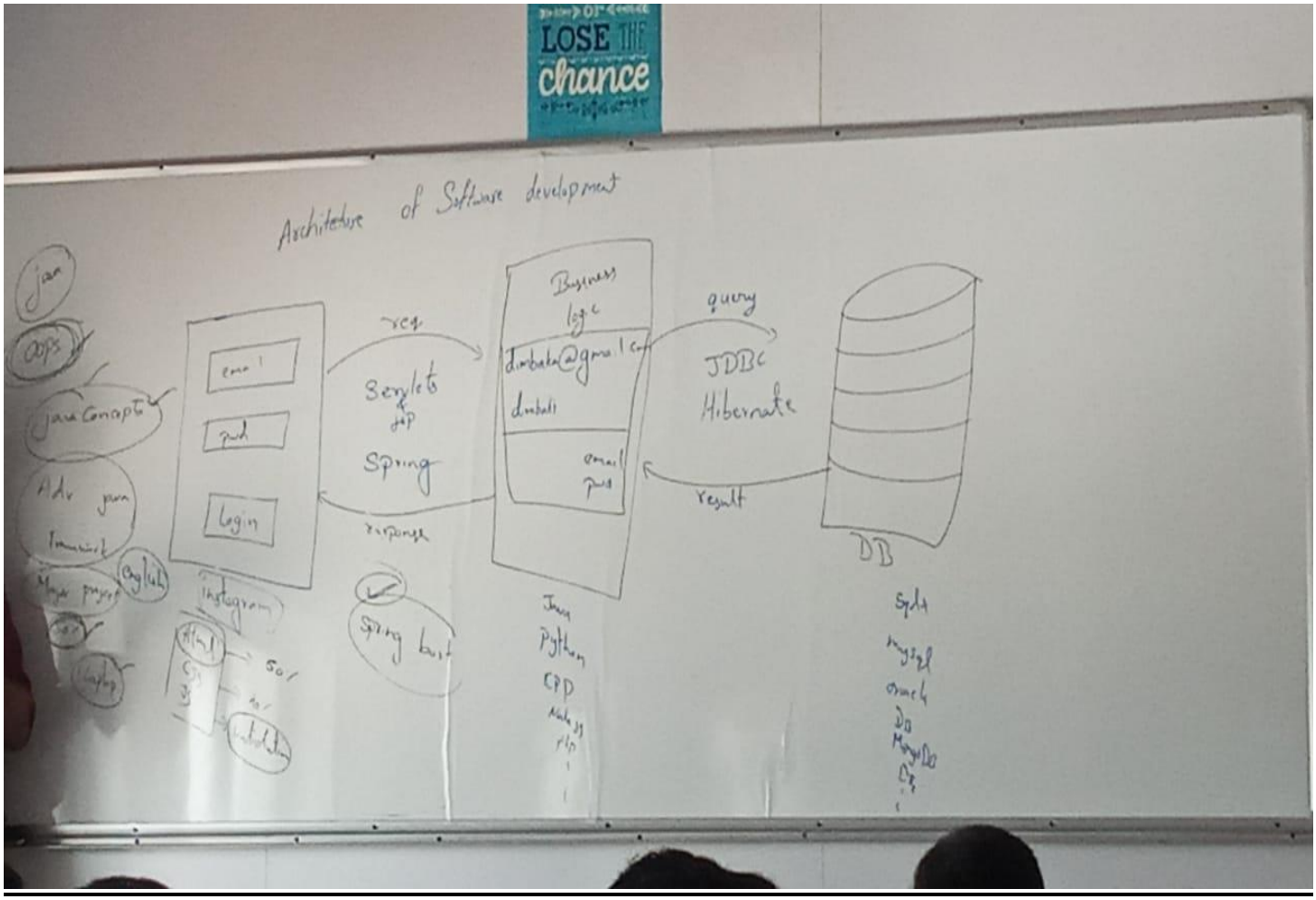
Advance Java

COURSE DETAILS

- FILE HANDLING
- Multi-threading
- Java 8 features
- Functional interface
- Lambda Expression
- String API

ARCHITECTURE OF SOFTWARE DEVELOPMENT



File handling

C->create

R -> Read

U -> Update/modify

D -> Delete

We have to use the java.io package for CURD.

By using file class we can create file. File class contains some methods like canRead(), canWrite(), length(), createNewFile(), mkdir() //imp methods, delete(), exists(), list().

