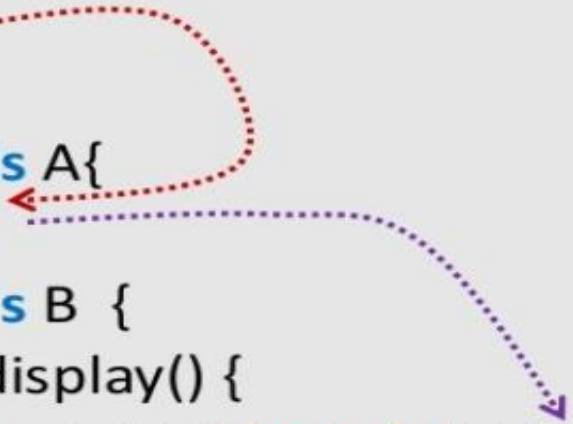
A is Parent Class of B

B is Child Class of A and Parent Class of C

C is Child Class of B

Multilevel Inheritance Example

```
class A {
  int data=10;
}
class B extends A{
}
class C extends B  {
   public void display() {
     System.out.println("Data is:"+data);
   }
 public static void main(String args[]) {
     C obj = new C();
     obj.display();
  }
}
```

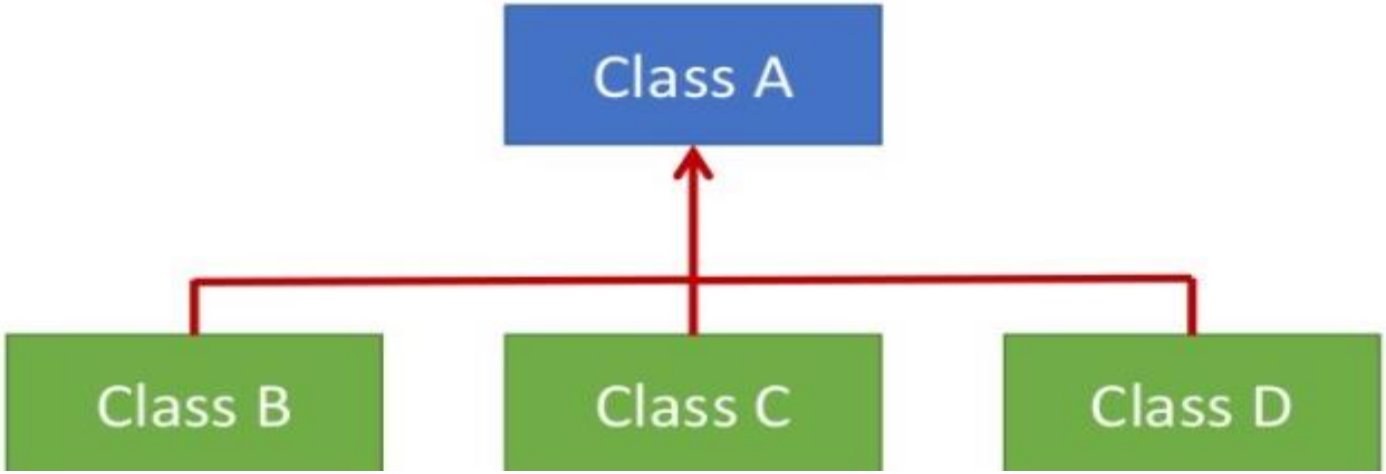


Here Class B inherit the properties of Class A and Class C inherit the properties of Class B

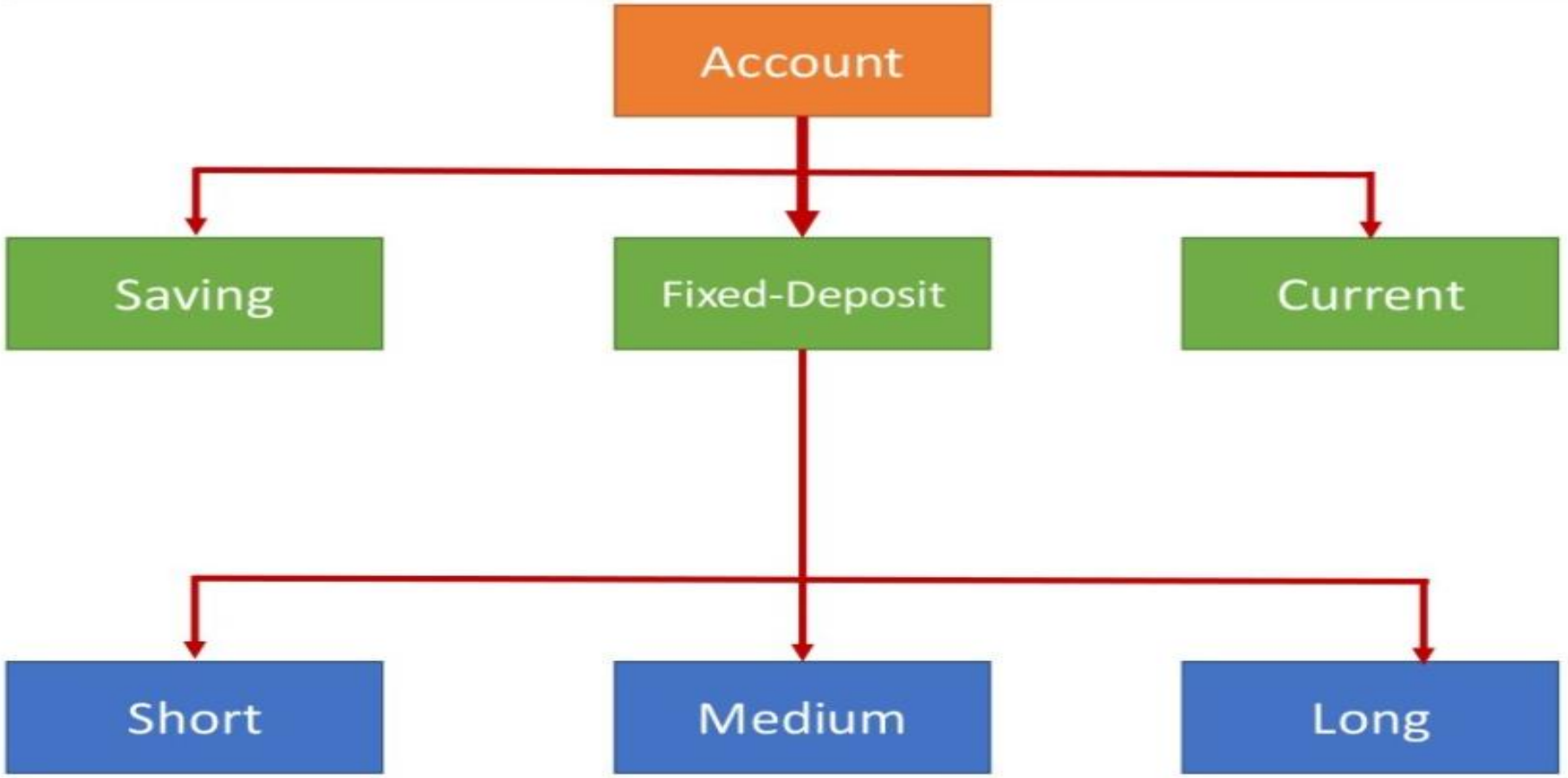
Output is:

