**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

**Code:**

class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger instance created.");

}

public static Logger getInstance() {

if (instance == null) {

synchronized (Logger.class) {

if (instance == null) {

instance = new Logger();

}

}

}

return instance;

}

public void log(String message) {

System.out.println("Log: " + message);

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("This is the first log message.");

Logger logger2 = Logger.getInstance();

logger2.log("This is the second log message.");

Logger logger3 = Logger.getInstance();

logger3.log("This is the third log message.");

System.out.println("\nVerifying instances:");

System.out.println("logger1 == logger2: " + (logger1 == logger2));

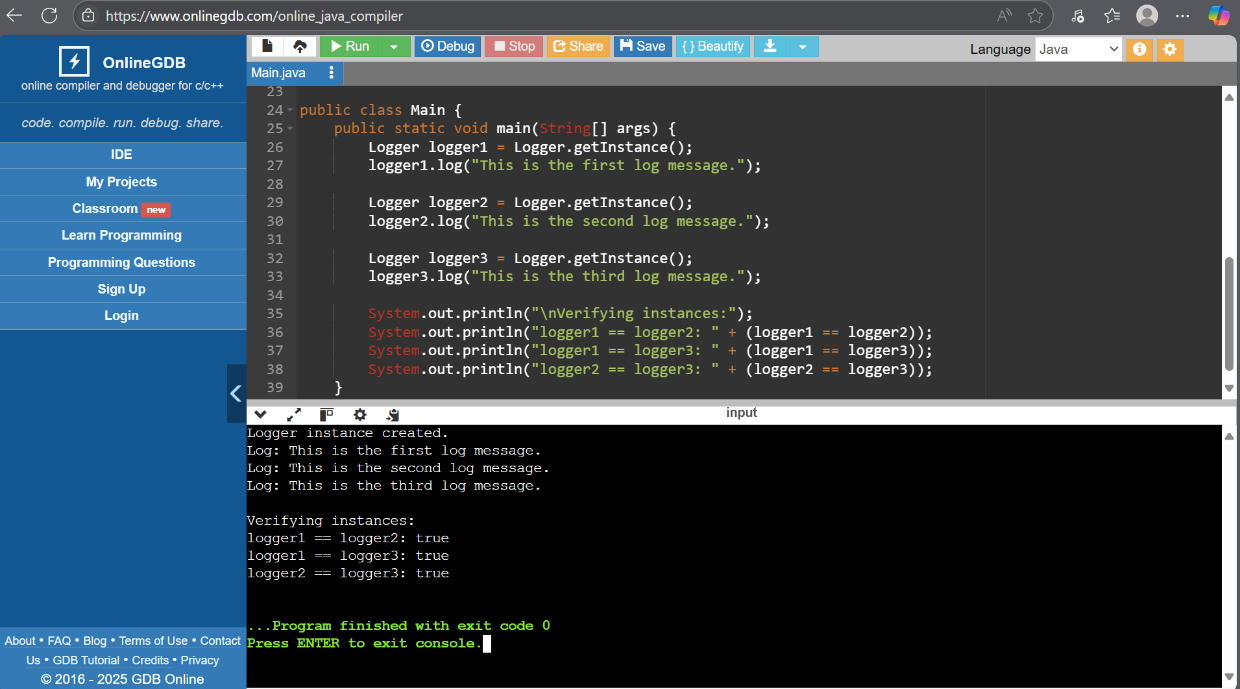
System.out.println("logger1 == logger3: " + (logger1 == logger3));

System.out.println("logger2 == logger3: " + (logger2 == logger3));

}

}

**Output:**



**Exercise 2: Implementing the Factory Method Pattern**

**Code:**

class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger instance created.");

}

public static Logger getInstance() {

if (instance == null) {

synchronized (Logger.class) {

if (instance == null) {

instance = new Logger();

}

}

}

return instance;

}

public void log(String message) {

System.out.println("Log: " + message);

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("This is the first log message.");

Logger logger2 = Logger.getInstance();

logger2.log("This is the second log message.");

Logger logger3 = Logger.getInstance();

logger3.log("This is the third log message.");

System.out.println("\nVerifying instances:");

System.out.println("logger1 == logger2: " + (logger1 == logger2));

