**Java**

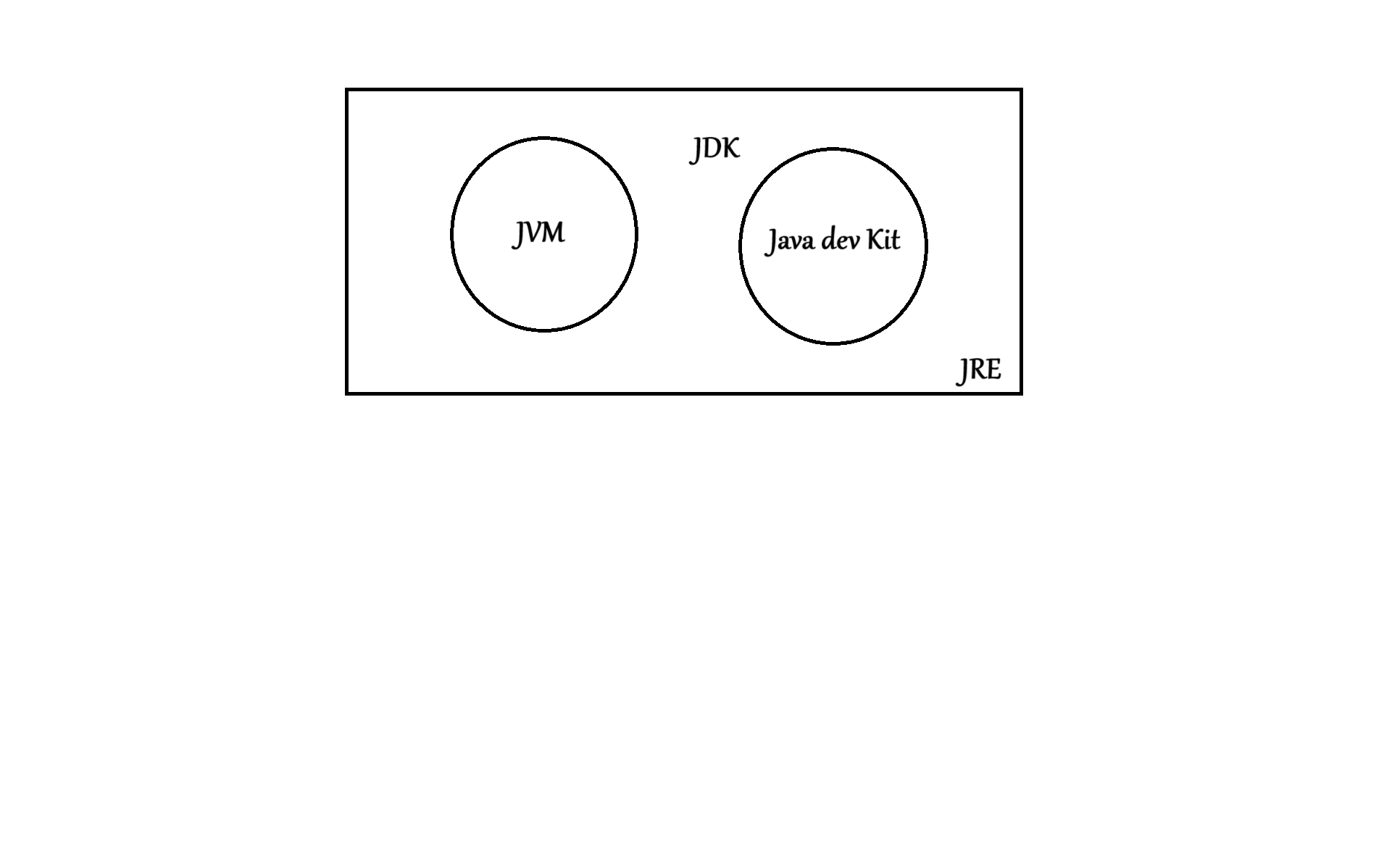
# **Java History**

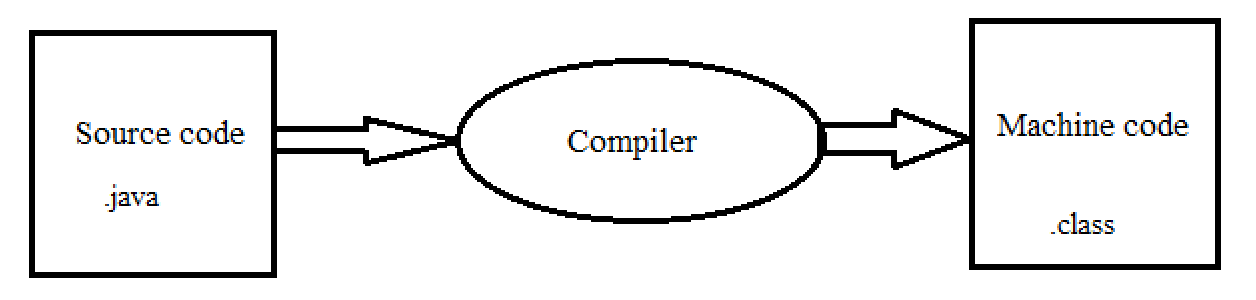
* It was developed by Sun microsystems.