To create a simple file

```
import java.io.File;
class createFile{
    public static void main(String[] args) {
       File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\file");  //make sure
to use slashes
     boolean b= file.mkdir();
     if(b == true){
        System.out.println("folder has been created");
     }
     else{
        System.out.println("folder already exists");
     }
    }
}
```

By reducing codes

```
import java.io.File;
class createFile{
    public static void main(String[] args) {
       File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\file");
     if(file.mkdir()){
        System.out.println("folder has been created");
     }
     else{
        System.out.println("folder already exists");
     }
    }
}
```

How to create text file

```
import java.io.File;
```

```
import java.io.IOException;

class createFile {
    public static void main(String[] args) throws IOException{
        File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\domain.txt");

        if (file.createNewFile()) {
            System.out.println("folder has been created");
        }

        else {
            System.out.println("folder already exists");
        }

    }
}
```

To delete a particular text

```
import java.io.File;
import java.io.IOException;

class createFile {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\domain.txt");
        if (file.createNewFile()) {
            System.out.println("folder has been created");
        }

        else {
            System.out.println("folder already exists");
            file.delete();
        }

    }
}
```

To know what’s inside of the folder

```
import java.io.File;
import java.io.IOException;
import java.util.Arrays;

class createFile {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java");
       String[]list = file.list();

       for(String str : list){
System.out.println(str);  //output will be file,java as it is present  inside of the
folder
       }
```

```
    }
}
```

To know whether the file you have created is readable, writeable and length is available or not

```
import java.io.File;
import java.io.IOException;
import java.util.Arrays;

class createFile {
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\demon.java");
        if (file.createNewFile()) {
           System.out.println("it's there");
        }

        else {
            System.out.println(file.canRead()); //true
            System.out.println(file.canWrite()); //true
            System.out.println(file.length()); //0



        }

    }
}
```

How to write a file

Stream

It is basically the sequence of data.

It is of 2 types and these are abstract classes.

1.Input stream

It contains fileInputStream, InputStream is used to read a file.

2.Output Stream

It contains fileOutputStream, OutputStream is used to write a file.

We can write anything inside of it by the help of write() and it is of byte type.

to write a program

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

class program {
    public static void main(String[] args) {
        File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\file\\jjk.txt");
        try {
            file.createNewFile();
            FileOutputStream fos = new FileOutputStream(file);
            String str = "domain expansion";
            byte[] arr = str.getBytes();
            fos.write(arr);
            fos.close();
            System.out.println("file is written");

        } catch (IOException e) {
            // fos.write();
            e.printStackTrace();
        }

    }
}
```

The asc key value for space is -1

To read

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class read {
    public static void main(String[] args) {
        File file = new File("C:\\Users\\ASUS\\Desktop\\Qspider Java\\file\\jjk.txt");
        try{
            FileInputStream fis = new FileInputStream(file);
            int num = fis.read();
            while(num != -1){

                System.out.println((char)num);
                num = fis.read();
            }
        }
        catch(IOException e){
            e.printStackTrace();
        }
```

```
    }
}
```

Interview question

```
class cls3{

    public static void main(String[] args) {

        System.out.println((int)((float)(int)10.6));

    }

} //output will be 10;
```

Serialization

Convert object into bytestream. It is mandatory to make an object to serialize format. It is a marker interface(without variable or anything).

Deserialization

Converts bytestream to object.

When we use transient keyword then the variable will not be serialized.

```
private transient int mobile;
```

Multi-Threading

Multi-tasking

Performing multiple task simultaneously.

It is of 2 types

1.Process based multi-tasking(Os level)

Process- execution of a task, it creates separate memory for separate process.

Example – separate memory for Whatsapp, separate memory for mail, separate memory for Youtube if there is multi-tasking going on. It is also a heavy weight process.

2.Thread based multi-tasking

Thread - It is the sub process.

Basically in thread we give our task to multiple threads, by doing it we can save a lot of time.

Thread will share the memory.

Thread is independent, one thread will not be dependent on another thread.

It is a light weight sub process.

Internally 3 threads will be created while running a program, known as main thread, thread scheduler, garbage collector.

When we run our Java Program internally JVM starts threads Namely,

→ Main

→Thread scheduler

→Garbage Collector

main thread: -

The main thread acts as the main () method of the program, which is the entry point of the program.

thread scheduler: -

The thread scheduler picks up those threads that are in the Runnable state. When there are multiple threads in Runnable state, there is no guarantee which thread it picks up first.

Garbage collector

When Java programs run on the JVM, objects are created on the heap area, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

Basically jvm creates the main thread, once main thread is created , jvm will start the main method.

Thread scheduler will schedule a thread to execute. By doing it we can save memory.

Multi-threading is mostly used in gaming companies but not more likely to be used in developing companies.

