ASSIGNMENT-2

1.
```
public class StringReverser {
    public static String reverseString(String input) {
        StringBuilder reversed = new StringBuilder();
        for (int i = input.length() - 1; i >= 0; i--) {
            reversed.append(input.charAt(i));
        }
        return reversed.toString();
    }
    public static void main(String[] args) {
        String originalString = "Hello, Srk!";
        String reversedString = reverseString(originalString);
        System.out.println("Reversed string: " + reversedString);
    }
}
```

OUTPUT IS: Reversed string: !krS ,olleH

2.
```
public class LongestSubstringLength {
    public static int findLongestSubstringLength(String s) {
        int n = s.length();
        int maxLength = 0;
        int[] lastIndex = new int[256]; // Store the last index of each character
        // Initialize all characters' last index to -1
        for (int i = 0; i < 256; i++) {
            lastIndex[i] = -1;
        }
        int start = 0; // Start of the current substring
        for (int end = 0; end < n; end++) {
            char currentChar = s.charAt(end);
            // If the character is already seen, update the start index
            if (lastIndex[currentChar] >= start) {
                start = lastIndex[currentChar] + 1;
            }
```

```
            // Update the last index of the current character
            lastIndex[currentChar] = end;
            // Update the maximum length
            maxLength = Math.max(maxLength, end - start + 1);
        }
        return maxLength;
    }
    public static void main(String[] args) {
        String input = "srksrkaa";
        int result = findLongestSubstringLength(input);
        System.out.println("Length of the longest substring: " + result);
    }
}
```

OUTPUT: Length of the longest substring: 4

3.
```
import java.time.LocalDate;
  import java.time.Period;

public class AgeCalculator {
    public static void main(String[] args) {
        String birthdateString = "2001-07-21"; // Replace with the actual birthdate
        LocalDate birthdate = LocalDate.parse(birthdateString);
        LocalDate currentDate = LocalDate.now();

        Period age = Period.between(birthdate, currentDate);

        int years = age.getYears();
        int months = age.getMonths();
        int days = age.getDays();

        System.out.println("Age: " + years + " years, " + months + " months, " + days + " days");
    }
}
```

OUTPUT: Age: 22 years, 10 months, 30 days

4. import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.time.DayOfWeek;

```
public class DayOfWeekCalculator {
    public static String getDayOfWeek(String dateString) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        LocalDate date = LocalDate.parse(dateString, formatter);
        DayOfWeek dayOfWeek = date.getDayOfWeek();
        return dayOfWeek.toString();
    }

    public static void main(String[] args) {
        String inputDate = "2001-07-21"; // Replace with the desired date
        String dayOfWeek = getDayOfWeek(inputDate);
        System.out.println("Day of the week: " + dayOfWeek);
    }
}
```

OUTPUT : Day of the week: SATURDAY

5.
```
public class ArrayRotator {
    public static void rotateArray(int[] nums, int k) {
        int n = nums.length;
        k %= n; // Handle cases where k is larger than array length
        // Reverse the entire array
        reverse(nums, 0, n - 1);
        // Reverse the first k elements
        reverse(nums, 0, k - 1);
        // Reverse the remaining elements
        reverse(nums, k, n - 1);
    }
```

```
    private static void reverse(int[] nums, int start, int end) {
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6, 7,8,9,10};
        int k = 5;
        rotateArray(arr, k);

        System.out.print("Rotated array: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

OUTPUT: Rotated array: 6 7 8 9 10 1 2 3 4 5

6.
```
public class MissingNumberFinder {
    public static int findMissingNumber(int[] nums) {
        int n = nums.length;
        int expectedSum = n * (n + 1) / 2;
        int actualSum = 0;
        for (int num : nums) {
            actualSum += num;
        }
        return expectedSum - actualSum;
    }
    public static void main(String[] args) {
```

```
        int[] arr = {0, 1, 2, 4, 5}; // Example array (missing 3)
        int missingNumber = findMissingNumber(arr);
        System.out.println("Missing number: " + missingNumber);
    }
}
```

OUTPUT: Missing number: 3

7.

