

Lambdas and Functional Programming

Q1. Create a functional interface Calculator with methods for addition, subtraction, multiplication, and division. Implement these methods using lambda expressions. Define the **Calculator functional interface** with methods for arithmetic operations. Implement the interface methods using lambda expressions for basic arithmetic operations.

```
import java.util.*;

@FunctionalInterface
interface Calculator {
    int calculate(int a, int b);
}

public class ArithmeticOperations {
    public static void main(String[] args) {

        Calculator add = (a, b) -> a + b;
        Calculator subtract = (a, b) -> a - b;
        Calculator multiply = (a, b) -> a * b;

        Calculator divide = (a, b) -> {
            if (b == 0) {
                throw new ArithmeticException("Division by zero is not allowed.");
            }
            return a / b;
        };

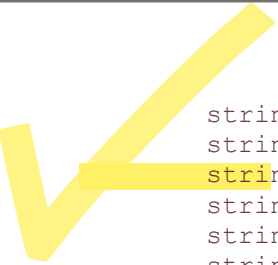
        int num1 = 10;
        int num2 = 5;

        System.out.println("Addition of " + num1 + " and " + num2 + " is: " +
            add.calculate(num1, num2));
        System.out.println("Subtraction of " + num1 + " and " + num2 + " is: " +
            subtract.calculate(num1, num2));
        System.out.println("Multiplication of " + num1 + " and " + num2 + " is: " +
            multiply.calculate(num1, num2));
        System.out.println("Division of " + num1 + " and " + num2 + " is: " +
            divide.calculate(num1, num2));
    }
}
```

Q2. Write a program that sorts a list of strings based on their lengths in descending order. Define a custom comparator using a lambda expression that compares strings based on their lengths. Use the custom comparator to sort the list of strings in descending order of length.

```
import java.util.*;

public class StringLengthSorter {
    public static void main(String[] args) {
        // Create a list of strings
        List<String> strings = new ArrayList<>();
        strings.add("India");
    }
}
```



```

strings.add("Canada");
strings.add("RKL");
strings.add("BBSR");
strings.add("UnitedKingdom");
strings.add("Fig");
strings.add("Nepal");

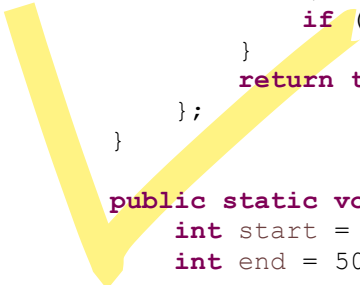
// Define a custom comparator using a lambda expression
Comparator<String> lengthComparator = (s1, s2) -> Integer.compare(s2.length(), s1.length());

// Sort the list of strings in descending order of length
Collections.sort(strings, lengthComparator);

// Print the sorted list
System.out.println("Strings sorted by length in descending order:");
for (String str : strings) {
    System.out.println(str);
}
}

```

Q3. Write a program that creates a sequence of prime numbers. Define a sequence using lambda expressions that generate prime numbers.



```

import java.util.function.IntPredicate;
import java.util.stream.IntStream;

public class PrimeNumberClass {
    public static IntPredicate primeNumbersInRange(int start, int end) {
        return number -> {
            if (number <= 1) return false;
            if (number == 2) return true;
            if (number % 2 == 0) return false;
            for (int i = 3; i <= Math.sqrt(number); i += 2) {
                if (number % i == 0) return false;
            }
            return true;
        };
    }

    public static void main(String[] args) {
        int start = 10;
        int end = 50;

        System.out.println("Prime numbers between " + start + " and " + end + ":");
        IntStream.rangeClosed(start, end)
            .filter(primeNumbersInRange(start, end))
            .forEach(System.out::println);
    }
}

```

N.B: The `primeNumbersInRange` method returns an `IntPredicate`, which is a functional interface representing a predicate (boolean-valued function) of one `int`-valued argument. This lambda expression checks if a number is prime by checking divisibility up to the square root of the number.

In the main method, we define the start and end of the range. We use `IntStream.rangeClosed(start, end)` to generate a stream of integers from start to end (inclusive). The stream is filtered using the prime-NumbersInRange predicate to retain only prime numbers. Finally, each prime number is printed using `forEach(System.out::println)`.

Q4. Create a functional interface Shape with a method `double area()` and a default method `void printArea()`. Implement the interface using lambda expressions for different shapes. Define the Shape functional interface with an area method. Implement the interface for shapes like circle, square, and rectangle using lambda expressions. Use the default method to print the area of each shape.

```
@FunctionalInterface
interface Shape {
    double area();

    default void printArea() {
        System.out.println("The area is: " + area());
    }
}
```

```
public class ShapeTest {
    public static void main(String[] args) {
        // Circle with radius 5
        Shape circle = () -> {
            double radius = 5;
            return Math.PI * radius * radius;
        };
    }
}
```

```
    // Square with side length 4
    Shape square = () -> {
        double side = 4;
        return side * side;
    };
}
```

```
    // Rectangle with width 3 and height 7
    Shape rectangle = () -> {
        double width = 3;
        double height = 7;
        return width * height;
    };
}
```

```
    // Print the area of each shape
    System.out.println("Circle:");
    circle.printArea();
}
```

```
    System.out.println("Square:");
    square.printArea();
}
```

```
    System.out.println("Rectangle:");
    rectangle.printArea();
}
```

```
public class q5ListStringUppercase {
    public static void main(String []args) {
        List<String> strings = Arrays.asList("apple",
        banana", "orange", "grape");
        strings.stream().map(String::
        toUpperCase)//convert to uppercase
        .filter(str-> !str.matches("^[AEIOU].*"))//filter out
        strings starting with vowels
        .forEach(System.out::println);
    }
}
```

Q5. Write a program that reads a list of strings, converts them to uppercase, filters out the strings starting with a vowel, and then prints the remaining strings.

```
import java.util.*;
import java.util.stream.Collectors;
```

```

public class StringTransformation {
    public static void main(String[] args) {
        List<String> strings = Arrays.asList("apple", "banana", "orange", "grape",
"kiwi", "melon");

        List<String> transformedStrings = strings.stream()
            .map(String::toUpperCase) // Convert to uppercase
            .filter(str -> !str.matches("[AEIOU].*")) // Filter out strings
starting with a vowel
            .collect(Collectors.toList()); // Collect the remaining strings in-
to a list

        System.out.println("Transformed strings:");
        transformedStrings.forEach(System.out::println); // Print the remaining
strings
    }
}

```

Q6. Write a program that demonstrates a function returning another function, where the inner function calculates the square of a number. Define a function that returns another function. The inner function should calculate the square of a given number. Demonstrate the use of the returned function to calculate squares.

```

import java.util.function.Function;

public class FunctionReturningFunction {

    // Define a function that returns another function
    public static Function<Integer, Integer> squareFunction() {
        return (Integer n) -> n * n;
    }

    public static void main(String[] args) {
        // Get the square function
        Function<Integer, Integer> squareFn = squareFunction();

        // Demonstrate the use of the returned function to calculate squares
        int number1 = 5;
        int number2 = 10;

        int square1 = squareFn.apply(number1);
        int square2 = squareFn.apply(number2);

        System.out.println("Square of " + number1 + " is: " + square1);
        System.out.println("Square of " + number2 + " is: " + square2);
    }
}

```

Q7. Write a program that calculates factorial using a recursive lambda expression. Define a recursive lambda expression to calculate factorial. Use the lambda expression to calculate factorial of a given number.

```

import java.util.function.Function;

public class RecursiveFactorial {

```

```

public static void main(String[] args) {
    // Define a recursive lambda expression to calculate factorial
    Function<Integer, Integer> factorial = new Function<>() {
        @Override
        public Integer apply(Integer n) {
            return n == 0 ? 1 : n * this.apply(n - 1);
        }
    };

    // Test the lambda expression to calculate factorial of a given number
    int num = 5;
    int result = factorial.apply(num);

    System.out.println("Factorial of " + num + " is: " + result);
}

```

Q8. Write a program that creates a thread using a lambda expression as the Runnable and prints "Hello, CSW2!" from the thread. Define a lambda expression that implements the Runnable interface and prints "Hello, CSW2!". Create a thread using the lambda expression and start the thread.

```

public class LambdaThreadExample {
    public static void main(String[] args) {
        // Define a lambda expression as the Runnable
        Runnable runnable = () -> {
            System.out.println(Thread.currentThread().getName() + " is running...");
            System.out.println("Hello, CSW2!");
        };

        // Create and start the thread using the lambda expression
        Thread thread = new Thread(runnable);
        thread.start();
    }
}

```

Q9. Write a program that implements the producer-consumer problem using threads. Define a bounded buffer class that implements a queue. Create producer and consumer classes that produce and consume items from the bounded buffer. Create and start multiple producer and consumer threads to demonstrate the producer-consumer problem.

```

import java.util.LinkedList;

class Buffer {
    private final LinkedList<Integer> list = new LinkedList<>();
    private final int capacity;

    public Buffer(int capacity) {
        this.capacity = capacity;
    }

    public synchronized void put(int item) throws InterruptedException {
        while (list.size() == capacity) {
            wait(); // Wait if buffer is full
        }
        list.add(item);
        System.out.println("Produced: " + item);
    }
}

```

```

        notify(); // Notify all waiting threads
    }

    public synchronized int get() throws InterruptedException {
        while (list.size() == 0) {
            wait(); // Wait if buffer is empty
        }
        int item = list.removeFirst();
        System.out.println("Consumed: " + item);
        notify(); // Notify all waiting threads
        return item;
    }
}

class ProducerThread extends Thread {
    private final Buffer buffer;

    public ProducerThread(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        try {
            for (int i = 0; i <= 4; i++) {
                buffer.put(i);
                Thread.sleep(2000); // Simulating work
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class ConsumerThread extends Thread {
    private final Buffer buffer;

    public ConsumerThread(Buffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        try {
            for (int i = 0; i <= 4; i++) {
                buffer.get();
                Thread.sleep(2000); // Simulating work
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class ProducerConsume {
    public static void main(String[] args) throws InterruptedException {
        Buffer buffer = new Buffer(4); // Buffer with capacity 2

        // Create and start producer and consumer threads
        ProducerThread producer = new ProducerThread(buffer);
        ConsumerThread consumer = new ConsumerThread(buffer);
    }
}

```

```

        producer.start();
        consumer.start();

        // Ensure both threads finish
        producer.join();
        consumer.join();
    }
}

```

Q10. Write a Java program that takes a `LocalDateTime` object and formats it to a custom string format ("yyyy-MM-dd HH:mm:ss"). Define a `LocalDateTime` object representing a specific date and time. Use `DateTimeFormatter` to format the `LocalDateTime` object to the desired string format.

```

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class LocalDateTimeFormatting {
    public static void main(String[] args) {
        // Define a LocalDateTime object representing a specific date and time
        LocalDateTime dateTime = LocalDateTime.of(2024, 5, 31, 15, 30, 45);

        // Use DateTimeFormatter to format the LocalDateTime object to the desired
        string format
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
        HH:mm:ss");
        String formattedDateTime = dateTime.format(formatter);

        // Print the formatted LocalDateTime
        System.out.println("Formatted LocalDateTime: " + formattedDateTime);
    }
}

