**Exercise 2: Implementing a Product Search Function for an E-Commerce Platform**

**Scenario:**

You are working on an e-commerce application. Users should be able to search for products based on keywords. You will implement a simple search functionality that retrieves products by matching their name or description.

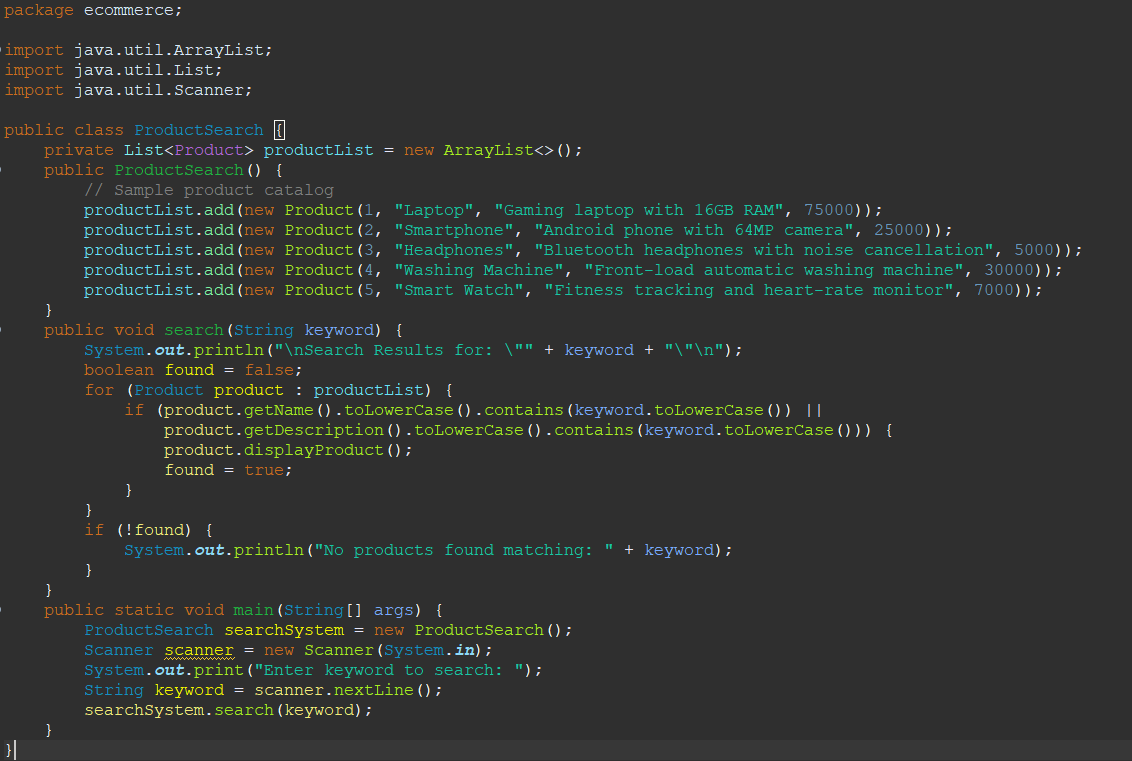
**Steps:**

1. Create a new Java project named EcommerceSearchFunction.
2. Define a Product class with attributes: id, name, description, and price.
3. Create a list of products to act as the product catalog.
4. Implement a search method that filters products based on the keyword.
5. Display search results in the console.