# **Features**

* Java is a high-level object-oriented programming language.
* Java is platform-independent.
* It supports multithreading for multitasking.

# **JDK (Java Development Kit)**





\***Note**

* Byte Code is Universally accepted language

# **Applications of java**

* Desktop applications
* Web applications

# **Programming Knowledge**

***Syntax:***

**class** class\_name {

//Functions and methods to be declared

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

// main code logics

}

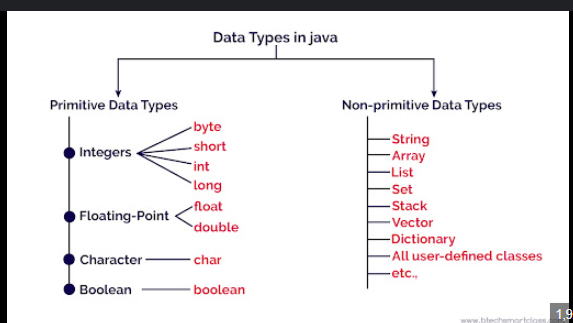
}

\***Note**

* A program cannot be run without the main method.

# **Data Types**

* Primitive
* Non – primitive



Memory space required to store datatypes:

* ***byte:*** 8 bits
* ***short:*** 4 bytes
* ***float:*** 4 bytes
* ***double:*** 8 bytes
* ***char:*** 2 bytes

# **Scanner class**

***Syntax:*** Scanner sc = new scanner(System.in); // for taking input

# **Conditional statements**

* + - 1. **if else condition**

**Syntax:**

if (condition) {

// operations

} else {

// operations if condition fails

}

* + - 1. **else if condition**

**Syntax:**

if (condition 1) {

// operations

} else if (condition 2) {

// operations if condition 1 fails

}

.

.

else {

// operations if all above conditions fail

}

* + - 1. **switch case**

***Syntax:***

switch (expression) {

case const\_expr1:

// operations

break;

case const\_expr2:

// operations

break;

.

.

default:

// operations

break;

}

# **Loops:**

1. **for loop**

***Syntax:***

for (initialization; condition; updation) {

// loop body

}

1. **while loop**

***Syntax:***

initialization;

while (condition) {

// loop body

updation;

}

1. **do-while loop**

***Syntax:***

initialization;

do {

// loop body

updation;

} while (condition);

# **Functions**

* A function commonly called a method is a block of code that performs a specific task and can be reused.
* Java methods are defined inside a class and are used to perform operations, results on data.
* Function name should not be any keyword.

***Syntax:***

return \_type function\_name(parameters) {

// function body

}

# **Array**

An array in java is a collection of elements (values or variables) each identified by an index or a key. All elements in an array are of the same datatype.

***Syntax:*** datatype[] var\_name = new datatype[size];

***Why use arrays?***

* To store multiple values in a single variable
* To avoid creating multiple variables
* Helps in iterating over a set of values using loops

\***Note**

* Array size cannot be altered after creating.
* It follows 0(zero) based indexing

***Q. Take an array as input from the user. Search for a given number x and print the index at which it occurs.***

***Program***

**import** **java.util.Scanner**;

**public** **class** search {

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

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

        System.out.print("Enter the size of the array: ");

*int* n **=** sc.nextInt();

*int*[] arr **=** **new** *int*[n];

        System.out.println("Enter the elements of the array:");

**for** (*int* i **=** 0; i **<** n; i**++**) {

            arr[i] **=** sc.nextInt();

        }

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

*int* x **=** sc.nextInt();

*int* index **=** **-**1;

**for** (*int* i **=** 0; i **<** n; i**++**) {

**if** (arr[i] **==** x) {

                index **=** i;

**break**;

            }

        }

**if** (index **!=** **-**1) {

            System.out.println("Number found at index: " **+** index);

        } **else** {

            System.out.println("Number not found in the array.");

        }

        sc.close();

