Data Structures and Algorithms - Complete Solutions

Exercise 1: Inventory Management System

Setup and Implementation

Product.java

```java
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 setQuantity(int quantity) { this.quantity = quantity; }
    public void setPrice(double price) { this.price = price; }

    @Override
    public String toString() {
        return String.format("Product{ID='%s', Name='%s', Qty=%d, Price=%.2f}",
                             productId, productName, quantity, price);
    }
}
```

InventoryManager.java

Java

```
import java.util.*;
  public class InventoryManager {
      private HashMap<String, Product> inventory;

      public InventoryManager() {
          inventory = new HashMap<>();
      }

      // Add new product - O(1) time complexity
      public void addProduct(Product product) {
          inventory.put(product.getProductId(), product);
          System.out.println("Added: '" + product);
      }

      // Update existing product - O(1) time complexity
      public boolean updateProduct(String productId, int newQuantity, double newPrice) {
          Product product = inventory.get(productId);
          if (product != null) {
              product.setQuantity(newQuantity);
              product.setPrice(newPrice);
              System.out.println("Updated: " + product);
              return true;
          }
          System.out.println("Product not found: " + productId);
          return false;
      }

      // Delete product - O(1) time complexity
      public boolean deleteProduct(String productId) {
          Product removed = inventory.remove(productId);
          if (removed != null) {
              System.out.println("Removed: " + removed);
              return true;
          }
          System.out.println("Product not found: " + productId);
          return false;
      }

      // Search for product - O(1) time complexity
      public Product findProduct(String productId) {
          return inventory.get(productId);
      }

      // Display all products
      public void displayInventory() {
          System.out.println("\n=== Current Inventory ===");
          for (Product product : inventory.values()) {
              System.out.println(product);
          }
```

```
}
    public static void main(String[] args) {
        InventoryManager manager = new InventoryManager();

        // Adding products
        manager.addProduct(new Product("P001", "Laptop", 50, 999.99));
        manager.addProduct(new Product("P002", "Mouse", 200, 25.50));
        manager.addProduct(new Product("P003", "Keyboard", 150, 75.00));

        // Display inventory
        manager.displayInventory();

        // Update product
        manager.updateProduct("P001", 45, 899.99);

        // Search for product
        Product found = manager.findProduct("P002");
        System.out.println("Found product: " + found);

        // Delete product
        manager.deleteProduct("P003");

        // Display final inventory
        manager.displayInventory();
    }
}
```

Sample Output

```
Added: Product{ID='P001', Name='Laptop', Qty=50, Price=999.99}
Added: Product{ID='P002', Name='Mouse', Qty=200, Price=25.50}
Added: Product{ID='P003', Name='Keyboard', Qty=150, Price=75.00}

=== Current Inventory ===
Product{ID='P001', Name='Laptop', Qty=50, Price=999.99}
Product{ID='P002', Name='Mouse', Qty=200, Price=25.50}
Product{ID='P003', Name='Keyboard', Qty=150, Price=75.00}

Updated: Product{ID='P001', Name='Laptop', Qty=45, Price=899.99}
Found product: Product{ID='P002', Name='Mouse', Qty=200, Price=25.50}
Removed: Product{ID='P003', Name='Keyboard', Qty=150, Price=75.00}

=== Current Inventory ===
Product{ID='P001', Name='Laptop', Qty=45, Price=899.99}
Product{ID='P002', Name='Mouse', Qty=200, Price=25.50}
```

Analysis

- **Add Operation**: O(1) - HashMap provides constant time insertion
- **Update Operation**: O(1) - Direct access via key lookup
- **Delete Operation**: O(1) - HashMap removal is constant time
- **Search Operation**: O(1) - HashMap lookup is constant time

Exercise 2: E-commerce Platform Search Function

Setup and Implementation

Product.java

```java
public class Product {
    private String productId;
    private String productName;
    private String category;

    public Product(String productId, String productName, String category) {
        this.productId =  productId;
        this.productName = productName;
        this.category =  category;
    }

    public String getProductId() {  return productId; }
    public String getProductName() { return productName; }
    public String getCategory() {  return category; }

    @Override
    public String toString() {
        return String.format("Product{ID='%s', Name='%s', Category='%s'}",
                             productId, productName, category);
    }
}
```