How to create a thread

It can be created by 2 ways

1. By extending Thread class

- **Create class & extend class thread**
- **Override run method**
- **Write your task in run()**
- **Create object of the thread type class and call start().**

```
class PrintNumbers extends Thread{
    public void run(){
        for(int i=1;i<=10;i++){
            try{
                System.out.println(i);
                Thread.sleep(1000);
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}

class cls5{
    public static void main(String[] args) {
        PrintNumbers t1 = new PrintNumbers();
        t1.start();  //  by the help of the start() it will create a stack.

        for(char ch='a';ch<'z';ch++){
            try{
           System.out.println(ch);
                Thread.sleep(1000);
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

2. By implementing runnable interface

Thread class

Thread is a class which is present inside in java.lang package.

Theread method

1. setPriority()
2. getPriority()
3. sleep
4. join
5. start()
6. run()
7. setName()
8. getName()
9. currentThread() // it will return the current thread object

how to print a name using thread

```
class read{
    public static void main(String[] args) {
        String name = Thread.currentThread().getName();
      System.out.println(name);  //main
        Thread.currentThread().setName("Tushar");
        String name2 = Thread.currentThread().getName();
  System.out.println(name2);//tushar

    }
}
```

Write a program related to sleep method

1,2,3 will be printed and then it will wait for 5 seconds and then it will print 4 and 5.

```
class read{
    public static void main(String[] args) throws InterruptedException {
        System.out.println(1);
        System.out.println(2);
        System.out.println(3);
        Thread.sleep(5000);
        System.out.println(4);
        System.out.println(5);

    }
}
```

Task

Print a-z, and the time gap should of 1s each time a character is printed/executed.

```
class read{
    public static void main(String[] args) throws InterruptedException {
       for(int i = 97; i <= 122;i++){
        System.out.println((char)i);
    Thread.sleep(2000);
       }
    }
}
```

Write to a program in which both number and letters will work at a time

Note – we can call the run() but it is a normal call.

We can't call start() twice because it is already assigned to a task.

Using one thread reference we can call start method only once.

We can override the start() but we can't achieve multi threading or multi-tasking through that. Through start() we can create stack memory.

Thread Scheduler gives priority to the threads.

Object modification/ Tracing

```
class cls6 {
    public static void main(String[] args) {
        Student st = new Student();
        st.name="suhas";
        System.out.println(st.name);// suhas
        Student s2 = m1(st);
        System.out.println(s2.name);// null
        System.out.println(st.name); // dinga
    }
    private static Student m1(Student s){
        s.name="dinga";
        return new Student();
    }

}

class Student{
```

```
    String name;
    int age;
    long mobile;
}
```

WE shouldn't point to the same object at the same time or ELSE there's gonna be inconsistent data.

Synchronization

By the help of synchronization we can allow all the threads to access by one thread.

Example – when we open bookmyshow app/web then when we check for movie tickets it shows that some tickets are already booked so this is one of the example of synchronization.

Data might be inconsistent without synchronization.

It can be achieved by the help of synchronized keyword.

Write a program which is kinda similar to bookMyshow in which what you have to do is check the seats and try booking tickets, if after booking it is full then show the result or if the seats are full then show the result.

Step-1

```
public class bookmyshow{
    private int totalSeats = 50;
    public synchronized void bookSeats (int n){
        if(n>totalSeats){
            System.out.println("seats are not available");

        }
        else{
            System.out.println("number of seats:" + totalSeats);
            totalSeats -= n;
            System.out.println("remaining seats are " + totalSeats);
        }
    }
}
```

Step-2

```
public class Tushar extends Thread{
```

```
    bookmyshow bms;

    public Tushar (bookmyshow s){

    this.bms=s;

    }

    public void run(){

    bms.bookSeats (40);

    }

    }
```

Step-3

```
public class Tushar extends Thread{

    bookmyshow bms;

    public Tushar (bookmyshow s){

    this.bms=s;

    }

    public void run(){

    bms.bookSeats (40);

    }

    }
```

Step-4

```
public class Makima extends Thread {
    bookmyshow bms;
    public Makima (bookmyshow bms){
        this.bms = bms;
    }
    public void run(){
        bms.bookSeats(30);
    }
}
```

Step-5

```
public class App {
    public static void main(String[] args) {
        bookmyshow bms = new bookmyshow();
        Thread Tushar = new Tushar(bms);
        Thread Makima = new Makima(bms);
        Tushar.start();
        Makima.start();
    }
}
```

SYNCHRONIZATION FOLLOWS LOCK RULE.

DEADLOCK

WE can avoid deadlock but we can't solve it.

Example there are 2 people, one of em is working and another one is cooking and at the same time they tried to call each other at the same time but they call won't happen as they are calling each other at the same time, So this the example of deadlock.

There is some chances of deadlock while performing multi-threading.

Step-1