Data is:10

- **3) Hierarchical Inheritance**
- In this **inheritance** multiple classes inherits from a **single** class i.e there is one super class and **multiple** sub classes. As we can see from the below diagram when a same class is having more than one sub class (or) more than one sub class has the same parent is called as **Hierarchical Inheritance**.



Hierarchical Inheritance(Real time Example)



Hierarchical Inheritance Example

```
class A {
.....
}
Class B extends A {
.....
}
Class C extends A {
.....
}
Class D extends A {
.....
}
```

A is a Parent Class of Class B, C and D

B is Child Class of Class A

C is Child Class of Class A

D is Child Class of Class A

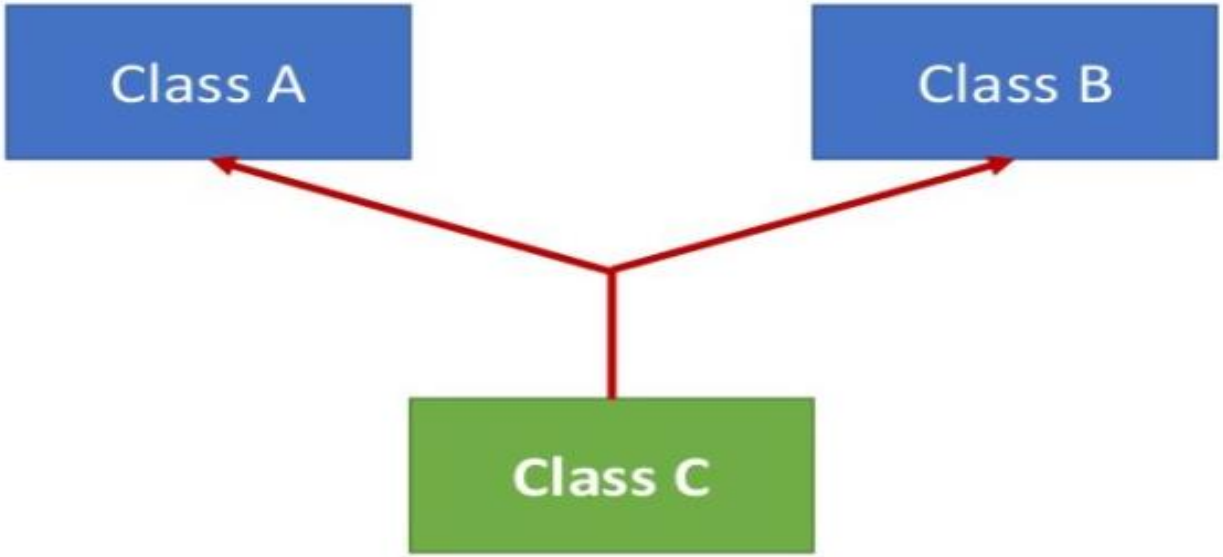
Hierarchical Inheritance Example

```
class A {
  int data=10;
}
class B extends A{
}
class C extends A  {
}
class D extends A  {
 public static void main(String args[]) {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
   System.out.println("Data in Class B is: "+obj1.data);
   System.out.println("Data in Class C is: "+obj2.data);
   System.out.println("Data in Class D is: "+obj3.data);
  }
}
```

Output is:

Data in Class B is: 10
Data in Class C is: 10
Data in Class D is: 10

Multiple Inheritance



Here Class A, B and C are three Classes. The C Class inherits A and B classes.in other words Class C inherit the properties of both Class A and Class B.

