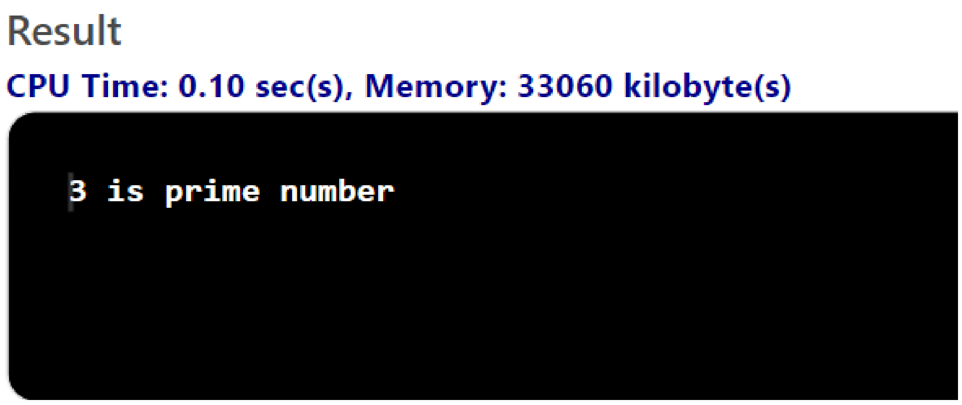
Coding Questions

By - Sakib Sadri

Q1 :- Write a program to find if a number is prime or not

```
public class Prime{
public static void main(String args[]){
  int i,m=0,flag=0;
  int n=3;
  m=n/2;
  if(n==0||n==1){
   System.out.println(n+" is not prime number");
  }else{
   for(i=2;i<=m;i++){
    if(n%i==0){
     System.out.println(n+" is not prime number");
     flag=1;
     break;
    }
   }
   if(flag==0)  { System.out.println(n+" is prime number"); }
  }
}
}
```

OUTPUT:-



Result

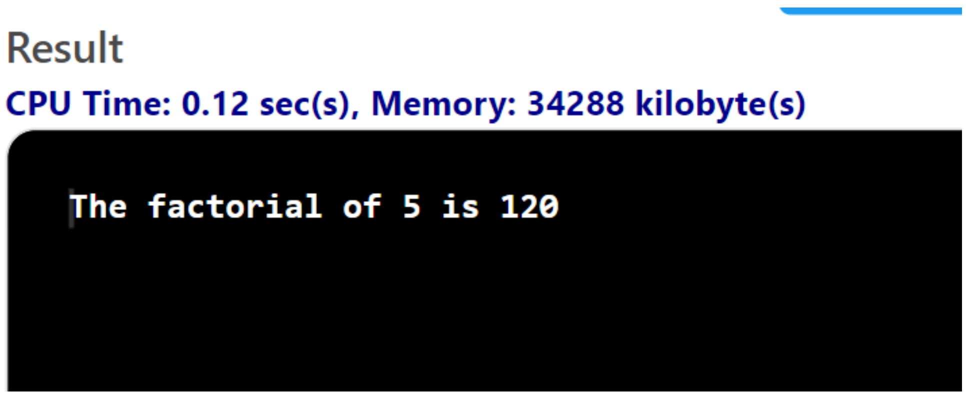
CPU Time: 0.10 sec(s), Memory: 33060 kilobyte(s)

```
3 is prime number
```

Q 2 Write a program to calculate the factorial of a given integer using a loop.

```
public class FactorialCalculator {
  public static void main(String[] args) {
    int number = 5; // Change this to the desired integer for which you want to calculate the factorial
    long factorial = calculateFactorial(number);
    System.out.println("The factorial of " + number + " is " + factorial);
  }

  public static long calculateFactorial(int n) {
    if (n == 0 || n == 1) {
      return 1;
    }
    long factorial = 1;
    for (int i = 2; i <= n; i++) {
      factorial *= i;
    }
    return factorial;
  }
}
```

OUTPUT :-



Result

CPU Time: 0.12 sec(s), Memory: 34288 kilobyte(s)

