**1)**

**//InventoryManager**

**import** java.util.HashMap;

**public** **class** InventoryManager {

**private** HashMap<String, Product> inventory;

**public** InventoryManager() {

inventory = **new** HashMap<>();

}

**public** **void** addProduct(Product product) {

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

}

**public** **void** updateProduct(String productId, String name, **int** qty, **double** price) {

Product product = inventory.get(productId);

**if** (product != **null**) {

product.setProductName(name);

product.setQuantity(qty);

product.setPrice(price);

} **else** {

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

}

}

**public** **void** deleteProduct(String productId) {

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

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

}

}

**public** **void** displayInventory() {

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

System.***out***.println(p);

}

}

}

**//Product**

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

**private** String productId;

**private** String productName;

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

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

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

**this**.productId = productId;

**this**.productName = productName;

**this**.quantity = quantity;

**this**.price = price;

}

// Getters and Setters

**public** String getProductId() {

**return** productId;

}

**public** String getProductName() {

**return** productName;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setProductName(String productName) {

**this**.productName = productName;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

@Override

**public** String toString() {

**return** productId + " | " + productName + " | Qty: " + quantity + " | $" + price;

}

}

**//Main**

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

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

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

Product p1 = **new** Product("P001", "Widget", 100, 2.99);

Product p2 = **new** Product("P002", "Gadget", 150, 5.49);

manager.addProduct(p1);

manager.addProduct(p2);

manager.displayInventory();

System.***out***.println("Updating P001...");

manager.updateProduct("P001", "Super Widget", 200, 3.99);

manager.displayInventory();

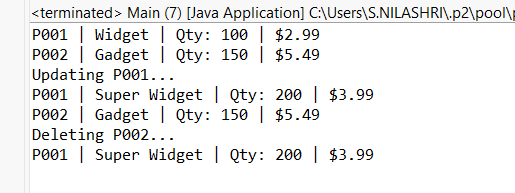
System.***out***.println("Deleting P002...");

manager.deleteProduct("P002");

manager.displayInventory();

}

}



**2)**

**//Product**

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

String productId;

String productName;

String category;

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

**this**.productId = productId;

**this**.productName = productName;

**this**.category = category;

}

@Override

**public** String toString() {

**return** productId + " | " + productName + " | " + category;

}

}

**//Search**

**import** java.util.Arrays;

**import** java.util.Comparator;

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

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

**for** (Product product : products) {

**if** (product.productName.equalsIgnoreCase(targetName)) {

**return** product;

}

}

**return** **null**;

}

**public** **static** Product binarySearch(Product[] products, String targetName) {

**int** left = 0;

**int** right = products.length - 1;

**while** (left <= right) {

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

**int** cmp = products[mid].productName.compareToIgnoreCase(targetName);

**if** (cmp == 0) {

**return** products[mid];

} **else** **if** (cmp < 0) {

left = mid + 1;

} **else** {

right = mid - 1;

}

}

**return** **null**;

}

**public** **static** **void** sortByProductName(Product[] products) {

Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName.toLowerCase()));

}

}

**//Main**

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

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

Product[] products = {

**new** Product("101", "Laptop", "Electronics"),

**new** Product("102", "Chair", "Furniture"),

**new** Product("103", "Phone", "Electronics"),

**new** Product("104", "Desk", "Furniture")

};

System.***out***.println("=== Linear Search ===");

Product result = Search.*linearSearch*(products, "Phone");

System.***out***.println(result != **null** ? result : "Product not found");

System.***out***.println("\n=== Binary Search ===");

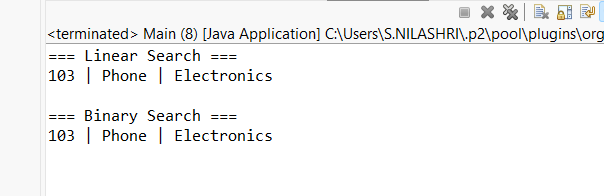
Search.*sortByProductName*(products); // Required before binary search

result = Search.*binarySearch*(products, "Phone");

System.***out***.println(result != **null** ? result : "Product not found");

}

}



**3)**

**//Order**

**public** **class** Order {

String orderId;

String customerName;