```
public class A {
    public synchronized void eat(C b) {
        System.out.println("A  is ready to call");
        b.relax();
        System.out.println("A task completed");
    }

    public synchronized void digestion() {
        System.out.println("digestion completed");
    }
}
```

Step-2

```
public class C {
    public synchronized void sleep(A a){
        System.out.println("B task is completed");
        a.digestion();
        System.out.println("b task is completed");

}
```

```
public synchronized void relax(){
    System.out.println("Relaxed well");
}
}
```

Step-3

```
public class T1 extends Thread {
    A a;
    C b;
     public T1(A a, C b){
        this.a = a;
        this.b = b;
     }
     public void run(){
        a.eat(b);
     }
}
```

Step-4

```
public class T2 extends Thread {
    C b;
    A a;
    public T2(C b, A a){
        this.b = b;
        this.a = a;

    }
    public void run(){
        b.sleep(a);
    }
}
```

Step-5

```
public class main {
    public static void main(String[] args) {
        A a = new A();
        C b = new C();
        T1 t1 = new T1(a, b);
        T2 t2 = new T2(b, a);
        t1.start(); // B task is completed
        t2.start(); // A is ready to call
    }
}
```

Inter Thread communication

One thread is gonna communicate to another thread.

One thread will come for another thread to come and notify.

To use the methods synchronization is mandatory.

Methods

wait();

wait(miliseconds);

wait(ms,ns);

notify();

notifyall();

WRITE A PROGRAM IN WHICH THERE IS PRESERVED BANK BALANCE BUT YOU WANT TO WITHDRAW SOME OF IT, IF THE BALANCE IS LESSER THAN THE ENTERED AMOUNT THEN SHOW SOMETHING AND IF THE BALANCE IS GREATER THAN THE ENTERED AMOUNT THEN LET IT WITHDRW THE AMOUNT.

STEP-1

```
import java.util.Scanner;

public class Bank {
   private int balance = 10000;
   public  synchronized void  withdraw(int amt){
    if(amt>balance){
        System.out.println("insufficient funds");
        try{
            wait();
            balance -= amt;
            System.out.println("amount widthdraw is been succefull");
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
    }
    else{
        balance -= amt;
        System.out.println("successfull withdrawl");
    }

   }
   public synchronized void deposit(){
    System.out.println("enter your amount");
    Scanner sc = new Scanner(System.in);
```

```
    int amt = sc.nextInt();
    balance += amt;
    System.out.println("deposited successfully");
    notify();

