1. Hello World program

here we write class which we have to include to access the System.out.println method and syntax is much similar to java language

class one

{

    public static void main(String args[])

    {

        System.out.println("Hello world!");

    }

}

## 2. Java Comments

## Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

## 2.1 Single-line Comments

Single-line comments start with two forward slashes (//).

// This is a comment

System.out.println("Hello World");

## 2.2 Multi-line Comments

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by Java.

/\* The code below will print the words Hello World

to the screen, and it is amazing \*/

System.out.println("Hello World");

## 3. Java Data Types

3.1Primitive data types

includes byte, short, int, long, float, double, boolean and char

A primitive data type specifies the size and type of variable values, and it has no additional methods

class datatype

{

    public static void main(String args[])

    {

        int myNum = 5;               // Integer (whole number)

        float myFloatNum = 5.99f;    // Floating point number

        char myLetter = 'D';         // Character

        boolean myBool = true;       // Boolean

        String myText = "Hello";     // String

        System.out.println("int:"+myNum);

        System.out.println("folat:"+myFloatNum);

        System.out.println("char:"+myLetter);

        System.out.println("boolean:"+myBool);

        System.out.println("string:"+myText);

    }

}

## 4. Java Operators

Operators are used to perform operations on variables and values.

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

## 4.1Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

public class arithope {

        public static void main(String[] args)

        {

            // initializing variables

            int num1 = 20, num2 = 10, sub = 0;

            // Displaying num1 and num2

            System.out.println("num1 = " + num1);

            System.out.println("num2 = " + num2);

            // subtracting num1 and num2

            sub = num1 - num2;

            System.out.println("Subtraction = " + sub);

            // addition  num1 and num2

            sub = num1 + num2;

            System.out.println("addition = " + sub);

            // Multiplication num1 and num2

            sub = num1 \* num2;

            System.out.println("Multiplication = " + sub);

            // Division num1 and num2

            sub = num1 / num2;

            System.out.println("Division = " + sub);

            // Modulus num1 and num2

            sub = num1 % num2;

            System.out.println("Modulus = " + sub);

        }

}

4.2Assignment operators

Assignment operators are used to assign values to variables.

class Assignmentop {

    public static void main(String[] args)

    {

        // Declaring variables

        int num1 = 10, num2 = 20;

        System.out.println("num1 = " + num1);

        System.out.println("num2 = " + num2);

        // Subtracting & Assigning values

        num1 -= num2;

        // Displaying the assigned values

        System.out.println(" Subtracting & Assigning values = " + num1);

         // Multiplying & Assigning values

         num1  \*= num2;

         // Displaying the assigned values

         System.out.println(" Multiplying & Assigning values = " + num1);

           // Dividing & Assigning values

           num1  /=  num2;

           // Displaying the assigned values

           System.out.println("Dividing & Assigning values = " + num1);

           // Moduling & Assigning values

           num1 %= num2;

           // Displaying the assigned values

           System.out.println("Moduling & Assigning values= " + num1);

           // Adding & Assigning values

           num1 += num2;

           // Displaying the assigned values

           System.out.println("Adding & Assigning valuess= " + num1);

    }

}

## 4.3Comparison Operators

Operators are considered special characters or symbols used to perform certain operations on variables or values

import java.util.Scanner;

public class Comparisonop {

    public static void main(String[] args) {

    int x, y;

    Scanner sc= new Scanner(System.in);

    //take the value of x as input from user and store it in variable x

    System.out.print("Enter the value of x : ");

    x = sc.nextInt();

    //take the value of y as input from user

    System.out.print("Enter the value of y : ");

    //store the value in variable y

    y = sc.nextInt();

    // equal

    System.out.println("equal = "+(x == y));

    // not equal

    System.out.println("not equal = "+(x != y));

    // less than

     System.out.println("less than = "+(x < y));

     // greater than

     System.out.println("greater than = "+(x > y));

     // less than or equal to

      System.out.println("less than or equal to "+(x <= y));

      // greater than or equal to

     System.out.println("greater than or equal to = "+(x >= y));

    }

    }

## 4.4 Logical Operators

1. **AND Operator**( **&&**) – if( a && b ) [if true execute else don’t]
2. **OR Operator** ( **||**) – if( a || b) [if one of them is true execute else don’t]
3. **NOT Operator** (**!**) – !(a<b) [returns false if a is smaller than b]

Here is an example depicting all the operators where the values of variables a, b, and c are kept the same for all the situations.

