|  |  |  |  |
| --- | --- | --- | --- |
| Design principles & Patterns | 1 | Design Patterns and Principles | Exercise 1: Implementing the Singleton Pattern |

public class Logger {

private static Logger instance;

private Logger() {

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

}

public static synchronized Logger getInstance() {

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("Starting application...");

Logger logger2 = Logger.getInstance();

logger2.log("Continuing execution...");

if (logger1 == logger2) {

System.out.println("Both logger1 and logger2 refer to the same instance.");

} else {

System.out.println("Different Logger instances exist!");

}

}

}

Output:

Logger instance created.

[LOG]: Starting application...

[LOG]: Continuing execution...

Both logger1 and logger2 refer to the same instance.

|  |  |  |  |
| --- | --- | --- | --- |
| Design principles & Patterns | 2 | Design Patterns and Principles | Exercise 2: Implementing the Factory Method Pattern |

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

System.out.println("Opening Word Document.");

}

}

class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF Document.");

}

}

class ExcelDocument implements Document {

public void open() {

System.out.println("Opening Excel Document.");

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

Output:

Opening Word Document.

Opening PDF Document.

Opening Excel Document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Data structures and Algorithms | 1 | Algorithms\_Data Structures | Exercise 2: E-commerce Platform Search Function |

import java.util.HashMap;

import java.util.Scanner;

class Product {

String productId;

String productName;

int quantity;

double price;

Product(String productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

void display() {

System.out.println(productId + " - " + productName + " | Qty: " + quantity + " | Price: $" + price);

}

}

class InventoryManager {

private HashMap<String, Product> inventory = new HashMap<>();

void addProduct(Product product) {

inventory.put(product.productId, product);

System.out.println("Product added.");

}

void updateProduct(String productId, int quantity, double price) {

Product product = inventory.get(productId);

if (product != null) {

product.quantity = quantity;

product.price = price;

System.out.println("Product updated.");

} else {

System.out.println("Product not found.");

}

}

void deleteProduct(String productId) {

if (inventory.remove(productId) != null) {

System.out.println("Product deleted.");

} else {

System.out.println("Product not found.");

}

}

void displayAllProducts() {

if (inventory.isEmpty()) {

System.out.println("Inventory is empty.");

} else {

for (Product p : inventory.values()) {

p.display();

}

}

}

}

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Scanner scanner = new Scanner(System.in);

boolean running = true;

while (running) {

System.out.println("\n1. Add 2. Update 3. Delete 4. View 5. Exit");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

System.out.print("ID: ");

String id = scanner.nextLine();

System.out.print("Name: ");

String name = scanner.nextLine();

System.out.print("Quantity: ");

int qty = scanner.nextInt();

System.out.print("Price: ");

double price = scanner.nextDouble();

scanner.nextLine();

manager.addProduct(new Product(id, name, qty, price));

break;

case 2:

System.out.print("Enter Product ID to update: ");

String uid = scanner.nextLine();

System.out.print("New Quantity: ");

int uq = scanner.nextInt();

System.out.print("New Price: ");

double up = scanner.nextDouble();

scanner.nextLine();

manager.updateProduct(uid, uq, up);

break;

case 3:

System.out.print("Enter Product ID to delete: ");

String did = scanner.nextLine();

manager.deleteProduct(did);

break;

case 4:

manager.displayAllProducts();

break;

case 5:

running = false;

break;

default:

System.out.println("Invalid option.");

}

}

scanner.close();

}

}

Output:Sample RUN1. Add 2. Update 3. Delete 4. View 5. Exit

1

ID: P101

Name: Mouse

Quantity: 50

Price: 19.99

Product added.

1. Add 2. Update 3. Delete 4. View 5. Exit

4

P101 - Mouse | Qty: 50 | Price: $19.99

1. Add 2. Update 3. Delete 4. View 5. Exit

2

Enter Product ID to update: P101

New Quantity: 100

New Price: 17.99

Product updated.

1. Add 2. Update 3. Delete 4. View 5. Exit

4

P101 - Mouse | Qty: 100 | Price: $17.99

1. Add 2. Update 3. Delete 4. View 5. Exit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Data structures and Algorithms | 2 | Algorithms\_Data Structures | Exercise 7: Financial Forecasting |

5

public class FinancialForecast {

// Recursive method

public static double forecastRecursive(double principal, double rate, int years) {

if (years == 0) {

return principal;

}

return forecastRecursive(principal \* (1 + rate), rate, years - 1);

}

// Tail-recursive method

public static double forecastTailRecursive(double principal, double rate, int years) {

return helper(principal, rate, years);

}

private static double helper(double amount, double rate, int yearsLeft) {

if (yearsLeft == 0) return amount;

return helper(amount \* (1 + rate), rate, yearsLeft - 1);

}

// Iterative method

public static double forecastIterative(double principal, double rate, int years) {

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

principal \*= (1 + rate);

}

return principal;

}

public static void main(String[] args) {

double principal = 1000.0;

double rate = 0.05;

int years = 5;

double resultRecursive = forecastRecursive(principal, rate, years);

double resultTail = forecastTailRecursive(principal, rate, years);

double resultIterative = forecastIterative(principal, rate, years);

System.out.printf("Future Value (Recursive): %.2f%n", resultRecursive);

System.out.printf("Future Value (Tail-Recursive): %.2f%n", resultTail);

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

}

}

Output:

Future Value (Recursive): 1276.28

Future Value (Tail-Recursive): 1276.28

Future Value (Iterative): 1276.28