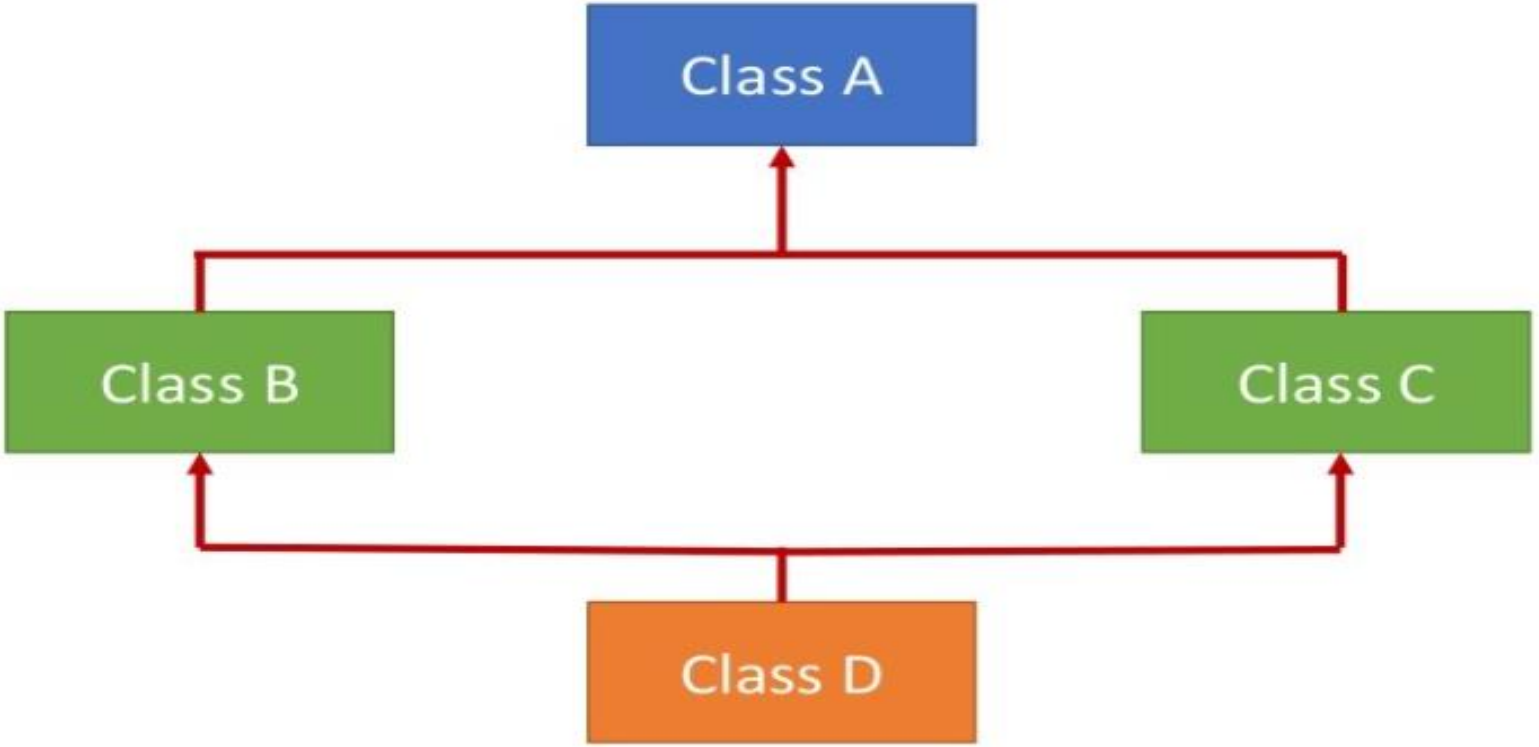
Multiple Inheritance

- **Why multiple inheritance is not supported in java?**
- Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.
- Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

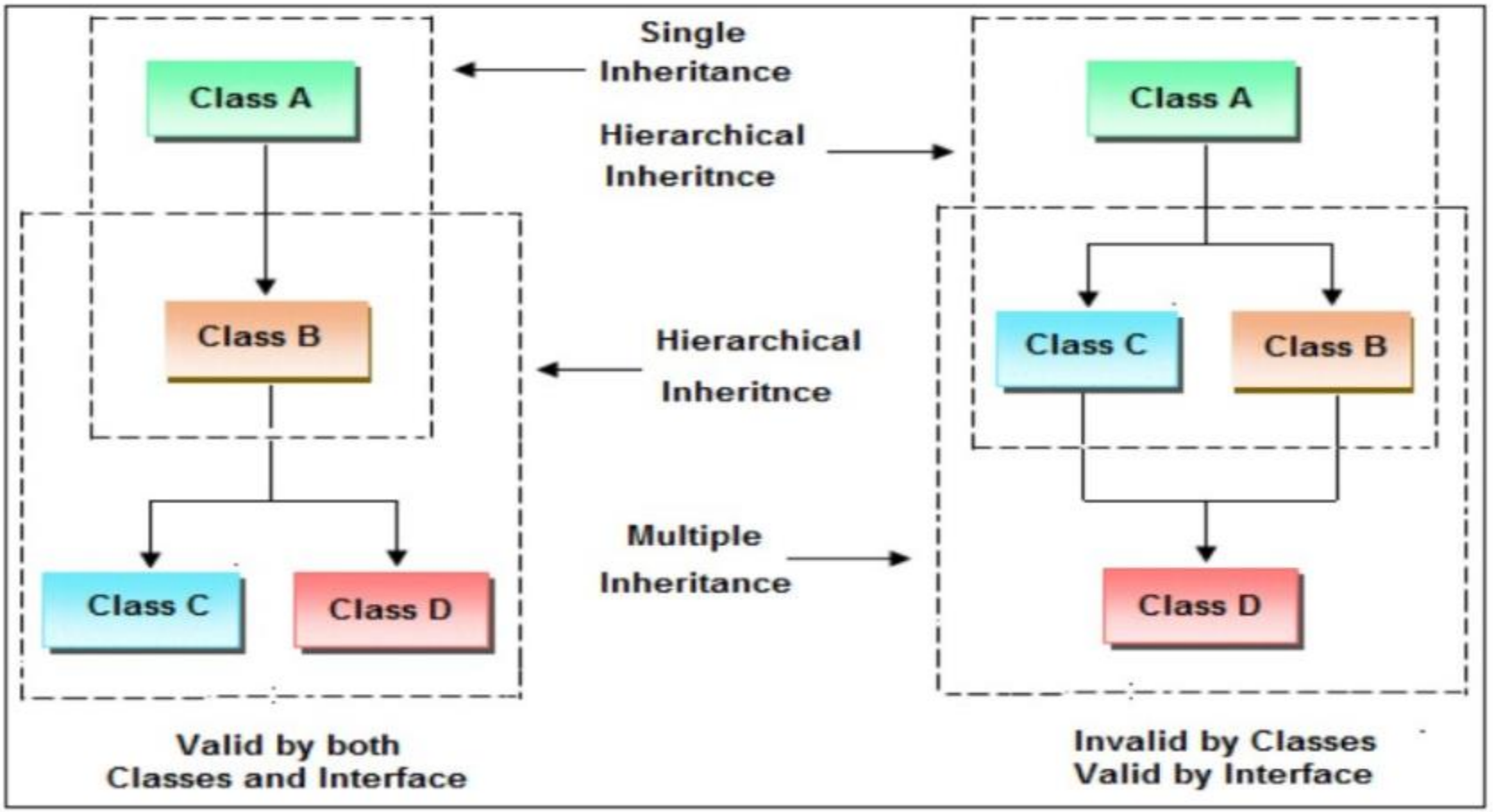
Multiple Inheritance Example

```
class A{
  void msg(){
  System.out.println("Hello"); }
}
class B{
  void msg(){
  System.out.println("Welcome");}
}
class C extends A,B{     //suppose if it were

 public static void main(String args[]){
  C obj=new C();
  obj.msg();    //Now which msg() method would be invoked?
  }
}
```

