Intersection of Two Arrays II

Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2,2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [4,9]

Explanation: [9,4] is also accepted.

Number of Uneq[redacted]rray

You are given a **0-indexed** array of positive i[redacted]ber of triplets (i, j, k) that meet the following conditions:

```
int n = 0;
boolean flag =false;
for (int i = 0; i < arr_1.length; i++) {
    for (int k :arr_2 ) {
        if (arr_1[i]!=k) {
                flag= true;
        }
        else{
                flag=false;
                break;
        }
    }
    if (flag==true) {
        str_1[n]=arr_1[i];
        n++;
    }
}

flag=false;
for (int i = 0; i < arr_2.length; i++) {
    for (int k :arr_1 ) {
        if (arr_2[i]!=k) {
                flag= true;
```

Find the Difference of Two Arrays

Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size 2 where:*

- answer[0] *is a list of all* ***distinct*** *integers in* nums1 *which are* ***not*** *present in* nums2.
- answer[1] *is a list of all* ***distinct*** *integers in* nums2 *which are* ***not*** *present in* nums1.

Note that the integers in the lists may be returned in **any** order.

Example 1:

Input: nums1 = [1,2,3], nums2 = [2,4,6]

Output: [[1,3],[4,6]]

Explanation:

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].

Power of Four

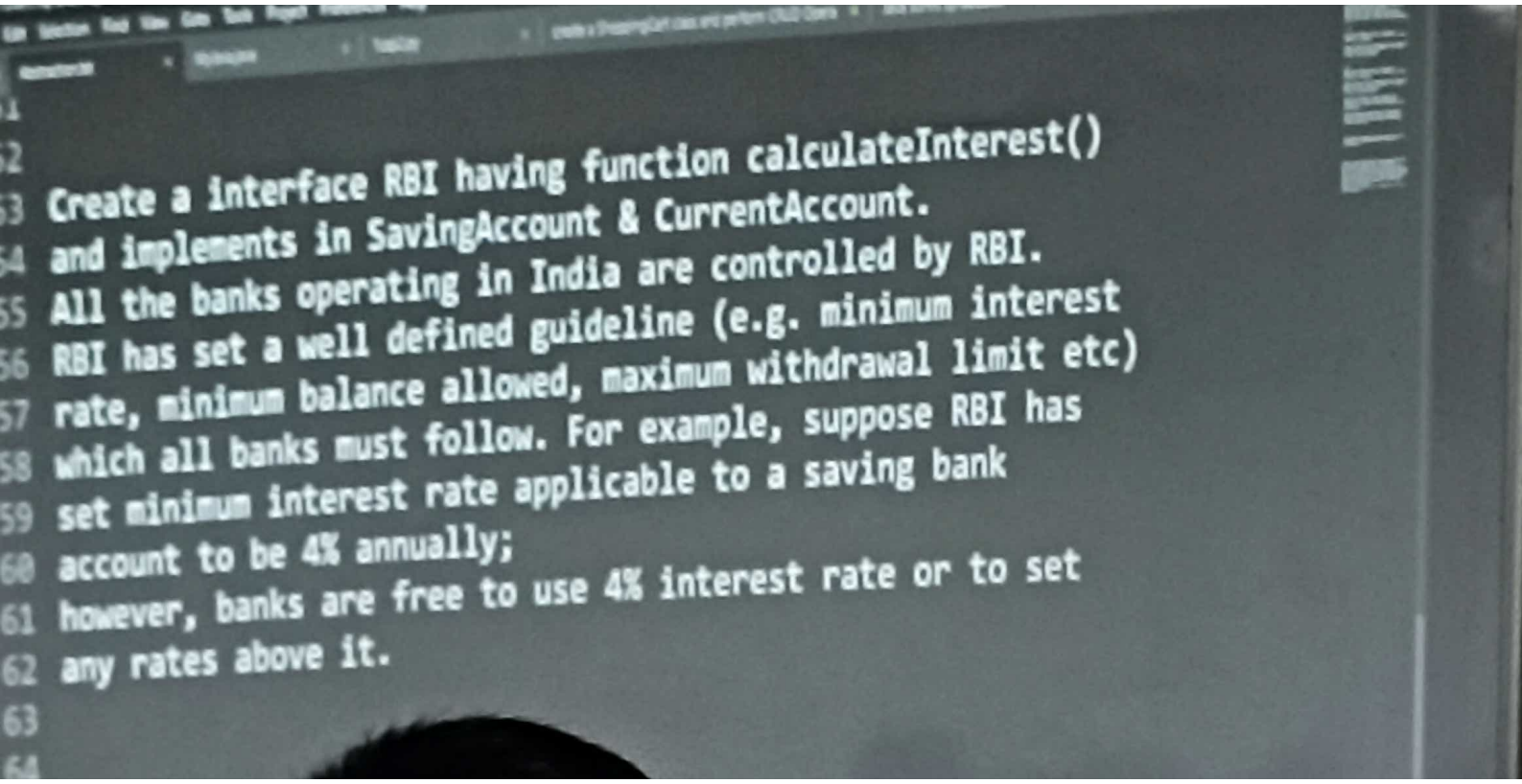
Given an integer n, return *true if it is a power of four. Otherwise, return false.*
An integer n is a power of four, if there exists an integer x such that $n == 4^x$.

Example 1:
Input: n = 16
Output: true

Example 2:
Input: n = 5
Output: false

Example 3:
Input: n =
Output

fference of Two Arrays


Create a interface RBI having function calculateInterest()
and implements in SavingAccount & CurrentAccount.
All the banks operating in India are controlled by RBI.
RBI has set a well defined guideline (e.g. minimum interest
rate, minimum balance allowed, maximum withdrawal limit etc)
which all banks must follow. For example, suppose RBI has
set minimum interest rate applicable to a saving bank
account to be 4% annually;
however, banks are free to use 4% interest rate or to set
any rates above it.

First Bad Version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.
Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.
You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.
Example 2:
Input: n = 1, bad = 1
Output: 1

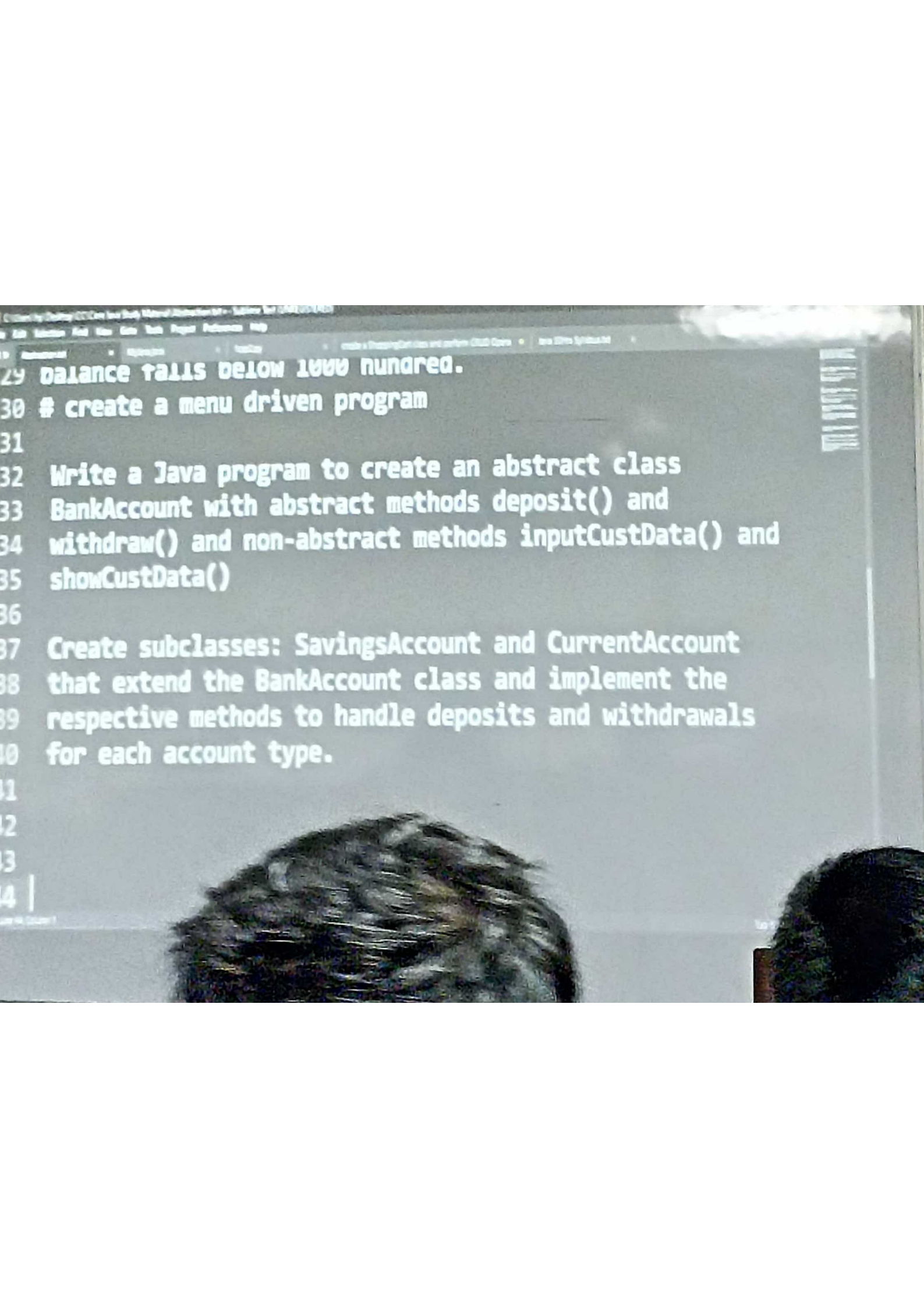
42 Create subclass Interest1, Interest2 class that can give
43 4% interest on SavingsAccounts and 2% interest on
44 CurrentAccounts

balance falls below 1000 hundred.
create a menu driven program

Write a Java program to create an abstract class BankAccount with abstract methods deposit() and withdraw() and non-abstract methods inputCustData() and showCustData()

Create subclasses: SavingsAccount and CurrentAccount that extend the BankAccount class and implement the respective methods to handle deposits and withdrawals for each account type.

```
29 balance falls below 1000 hundred.
30 # create a menu driven program
31
32 Write a Java program to create an abstract class
33 BankAccount with abstract methods deposit() and
34 withdraw() and non-abstract methods inputCustData() and
35 showCustData()
36
37 Create subclasses: SavingsAccount and CurrentAccount
38 that extend the BankAccount class and implement the
39 respective methods to handle deposits and withdrawals
40 for each account type.
41
42
43
44
```



No.	Abstract Class	Interface
5)	A java class is able to extends **only one** abstract class at a time	A java class is able to implements **multiple** interfaces at a time
	Inside abstract class it is possible to declare **methods body** & **constructors**	It is not possible to declare **methods body** & **constructors**
7)	The variables of abstract class need not be **public static final**	The variables declared in interface by default **public static final**

No.	Abstract Class	Interface
1)	It is declared with **abstract** modifier	It is declare by using **interface** keyword
	The abstract allows declaring both abstract & concrete methods	The interface allows declaring only abstract methods
3)	Methods must declare with abstract modifier	Methods are by default **public abstract**
4)	In child class the implementation methods need not be **public** it means while overriding it is possible to declare any valid modifier	In implementation class the implementation methods must be **public**

Abstraction

- The process highlighting the set of services which is required to user and hiding the internal implementation is called abstraction.
- We are achieving abstraction concept by using Abstract classes & Interfaces.
- Bank ATM Screens hide the internal implementation and highlighting set of services like withdraw amount, money transfer, change PIN, ...etc).

Encapsulation

- The process of binding the data(variables) and code(methods) as a single unit is called encapsulation.
- The process of hiding the implementation details to user is called encapsulation.
- We are achieving this concept by declaring variables as a private modifier because it is possible to access private members with in the class only but not outside of that class.

Data Hiding

- The main objective is data hiding is security and it is possible to hide the data by using **private modifier**.
- If any variable declared as a **private** it is possible to access those variables only inside the class is called data hiding.

Fully or Tightly encapsulated class

The class contains only **private properties** that class is said to be tightly encapsulated class.

```
class Emp
{       private int eid;
        private String ename;
}
```

Data Hiding

- The main objective is data hiding is security and it is possible to hide the data by using **private modifier**.
- If any variable declared as a **private** it is possible to access those variables only inside the class is called data hiding.

Fully or Tightly encapsulated class

The class contains only **private properties** that class is said to be tightly encapsulated class.

```
class Emp
{       private int eid;
        private String ename;
}
```

Encapsulation

- The process of binding the data(variables) and code(methods) as a single unit is called encapsulation.
- The process of hiding the implementation details to user is called encapsulation.
- We are achieving this concept by declaring variables as a private modifier because it is possible to access private members with in the class only but not outside of that class.

Abstraction

- The process highlighting the set of services which is required to user and hiding the internal implementation is called abstraction.
- We are achieving abstraction concept by using Abstract classes & Interfaces.
- Bank ATM Screens hide the internal implementation and highlighting set of services like withdraw amount, money transfer, change PIN, ...etc).