}
}
```

STEP-2

```
public class RYOIKI extends Thread {
    Bank bank;
    public RYOIKI(Bank b){
this.bank = b;
    }
    public void run(){
        bank.withdraw(20000);
    }
}
```

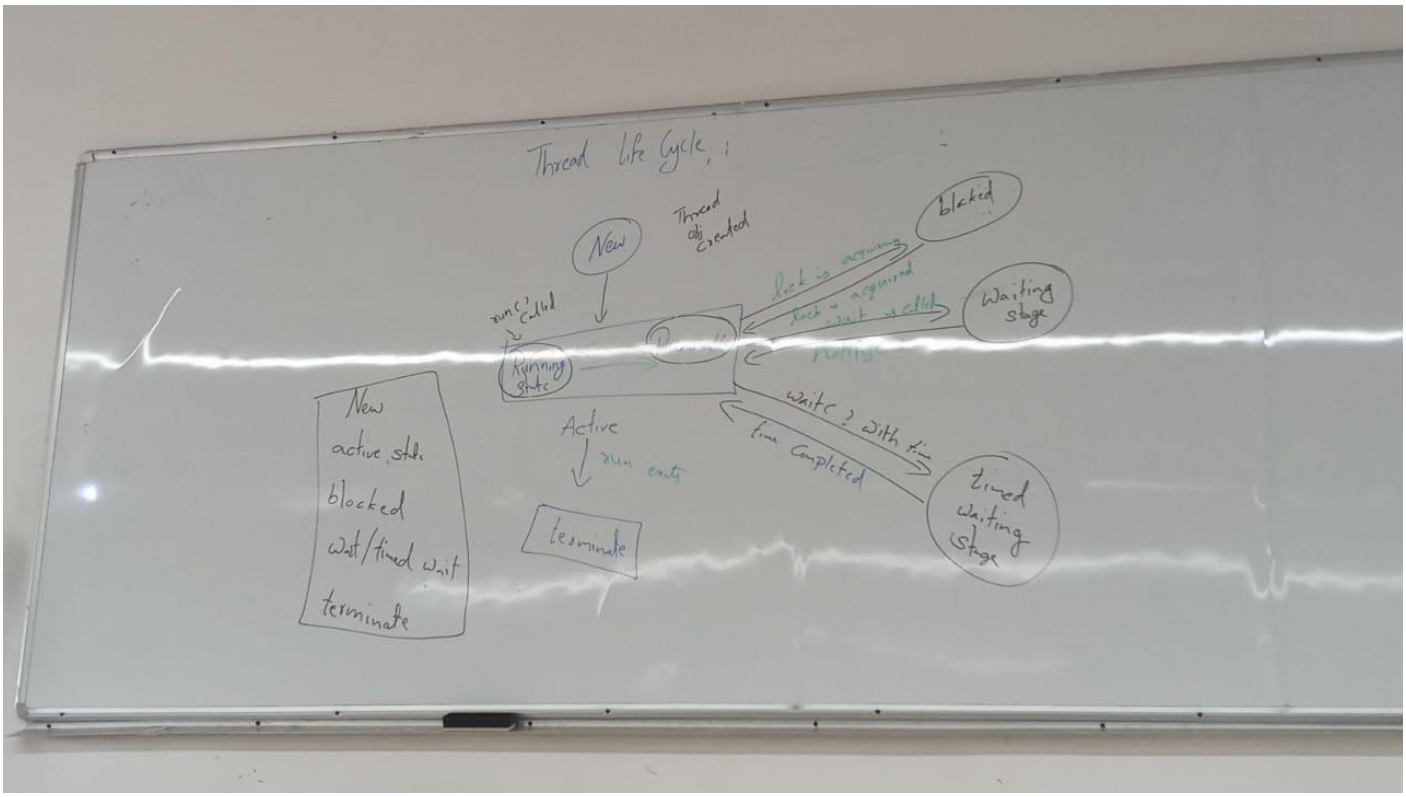
STEP-3

```
public class tenkai extends Thread {
    Bank bank;
    public tenkai(Bank b){
this.bank = b;
    }
    public void run(){
        bank.deposit();
    }
}
```

STEP-4

```
public class Mainbank {
    public static void main(String[] args) {
        Bank sbi = new Bank();
        Thread RYOIKI = new RYOIKI(sbi);
        RYOIKI.start();
        Thread tenkai = new tenkai(sbi);
        tenkai.start();
    }
}
```

Thread cycle



Thread life cycle goes like

- **new**
- **active state**
- **blocked state**
- **wait/timed wait**
- **terminate**

Thread goes from running state to runnable state due to the above reasons.

Creating thread by implementing runnable interface

Runnable interface

It is a functional interface, Contains only one abstract method known as functional interface.

It is more efficient than the thread class.

Empty interface is known as marker interface.

- **Create class and implement runnable interface**
- **Write the task**
- **Create object of runnable type.**

- Pass the object reference to thread constructor.
- Call start method using the thread object.

Java8 features (java concept - imp interview question)

- Default and static concrete methods are allowed.
- Functional Interface(can have only one abstract method)
- Lambda expressions(similar to java script arrow function)
- Method reference(alternative of lambda expression)
- Stream API(can reduce the line of code)
- Date & time API(showing data and time functionality)
- forEach()

Default and static methods

It got introduced due to the reason where by changing one interface was affecting all other child interfaces, which is why default and static methods were introduced as java 8 feature. By default interface is abstract which is why using abstract keyword before interface is optional. By default the access modifier in interface is public which is why it is important to give higher priority or same in its child implementation.

Static is allowed because we can call the method in various ways, we can even call the methods without creating an object.

We can create main method inside of an interface. It is possible because it is static in nature.

```
public interface A {
   public static void main(String[] args) {
      System.out.println("Hii"); // hii
   }
}
```

Static methods in interface default and static methods were introduced in java 8 and this static method can’t be inherited to a child class.

Possible scenario

```
class A {
    public static void m1() {
       System.out.println("hii");
    }
 }

 class B extends A {
    public static void main(String[] args) {
       B.m1();// hii
    }
 }
```

It is impossible when it comes to interface, which means we can't inherit static methods to its child class/interface and here is an example.