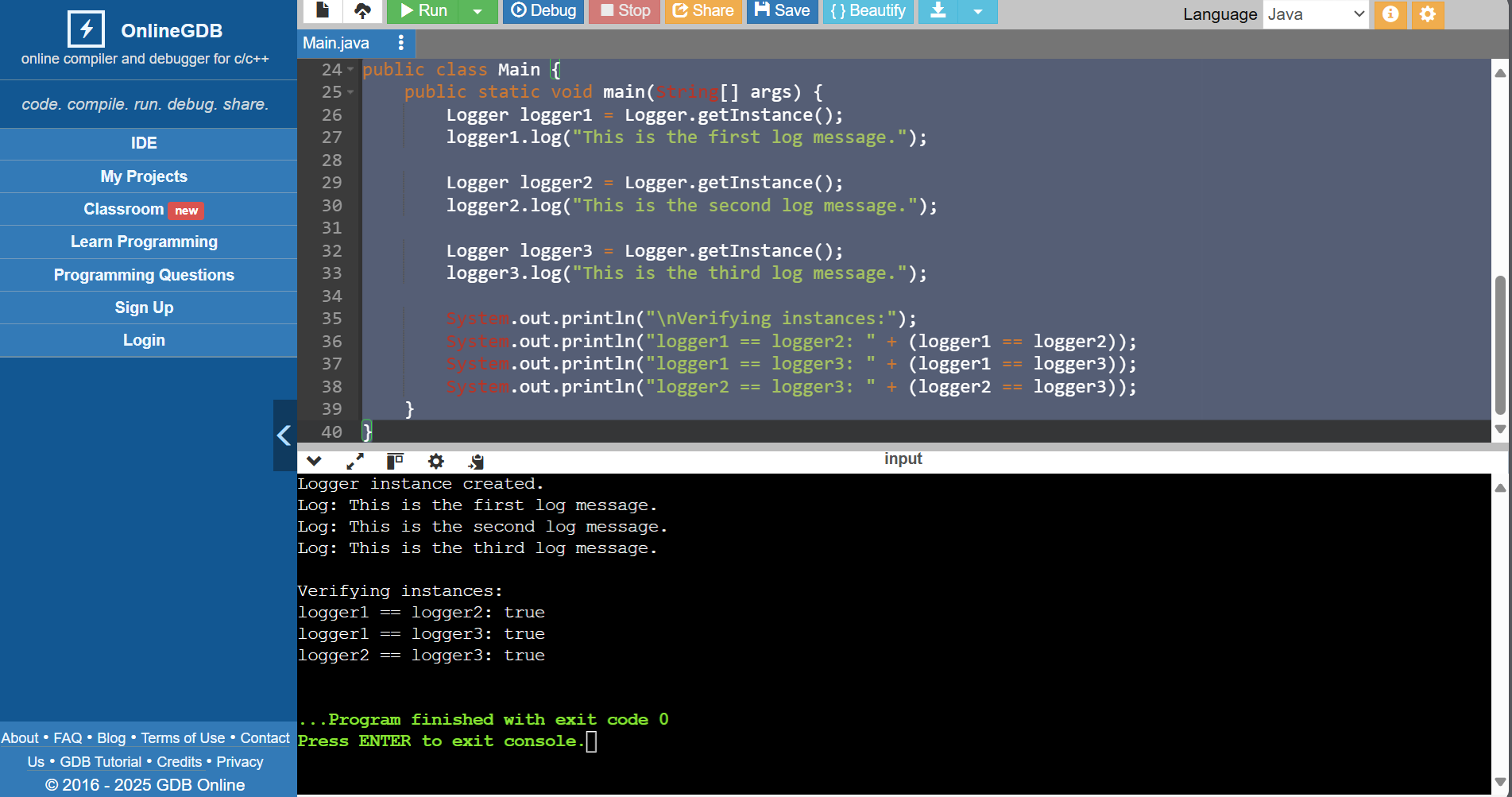
System.out.println("logger1 == logger3: " + (logger1 == logger3));

System.out.println("logger2 == logger3: " + (logger2 == logger3));

}

}

**Output:**



**Algorithms\_Data Structures**

**Exercise 2: E-commerce Platform Search Function**

**Code:**

import java.util.Arrays;

class Product implements Comparable<Product> {

int id;

