Java Assignment-5 Streams and Lambda

1.Write the following methods that return a lambda expression performing a specified action:

PerformOperation isOdd(): The lambda expression must return if a number is odd or if it is even.
PerformOperation isPrime(): The lambda expression must return if a number is prime or if it is composite.
PerformOperation isPalindrome(): The lambda expression must return if a number is a palindrome or if it is not.
Input Format

Input is handled for you by the locked stub code in your editor.

Sample Input
The first line contains an integer, (the number of test cases).

The subsequent lines each describe a test case in the form of space-separated integers:
The first integer specifies the condition to check for (for Odd/Even, for Prime, or for Palindrome).
The second integer denotes the number to be checked.

```
5
1 4
2 5
3 898
1 3
2 12
```

Sample Output
```
EVEN
PRIME
```

PALINDROME
ODD
COMPOSITE
Language

CODE:

```
import java.util.function.Predicate;
public class LambdaExpressions {
    public static Predicate<Integer> isOdd() {
        return number -> number % 2 != 0;
    }
    public static Predicate<Integer> isPrime() {
        return number -> {
            if (number <= 1) {
                return false;
            }
            for (int i = 2; i * i <= number; i++) {
                if (number % i == 0) {
                    return false;
                }
            }
            return true;
        };
    }
    public static Predicate<Integer> isPalindrome() {
        return number -> {
            String str = String.valueOf(number);
            String reversed = new StringBuilder(str).reverse().toString();
            return str.equals(reversed);
        };
    }
    public static void main(String[] args) {
        // Sample input
        int[] testCases = {1, 2, 3, 1, 2};
        int[] numbers = {4, 5, 898, 3, 12};
        for (int i = 0; i < testCases.length; i++) {
            int condition = testCases[i];
            int number = numbers[i];
            switch (condition) {
                case 1:
                    if (isOdd().test(number)) {
                        System.out.println("ODD");
                    } else {
                        System.out.println("EVEN");
                    }
                    break;
                case 2:
                    if (isPrime().test(number)) {
                        System.out.println("PRIME");
                    } else {
                        System.out.println("COMPOSITE");
                    }
                    break;
                case 3:
                    if (isPalindrome().test(number)) {
                        System.out.println("PALINDROME");
                    } else {
                        System.out.println("NOT PALINDROME");
                    }
                    break;
            }
        }
    }
}
```

OUTPUT:
```
EVEN
PRIME
PALINDROME
ODD
COMPOSITE
```

2.Write a Java program for implementing Runnable using Lambda expression

Code:
```
public class RunnableLambdaExample {
    public static void main(String[] args) {
        // Create a Runnable using a lambda expression
        Runnable myRunnable = () -> {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Count: " + i);
            }
        };

        // Create a thread and start it with the Runnable
        Thread thread = new Thread(myRunnable);
        thread.start();
    }
}
```

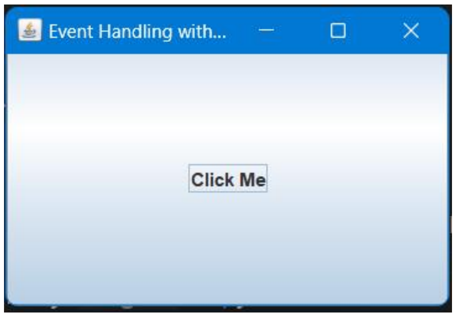
Output:
```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

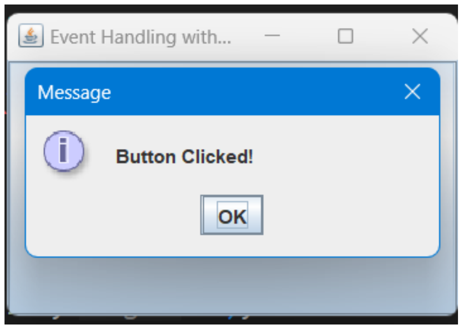
3.Write a Java program for event handling using Java 8 Lambda expressions

Code:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

public class EventHandlingWithLambda {
    public static void main(String[] args) {
        // Create a new JFrame
        JFrame frame = new JFrame("Event Handling with Lambda");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);

        // Create a button
        JButton button = new JButton("Click Me");

        // Add an ActionListener using a lambda expression
        button.addActionListener(e -> {
            // Define the action to be taken when the button is clicked
            JOptionPane.showMessageDialog(frame, "Button Clicked!");
        });

        // Add the button to the frame
        frame.getContentPane().add(button, BorderLayout.CENTER);

        // Display the frame
        frame.setVisible(true);
    }
}
```

Output:





Button Clicked!

4.Write a Java program for Iterating over List using Lambda expressions

Code:

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

public class ListIterationWithLambda {
    public static void main(String[] args) {
        // Create a list of integers
        List<Integer> numbers = new ArrayList<>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        numbers.add(4);
        numbers.add(5);

        // Iterate over the list using lambda expressions
        numbers.forEach(number -> System.out.println(number));
    }
}
```

Output:

```
1
2
3
4
5
```

5.Write a Java program to combine Predicate in Lambda Expressions

Code:

```
import java.util.function.Predicate;

public class CombinePredicatesWithLambda {
    public static void main(String[] args) {
        // Create a list of integers
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

        // Create Predicate objects using lambda expressions
        Predicate<Integer> isEven = number -> number % 2 == 0;
        Predicate<Integer> isGreaterThan5 = number -> number > 5;

        // Combine Predicates using lambda expressions
        Predicate<Integer> isEvenAndGreaterThan5 =
isEven.and(isGreaterThan5);
        Predicate<Integer> isEvenOrGreaterThan5 =
isEven.or(isGreaterThan5);
        Predicate<Integer> notEven = isEven.negate();

        // Apply the combined Predicates to the list of numbers
        System.out.println("Numbers that are even and greater than 5:");
        for (int number : numbers) {
            if (isEvenAndGreaterThan5.test(number)) {
                System.out.print(number + " ");
            }
        }
        System.out.println();

        System.out.println("Numbers that are either even or greater than 5:");
        for (int number : numbers) {
            if (isEvenOrGreaterThan5.test(number)) {
                System.out.print(number + " ");
            }
        }
        System.out.println();

        System.out.println("Numbers that are not even:");
        for (int number : numbers) {
            if (notEven.test(number)) {
                System.out.print(number + " ");
            }
        }
    }
}
```

Output:

```
Numbers that are even and greater than 5:
6 8 10
Numbers that are either even or greater than 5:
2 4 6 7 8 9 10
Numbers that are not even:
1 3 5 7 9
```

6.Write a Java program for creating a List of String by filtering

Code:

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

```
public class FilterListStrings {
    public static void main(String[] args) {
        // Create a list of strings
        List<String> stringList = new ArrayList<>();
        stringList.add("apple");
        stringList.add("banana");
        stringList.add("cherry");
        stringList.add("date");
        stringList.add("grape");
        stringList.add("fig");

        // Use Stream API to filter the list and create a new list
        List<String> filteredList = stringList.stream()
                .filter(s -> s.startsWith("b") || s.startsWith("c"))
                .collect(Collectors.toList());

        // Print the filtered list
        System.out.println("Filtered List: " + filteredList);
    }
}
```

Output:

```
Filtered List: [banana, cherry]
```

7.Write a Java program for creating a Sub List by Copying distinct values

Code:

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

public class CreateDistinctSublist {
    public static void main(String[] args) {
        // Create a list of integers with duplicates
        List<Integer> numbers = new ArrayList<>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(2);
        numbers.add(3);
        numbers.add(4);
        numbers.add(4);
        numbers.add(5);

        // Use Stream API to create a distinct sub-list
        List<Integer> distinctSublist = numbers.stream()
                .distinct()
                .collect(Collectors.toList());

        // Print the distinct sub-list
        System.out.println("Distinct Sub-list: " + distinctSublist);
    }
}
```

Output:

```
Distinct Sub-list: [1, 2, 3, 4, 5]
```