No.	Abstract Class	Interface
5)	A java class is able to extends **only one** abstract class at a time	A java class is able to implements **multiple** interfaces at a time
6)	Inside abstract class it is possible to declare **methods body & constructors**	It is not possible to declare **methods body & constructors**
7)	The variables of abstract class need not be **public static final**	The variables declared in interface by default **public static final**

No.	Abstract Class	Interface
1)	It is declared with **abstract** modifier	It is declare by using **interface** keyword
2)	The abstract allows declaring both abstract & concrete methods	The interface allows declaring only abstract methods
3)	Methods must declare with abstract modifier	Methods are by default **public abstract**
4)	In child class the implementation methods need not be **public** it means while overriding it is possible to declare any valid modifier	In implementation class the implementation methods must be **public**

```
//Inside the abstract class we can declare the constructor
//abstract class constructor is executed but object is not created
abstract class Test
{
        Test()  {
                System.out.println("Abstract class constructor");
        }
}
class MyJava extends Test
{
        MyJava()
        {
                System.out.println("Normal class constructor");
        }
        public static void main(String[] args)
        {
                new MyJava();
        }
}
```

```
//If the child class is unable to provide implementation of
//all methods of abstract class then we can declare the child class
//with abstract modifier and complete the remaining method
//implementations in next created child classes.
abstract class Test
{
        abstract void m1();
        abstract void m2();
        abstract void m3();
        void m4(){
                System.out.println("m4 method");
        }
}
abstract class Test1 extends Test
{       void m1(){
        System.out.println("m1 method");
        }
}
abstract class Test2 extends Test1
{       void m2(){
        System.out.println("m2 method");
```

```
File Edit Format View Help
        }
}
abstract class Test2 extends Test1
{       void m2(){
        System.out.println("m2 method");
        }

    s MyJava extends Test2
{
        void m3(){
                System.out.println("m3 method");
        }
        public static void main(String[] args)
        {
        MyJava t = new MyJava();
        t.m1(); t.m2();
        t.m3(); t.m4();
        }
}
```

File Edit Format View Help

```
//Inside the abstract class we declare main method.
abstract class Test
{       public static void main(String[] args)
        {
                System.out.println("Abstract class main");
        }
}
```

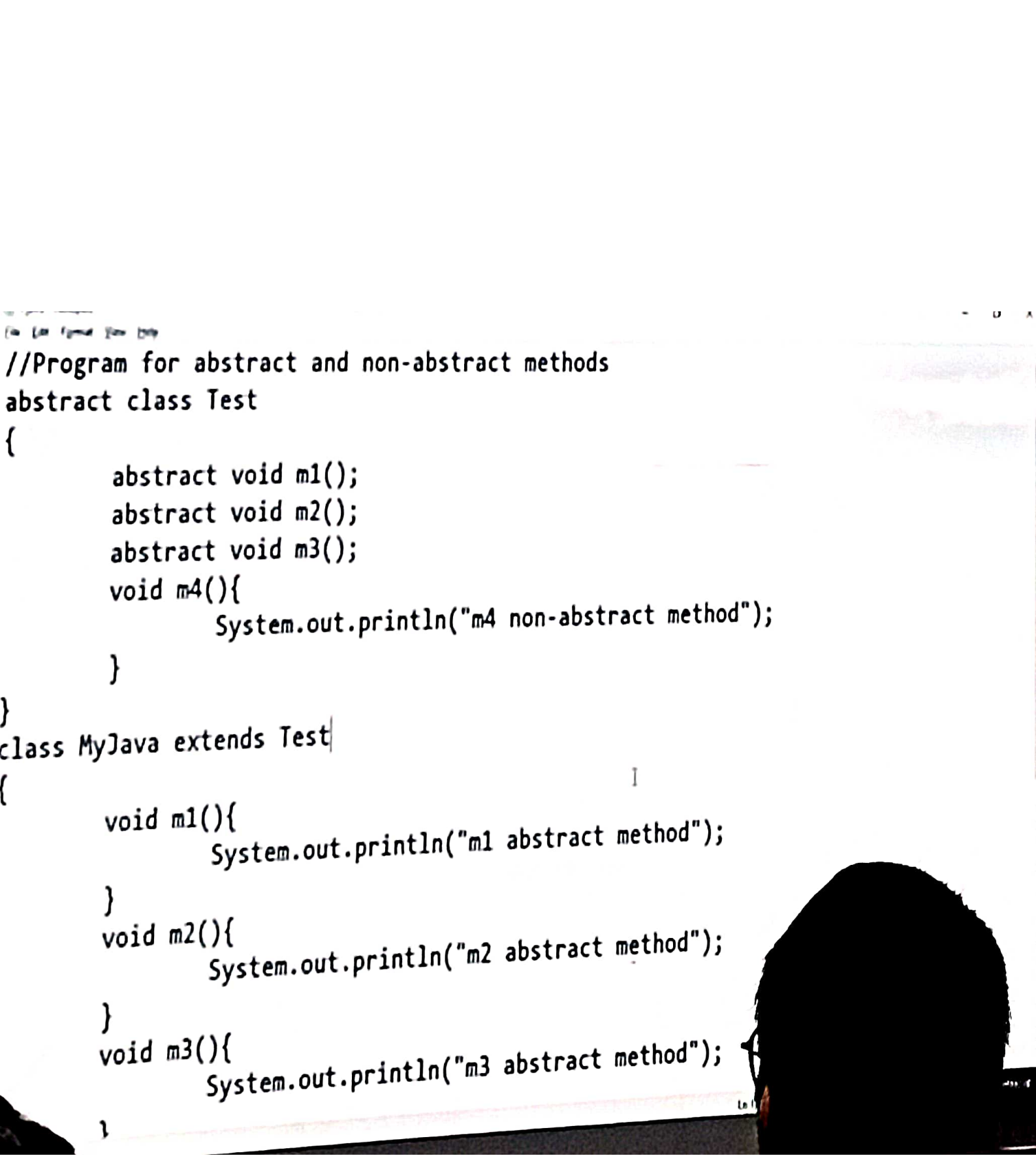
```
//If the child class is unable to provide implementation of
//all methods of abstract class then we can declare the child class
//with abstract modifier and complete the remaining method
//implementations in next created child classes.
abstract class Test
{
        abstract void m1();
        abstract void m2();
        abstract void m3();
        void m4(){
                System.out.println("m4 method");
        }
}
abstract class Test1 extends Test
{       void m1(){
        System.out.println("m1 method");
        }

abstract class Test2 extends Test1
        void m2(){
        System.out.println("m2 method");
```

```
//Program for abstract and non-abstract methods
abstract class Test
{
        abstract void m1();
        abstract void m2();
        abstract void m3();
        void m4(){
                System.out.println("m4 non-abstract method");
        }
}
class MyJava extends Test
{
        void m1(){
                System.out.println("m1 abstract method");
        }
        void m2(){
                System.out.println("m2 abstract method");
        }
        void m3(){
                System.out.println("m3 abstract method");
        }
```

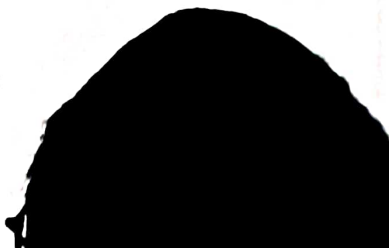
File Edit Format View Help

```
                System.out.println("m1 abstract method");
        }
        void m2(){
                System.out.println("m2 abstract method");
        }
        void m3(){
                System.out.println("m3 abstract method");
        }
        public static void main(String[] args)
        {
                MyJava t = new MyJava();
                t.m1(); t.m2();
                t.m3(); t.m4();

                Test t1 = new MyJava(); //abstract class reference variable Child
class object
                t1.m1(); t1.m2();
                t1.m3(); t1.m4();
        }
}
```

```
//Program for abstract and non-abstract methods
abstract class Test
{
        abstract void m1();
        abstract void m2();
        abstract void m3();
        void m4(){
                System.out.println("m4 non-abstract method");
        }
}
class MyJava extends Test
{
        void m1(){
                System.out.println("m1 abstract method");
        }
        void m2(){
                System.out.println("m2 abstract method");
        }
        void m3(){
                System.out.println("m3 abstract method");
        }
```



```
//Program for abstract and non-abstract methods
abstract class Test
{
        abstract void m1();
        abstract void m2();
        abstract void m3();
        void m4(){
                System.out.println("m4 non-abstract method");
        }
}
class MyJava extends Test
{
        void m1(){
                System.out.println("m1 abstract method");
        }
        void m2(){
                System.out.println("m2 abstract method");
        }
        void m3(){
```



- A class must be prefixed with **abstract** if it has one or more methods with **abstract** keyword known as **abstract class**
- An **abstract** method is only declared but not implemented
- An **abstract class** cannot be instantiated but can be inherited by another class
- Abstract class may contain **abstract** and non **abstract methods**
- The inheriting class must implement all the **abstract** methods or else the subclass should also be declared as **abstract**

```
//Inside the interface it is possible to declare the main method from
java 8.
interface I1
{
        public static void main(String[] args)
        {
        System.out.println("interface main method");
        }
}
```

```
//Inside the interface it is possible to declare the default & static methods from Java8 version.
interface I1
{
        void m1();
        default void m2()               {
                System.out.println("default method");
        }
        static void m3()
        {
                System.out.println("static method");
        }
}
class Test implements I1
{
        public void m1()
        {
        System.out.println("abstract method");
        }
        public static void main(String[] args)
        {
                Test t = new Test();
                t.m1();
                t.m2();
                I1.m3();
        }
```

```
//Inside the interface it is possible to declare the main method from
java 8.
interface I1
{
        public static void main(String[] args)
        {
        System.out.println("interface main method");
        }
}
```

Untitled - Notepad
File Edit Format View Help

```
//If interfaces contains same default method then we can override that method in
implementation class.
interface I1
{       default void m1()       {
                System.out.println("I1 m1() method");
        }
}
interface I2
{       default void m1()       {
                System.out.println("I2 m1() method");
        }
}
class Test implements I1,I2
{       public void m1()        {
                System.out.println("m1 method common implementation");
        }
        public static void main(String[] args)
        {
                new Test().m1();
        }
```

```
//Inside the interface it is possible to declare the default & static methods from java8 version.
interface I1
{
	void m1();
	default void m2()
	{
		System.out.println("default method");
	}
	static void m3()
	{
		System.out.println("static method");
	}
}
class Test implements I1
{
	public void m1()
	{
	System.out.println("abstract method");
	}
	public static void main(String[] args)
	{
		Test t = new Test();
		t.m1();
		t.m2();
		I1.m3();
	}
}
```

```
//If interfaces contains same default method then we can override that method in
implementation class.
interface I1
{       default void m1()       {
                System.out.println("I1 m1() method");
        }

interface I2
{       default void m1()       {
                System.out.println("I2 m1() method");
        }
}
class Test implements I1,I2
{       public void m1()        {
                System.out.println("m1 method common implementation");
        }
        public static void main(String[] args)
        {
                new Test().m1();
        }
}
```

```
/Inside the interface it is possible to declare the default & static methods from java8 version.
nterface I1

        void m1();
        default void m2()
        {
                System.out.println("default method");
        }
        static void m3()
        {
                System.out.println("static method");
        }

lass Test implements I1

        public void m1()
        {
        System.out.println("abstract method");
        }
        public static void main(String[] args)
        {
                Test t = new Test();
                t.m1();
                t.m2();
                I1.m3();
        }
```

Marker interface:

The interface does not contain any methods but whenever our class implements that interface, our class must acquire some capabilities to perform some operations provided by JVM, such type of interfaces are called **marker interfaces**.

java.io.Serializable

provides serialization capabilities

java.lang.Cloneable

provides cloning capabilities

java.util.RandomAccess

data accessing capabilities

File Edit Format View Help

```
//Program to achieve multiple inheritance
interface Abc
{
  public void show();
}
class Pqr
{
      tic int i=20;
}
class Xyz extends Pqr implements Abc {
    public void show()
    {
      System.out.println("I m in abc");
    }

    public static void main(String[] args) {
        Abc a;
        Xyz z=new Xyz();
        a=z;
        a.show();
        System.out.println(i);
    }
}
```

1.java - Notepad

File Edit Format View Help

```
        {
                System.out.println("m1 2-arguments method");
                return 22.2;
        }
        public int m1(char ch)
        {
                System.out.println("m1 1-argument method");
                return 40;
        }

        public static void main(String[] args)
        {
                Test t = new Test();
                int i = t.m1('p');
                System.out.println("Value="+i);
                double d = t.m1(22,"Raj");
                System.out.println("Value="+d);
        }
}
```

```
//program for overloading interface methods
interface It1
{
        double m1(int a,String str);
        int m1(char ch);
}
cl      est implements It1
{       //overloaded methods
        public double m1(int a,String str)
        {
                System.out.println("m1 2-arguments method");
                return 22.2;
        }
        public int m1(char ch)
        {
                System.out.println("m1 1-argument method");
                return 40;
        }

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

```
//Nested interface: Declaring interface inside the another interface
interface I1
{       void m1();
        interface I2
        {
                void m2();
        }

ass Test implements I1,I1.I2
{
        public void m1(){
                System.out.println("m1 method");
        }
        public void m2(){
                System.out.println("m2 method");
        }
        public static void main(String[] args)
        {
                Test t = new Test();
                t.m1();
                t.m2();
        }
```

```
//Program to achieve multiple inheritance
interface Abc
{
  public void show();
}
class Pqr
{
    static int i=20;
}
class Xyz extends Pqr implements Abc {
    public void show()
    {
      System.out.println("I m in abc");
    }

    public static void main(String[] args) {
        Abc a;
        Xyz z=new Xyz();
        a=z;
        a.show();
        System.out.println(i);
    }
}
```