- **5) Hybrid inheritance**
- Any combination of previous three inheritance (single, hierarchical and multi level) is called as hybrid inheritance.



Hybrid inheritance



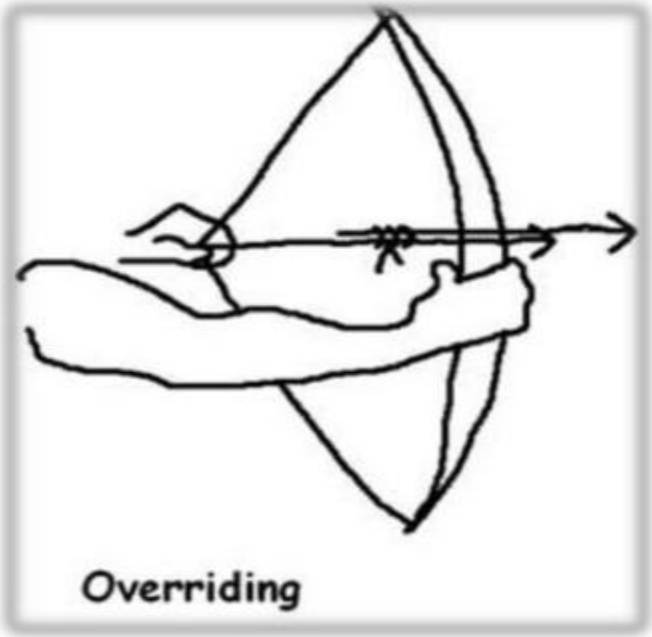
Inheritance in Java

- **Advantage of inheritance**
- If we develop any application using concept of Inheritance than that application have following advantages,
- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

Inheritance in Java

- **Disadvantages of Inheritance**
- Inheritance base class and child classes are tightly coupled. Hence If you change the code of parent class, it will get affects to the all the child classes.
- In class hierarchy many data members remain unused and the memory allocated to them is not utilized. Hence affect performance of your program if you have not implemented inheritance correctly.

Method Overriding in Java



Polymorphism in Java

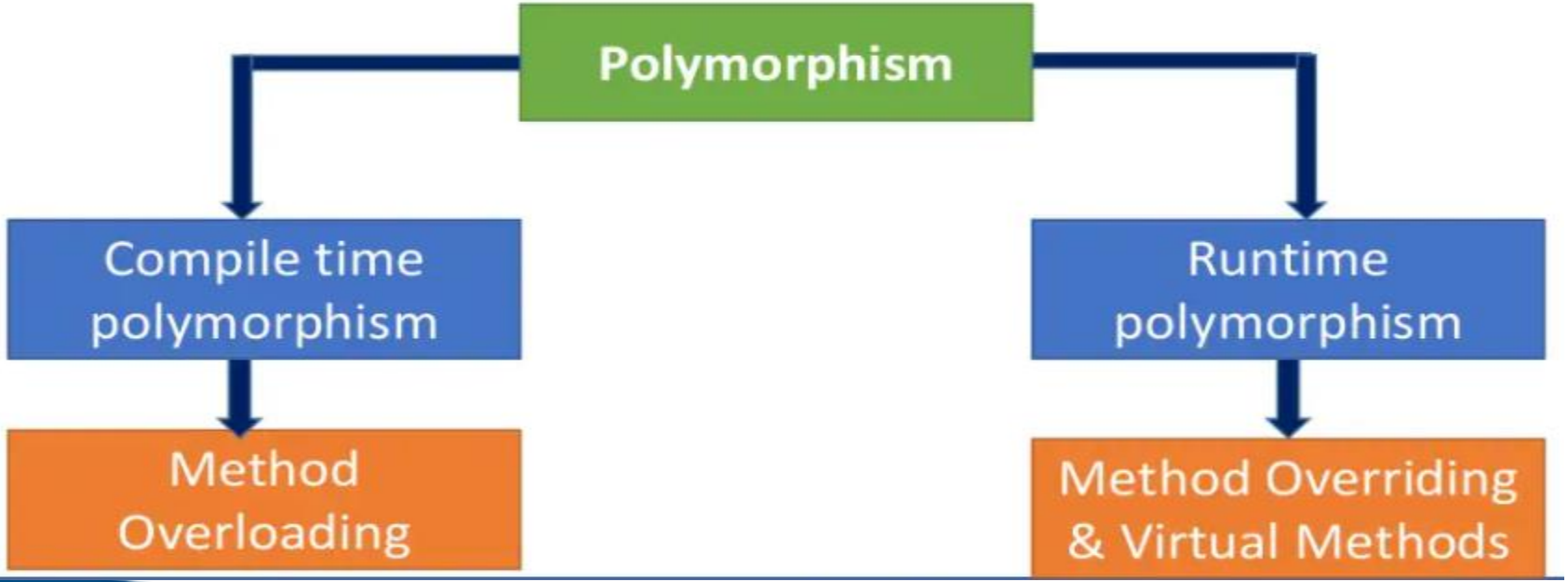
- **Polymorphism in java** is a concept by which we can perform a *single action by different ways*.

- Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Polymorphism in Java

How to achieve Polymorphism in Java ?

