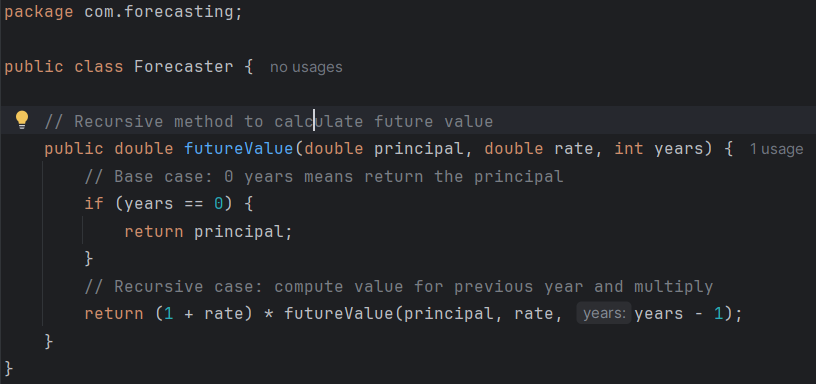
Recursion is a method where a function calls itself to solve a problem. Each recursive call solves a smaller part of the original problem until it reaches a **base case**, which stops the recursion.

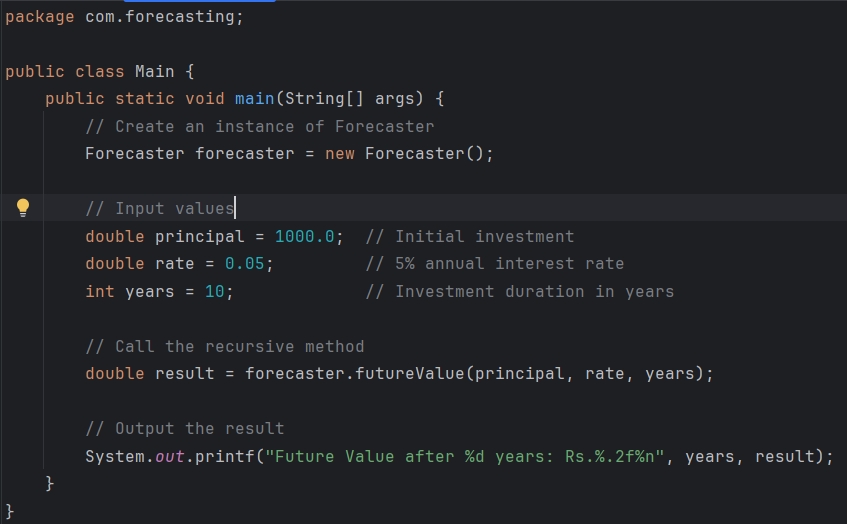
**Why use recursion?**

* **Simplifies problems** like tree traversal, factorials, Fibonacci numbers.
* Breaks a problem down into smaller, more manageable parts.
* Often aligns closely with the mathematical definition of the problem.

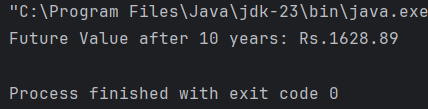
**Class - Forecaster.java**

****

**Class – Main.java**

****

**Output –**

****

# 1. Time Complexity of the Recursive Algorithm

The recursive method for calculating future value works by calling itself repeatedly, reducing the number of years by one in each call until it reaches zero. For each year, the method performs a single multiplication and one recursive call.

**Time Complexity**The time complexity of this approach is linear in relation to the number of years. If you are forecasting for "n" years, the method will execute approximately "n" calls before reaching the base case. Therefore, the time complexity is O(n), which means the processing time increases linearly as the number of years increases.

**Space Complexity**Each recursive call adds a new frame to the call stack. Thus, the space complexity is also linear (O(n)). If the number of years is large, this could lead to a stack overflow, especially in environments with limited stack memory.

# 2. How to Optimize the Recursive Solution

Recursive solutions are elegant and easy to understand, but they are not always the most efficient in practice. Here are two ways to optimize or replace the recursive approach:

**A. Use an Iterative Approach**

Instead of using recursion, you can use a simple loop to compute the future value. This avoids the overhead of recursive calls and eliminates the risk of stack overflow.

Benefits:

* Lower memory usage since it does not add to the call stack.
* Better suited for large input values.