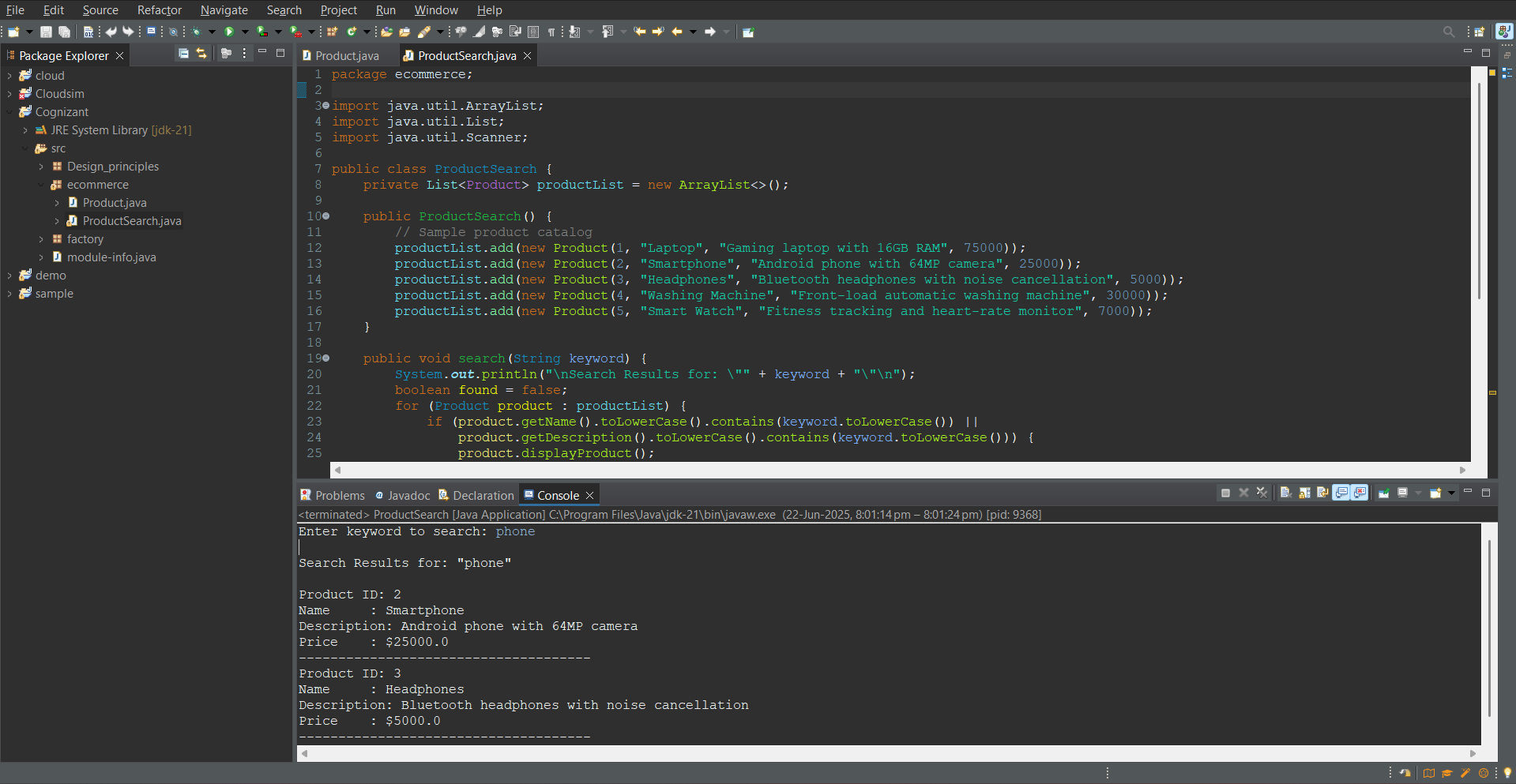
**Code:**

**Product Class**

**ProductSearch Class**

****

**Output Screenshot**



**Exercise 8: Financial Forecasting with Annual Contributions**

**Scenario:**

You are developing a forecasting tool for a user who invests an initial amount and also contributes a fixed sum annually. Each year, interest is applied to the **previous value + contribution**. This type of forecasting is common in retirement planning and long-term investment projections.

**Steps:**

**1. Understand the Modified Forecasting Formula**

This model uses **compound interest with contributions**, calculated recursively:

FV(n)=(FV(n−1)+annualContribution)×(1+r)FV(0)=initialInvestmentFV(n) = (FV(n - 1) + annualContribution) \times (1 + r) \\ FV(0) = initialInvestmentFV(n)=(FV(n−1)+annualContribution)×(1+r)FV(0)=initialInvestment

Where:

