**1.** **Why Multiple inheritance is not supported in JAVA**

The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called

Suppose c is the child class extending from both parent class a and parent class b with some methods defined in them. Then child class cannot understand which class method to call. so there is a confusion here which leads to ambiguity and leads to **compile time error.**

**// Error: Test is inheriting from multiple classes**

class Test extends Parent1, Parent2

{

   public static void main(String args[])

   {

       Test t = new Test();

       t.fun();

   }

**However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritances.**

**b) Variable and Data Types**

Variable is a name of the memory location to store values.

We used to store data and use the data once it required.

* Primitive – boolean, char, byte, int, long, float, double,
* Non-Primitive – String, array, classes, interfaces etc.

**2. Loop** (Loop allows to run certain piece of code multiple times)

**a) While loop** -> It will execute only if condition is true

int count = 10;

While(count<50)

{

S.O.P(“Print” +count)

}

**b) Do while loop** -> Same as while loop, but in this loop code or logic will execute at least once

do{

} while(condition)

**3. Condition**

It gives the power to take decision when to execute and what to be done of condition true

**Example-**

Run test case if browser does not equals to null.

Fetch records if excel exist

If(condition) **check the condition is true, then execute the code.**

{

}

If(condition) **-- if condition is true execute if block or execute else block**

{

}

else

{}

**Nested if**

if(){

}

elseif() {

}

else(){}

**Switch statement** -> It allows you match with multiple cases

switch(expression)

{

case value:

//code here

break;

}

**Point for switch statement**

Data type should be same for comparison

We can have multiple cases

We can use only integers String and Enum

Break is optional, but if not provided it will check all cases till the end.

Default is option

**4. Class and Objects**

Class is an entity which binds data members (data type) and member methods (function) into single unit.

Class is best example of encapsulation

**Methods** is a block of statement that perform particular task.

(<Access specifier> <static/non static> <optional-return type> <method name>)

Public static void main ()

1. Non-static method can be called through Object using (.) operator
2. Static method can be called through Class Name directly using (.) operator

* Class name. Static variable
* Class name. Static methods

**A static method** can access only static members. A non-static method can access both static and **non-static members** because at the time when the static method is called, the class might not be instantiated.

**5. Parameter**

To make our program dynamic we can pass parameter while calling.

Parameter can be compiled time and run rime as well.

**6. Polymorphism** (One thing in multiple forms)

Polymorphism allows us to perform a task in multiple way.

**a) Method Overloading** – Method can be overloaded if and only if method has the same name but with different signature (Signature -> Number of argument, Type of argument, Order of argument)

At compile time, Java knows which method to invoke by checking the method signatures. So, this is called compile time polymorphism or static binding.

**b) Method Overriding -** Run Time, Late binding

* Overriding means to override the functionality of an existing method.
* Declaring a method in child class which is already present in the parent class is called Method Overriding.

In this case, if we call the method with child class object, then the child class method is called. To call the parent class method we have to use **super**keyword.

**Runtime polymorphism** - During method call, which method (parent class or child class) is to be executed is determined by the type of an object (object used for call)

**Can we override static methods in java?**  
We can declare static methods with same signature in subclass, ***but it is not considered overriding as there won’t be any run-time polymorphism***. Hence the answer is **‘No’**.

If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class. **if static method is redefined by a derived class, then it is not overriding.**

**7. String Class**

String class has many in-built methods which help you to perform operations on string - Comparing, converting, split, concatenate, etc.

The **equalsIgnoreCase()** method compares two strings irrespective of the case (lower or upper) of the string. Return type is Boolean.

String str=”Payment $100 paid”

str.charAt(8)

str.indexOf(“$”);

str.substring(8) // $100 paid (o/p)

str.length()

str.replace("Microsoft", "W3Schools")

str.toUpperCase()

str1.EqualsIgnoreCase(str2)

str.concat();

char[] inp = str.toCharArray(); //string to char array

**Void** is the Java keyword that tells the compiler that a function will not be returning any value after it is executed. **Return** is the Java keyword that tells the compiler what will be returned when a function is finished.