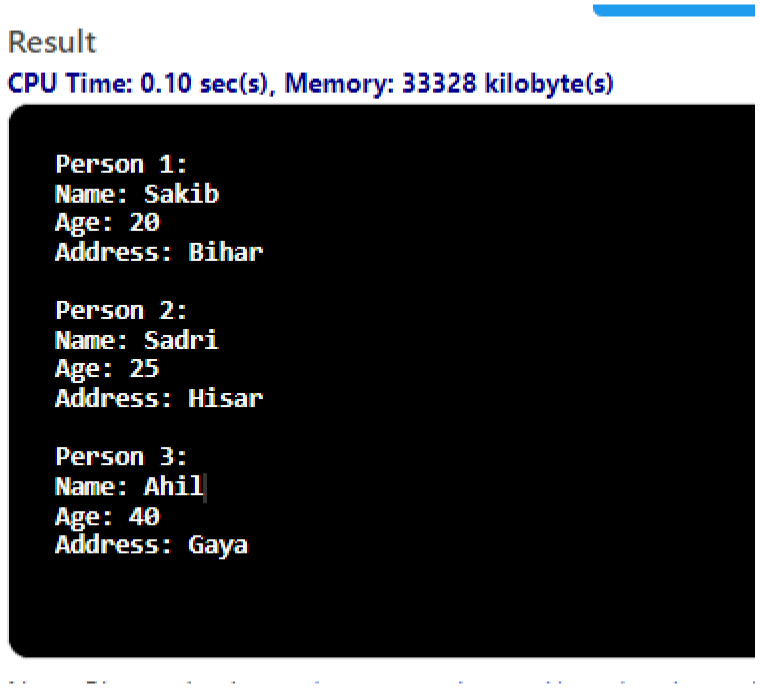
```
The factorial of 5 is 120
```

3 Create a Java class called Person with attributes like name, age, and address. Write a program that creates multiple Person objects, sets their attributes, and displays the information.

```
public class Person {
  private String name;
  private int age;
  private String address;
  public Person(String name, int age, String address) {
    this.name = name;
    this.age = age;
    this.address = address;
  }
  public void displayInfo() {
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Address: " + address);
  }
  public static void main(String[] args) {
    // Create multiple Person objects
    Person person1 = new Person("Sakib", 20, "Bihar");
```

```
        Person person2 = new Person("Sadri", 25, "Hisar");

        Person person3 = new Person("Ahil", 40, "Gaya");

        // Display information for each person

        System.out.println("Person 1:");

        person1.displayInfo();

        System.out.println();

        System.out.println("Person 2:");

        person2.displayInfo();

        System.out.println();

        System.out.println("Person 3:");

        person3.displayInfo();

    }

}
```

OUTPUT :-



Result

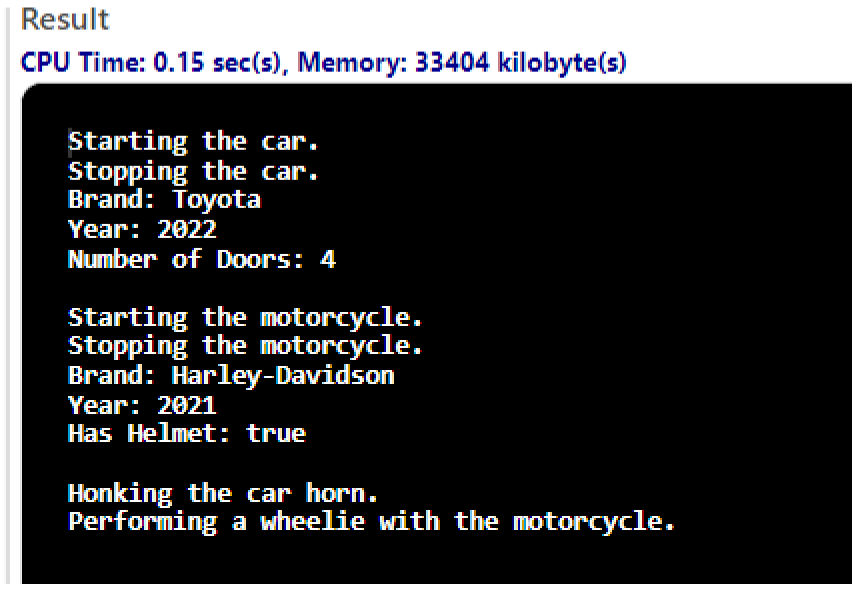
CPU Time: 0.10 sec(s), Memory: 33328 kilobyte(s)

```
Person 1:
Name: Sakib
Age: 20
Address: Bihar

Person 2:
Name: Sadri
Age: 25
Address: Hisar

Person 3:
Name: Ahil
Age: 40
Address: Gaya
```

4 Create a Java class hierarchy representing different types of vehicles, including a base class Vehicle and subclasses like Car and Motorcycle. Each class should have attributes and methods related to vehicles. Demonstrate method overriding and polymorphism

```
class Vehicle {
    private String brand;
    private int year;

    public Vehicle(String brand, int year) {
        this.brand = brand;
        this.year = year;
    }

    public void start() {
        System.out.println("Starting the vehicle.");
    }

    public void stop() {
        System.out.println("Stopping the vehicle.");
    }

    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Year: " + year);
    }
}

class Car extends Vehicle {
    private int numberOfDoors;
    public Car(String brand, int year, int numberOfDoors) {
        super(brand, year);
        this.numberOfDoors = numberOfDoors;
    }

    @Override
    public void start() {
        System.out.println("Starting the car.");
    }

    @Override
    public void stop() {
        System.out.println("Stopping the car.");
    }

    public void honk() {
        System.out.println("Honking the car horn.");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Number of Doors: " + numberOfDoors);
    }
}

class Motorcycle extends Vehicle {
    private boolean hasHelmet;
    public Motorcycle(String brand, int year, boolean hasHelmet) {
        super(brand, year);
        this.hasHelmet = hasHelmet;
    }

    @Override
    public void start() {
        System.out.println("Starting the motorcycle.");
    }

    @Override
    public void stop() {
        System.out.println("Stopping the motorcycle.");
    }

    public void wheelie() {
        System.out.println("Performing a wheelie with the motorcycle.");
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Has Helmet: " + hasHelmet);
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle vehicle1 = new Car("Toyota", 2022, 4);
        Vehicle vehicle2 = new Motorcycle("Harley-Davidson", 2021, true);

        // Demonstrate method overriding and polymorphism
        vehicle1.start();
        vehicle1.stop();
        vehicle1.displayInfo();
        System.out.println();

        vehicle2.start();
        vehicle2.stop();
        vehicle2.displayInfo();
        System.out.println();

        // Using type-specific methods
        if (vehicle1 instanceof Car) {
            ((Car) vehicle1).honk();
        }

        if (vehicle2 instanceof Motorcycle) {
            ((Motorcycle) vehicle2).wheelie();
        }
    }
}
```

