Java Programming 4-3: Recursion Practice

Activities

Vocabulary Definitions

1. **Non-linear Recursion**: The process of recursively calling two or more copies of the same method within a method. For example, in the Fibonacci sequence, you call the method for both `n-1` and `n-2`.
2. **Linear Recursion**: The process of calling one and only one copy of the same method within a method. For example, calculating factorial where you call the method with `n-1`.
3. **Recursion**: The process of solving a problem by solving smaller instances of the same problem until a base case is reached, then combining the results.
4. **Base Case**: The condition under which the recursion stops. For example, in the factorial calculation, the base case is when `d` is less than or equal to 1.
5. **Recursive Case**: The part of the method that includes a recursive call, solving a smaller instance of the problem. For example, `d * factorial(d - 1)`.

Try It/Solve It

1. Linear Recursion: Factorial Calculation

Here's how to implement a class for calculating factorial using linear recursion:

```
public class Linear {
    // Method to calculate factorial
    public static double factorial(double d) {
        // Base case
        if (d <= 1) {
            return 1;
        } else {
            // Recursive case
            return d * factorial(d - 1);
        }
    }

    // Main method to test factorial calculation
```

```
   public static void main(String[] args) {
      double d = 5.0;
      double result = factorial(d);
      System.out.println("Factorial [" + result + "] of [" + d + "]");
   }
}
```

Non-Linear Recursion: Fibonacci Sequence

```
public class NonLinear {
   // Method to calculate Fibonacci number
   public static double fibonacci(double d) {
      // Base case
      if (d < 2) {
         return d;
      } else {
         // Recursive case
         return fibonacci(d - 1) + fibonacci(d - 2);
      }
   }

   // Main method to test Fibonacci calculation
   public static void main(String[] args) {
      double d = 5.0;

      // Check if arguments are provided
      if (args.length > 0) {
         try {
            d = Double.parseDouble(args[0]);
         } catch (NumberFormatException e) {
            System.out.println("Invalid input, using default value 5.0");
            d = 5.0;
         }
      }

      // Print Fibonacci values up to the given index
      for (int i = 0; i <= d; i++) {
         double result = fibonacci(i);
         System.out.println("Fibonacci index [" + i + "] value [" + result + "]");
      }
   }
}
```

Factorial Trace Calculation for 7

```
public class FactorialTrace {
   // Method to calculate factorial
   public static double factorial(double d) {
      // Base case
      if (d <= 1) {
         return 1;
      } else {
         // Recursive case
         return d * factorial(d - 1);
      }
```

```
    }

    // Main method to compute and trace factorial of 7
    public static void main(String[] args) {
        double d = 7.0;
        double result = factorial(d);
        System.out.println("Factorial [" + result + "] of [" + d + "]");
    }
}
```

- **Linear Class**: Computes the factorial of a number using linear recursion.
- **NonLinear Class**: Computes Fibonacci numbers using non-linear recursion.
- **FactorialTrace Class**: Computes and prints the factorial of 7, demonstrating the recursive trace calculation.