**8. ARRAY**

**Array is the collection of Similar data types.** (It stores multiple values of same datatypes)

datatype arrayname[] = new datatype[size];

**Example program:**

int id[]=new int[5]

for(int i=0; i< id.length; i++)

{

S.O.P(“Student id is..” + id[i]);

}

**String Reverse:**

String s=”zakir”

String t=””

For(int i=s.length()-1; i>=0; i--)

{

t=t+s.charAt(i);

}

S.O.P(t);

**Duplicate Characters in a string:**

public static void main(String argu[]) {

String str = "w3schools";

char[] inp = str.toCharArray();

System.out.println("Duplicate Characters are:");

for (int i = 0; i < str.length(); i++) {

for (int j = i + 1; j < str.length(); j++) {

if (inp[i] == inp[j])

{

System.out.println(inp[j]);

}

}

}

**9. Java Collection** - Dynamic data structure

* **Java Collections** are predefined set of classes or data structures which can be used to store multiple items in a single unit.
* All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections
* Dynamically allocated data structures in Java (such as Hashtable, HashSet, HashMap,LinkedList, Vector, Stack, ArrayList

## **When to use List and SET**

Use list for storing non-unique objects as per insertion order and use set for storing unique objects in random order.

**a) List**

* List is an interface. List may contain duplicate elements.
* List is an ordered collection (sometimes called a sequence).
* Maintains insertion order

Class implement list interface -> Arraylist**,** Linked list, Vector

ArrayList is an implementation of the List interface. The arraylist class has only a few methods in addition to the methods available in the List interface.

**Arraylist** –

* Can grow dynamically, you can access and insert any value in any index
* Allows random access because array work at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

**Example:**

**ArrayList<String> a=new ArrayList<String>();**

a.add("rahul");

a.add(0, "student");

System.out.println(a.contains(“java”));

a.indexOf(“Rahul”)

a.size()

**What is the difference between Array and ArrayList in Java**

a. Array is static in size while ArrayList is dynamic in size.  
b. Array can contain primitive data types while ArrayList can not contain primitive data types

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses **dynamic array** to store the elements. | LinkedList internally uses **doubly linked list** to store the elements. |
| 2) ArrayList class can **act as a list** only because it implements List only. | LinkedList class can act as **a list and queue** both because it implements List and Deque interfaces. |
| 3) ArrayList is better for **storing and accessing data.** | LinkedList is better for **manipulating the data** |

**b) Set**

* Set is interface in java which extends Collection Interface.
* Set does not accept duplicate values.
* Set is unordered collection
* Can’t be accessed using index

Class implement Set interface -> HashSet, LinkedHashSet, TreeSet

**i) Hashset**

* It stores the elements by using a mechanism called hashing. It means we cannot access using index
* HashSet contains unique elements only.

**Hashset<String> a=new Hashset <String>();**

**ii) LinkedHashSet -** Similar to HashSet but it maintains insertion order.

**iii) TreeSet -** Maintains ascending order

**Methods in Set Interface:**

1.     **add():** This method is used to add one object to the collection at a time.

2.     **clear():** This method is used to remove all elements from the collection.

3.     **contains():** This method is used to verify whether a specified element is present in the collection or not.

4.     **isEmpty():** This method is used to check whether the collection is empty or not.

5.     **iterator():**This is used to return an Iterator object, which may be used to retrieve an object from the collection.

6.     **remove():**This is used to removes a specified object from the collection.

7.     **size():**This is used to know the size or the number of elements present in the collection.

**Map object has unique keys each containing some value, while Set contain only unique values.**

**c) Map**

* Object that maps key to the values.
* A map cannot contain duplicate keys.

Main implementations of map interface – HashMap, LinkedHashMap, TreeMap

**HashMap<String, String> hm = new HashMap<String, String>();**

hm.put("first", "FIRST INSERTED");

HashMap<Integer, String> map = new HashMap<Integer, String>();

map.put (1, "Mark");

map.put (2, "Tarryn");

List<String> list = new ArrayList<String>(map.values());

for (String s : list) {

    System.out.println(s);

}

HashMap<Key, Value> map; // Assigned or populated somehow.

For a list of values:

List<Value> values = new ArrayList<Value>(map.values());

For a list of keys:

List<Key> keys = new ArrayList<Key>(map.keySet());

**10. Access Modifier** – helps to set the level of access you want for the class, variables as well as methods

* **Default**- Access method anywhere only in the package.
* **Public** – variable/method as public: then you can have access across all the packages
* **Private** –variable/methods as private: You cannot access outside the class of same package.
* **Protected**- variable/methods as protected: You can access those inside the package + sub class (by using inheritance) extends of the parent class (out the package).

If a method is set to protected inside a class, it will be accessible from its sub classes defined in the same or different package.

With interfaces, all fields are automatically public, static, and final, and all methods that are declared or defined (as default methods) are public. We can implement any number of interfaces unlike abstract class which can extend only one class.

