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

**Scenario:**

**You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.**

Product.java

Package algo;

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

@Override

public String toString() {

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

}

}

Main.java

Package algo

import java.util.Arrays;

import java.util.Comparator;

public class Main {

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

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

if (products[i].productId == targetId) {

return i;

}

}

return -1;

}

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

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

while (left <= right) {

int mid = left + (right - left) / 2;

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

return mid;

} else if (products[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

new Product(105, "Laptop", "Electronics"),

new Product(101, "Shirt", "Apparel"),

new Product(110, "Book", "Education"),

new Product(102, "Shoes", "Apparel"),

new Product(108, "Smartphone", "Electronics"),

new Product(103, "Blender", "Home"),

new Product(107, "Table", "Furniture"),

new Product(104, "Pen", "Stationery"),

new Product(109, "Headphones", "Electronics"),

new Product(106, "Chair", "Furniture")

};

int searchId = 108;

long startLinear = System.nanoTime();

int linearIndex = linearSearch(products, searchId);

long endLinear = System.nanoTime();

System.out.println("Linear Search:");

if (linearIndex != -1) {

System.out.println("Product found: " + products[linearIndex]);

} else {

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

}

System.out.println("Time taken: " + (endLinear - startLinear) + " ns\n");

Arrays.sort(products, Comparator.comparingInt(p -> p.productId));

long startBinary = System.nanoTime();

int binaryIndex = binarySearch(products, searchId);

long endBinary = System.nanoTime();

System.out.println("Binary Search:");

if (binaryIndex != -1) {

System.out.println("Product found: " + products[binaryIndex]);

} else {

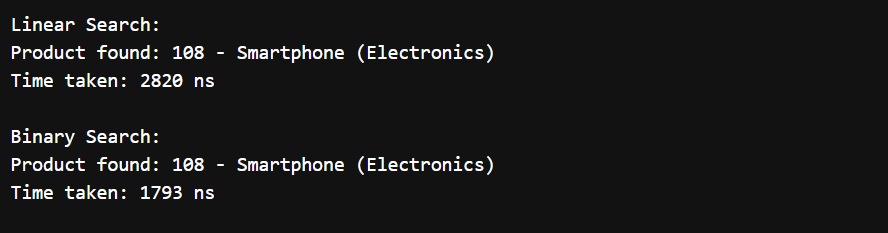
System.out.println("Product not found.");

}

System.out.println("Time taken: " + (endBinary - startBinary) + " ns\n");

}

}



**Exercise 5: Task Management System**

**Scenario:**

**You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.**

class Task {

int taskId;

String taskName;

String status;

Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

@Override

public String toString() {

return taskId + ": " + taskName + " [" + status + "]";

}

}

class TaskNode {

Task task;

TaskNode next;

TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

class TaskLinkedList {

private TaskNode head;

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

return;

}

TaskNode current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public Task searchTask(int taskId) {

TaskNode current = head;

while (current != null) {

if (current.task.taskId == taskId) {

return current.task;

}

current = current.next;

}

return null;

}

public void traverseTasks() {

TaskNode current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

public boolean deleteTask(int taskId) {

if (head == null) return false;

if (head.task.taskId == taskId) {

head = head.next;

return true;

}

TaskNode current = head;

while (current.next != null) {

if (current.next.task.taskId == taskId) {

current.next = current.next.next;

return true;

}

current = current.next;

}

return false;

}

}

public class TaskManager{

public static void main(String[] args) {

TaskLinkedList tasks = new TaskLinkedList();

tasks.addTask(new Task(1, "Design UI", "Pending"));

tasks.addTask(new Task(2, "Implement Backend", "In Progress"));

tasks.addTask(new Task(3, "Write Tests", "Pending"));

System.out.println("All tasks:");

tasks.traverseTasks();

System.out.println("\nSearching for task 2:");

Task found = tasks.searchTask(2);

System.out.println(found != null ? found : "Task not found");

System.out.println("\nDeleting task 1:");

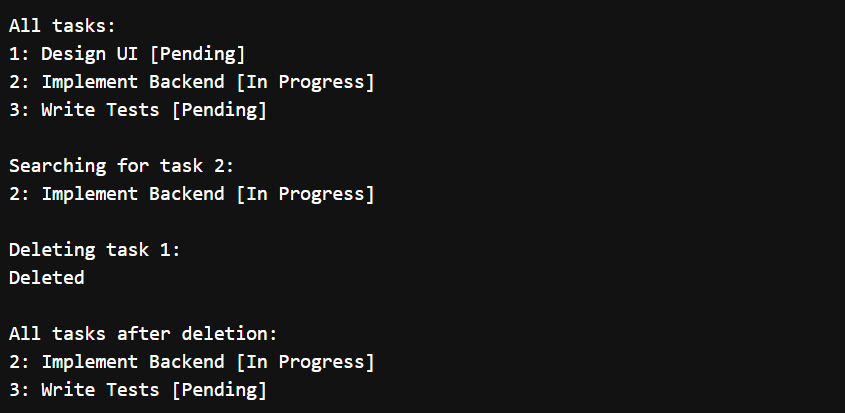
System.out.println(tasks.deleteTask(1) ? "Deleted" : "Not found");

System.out.println("\nAll tasks after deletion:");

tasks.traverseTasks();

}

}



**Exercise 7: Financial Forecasting**

**Scenario:**

**You are developing a financial forecasting tool that predicts future values based on past data.**

public class Recursive {

public static double futureValue(double initial, double rate, int periods) {

if (periods == 0) {

return initial;

}

return futureValue(initial, rate, periods - 1) \* (1 + rate);

}

public static void main(String[] args) {

double initial = 1000.0;

double rate = 0.05;

int periods = 10;

double result = futureValue(initial, rate, periods);

System.out.printf("Future value after %d periods: %.2f\n", periods, result);

}

}