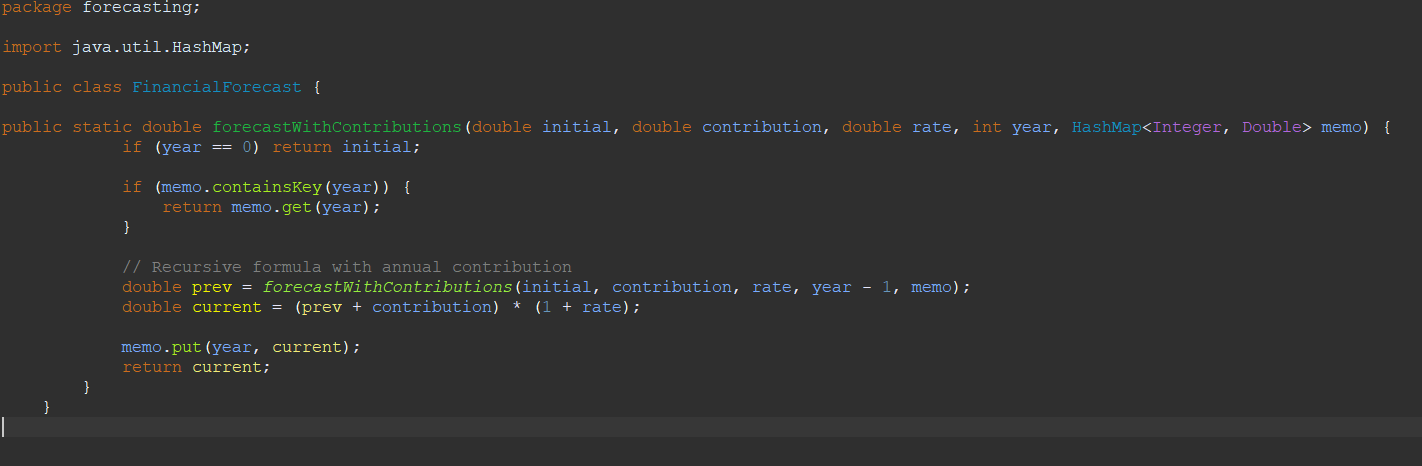
* FV(n) = Future value after n years
* r = annual growth rate (in decimal)
* annualContribution = fixed contribution per year

**2. Setup:**

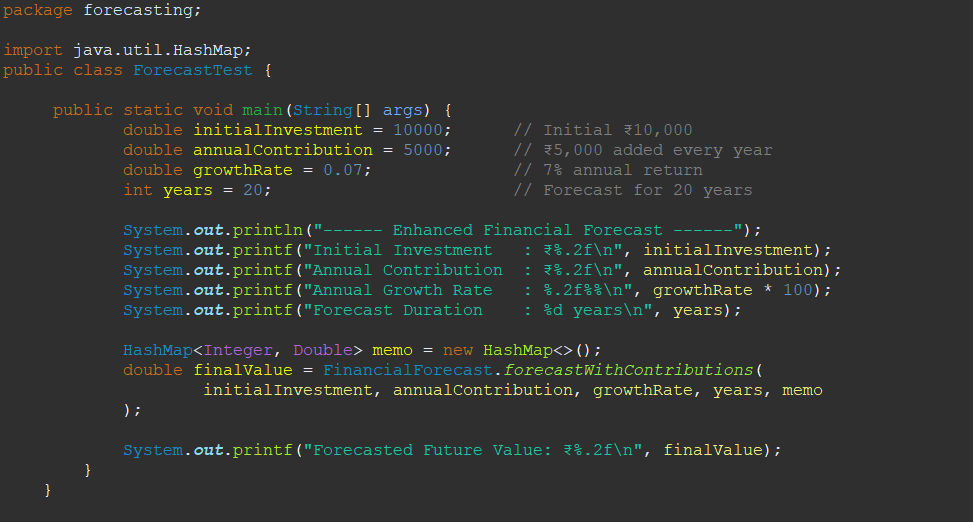
We will use **Java**, and **HashMap** for memoization to store previously computed results for efficiency.

