**Design principles & Patterns**

**Exercise 1: Implementing the Singleton Pattern**

**Code:**

**Singleton class:**

package Design\_Patterns;

public class Singleton {

private static Singleton *instance*;

private Singleton() {

System.*out*.println("Singleton instance created");

}

public static synchronized Singleton getInstance() {

if (*instance* == null) {

*instance* = new Singleton();

}

return *instance*;

}

public void showMessage() {

System.*out*.println("Singleton Class");

}

}

**Main Method:**

package Design\_Patterns;

public class Main {

public static void main(String[] args) {

Singleton s1 = Singleton.*getInstance*();

s1.showMessage();

Singleton s2 = Singleton.*getInstance*();

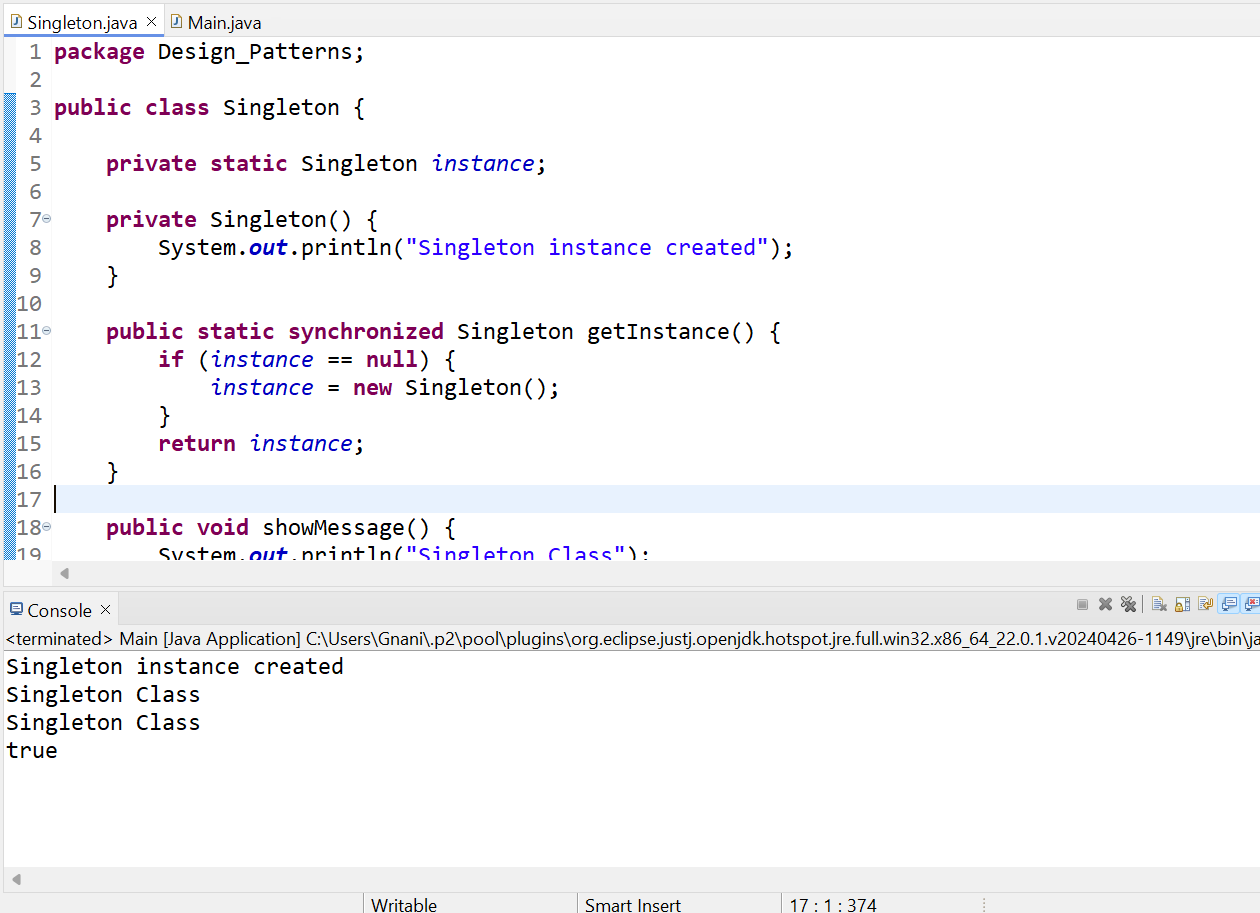
s2.showMessage();

System.*out*.println(s1 == s2);

}

}

**Output:**

****

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

**Notification.java**

package com.Factory;

public interface Notification {

void notifyUser();

}

**EmailNotification.java**

package com.Factory;

public class EmailNotification implements Notification {

public void notifyUser() {

System.*out*.println("Sending an Email Notification");

}

}

**SMSNotification.java**

package com.Factory;

public class SMSNotification implements Notification {

public void notifyUser() {

System.*out*.println("Sending an SMS Notification");

}

}

**PushNotification.java**

package com.Factory;

public class PushNotification implements Notification {

public void notifyUser() {

System.*out*.println("Sending a Push Notification");