```
// Abstract class representing a Vehicle
abstract class Vehicle {
    private String make;
    private String model;
    public Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }
    // Abstract methods
    public abstract void accelerate();
    public abstract void brake();

    // Getters for make and model
    public String getMake() {
        return make;
    }
    public String getModel() {
        return model;
    }
}
// Car class (subclass of Vehicle)
class Car extends Vehicle {
    private int numOfDoors;
    public Car(String make, String model, int numOfDoors) {
        super(make, model);
        this.numOfDoors = numOfDoors;
```

```
    }
    @Override
    public void accelerate() {
        System.out.println("Car is accelerating...");
    }
    @Override
    public void brake() {
        System.out.println("Car is applying brakes...");
    }
    public void printNumOfDoors() {
        System.out.println("Number of doors: " + numOfDoors);
    }
}
// Motorcycle class (subclass of Vehicle)
class Motorcycle extends Vehicle {
    private boolean hasFairing;
    public Motorcycle(String make, String model, boolean hasFairing) {
        super(make, model);
        this.hasFairing = hasFairing;
    }
    @Override
    public void accelerate() {
        System.out.println("Motorcycle is revving up...");
    }
    @Override
    public void brake() {
        System.out.println("Motorcycle is slowing down...");
    }
}
public class Main {
    public static void main(String[] args) {
        // Create instances of Car and Motorcycle
```

```
        Car myCar = new Car("Toyota", "Camry", 4);
        Motorcycle myMotorcycle = new Motorcycle("Harley-Davidson", "Sportster", true);

        // Demonstrate behaviors
        System.out.println("Car details: " + myCar.getMake() + " " + myCar.getModel());
        myCar.printNumOfDoors(); // Print the number of doors
        myCar.accelerate();
        myCar.brake();
        System.out.println("\nMotorcycle details: " + myMotorcycle.getMake() + " " +
myMotorcycle.getModel());
        myMotorcycle.accelerate();
        myMotorcycle.brake();
    }
}
```

OUTPUT : Car details: Toyota Camry

Number of doors: 4

Car is accelerating...

Car is applying brakes...

Motorcycle details: Harley-Davidson Sportster

Motorcycle is revving up...

Motorcycle is slowing down...

8.

```
import java.util.ArrayList;
import java.util.List;
// Abstract class representing a LibraryMaterial
abstract class LibraryMaterial {
    private String title;
    private String author;
    public LibraryMaterial(String title, String author) {
        this.title = title;
        this.author = author;
    }
    // Abstract methods
```

```
    public abstract void checkout();
    public abstract void returnMaterial();
    // Getters for title and author
    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }
}
// Book class (subclass of LibraryMaterial)
class Book extends LibraryMaterial {
    private int numberOfPages;
    public Book(String title, String author, int numberOfPages) {
        super(title, author);
        this.numberOfPages = numberOfPages;
    }
    @Override
    public void checkout() {
        System.out.println("Book '" + getTitle() + "' checked out.");
    }
    @Override
    public void returnMaterial() {
        System.out.println("Book '" + getTitle() + "' returned.");
    }
}
// Magazine class (subclass of LibraryMaterial)
class Magazine extends LibraryMaterial {
    private int issueNumber;
    public Magazine(String title, String author, int issueNumber) {
        super(title, author);
        this.issueNumber = issueNumber;
```

```
    }

    @Override
    public void checkout() {
        System.out.println("Magazine '" + getTitle() + "' checked out.");
    }
    @Override
    public void returnMaterial() {
        System.out.println("Magazine '" + getTitle() + "' returned.");
    }
}
// Audiobook class (subclass of LibraryMaterial)
class Audiobook extends LibraryMaterial {
    private String narrator;
    public Audiobook(String title, String author, String narrator) {
        super(title, author);
        this.narrator = narrator;
    }
    @Override
    public void checkout() {
        System.out.println("Audiobook '" + getTitle() + "' checked out.");
    }
    @Override
    public void returnMaterial() {
        System.out.println("Audiobook '" + getTitle() + "' returned.");
    }
}
// LibrarySystem class
class LibrarySystem {
    private List<LibraryMaterial> materials = new ArrayList<>();
    public void addMaterial(LibraryMaterial material) {
        materials.add(material);
```

```
    }

    public void displayAvailableMaterials() {
        System.out.println("Available materials:");
        for (LibraryMaterial material : materials) {
            System.out.println(material.getTitle() + " by " + material.getAuthor());
        }
    }
}
public class Main {
    public static void main(String[] args) {
        LibrarySystem library = new LibrarySystem();
        // Add materials to the library
        library.addMaterial(new Book("The Great Gatsby", "F. Scott Fitzgerald", 1145));
        library.addMaterial(new Magazine("National Geographic", "Various Authors", 1113));
        library.addMaterial(new Audiobook("Harry Potter and the Sorcerer's Stone", "J.K. Rowling",
"Jim Dale"));
        // Display available materials
        library.displayAvailableMaterials();
        // Demonstrate checkout and return
        LibraryMaterial book = new Book("The Great Gatsby", "F. Scott Fitzgerald", 180);
        book.checkout();
        book.returnMaterial();
    }
}
```

OUTPUT: Available materials:

The Great Gatsby by F. Scott Fitzgerald

National Geographic by Various Authors

Harry Potter and the Sorcerer's Stone by J.K. Rowling

Book 'The Great Gatsby' checked out.

Book 'The Great Gatsby' returned.

9. import java.util.ArrayList;
import java.util.List;

// Custom exceptions
class InvalidEmployeeDataException extends Exception {
    public InvalidEmployeeDataException(String message) {
        super(message);
    }
}

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

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

// Employee class
class Employee {
    private String id;
    private String name;
    private double salary;
    private static final double MINIMUM_SALARY = 1000; // Example minimum salary threshold

    public Employee(String id, String name, double salary) {
        this.id = id;
        this.name = name;

```
        this.salary = salary;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    public static double getMinimumSalary() {
        return MINIMUM_SALARY;
    }
}

// HRManager class
class HRManager {
    private List<Employee> employeeList;

    public HRManager() {
        employeeList = new ArrayList<>();
    }

    public void addEmployee(String id, String name, double salary)
            throws InvalidEmployeeDataException, EmployeeAlreadyExistsException,
SalaryBelowMinimumException {
        // Validate employee data
        if (id == null || name == null || id.isEmpty() || name.isEmpty() || salary <= 0) {
```

```
            throw new InvalidEmployeeDataException("Invalid employee data provided");
        }

        // Check if employee already exists
        for (Employee employee : employeeList) {
            if (employee.getId().equals(id)) {
                throw new EmployeeAlreadyExistsException("Employee with ID " + id + " already
exists");
            }
        }

        // Check salary threshold
        if (salary < Employee.getMinimumSalary()) {
            throw new SalaryBelowMinimumException("Salary cannot be below the minimum threshold:
" +
                    Employee.getMinimumSalary());
        }

        // Add employee to the list
        employeeList.add(new Employee(id, name, salary));
        System.out.println("Employee added successfully: " + name);
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        HRManager hrManager = new HRManager();

        // Test adding employees
        try {
            hrManager.addEmployee("1001", "John Doe", 1500);
            hrManager.addEmployee("1002", "Jane Smith", 800);
```

```
        } catch (InvalidEmployeeDataException | EmployeeAlreadyExistsException |
                SalaryBelowMinimumException e) {
            System.out.println("Failed to add employee: " + e.getMessage());
        }
    }
}
```

18.
```
import java.util.LinkedList;
public class LinkedListExample {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();
        // Adding elements at the end of the list
        list.add(10);
        list.addLast(20);
        list.offer(30);
        list.offerLast(40);

        // Printing the elements of the list
        System.out.println("Elements after adding at the end: " + list); // Output: [10, 20, 30, 40]

        // Adding elements at the beginning of the list
        list.offerFirst(1);
        list.addFirst(2);

        // Printing the elements of the list
        System.out.println("Elements after adding at the beginning: " + list); // Output: [2, 1, 10, 20, 30,
40]
    }
}
```

Output:

```
Elements after adding at the end: [10, 20, 30, 40]
Elements after adding at the beginning: [2, 1, 10, 20, 30, 40]
```

19.
```
import java.util.LinkedList;
```

```
public class LinkedListExample {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<String>();

        // Adding elements at the end of the list
        list.add("ONE");
        list.add("TWO");
        list.add("THREE");
        list.add("FOUR");