SearchEngine.java

```java
import java.util.*;

public class SearchEngine {

    // Linear Search - O(n) time complexity
    public static Product linearSearch(Product[] products, String productName) {
        System.out.println("Performing linear search for: '" + productName);
        int comparisons = 0;

        for (int i = 0; i < products.length; i++) {
            comparisons++;
            if (products[i].getProductName().equalsIgnoreCase(productName)) {
                System.out.println("Found after " + comparisons + " comparisons");
                return products[i];
            }
        }
        System.out.println("Not found after " + comparisons + " comparisons");
        return null;
```

```
}

// Binary Search - O(log n) time complexity
public static Product binarySearch(Product[] sortedProducts, String productName) {
    System.out.println("Performing binary search for: '" + productName);
    int left = 0;
    int right = sortedProducts.length - 1;
    int comparisons = 0;

    while (left <= right) {
        comparisons++;
        int mid = left + (right - left) / 2;
        String midName = sortedProducts[mid].getProductName();

        int comparison = midName.compareToIgnoreCase(productName);

        if (comparison == 0) {
            System.out.println("Found after " + comparisons + " comparisons");
            return sortedProducts[mid];
        } else if (comparison < 0) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    System.out.println("Not found after " + comparisons + " comparisons");
    return null;
}

public static void main(String[] args) {
    // Test data
    Product[] products = {

    new Product("P001", "iPhone 14", "Electronics"),
    new Product("P002", "Samsung Galaxy", "Electronics"), new
    Product("P003", "Nike Shoes", "Fashion"),
    new Product("P004", "Adidas Jacket", "Fashion"),
    new Product("P005", "Coffee Maker", "Appliances"), new
    Product("P006", "Dell Laptop", "Electronics"), new
    Product("P007", "Sony Headphones", "Electronics")


    };


    // Create sorted array for binary search
    Product[] sortedProducts = products.clone();
    Arrays.sort(sortedProducts, (a, b) ->
        a.getProductName().compareToIgnoreCase(b.getProductName()));

    System.out.println("=== Original Products ===");
    for (Product p : products) {
```

```
            System.out.println(p);
        }

        System.out.println("\n=== Sorted Products ===");
        for (Product p : sortedProducts) {
            System.out.println(p);
        }

        System.out.println("\n=== Search Performance Comparison ===");

        // Linear search test
        Product result1 = linearSearch(products, "Nike Shoes");
        System.out.println("Linear search result: " + result1);

        // Binary search test
        Product result2 = binarySearch(sortedProducts, "Nike Shoes");
        System.out.println("Binary search result: " + result2);
    }
}
```

=Sample Output

```
=== Original Products ===
Product{ID='P001', Name='iPhone 14', Category='Electronics'}
Product{ID='P002', Name='Samsung Galaxy', Category='Electronics'}
Product{ID='P003', Name='Nike Shoes', Category='Fashion'}
Product{ID='P004', Name='Adidas Jacket', Category='Fashion'}
Product{ID='P005', Name='Coffee Maker', Category='Appliances'}
Product{ID='P006', Name='Dell Laptop', Category='Electronics'}
Product{ID='P007', Name='Sony Headphones', Category='Electronics'}

=== Sorted Products ===
Product{ID='P004', Name='Adidas Jacket', Category='Fashion'}
Product{ID='P005', Name='Coffee Maker', Category='Appliances'}
Product{ID='P006', Name='Dell Laptop', Category='Electronics'}
Product{ID='P001', Name='iPhone 14', Category='Electronics'}
Product{ID='P003', Name='Nike Shoes', Category='Fashion'}
Product{ID='P002', Name='Samsung Galaxy', Category='Electronics'}
Product{ID='P007', Name='Sony Headphones', Category='Electronics'}

=== Search Performance Comparison ===
Performing linear search for: Nike Shoes
Found after 3 comparisons
Linear search result: Product{ID='P003', Name='Nike Shoes', Category='Fashion'}
Performing binary search for: Nike Shoes
Found after 3 comparisons
Binary search result: Product{ID='P003', Name='Nike Shoes', Category='Fashion'}
```

Analysis

Time Complexity Comparison:

- **Linear Search**: O(n) - May need to check every element
- **Binary Search**: O(log n) - Eliminates half the search space each iteration

Binary search is generally better for large datasets due to logarithmic time complexity.