- Polymorphism principal is divided into two sub principal they are:
  - Static or Compile time polymorphism
  - Dynamic or Runtime polymorphism



Method Overriding in Java

- Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.

OR

- In other words If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java.**

OR

- In other words If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Without Inheritance Method Overriding is not Possible

- **Advantage of Java Method Overriding**
  - Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
  - Method Overriding is used for Runtime Polymorphism

- **Rules for Method Overriding**
  - method must have same name as in the parent class.
  - method must have same parameter as in the parent class.
  - must be IS-A relationship (inheritance).

Problem Without Method Overriding

```
class Vehicle{
  void run() {
   System.out.println("Vehicle is running");
   }
}
class Bike extends Vehicle{

  public static void main(String args[]){
  Bike obj = new Bike();
  obj.run();
  }
}
```

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

Output is:
Vehicle is running

Example of Method Overriding

```
class Vehicle{
  void run(){
  System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
  void run(){
  System.out.println("Bike is running safely");}

public static void main(String args[]){
   Bike obj = new Bike();
   obj.run();
 }
}
```

Method name are same

Output is:
Bike is running safely

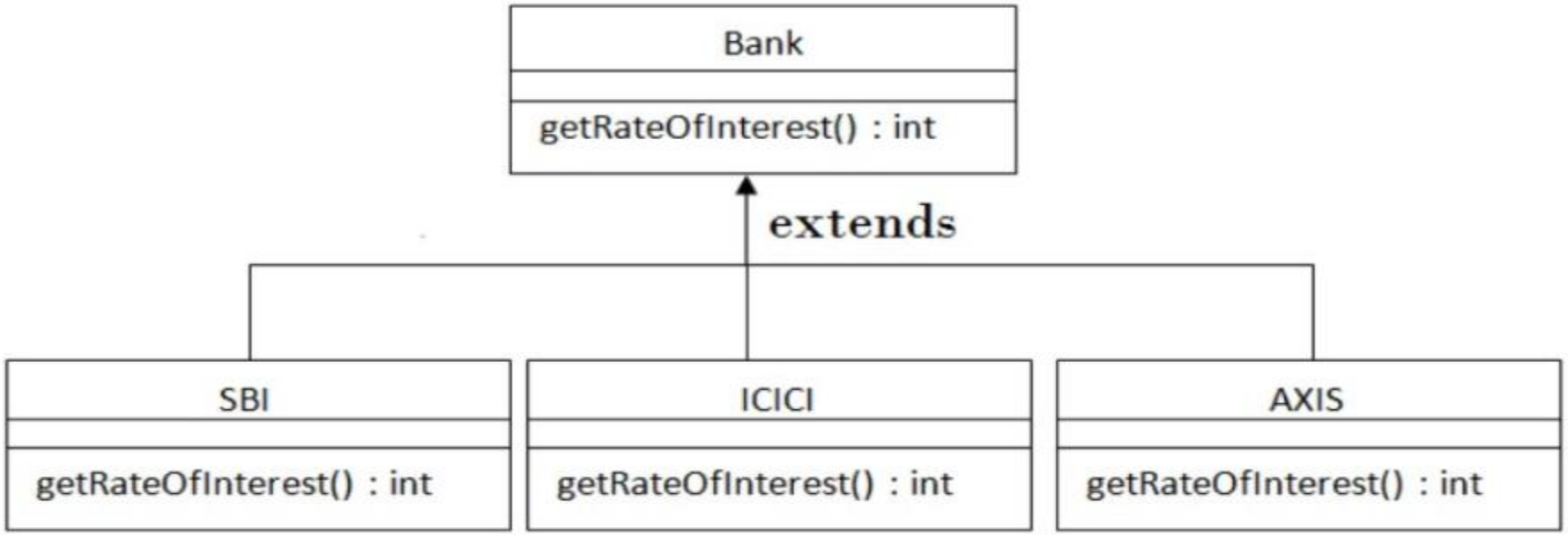
Another Example of Runtime Polymorphism

```
class Animal {
  public void move(){
    System.out.println("Animals can move");  }
}
class Dog extends Animal {
  public void move() {
    System.out.println("Dogs can walk and run");  }
}
public class TestDog {
  public static void main(String args[]) {
    Animal a = new Animal();    // Animal reference and object
    Animal b = new Dog();    // Animal reference but Dog object
    a.move();
    b.move();
  }
}
```

Output is:
Animals can move
Dogs can walk and run

Real Example of Java Method Overriding

- Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.



Real Example of Java Method Overriding

```
class Bank{
  int getInterest(){return 0;}
}
class SBI extends Bank{
  int getInterest(){return 8;}
}

class ICICI extends Bank{
  int getInterest(){return 7;}
}
class AXIS extends Bank{
  int getInterest(){return 9;}
}
```

```
class Test{
public static void main(String args[]){
 SBI s=new SBI();
 ICICI i=new ICICI();
 AXIS a=new AXIS();
 System.out.println("SBI Rate of Interest: "+s.getInterest());
 System.out.println("ICICI Rate of Interest: "+i.getInterest());
 System.out.println("AXIS Rate of Interest: "+a.getInterest());
 }
}
```

Output is:

SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9

super keyword in java

super keyword in java

- Super keyword in java is a reference variable that is used to refer parent class object. Super is an implicit keyword create by JVM and supply each and every java program for performing important role in three places.
  - At variable level
  - At method level
  - At constructor level
- **Need of super keyword:**
- Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

super keyword in java

- **Usage of java super Keyword**
- We can user super keyword by using three ways:

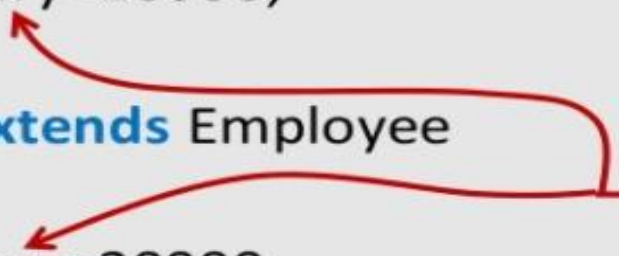
- Accessing Instance Variable of Parent Class (Variable Level)
- Accessing Parent Class Method (Method Level)
- Accessing Parent Class Constructor (Constructor Level)

- **Syntax**

super.baseclass datamember name

Program without using super keyword

```
class Employee {
  float salary=10000;
}
class HR extends Employee
{
  float salary=20000;
  void display() {
   //print current class salary
   System.out.println("Salary is: "+salary);
  }
public static void main(String[] args) {
  HR obj=new HR();
  obj.display();
 }
}
```



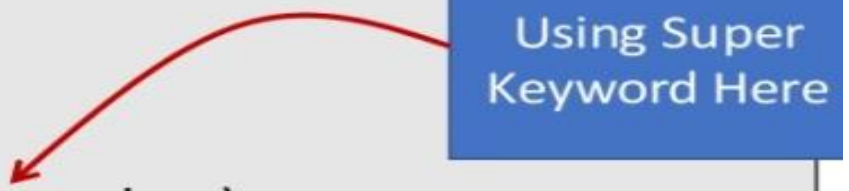
Both Instance Variables Having same Name

Output is:

Salary is: 20000

Program using super keyword at Variable level

```
class Employee {
  float salary=10000;
}
class HR extends Employee
{
  float salary=20000;
  void display() {
   //print base class salary
   System.out.println("Salary is: "+super.salary);
  }
public static void main(String[] args) {
  HR obj=new HR();
  obj.display();
 }
}
```



Output is:
Salary is: 10000

super keyword in java

- **Super at Method Level**
- The **super keyword** can also be used to invoke or call parent class method. It should be use in case of method overriding. In other word **super keyword** use when base class method name and derived class method name have same name.
- **Syntax**

```
Super.method-name();
```

Example of super keyword at Method level

```
class Student{
  void message(){
  System.out.println("Good Morning Sir"); }
}
class Faculty extends Student{
  void message(){
  System.out.println("Good Morning Students");
  }
  void display(){
   message();   //will invoke or call current class message() method
   super.message();  //will invoke or call parent class message() method
  }
public static void main(String args[]) {
 Faculty s=new Faculty();
 s.display();
 }
}
```

Output is:

Good Morning Students
Good Morning Sir

final keyword in java

Final Keyword In Java

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context.
- Final can be:
  - variable
  - method
  - class
- The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

final variable

- **final variables:** If you use a variable as a final then the value of variable is become constant. We cannot change the value of a final variable once it is initialized.
- **Syntax**

```
final data_type variable_name= Some_Value;
```

- **Example**

```
final int data= 10;
```

Example of final variable

```
class Test{

  final int MAX_VALUE=99;
  void myMethod(){
    MAX_VALUE=101;
  }
  public static void main(String args[]){
    Test obj=new  Test();
    obj.myMethod();
  }
}
```

We got a compilation error in this program because we tried to change the value of a final variable "MAX_VALUE".

Compiler Error

```
Exception in thread "main"
java.lang.RuntimeException: Uncompilable
source code - cannot assign a value to final
variable MAX_VALUE
```

Final Keyword In Java

- **Final at method level** It makes a method final, meaning that sub classes can not override this method. The compiler checks and gives an error if you try to override the method.
- When we want to restrict overriding, then make a method as a final.
- **Syntax**

```
public final void fun()
{
.....
}
```

Example of Final Keyword at Method Level

```
class Bike{
  final void run(){
  System.out.println("running");}
}
class Honda extends Bike{
  void run(){
   System.out.println("running safely with 100kmph");
}
  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

Final Method cannot override it

Output is:
Compile Time Error

Final Keyword In Java

- **Final at class level** It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.
- **Syntax**

```
public final class A
{
.....
.....
}
public class B extends  A
{
// it gives an error, because we can not inherit final class
}
```

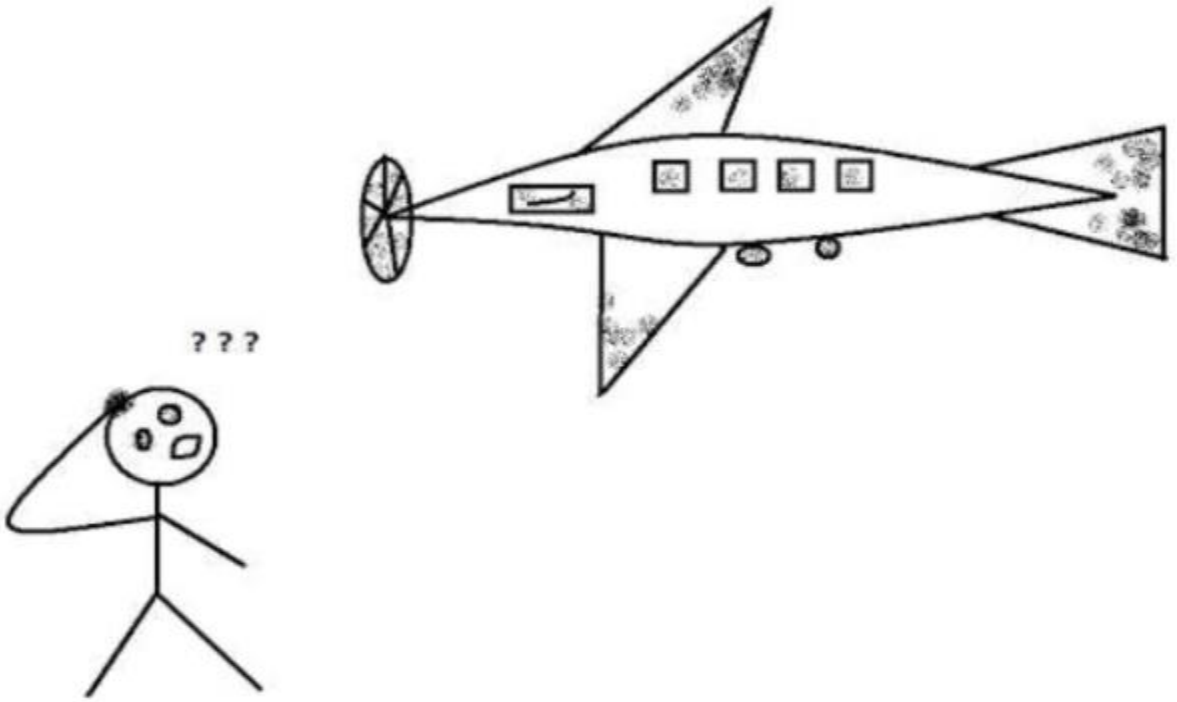
Example of final keyword at class level

```
final class Bike{}
class Honda extends Bike{
  void run(){
   System.out.println("Running");
}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

If a class is final, then we cannot extend it.