1. **Super Keyword** -> We can call the parent variable, methods and constructors using super keyword (inside Inheritance)
2. **This keyword** -> we can call the class variable using this keyword.
3. **Final keyword** -> It act as a constant. Final class cannot be inherited. Final methods cannot be override.

finally

{

//This block is executed irrespective of exception thrown or not

(We can use for -- driver.close(), Or to delete cookies)

}

**11) Try/Catch Exception**

try

{

}

Catch(Exception e)

{

System.out.println(“”);

}

**10. Inheritance**

* Client agreement between methods and class implementation
* Inherit the properties of the parent class
* Using inheritance, we can use parent class variables and methods using child class object

**Uses:**

Code Reusability

No Code duplication

For Method Overriding

**Scenario:**

1. Child class reference and child class object – Allow to access all the methods of base class and child class.

2. Base class reference and base class object – Allow to access all methods of base class only.

**11. Abstraction**

Process of hiding the implementation details and showing only functionality to the user

An abstract method is simply a function definition that serves to tell the programmer that method must be implemented in a child class.

**Abstract class -- A** class that is declared as abstract is known as abstract class**. It needs to be extended and its method implemented.** It cannot be instantiated.

**Abstract method --** Method declared as abstract and does’nt have implementation is known as abstract method.

## **Abstract Class vs Interface in Java**

**Difference No.1: Abstract class can extend only one class or one abstract class at a time**

class Example1{

public void display1(){

System.out.println("display1 method");

}}

abstract class Example2{

public void display2(){

System.out.println("display2 method");

}}

abstract class Example3 extends Example1{

abstract void display3();

}

class Example4 extends Example3{

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example4 obj=new Example4();

obj.display3();

} }

**Output:**

display3 method

**Interface can extend any number of interfaces at a time**

interface Example1{

public void display1();

}

interface Example2 {

public void display2();

}

interface Example3 extends Example1,Example2{

}

class Example4 implements Example3{

public void display1(){

System.out.println("display2 method");

}

public void display2(){

System.out.println("display3 method");

}}

class Demo{

public static void main(String args[]){

Example4 obj=new Example4();

obj.display1();

}}

**Output:** display2 method

### **Difference No.2: Abstract class can be extended (inherited) by a class or an abstract class**

class Example1{

public void display1(){

System.out.println("display1 method");

}

}

abstract class Example2{

public void display2(){

System.out.println("display2 method");

}

}

abstract class Example3 extends Example2{

abstract void display3();

}

class Example4 extends Example3{

public void display2(){

System.out.println("Example4-display2 method");

}

public void display3(){

System.out.println("display3 method");

}

}

class Demo{

public static void main(String args[]){

Example4 obj=new Example4();

obj.display2();

}

}

Output:

Example4-display2 method

**1. Interfaces can be extended only by interfaces.**

**2. Classes has to implement them instead of extend**

interface Example1{

public void display1();

}

interface Example2 extends Example1{

}

class Example3 implements Example2{

public void display1(){

System.out.println("display1 method");

}

}

class Demo{

public static void main(String args[]){

Example3 obj=new Example3();

obj.display1();

}

}

**Output:** **display1 method**

### **Difference No.3: Abstract class can have both abstract and concrete methods**

abstract class Example1 {

abstract void display1();

public void display2(){

System.out.println("display2 method");

}

}

class Example2 extends Example1{

public void display1(){

System.out.println("display1 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Interface can only have abstract methods, they cannot have concrete methods**

interface Example1{

public abstract void display1();

}

class Example2 implements Example1{

public void display1(){

System.out.println("display1 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}}

**Output: display1 method**

### **Difference No.4: In abstract class, the keyword ‘abstract’ is mandatory to declare a method as an abstract**

abstract class Example1{

public abstract void display1();

}

class Example2 extends Example1{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**In interfaces, the keyword ‘abstract’ is optional to declare a method as an abstract because all the methods are abstract by default**

interface Example1{

public void display1();

}

class Example2 implements Example1{

public void display1(){

System.out.println("display1 method");

}

public void display2(){

System.out.println("display2 method");

}

}

class Demo{

public static void main(String args[]){

Example2 obj=new Example2();

obj.display1();

}

}

**Difference No.5:**

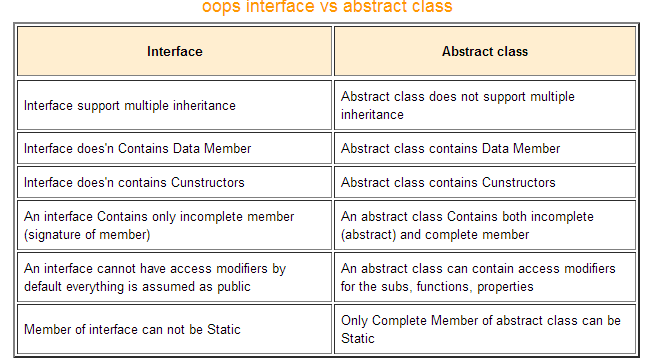
Abstract class can have protected and public abstract methods

**Interface can have only public abstract methods**

**Difference No.6:**

Abstract class can have static, final or static final variables with any access specifier

Interface can have only public static final (constant) variable.