## 4.4.1 ****Logical ‘AND’ Operator (&&)****

## This operator returns true when both the conditions under consideration are satisfied or are true. If even one of the two yields false, the operator results false. In Simple terms, ***cond1 && cond2 returns true when both cond1 and cond2 are true***

public class Logicalopand {

     public static void main(String[] args)

    {

        // initializing variables

        int a = 10, b = 20, c = 20, d = 0;

        // Displaying a, b, c

        System.out.println("Var1 = " + a);

        System.out.println("Var2 = " + b);

        System.out.println("Var3 = " + c);

        // using logical AND to verify

        // two constraints

        if ((a < b) && (b == c)) {

            d = a + b + c;

            System.out.println("The sum is: " + d);

        }

        else

            System.out.println("False conditions");

    }

}

## 4.4.2 ****Logical ‘OR’ Operator (||)****

This operator returns true when one of the two conditions under consideration is satisfied or is true. If even one of the two yields true, the operator results true.

public class Logicalopor {

    public static void main(String[] args)

    {

        // initializing variables

        int a = 10, b = 1, c = 10, d = 30;

        // Displaying a, b, c

        System.out.println("Var1 = " + a);

        System.out.println("Var2 = " + b);

        System.out.println("Var3 = " + c);

        System.out.println("Var4 = " + d);

        // using logical OR to verify

        // two constraints

        if (a > b || c == d)

            System.out.println("One or both + the conditions are true");

        else

            System.out.println("Both the + conditions are false");

    }

}

## 4.4.3 ****Logical ‘NOT’ Operator (!)****

## Unlike the previous two, this is a unary operator and returns true when the condition under consideration is not satisfied or is a false condition. Basically, if the condition is false, the operation returns true and when the condition is true, the operation returns false.

public class Logicalopnot {

     public static void main(String[] args)

    {

        // initializing variables

        int a = 10, b = 1;

        // Displaying a, b, c

        System.out.println("Var1 = " + a);

        System.out.println("Var2 = " + b);

        // Using logical NOT operator

        System.out.println("!(a < b) = " + !(a < b));

        System.out.println("!(a > b) = " + !(a > b));

    }

}

5.Condition

# **5.1 Java If ... Else**

The [Java](https://www.javatpoint.com/java-tutorial) if statement is used to test the condition. It checks [boolean](https://www.javatpoint.com/boolean-keyword-in-java) condition: true or false. There are various types of if statement in Java.

public class ifcondi {

    public static void main(String[] args) {

        //defining an 'age' variable

        int age=20;

        //checking the age

        if(age>18){

            System.out.print("Age is greater than 18");

        }

    }

    }

## 5.2 Java if-else Statement

## The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

public class ifelsecondi {

    public static void main(String[] args) {

        //defining a variable

        int number=13;

        //Check if the number is divisible by 2 or not

        if(number%2==0){

            System.out.println("even number");

        }else{

            System.out.println("odd number");

        }

    }

    }

# 6.Java Switch Statement

The Java switch statement executes one statement from multiple conditions. It is like [if-else-if](https://www.javatpoint.com/java-if-else) ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.

public class switchstat {

    public static void main(String[] args) {

        //Declaring a variable for switch expression

        int number=20;

        //Switch expression

        switch(number){

        //Case statements

        case 10: System.out.println("10");

        break;

        case 20: System.out.println("20");

        break;

        case 30: System.out.println("30");

        break;

        //Default case statement

        default:System.out.println("Not in 10, 20 or 30");

        }

    }

    }

7.Loop

In Java, there are three kinds of loops which are – the for loop, the while loop, and the do-while loop. All these three loop constructs of Java executes a set of repeated statements as long as a specified condition remains true. This particular condition is generally known as loop control.