Compiler Error

Abstraction in Java



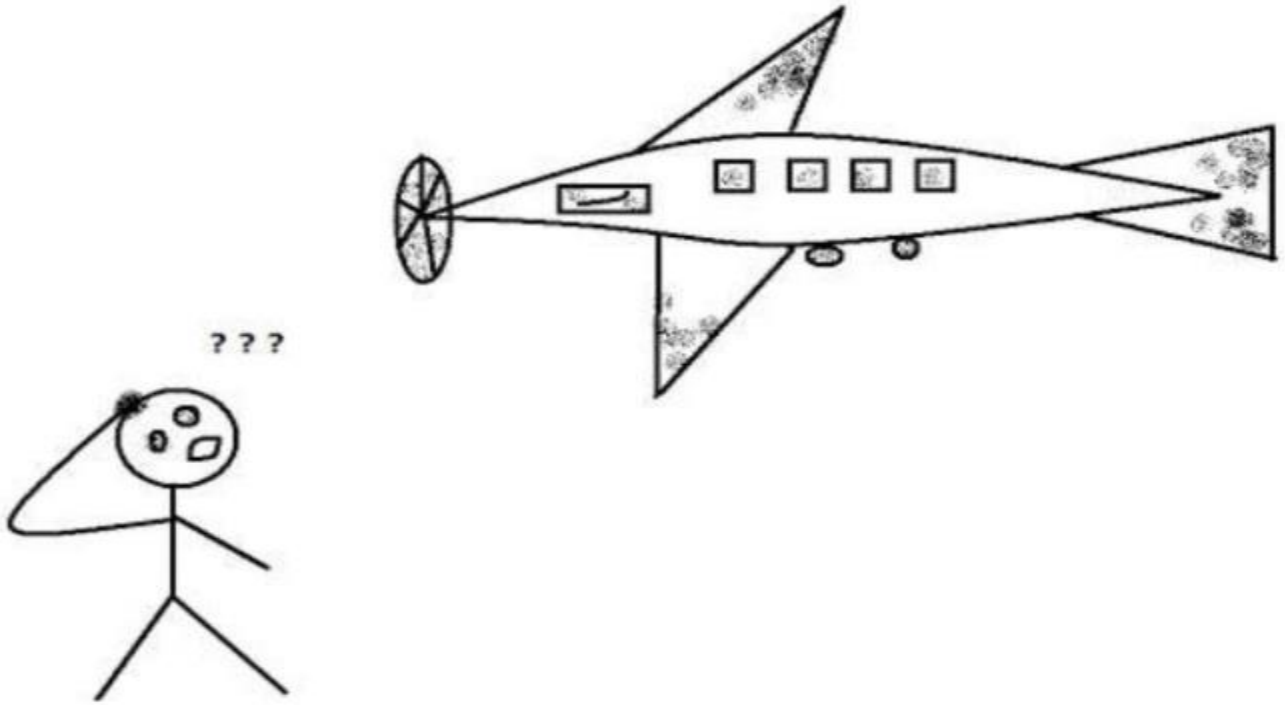
Abstraction in Java

- Another good feature of OOPS is that we can hide the complex and unnecessary details of any object and can display only the required details which are necessary to be displayed so that a person can use that object.
- **Abstraction** is the concept of exposing only the required essential characteristics and behavior with respect to a context.
- Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Abstraction in Java

An includes the essential detail relative to the perspective of the viewer

Abstraction in Java



Suppose there is an airplane flying in the sky. We know that it is flying, but we don't know how exactly it is flying and what are the underlying mechanisms, working and all.

Abstraction in Java

- **Note:** Data abstraction can be used to provide security for the data from the unauthorized methods.
- **Note:** In java language data abstraction can be achieve using class.

- **How to Achieve Abstraction ?**
- There are two ways to achieve abstraction in java
- Abstract class (0 to 100%)
- Interface (Achieve 100% abstraction)

Abstract class in Java

- We know that every java program must start with a concept of class that is without classes concept there is no java program perfect.
  In java programming we have two types of classes they are

- Concrete class (also call normal class)
- Abstract class

Abstract class in Java

- **Concrete class in Java**
- A concrete class is one which is containing fully defined methods or implemented method.

```
class Normal
{
 void  msg()  //method
 {
  System.out.println("Hello Students");
 }
public static void main(String args[]){
 Normal n= new Normal();   //creating an object
 n.msg();
 }
}
```

Abstract Method in Java

- Method that are declared without any body within an abstract class are called abstract method.
- The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.
- **Syntax :**

```
abstract ReturnType methodName(List of formal parameter);
```

- **Example :**

```
abstract void run();    // No definition
abstract int sum(int a, int b); // No definition
```

Example of Abstract Class that has Abstract Method

```
abstract class A {
  abstract void msg();
  }
class B extends A
{
  void msg(){
  System.out.println("Hello My Name is Adil");
 }
 public static void main(String[] args) {
    B = new B();
    b.msg();
 }
}
```

Abstract class with abstract method

Abstract Method Implementation is provided by the B(subclass) class.