**Packages** – Set of classes and interfaces

**11. Ways to iterate list:**

**Iterator** is generic and doesn't **return** a specific **type** unless you define it in the ArrayList.

**Iterator** is an interface (part of the **Java** Collections) that **returns** the **type** that was passed to it. The **Iterator** is used to traverse the list of elements, and remove an element if necessary.

public class CrunchifyIterateThroughList {

public static void main(String[] argv) {

// create list

List<String> crunchifyList = new ArrayList<String>();

crunchifyList.add("eBay");

crunchifyList.add("Paypal");

crunchifyList.add("Google");

crunchifyList.add("Yahoo");

**// iterate via "For loop"**

System.out.println(" For Loop Example.");

for (int i = 0; i < crunchifyList.size(); i++) {

System.out.println(crunchifyList.get(i));

}

**// iterate via "New way to loop"**

System.out.println("Advance For Loop Example..");

for (String temp : crunchifyList) {

System.out.println(temp);

}

**// iterate via "iterator loop**"

System.out.println("Iterator Example...");

Iterator<String> it= crunchifyList.iterator();

while (it.hasNext()) {

System.out.println (it.next());

}

**// iterate via "while loop**"

System.out.println("While Loop Example....");

int i = 0;

while (i < crunchifyList.size()) {

System.out.println(crunchifyList.get(i));

i++;

}

**// collection stream() util: Returns a sequential Stream with this collection as its source**

System.out.println(" collection stream() util....");

crunchifyList.forEach((temp) -> {

System.out.println(temp);

}

}

}

**JAVA Programs:**

**Sorting an array:**

 public static void main(String[] args)

    {

        int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};

        Arrays.sort(arr);

        System.out.printf("Modified arr[] : %s", Arrays.toString(arr));

    }

**Remove special characters in string (Replace)**

    public class Main {

    public static void main(String[] args) {

        String str="gasd@!dfas%";

        String newStr="";

          for (int i=0;i<str.length();i++)

          {

              //Ascci range for a-z(97-122) A-Z (65-90)

              if (str.charAt(i)>64&&str.charAt(i)<121)

              {

                    newStr+=str.charAt(i);

              }

}

         System.out.println("String before filter "+str);

        System.out.println("String after filter "+newStr);

 }

}

String str = "Hello +-^ my + - friends ^ ^^-- ^^^ +!";

str = str.replaceAll("[-+^]\*", "");

**How to take input from user in java:**

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

{

**int** num;

**float** fnum;

String str;

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

//Get input String

System.***out***.println("Enter a string: ");

str = in.nextLine();

String rep=str.replaceAll("[#$%]"," ");

System.***out***.println("Replaced string is" +rep);

 }

     str = in.nextLine();

     num = in.nextInt();

     fnum = in.nextFloat();

**Reverse a list**

List<String> mentors = new ArrayList<String>();

mentors.add("Rasheed");

mentors.add("Gopi");

//Collections.sort(mentors);

Collections.reverse(mentors); //For list

for (String mentor : mentors)

{

System.out.println(mentor);

}

**Swapping of two numbers**

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

   {

**int** x, y, temp;

      System.***out***.println("Enter x and y");

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

      x = in.nextInt();

      y = in.nextInt();

      System.***out***.println("Before Swapping \n x = "+x+"\n y = "+y);

      temp = x;

      x=y;

      y=temp;

      System.***out***.println("After Swapping \n x = "+x+"\n y = "+y);

}

**Fibonacci series**

    public static void main(String a[]){

         int febCount = 15;

         int[] feb = new int[febCount];

         feb[0] = 0;

         feb[1] = 1;

         for(int i=2; i < febCount; i++){

             feb[i] = feb[i-1] + feb[i-2];

         }

         for(int i=0; i< febCount; i++){

                 System.out.print(feb[i] + " ");

         } }

A final class can't be extended, an abstract class needs to be extended in order to be instantiated.

NOTE :- A final class cannot have abstract methods and an abstract class cannot be declared final

* Whenever static variable is declared in java it belongs to class not the object.