## 7.1Java for Loop

## A simple for loop is the same as [C](https://www.javatpoint.com/c-programming-language-tutorial)/[C++](https://www.javatpoint.com/cpp-tutorial). We can initialize the [variable](https://www.javatpoint.com/java-variables), check condition and increment/decrement value. It consists of four parts:

public class forloop {

    public static void main(String[] args) {

        //Code of Java for loop

        for(int i=1;i<=10;i++){

            System.out.println(i);

        }

    }

    }

# **7.2Java While Loop**

The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while [loop](https://www.javatpoint.com/java-for-loop)

public class whilelooop {

       public static void main(String[] args) {

        int i=1;

        while(i<=10){

            System.out.println(i);

        i++;

        }

    }

    }

# **7.3Java do-while Loop**

Java do-while loop is called an **exit control loop.** Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java do-while loop is executed at least once because condition is checked after loop body.

public class dowhilelooop {

    public static void main(String[] args) {

        int i=1;

        do{

            System.out.println(i);

        i++;

        }while(i<=10);

    }

    }

## 7.4 Java Break Statement with Loop

## When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

public class breaklooop {

    public static void main(String[] args) {

        //using for loop

        for(int i=1;i<=10;i++){

            if(i==5){

                //breaking the loop

                break;

            }

            System.out.println(i);

        }

    }

    }

# **8. Java Arrays**

Normally, an array is a collection of similar type of elements which has contiguous memory location.

**Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

## Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

## 8.1 Single Dimensional Array in Java

## Definition. A One-Dimensional Array is the simplest form of an Array in which the elements are stored linearly and can be accessed individually by specifying the index value of each element stored in the array.

public class singlearray {

    public static void main(String args[]){

    int a[]=new int[5];//declaration and instantiation

    a[0]=10;//initialization

    a[1]=20;

    a[2]=70;

    a[3]=40;

    a[4]=50;

    //traversing array

    for(int i=0;i<a.length;i++)//length is the property of array

    System.out.println(a[i]);

    }}

## 8.2Multidimensional Array in Java

## Multidimensional Arrays can be defined in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form in row major order.0

public class multiarray {

    public static void main(String args[]){

    //declaring and initializing 2D array

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

    //printing 2D array

    for(int i=0;i<3;i++){

     for(int j=0;j<3;j++){

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

     }

     System.out.println();

    }

    }}

# **9. Java Methods**

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

public class methood {

    static void myMethod() {

      System.out.println("student bread");

    }

    public static void main(String[] args) {

      myMethod();

      myMethod();

      myMethod();

    }

  }

# 9.1**Java Method Parameters**

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

public class methoodparameter {

    static void myMethod(String fname, int age) {

      System.out.println(fname + " is " + age);

    }

    public static void main(String[] args) {

      myMethod("mehul", 20);

      myMethod("bhavaya", 20);

      myMethod("ronak", 19);

    }

  }

# 9.2Java Method Overloading

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading.**

If we have to perform only one operation, having same name of the methods increases the readability of the [program](https://www.javatpoint.com/java-programs).

public class methodoverlo {

    static int plusMethodInt(int x, int y) {

        return x + y;

      }

      static double plusMethodDouble(double x, double y) {

        return x + y;

      }

      public static void main(String[] args) {

        int myNum1 = plusMethodInt(8, 5);

        double myNum2 = plusMethodDouble(4.3, 6.26);

        System.out.println("int: " + myNum1);

        System.out.println("double: " + myNum2);

      }

    }

# 10. Java Classes and Objects

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes,** such as weight and color, and **methods,** such as drive and brake.

## A Class is like an object constructor, or a "blueprint" for creating objects.

class Student{

    //defining fields

    int id;//field or data member or instance variable

    String name;

    //creating main method inside the Student class

    public static void main(String args[]){

     //Creating an object or instance

     Student s1=new Student();//creating an object of Student

     //Printing values of the object

     System.out.println(s1.id);//accessing member through reference variable

     System.out.println(s1.name);

    }

   }

# 11. **Method in Java**

**method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code.

We write a method once and use it many times. We do not require to write code again and again.

It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

### 11.1 **Predefined Method**

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.