Output is:

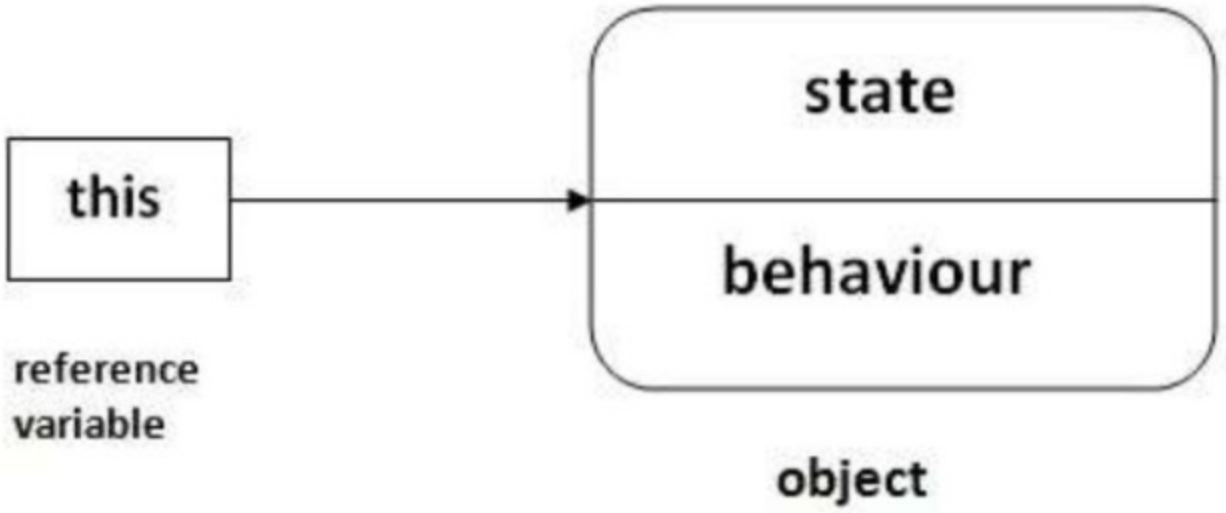
Hello My Name is Adil

Abstract Class Without Any Abstract Method

```
abstract class Base {
   void fun() {
    System.out.println("Base Class Method"); }
}
class Derived extends Base {


 public static void main(String args[]) {
   Derived d = new Derived();
   d.fun();
   }
}
```

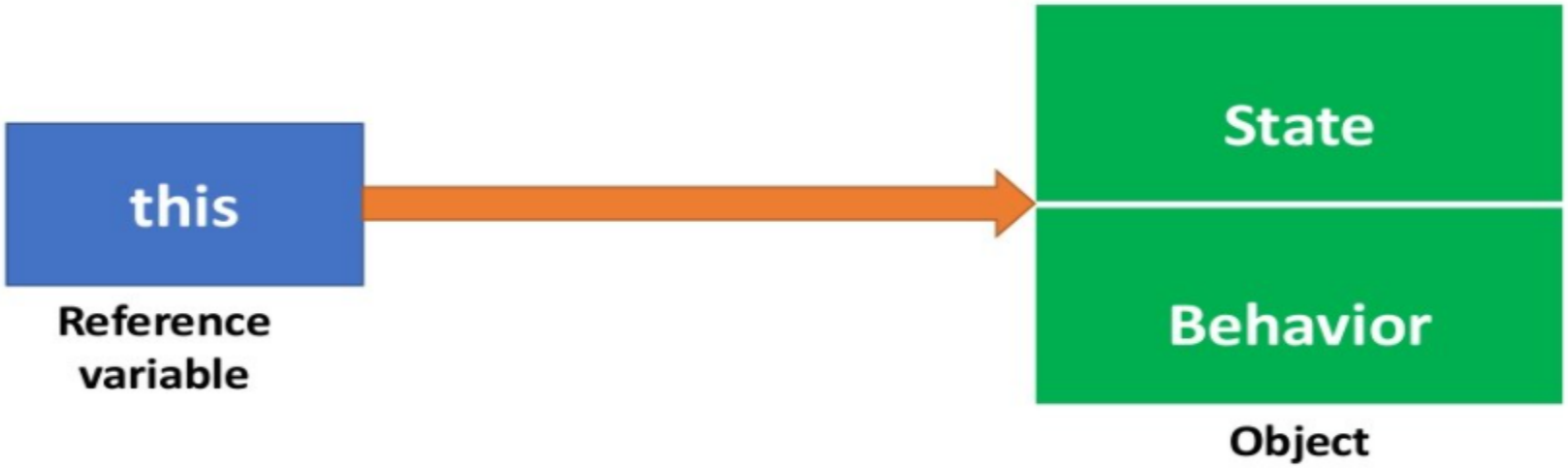
Output is:
Base Class Method

this keyword in java



This keyword in java

- **this** is a reference variable that refers to the current object. It is a keyword in java language represents current class object.



This keyword in java

- **Usage of this keyword**
- It can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- It can be used to invoke current class method (implicitly)
- It can be passed as an argument in the method call.
- It can be passed as argument in the constructor call.
- It can also be used to return the current class instance.

This keyword in java

- **this . (this dot)**
- which can be used to differentiate variable of class and formal parameters of method or constructor.
- **"this"** keyword are used for two purpose, they are
  - It always points to current class object.
  - Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity (no clarity between formal parameter and data members of the class).
- To differentiate between formal parameter and data member of the class, the data members of the class must be preceded by **"this"**.

This keyword in java

- **Syntax**

This is dot operator

this.data member of current **class**.

- **Note:** If any variable is preceded by **"this"** JVM treated that variable as class variable.

Solution of The Previous Problem by this Keyword-1

```
class Student{
   int id;
   String name;

   Student(int id, String name){
   this.id = id;
   this.name = name;
   }
   void display()
   {
      System.out.println(id+"   "+name);
   }
```

using this keyword to distinguish between local variable and instance variable.

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

   Student s1 = new Student(11,"Adil");
   Student s2 = new Student (22,"Hina");
   s1.display();
   s2.display();
   }
}
```

Output is:

11 Adil

22 Hina