```
//Nested interface: Declaring interface inside the another interface
interface I1
{       void m1();
        interface I2
        {
                void m2();
        }
};
class Test implements I1,I1.I2
{
        public void m1(){
                System.out.println("m1 method");
        }
        public void m2(){
                System.out.println("m2 method");
        }
        public static void main(String[] args)
        {
                Test t = new Test();
                t.m1();
                t.m2();
        }
```

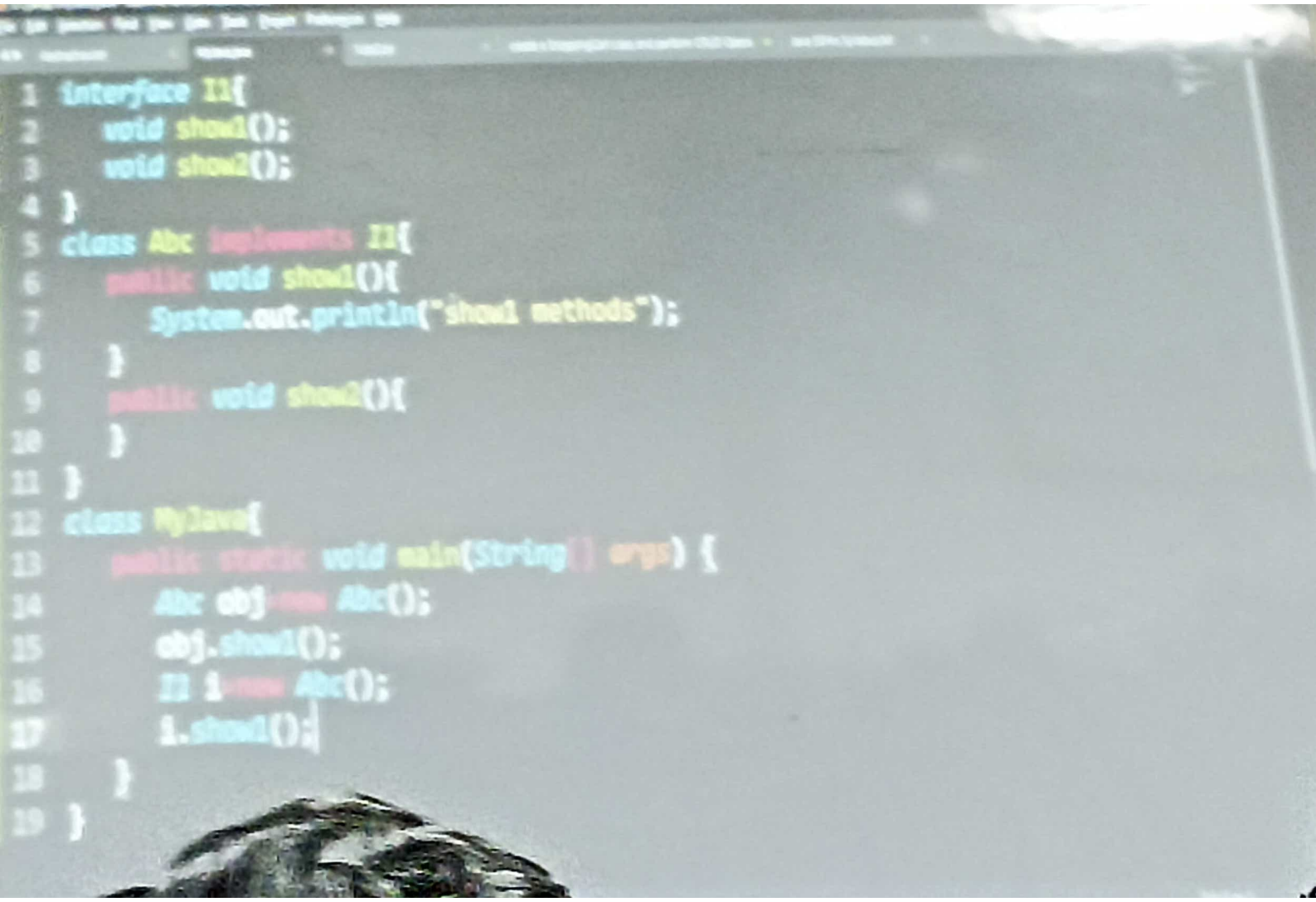
```
    {
        System.out.println("m1 2-arguments method");
        return 22.2;
    }
    public int m1(char ch)
    {
        System.out.println("m1 1-argument method");
        return 40;
    }

    public static void main(String[] args)
    {
        Test t = new Test();
        int i = t.m1('p');
        System.out.println("Value="+i);
        double d = t.m1(22,"Raj");
        System.out.println("Value="+d);
    }
}
```

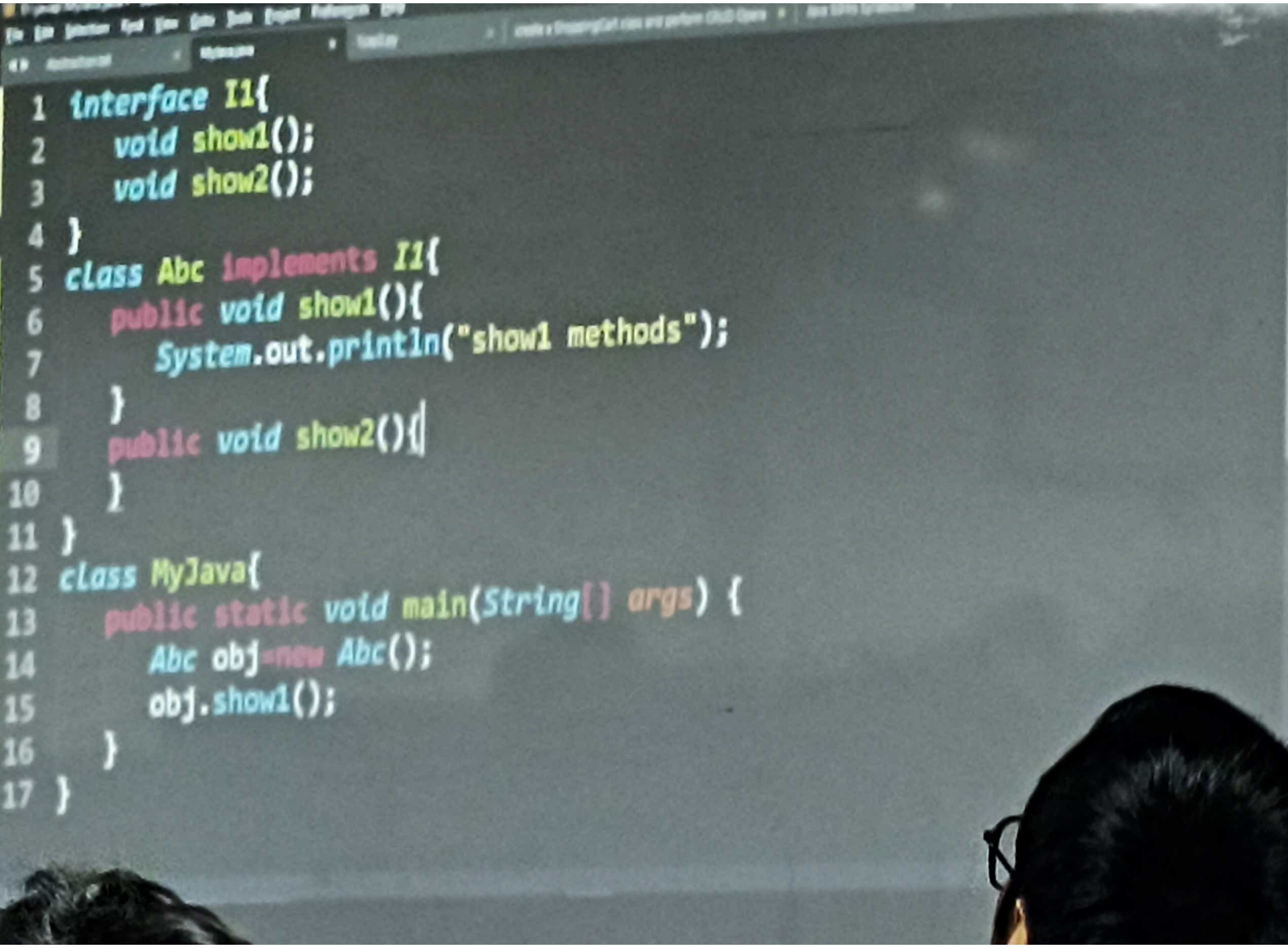
```
//program for overloading interface methods
interface It1
{
        double m1(int a,String str);
        int m1(char ch);
}
class Test implements It1
{       //overloaded methods
        public double m1(int a,String str)
        {
                System.out.println("m1 2-arguments method");
                return 22.2;
        }
        public int m1(char ch)
        {
                System.out.println("m1 1-argument method");
                return 40;
        }
```

```
class MHB implements RBI
{
        public void findInterest(){
                System.out.println("MHB interest logic implementation");
        }
}
class Banking
{
        public static void main(String[] args)
        {
                RBI r1,r2,r3;
                r1=new SBI();
                r2=new PNB();
                r3=new MHB();
                r1.findInterest();
                r2.findInterest();
                r3.findInterest();
        }
```

```
//implementing interface in multiple classes
interface RBI
{
        void findInterest();
}
class SBI implements RBI
{
        public void findInterest(){
                System.out.println("SBI interest logic implementation");
        }

class PNB implements RBI

        public void findInterest(){
                System.out.println("PNB interest logic implementation");
        }

lass MHB implements RBI

        public void findInterest(){
                System.out.println("MHB interest logic
```

```
//implementing interface
interface I1
{
        // public, static and final
        int a = 10;
        // public and abstract methods
        void m1();
        void m2();
}
class Test implements I1
{
        public void m1(){
                System.out.println("m1 method implementation");
                System.out.println(a);
        }
        public void m2(){
                System.out.println("m2 method implementation");
                System.out.println(a);
        }
```


interface I1{
void show1();
void show2();
}
class Abc implements I1{
public void show1(){
System.out.println("show1 methods");
}
public void show2(){
}
}
class Mylava{
public static void main(String[] args) {
Abc obj=new Abc();
obj.show1();
I1 i=new Abc();
i.show1();
}
}


interface I1{
void show1();
void show2();
}
class Abc implements I1{
public void show1(){
System.out.println("show1 methods");
}
public void show2(){
}
}
class MyJava{
public static void main(String[] args) {
Abc obj=new Abc();
obj.show1();
}
}

Important points:

- It is not possible to create the object of interface because interfaces are by default **abstract**
- All methods in interfaces are by default **public** and **abstract**
- Inside the interface **constructor** declaration is not allowed
- A class can extends only one class but one interface can extends more than one interface
- Interface contains only **static fields** but not instance fields
- Interface is implemented by class using **implements** keyword

Syntax:

```
interface interface_name
{
    public static final variable declaration;
    abstract method declaration;
}
```

Ex:

```
interface I1
{
    void m1();
    void m2();
    void m3();
}
```

- When we have multiple solutions to solve the same problem in that case we should go for **interface**
- Interfaces are syntactically similar to classes and their methods are declared without any body

Why do we use interface?

- It is used to achieve **100% abstraction**
- Java does not support **multiple inheritance** in case of class, but by using **interface** it can achieve **multiple inheritance**
- It is also used to achieve **loose coupling**
- Variables in interface are **public, static** and **final**

```
Movie Ticket Booking Problem Statement
-------------------------------------------------------------
->Provide Admin Login Facility with username & password, then only admin
can do all task
->Provide User Registration & Login Facility with username & password,
then only user can do all task
-> User registration data(name, phone, address, username, password)
-> Assign unique ID to identify each records
-----------------------------------------------

    Admin Login
        Add list of movies
        Select Movie that is Avaliable
                Select a movie which shows is going on now. Ticket booking
                will be started only for this movie
        Buy a Ticket
                Admin can book ticket on ticket_counter ,  if seat is
                already booked show message "Seat already booked"
        Show Ticket (status of ticket)
                Show booked ticket status with seat_number,
```

booked

Logout

User Login

Buy a Ticket

User can book ticket on ticket_counter, if seat is already booked show message "Seat already booked"

Show Ticket (status of ticket)

Show booked ticket status with seat_number,

Cancel Ticket

After cancellation of ticket, show paid amount will be refund with 20% deduction

Logout

Exit

```
-> Assign unique ID to identify each records
-----------------------------------------------