 It is also known as the **standard library method** or **built-in method**.

public class Predefinedmethodd

 {

public static void main(String[] args)

{

// using the max() method of Math class

System.out.print("The maximum number is: " + Math.max(9,7));

}

}

### 11.2 **User-defined Method**

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

We have defined the above method named findevenodd().

import java.util.Scanner;

public class EvenOdd

{

public static void main (String args[])

{

//creating Scanner class object

Scanner scan=new Scanner(System.in);

System.out.print("Enter the number: ");

//reading value from user

int num=scan.nextInt();

//method calling

findEvenOdd(num);

}

//user defined method

public static void findEvenOdd(int num)

{

//method body

if(num%2==0)

System.out.println(num+" is even");

else

System.out.println(num+" is odd");

}

}

# 12. **Constructors in Java**

In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

## 12.1 Java Default Constructor

## A constructor is called "Default Constructor" when it doesn't have any parameter.

public class DefaultConstructorr {

//Java Program to create and call a default constructor

    //creating a default constructor

    DefaultConstructorr(){System.out.println("Bike is created");}

    //main method

    public static void main(String args[]){

    //calling a default constructor

    DefaultConstructorr b=new DefaultConstructorr();

    }

    }

### **12.2 Java Parameterized Constructor**

## A constructor which has a specific number of parameters is called a parameterized constructor.

## The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

public class parameterizedconstructorr {

    int id;

    String name;

    //creating a parameterized constructor

    parameterizedconstructorr(int i,String n){

    id = i;

    name = n;

    }

    //method to display the values

    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){

    //creating objects and passing values

    parameterizedconstructorr s1 = new parameterizedconstructorr(111,"mehul");

    parameterizedconstructorr s2 = new parameterizedconstructorr(222,"bhavaya");

    //calling method to display the values of object

    s1.display();

    s2.display();

   }

}

# 13.Java static keyword

The **static keyword**in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class).

The static keyword belongs to the class than an instance of the class.

class Student{

    int rollno;//instance variable

    String name;

    static String college ="ITS";//static variable

    //constructor

    Student(int r, String n){

    rollno = r;

    name = n;

    }

    //method to display the values

    void display (){System.out.println(rollno+" "+name+" "+college);}

 }

 //Test class to show the values of objects

 public class TestStaticVariable1{

  public static void main(String args[]){

  Student s1 = new Student(111,"mehul");

  Student s2 = new Student(222,"bhavya");

  //we can change the college of all objects by the single line of code

  //Student.college="BBDIT";

  s1.display();

  s2.display();

  }

 }

# 14.this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

class Student{

    int rollno;

    String name;

    float fee;

    Student(int rollno,String name,float fee){

    this.rollno=rollno;

    this.name=name;

    this.fee=fee;

    }

    void display(){System.out.println(rollno+" "+name+" "+fee);}

    }

    class TestThis2{

    public static void main(String args[]){

    Student s1=new Student(111,"mehul",5000f);

    Student s2=new Student(112,"bhavaya",6000f);

    s1.display();

    s2.display();

    }}

# 15.Inheritance in Java

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts)

he idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java)

that are built upon existing classes.

## 15.1 Single Inheritance

## When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

class Animal{

    void eat(){System.out.println("eating...");}

    }

    class Dog extends Animal{

    void bark(){System.out.println("barking...");}

    }

    class TestInheritance{

    public static void main(String args[]){

    Dog d=new Dog();

    d.bark();

    d.eat();

    }}

## 15.2 Multilevel Inheritance

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

class Animal{

    void eat(){System.out.println("eating...");}

    }

    class Dog extends Animal{

    void bark(){System.out.println("barking...");}

    }

    class BabyDog extends Dog{

    void weep(){System.out.println("weeping...");}

    }

    class TestInheritance2{

    public static void main(String args[]){

    BabyDog d=new BabyDog();

    d.weep();

    d.bark();

    d.eat();

    }}

## 15.3 Hierarchical Inheritance

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

class Animal{

    void eat(){System.out.println("eating...");}

    }

    class Dog extends Animal{

    void bark(){System.out.println("barking...");}

    }

    class Cat extends Animal{

    void meow(){System.out.println("meowing...");}

    }

    class TestInheritance3{

    public static void main(String args[]){

    Cat c=new Cat();

    c.meow();

    c.eat();

    //c.bark();//C.T.Error

    }}

# 16. **Abstract class in Java**

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial)

. It can have abstract and non-abstract methods (method with the body).