```
public interface A {
    public static void m1() {
       System.out.println("hii");
    }
 }

 class B implements A {
    public static void main(String[] args) {
       A.m1(); //hi

       B.m1();// error cuz we can't inherit it


    }
 }
```

```
 interface A {

      void m1();

      public static void staticMethod(){

         System.out.println("it is a static method");

      }

      default void defaultMethod(){

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

      }

      default int sum(int a, int b){
```

```
        return a +b;
      }
 }


 class B implements A {
   public void m1(){
      System.out.println(" m1() overidden, output will be  this");
   }
 }
  public class main1 {
public static void main(String[] args) {
       A  a = new B();
       a.defaultMethod(); // default method
       A.staticMethod();  // it is a static method
       int sum = a.sum(11, 12);
       System.out.println(sum);  //23
}
 }
```

To make multiple-inheritance possible with the help of interface

```
 interface A {
      default void m1(){
         System.out.println("Hello");
      }
 }

interface B {
   default void m1(){
      System.out.println("Hii");
     }
 }
class C implements A,B{

public void m1(){
   A.super.m1();  // hello
   B.super.m1();  // hii
}
}
public class main1{
   public static void main(String[] args) {
      C c = new C();
```

```
        c.m1();
    }
}
```

We can achieve it with the help of super keyword.

Functional interface

It should have only one abstract method.

Correct way to use functional interface

To achieve functional interface we have to use @FunctionalInterface

```
@FunctionalInterface
interface A {
void m1();
}
```

It will show error if we tryna add more than one abstract class

```
@FunctionalInterface
interface A {
void m1();
void m2(); // error
}
```

Possible ways even tho we use abstract method

```
@FunctionalInterface
interface A {
void m1();
public abstract String toString(); // it is an abstract class but won't give error as the
compiler is going to ignore it as tOsTRING methods includes in every class of java.
boolean equals(Object o); // compiler is gonna ignore it too.

}
```

Types

Useful for stream API.

```
1. interface Predicate <T>
{
public Boolean test(T t)
}
```

```
2. interface Consumer <T>
 {
Public void accept(T t);
}
```

```
3.interface Supplier<R>
{
     Public R get();
}
```

```
4. interface Function <T, R> {
   public R apply(T t);
}
```

Lambda expressions

It is an anonymous function/method.

It doesn’t have modifier, return type, method namew.

These expressions are used to give the implementation to the functional interface.

It will give you the object of functional interface.

If we are taking the parent reference then in the compile –time the parent reference will be compiled meanwhile in runtime the child class will be executed.

By the help of lambda expression we can reduce lines of codes.

Program using normal way and at the same time using lambda expresision

```
interface A {
    public void m1();
}

public class mai implements A {
    public void m1() {
        System.out.println("hii");

    }
}

class main1 {
    public static void main(String[] args) {
        A a = new mai();
        a.m1(); // hii
        A a2 = () -> {
            System.out.println("hello");
        };
        a2.m1(); // hello will be the output, we used lambda expression here which is
similar to
                 // arrow function

    }
}
```

```
interface A {
    public void m1(int x,int y);
}

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

```
        A a2 = (int x, int y) -> {
            System.out.println(x+y);
        };
        a2.m1(10,20);  // we can use lambda expression without needing to implementing it

        // we can also reduce the lines of code

        A a3 = (x,y)->{
            System.out.println(x+y);
        };
        a3.m1(5,5); //10
    }

}
```

When multiple arguments are there it is mandatory to have curly braces but if there is only one argument then curly braces is not mandatory.

```
interface A  {
    public int m1(int side);
}

 class mai implements A {
       public int m1(int side){
        return side * side;
       }
    }
    class main1 {
        public static void main(String[] args) {
            A a = new mai();
           int m1 = a.m1(2);
           System.out.println(m1);

           A a1 = (int s) ->{
            return s*s;
           };
           System.out.println(a1.m1(5));  //25


              A a2 = s -> s*s;
           System.out.println(a2.m1(3));  // by reducing code, output will be 9 here

        }
    }



import java.util.HashMap;
import java.util.Map;

class demo1 <T,R>{
```

```
    public void add(T a){
 System.out.println(a);
    }
public void put(T t, R r){
    System.out.println(t);
    System.out.println(r);
}
public static void main(String[] args) {
    Map <String,Integer> map = new HashMap();
    map.put("maths", 100);
    demo1 <Integer, String> d1 = new demo1<>();
    d1.put(1000, "science");
}
}
```

Check whether the given numbers are even or odd

```
import java.util.function.Predicate;
class demo1 <T,R>{
     public static void main(String[] args) {
        Predicate<Integer> obj = n -> n % 2 == 0;
        System.out.println(obj.test(50));  //true
     }
}
```