```

Q11. Write a Java program that converts a given date and time in UTC to the local date and time of a specific time zone (e.g., "America/New_York"). Parse a string representing a date and time in UTC format. Convert the parsed `Instant` to a `ZonedDateTime` using a specific `ZoneId`.

```

import java.time.Instant;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

public class UTCtoLocalDateTime {
    public static void main(String[] args) {
        // Parse a string representing a date and time in UTC format
        String utcDateTimeString = "2024-05-31T10:15:30Z";
        Instant utcInstant = Instant.parse(utcDateTimeString);

        // Convert the parsed Instant to a ZonedDateTime using a specific ZoneId
        ZoneId zoneId = ZoneId.of("America/New_York");
        ZonedDateTime localDateTime = utcInstant.atZone(zoneId);

        // Format the local date and time
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
        HH:mm:ss");
        String formattedDateTime = localDateTime.format(formatter);
    }
}

```

```

        // Print the converted local date and time
        System.out.println("Local Date and Time in America/New_York: " + formattedDateTime);
    }
}

```

N.B.: We parse a string representing a date and time in UTC format ("2024-05-31T10:15:30Z") using the `Instant.parse()` method, which returns an `Instant` object. We convert the parsed `Instant` to a `ZonedDateTime` object using a specific `ZoneId` ("America/New_York") with the `atZone()` method. We format the local date and time using `DateTimeFormatter` with the desired pattern ("yyyy-MM-dd HH:mm").

Q12. Write a program which calculate the simple interest of your current account with a rate of interest 8% per annum. Your program ask you to enter the start date, and total amount and print the total amount after adding interest amount. Use DATE and Time API.

```

import java.time.*;
import java.util.*;
public class SimpleInterestCalculator {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        // Get the start date
        System.out.print("Enter the start date (YYYY-MM-DD): ");
        String startDateStr = sc.next();
        LocalDate startDate = LocalDate.parse(startDateStr);

        // Get the total amount
        System.out.print("Enter the total amount: ");
        double totalAmount = sc.nextDouble();

        // Calculate the interest amount (simple interest formula: I = P * R * T)
        double rateOfInterest = 0.08;
        LocalDate currentDate = LocalDate.now();
        long days = startDate.until(currentDate).getDays(); // Calculate the number
of days between start date and current date
        double interestAmount = (totalAmount * rateOfInterest * days) / 365;

        // Calculate the total amount after adding interest
        double totalAmountWithInterest = totalAmount + interestAmount;

        // Print the total amount after adding interest
        System.out.println("Total amount after adding interest: " + totalAmountWithInterest);
        sc.close();
    }
}

```



Q13. Using the new Date and Time API, write Java code to:

- Get the current date and time
- Calculate the duration of 2 weeks from today
- Display the date in the format "MM/dd/yyyy"

```

import java.time.LocalDateTime;

```



```

import java.time.format.DateTimeFormatter;

public class DateTimeExample {
    public static void main(String[] args) {
        // Get the current date and time
        LocalDateTime currentDateTime = LocalDateTime.now();
        System.out.println("Current Date and Time: " + currentDateTime);

        // Calculate the duration of 2 weeks from today
        LocalDateTime twoWeeksFromNow = currentDateTime.plusWeeks(2);
        System.out.println("Two weeks from today: " + twoWeeksFromNow);

        // Display the date in the format "MM/dd/yyyy"
        DateTimeFormatter dateFormatter = DateTimeForma-
ter.ofPattern("MM/dd/yyyy");
        String formattedDate = currentDateTime.format(dateFormatter);
        System.out.println("Formatted Date: " + formattedDate);
    }
}

```

Q14. Calculate a person's age, given their date of birth and the current date, using LocalDate API.

```

import java.time.*;

public class AgeCalculator {

    public static void main(String[] args) {

        // Current Date
        LocalDate current = LocalDate.now();

        // Date of Birth
        LocalDate dob = LocalDate.of(1990, 05, 15);

        // Calculate the age by comparing year, month, and day components
        int age = current.getYear() - dob.getYear();

        // Check if the birthday hasn't occurred yet this year
        if (current.getMonthValue() < dob.getMonthValue() ||
            (current.getMonthValue() == dob.getMonthValue() &&
             current.getDayOfMonth() < dob.getDayOfMonth())) {
            age--;
        }

        // Print the age
        System.out.println("Age: " + age + " years");
    }
}

```