}

}

**NotificationFactory.java**

package com.Factory;

public class NotificationFactory {

public Notification createNotification(String type) {

if (type == null || type.isEmpty())

return null;

switch (type.toLowerCase()) {

case "sms":

return new SMSNotification();

case "email":

return new EmailNotification();

case "push":

return new PushNotification();

default:

return null;

}

}

}

**Main.java**

package com.Factory;

public class Main {

public static void main(String[] args) {

NotificationFactory factory = new NotificationFactory();

Notification n1 = factory.createNotification("email");

n1.notifyUser();

Notification n2 = factory.createNotification("sms");

n2.notifyUser();

Notification n3 = factory.createNotification("push");

n3.notifyUser();

}

}

**Output:**

****

**Data structures and Algorithms**

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

**Code :**

**Product.java**

package com.Search;

public class Product implements Comparable<Product> {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

@Override

public int compareTo(Product other) {

return Integer.*compare*(this.productId, other.productId);

}

}

**ECommerceSearch.java**

package com.Search;

import java.util.\*;

public class ECommerceSearch {

public static Product linearSearch(Product[] products, String keyword) {

for (Product p : products) {

if (p.productName.toLowerCase().contains(keyword.toLowerCase())) {

return p;

}

}

return null;

}

public static Product binarySearch(Product[] products, int productId) {

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (products[mid].productId == productId)

return products[mid];

else if (products[mid].productId < productId)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(103, "Dell Laptop", "Electronics"),

new Product(101, "Nike Shoes", "Footwear"),

new Product(104, "iPhone", "Mobiles"),

new Product(102, "Boat Headphones", "Accessories")

};

Scanner sc = new Scanner(System.*in*);

System.*out*.print("Enter product name keyword to search (linear): ");

String keyword = sc.nextLine();

Product result1 = *linearSearch*(products, keyword);

if (result1 != null)

System.*out*.println("Linear Search Found: " + result1);

else

System.*out*.println("Linear Search: Product not found.");

Arrays.*sort*(products);

System.*out*.print("Enter product ID to search (binary): ");

int id = sc.nextInt();

Product result2 = *binarySearch*(products, id);

if (result2 != null)

System.*out*.println("Binary Search Found: " + result2);

else

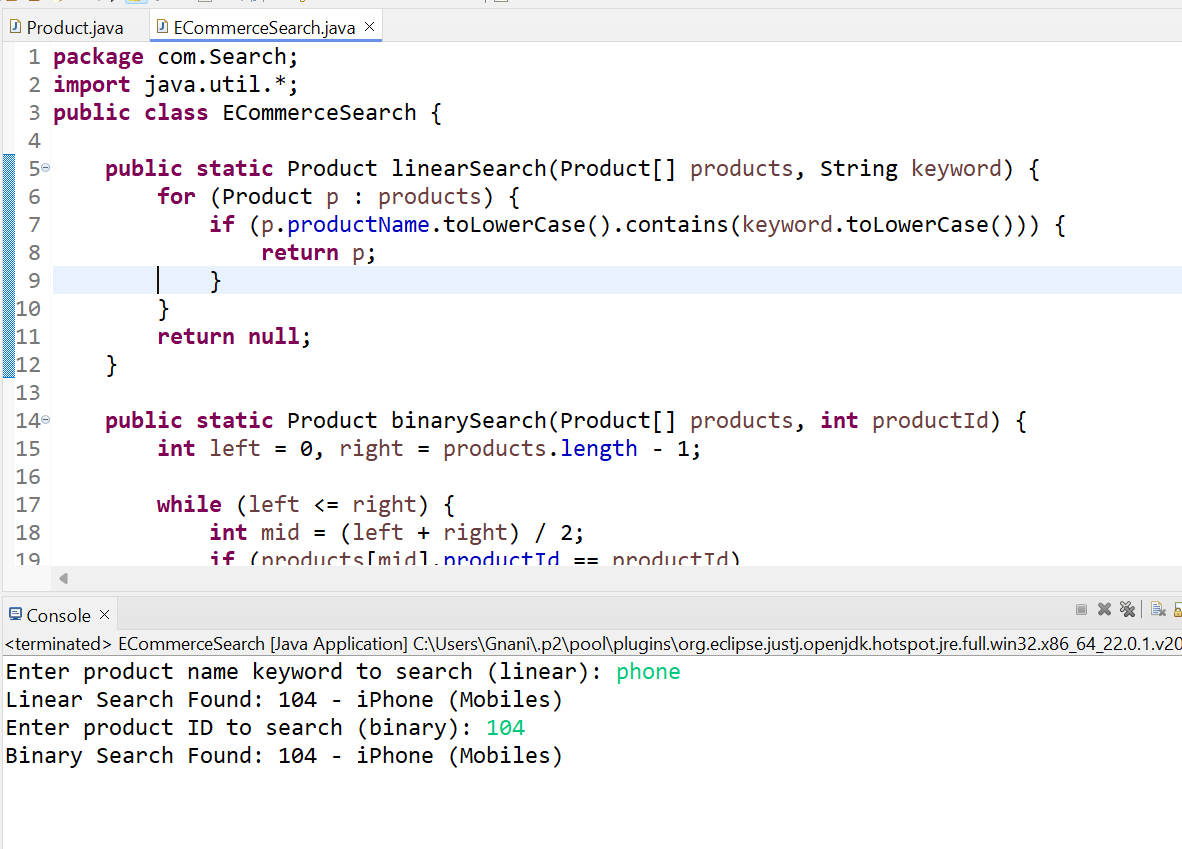
System.*out*.println("Binary Search: Product not found.");

sc.close();

}

}

**Output:**

****

**Exercise 7: Financial Forecasting**

**Code:**

package com.Financial;

import java.util.\*;

public class FinancialForecast {

public static double forecastRevenue(int months, double currentRevenue, double growthRate) {

if (months == 0) return currentRevenue;

return *forecastRevenue*(months - 1, currentRevenue \* (1 + growthRate), growthRate);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.*in*);

System.*out*.print("Enter current revenue (in crores): ");

double revenue = sc.nextDouble();

System.*out*.print("Enter monthly growth rate (%): ");

double rate = sc.nextDouble();

System.*out*.print("Enter number of months to forecast: ");

int n = sc.nextInt();

double growthRate = rate / 100;

double futureRevenue = *forecastRevenue*(n, revenue, growthRate);

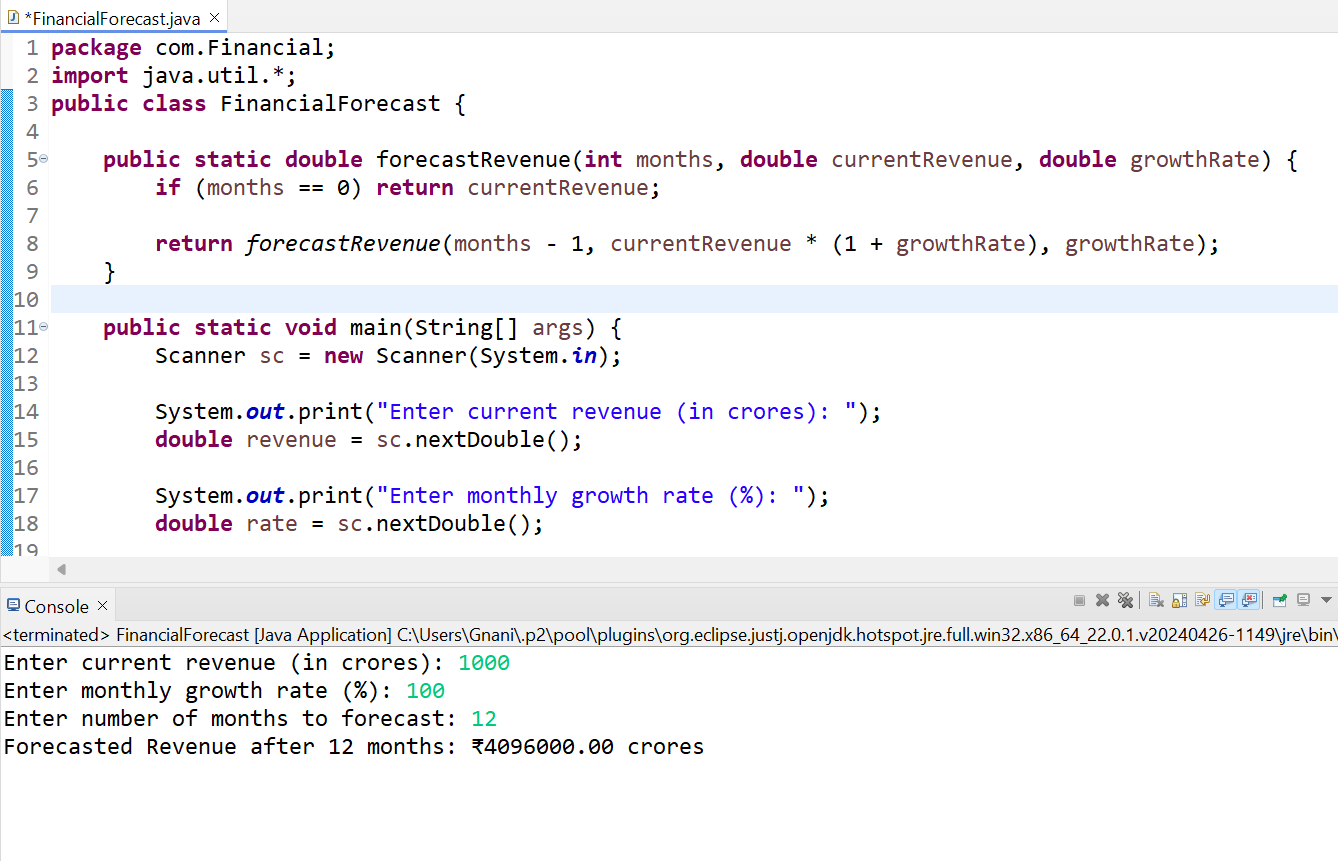
System.*out*.printf("Forecasted Revenue after %d months: ₹%.2f crores\n", n, futureRevenue);

sc.close();

}

}

**Output:**

****