**double** totalPrice;

**public** Order(String orderId, String customerName, **double** totalPrice) {

**this**.orderId = orderId;

**this**.customerName = customerName;

**this**.totalPrice = totalPrice;

}

@Override

**public** String toString() {

**return** orderId + " | " + customerName + " | Rs" + totalPrice;

}

}

**//Sorter**

**public** **class** Sorter {

**public** **static** **void** bubbleSort(Order[] orders) {

**int** n = orders.length;

**for** (**int** i = 0; i < n - 1; i++) {

**for** (**int** j = 0; j < n - 1 - i; j++) {

**if** (orders[j].totalPrice < orders[j + 1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

**public** **static** **void** quickSort(Order[] orders, **int** low, **int** high) {

**if** (low < high) {

**int** pi = *partition*(orders, low, high);

*quickSort*(orders, low, pi - 1);

*quickSort*(orders, pi + 1, high);

}

}

**private** **static** **int** partition(Order[] orders, **int** low, **int** high) {

Order pivot = orders[high];

**int** i = low - 1;

**for** (**int** j = low; j < high; j++) {

**if** (orders[j].totalPrice >= pivot.totalPrice) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

**return** i + 1;

}

}

**//Main**

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

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

Order[] orders = {

**new** Order("O001", "Alice", 250.75),

**new** Order("O002", "Bob", 130.00),

**new** Order("O003", "Charlie", 480.00),

**new** Order("O004", "Diana", 320.50)

};

System.***out***.println("=== Original Orders ===");

*printOrders*(orders);

System.***out***.println("\n=== Sorted by Bubble Sort (Descending) ===");

Sorter.*bubbleSort*(orders);

*printOrders*(orders);

orders = **new** Order[] {

**new** Order("O001", "Alice", 250.75),

**new** Order("O002", "Bob", 130.00),

**new** Order("O003", "Charlie", 480.00),

**new** Order("O004", "Diana", 320.50)

};

System.***out***.println("\n=== Sorted by Quick Sort (Descending) ===");

Sorter.*quickSort*(orders, 0, orders.length - 1);

*printOrders*(orders);

}

**public** **static** **void** printOrders(Order[] orders) {

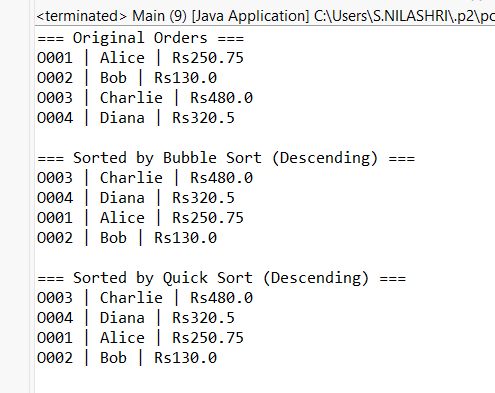
**for** (Order order : orders) {

System.***out***.println(order);

}

}

}



**4)**

**//Employee**

**public** **class** Employee {

String employeeId;

String name;

String position;

**double** salary;

**public** Employee(String employeeId, String name, String position, **double** salary) {

**this**.employeeId = employeeId;

**this**.name = name;

**this**.position = position;

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** employeeId + " " + name + " " + position + " Rs " + salary;

}

}

**//EmployeeManager**

**public** **class** EmployeeManager {

**private** Employee[] employees;

**private** **int** size;

**public** EmployeeManager(**int** capacity) {

employees = **new** Employee[capacity];

size = 0;

}

**public** **void** addEmployee(Employee employee) {

**if** (size < employees.length) {

employees[size++] = employee;

} **else** {

System.***out***.println("Employee array is full.");

}

}

**public** Employee searchEmployee(String id) {

**for** (**int** i = 0; i < size; i++) {

**if** (employees[i].employeeId.equals(id)) {

**return** employees[i];

}

}

**return** **null**;

}

// Traverse (display all employees)

**public** **void** displayEmployees() {

**if** (size == 0) {

System.***out***.println("No employees in the system.");

**return**;

}

**for** (**int** i = 0; i < size; i++) {

System.***out***.println(employees[i]);

}

}

**public** **void** deleteEmployee(String id) {

**for** (**int** i = 0; i < size; i++) {

**if** (employees[i].employeeId.equals(id)) {

**for** (**int** j = i; j < size - 1; j++) {

employees[j] = employees[j + 1];

