JAVA PROGRAMS

24 October 2025 11:05

Number-Based Programs

```
// 1. Check if a number is Palindrome
int num = 121, rev = 0, temp = num;
while (temp != 0) {
   rev = rev * 10 + temp % 10;
   temp /= 10;
}
System.out.println(num == rev ? "Palindrome" : "Not Palindrome");
// 2. Check if a number is Armstrong (e.g., 153 = 1³ + 5³ + 3³)
int n = 153, sum = 0, temp2 = n;
while (temp2 != 0) {
   int digit = temp2 % 10;
   sum += digit * digit * digit;
   temp2 /= 10;
}
System.out.println(n == sum ? "Armstrong" : "Not Armstrong");
// 3. Print Fibonacci series up to n terms
int a = 0, b = 1, c, terms = 10;
System.out.print(a + " " + b);
for (int i = 2; i < terms; i++) {
   c = a + b;
   System.out.print(" " + c);
   a = b;
   b = c;
}
// 4. Check if a number is Prime
int num2 = 29, isPrime = 1;
for (int i = 2; i <= num2 / 2; i++) {
   if (num2 % i == 0) {
      isPrime = 0;
      break;
   }
}
System.out.println(isPrime == 1 ? "Prime" : "Not Prime");
// 5. Find factorial of a number
int fact = 1, number = 5;
for (int i = 1; i <= number; i++) {
   fact *= i;
}
System.out.println("Factorial: " + fact);
```

String-Based Programs

```
// 6. Reverse a string
String str = "hello";
String revStr = new StringBuilder(str).reverse().toString();
System.out.println("Reversed: " + revStr);
// 7. Check if a string is Palindrome
String s = "madam";
String reversed = new StringBuilder(s).reverse().toString();
System.out.println(s.equals(reversed) ? "Palindrome" : "Not Palindrome");
// 8. Count vowels and consonants
int vowels = 0, consonants = 0;
```

```
for (char ch : s.toLowerCase().toCharArray()) {
   if ("aeiou".indexOf(ch) != -1) vowels++;
   else if (Character.isLetter(ch)) consonants++;
}
System.out.println("Vowels: " + vowels + ", Consonants: " + consonants);
// 9. Remove duplicates from string
String input = "programming";
StringBuilder result = new StringBuilder();
for (char ch : input.toCharArray()) {
   if (result.indexOf(String.valueOf(ch)) == -1) result.append(ch);
}
System.out.println("Without duplicates: " + result);
// 10. Find frequency of each character
Map<Character, Integer> freq = new HashMap<>();
for (char ch : input.toCharArray()) {
   freq.put(ch, freq.getOrDefault(ch, 0) + 1);
}
System.out.println(freq);
```

Array-Based Programs

```
// 11. Find largest element in array
int[] arr = {3, 5, 1, 9, 2};
int max = arr[0];
for (int val : arr) {
   if (val > max) max = val;
}
System.out.println("Max: " + max);
// 12. Sort array (Bubble Sort)
for (int i = 0; i < arr.length - 1; i++) {
   for (int j = 0; j < arr.length - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
         int temp = arr[j];
         arr[j] = arr[j + 1];
         arr[j + 1] = temp;
      }
   }
}
System.out.println(Arrays.toString(arr));
// 13. Find second largest element
Arrays.sort(arr);
System.out.println("Second largest: " + arr[arr.length - 2]);
// 14. Reverse an array
for (int i = 0, j = arr.length - 1; i < j; i++, j--) {
   int temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
}
System.out.println("Reversed: " + Arrays.toString(arr));
// 15. Merge two arrays
int[] a1 = {1, 2}, a2 = {3, 4};
int[] merged = new int[a1.length + a2.length];
System.arraycopy(a1, 0, merged, 0, a1.length);
System.arraycopy(a2, 0, merged, a1.length, a2.length);
System.out.println("Merged: " + Arrays.toString(merged));
```

Loop & Recursion

```
// 16. Print pattern: pyramid
for (int i = 1; i <= 5; i++) {
```

```
    for (int j = 1; j <= i; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
// 17. Sum of digits using loop
int num3 = 1234, sumDigits = 0;
while (num3 != 0) {
    sumDigits += num3 % 10;
    num3 /= 10;
}
System.out.println("Sum of digits: " + sumDigits);
// 18. Sum of digits using recursion
int sumRec(int n) {
    return n == 0 ? 0 : n % 10 + sumRec(n / 10);
}
// 19. Factorial using recursion
int factorial(int n) {
    return n <= 1 ? 1 : n * factorial(n - 1);
}
// 20. Fibonacci using recursion
int fib(int n) {
    return n <= 1 ? n : fib(n - 1) + fib(n - 2);
}
```