OUTPUT :-


Result

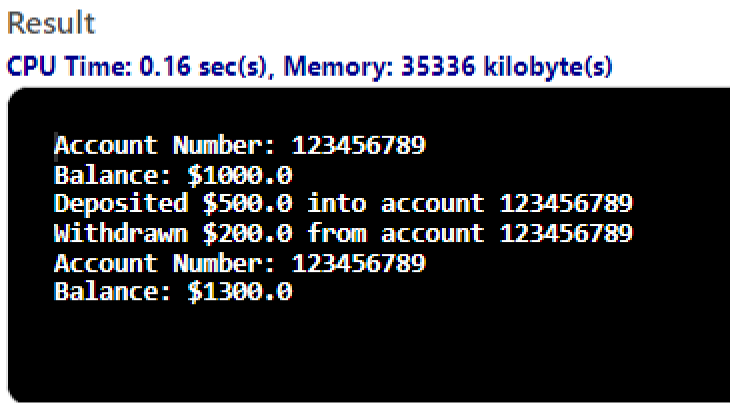
CPU Time: 0.15 sec(s), Memory: 33404 kilobyte(s)

```
Starting the car.
Stopping the car.
Brand: Toyota
Year: 2022
Number of Doors: 4

Starting the motorcycle.
Stopping the motorcycle.
Brand: Harley-Davidson
Year: 2021
Has Helmet: true

Honking the car horn.
Performing a wheelie with the motorcycle.
```

5 Create a Java class BankAccount with private attributes like accountNumber and balance. Implement methods to deposit, withdraw, and check the balance while encapsulating the internal state.

```java
public class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited $" + amount + " into account " + accountNumber);
        } else {
            System.out.println("Invalid deposit amount. Amount must be greater than 0.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            System.out.println("Withdrawn $" + amount + " from account " + accountNumber);
        } else if (amount <= 0) {
            System.out.println("Invalid withdrawal amount. Amount must be greater than 0.");
        } else {
            System.out.println("Insufficient funds to withdraw $" + amount + " from account " +
accountNumber);
        }
    }

    public double getBalance() {
        return balance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public void displayAccountInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: $" + balance);
    }
    public static void main(String[] args) {
        BankAccount account = new BankAccount("123456789", 1000.0);


        account.displayAccountInfo();
        account.deposit(500.0);
        account.withdraw(200.0);
        account.displayAccountInfo();
    }
}
```

OUTPUT :-



Result

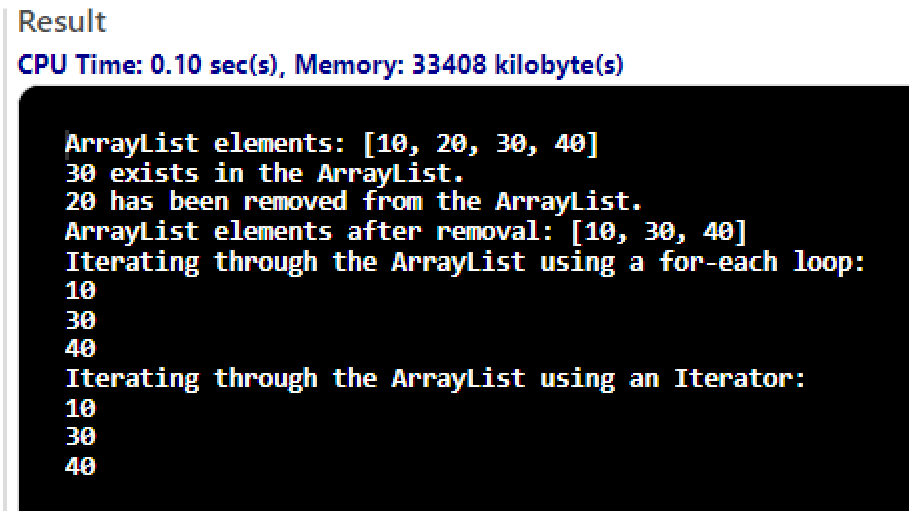
CPU Time: 0.16 sec(s), Memory: 35336 kilobyte(s)

```
Account Number: 123456789
Balance: $1000.0
Deposited $500.0 into account 123456789
Withdrawn $200.0 from account 123456789
Account Number: 123456789
Balance: $1300.0
```