**Code**

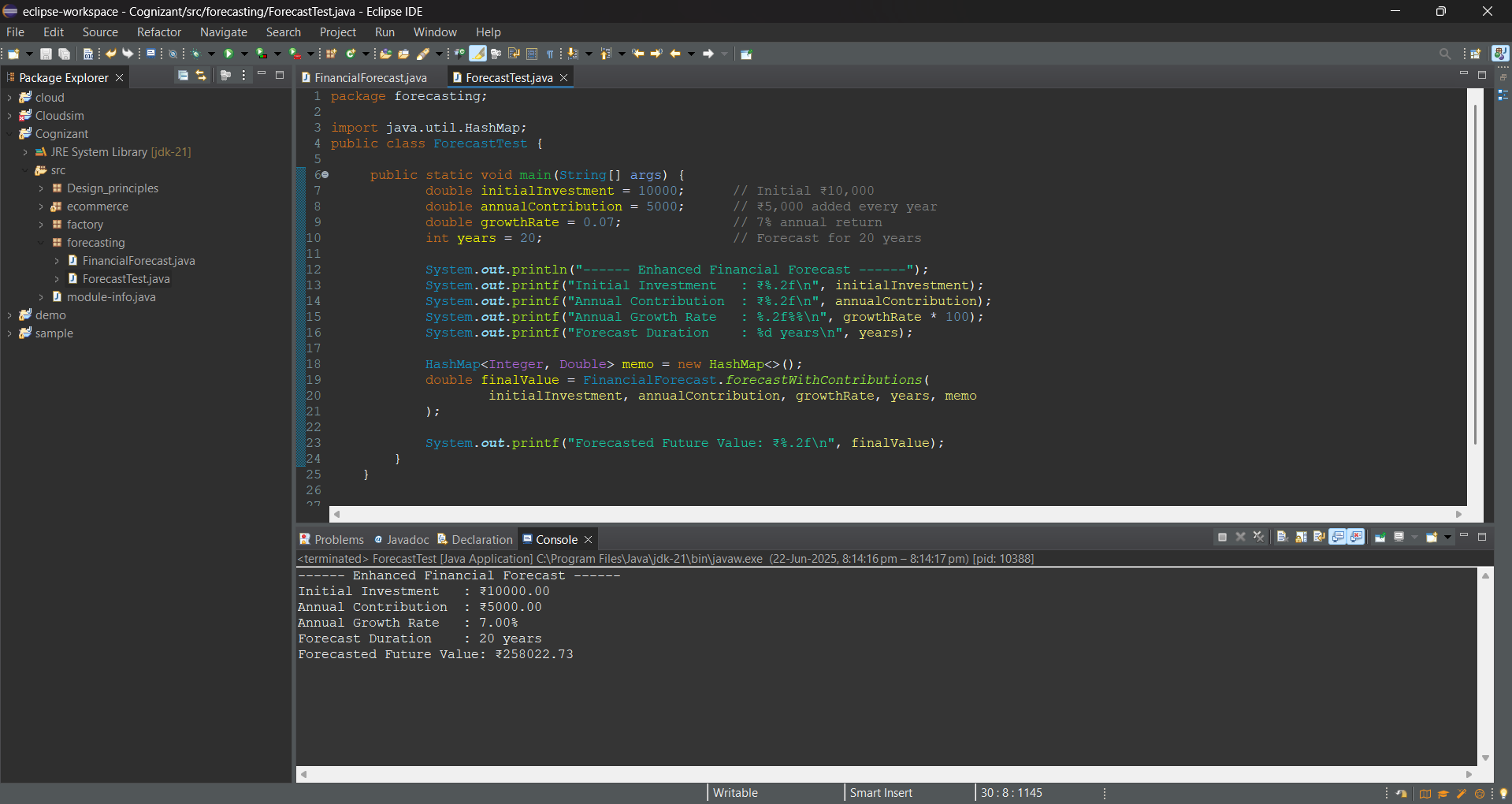
**FinancialForecast.java**

****

**ForecastTest.java**

****

**Sample Output:**

****

**Analysis:**

**1 .Time Complexity**

* Without memoization: Exponential (recomputes the same year multiple times).
* **With memoization:**
  + Each year’s value computed only once
  + get() and put() in HashMap are average **O(1)**
  + **Overall Time Complexity: O(n)**

**2 .Space Complexity:**

* **O(n)** for storing memoized values in the HashMap.

**3 . Optimization Summary:**

* Use memoization to **avoid redundant recursion**.
* Store intermediate values in a map using year as the key: