| **Exercise 1: Inventory Management System**  **Scenario:** You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.  **Solutions:**  **Project Name: InventoryManagementSystem**  **Product.java**  public class Product {  private int productId;  private String productName;  private int quantity;  private double price;  public Product(int productId, String productName, int quantity, double price) {  this.productId = productId;  this.productName = productName;  this.quantity = quantity;  this.price = price;  }  // Getters and Setters  public int 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 "Product [ID=" + productId + ", Name=" + productName +  ", Quantity=" + quantity + ", Price=₹" + price + "]";  }  }  **InventoryManager.java**  import java.util.HashMap;  public class InventoryManager {  private HashMap<Integer, Product> inventory = new HashMap<>();  // Add product  public void addProduct(Product product) {  inventory.put(product.getProductId(), product);  System.out.println("Product added: " + product);  }  // Update product  public void updateProduct(int productId, String name, int quantity, double price) {  Product product = inventory.get(productId);  if (product != null) {  product.setProductName(name);  product.setQuantity(quantity);  product.setPrice(price);  System.out.println("Product updated: " + product);  } else {  System.out.println("Product not found!");  }  }  // Delete product  public void deleteProduct(int productId) {  if (inventory.containsKey(productId)) {  Product removed = inventory.remove(productId);  System.out.println("Product deleted: " + removed);  } else {  System.out.println("Product not found!");  }  }  // Display all  public void displayInventory() {  System.out.println("Current Inventory:");  for (Product product : inventory.values()) {  System.out.println(product);  }  }  }  **InventoryTest.java**  public class InventoryTest {  public static void main(String[] args) {  InventoryManager manager = new InventoryManager();  // Add Products  manager.addProduct(new Product(101, "Laptop", 10, 55000));  manager.addProduct(new Product(102, "Mouse", 50, 500));  manager.addProduct(new Product(103, "Keyboard", 30, 1200));  // Update Product  manager.updateProduct(102, "Wireless Mouse", 60, 650);  // Delete Product  manager.deleteProduct(103);  // Display Inventory  manager.displayInventory();  }  }  **Output:**    **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.  **Solutions:**  **Java Project:EcommerceSearchSystem**  **Product.java**  public class Product {  private int productId;  private String productName;  private String category;  public Product(int productId, String productName, String category) {  this.productId = productId;  this.productName = productName;  this.category = category;  }  public int getProductId() { return productId; }  public String getProductName() { return productName; }  public String getCategory() { return category; }  @Override  public String toString() {  return "Product [ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";  }  }  **SearchAlgorithms.java**  import java.util.Arrays;  import java.util.Comparator;  public class SearchAlgorithms {  // Linear search by product name  public static Product linearSearch(Product[] products, String name) {  for (Product product : products) {  if (product.getProductName().equalsIgnoreCase(name)) {  return product;  }  }  return null;  }  // Binary search by product name (sorted array required)  public static Product binarySearch(Product[] products, String name) {  Arrays.sort(products, Comparator.comparing(Product::getProductName));  int left = 0, right = products.length - 1;  while (left <= right) {  int mid = left + (right - left) / 2;  int comparison = products[mid].getProductName().compareToIgnoreCase(name);  if (comparison == 0) {  return products[mid];  } else if (comparison < 0) {  left = mid + 1;  } else {  right = mid - 1;  }  }  return null;  }  }  **SearchTest.java**  public class SearchTest {  public static void main(String[] args) {  Product[] products = {  new Product(101, "Laptop", "Electronics"),  new Product(102, "Shoes", "Footwear"),  new Product(103, "Keyboard", "Electronics"),  new Product(104, "Watch", "Accessories"),  new Product(105, "Phone", "Electronics")  };  // Linear Search  Product result1 = SearchAlgorithms.linearSearch(products, "Watch");  System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not found"));  // Binary Search  Product result2 = SearchAlgorithms.binarySearch(products, "Keyboard");  System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not found"));  }  }  **Output:**    **Exercise 3: Sorting Customer Orders**  **Scenario:**  You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.  **Solutions:**  **Java Project:**  **Order.java**  public class Order {  private int orderId;  private String customerName;  private double totalPrice;  public Order(int orderId, String customerName, double totalPrice) {  this.orderId = orderId;  this.customerName = customerName;  this.totalPrice = totalPrice;  }  public double getTotalPrice() {  return totalPrice;  }  public String toString() {  return "Order [ID=" + orderId + ", Customer=" + customerName + ", Price=₹" + totalPrice + "]";  }  }  **SortAlgorithms.java**  public class SortAlgorithms {  // Bubble Sort by totalPrice  public static void bubbleSort(Order[] orders) {  int n = orders.length;  for (int i = 0; i < n - 1; i++) {  boolean swapped = false;  for (int j = 0; j < n - i - 1; j++) {  if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {  Order temp = orders[j];  orders[j] = orders[j + 1];  orders[j + 1] = temp;  swapped = true;  }  }  if (!swapped) break;  }  }  // Quick Sort by totalPrice  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) {  double pivot = orders[high].getTotalPrice();  int i = low - 1;  for (int j = low; j < high; j++) {  if (orders[j].getTotalPrice() <= pivot) {  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;  }  }  **OrderSortTest.java**  public class OrderSortTest {  public static void main(String[] args) {  Order[] orders = {  new Order(101, "Harini", 2500),  new Order(102, "Abinaya", 1800),  new Order(103, "Sathya", 5400),  new Order(104, "Ravi", 1300)  };  System.out.println("Original Orders:");  for (Order o : orders) System.out.println(o);  // Bubble Sort  SortAlgorithms.bubbleSort(orders);  System.out.println("\nOrders Sorted by Bubble Sort:");  for (Order o : orders) System.out.println(o);  // Reset orders  orders = new Order[]{  new Order(101, "Harini", 2500),  new Order(102, "Abinaya", 1800),  new Order(103, "Sathya", 5400),  new Order(104, "Ravi", 1300)  };  // Quick Sort  SortAlgorithms.quickSort(orders, 0, orders.length - 1);  System.out.println("\nOrders Sorted by Quick Sort:");  for (Order o : orders) System.out.println(o);  }  }  **Output:**  **Exercise 4: Employee Management System**  **Scenario:** You are developing an employee management system for a company. Efficiently managing employee records is crucial.  **Solutions:**  **Java Project:EmployeeManagementSystem**  **Employee.java**  public class Employee {  private int employeeId;  private String name;  private String position;  private double salary;  public Employee(int employeeId, String name, String position, double salary) {  this.employeeId = employeeId;  this.name = name;  this.position = position;  this.salary = salary;  }  public int getEmployeeId() {  return employeeId;  }  public String toString() {  return "Employee [ID=" + employeeId + ", Name=" + name +  ", Position=" + position + ", Salary=₹" + salary + "]";  }  }  **EmployeeManager.java**  public class EmployeeManager {  private Employee[] employees;  private int size = 0;  public EmployeeManager(int capacity) {  employees = new Employee[capacity];  }  // Add employee  public void addEmployee(Employee employee) {  if (size < employees.length) {  employees[size++] = employee;  System.out.println("Employee added: " + employee);  } else {  System.out.println("Array is full. Cannot add more employees.");  }  }  // Search employee by ID  public Employee searchEmployee(int id) {  for (int i = 0; i < size; i++) {  if (employees[i].getEmployeeId() == id) {  return employees[i];  }  }  return null;  }  // Traverse all employees  public void listEmployees() {  if (size == 0) {  System.out.println("No employees found.");  return;  }  System.out.println("All Employees:");  for (int i = 0; i < size; i++) {  System.out.println(employees[i]);  }  }  // Delete employee by ID  public void deleteEmployee(int id) {  boolean found = false;  for (int i = 0; i < size; i++) {  if (employees[i].getEmployeeId() == id) {  found = true;  for (int j = i; j < size - 1; j++) {  employees[j] = employees[j + 1];  }  employees[--size] = null;  System.out.println("Employee with ID " + id + " deleted.");  break;  }  }  if (!found) {  System.out.println("Employee with ID " + id + " not found.");  }  }  }  **EmployeeTest.java**  public class EmployeeTest {  public static void main(String[] args) {  EmployeeManager manager = new EmployeeManager(5);  // Add employees  manager.addEmployee(new Employee(1, "Harini", "Developer", 50000));  manager.addEmployee(new Employee(2, "Abinaya", "Manager", 65000));  manager.addEmployee(new Employee(3, "Sathya", "Tester", 45000));  // List all employees  manager.listEmployees();  // Search employee  Employee e = manager.searchEmployee(2);  System.out.println(e != null ? "Found: " + e : "Employee not found.");  // Delete employee  manager.deleteEmployee(2);  // List again  manager.listEmployees();  }  }  **Output:**    **Exercise 5: Task Management System**  **Scenario:**  You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.  **Solutions:**  **Java Project:TaskManagementSystem**  **Task.java**  public class Task {  private int taskId;  private String taskName;  private String status;  public Task(int taskId, String taskName, String status) {  this.taskId = taskId;  this.taskName = taskName;  this.status = status;  }  public int getTaskId() {  return taskId;  }  public String toString() {  return "Task [ID=" + taskId + ", Name=" + taskName + ", Status=" + status + "]";  }  }  **TaskNode.java**  public class TaskNode {  Task task;  TaskNode next;  public TaskNode(Task task) {  this.task = task;  this.next = null;  }  }  **TaskLinkedList.java**  public class TaskLinkedList {  private TaskNode head;  // Add task at the end  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;  }  System.out.println("Task added: " + task);  }  // Search task by ID  public Task searchTask(int id) {  TaskNode current = head;  while (current != null) {  if (current.task.getTaskId() == id) {  return current.task;  }  current = current.next;  }  return null;  }  // Traverse all tasks  public void listTasks() {  if (head == null) {  System.out.println("No tasks available.");  return;  }  System.out.println("All Tasks:");  TaskNode current = head;  while (current != null) {  System.out.println(current.task);  current = current.next;  }  }  // Delete task by ID  public void deleteTask(int id) {  if (head == null) {  System.out.println("List is empty.");  return;  }  if (head.task.getTaskId() == id) {  head = head.next;  System.out.println("Task with ID " + id + " deleted.");  return;  }  TaskNode current = head;  while (current.next != null && current.next.task.getTaskId() != id) {  current = current.next;  }  if (current.next == null) {  System.out.println("Task with ID " + id + " not found.");  } else {  current.next = current.next.next;  System.out.println("Task with ID " + id + " deleted.");  }  }  }  **TaskManagerTest.java**  public class TaskManagerTest {  public static void main(String[] args) {  TaskLinkedList manager = new TaskLinkedList();  // Add Tasks  manager.addTask(new Task(1, "Design UI", "Pending"));  manager.addTask(new Task(2, "Develop Backend", "In Progress"));  manager.addTask(new Task(3, "Write Tests", "Not Started"));  // List Tasks  manager.listTasks();  // Search Task  Task task = manager.searchTask(2);  System.out.println(task != null ? "Found: " + task : "Task not found.");  // Delete Task  manager.deleteTask(2);  // List again  manager.listTasks();  }  }  **Output:**    **Exercise 6: Library Management System**  **Scenario:** You are developing a library management system where users can search for books by title or author.  **Solutions:**  **Java Project:LibrarySearchSystem**  **Book.java**  public class Book implements Comparable<Book> {  private int bookId;  private String title;  private String author;  public Book(int bookId, String title, String author) {  this.bookId = bookId;  this.title = title.toLowerCase(); // For case-insensitive search  this.author = author;  }  public String getTitle() {  return title;  }  public String toString() {  return "Book [ID=" + bookId + ", Title=" + title + ", Author=" + author + "]";  }  @Override  public int compareTo(Book other) {  return this.title.compareTo(other.title);  }  }  **BookSearch.java**  import java.util.Arrays;  public class BookSearch {  // Linear Search  public static Book linearSearch(Book[] books, String title) {  title = title.toLowerCase();  for (Book book : books) {  if (book.getTitle().equals(title)) {  return book;  }  }  return null;  }  // Binary Search  public static Book binarySearch(Book[] books, String title) {  title = title.toLowerCase();  int left = 0;  int right = books.length - 1;  while (left <= right) {  int mid = left + (right - left) / 2;  int cmp = books[mid].getTitle().compareTo(title);  if (cmp == 0) {  return books[mid];  } else if (cmp < 0) {  left = mid + 1;  } else {  right = mid - 1;  }  }  return null;  }  }  **LibraryTest.java**  import java.util.Arrays;  public class LibraryTest {  public static void main(String[] args) {  Book[] books = {  new Book(101, "Java Programming", "James Gosling"),  new Book(102, "Python Basics", "Guido van Rossum"),  new Book(103, "C Programming", "Dennis Ritchie"),  new Book(104, "Data Structures", "Narasimha Karumanchi")  };  // Linear Search (no need to sort)  System.out.println(" Linear Search:");  Book result = BookSearch.linearSearch(books, "Python Basics");  System.out.println(result != null ? "Found: " + result : "Book not found.");  // Sort for Binary Search  Arrays.sort(books); // uses compareTo()  System.out.println("\nSorted Books for Binary Search:");  for (Book b : books) System.out.println(b);  // Binary Search  System.out.println("\n Binary Search:");  Book result2 = BookSearch.binarySearch(books, "Python Basics");  System.out.println(result2 != null ? "Found: " + result2 : "Book not found.");  }  }  **Output:**    **Exercise 7: Financial Forecasting**  **Scenario:**  You are developing a financial forecasting tool that predicts future values based on past data  **Solutions:**  **Java Project:**  **FinancialForecast.java**  public class FinancialForecast {  // Recursive method to calculate future value  public static double calculateFutureValue(double initialAmount, double growthRate, int years) {  if (years == 0) {  return initialAmount;  }  return (1 + growthRate) \* calculateFutureValue(initialAmount, growthRate, years - 1);  }  // Optimized version using memoization (if needed for large inputs)  public static double calculateFutureValueMemo(double initialAmount, double growthRate, int years, Double[] memo) {  if (years == 0) return initialAmount;  if (memo[years] != null) return memo[years];  memo[years] = (1 + growthRate) \* calculateFutureValueMemo(initialAmount, growthRate, years - 1, memo);  return memo[years];  }  }  **ForecastTest.java**  public class ForecastTest {  public static void main(String[] args) {  double initialAmount = 10000.0; // Rs.10,000  double growthRate = 0.10; // 10% per year  int years = 5;  double result = FinancialForecast.calculateFutureValue(initialAmount, growthRate, years);  System.out.printf("📈 Future value after %d years: ₹%.2f\n", years, result);  // Optimized approach for larger input  Double[] memo = new Double[years + 1];  double optimizedResult = FinancialForecast.calculateFutureValueMemo(initialAmount, growthRate, years, memo);  System.out.printf("⚡ Optimized future value after %d years: ₹%.2f\n", years, optimizedResult);  }  }  **Output:** |
| --- |