**Abstraction**is a process of hiding the implementation details and showing only functionality to the user.

abstract class Shape{

    abstract void draw();

    }

    //In real scenario, implementation is provided by others i.e. unknown by end user

    class Rectangle extends Shape{

    void draw(){System.out.println("drawing rectangle");}

    }

    class Circle1 extends Shape{

    void draw(){System.out.println("drawing circle");}

    }

    //In real scenario, method is called by programmer or user

    class TestAbstraction1{

    public static void main(String args[]){

    Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method

    s.draw();

    }

    }

# 17.Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve *[abstra](https://www.javatpoint.com/abstract-class-in-java)*[ction.](https://www.javatpoint.com/abstract-class-in-java)

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

It cannot be instantiated just like the abstract class.

interface Drawable{

    void draw();

    }

    //Implementation: by second user

    class Rectangle implements Drawable{

    public void draw(){System.out.println("drawing rectangle");}

    }

    class Circle implements Drawable{

    public void draw(){System.out.println("drawing circle");}

    }

    //Using interface: by third user

    class TestInterface1{

    public static void main(String args[]){

    Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()

    d.draw();

    }}

# 18. **Java ArrayList**

Java **ArrayList** class uses a dynamic [*array*](https://www.javatpoint.com/array-in-java).

for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package. It is like the Vector in C++.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of the List interface here. The ArrayList maintains the insertion order internally.

import java.util.\*;

 public class ArrayListExample1{

 public static void main(String args[]){

  ArrayList<String> list=new ArrayList<String>();//Creating arraylist

      list.add("Mango");//Adding object in arraylist

      list.add("Apple");

      list.add("Banana");

      list.add("Grapes");

      //Printing the arraylist object

      System.out.println(list);

 }

}

# 19. **Java File Class**

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

import java.io.\*;

public class FileDemo {

    public static void main(String[] args) {

        try {

            File file = new File("javaFile123.txt");

            if (file.createNewFile()) {

                System.out.println("New File is created!");

            } else {

                System.out.println("File already exists.");

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

# 20. **Java Create and Write To Files**

20.1 **Java Create file**

to create a file in Java, you can use the createNewFile() method.

 This method returns a boolean value: true if the file was successfully created, and false if the file already exists. Note that the method is enclosed in a try...catch block. This is necessary because it throws an IOException if an error occurs (if the file cannot be created for some reason):

import java.io.File;  // Import the File class

import java.io.IOException;  // Import the IOException class to handle errors

public class CreateFile {

  public static void main(String[] args) {

    try {

      File myObj = new File("filename.txt");

      if (myObj.createNewFile()) {

        System.out.println("File created: " + myObj.getName());

      } else {

        System.out.println("File already exists.");

      }

    } catch (IOException e) {

      System.out.println("An error occurred.");

      e.printStackTrace();

    }

  }

}

20.2 **Write To Files**

In the following example, we use the FileWriter class together with its write() method to write some text to the file we created in the example above. Note that when you are done writing to the file, you should close it with the close() method:

import java.io.FileWriter;   // Import the FileWriter class

import java.io.IOException;  // Import the IOException class to handle errors

public class WriteToFile {

  public static void main(String[] args) {

    try {

      FileWriter myWriter = new FileWriter("filename.txt");

      myWriter.write("Files in Java might be tricky, but it is fun enough!");

      myWriter.close();

      System.out.println("Successfully wrote to the file.");

    } catch (IOException e) {

      System.out.println("An error occurred.");

      e.printStackTrace();

    }

  }

}

# 21.Java Read Files

import java.io.File;  // Import the File class

import java.io.FileNotFoundException;  // Import this class to handle errors

import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {

  public static void main(String[] args) {

    try {

      File myObj = new File("filename.txt");

      Scanner myReader = new Scanner(myObj);

      while (myReader.hasNextLine()) {

        String data = myReader.nextLine();

        System.out.println(data);

      }

      myReader.close();

    } catch (FileNotFoundException e) {

      System.out.println("An error occurred.");

      e.printStackTrace();

    }

  }

}

# 22.Java Delete Files

To delete a file in Java, use the delete() method:

import java.io.File;  // Import the File class

public class DeleteFile {

  public static void main(String[] args) {

    File myObj = new File("filename.txt");

    if (myObj.delete()) {

      System.out.println("Deleted the file: " + myObj.getName());

    } else {

      System.out.println("Failed to delete the file.");

    }

  }

}