}

employees[size - 1] = **null**;

size--;

System.***out***.println("Employee with ID " + id + " deleted.");

**return**;

}

}

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

}

}

**//Main**

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

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

EmployeeManager manager = **new** EmployeeManager(5);

manager.addEmployee(**new** Employee("E101", "Alice", "Manager", 75000));

manager.addEmployee(**new** Employee("E102", "Bob", "Developer", 55000));

manager.addEmployee(**new** Employee("E103", "Charlie", "Designer", 50000));

System.***out***.println("=== All Employees ===");

manager.displayEmployees();

System.***out***.println("\n=== Search for E102 ===");

Employee e = manager.searchEmployee("E102");

System.***out***.println(e != **null** ? e : "Employee not found");

System.***out***.println("\n=== Delete E102 ===");

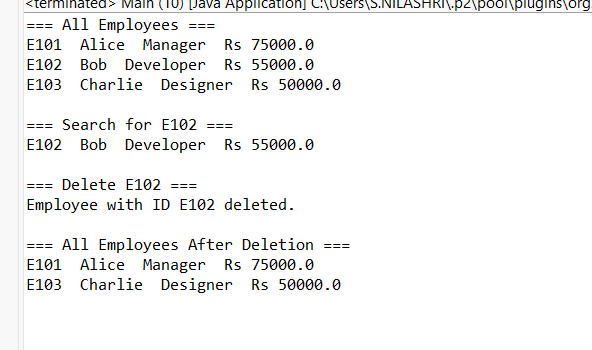
manager.deleteEmployee("E102");

System.***out***.println("\n=== All Employees After Deletion ===");

manager.displayEmployees();

}

}



5)

**//Task**

**public** **class** Task {

String taskId;

String taskName;

String status;

**public** Task(String taskId, String taskName, String status) {

**this**.taskId = taskId;

**this**.taskName = taskName;

**this**.status = status;

}

@Override

**public** String toString() {

**return** taskId + " | " + taskName + " | " + status;

}

}

**//TaskNode**

**public** **class** TaskLinkedList {

**private** TaskNode head;

**public** **void** addTask(Task task) {

TaskNode newNode = **new** TaskNode(task);

**if** (head == **null**) {

head = newNode;

} **else** {

TaskNode current = head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newNode;

}

}

**public** Task searchTask(String taskId) {

TaskNode current = head;

**while** (current != **null**) {

**if** (current.task.taskId.equals(taskId)) {

**return** current.task;

}

current = current.next;

}

**return** **null**;

}

**public** **void** displayTasks() {

**if** (head == **null**) {

System.***out***.println("No tasks in the list.");

**return**;

}

TaskNode current = head;

**while** (current != **null**) {

System.***out***.println(current.task);

current = current.next;

}

}

**public** **void** deleteTask(String taskId) {

**if** (head == **null**) **return**;

**if** (head.task.taskId.equals(taskId)) {

head = head.next;

System.***out***.println("Task deleted.");

**return**;

}

TaskNode current = head;

**while** (current.next != **null** && !current.next.task.taskId.equals(taskId)) {

current = current.next;

}

**if** (current.next != **null**) {

current.next = current.next.next;

System.***out***.println("Task deleted.");

} **else** {

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

}

}

}

**//TaskLinkedList**

**public** **class** TaskLinkedList {

**private** TaskNode head;

**public** **void** addTask(Task task) {

TaskNode newNode = **new** TaskNode(task);

**if** (head == **null**) {

head = newNode;

} **else** {

TaskNode current = head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newNode;

}

}

**public** Task searchTask(String taskId) {

TaskNode current = head;

**while** (current != **null**) {

**if** (current.task.taskId.equals(taskId)) {

**return** current.task;

}

current = current.next;

}

**return** **null**;

}

**public** **void** displayTasks() {

**if** (head == **null**) {

System.***out***.println("No tasks in the list.");

**return**;

}

TaskNode current = head;

**while** (current != **null**) {

System.***out***.println(current.task);

current = current.next;

}

}

**public** **void** deleteTask(String taskId) {

**if** (head == **null**) **return**;

**if** (head.task.taskId.equals(taskId)) {

head = head.next;

System.***out***.println("Task deleted.");

**return**;

}

TaskNode current = head;