String name;

public Product(int id, String name) {

this.id = id;

this.name = name;

}

@Override

public String toString() {

return "Product{id=" + id + ", name='" + name + "'}";

}

@Override

public int compareTo(Product other) {

return Integer.compare(this.id, other.id);

}

}

class ProductSearch {

public Product linearSearch(Product[] products, int idToFind) {

for (Product product : products) {

if (product.id == idToFind) {

return product;

}

}

return null;

}

public Product binarySearch(Product[] sortedProducts, int idToFind) {

int low = 0;

int high = sortedProducts.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

Product midProduct = sortedProducts[mid];

if (midProduct.id == idToFind) {

return midProduct;

} else if (midProduct.id < idToFind) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

}

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop"),

new Product(105, "Mouse"),

new Product(103, "Keyboard"),

new Product(201, "T-Shirt"),

new Product(202, "Jeans"),

new Product(301, "Coffee Maker"),

new Product(102, "Monitor")

};

ProductSearch searcher = new ProductSearch();

System.out.println("--- Linear Search Test ---");

int searchId1 = 103;

Product foundProduct1 = searcher.linearSearch(products, searchId1);

if (foundProduct1 != null) {

System.out.println("Product found (Linear): " + foundProduct1);

} else {

System.out.println("Product with ID " + searchId1 + " not found (Linear).");

}

int searchId2 = 999;

Product foundProduct2 = searcher.linearSearch(products, searchId2);

if (foundProduct2 != null) {

System.out.println("Product found (Linear): " + foundProduct2);

} else {

System.out.println("Product with ID " + searchId2 + " not found (Linear).");

}

System.out.println("\n--- Binary Search Test (requires sorted array) ---");

Product[] sortedProducts = Arrays.copyOf(products, products.length);

Arrays.sort(sortedProducts);

System.out.println("Sorted Products:");

for(Product p : sortedProducts) {

System.out.println(p);

}

int searchId3 = 201;

Product foundProduct3 = searcher.binarySearch(sortedProducts, searchId3);

if (foundProduct3 != null) {

System.out.println("\nProduct found (Binary): " + foundProduct3);

} else {

System.out.println("\nProduct with ID " + searchId3 + " not found (Binary).");

}

int searchId4 = 101;

Product foundProduct4 = searcher.binarySearch(sortedProducts, searchId4);

if (foundProduct4 != null) {

System.out.println("Product found (Binary): " + foundProduct4);

} else {

System.out.println("Product with ID " + searchId4 + " not found (Binary).");

}

int searchId5 = 500;

Product foundProduct5 = searcher.binarySearch(sortedProducts, searchId5);

if (foundProduct5 != null) {

System.out.println("Product found (Binary): " + foundProduct5);