Miscellaneous

```
// 21. Swap two numbers without temp
int x = 5, y = 10;
x = x + y;
y = x - y;
x = x - y;
System.out.println("x: " + x + ", y: " + y);
// 22. Check leap year
int year = 2024;
boolean isLeap = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
System.out.println(isLeap ? "Leap Year" : "Not Leap Year");
// 23. Count digits in a number
int count = 0, num4 = 12345;
while (num4 != 0) {
    count++;
    num4 /= 10;
}
System.out.println("Digits: " + count);
// 24. Reverse number using recursion
int reverseNum(int n, int rev) {
    return n == 0 ? rev : reverseNum(n / 10, rev * 10 + n % 10);
}
// 25. Check even or odd
int val = 7;
System.out.println(val % 2 == 0 ? "Even" : "Odd");
```

OOP Basics

```
// 26. Class with constructor
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
```

```
    void greet() {
        System.out.println("Hello " + name);
    }
}
// 27. Inheritance example
class Animal {
    void sound() { System.out.println("Animal sound"); }
}
class Dog extends Animal {
    void sound() { System.out.println("Bark"); }
}
// 28. Interface example
interface Drawable {
    void draw();
}
class Circle implements Drawable {
    public void draw() { System.out.println("Drawing Circle"); }
}
// 29. Method overloading
void greet() { System.out.println("Hi"); }
void greet(String name) { System.out.println("Hi " + name); }
// 30. Method overriding
class Parent {
    void show() { System.out.println("Parent"); }
}
class Child extends Parent {
    void show() { System.out.println("Child"); }
}
```

OOPS

1. Class and Object

```
// A class is a blueprint; an object is an instance of that blueprint.
class Car {
    String model;
    int year;
void display() {
        System.out.println("Model: " + model + ", Year: " + year);
    }
}
public class Main {
    public static void main(String[] args) {
        Car c1 = new Car(); // Object creation
        c1.model = "Honda City";
        c1.year = 2022;
        c1.display();
    }
}
```

2.Constructor

```
// Constructor initializes object state when it's created.
class Student {
    String name;
    int age;
Student(String n, int a) {
```

```
        name = n;
        age = a;
    }
void show() {
        System.out.println(name + " is " + age + " years old.");
    }
}
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Ravi", 20);
        s1.show();
    }
}
```

3. Method Overloading

```
// Same method name, different parameters.
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
double add(double a, double b) {
        return a + b;
    }
}
public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(2, 3));       // int version
        System.out.println(calc.add(2.5, 3.5));   // double version
    }
}
```

4. Inheritance

```
// Child class inherits properties and methods from parent class.
class Animal {
    void sound() {
        System.out.println("Animal makes sound");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks");
    }
}
public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.sound(); // inherited method
        d.bark();  // child method
    }
}
```

5. Method Overriding

```
// Child class provides its own version of a method.
class Vehicle {
    void run() {
```

```
    System.out.println("Vehicle is running");
  }
}
class Bike extends Vehicle {
  void run() {
    System.out.println("Bike is running safely");
  }
}
public class Main {
  public static void main(String[] args) {
    Vehicle v = new Bike(); // runtime polymorphism
    v.run(); // calls overridden method
  }
}
```

6. Encapsulation

```
// Wrapping data with access control using private fields and public methods.
class Account {
  private double balance;
public void deposit(double amount) {
    if (amount > 0) balance += amount;
  }
public double getBalance() {
    return balance;
  }
}
public class Main {
  public static void main(String[] args) {
    Account acc = new Account();
    acc.deposit(5000);
    System.out.println("Balance: " + acc.getBalance());
  }
}
```

7. Abstraction (Abstract Class)

```
// Abstract class can't be instantiated; it defines a template.
abstract class Shape {
  abstract void draw(); // abstract method
}
class Circle extends Shape {
  void draw() {
    System.out.println("Drawing Circle");
  }
}
public class Main {
  public static void main(String[] args) {
    Shape s = new Circle();
    s.draw();
  }
}
```

8. Interface

```
// Interface defines a contract; classes implement it.
interface Printable {
  void print();
}
```

```
class Document implements Printable {
   public void print() {
      System.out.println("Printing document...");
   }
}
public class Main {
   public static void main(String[] args) {
      Printable p = new Document();
      p.print();
   }
}
```

9. Polymorphism

```
// One interface, many implementations.
class Animal {
   void speak() {
      System.out.println("Animal speaks");
   }
}
class Cat extends Animal {
   void speak() {
      System.out.println("Meow");
   }
}
class Cow extends Animal {
   void speak() {
      System.out.println("Moo");
   }
}
public class Main {
   public static void main(String[] args) {
      Animal a;
      a = new Cat(); a.speak();
      a = new Cow(); a.speak();
   }
}
```

10. Static Keyword

```
// Static members belong to the class, not objects.
class Counter {
   static int count = 0;
Counter() {
      count++;
      System.out.println("Count: " + count);
   }
}
public class Main {
   public static void main(String[] args) {
      new Counter();
      new Counter();
      new Counter(); // count increases across all instances
   }
}
```

SQL PROGRAMS

24 October 2025 11:12

Employee Table

```
CREATE TABLE Employee (
   emp_id INT PRIMARY KEY,
   name VARCHAR(50),
   department_id INT,
   salary INT,
   hire_date DATE,
   manager_id INT
);
```

Department Table

```
CREATE TABLE Department (
   department_id INT PRIMARY KEY,
   dept_name VARCHAR(50)
);
```

Bonus Table

```
CREATE TABLE Bonus (
   emp_id INT,
   bonus_amount INT,
   FOREIGN KEY (emp_id) REFERENCES Employee(emp_id)
);
```

Sample Data

```
-- Departments
INSERT INTO Department VALUES (1, 'HR'), (2, 'IT'), (3, 'Finance');
-- Employees
INSERT INTO Employee VALUES
(101, 'Ravi', 1, 50000, '2020-01-15', NULL),
(102, 'Priya', 2, 60000, '2019-03-10', 101),
(103, 'Amit', 2, 55000, '2021-06-01', 101),
(104, 'Sara', 3, 70000, '2018-11-20', NULL),
(105, 'John', 1, 45000, '2022-02-05', 101);
-- Bonuses
INSERT INTO Bonus VALUES (101, 5000), (102, 3000), (104, 7000);
```

SQL Concepts with Queries

◇ 1. Basic Queries

```
SELECT * FROM Employee;
SELECT name, salary FROM Employee WHERE salary > 50000;
SELECT DISTINCT department_id FROM Employee;
```

◇ 2. Filtering and Sorting

```
SELECT * FROM Employee WHERE department_id = 2 ORDER BY salary DESC;
SELECT * FROM Employee WHERE name LIKE 'P%';
SELECT * FROM Employee WHERE salary BETWEEN 45000 AND 60000;
SELECT * FROM Employee WHERE manager_id IS NULL;
```

◇ 3. Aggregation

```
SELECT department_id, COUNT(*) AS emp_count FROM Employee GROUP BY department_id;
SELECT department_id, AVG(salary) FROM Employee GROUP BY department_id HAVING
```

```
AVG(salary) > 50000;
```

◇ 4. Joins

```
-- INNER JOIN
SELECT e.name, d.dept_name
FROM Employee e
JOIN Department d ON e.department_id = d.department_id;
-- LEFT JOIN
SELECT e.name, b.bonus_amount
FROM Employee e
LEFT JOIN Bonus b ON e.emp_id = b.emp_id;
-- RIGHT JOIN
SELECT e.name, b.bonus_amount
FROM Bonus b
RIGHT JOIN Employee e ON e.emp_id = b.emp_id;
-- FULL OUTER JOIN (if supported)
SELECT e.name, b.bonus_amount
FROM Employee e
FULL OUTER JOIN Bonus b ON e.emp_id = b.emp_id;
```

◇ 5. Subqueries

```
-- Scalar subquery
SELECT name FROM Employee
WHERE salary > (SELECT AVG(salary) FROM Employee);
-- IN subquery
SELECT name FROM Employee
WHERE department_id IN (SELECT department_id FROM Department WHERE dept_name = 'IT');
```

◇ 6. Set Operations

```
-- UNION
SELECT name FROM Employee
UNION
SELECT dept_name FROM Department;
-- UNION ALL
SELECT name FROM Employee
UNION ALL
SELECT dept_name FROM Department;
```

◇ 7. CASE Expression

```
SELECT name, salary,
  CASE
    WHEN salary >= 60000 THEN 'High'
    WHEN salary >= 50000 THEN 'Medium'
    ELSE 'Low'
  END AS salary_grade
FROM Employee;
```

◇ 8. Modifying Data

```
UPDATE Employee SET salary = salary + 2000 WHERE emp_id = 105;
DELETE FROM Bonus WHERE emp_id = 102;
```

◇ 9. Constraints

```
-- Add NOT NULL constraint
ALTER TABLE Employee MODIFY name VARCHAR(50) NOT NULL;
-- Add UNIQUE constraint
ALTER TABLE Employee ADD CONSTRAINT unique_name UNIQUE(name);
```

◇ 10. Views

```
CREATE VIEW HighEarners AS
SELECT name, salary FROM Employee WHERE salary > 60000;
```

```
SELECT * FROM HighEarners;
```

◇ 11. Index

```
CREATE INDEX idx_salary ON Employee(salary);
```

◇ 12. Transactions

```
BEGIN TRANSACTION;
UPDATE Employee SET salary = salary - 1000 WHERE emp_id = 103;
INSERT INTO Bonus VALUES (103, 2000);
COMMIT;
-- Use ROLLBACK instead of COMMIT to undo
```

◇ 13. Stored Procedure (syntax may vary by RDBMS)

```
-- SQL Server / MySQL
DELIMITER //
CREATE PROCEDURE GetAllEmployees()
BEGIN
  SELECT * FROM Employee;
END //
DELIMITER ;
CALL GetAllEmployees();
```

◇ 14. Trigger

```
-- Example: Log salary updates
CREATE TABLE SalaryLog (
    emp_id INT,
    old_salary INT,
    new_salary INT,
    changed_on DATETIME
);
DELIMITER //
CREATE TRIGGER trg_salary_update
BEFORE UPDATE ON Employee
FOR EACH ROW
BEGIN
  IF OLD.salary != NEW.salary THEN
    INSERT INTO SalaryLog VALUES (OLD.emp_id, OLD.salary, NEW.salary, NOW());
  END IF;
END;
//
DELIMITER ;
```