        // Printing the elements of the list
        System.out.println("Original list: " + list); // Output: [ONE, TWO, THREE, FOUR]

        // Replacing an element at index 2 with "ZERO"
        list.set(2, "ZERO");

        // Printing the updated list
        System.out.println("Updated list: " + list); // Output: [ONE, TWO, ZERO, FOUR]
    }
}
```

Output:

Original list: [ONE, TWO, THREE, FOUR]

Updated list: [ONE, TWO, ZERO, FOUR]

21.
```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 2, 4, 3, 5);
        List<Integer> distinctNumbers = numbers.stream()
                .distinct().collect(Collectors.toList());
```

```
        System.out.println("Distinct numbers: " + distinctNumbers);
    }
}
```

OUTPUT:

Distinct numbers: [1, 2, 3, 4, 5]

22.

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
public class Main {
    public static void main(String[] args) {
        List<Double> decimals = Arrays.asList(3.14, 1.23, 2.71, 0.99, 4.56);
        List<Double> sortedDecimals = decimals.stream()
                .sorted((a, b) -> Double.compare(b, a)) // Sort in descending order
                .collect(Collectors.toList());
        System.out.println("Sorted decimals (in reverse order): " + sortedDecimals);
    }
}
```

OUTPUT: Sorted decimals (in reverse order): [4.56, 3.14, 2.71, 1.23, 0.99]

23.

```
import java.util.Arrays;
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(10, 15, 20,11,12,56 ,25, 30, 35, 40);
        System.out.println("Multiples of 5:");
        numbers.stream()
                .filter(num -> num % 5 == 0)
                .forEach(System.out::println);
    }
}
```

OUTPUT: Multiples of 5:

10

15

20

25

30

35

40

24. import java.util.Arrays;

import java.util.List;

public class Main {

```
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(10, 5, 20, 15, 30, 25);
        // Find maximum value
        int max = numbers.stream()
                .mapToInt(Integer::intValue)
                .max()
                .orElse(Integer.MIN_VALUE);
        // Find minimum value
        int min = numbers.stream()
                .mapToInt(Integer::intValue)
                .min()
                .orElse(Integer.MAX_VALUE);
        System.out.println("Maximum value: " + max);
        System.out.println("Minimum value: " + min);
    }}
```

OUTPUT: Maximum value: 30

Minimum value: 5

25. public class Main {

```
    public static void main(String[] args) {
        int number = 12345; // Replace with your desired number
        int sum = String.valueOf(number)
                .chars() // Convert to IntStream of character codes
                .map(Character::getNumericValue) // Convert character codes to integers
```

```
            .sum(); // Calculate the sum
        System.out.println("Sum of digits: " + sum);
    }
}
```

OUTPUT: Sum of digits: 15

26.
```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        int[] numbers = {10, 5, 20, 15, 30, 25};
        int secondLargest = Arrays.stream(numbers)
                .boxed() // Convert to Integer stream
                .sorted((a, b) -> b - a) // Sort in descending order
                .skip(1) // Skip the largest element
                .findFirst() // Get the second largest
                .orElse(Integer.MIN_VALUE); // Handle empty array case
        System.out.println("Second largest number: " + secondLargest);
    }
}
```

OUTPUT: Second largest number: 25

27.
```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};

        // Calculate the sum
        int sum = Arrays.stream(numbers).sum();
        // Calculate the average
        double average = Arrays.stream(numbers)
                .average()
                .orElse(Double.NaN); // Handle empty array case
        System.out.println("Sum: " + sum);
        System.out.println("Average: " + average);
```

```
    }
}
```

OUTPUT: Sum: 150

Average: 30.0

28.
```
import java.util.Arrays;
import java.util.stream.Collectors;
public class Main {
    public static void main(String[] args) {
        String input = "Siva Rama Krishna!";
        String reversedWords = Arrays.stream(input.split("\\s+")) // Split by whitespace
                .map(word -> new StringBuilder(word).reverse().toString())
                .collect(Collectors.joining(" "));
        System.out.println("Reversed words: " + reversedWords);
    }
}
```

OUTPUT: Reversed words: aviS amaR !anhsirK