    Admin Login
        Add list of movies
        Select Movie that is Avaliable
                Select a movie which shows is going on now. Ticket booking
                will be started only for this movie
        Buy a Ticket
                Admin can book ticket on ticket_counter ,  if seat is
                already booked show message "Seat already booked"
        Show Ticket (status of ticket)
                Show booked ticket status with seat_number,
        Cancel Ticket
                After cancellation of ticket, same seat number can be
                booked
        Logout
-------------------------
    User Login
        Buy a Ticket
```

Problem statement(1).txt - Notepad

File Edit Format View Help

Walmart Labs

1)Longest consecutive subsequence

Given an array of positive integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Example 1 :

Input:

N = 7

a[] = {2,6,1 ,9,4,5,3}

Output:

6

Explanation:

The consecutive numbers here

are 1 , 2, 3, 4, 5, 6. These 6

numbers form the longest consecutive

Subsquence.

Walmart Labs

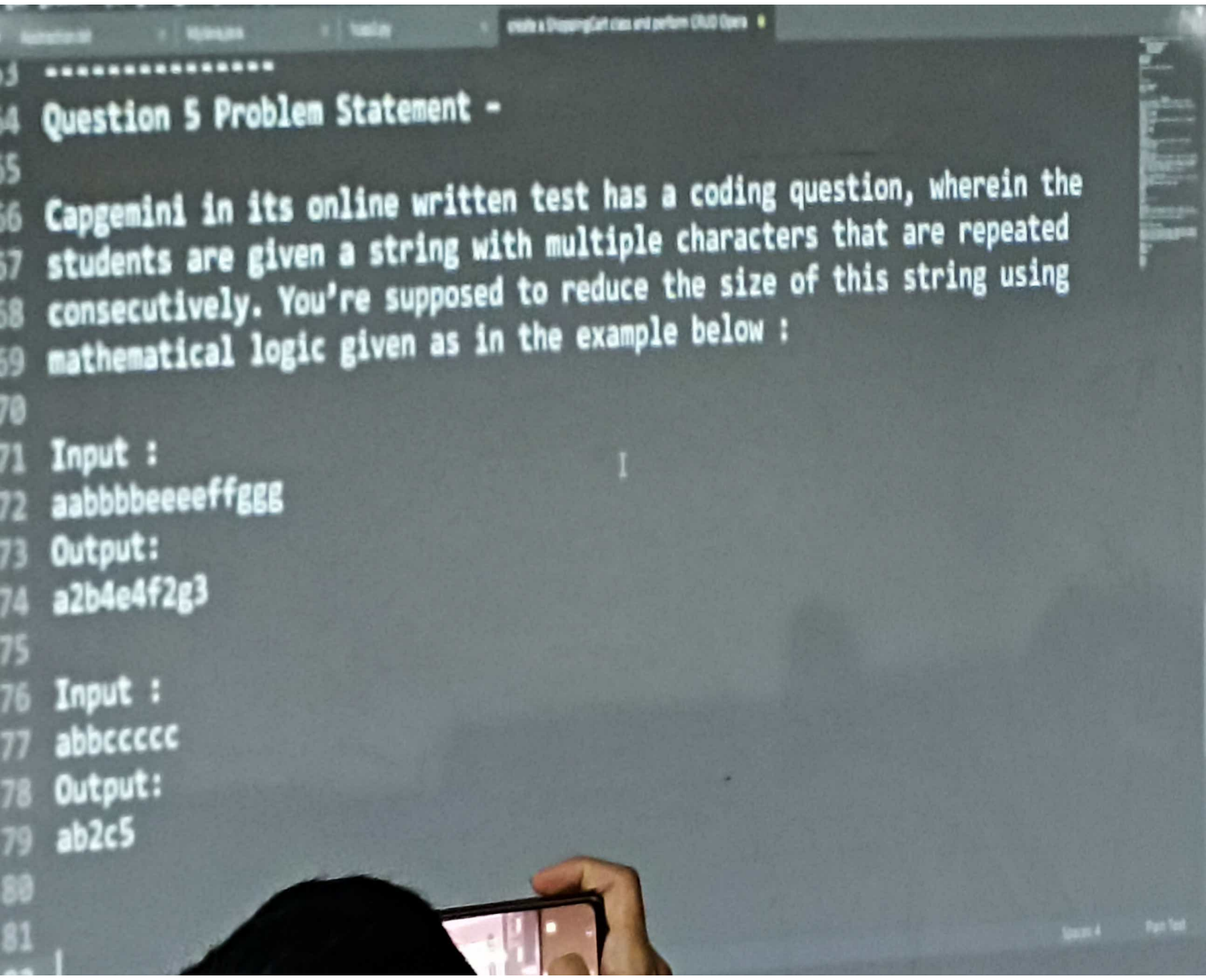
1)Longest consecutive subsequence
Given an array of positive integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Example 1 :
Input:
N = 7
a[] = {2,6,1 ,9,4,5,3}
Output:
6
Explanation:
The consecutive numbers here
are 1 , 2, 3, 4, 5, 6. These 6
numbers form the longest consecutive
Subsquence.

Capgemini in its online written test has a coding question, wherein the students are given a string with multiple characters that are repeated consecutively. You're supposed to reduce the size of this string using mathematical logic given as in the example below :

Input :
aabbbbeeeeffggg
Output:
a2b4e4f2g3

Input :
abbccccc
Output:
ab2c5


Question 5 Problem Statement -
Capgemini in its online written test has a coding question, wherein the students are given a string with multiple characters that are repeated consecutively. You're supposed to reduce the size of this string using mathematical logic given as in the example below :
Input :
aabbbbeeeeffggg
Output:
a2b4e4f2g3
Input :
abbccccc
Output:
ab2c5

java.lang.Runtime

- It is used to interact with java runtime environment
- **Runtime** class is provide the facility to execute a process, to call garbage collector, to check free memory & total memory

We can call garbage collector in two ways

- **System** class gc() method
- **Runtime** class gc() method

- **Runtime** class gc() method is a instance method it is directly interact with garbage collector
- **System** class gc() method is static method it is internally calling **Runtime** class gc() method to call the garbage collector

```
//opening notepad, shutdown the system and restart the system by using
Runtime class
class Test
{
        public static void main(String[] args) throws Exception
        {
        Runtime.getRuntime().exec("notepad");
//      Runtime.getRuntime().exec("shutdown -s -t 0");
//      Runtime.getRuntime().exec("shutdown -r -t 0");
        }
}
```

Important Points

- To call the garbage collector explicitly use **gc()** method it is a static method of **System** class
- The **finalize()** method present in **Object** class & it is called by garbage collector just before destroying the object
- If we are overriding **finalize()** method then our class **finalize()** method is executed
- If we are not overriding **finalize()** method then **Object** class **finalize()** method is executed and the **Object** class **finalize()** method is having empty implementation

```
//By anonymous Object (Nameless object)
class Test
{
        Test()
        {
                System.out.print("Hello");
        }
        public static void main(String[] args)
        {
                new Test();
                System.gc();
        }
}
```

- The process of destroying unreferenced objects is called **garbage collector**.
- When object is unreferenced it is considered as unused object so that JVM automatically destroy that unused object.
- In C language, we allocate memory by using **malloc()** and destroy that memory by using **free()**, here the developer is responsible for both operations.
- In CPP language, we allocate memory by using **constructors** and we destroy that memory by using **destructors**, here the developer is responsible for both operations.

```
//Whenever we are reassigning the reference variable,
//then objects are automatically eligible for GC.
class Test
{
        public static void main(String[] args)
        {
                StringBuffer s1 = new StringBuffer("Mohit");
                StringBuffer s2 = new StringBuffer("Rajat");
                s1 = s2;
                System.out.println(s1);
                System.out.println(s2);
        }
}
```

- The process of destroying unreferenced objects is called **garbage collector.**
- When object is unreferenced it is considered as unused object so that JVM automatically destroy that unused object.
- In C language, we allocate memory by using **malloc()** and we destroy that memory by using **free(),** here the developer is responsible for both operations.
- In CPP language, we allocate memory by using **constructors** and we destroy that memory by using **destructors**, here the developer is responsible for both operations.
- In java, developer is responsible to allocate the memory by creating of **object** and memory will be destroyed by **garbage collector** and it is a part of the JVM

```
//Whenever we are assigning null constants to objects then objects
//are eligible for Garbage Collector
class Test
{
        public static void main(String[] args)
        {
        Test t1=new Test();
        Test t2=new Test();
        System.out.println(t1);
        System.out.println(t2);
        t1=null;        //t1 object is eligible for Garbage Collector
        t2=null;        //t2 object is eligible for Garbage Collector
        System.out.println(t1);
        System.out.println(t2);
        }
}
```

- The process of destroying unreferenced objects is called **garbage collector**.
- When object is unreferenced it is considered as unused object so that JVM automatically destroy that unused object.

```
//opening notepad, shutdown the system and restart the system by using
Runtime class
class Test
{
        public static void main(String[] args) throws Exception
        {
        Runtime.getRuntime().exec("notepad");
//      Runtime.getRuntime().exec("shutdown -s -t 0");
//      Runtime.getRuntime().exec("shutdown -r -t 0");
        }
}
```

Important Points

- To call the garbage collector explicitly use **gc()** method it is a static method of **System** class
- The **finalize()** method present in **Object** class & it is called by garbage collector just before destroying the object
- If we are overriding **finalize()** method then our class **finalize()** method is executed
- If we are not overriding **finalize()** method then **Object** class **finalize()** method is executed and the **Object** class **finalize()** method is having empty implementation

java.lang.Runtime

- It is used to interact with java runtime environment
- Runtime class is provide the facility to execute a process, to call garbage collector, to check free memory & total memory

We can call garbage collector in two ways

- System class gc() method
- Runtime class gc() method

- Runtime class gc() method is a instance method it is directly interact with garbage collector
- System class gc() method is static method it is internally calling Runtime class gc() method to call the garbage collector

```
//By anonymous Object (Nameless object)
class Test
{
        Test()
        {
                System.out.print("Hello");
        }
        public static void main(String[] args)
        {
                new Test();
                System.gc();
        }
}
```

```
//By anonymous Object (Nameless object)
class Test
{
        Test()
        {
                System.out.print("Hello");
        }
        public static void main(String[] args)
        {
                new Test();
                System.gc();
        }
}
```

