1. **Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

Logger.java

**public** **class** Logger {

**private** **static** Logger *a*;

**protected** Logger(){

System.***out***.println("Singleton design pattern has been created.");

}

**public** **static** Logger get\_instance() {

**if** (*a*==**null**)

*a*=**new** Logger();

**return** *a*;

}

**public** **void** print\_msg(String msg) {

System.***out***.println("LOG :"+msg);

}

}

Singleton\_Design.java

**public** **class** Singleton\_Design {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Logger log1=Logger.*get\_instance*();

Logger log2=Logger.*get\_instance*();

log1.print\_msg("Log 1 loaded");

log2.print\_msg("Log 2 loaded");

**if**(log1==log2)

System.***out***.println("Same reference");

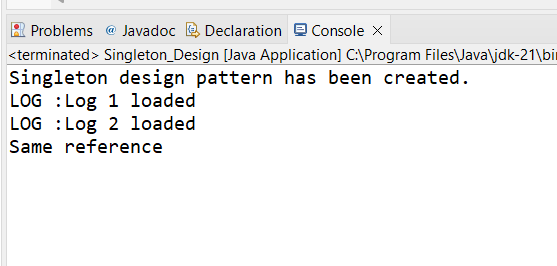
**else**

System.***out***.println("Not a Same reference");

}

}

**Output:**

****

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

Document.java

package week1;

public interface Document {

void open();

}

Concrete\_Classes.java

package week1;

class Concrete\_Classes\_1 implements Document{

@Override

public void open() {

System.*out*.println("Word Document created");

}

}

class Concrete\_Classes\_2 implements Document{

@Override

public void open() {

System.*out*.println("PDF Document created");

}

}

class Concrete\_Classes\_3 implements Document{

@Override

public void open() {

System.*out*.println("Excel Document created");

}

}

FactoryMethod.java

package week1;

public abstract class FactoryMethod {

abstract Document createDocument();

}

class WordDocumentFactory extends FactoryMethod{

@Override

Document createDocument() {

// TODO Auto-generated method stub

return new Concrete\_Classes\_1();

}

}

class PDFDocumentFactory extends FactoryMethod{

@Override

Document createDocument() {

// TODO Auto-generated method stub

return new Concrete\_Classes\_2();

}

}

class ExcelDocumentFactory extends FactoryMethod{

@Override

Document createDocument() {

// TODO Auto-generated method stub

return new Concrete\_Classes\_3();

}

}

FactoryMethodPatternExample.java

**public** **class** FactoryMethodPatternExample {

**public** **static** **void** main(String[] args) {

FactoryMethod a=**new** WordDocumentFactory();

Document word=a.createDocument();

word.open();

a=**new** PDFDocumentFactory();

Document pdf=a.createDocument();

pdf.open();

a=**new** ExcelDocumentFactory();

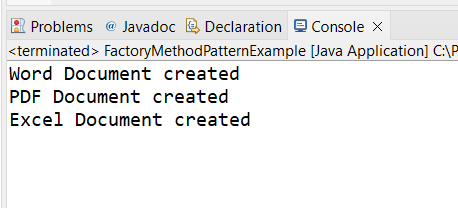
Document exl=a.createDocument();

exl.open();

}

}

**Output:**

****

**2) Data Structures and Algorithms**

**Exercise 1: Inventory Management System**

Product.java

**package** week1\_DSA;

**public** **class** Product {

**private** **int** p\_id;

**private** String p\_name;

**private** **int** quantity;

**private** **double** price;

**public** Product(**int** p\_id, String p\_name, **int** quantity, **double** price) {

**this**.p\_id = p\_id;

**this**.p\_name = p\_name;

**this**.quantity = quantity;

**this**.price = price;

}

**public** **int** getProductId() { **return** p\_id; }

**public** String getProductName() { **return** p\_name; }

**public** **int** getQuantity() { **return** quantity; }

**public** **double** getPrice() { **return** price; }

**public** **void** setQuantity(**int** quantity) { **this**.quantity = quantity; }

**public** **void** setPrice(**double** price) { **this**.price = price; }

@Override

**public** String toString() {

**return** "Product{" +

"ID=" + p\_id +

", Name='" + p\_name + '\'' +

", Qty=" + quantity +

", Price=" + price +

'}';

}

}

InventoryManager.java

package week1\_DSA;

import java.util.HashMap;

public class InventoryManager {

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

public void addProduct(Product product) {

inventory.put(product.getProductId(), product);

}

public void updateProduct(int productId, int newQuantity, double newPrice) {

if (inventory.containsKey(productId)) {

Product p = inventory.get(productId);

p.setQuantity(newQuantity);

p.setPrice(newPrice);

} else {

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

}

}

public void deleteProduct(int productId) {

inventory.remove(productId);

}

public void displayInventory() {

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

System.*out*.println(p);

}

}

}

Main.java

**package** week1\_DSA;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

InventoryManager manager = **new** InventoryManager();

// Add products

manager.addProduct(**new** Product(101, "Laptop", 10, 75000.0));

manager.addProduct(**new** Product(102, "Mouse", 50, 500.0));

// Update

manager.updateProduct(101, 8, 74000.0);

// Display

manager.displayInventory();

// Delete

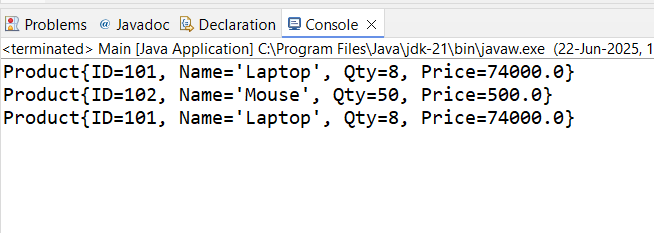
manager.deleteProduct(102);

manager.displayInventory();

}

}

**Output**

****

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

Search.java

**package** week1\_DSA;

**public** **class** Search{

**public** **static** **int** linearSearch(Product[] products, **int** targetId) {

**for** (**int** i = 0; i < products.length; i++) {

**if** (products[i].getProductId() == targetId) {

**return** i;

}

}

**return** -1; // Not found

}

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

**int** low = 0, high = products.length - 1;

**while** (low <= high) {

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

**if** (products[mid].getProductId() == targetId)

**return** mid;

**else** **if** (products[mid].getProductId() < targetId)

low = mid + 1;

**else**

high = mid - 1;

}

**return** -1;

}

}

Main.java

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** idToSearch = 103;

**int** indexLinear = Search.*linearSearch*(products, idToSearch);

**int** indexBinary = Search.*binarySearch*(products, idToSearch);

System.***out***.println("Linear Search: " + (indexLinear != -1 ? products[indexLinear] : "Not Found"));

System.***out***.println("Binary Search: " + (indexBinary != -1 ? products[indexBinary] : "Not Found"));

idToSearch = 104;

indexLinear = Search.*linearSearch*(products, idToSearch);

indexBinary = Search.*binarySearch*(products, idToSearch);

System.***out***.println("Linear Search: " + (indexLinear != -1 ? products[indexLinear] : "Not Found"));

System.***out***.println("Binary Search: " + (indexBinary != -1 ? products[indexBinary] : "Not Found"));

}

}

**Output**