To sort an array

```
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;

public class imple {
    public static void main(String[] args) {
        ArrayList <Integer> list = new ArrayList();
         int [] arr ={1,5,3,4};
         Arrays.sort(arr);  // will give the output in address format

         System.out.println(Arrays.toString(arr));  // [1,3,4,5] , it will convert the
address to string format
    }
}
```

To sort ArrayList

```
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
```

```java
public class imple {
    public static void main(String[] args) {
        ArrayList <Integer> list = new ArrayList();
         list.add(10);
         list.add(5);
         list.add(9);
         list.add(11);
         Collections.sort(list); // it will sort the list
         System.out.println(list); // [5,9,10,11]
    }
}
```

```java
import java.util.List;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.function.Predicate;

public class imple {
    public static void main(String[] args) {
         List<String> list2 = Arrays.asList("dinga","dingi", "donkey","monkey");
        //  Predicate<String> p = s ->s.startsWith("d");
        //  for(String s: list2){
        //     if(p.test(s)){
        //         System.out.println(s);   // it will give all the output having d in all
their names.
        //     }
        //  }
        list2.forEach(s-> System.out.println(s));  // will get all the names in the output
like dinga, dingi,donkey and monkey.

    }
}
```

Method reference

It is an alternative of lambda expression.

To check method reference, for that we have to check whether the modifier type is static or not, If it is non-static then we have to create object.

If the return type is void then we can have any kind of return type in child class like whether it is Boolean or int or anything.

If the modifier is something except void then we have to mention the same modifier in the child class too.

The parent and child's method argument need to be blank or need to be matched.

Example of method reference

```
        ABC obj2 = demo::m1;  // method refernce
        obj2.hii();// calling
```

```
interface ABC {
    void hii();
}
public class Demo2 {
    public void m1(){
        System.out.println("A good day");
        System.out.println("Hello");
    }
    public static void main(String[] args) {
        ABC obj1 = () -> System.out.println("bye");
        obj1.hii();
        Demo2 demo = new Demo2();
        ABC obj2 = demo::m1;  // method refernce
        obj2.hii();// calling

    }
}
```

Modifier and return type isn't restricted in method reference but arguments need to be matched.

by using static and non-static modifier

```
interface A{
    void m1();
}
public class Demo2{
    public static void hii(){
        System.out.println("Domain expansion:infinite void");
    }
    public void bye(){
        System.out.println("Domain expansion: melavolent shrine");
    }
```

```
   public static void main(String[] args) {
    A obj = Demo2::hii;
    obj.m1(); // by using static modifier, domain expansion infinite void
    Demo2 demo = new Demo2();
    A obj2 = demo::bye;
    obj2.m1();  // by using non-static method, domain expansion melavolent shirne
   }
}
```

By using Boolean as modifier

```
interface A{
    void m1();
}
public class Demo2{
    public static void hii(){
        System.out.println("Domain expansion:infinite void");
    }
    public void bye(){
        System.out.println("Domain expansion: melavolent shrine");
    }
    public static boolean baka(){
        System.out.println("run baka");
        return true;
    }
   public static void main(String[] args) {
    A obj = Demo2::hii;
    obj.m1(); // by using static modifier, domain expansion infinite void
    Demo2 demo = new Demo2();
    A obj2 = demo::bye;
    obj2.m1();  // by using non-static method, domain expansion melavolent shirne
    A obj3 = Demo2::baka;
    obj3.m1(); // run baka, The modifier here is boolean
   }
}
```

By using int as modifier

```
interface A{
    int m1(int a , int b);
}
public class Demo2{
    public static int hii(int a, int b){
       return a+b;
    }
    public static int bye(int x,int y){
       return x*y;
    }

   public static void main(String[] args) {
    A a = Demo2::hii;
   System.out.println(a.m1(1,2));  //3
```

```
    A a2 = Demo2::bye;
    System.out.println(a2.m1(1,2));  //2
   }
}
```

To iterate an array with the help of method reference

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class Demo2 {

public static void m1(int a){
    System.out.println(a);
}
public static void main(String[] args) {
    List <Integer> list = Arrays.asList(10,7,12,5);
    Consumer<Integer>  cum = Demo2::m1;
    list.forEach(cum);   // 10,7,12,5 will be the output