```
//Whenever we are assigning null constants to objects then objects
//are eligible for Garbage Collector
class Test
{
        public static void main(String[] args)
        {
        Test t1=new Test();
        Test t2=new Test();
        System.out.println(t1);
        System.out.println(t2);
        t1=null;          //t1 object is eligible for Garbage Collector
        t2=null;          //t2 object is eligible for Garbage Collector
        System.out.println(t1);
        System.out.println(t2);
        }
```

- The process of destroying unreferenced objects is called **garbage collector**.
- When object is unreferenced it is considered as unused object so that JVM automatically destroy that unused object.
- In C language, we allocate memory by using **malloc()** and we destroy that memory by using **free()**, here the developer is responsible for both operations.
- In CPP language, we allocate memory by using **constructors** and we destroy that memory by using **destructors**, here the developer is responsible for both operations.

- The process of destroying unreferenced objects is called **garbage collector**.
- When object is unreferenced it is considered as unused object so that JVM automatically destroy that unused object.
- In C language, we allocate memory by using **malloc()** and we destroy that memory by using **free()**, here the developer is responsible for both operations.
- In CPP language, we allocate memory by using **constructors** and we d[redacted]ory by using **destructors**, here the developer is [redacted]perations.

Accenture:-

1 Problem Description :

The function accepts two positive integers ‘r’ and ‘unit’ and a positive integer array ‘arr’ of size ‘n’ as its argument ‘r’ represents the number of rats present in an area, ‘unit’ is the amount of food each rat consumes and each ith element of array ‘arr’ represents the amount of food present in ‘i+1’ house number, where 0 <= i

Note:

- Return -1 if the array is null
- Return 0 if the total amount of food from all houses is not sufficient for all the rats.
- Computed values lie within the integer range.

Example:

Input:

- r: 7
- unit: 2
- n: 8
- arr: 2 8 3 5 7 4 1 2

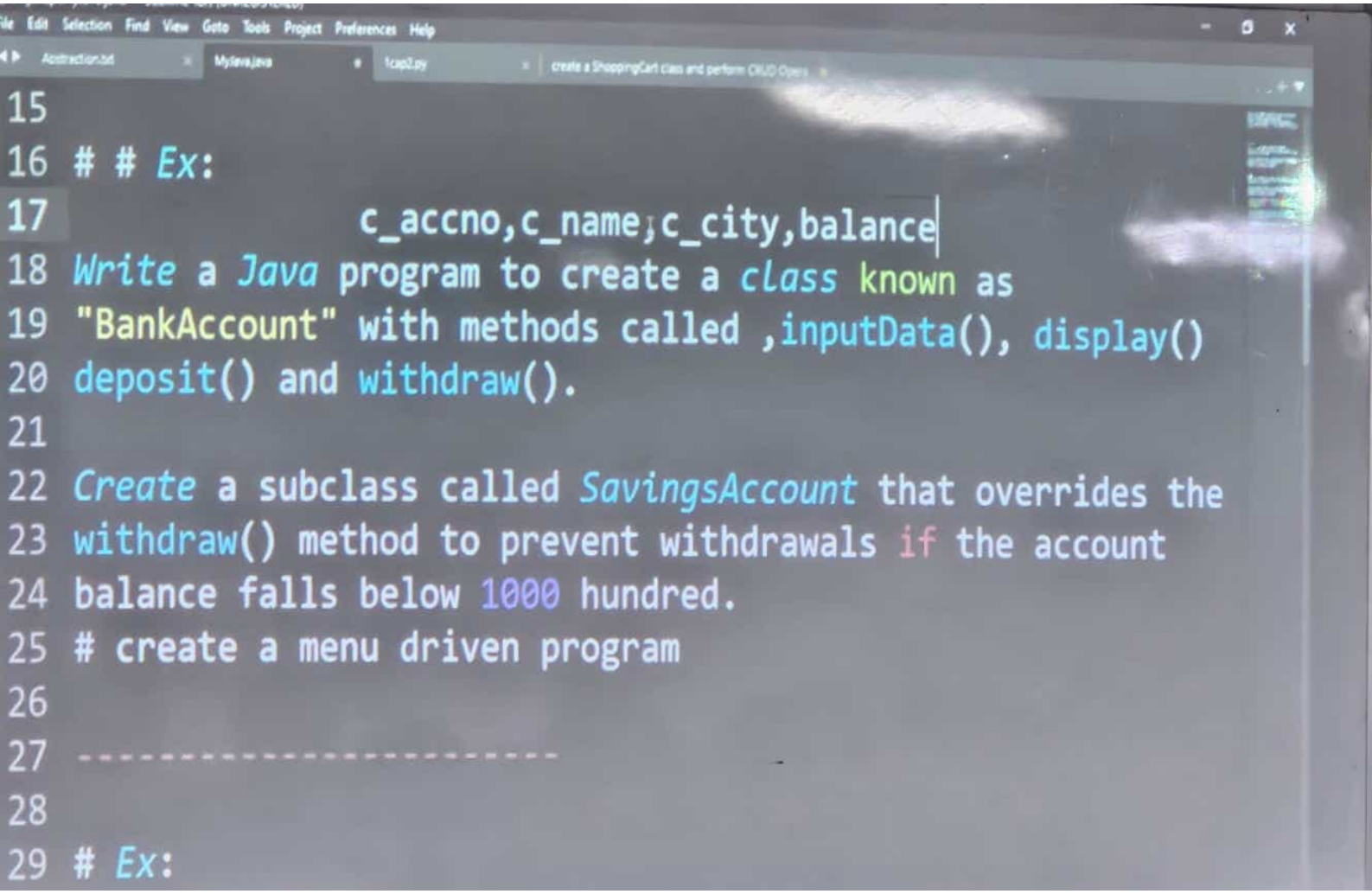
Output:

4

Explanation:

Total amount of food required for all rats = r * unit = 7 * 2 = 14.

The amount of food in 1st houses = 2+8+3+5 = 18. Since, amount of food in 1st 4 houses is sufficient for all the rats. Thus, output is 4.

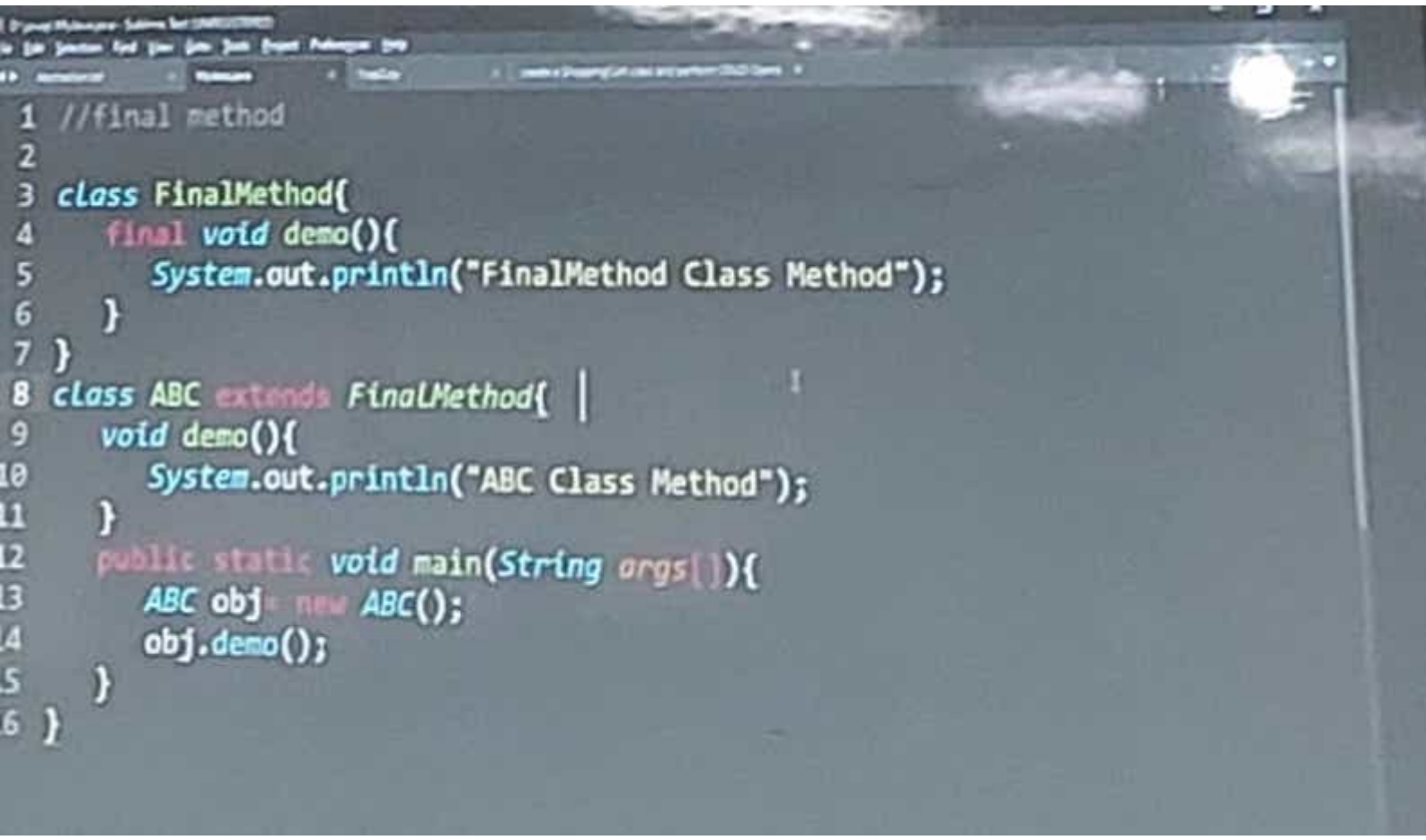

File Edit Selection Find View Goto Tools Project Preferences Help
Abstraction.txt
Mylava.java
1cap2.py
create a ShoppingCart class and perform CRUD Opers
15
16 # # Ex:
17 c_accno,c_name,c_city,balance
18 Write a Java program to create a class known as
19 "BankAccount" with methods called ,inputData(), display()
20 deposit() and withdraw().
21
22 Create a subclass called SavingsAccount that overrides the
23 withdraw() method to prevent withdrawals if the account
24 balance falls below 1000 hundred.
25 # create a menu driven program
26
27 ----------
28
29 # Ex:

```
15
16 # # Ex:
17                  c_accno,c_name;c_city,balance
18 Write a Java program to create a class known as
19 "BankAccount" with methods called ,inputData(), display()
20 deposit() and withdraw().
21
22 Create a subclass called SavingsAccount that overrides the
23 withdraw() method to prevent withdrawals if the account
24 balance falls below 1000 hundred.
25 # create a menu driven program
26
27 ************************
28
29 # Ex:
```

Program to check if a given number is a strong number or not is discussed here. A strong number is a number in which the **sum of the factorial of the digits is equal to the number itself.**

123 = 1! + 2! + 3! = 1 + 2 + 6 = 9 

145 = 1! + 4! + 5! = 1 + 24 + 120 = 145 


//final method
class FinalMethod{
final void demo(){
System.out.println("FinalMethod Class Method");
}
}
class ABC extends FinalMethod{
void demo(){
System.out.println("ABC Class Method");
}
public static void main(String args[]){
ABC obj = new ABC();
obj.demo();
}
}

```java
//Program to use of final variable
class FinalDemo{
    final int MAX=89;
    void me(){
        MAX=110;
    }
    public static void main(String args[]){
        FinalDemo obj=new  FinalDemo();
        obj.me();
    }
}
```

```
//final method

class FinalMethod{
   final void demo(){
      System.out.println("FinalMethod Class Method");
   }
}

class ABC extends FinalMethod{
   void demo(){
      System.out.println("ABC Class Method");
   }

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

final variable

final variables are the constants which cannot change the value of a **final** variable once it is initialized.

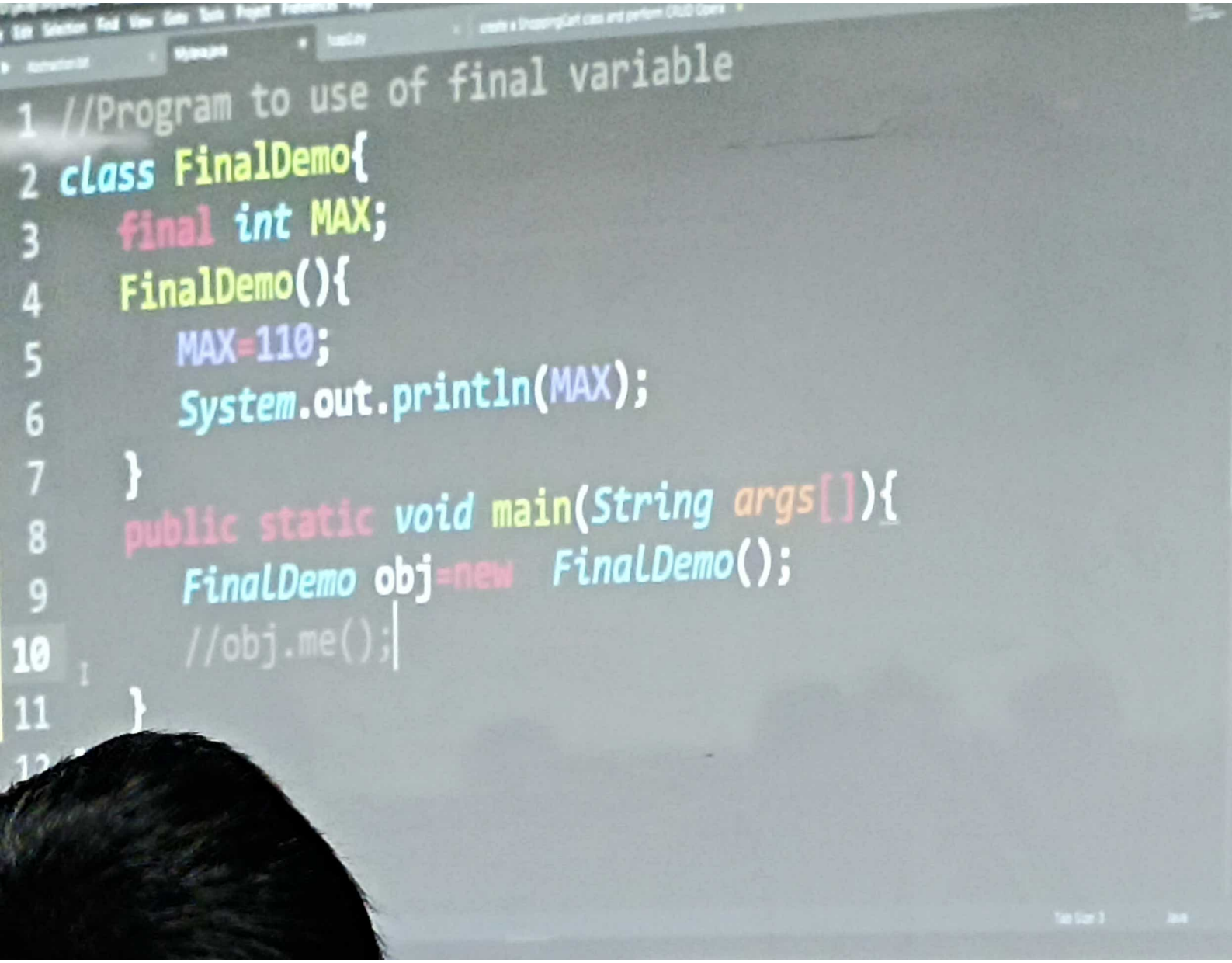
final method

A **final** method cannot be overridden which means even though a subclass can call the final method of parent class but we cannot override it.

final class

A class is declared as **final** then this class cannot be inherited.