} else {

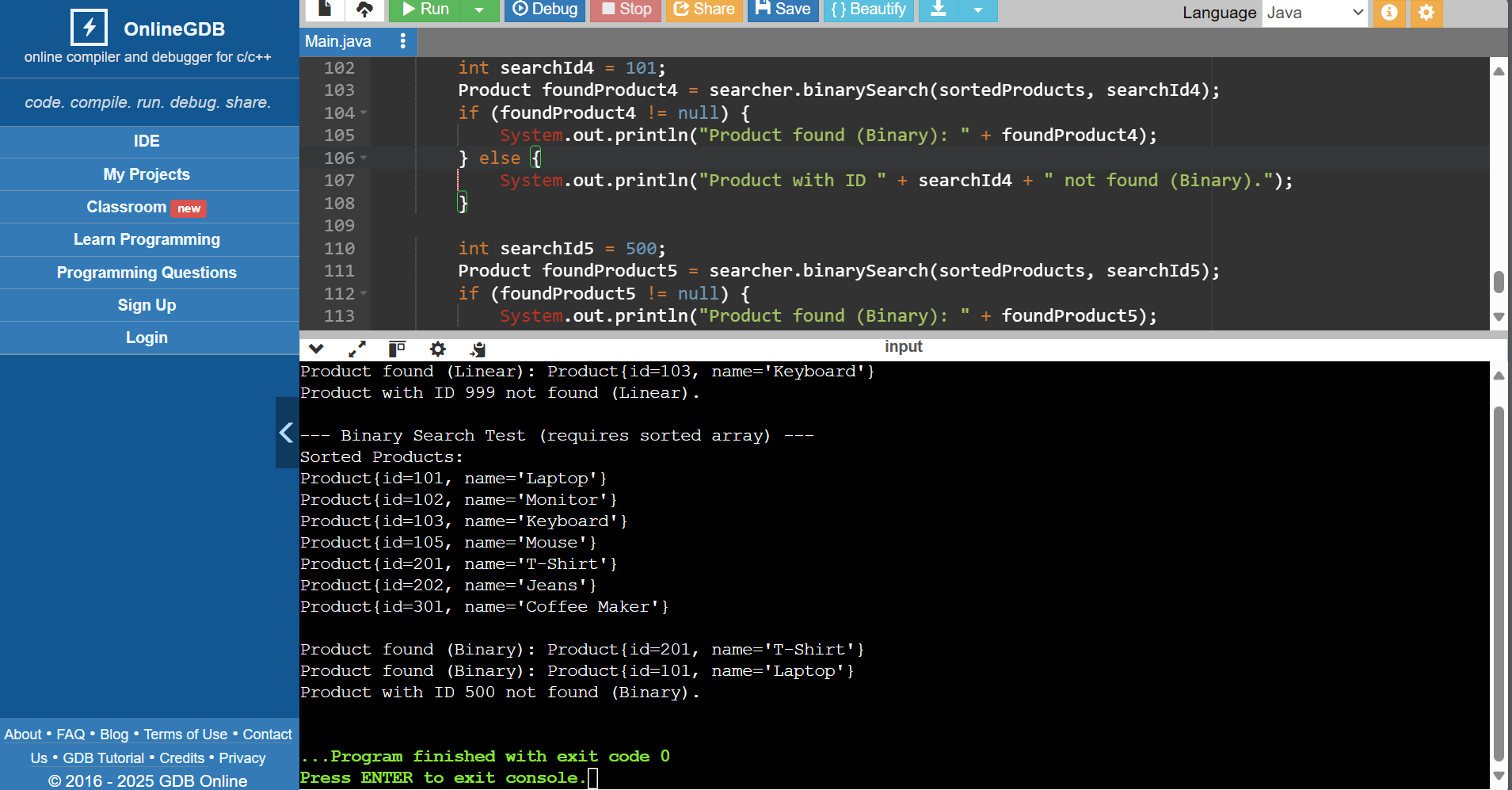
System.out.println("Product with ID " + searchId5 + " not found (Binary).");

}

}

}

**Output :**

****

**Exercise 7: Financial Forecasting**

**Code:**

class FinancialForecaster {

public double predictFutureValueRecursive(double currentValue, double growthRate, int remainingPeriods) {

if (remainingPeriods == 0) {

return currentValue;

}

double valueAfterOnePeriod = currentValue \* (1 + growthRate);

return predictFutureValueRecursive(valueAfterOnePeriod, growthRate, remainingPeriods - 1);

}

public double predictFutureValueIterative(double initialValue, double growthRate, int numPeriods) {

double futureValue = initialValue;

for (int i = 0; i < numPeriods; i++) {

futureValue \*= (1 + growthRate);

}

return futureValue;

}

}

public class Main {

public static void main(String[] args) {

FinancialForecaster forecaster = new FinancialForecaster();

double initialInvestment = 1000.0;

double annualGrowthRate = 0.07;

int forecastYears = 10;

System.out.println("--- Financial Forecasting ---");

System.out.printf("Initial Investment: Rs%.2f%n", initialInvestment);

System.out.println("Annual Growth Rate: " + (annualGrowthRate \* 100) + "%");

System.out.println("Forecast Years: " + forecastYears);

double futureValueRecursive = forecaster.predictFutureValueRecursive(initialInvestment, annualGrowthRate, forecastYears);

System.out.printf("\nFuture Value (Recursive): Rs%.2f%n", futureValueRecursive);

double futureValueIterative = forecaster.predictFutureValueIterative(initialInvestment, annualGrowthRate, forecastYears);

System.out.printf("Future Value (Iterative): Rs%.2f%n", futureValueIterative);

double initial = 500;

double rate = 0.10;

int periods = 5;