**while** (current.next != **null** && !current.next.task.taskId.equals(taskId)) {

current = current.next;

}

**if** (current.next != **null**) {

current.next = current.next.next;

System.***out***.println("Task deleted.");

} **else** {

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

}

}

}

**//Main**

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

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

TaskLinkedList taskList = **new** TaskLinkedList();

taskList.addTask(**new** Task("T001", "Design UI", "Pending"));

taskList.addTask(**new** Task("T002", "Implement Backend", "In Progress"));

taskList.addTask(**new** Task("T003", "Write Tests", "Pending"));

System.***out***.println("=== All Tasks ===");

taskList.displayTasks();

System.***out***.println("\n=== Searching for Task T002 ===");

Task found = taskList.searchTask("T002");

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

System.***out***.println("\n=== Deleting Task T002 ===");

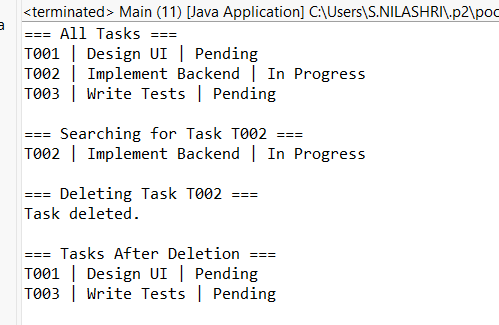
taskList.deleteTask("T002");

System.***out***.println("\n=== Tasks After Deletion ===");

taskList.displayTasks();

}

}



**6)**

**//Book**

**public** **class** Book {

String bookId;

String title;

String author;

**public** Book(String bookId, String title, String author) {

**this**.bookId = bookId;

**this**.title = title;

**this**.author = author;

}

@Override

**public** String toString() {

**return** bookId + " | " + title + " | " + author;

}

}

**//SearchLibrary**

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** SearchLib {

**public** **static** Book linearSearchByTitle(Book[] books, String targetTitle) {

**for** (Book book : books) {

**if** (book.title.equalsIgnoreCase(targetTitle)) {

**return** book;

}

}

**return** **null**;

}

**public** **static** Book binarySearchByTitle(Book[] books, String targetTitle) {

**int** left = 0, right = books.length - 1;

**while** (left <= right) {

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

**int** cmp = books[mid].title.compareToIgnoreCase(targetTitle);

**if** (cmp == 0) {

**return** books[mid];

} **else** **if** (cmp < 0) {

left = mid + 1;

} **else** {

right = mid - 1;

}

}

**return** **null**;

}

**public** **static** **void** sortBooksByTitle(Book[] books) {

Arrays.*sort*(books, Comparator.*comparing*(b -> b.title.toLowerCase()));

}

}

**//Main**

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

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

Book[] books = {

**new** Book("B101", "Java Programming", "Alice"),

**new** Book("B102", "Data Structures", "Bob"),

**new** Book("B103", "Algorithms", "Charlie"),

**new** Book("B104", "Operating Systems", "David")

};

System.***out***.println("=== Linear Search for 'Algorithms' ===");

Book result = SearchLib.*linearSearchByTitle*(books, "Algorithms");

System.***out***.println(result != **null** ? result : "Book not found");

SearchLib.*sortBooksByTitle*(books);

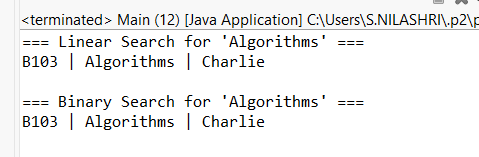
System.***out***.println("\n=== Binary Search for 'Algorithms' ===");

result = SearchLib.*binarySearchByTitle*(books, "Algorithms");

System.***out***.println(result != **null** ? result : "Book not found");

}

}



**7)**

**//FinancialForcast**

**public** **class** FinancialForecast {

**public** **static** **double** futureValue(**double** presentValue, **double** rate, **int** years) {

**if** (years == 0) {

**return** presentValue;

}

**return** *futureValue*(presentValue \* (1 + rate), rate, years - 1); // Recursive call

}

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

**double** presentValue = 10000;

**double** rate = 0.05;

**int** years = 5;

**double** result = *futureValue*(presentValue, rate, years);

System.***out***.print("Future value after "+ years+" Rs : "+ result);

}

}