    }

}

***Q. Take input in 2d array***

**import** **java.util.Scanner**;

**public** **class** twodarray {

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

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

        System.out.print("Enter number of rows: ");

*int* rows **=** sc.nextInt();

        System.out.print("Enter number of columns: ");

*int* cols **=** sc.nextInt();

*int*[][] array **=** **new** *int*[rows][cols];

        System.out.println("Enter the elements:");

**for** (*int* row **=** 0; row **<** rows; row**++**) {

**for** (*int* col **=** 0; col **<** cols; col**++**) {

                array[row][col] **=** sc.nextInt();

            }

        }

        System.out.println("The array is:");

**for** (*int* row **=** 0; row **<** array.length; row**++**) {

**for** (*int* col **=** 0; col **<** array[row].length; col**++**) {

                System.out.print(array[row][col] **+** " ");

            }

            System.out.println();

        }

        sc.close();

    }

}

**Strings**

***Program***

**public** **class** strings {

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

*String* str1 **=** "Hello";

*String* str2 **=** "World";

        // Concatenating strings

*String* result1 **=** str1 **+** " " **+** str2;

        System.out.println("Using + operator: " **+** result1);

        System.out.println("Printing the string one char at a time: ");

**for** (*int* i **=** 0; i **<** str1.length(); i**++**) {

            System.out.print(str1.charAt(i) **+** " ");

        }

        System.out.println();

*String* result2 **=** str1.concat(" ").concat(str2);

        System.out.println("Using concat() method: " **+** result2);

        // Comparison of strings

*String* str3 **=** "Hello";

*String* str4 **=** "hello";

        System.out.println("Comparing strings using equals(): " **+** str3.equals(str4));

        System.out.println("Comparing strings using equalsIgnoreCase(): " **+** str3.equalsIgnoreCase(str4));

        System.out.println("Comparing strings using compareTo(): " **+** str3.compareTo(str4));

        System.out.println("Comparing strings using compareToIgnoreCase(): " **+** str3.compareToIgnoreCase(str4));

        System.out.println("Comparing strings using == operator: " **+** (str3 **==** str4));

        // substring example

*String* str5 **=** "Hello, World!";

*String* subStr1 **=** str5.substring(0, 5);

*String* subStr2 **=** str5.substring(7);

        System.out.println("Substring of '" **+** str5 **+** "': " **+** subStr1 **+** " and " **+** subStr2);

    }

}

**StringBuilder**

***Syntax:*** StringBuilder sb = new StringBuilder(any string);

***Program: Reverse a string***

**public** **class** stringreverse {

**public** **static** *String* reverse(*String* *input*) {

*StringBuilder* sb **=** **new** StringBuilder(input);

**for** (*int* front **=** 0, back **=** sb.length() **-** 1; front **<** back; front**++**, back**--**) {

*char* temp **=** sb.charAt(front);

            sb.setCharAt(front, sb.charAt(back));

            sb.setCharAt(back, temp);

        }

**return** sb.toString();

    }

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

*String* str **=** "Hello";

        System.out.println("Original String: " **+** str);

        System.out.println("Reversed String: " **+** reverse(str));

    }

}

# **OOP**

* Java is an object-oriented programming language. Everything in Java revolves around objects and classes.
* It has 4 pillars:

1. Encapsulation
2. Abstraction
3. Polymorphism
4. Inheritance

**Constructor**

* It is called automatically when an object of class is created.
* It has the same name as the class.
* Doesn’t have any return type ***(not even void)***.
* It used to initialize the object.

***Types of constructors***

1. Default constructor (Non-Parameterized)
2. Parameterized constructor
3. Copy constructor

***Program***

**class** pen {

*String* color;

*String* type;

**public** pen(*String* *color*, *String* *type*) {

        this.color **=** color;

        this.type **=** type;

    }

**public** *void* write() {

        System.out.println("Writing with " **+** type **+** " pen of color " **+** color);

    }

}

**class** Student {

*String* name;

*int* age;

**public** Student(*String* *name*, *int* *age*) {

        this.name **=** name;

        this.age **=** age;

    }

**public** *void* display(){

        System.out.println("Name: " **+** name **+** ", Age: " **+** age);

    }

**public** Student(*Student* *sh*) {

        this.name **=** sh.name;

        this.age **=** sh.age;

    }

}

**public** **class** oops {

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

        pen p1 **=** **new** pen("blue", "ball-point");

        p1.write();

        pen p2 **=** **new** pen("red", "gel");

        p2.write();

*Student* s1 **=** **new** Student("Alice", 20);

        s1.display();

*Student* s2 **=** **new** Student("Bob", 22);

        s2.display();

*Student* s3 **=** **new** Student(s1);

        s3.display();

    }

}

# **Garbage collections:**

Java don’t have the destructors it uses the concept of garbage collector.

# **Inheritance**

Inheritance is a mechanism by which one class acquires the properties and behaviors of another class.

**Types of inheritance**

Parent

Child

# **Polymorphism**

# It enables run time polymorphism via method overriding. Code maintenance is easy with polymorphism.

# 

# **Programs**

# **Demo Program**

**public class demo {**

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

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

**}**

**}**

# **Pattern Printing**

**public** **class** patternprinting {

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

*int* n **=** 4;

**for** (*int* i **=** 0; i **<** n; i**++**) {

**for** (*int* j **=** 0; j **<** n; j**++**) {

                System.out.print(" \* ");

            }

            System.out.println();

        }

**for** (*int* i **=** 1; i **<=** n; i**++**) {

**for** (*int* j **=** 1; j **<=** i; j**++**) {

                System.out.print(" \* ");

            }

            System.out.println();

        }

**for** (*int* i **=** 1; i **<=** n; i**++**) {

**for** (*int* j **=** 1; j **<=** 4; j**++**) {

**if** (i **==** 1 **||** i **==** n **||** j **==** 1 **||** j **==** n) {

                    System.out.print(" \* ");

                } **else** {

                    System.out.print("   ");

                }

            }

            System.out.println();

        }

 int k=1;

        for (int i = 1; i <= n; i++) {

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

                System.out.print(k + " ");

                k++;

            }

            System.out.println();

        }

    }

}

# **Even Odd**

public class evenodd {

    public static void main(String[] args) {

        java.util.Scanner sc = new java.util.Scanner(System.in);

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

        int num = sc.nextInt();

        if (num % 2 == 0) {

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

        } else {

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

        }

        sc.close();

    }

}

**Prime number**

**import** **java.util.Scanner**;

**public** **class** primenum {

**public** *boolean* prime(*int* *num*) {

*boolean* isPrime **=** true;

**if** (num **<=** 1) {

            isPrime **=** false;

        } **else** {

**for** (*int* i **=** 2; i **<=** Math.sqrt(num); i**++**) {

**if** (num **%** i **==** 0) {

                    isPrime **=** false;

**break**;

                }

            }

        }

**return** isPrime;

    }

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

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

*int* num **=** sc.nextInt();

        prim**enum** p = new primenum();

        if (p.prime(num)) {

            System.out.println(num **+** " is a prime number.");

        } **else** {

            System.out.println(num **+** " is not a prime number.");

        }

        sc.close();

    }

}

# **Sum of n natural numbers**

**import** **java.util.Scanner**;

**public** **class** sumofnnaturalnumbers {

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

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

        System.out.print("Enter a number to get sum of n natural numbers: ");

*int* n **=** sc.nextInt();

*int* sum **=** 0;

**for** (*int* i **=** 1; i **<=** n; i**++**) {

            sum **+=** i;

        }

        System.out.println("The sum of the first " **+** n **+** " natural numbers is: " **+** sum);

        sc.close();

    }

}

# **Factorial (With Exception handling for negative numbers):**

import java.util.Scanner;

public class factorial {

    public static int fact(int n) {

        if (n == 0 || n == 1) {

            return 1;

        } else {

            return n \* fact(n - 1);

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number to calculate its factorial: ");

        try {

            int number = sc.nextInt();

            if (number < 0) {

                throw new IllegalArgumentException("Factorial is not defined for negative numbers.");

            }

            int result = fact(number);

            System.out.println("The factorial of " + number + " is: " + result);

        } catch (IllegalArgumentException e) {

            System.out.println("Error: " + e.getMessage());

        } catch (Exception e) {

            System.out.println("Invalid input. Please enter a valid integer.");

        } finally {

            sc.close();

        }

    }

}

# **Tasks**

# **Task 1**

1. take 2 variables a and b (int)
2. Create sum variable
3. Create sub variable
4. Print sum
5. Print sub

# **Program**

**import** **java.util.Scanner**;

**public** **class** task1 {

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

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

*int* a, b, sum, sub;

System.out.println("Enter two numbers:");

a **=** sc.nextInt();

b **=** sc.nextInt();

sum **=** a **+** b;

sub **=** a **-** b;

System.out.println("Sum: " **+** sum);

System.out.println("Subtraction: " **+** sub);

sc.close();

}

}

**Task 2**

1. a. create buttons
2. b. if button1 hello
3. c. button2 namaste
4. d. button3 good morning
5. e. else bonjour

# **Program**

**import** **java.util.Scanner**;

**public** **class** task2 {

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

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

        System.out.print("Enter the button number(1 - 3): ");

*int* button **=** sc.nextInt();

**if** (button **==** 1) {

            System.out.println("Hello");

        } **else** **if** (button **==** 2) {

            System.out.println("Namaste");

        } **else** **if** (button **==** 3) {

            System.out.println("Good Morning");

        } **else** {

            System.out.println("Bonjour");

        }

        sc.close();

    }

}

# **Task 3**

Write table of 2 by using for loop

# **Program**

public class table {

    public static void main(String[] args) {

        int t=2;

        for (int k=1;k<=10;k++){

            System.out.println(t+"\*"+k+"="+(t\*k));

        }

    }

}

# **Task 4**

a. take 5 numbers from user

b. print even number

c. print odd numbers

# **Program**

**import** **java.util.Scanner**;

**public** **class** eveodd {

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

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

*int* nums[] **=** **new** *int*[5];

        System.out.println("Enter 5 numbers: ");

**for** (*int* i **=** 0; i **<** 5; i**++**) {

            System.out.print("Number " **+** (i **+** 1) **+** ": ");

            nums[i] **=** sc.nextInt();

        }

        System.out.print("Even numbers: ");

**for** (*int* i **=** 0; i **<** 5; i**++**) {

**if** (nums[i] **%** 2 **==** 0) {

                System.out.print(nums[i] **+** "\t");

            }

        }

        System.out.println();

        System.out.print("Odd numbers: ");

**for** (*int* i **=** 0; i **<** 5; i**++**) {

**if** (nums[i] **%** 2 **!=** 0) {

                System.out.print(nums[i] **+** "\t");

            }

        }

  sc.close();

    }

}

**Task 5**

Make function to add two numbers and return it

***Program***

**import** **java.util.Scanner**;

**public** **class** task5 {

**public** *int* add() {

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

*int* a **=** sc.nextInt();

*int* b **=** sc.nextInt();

        sc.close();

**return** a **+** b;

    }

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

        task5 obj **=** **new** task5();

*int* result **=** obj.add();

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

    }

}

# **Task 6**

a. function for counting digits in a number

b. function to check if string is palindrome

c. function to reverse a number

***Program***

**import** **java.util.Scanner**;

**public** **class** task6 {

**public** *void* countDigits () {

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

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

*int* n **=** sc.nextInt();

*int* count **=** 0;

**while** (n **>** 0) {

            n **/=** 10;

            count**++**;

        }

        System.out.println("Number of digits: " **+** count);

        sc.close();

    }

**public** *void* palindrome() {

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

        System.out.print("Enter a string: ");

*String* str **=** sc.next();

*String* reversed **=** **new** StringBuilder(str).reverse().toString();

**if** (str.equals(reversed)) {

            System.out.println("The string is a palindrome.");

        } **else** {

            System.out.println("The string is not a palindrome.");

        }

        sc.close();

    }

**public** *void* reverseNumber() {

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

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

*int* n **=** sc.nextInt();

*int* reversed **=** 0;

**while** (n **!=** 0) {

*int* digit **=** n **%** 10;

            reversed **=** reversed **\*** 10 **+** digit;

            n **/=** 10;

        }

        System.out.println("Reversed number: " **+** reversed);

        sc.close();

    }

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

        task6 t **=** **new** task6();

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

        System.out.println("Choose an option:");

        System.out.println("1. Count digits in a number");

        System.out.println("2. Check if a string is a palindrome");

*int* choice **=** sc.nextInt();

**switch** (choice) {

**case** 1**:**

                t.countDigits();

**break**;

**case** 2**:**

                t.palindrome();

**break**;

**case** 3**:**

                t.reverseNumber();

**break**;

**default:**

                System.out.println("Invalid choice.");

        }

        sc.close();

    }

}

Prime Number:

**import** **java.util.Scanner**;

**public** **class** findprime {

**public** *void* isprime(*int* *n*) {

*int* i, m **=** n **/** 2, flag **=** 0;

**if** (n **==** 0 **||** n **==** 1) {

            System.out.println(n **+** " is not a prime number.");

        } **else** {

**for** (i **=** 2; i **<=** m; i**++**) {

**if** (n **%** i **==** 0) {

                    System.out.println(n **+** " is not a prime number.");

                    flag **=** 1;

**break**;

                }

            }

**if** (flag **==** 0) {

                System.out.println(n **+** " is a prime number.");

            }

        }

    }

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

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

        System.out.print("Enter a number to check if it is prime: ");

*int* n **=** sc.nextInt();

        findprime fp **=** **new** findprime();

        fp.isprime(n);

        sc.close();

    }

}

This keyword: used to pass the instance variable .

All constructor:



Inheritance:



Polymorphism:

* Polymorphism is the one of the core concept of OOP.
* It allows objects to take many forms
* Poly(many) & morph(forms)
* We can called it as “one name many forms”.
* It is also known as static binding or method overloading.
* Compile time polymorphism is achieved by method overloading.
* Method resolution happens at compile time.

Benefits of polymorphism:

🡪code reusability.

🡪easier to maintain and extend.

🡪it promotes flexibility and scalability.

🡪it enables loose coupling.

***Conditions for compile time polymorphism (method overloading)***

* Same method name.
* All overloaded method must have the same name (same method name)
* Different parameters list

🡪parameter must differ in.

* + 1. Number of parameters.
    2. Type of parameters.
    3. Order of parameters.
* Return type alone can’t differentiate.
* You can’t overload methods only by changing the return type.

***Condition for Run time polymorphism (method overriding)***

* Also called as method overriding.
* Late binding, dynamic binding(***things a decide at runntime by JVM***)
* It is a type of polymorphism where the specific method to be called is determine that run time not at compile time.
* This is achieved through method overriding, where a sub class provides specific implementation from method inherited from its parent class.
* The JVM decides which method to execute based on the actual **object type** at runtime.

**Rules:**

only possible with inheritance (must)

method must be overridden not overloaded.

One instance’s methods are polymorphic (not static or private methods).

Static method resolve at compile time that’s why not possible.

Private or final method methods can’t be ovrrided.

**Benefits**:

* Flexibility
* Loose coupling
* Reusability
* Scalability

**Abstraction:**

* It is one of the four pillar oop in java.
* It means id internal implementation details and showing only essential feature to the user.

Key Points:

* It hides complex code and show only the necessary parts.
* Implementation is done Using abstract class and interfaces.
* Users only sees what’s necessary.
* Eg. Driving a car – you use the steering, break, gear but don’t know the internal engine mechanics.

Why we use abstraction:

* It improves code maintainability.
* Achieve loose coupling(dependencies/ dependent on others).
* Enhance security.
* It encourage modular programming.

Properties:

* An abstract class must be declared with abstract keyword.
* It can have abstract as well as non abstract methods.
* It can’t be instantiate.
* It can have constructor’s and static method also.
* It can have final methods which will force the subclass not to change the body of method.
* ***Jab bhi ham derived class ke constructor ko call krte hai toh pehele base class ka constructor call hota hai then derived class ka const. call hota hai***.
* In abstract it extends one class only.

What is abstract and non abstract method:-

🡪non abstract methos also called as concrete method.

Que: create a java program using abstraction to model a payment system.

Req:-

🡪create an abstract class name payment.

🡪add an abstract method process payment (double amount)

🡪create two subclasses

1.credit card payment

2.UPI payment.

🡪implement the process payment method in both subclasses with custom logic (print statement)

🡪in the main method

1. use abstract class ref. to call the method for both payment types.

Interface:

* An interface in java used to group related abstract methods.
* It defines what a class must do but not how to does it.

**Key features:**

* It uses the interface keyword
* All the methods and abstract by default
* ***All the fields (variables) / interfaces are public, static and final by default (till java 7)***
* ***To inherit we use implement not extend.***
* we can implement multiple inheritance using interfaces.
* For java 8+ it can have default and static methods.
* In interface there is no non abstract method allowed.
* ***It used to achieve 100% abstraction.***

***Why interface can’t be constructor?***

* An interface in not a class, it’s just a contract (blueprint) for what a class should do.
* Since interfaces can’t be instantiated, they don’t need constructor.

Interface and Abstraction Difference:

**Abstraction🡪**

* It can’t be instantiated.
* It contain both abstract as well non abstract methods.
* Implementation method:

🡪it can have both implemented and abstract methods.

* Inheritance:

🡪Class can inherit for only one abstract class.

* Access modifiers:

🡪Methods and properties can have any access modifiers (public, private)

* Variables:

🡪it can have members variables (final, on final, static, non-static).

**Interface🡪**

* It specifies the set of methods a class must implement.
* Methods are abstract by default.
* For java 8+ it can have default and static methods
* Inheritance:

🡪A class can implement multiple interfaces.

* Implementation methods:

🡪method are abstract by default.

* Access modifiers:

🡪Methods and properties are implicitly public.