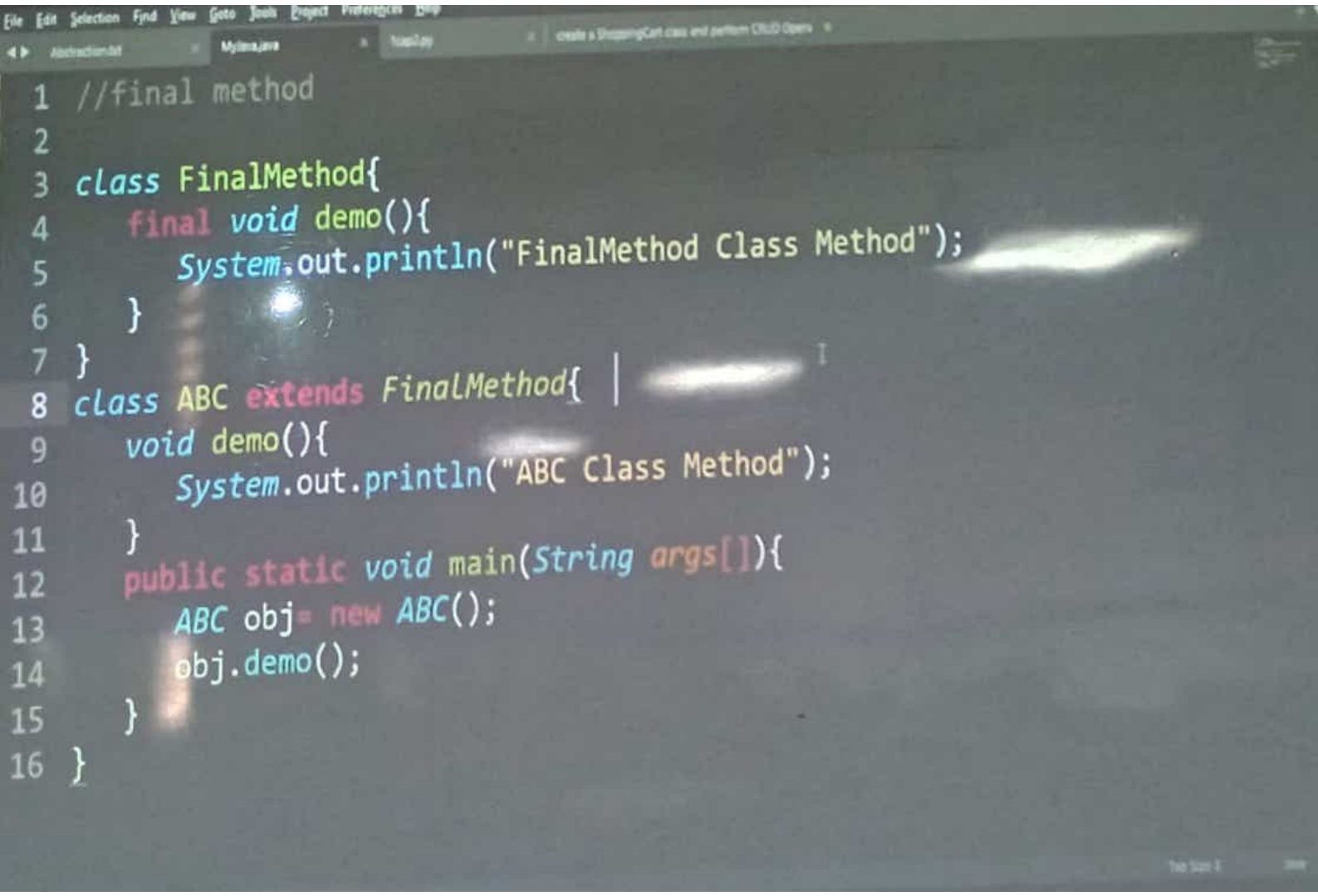
```
//final class
final class A{
}

class B extends A{
   void demo(){
      System.out.println("I am in A");
   }
   public static void main(String args[]){
      B obj= new B();
      obj.demo();
   }
}
```

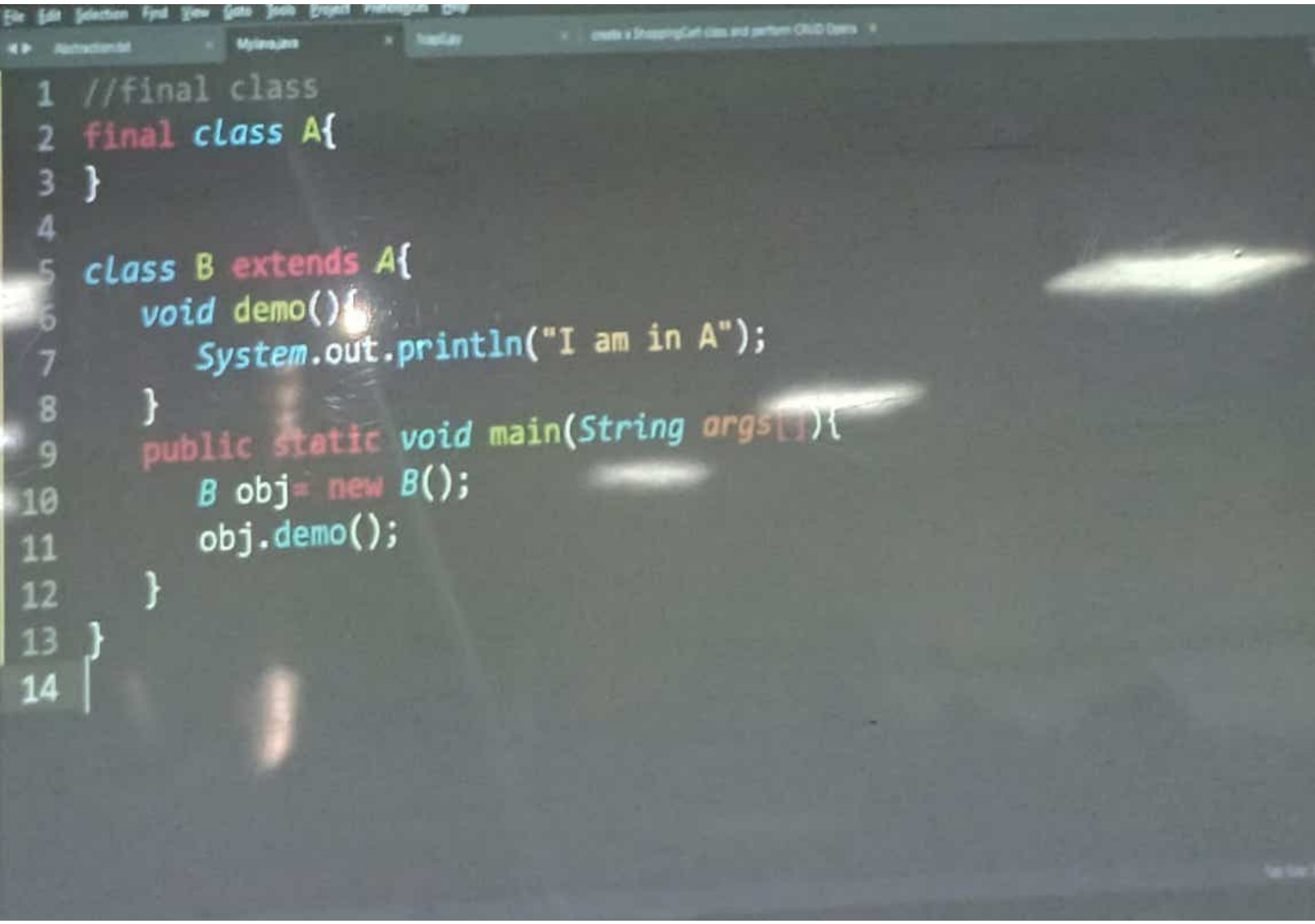

//Program to use of final variable
class FinalDemo{
final int MAX;
FinalDemo(){
MAX=110;
System.out.println(MAX);
}
public static void main(String args[]){
FinalDemo obj=new FinalDemo();
//obj.me();
}

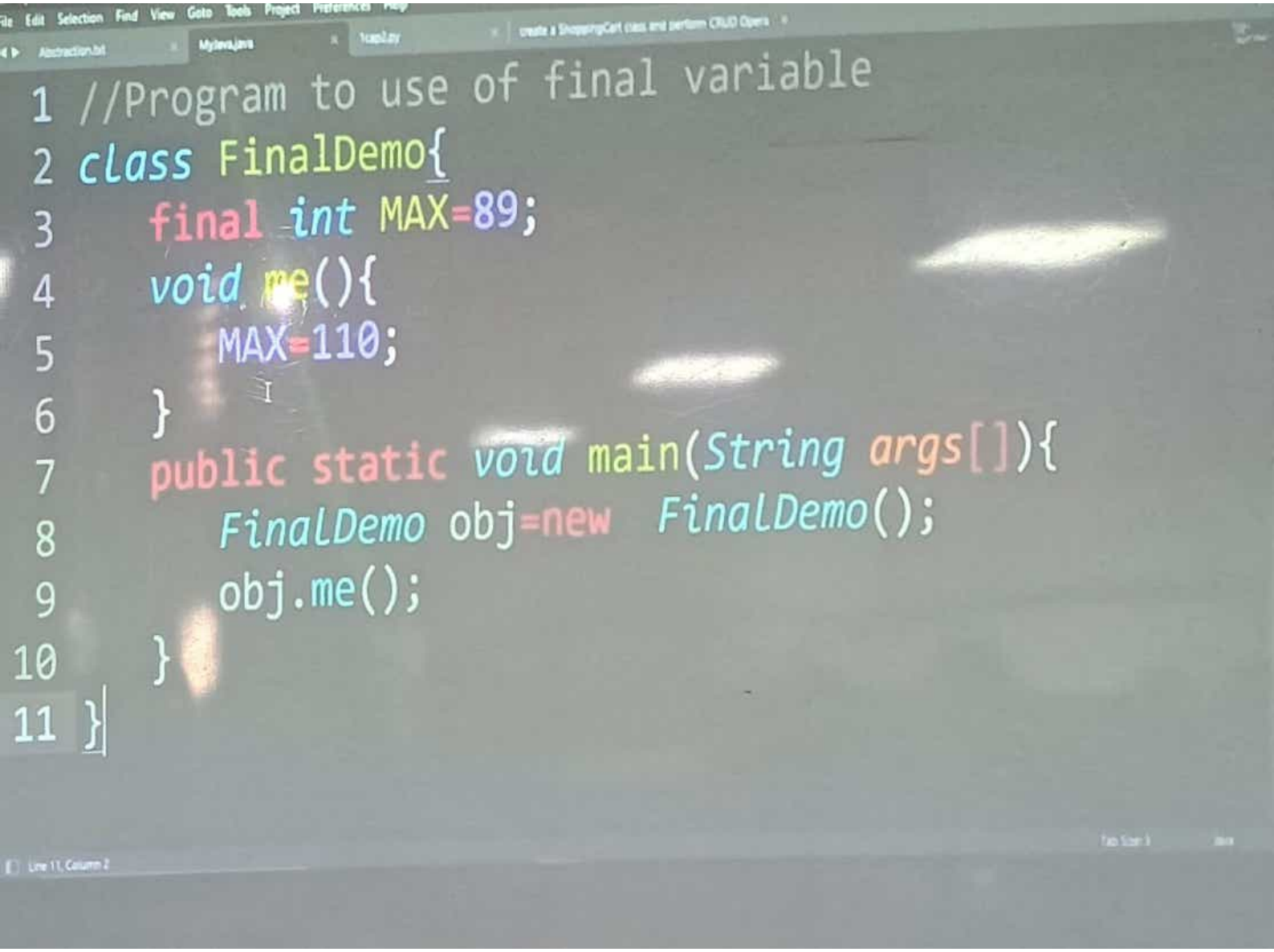
The **final** keyword in java is used to restrict the user.

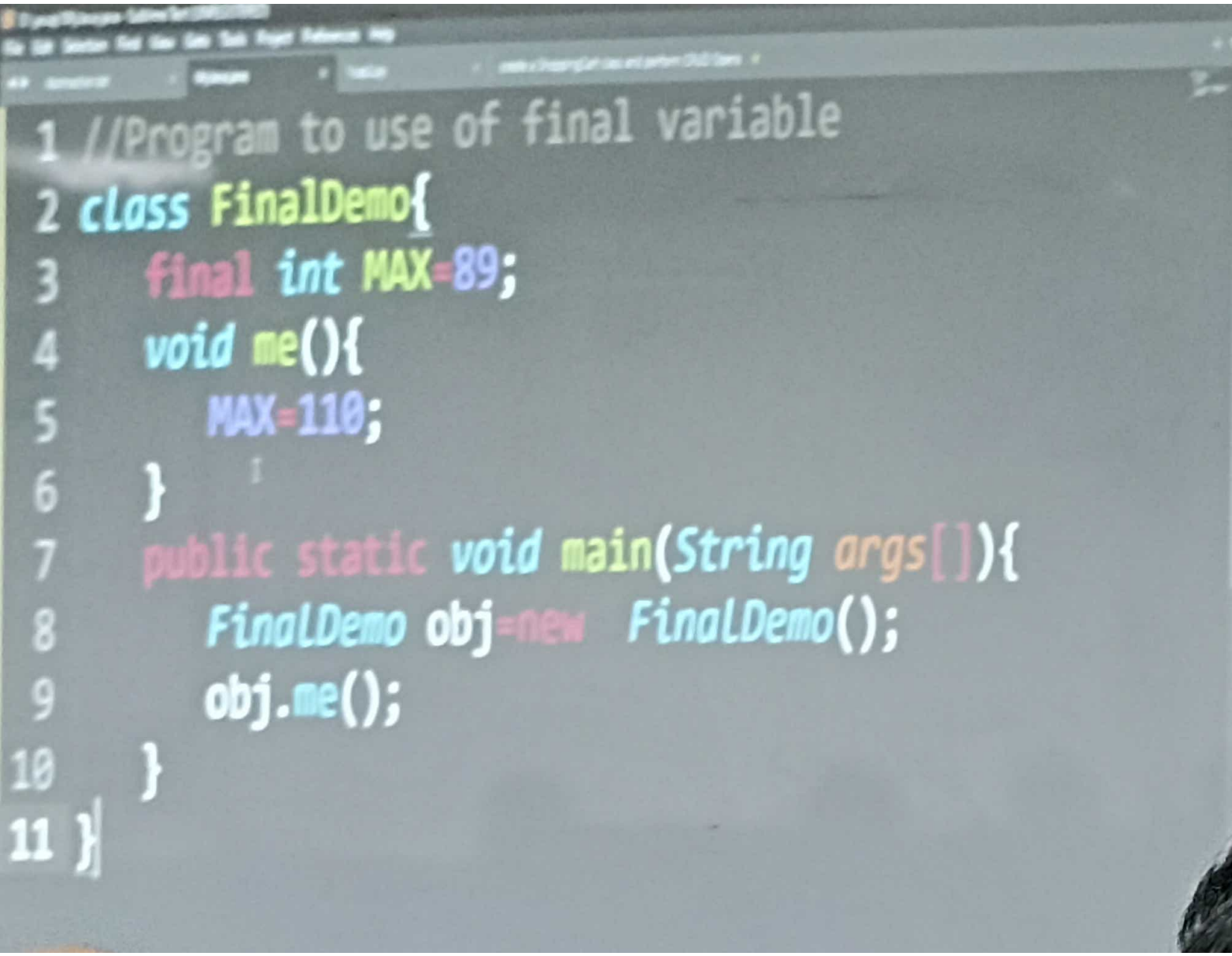
The final keyword can be:

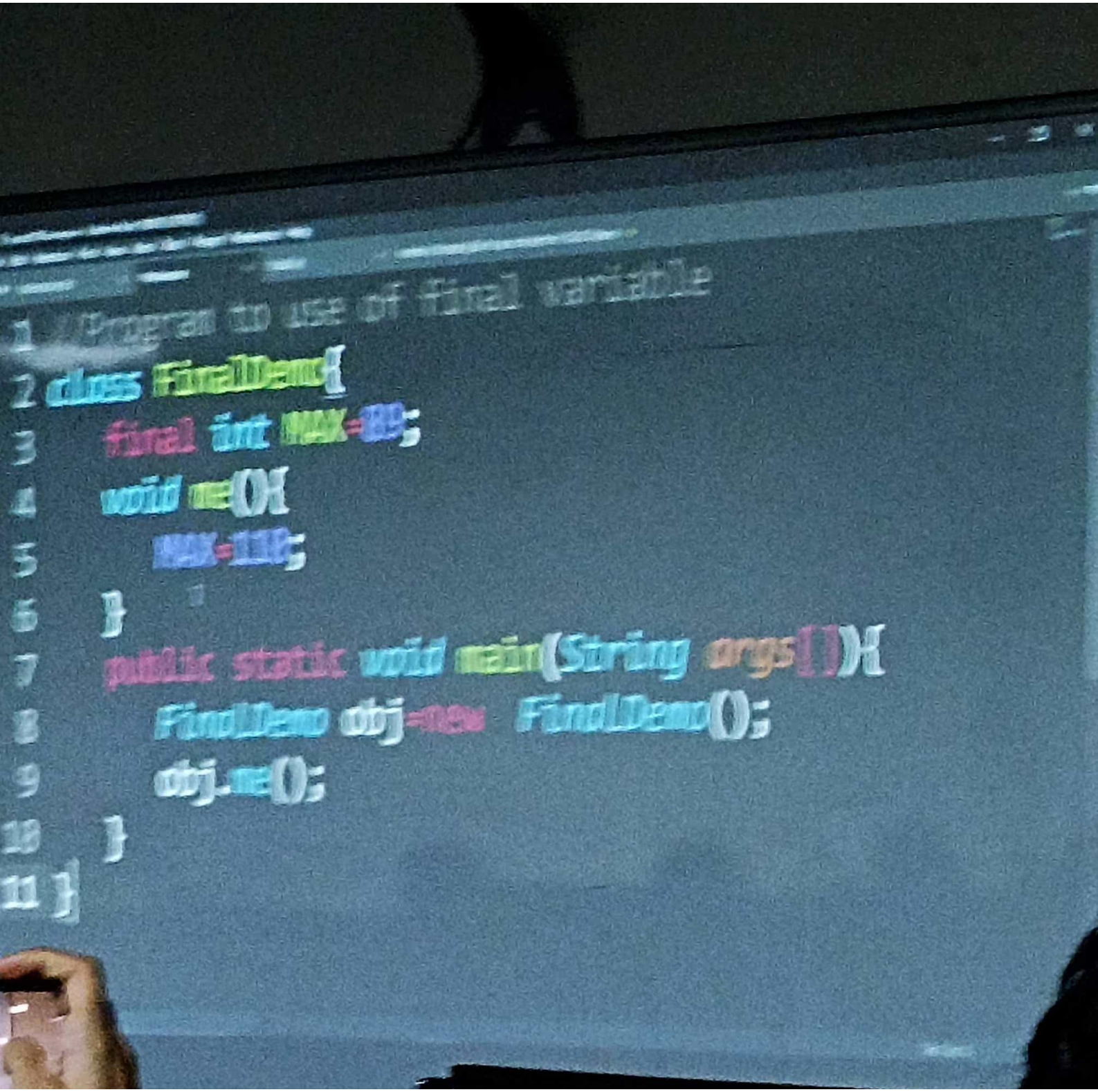
- **final** variable
- **final** method
- **final** class


File Edit Selection Find View Goto Tools Project Preferences Help
Abstraction.txt
MyJava.java
Main.py
create a ShoppingCart.class and perform CRUD Opera
1 //final method
2
3 class FinalMethod{
4 final void demo(){
5 System.out.println("FinalMethod Class Method");
6 }
7 }
8 class ABC extends FinalMethod{
9 void demo(){
10 System.out.println("ABC Class Method");
11 }
12 public static void main(String args[]){
13 ABC obj= new ABC();
14 obj.demo();
15 }
16 }


//final class
final class A{
}
class B extends A{
void demo(){
System.out.println("I am in A");
}
public static void main(String args[]){
B obj= new B();
obj.demo();
}
}


File Edit Selection Find View Goto Tools Project Preferences Help
Abstraction.txt
MyJava.java
1capi.py
create a ShoppingCart class and perform CRUD Opera
1 //Program to use of final variable
2 class FinalDemo{
3 final int MAX=89;
4 void me(){
5 MAX=110;
6 }
7 public static void main(String args[]){
8 FinalDemo obj=new FinalDemo();
9 obj.me();
10 }
11 }
Line 11, Column 2


1 //Program to use of final variable
2 class FinalDemo{
3 final int MAX=89;
4 void me(){
5 MAX=110;
6 }
7 public static void main(String args[]){
8 FinalDemo obj=new FinalDemo();
9 obj.me();
10 }
11 }



final variable

final variables are the constants which cannot change the value of a final variable once it is initialized.

final method

A final method cannot be overridden which means even though a subclass can call the final method of parent class but we cannot override it.

final class

A class is declared as final then this class cannot be inherited.

The **final** keyword in java is used to restrict the user.

The final keyword can be:

- **final** variable
- **final** method
- **final** class

super at constructor level

- super keyword is used to access the parent class constructor. super can also call both parametric as well as default constructors depending upon the situation.

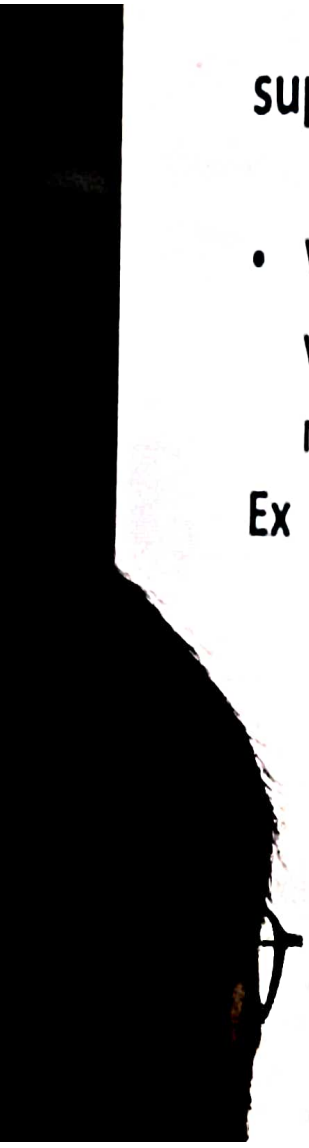
Ex

```
/Program to use super at method level
lass Person

   void message()   {
      System.out.println("I am person class");
   }

lass Student extends Person

   void message()   {
      System.out.println("I am student class");
   }
   void display()   {
      message();      // will invoke or call current class message() method
      super.message();// will invoke or call parent class message() method
   }

lass Test

   public static void main(String args[])  {
      Student s = new Student();
      s.display();
   }
```

super at method level

- When we want to call parent class method in child class and whenever a parent class and child class have same named methods then to resolve this ambiguity we use super keyword.

Ex



```java
//Program to use super at variable level
class Vehicle{
    int speed = 170;
}
class Car extends Vehicle
{
    int speed = 130;
    void display() {
        /* print speed of base class (vehicle) */
        System.out.println("Maximum Speed: " + super.speed);
    }
}
class Test
{
    public static void main(String[] args)   {
        Car s = new Car();
        s.display();
    }
}
```

```
//Program to use super at variable level
class Vehicle{
    int speed = 170;
}
class Car extends Vehicle
{
    int speed = 130;
    void display() {
        /* print speed of base class (vehicle) */
        System.out.println("Maximum Speed: " + super.speed);
    }
}
class Test
{
    public static void main(String[] args)   {
        Car s = new Car();
        s.display();
    }
```

It allows the methods to take any number of arguments

```
// m1() method will be executed 4 times.
class Test {
        void m1(int .. n)    {
        System.out.println("Ashish");
        }
        public static void main(String[] args)   {
                Test t=new Test();
                t.m1();
                t.m1(10);
                t.m1(10,20);
                t.m1(10,20,30);
        }
}
```

Output

```
Ashish
Ashish
Ashish
Ashish
```

If method contains both var-argument method & normal argument method then it will prints normal argument value.

```
class Test
{
    void m1(int... a)    {
        System.out.println("variable argument="+a);
    }
    void m1(int a)       {
        System.out.println("normal argument="+a);
    }
    public static void main(String[] args)   {
        Test t=new Test();
        t.m1(10);
    }
```

Normal arguments along with variable arguments.

```
class Test
{
   void m1(char ch,int... a)  {
      System.out.print(ch);
      for (int a1:a)     {
      System.out.print(" "+a1);
      }
      System.out.println();
   }
   public static void main(String[] args)  {
   Test t=new Test();
   t.m1('A');   t.m1('B',10);  t.m1('C',10,20);
   t.m1('D',10,20,30,40);
   }
```

Output

```
A
B 10
C 10 20
D 10 20 30 40
```

If method contains both var-argument method & normal argument method then it will prints normal argument value.

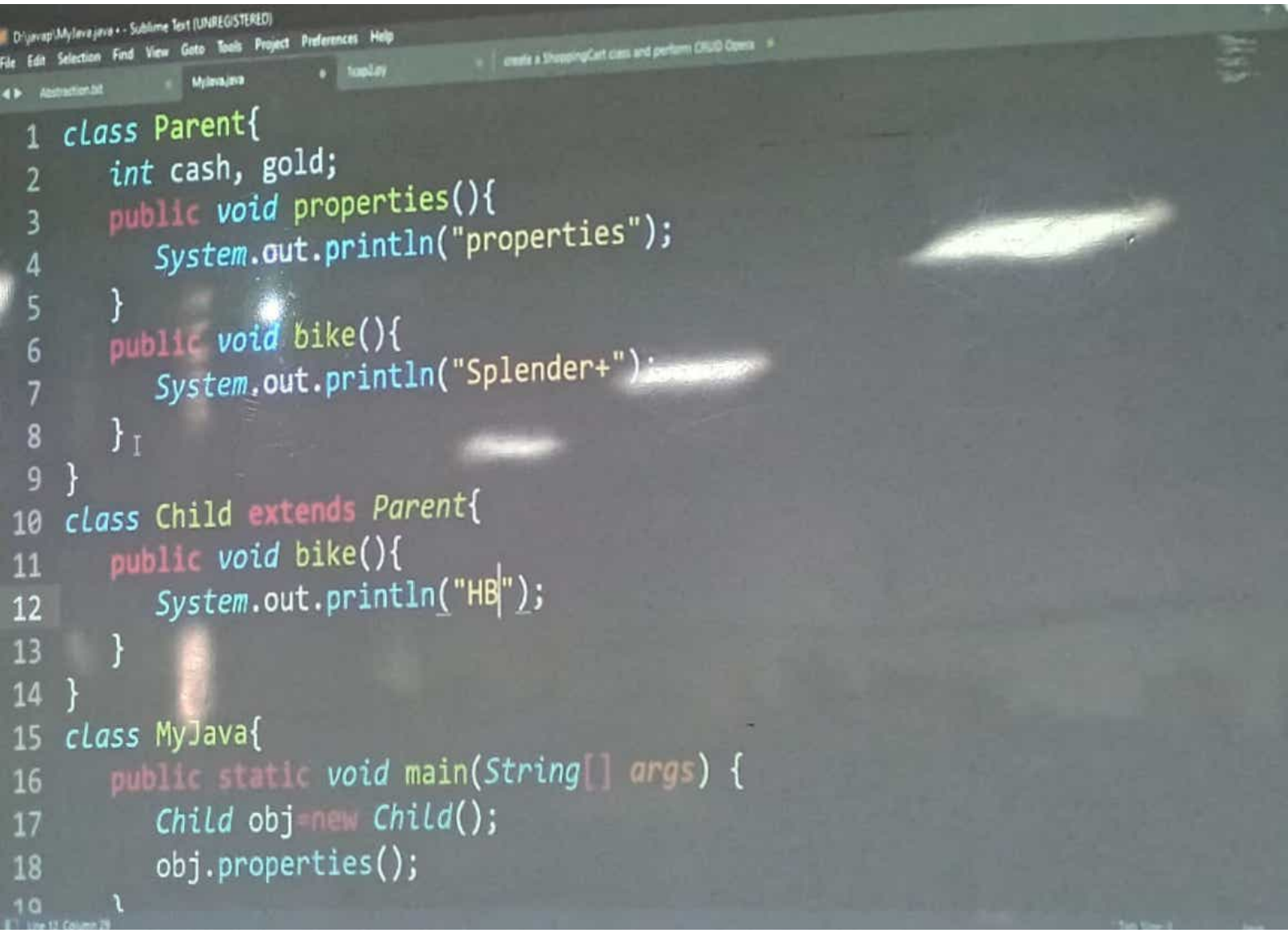
```
class Test
{
    void m1(int... a)    {
        System.out.println("variable argument="+a);
    }
    void m1(int a)       {
        System.out.println("normal argument="+a);
    }
    public static void main(String[] args)   {
        Test t=new Test();
        t.m1(10);
    }
```