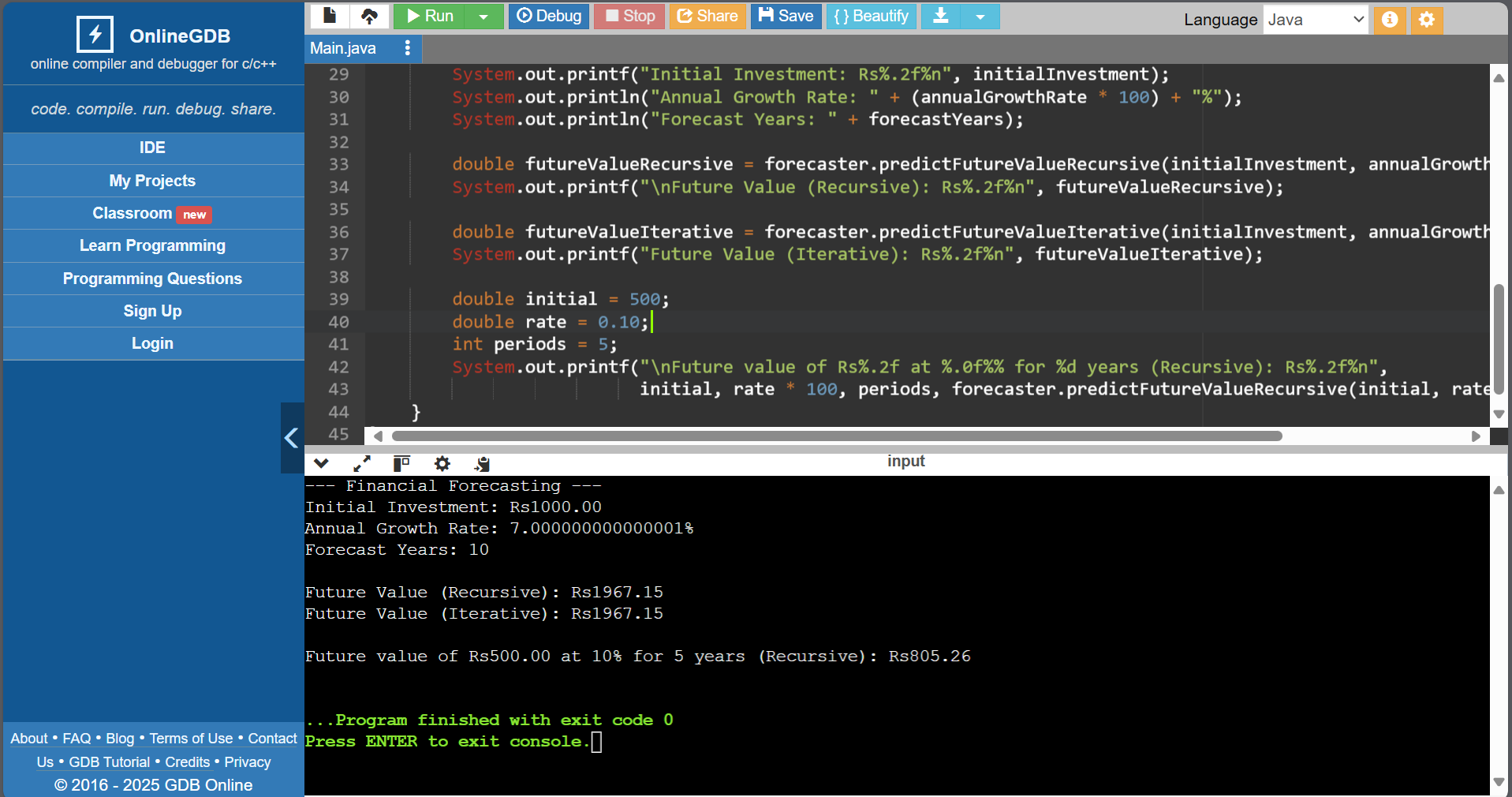
System.out.printf("\nFuture value of Rs%.2f at %.0f%% for %d years (Recursive): Rs%.2f%n",

initial, rate \* 100, periods, forecaster.predictFutureValueRecursive(initial, rate, periods));

}

}

**Output :**

****