* Variables:

🡪Variables are implicitly public, static, final by default.

**Final Keyword:**

* In java final keyword is a modifier that can be used with variables, methods and classes and it has different meaning depending on the context.

**Final variable:**

* Once assigned the value of the final variable can’t be change.
* It can be applied to instance local variable or static variable.
* It offered you to create constants.
* Best Practice – use final with static for constants.

**Final method:**

* A method marks as a final can’t be override in subclasses.
* It’s used to prevent method behaviors from being changed.

**Final class:**

* A class mark as a final can’t be inherited.
* It is used to prevent subclasses.

**Static Keyword:**

* The static keyword is used for memory management and defines members (variables, method, blocks, nested classes) that belong to the class rather the to the instances of the class(objects).

**Static Variable (class variable):**

* It shared among all instances of a class.
* Only one copy exists in memory, regardless of the no of objects created.

**Static method:**

* Static method belongs to the class not instance.
* It can be called without creating object.
* It can access only static data directly (no this keyword).

**Static class:**

* Only nested classes can be static (not top level).
* It can access only static member of the outer class.

**Static Block:**

* Static block used for static initialization.
* Executes once when the class is loaded (before the main method or any object is created).

Questions:

1. Armstrong Number: is the number i.e equal to sum of its on digits each raise to power of number of digit.

2.latgest element in arr.

**What is Database:**

* A database is the organized collection of data that can be easily accessed, manage and updated.

**What is DBMS:**

* Database management system.
* A DBMS is a software that allows user to interact with databases for storing and retrieving and manipulating the data.
* Eg. Of DBMS: Mysql, oracle, SqlServer, postgre SQL, mongodb(no sql), SQL lite.

**Why use DBMS?**

* Efficient data management.
* Easy data retrieval.
* Less redundancy.
* Security and access control.
* Backup and recovery.

**Types of DBMS:**

1. **Hirarchical:** data in tree like structure.

**🡪**Eg. IBM, IMS

1. **Network:** Data has many to many relationships.

**🡪**Eg. IDS, TurboImg.

1. **Relational:** data in tables(rows and columns) mostly used.

🡪Mysql, oracle.

1. **No SQL:** this is not tabular, used for big data.

**🡪** Mongodb.

**RDBMS**: relational database management system

* RDBMS is the type on database based on the relational mode of data.
* Data is stored in table with rows and columns.
* Each table has a unique name and follows a define structure (schema).
* SQL is use to interact with the database.

Key concepts of RDBMS:

**Table/ relation**:

🡪a collection of rows and columns.

**Row:**

🡪a single record in a table.

**Column/attributes:**

🡪a property or field of a table.

**Schema:**

🡪structure of a database (tables, columns, datatypes).

**Primary Key:**

🡪uniquely identifies each row in a table.

**Foreign key:**

🡪it creates a link between two tables.

**Constraints**:

🡪restriction/ conditions.

e.g.

|  |  |  |  |
| --- | --- | --- | --- |
| Stud ID | Name | Age | Course ID |
| 1 | Piyush | 19 | 101 |
| 2 | Neha | 19 | 102 |

**Features of RDBMS:**

**1. Data Integrity:**

**🡪** It ensures accuracy and consistency.

**2. Relationship:**

**🡪**It manages relation between tables.

**3. Data security:**

🡪Access control and permissions.

**4. Concurrency control:**

🡪Multiple user can work seamlessly.

1. **Normalization**:

🡪Reduces data redundancy.

1. **ACID Property:**

🡪 It ensures transaction reliability.

**ACID Property:**

* A - All or null rule for transactions.
* C - consistency (it maintains valid states).
* I – Isolation (concurrent transaction are isolated).
* D – Durability (changes are permanent).

**Relationship in RDBMS:**

🡪**one to one**: one person one passport.

🡪**one to many**: one course many student.

🡪**many to many**: students projects (via join table).

DDL (data defination language):

* DDL is a category of SQL command used to define unmanages database structures like tables, schemas, constants and indexes (structure of database not only data itself).

**Key DDL commands:**

1. Create – create database obj like table.

2. Alter – modifying the existing structure.

3. Drop – delete database objects.

4. truncate – remove all data form a table but structure remains.(CANT BE ROLLBACK) AND FASTER THAN DELETE.

5. rename – rename the name of database object.