It allows the methods to take any number of arguments

```
// m1() method will be executed 4 times.
class Test {
        void m1(int... n)    {
        System.out.println("Ashish");
        }
        public static void main(String[] args)   {
                Test t=new Test();
                t.m1();
                t.m1(10);
                t.m1(10,20);
                t.m1(10,20,30);
        }
```

Output

```
Ashish
Ashish
Ashish
Ashish
```


File Edit Selection Find View Goto Tools Project Preferences Help
Abstraction.txt
MyJava.java
class Parent{
int cash, gold;
public void properties(){
System.out.println("properties");
}
public void bike(){
System.out.println("Splender+");
}
}
class Child extends Parent{
public void bike(){
System.out.println("HB");
}
}
class MyJava{
public static void main(String[] args) {
Child obj=new Child();
obj.properties();

*5 Overriding.java - Notepad

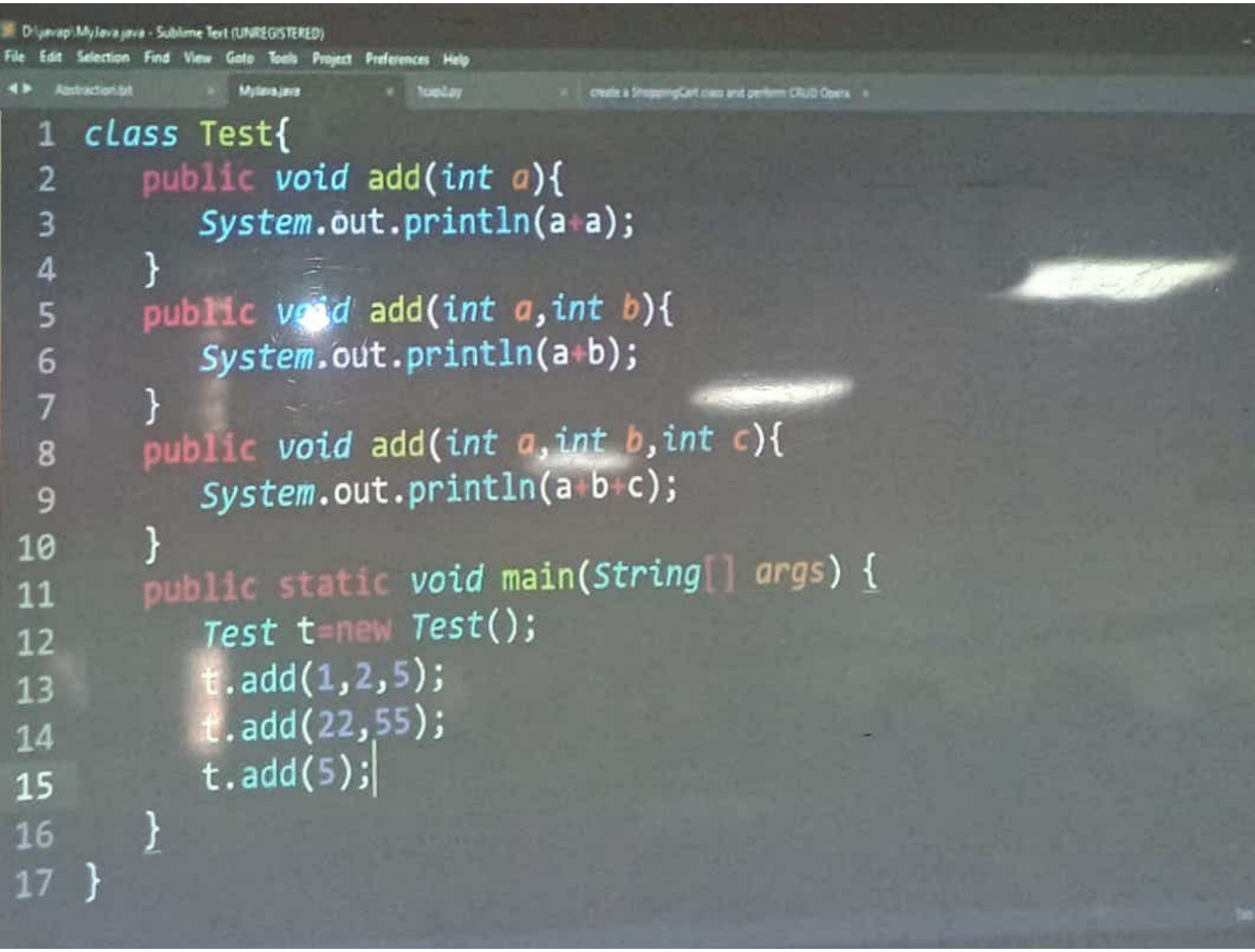
File Edit Format View Help

```
//Method overriding
class A{
      public int a=20;
            public void show(){
                  System.out.println("Class A show");
                  //System.out.println("a = "+a);
            }
};
class B extends A{
      public int a=40;
            public void show(){
                  System.out.println("Class B show");
                  //System.out.println("a = "+a);
            }
};
class Test{
            public static void main(String[] args)
            {
              B obj=new B();
```

6 Hierarchy.java - Notepad

File Edit Format View Help

```
//Constructor calling hierarchy
class A{
        public A(){
           System.out.println("Class A Constructor");
        }
        public void showA(){
            System.o  println("Class A show");
        }
};
class B extends A{
        public B(){
           System.out.println("Class B Constructor");
        }
        public void showB(){
            System.out.println("Class B show");
        }
};
class Test{
       public static void main(String[] args)
       {
         B obj=new B();
         obj.showB();
       }
```


D:\javap\MyJava.java - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Abstraction.txt
MyJava.java
1 class Test{
2 public void add(int a){
3 System.out.println(a+a);
4 }
5 public void add(int a,int b){
6 System.out.println(a+b);
7 }
8 public void add(int a,int b,int c){
9 System.out.println(a+b+c);
10 }
11 public static void main(String[] args) {
12 Test t=new Test();
13 t.add(1,2,5);
14 t.add(22,55);
15 t.add(5);
16 }
17 }

```
//Constructor calling hierarchy
class A{
        public A(){
           System.out.println("Class A Constructor");
        }
        public void showA(){
            System.out.println("Class A show");
        }
};
class B extends A{
        public B(){
           System.out.println("Class B Constructor");
        }
        public void showB(){
            System.out.println("Class B show");
        }
};
class Test{
       public static void main(String[] args)
       {
         B obj=new B();
         obj.showB();
```

```
//Method overriding

class A{
     public int a=20;
         public void show(){
             System.out.println("Class A show");
             //System.out.println("a = "+a);
         }
};
class B extends A{
     public int a=40;
         public void show(){
             System.out.println("Class B show");
             //System.out.println("a = "+a);
         }
};
class Test{
        public static void main(String[] args)
        {
```

achical.java
tilevel.java
le.java
ot4paperma... 1
rstnlast.java 1
ion.class
ion.java 1
iontrycatch.class
trong.java 1
array.java
java 1
ava 2
t.java 1
java 1
nul.java 1
lass
otionthrow.j... 1
ava 1
rloop.java 1
acci.java 1
no.java 1

C is also known as structured orien

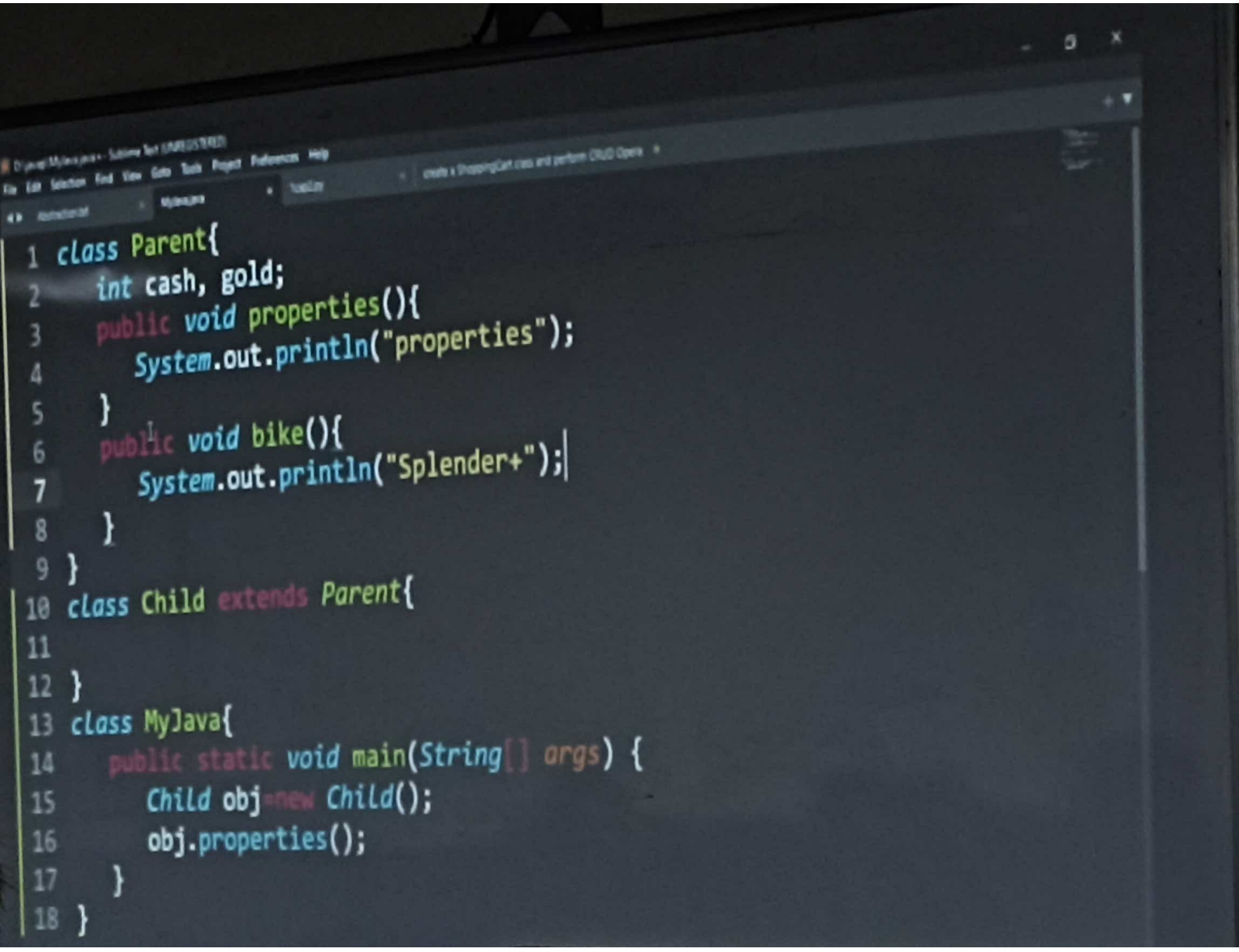
File Edit View

Loops
It is used to execute grp of statem
For loop is used when range is given
while loop is used when range is not given
do while is used for menu driven programs

Array is collection of finite ordered homogenous data elements
ERROR AND EXCEPTION
Two types of exception handling
try catch and throws
exception handling keywords try catch finally throws

Three ways to print exception info
toString() exception with message =Java.lang.Arithmetic.exception /by zero
getMessage() just message = /by zero
printStackTrace() along with 1 it prints where it is

Overloading
function is same three parameter are passed
it will automatically select the function with appr parameters



```java
class Parent{
    int cash, gold;
    public void properties(){
        System.out.println("properties");
    }
    public void bike(){
        System.out.println("Splender+");
    }
}
class Child extends Parent{

}
class MyJava{
    public static void main(String[] args) {
        Child obj=new Child();
        obj.properties();
    }
}
```

There are two types of polymorphism in java

- Compile time polymorphism / static binding / early binding

  Ex: method overloading

  If a class contains more than one method with same name but different number of arguments **or** same number of arguments but different data types those methods are called overloaded methods.

- Runtime polymorphism / dynamic binding / late binding.

  Ex: method overriding

  In method overriding same method name and same number & type of parameters but in different class and there should be parent child relationship

Polymorphism

- Polymorphism means having many forms.
- We can define polymorphism as the ability of a message to be displayed in more than one form.
- Eg: A person in real world can perform many task

- ✓ *In shopping mall behave like customer*
- ✓ *In school behave like student*
- ✓ *In bus behave like passenger*
- ✓ *At home behave like son*