6 Write a Java program that demonstrates various operations on an ArrayList. Perform operations like adding elements, removing elements, checking if an element exists, and iterating through the list.

```
import java.util.ArrayList;
import java.util.Iterator;
public class ArrayListOperations {
    public static void main(String[] args) {
        // Create an ArrayList of integers
        ArrayList<Integer> arrayList = new ArrayList<>();
        // Adding elements to the ArrayList
        arrayList.add(10);
        arrayList.add(20);
        arrayList.add(30);
        arrayList.add(40);
        // Display the ArrayList
        System.out.println("ArrayList elements: " + arrayList);
        // Checking if an element exists in the ArrayList
        int elementToCheck = 30;
        if (arrayList.contains(elementToCheck)) {
            System.out.println(elementToCheck + " exists in the ArrayList.");
        } else {
            System.out.println(elementToCheck + " does not exist in the ArrayList.");
        }
        // Removing an element from the ArrayList
        int elementToRemove = 20;
        if (arrayList.contains(elementToRemove)) {
            arrayList.remove(Integer.valueOf(elementToRemove)); // Use Integer.valueOf() to remove by
value
            System.out.println(elementToRemove + " has been removed from the ArrayList.");
        } else {
            System.out.println(elementToRemove + " does not exist in the ArrayList.");
        }
        // Display the ArrayList after removal
        System.out.println("ArrayList elements after removal: " + arrayList);
        // Iterating through the ArrayList using a for-each loop
        System.out.println("Iterating through the ArrayList using a for-each loop:");
        for (int number : arrayList) {
            System.out.println(number);
        }
        // Iterating through the ArrayList using an Iterator
        System.out.println("Iterating through the ArrayList using an Iterator:");
        Iterator<Integer> iterator = arrayList.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

OUTPUT :-



Result

CPU Time: 0.10 sec(s), Memory: 33408 kilobyte(s)

```
ArrayList elements: [10, 20, 30, 40]
30 exists in the ArrayList.
20 has been removed from the ArrayList.
ArrayList elements after removal: [10, 30, 40]
Iterating through the ArrayList using a for-each loop:
10
30
40
Iterating through the ArrayList using an Iterator:
10
30
40
```

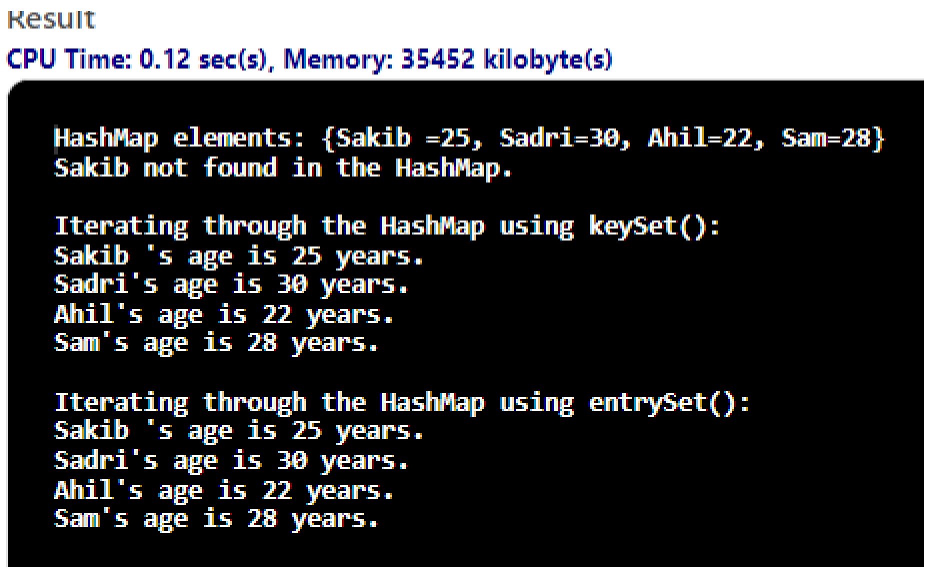
7 Create a Java program that demonstrates the use of a HashMap to store key-value pairs. Perform operations like adding key-value pairs, retrieving values by keys, and iterating through the map.

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
public class HashMapOperations {
    public static void main(String[] args) {
```

```
 // Create a HashMap to store key-value pairs (String to Integer)
 HashMap<String, Integer> hashMap = new HashMap<>();
 // Adding key-value pairs to the HashMap
 hashMap.put("Sakib ", 25);
 hashMap.put("Sadri", 30);
 hashMap.put("Ahil", 22);
 hashMap.put("Sam", 28);
 // Display the HashMap
 System.out.println("HashMap elements: " + hashMap);
 // Retrieving values by keys
 String nameToRetrieve = "Sakib";
 if (hashMap.containsKey(nameToRetrieve)) {
   int age = hashMap.get(nameToRetrieve);
   System.out.println(nameToRetrieve + "'s age is " + age + " years.");
 } else {
   System.out.println(nameToRetrieve + " not found in the HashMap.");
 }
System.out.println();
 // Iterating through the HashMap using keySet()
 System.out.println("Iterating through the HashMap using keySet():");
 Set<String> keys = hashMap.keySet();
 for (String key : keys) {
   int age = hashMap.get(key);
   System.out.println(key + "'s age is " + age + " years.");
 }
  System.out.println();
 // Iterating through the HashMap using entrySet()
 System.out.println("Iterating through the HashMap using entrySet():");
 Set<Map.Entry<String, Integer>> entrySet = hashMap.entrySet();
```

```
        Iterator<Map.Entry<String, Integer>> iterator = entrySet.iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Integer> entry = iterator.next();
            String key = entry.getKey();
            int age = entry.getValue();
            System.out.println(key + "'s age is " + age + " years.");
        }
    }
}
```

OUTPUT :-



Result

CPU Time: 0.12 sec(s), Memory: 35452 kilobyte(s)

```
HashMap elements: {Sakib =25, Sadri=30, Ahil=22, Sam=28}
Sakib not found in the HashMap.

Iterating through the HashMap using keySet():
Sakib 's age is 25 years.
Sadri's age is 30 years.
Ahil's age is 22 years.
Sam's age is 28 years.

Iterating through the HashMap using entrySet():
Sakib 's age is 25 years.
Sadri's age is 30 years.
Ahil's age is 22 years.
Sam's age is 28 years.
```

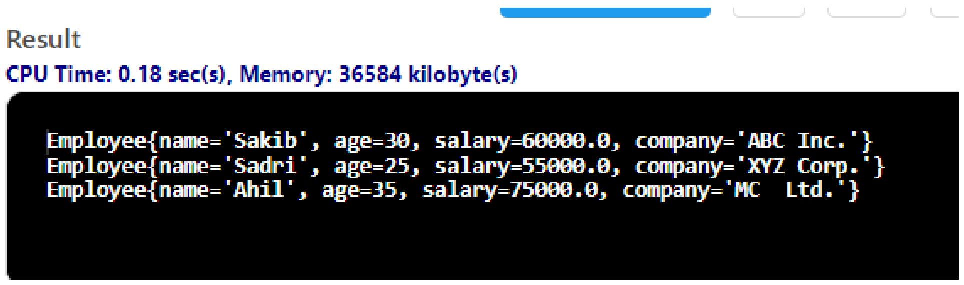
8 Create a list to store Employee class having attributes (name, age, salary, company).

```
import java.util.ArrayList;
import java.util.List;


class Employee {
```

```
private String name;
private int age;
private double salary;
private String company;

public Employee(String name, int age, double salary, String company) {
   this.name = name;
   this.age = age;
   this.salary = salary;
   this.company = company;
}

// Getters and setters (optional)
public String getName() {
   return name;
}

public int getAge() {
   return age;
}

public double getSalary() {
   return salary;
}

public String getCompany() {
   return company;
}
```

```
    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                ", company='" + company + '\'' +
                '}';
    }
}

public class EmployeeListDemo {
    public static void main(String[] args) {
        // Create a list to store Employee objects
        List<Employee> employeeList = new ArrayList<>();

        // Add Employee objects to the list
        employeeList.add(new Employee("Sakib", 30, 60000.0, "ABC Inc."));
        employeeList.add(new Employee("Sadri", 25, 55000.0, "XYZ Corp."));
        employeeList.add(new Employee("Ahil", 35, 75000.0, "MC  Ltd."));

        // Display the list of employees
        for (Employee employee : employeeList) {
            System.out.println(employee);
        }
    }
}
```

OUTPUT:-



CPU Time: 0.18 sec(s), Memory: 36584 kilobyte(s)

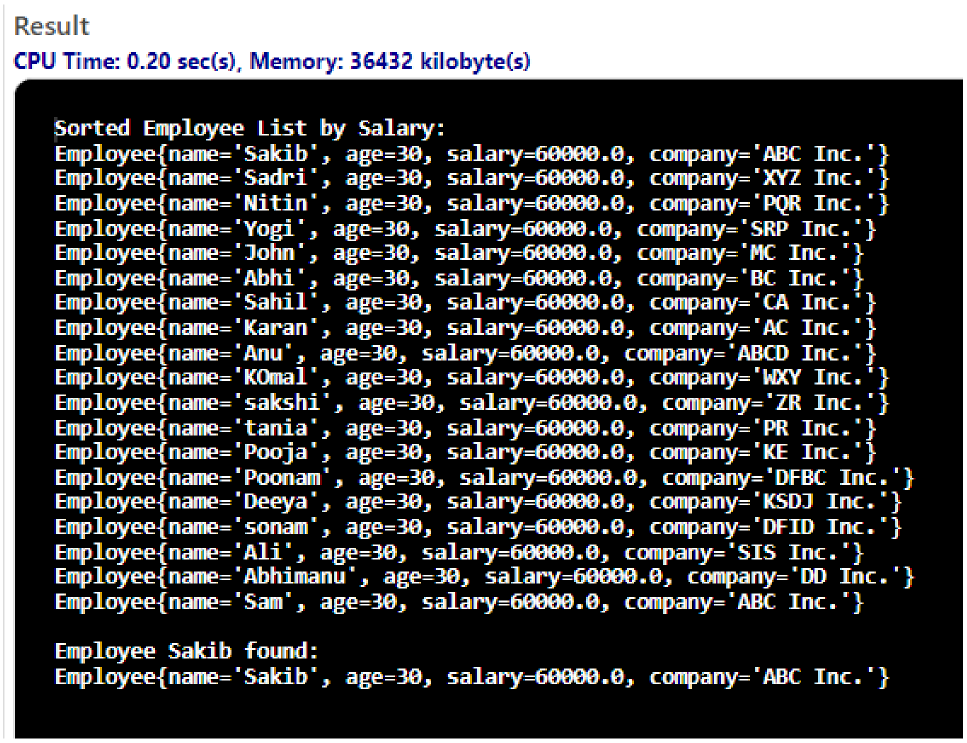
```
Employee{name='Sakib', age=30, salary=60000.0, company='ABC Inc.'}
Employee{name='Sadri', age=25, salary=55000.0, company='XYZ Corp.'}
Employee{name='Ahil', age=35, salary=75000.0, company='MC  Ltd.'}
```

9 In the above employee list store details of 20 employees, sort the employees based on salary, search for an employee on the list.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

class Employee {
  private String name;
  private int age;
  private double salary;
  private String company;

  public Employee(String name, int age, double salary, String company) {
    this.name = name;
    this.age = age;
    this.salary = salary;
    this.company = company;
  }

  // Getters and setters (optional)
  public String getName() {
    return name;
  }

  public int getAge() {
    return age;
  }

  public double getSalary() {
    return salary;
  }

  public String getCompany() {
    return company;
  }

  @Override
  public String toString() {
    return "Employee{" +
            "name='" + name + '\'' +
            ", age=" + age +
            ", salary=" + salary +
            ", company='" + company + '\'' +
            '}';
  }
}

public class EmployeeListDemo {
  public static void main(String[] args) {
    // Create a list to store Employee objects
```

```
List<Employee> employeeList = new ArrayList<>();

// Add Employee objects to the list
employeeList.add(new Employee("Sakib", 30, 60000.0, "ABC Inc."));
employeeList.add(new Employee("Sadri", 30, 60000.0, "XYZ Inc."));
 employeeList.add(new Employee("Nitin", 30, 60000.0, "PQR Inc."));
employeeList.add(new Employee("Yogi", 30, 60000.0, "SRP Inc."));
employeeList.add(new Employee("John", 30, 60000.0, "MC Inc."));
employeeList.add(new Employee("Abhi", 30, 60000.0, "BC Inc."));
employeeList.add(new Employee("Sahil", 30, 60000.0, "CA Inc."));
employeeList.add(new Employee("Karan", 30, 60000.0, "AC Inc."));
employeeList.add(new Employee("Anu", 30, 60000.0, "ABCD Inc."));
         employeeList.add(new Employee("KOmal", 30, 60000.0, "WXY Inc."));
          employeeList.add(new Employee("sakshi", 30, 60000.0, "ZR Inc."));
           employeeList.add(new Employee("tania", 30, 60000.0, "PR Inc."));
            employeeList.add(new Employee("Pooja", 30, 60000.0, "KE Inc."));
             employeeList.add(new Employee("Poonam", 30, 60000.0, "DFBC Inc."));
              employeeList.add(new Employee("Deeya", 30, 60000.0, "KSDJ Inc."));
               employeeList.add(new Employee("sonam", 30, 60000.0, "DFID Inc."));
                employeeList.add(new Employee("Ali", 30, 60000.0, "SIS Inc."));
                 employeeList.add(new Employee("Abhimanu", 30, 60000.0, "DD Inc."));
                  employeeList.add(new Employee("Sam", 30, 60000.0, "ABC Inc."));
// Add more employees here...
// Sort employees based on salary
Collections.sort(employeeList, (e1, e2) -> Double.compare(e1.getSalary(), e2.getSalary()));
// Display the sorted list of employees
System.out.println("Sorted Employee List by Salary:");
for (Employee employee : employeeList) {
    System.out.println(employee);
```

```
        }
        // Search for an employee by name (e.g., "Alice")
        String searchName = "Sakib";
        boolean found = false;
        for (Employee employee : employeeList) {
            if (employee.getName().equals(searchName)) {
                found = true;
                System.out.println("\nEmployee " + searchName + " found:");
                System.out.println(employee);
                break; // Exit the loop once found
            }
        }
        if (!found) {
            System.out.println("\nEmployee " + searchName + " not found.");
        }
    }
}
```



Result

CPU Time: 0.20 sec(s), Memory: 36432 kilobyte(s)

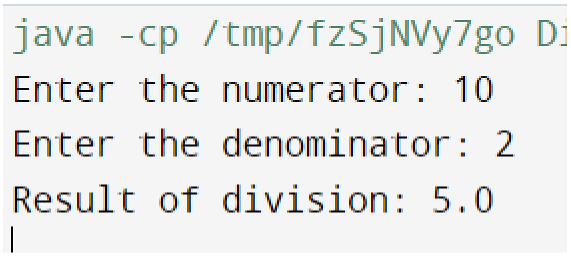
```
Sorted Employee List by Salary:
Employee{name='Sakib', age=30, salary=60000.0, company='ABC Inc.'}
Employee{name='Sadri', age=30, salary=60000.0, company='XYZ Inc.'}
Employee{name='Nitin', age=30, salary=60000.0, company='PQR Inc.'}
Employee{name='Yogi', age=30, salary=60000.0, company='SRP Inc.'}
Employee{name='John', age=30, salary=60000.0, company='MC Inc.'}
Employee{name='Abhi', age=30, salary=60000.0, company='BC Inc.'}
Employee{name='Sahil', age=30, salary=60000.0, company='CA Inc.'}
Employee{name='Karan', age=30, salary=60000.0, company='AC Inc.'}
Employee{name='Anu', age=30, salary=60000.0, company='ABCD Inc.'}
Employee{name='KOmal', age=30, salary=60000.0, company='WXY Inc.'}
Employee{name='sakshi', age=30, salary=60000.0, company='ZR Inc.'}
Employee{name='tania', age=30, salary=60000.0, company='PR Inc.'}
Employee{name='Pooja', age=30, salary=60000.0, company='KE Inc.'}
Employee{name='Poonam', age=30, salary=60000.0, company='DFBC Inc.'}
Employee{name='Deeya', age=30, salary=60000.0, company='KSDJ Inc.'}
Employee{name='sonam', age=30, salary=60000.0, company='DFID Inc.'}
Employee{name='Ali', age=30, salary=60000.0, company='SIS Inc.'}
Employee{name='Abhimanu', age=30, salary=60000.0, company='DD Inc.'}
Employee{name='Sam', age=30, salary=60000.0, company='ABC Inc.'}

Employee Sakib found:
Employee{name='Sakib', age=30, salary=60000.0, company='ABC Inc.'}
```

10 Write a Java program that demonstrates the use of a try-catch block to handle an ArithmeticException. Prompt the user to enter two integers and perform division, catching and handling the exception if division by zero occurs.

```
import java.util.Scanner;


public class DivisionDemo {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);


    try {

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

      int numerator = scanner.nextInt();


      System.out.print("Enter the denominator: ");
```

```
            int denominator = scanner.nextInt();

            // Perform division
            double result = divide(numerator, denominator);
            System.out.println("Result of division: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        } catch (Exception e) {
            System.out.println("Error: Invalid input.");
        } finally {
            scanner.close();
        }
    }

    public static double divide(int numerator, int denominator) {
        if (denominator == 0) {
            throw new ArithmeticException("Division by zero");
        }
        return (double) numerator / denominator;
    }
}
```

Output :-

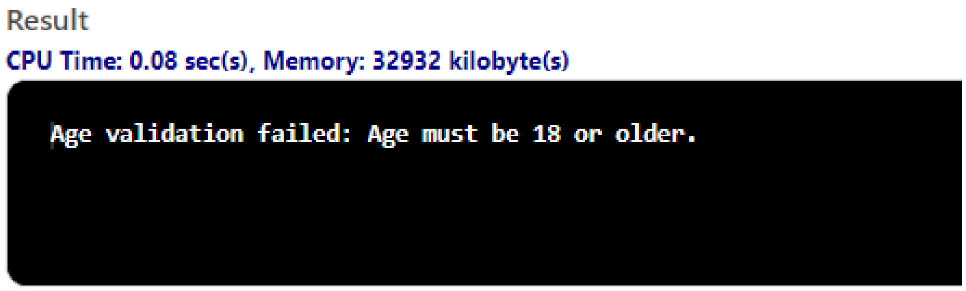


```
java -cp /tmp/fzSjNVy7go Di
Enter the numerator: 10
Enter the denominator: 2
Result of division: 5.0
```

11 Create a custom exception class called InvalidAgeException. Write a Java program that uses this custom exception to validate the age of a user. If the user enters an age less than 18, throw the InvalidAgeException.

```
class InvalidAgeException extends Exception {
  public InvalidAgeException(String message) {
    super(message);
  }
}

public class AgeValidator {
  public static void main(String[] args) {
    try {
      int age = getUserAge();
      validateAge(age);
      System.out.println("Age validation successful. You are eligible.");
    } catch (InvalidAgeException e) {
      System.out.println("Age validation failed: " + e.getMessage());
    }
  }

  public static int getUserAge() {
    // Simulate getting user age (replace this with your user input method)
    // For this example, let's assume the user is 16 years old.
    return 16;
  }

  public static void validateAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or older.");
        }
    }
}
```

Output:



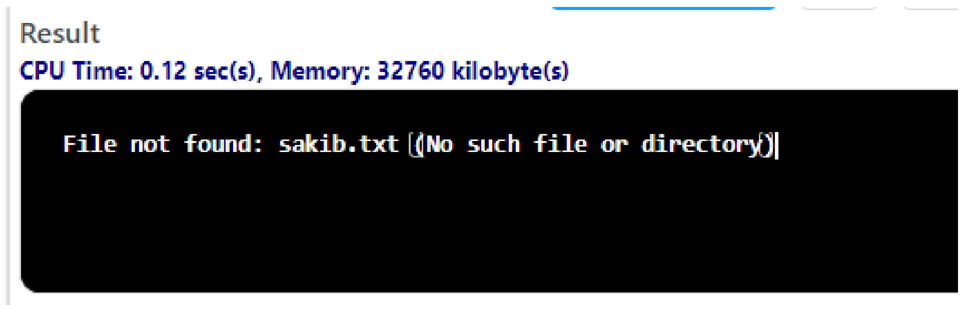
Result

CPU Time: 0.08 sec(s), Memory: 32932 kilobyte(s)

```
Age validation failed: Age must be 18 or older.
```

12 Write a Java program that attempts to read a file using FileInputStream, catching and handling any FileNotFoundException that may occur. Display an error message if the file is not found.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class FileReadDemo {
    public static void main(String[] args) {
        String fileName = "sakib.txt"; // Replace with the actual file name
        try {
            FileInputStream fileInputStream = new FileInputStream(fileName);
            // Read the file content (for demonstration purposes)
            int data;
            while ((data = fileInputStream.read()) != -1) {
                System.out.print((char) data);
            }
            // Close the FileInputStream when done
            fileInputStream.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error reading the file: " + e.getMessage());
        }
    }
}
```



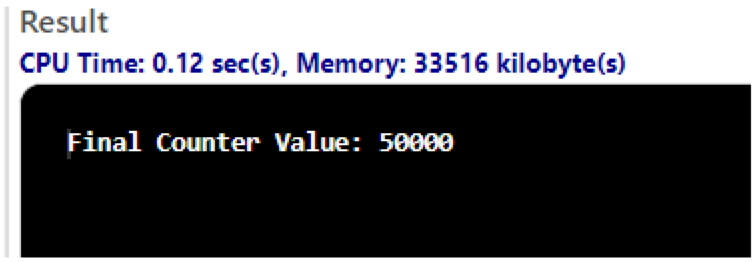
Result

CPU Time: 0.12 sec(s), Memory: 32760 kilobyte(s)

```
File not found: sakib.txt (No such file or directory)
```

13 Write a Java program that uses multithreading to create multiple threads, each of which increments a shared counter variable. The goal is to demonstrate how multiple threads can access and modify shared data concurrently. Ensure that the final value of the counter is the expected sum of increments from all threads.

```
public class MultiThreadedCounterDemo {
    public static void main(String[] args) {
        // Create a shared counter variable
        Counter counter = new Counter();
        // Number of threads to create
        int numThreads = 5;

        // Create and start multiple threads
        Thread[] threads = new Thread[numThreads];
        for (int i = 0; i < numThreads; i++) {
            threads[i] = new IncrementThread(counter);
            threads[i].start();
        }

        // Wait for all threads to complete
        try {
            for (Thread thread : threads) {
                thread.join();
            }
        } catch (InterruptedException e) {
            System.err.println("Thread interrupted: " + e.getMessage());
        }

        // Display the final value of the counter
        System.out.println("Final Counter Value: " + counter.getValue());
    }
}
class Counter {
    private int value = 0;
    public synchronized void increment() {
        value++;
    }
    public int getValue() {
        return value;
    }
}
class IncrementThread extends Thread {
    private Counter counter;
    public IncrementThread(Counter counter) {
        this.counter = counter;
    }
    @Override
    public void run() {
        for (int i = 0; i < 10000; i++) {
            counter.increment();
        }
    }
}
```

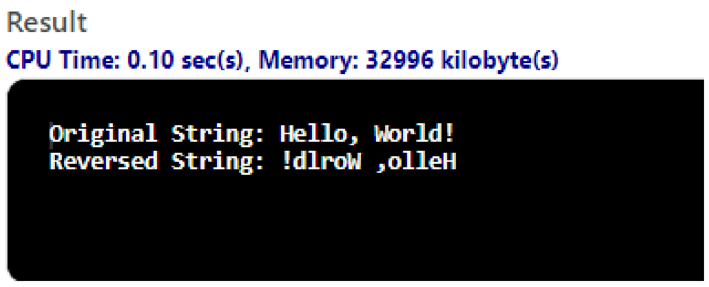

Result

CPU Time: 0.12 sec(s), Memory: 33516 kilobyte(s)

```
Final Counter Value: 50000
```

14 Write a Java program to reverse a given string without using any built-in reverse functions or libraries. You should implement your own logic to reverse the characters of the string.

```
public class ReverseString {
    public static void main(String[] args) {
        String originalString = "Hello, World!";
        String reversedString = reverse(originalString);
        System.out.println("Original String: " + originalString);
        System.out.println("Reversed String: " + reversedString);
    }
    public static String reverse(String str) {
        char[] charArray = str.toCharArray();
        int start = 0;
        int end = charArray.length - 1;
        while (start < end) {
            // Swap characters at start and end positions
            char temp = charArray[start];
            charArray[start] = charArray[end];
            charArray[end] = temp;
            // Move indices towards each other
            start++;
            end--;
        }
        return new String(charArray);
    }
}
```



Result

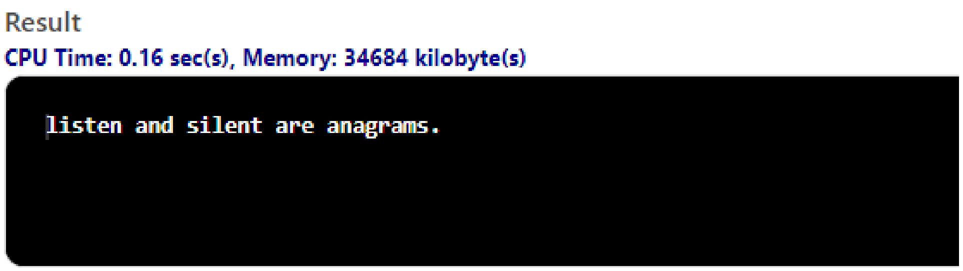
CPU Time: 0.10 sec(s), Memory: 32996 kilobyte(s)

```
Original String: Hello, World!
Reversed String: !dlroW ,olleH
```

15 Write a Java program to check if two strings are anagrams of each other. Anagrams are strings that have the same characters but in a different order.

```
import java.util.Arrays;
```

```
public class AnagramChecker {
    public static void main(String[] args) {
        String str1 = "listen";
        String str2 = "silent";

        if (areAnagrams(str1, str2)) {
            System.out.println(str1 + " and " + str2 + " are anagrams.");
        } else {
            System.out.println(str1 + " and " + str2 + " are not anagrams.");
        }
    }

    public static boolean areAnagrams(String str1, String str2) {
        // Remove spaces and convert to lowercase for case-insensitive comparison
        str1 = str1.replaceAll("\\s", "").toLowerCase();
        str2 = str2.replaceAll("\\s", "").toLowerCase();

        // Check if the lengths are equal
        if (str1.length() != str2.length()) {
            return false;
        }

        // Convert the strings to character arrays and sort them
        char[] charArray1 = str1.toCharArray();
        char[] charArray2 = str2.toCharArray();
        Arrays.sort(charArray1);
        Arrays.sort(charArray2);

        // Compare the sorted arrays
        return Arrays.equals(charArray1, charArray2);
    }
}
```



Result

CPU Time: 0.16 sec(s), Memory: 34684 kilobyte(s)

```
listen and silent are anagrams.
```