**Database Creation:**

Database Creation:

**🡪**CREATE DATABASE DB\_NAME;

Add Database column:

🡪CREATE TABLE TABLE\_NAME(

STUD\_ID INT PRIMARY KEY,

STUD\_NAME VARCHAR,

AGE INT,

COURSE VARCHAR(100));

Alter:

🡪**ALTER** TABLE STUDENT **MODIFY** AGE BIGINT;

🡪 **ALTER** TABLE STUDENT **MODIFY** AGE SMALLINT;

Drop COLUMN:

🡪**ALTER** TABLE STUDENT DROP COLUMN AGE;

Drop:

🡪**DROP** TABLE STUDENT;

Truncate:

🡪**TRUNCATE** TABLE STUDENT;

Rename:

🡪**RENAME** TABLE STUDENT **TO** STUDENT\_INFO;

Rename column:

🡪**ALTER** TABLE STUDENT\_INFO **RENAME** COLUMN AGE TO STUDENT\_AGE;

**DML (Data Manipulation language):**

* It’s a subset of SQL which is used to mange data stored in database table.
* **Focus on perform operation on data not in table structure.**

-Insert: add new data(rows) into a table.

-Update: modify existing data in a table.

-Delete: remove data from table.

Insert:

🡪INSET INTO TABLE STUDENT

(Name, Age, Rollcall) VALUES

(“XYZ”, 23, 2395),

(“XYZZ”, 223, 21395);

Update:

🡪UPDATE STUDENT

SET SALARY = 35000

WHERE ID = 002;

3.Delete:

🡪DELETE FORM STUDENT

WHERE ID > 7;

4.Delete:

🡪DELETE FORM STUDENT

WHERE ID = 1;

**What is DCL:**

* Data control Language.
* It is a subset of SQL which is used to control access and permission on the data stored on a database.
* Focus: **Security and access control (who can do what in the database).**

-**Grant:** Give permissions to user.

**-Revok**: Take back permission from the user.

**Common privileges:**

1. select

2. insert

3. update

4. delete

5. execute

6. uses

7. call

What is Joins in SQL:

* A join is used to combine rows from two or more tables based on a related columns between them, typically a foreign key.

**Real world use-cases:**

1. combine employee info with department info, student with course, order with customer etc.

**Type of SQL joins:**

**1.** **Inner Join**: it returns only matching rows from both table.

**2. Outer join: 🡪**

**i.** **Left join**: all records form the left table and matching records from the right table.

**ii.** **Right join**: Returns all rows from the right table + matching from the left

**iii. Full Join:** all records of right table and all records of left table.

**TCL**:

* Transaction control language.
* It is a subset of SQL which is used to manage transaction in a database.
* It ensures that a group of operations (like multiple insert, update, delete) can be executes as a single unit of work.
* Goal: **Maintain data integrity and consistency in multi-step operation.**

**What is transaction?**

* It’s a sequence of one or more SQL operation that are treated as a single logical unit.
* It must follow the ACID properties.

**Common TCL Commands:**

1. commit: save all the change made in the transaction.

2. Rollback: undo all changes made in the transaction.

3. save point: commit rollback.

**JDBC:**

* Java database connectivity.
* JDBC is an API (application Programming interface) provided by java to connect and interact with relational databases.

**Why use JDBC:**

* To perform crud (read, update, delete) operation.
* Java application can interact with MYSQL, oracle, post-gre SQL, etc.

**JDBC Archi:**

* **Driver:** interface implemented database vendors. E.g. MySQL, JDBC driver.
* **Connection:** interface for opening the session with the database.
* **Statement/ prepared statement**: interface to execute twice.
* **Result set:** it stores the result retrieve from the database.
* **Driver manager:** it is responsible for managing JDBC drivers and connecting to the appropriate database.

**Steps to use JDBC:**

1. load the JDBC drivers.

2.establish the connection.

3.create a statement.

4.execute the query.

5.process the result.

6.close the connection.

**🡨Threads in java🡪**

🡪In java thread is process of executing multiple task simultaneously.

🡪it is an light-weight, smallest unit of process.

What is multithreading?

🡪multithreading in java allows you to run two or more parts of a program(threads)

Concurrently to make better use of CPU.

How to create a thread:

* Two way to create a threads In java.

1.By extending the thread class.

2.By implementing Runnable interface.

🡪start method :

* Creates a new thread and execute the run method in that thread.

Important Threads Methods:

1. start(): starts a new thread.

2. run(): code executed by the thread.

3. sleep(ms): pauses thread for some milliseconds.

4. join(): it waits for thread to finish.

5. isAlive(): checks thread the alive or not(running or not).

6. setName(String): sets a thread name.

7. getName(): returns the thread name.

8. setPriority(int): sets the priority (1 to 10) of the thread.

9. getPriority(): get the threads priority.

10.