}
}
```

Stream API

It is the sequence of objects.

Stream API - predefined classes and interfaces are there and using those things we can perform operations on objects.

We can reduce the lines of code by using stream Api.

There is a class stream, inside that stream class we have methods, it is inside in the package of java.util.Stream.

Functional interface and lambda expression is mandatory here.

Methods

These returns Stream.

Stream method is present inside the collection interface. Stream method converts data into stream format or stream of object. By changing the stream it will not affect the original list. We can also modify it after using the stream().

It doesn’t harm the original array but just make the copy over the original array and we can perform changes in it.

Filter (By using this we can filter-out things, alternative of if statement)

Map()

Min() Max()

Collect() Count()

Distinct(it will remove the duplicate data) Stream(default method)

In collect() it contains some classes like toList(), toSet(), toMap().toList() converts the data into a list format and collect() collects/return it as a list.

<u>Write a program to remove duplicate numbers in a list</u>

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Demo2{
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(5,10,5,3,3,20);
        Stream <Integer> stream = list.stream();
        Stream <Integer> stream2 = stream.distinct();
        List<Integer>removeDuplicateList = stream2.collect( Collectors.toList());
        System.out.println(removeDuplicateList); // 5,10,3,20 , it will remove the
dupllicate value.

        // by trying different way

        List<Integer> list2 = list.stream().distinct().collect(Collectors.toList());
        System.out.println(list2);  // 5,10,3,20 , by doing this we reduced the code by a
single line only

//by trying another way
 list.stream().distinct().forEach(s-> System.out.print(s));  // it will also remove the
duplicate data


    }
}
```

We can perform intermediate operation and terminal operation.

Intermediate operation

Input -> operation 1 -> operation2 –

It will return stream as an object.

Methods

Filter() Map() Distinct() Sorted()

terminal operation.

It will return the output.

Terminal operation -> output

Methods

Collect()

Foreach()

Reduce()

Write a program to print the even numbers

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Demo2{
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1,3,2,5,8,12,9,15);
       list.stream().filter(s->s%2==0).forEach(s->System.out.println(s)); //2,8,12

 // by using for each loop
       for (Integer integer : list) {
        if(integer %2 == 0){
            System.out.println("even number  " + integer);
        }
       }

    }
}
```

WRITE A PROGRAM TO DISPLAY THE DOUBLED NUMBERS

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collector;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Demo2 {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 3, 2, 5, 8, 12, 9, 15, 8, 5, 2, 12, 6);
  List <Integer> list2 = list.stream().map(s-> s*s).collect(Collectors.toList());
  System.out.println(list2);  // will get doubled output
    }
}
```

Write a program in which you have to add the list values by 1 and then filter out the even numbers and then remove the duplicate data and then print the output.

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collector;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Demo2 {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 3, 2, 5, 8, 12, 9, 15, 8, 5, 2, 12, 6);
  List <Integer> list2 = list.stream().map(s-> s+1).filter(s-
>s%2==0).distinct().  collect(Collectors.toList());
  System.out.println(list2);  // first here we added the value by 1 then we filtered out
the even numbers and then we removed the duplicate data so the output will be
[2,4,6,10,16]
    }
}
```

Write a program in which you have to sort out the list and then remove the duplicate data and then filter out the odd numbers and then add 1 to the list and print the result.

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collector;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Demo2 {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 3, 2, 5, 8, 12, 9, 15, 8, 5, 2, 12, 6);
 list.stream().sorted().distinct().filter(s -> s % 2 != 0). map(s->s+1).forEach(s-
>System.out.println(s));;
    }
}
```

Wap in which you have to replace the d

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collector;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Lambda {
    public static void main(String[] args) {
       String[]str = {"domain", "expansion","melavolent","shrine","doraemon"};
       Stream<String> stream = Arrays.stream(str);

       stream.map(p->p.replace("d", " ")).forEach(z->System.out.println(z));

    }
}
```

IntStream

It is present inside of java util.stream; package.

The arguments presents inside of method do return predicate.

Method

range(); it is a static method.

anyMatch(); // will return true if there is any match is found

noneMatch(); // will return true if there is no match found

write a program and check whether there is match or not

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collector;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class Lambda {
    public static void main(String[] args) {
    boolean b= IntStream.range(1, 10).anyMatch(s->s%2==0);
    boolean b1= IntStream.range(1, 10).noneMatch(s->s%2==0);

    System.out.println(b);  //true

    System.out.println(b1);  // false as there is match

    }
}
```

Write a program to check the prime number

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collector;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class Lambda {
    public static void main(String[] args) {
    boolean b= IntStream.range(1, 10).anyMatch(s->s%s==0);
    boolean b1= IntStream.range(1, 10).noneMatch(s->s%s==0);

    if(b){
        System.out.println("not a prime number");
    }
    else{
        System.out.println("prime number");  // prime number
    }




    }
}
```