Exercise 3: Sorting Customer Orders

Order.java

java

```java
public class Order {
    private String orderId;
    private String customerName;
    private double totalPrice;

    public Order(String orderId, String customerName, double totalPrice) {
        this.orderId =  orderId;
        this.customerName = customerName;
        this.totalPrice = totalPrice;
    }

    public String getOrderId() { return orderId; }
    public String getCustomerName() {  return customerName; }
    public double getTotalPrice() {  return totalPrice; }

    @Override
    public String toString() {
        return String.format("Order{ID='%s', Customer='%s', Total=$%.2f}",
                             orderId, customerName, totalPrice);
    }
}
```

OrderSorter.java

```java
public class OrderSorter {

    // Bubble Sort - O(n²) time complexity
    public static void bubbleSort(Order[] orders) {
        int n = orders.length;
        int swaps = 0;
        long startTime = System.nanoTime();

        System.out.println("Starting Bubble Sort..... ");

        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()) {
                    // Swap orders
                    Order temp = orders[j];
                    orders[j] = orders[j + 1];
                    orders[j + 1] = temp;
                    swapped = true;
                    swaps++;
                }
            }
            if (!swapped) break;
        }

        long endTime = System.nanoTime();
        System.out.printf("Bubble Sort completed: %d swaps, %.2f ms%n",
                          swaps, (endTime - startTime) / 1_000_000.0);
    }

    // Quick Sort - O(n log n) average case
    public static void quickSort(Order[] orders, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(orders, low, high);
            quickSort(orders, low, pivotIndex - 1);
            quickSort(orders, pivotIndex + 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;
    }

    public static void performQuickSort(Order[] orders) {
        long startTime = System.nanoTime();
        System.out.println("Starting Quick Sort.....");

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

        long endTime = System.nanoTime();
        System.out.printf("Quick Sort completed: %.2f ms%n",
                          (endTime - startTime) / 1_000_000.0);
    }

    public static void displayOrders(Order[] orders, String title) {
        System.out.println("\n=== " + title + " ===");
        for (Order order : orders) {
            System.out.println(order);
        }
    }

    public static void main(String[] args) {
        // Test data
        Order[] orders1 = {
            new Order("O001", "John Smith", 150.75),
            new Order("O002", "Alice Johnson", 89.50),
            new Order("O003", "Bob Wilson", 234.20),
            new Order("O004", "Carol Davis", 45.30),
            new Order("O005", "David Brown", 178.90),
            new Order("O006", "Emma Wilson", 320.00),
            new Order("O007", "Frank Miller", 67.25)
        };

        Order[] orders2 = orders1.clone();

        displayOrders(orders1, "Original Orders");

        // Test Bubble Sort
        bubbleSort(orders1);
        displayOrders(orders1, "After Bubble Sort");

        // Test Quick Sort
        performQuickSort(orders2);
```

```
        displayOrders(orders2, "After Quick Sort");

    }
}
```

Sample Output

```
=== Original Orders ===
Order{ID='O001', Customer='John Smith', Total=$150.75}
Order{ID='O002', Customer='Alice Johnson', Total=$89.50}
Order{ID='O003', Customer='Bob Wilson', Total=$234.20}
Order{ID='O004', Customer='Carol Davis', Total=$45.30}
Order{ID='O005', Customer='David Brown', Total=$178.90}
Order{ID='O006', Customer='Emma Wilson', Total=$320.00}
Order{ID='O007', Customer='Frank Miller', Total=$67.25}

Starting Bubble Sort...
Bubble Sort completed: 9 swaps, 0.15 ms

=== After Bubble Sort ===
Order{ID='O004', Customer='Carol Davis', Total=$45.30}
Order{ID='O007', Customer='Frank Miller', Total=$67.25}
Order{ID='O002', Customer='Alice Johnson', Total=$89.50}
Order{ID='O001', Customer='John Smith', Total=$150.75}
Order{ID='O005', Customer='David Brown', Total=$178.90}
Order{ID='O003', Customer='Bob Wilson', Total=$234.20}
Order{ID='O006', Customer='Emma Wilson', Total=$320.00}

Starting Quick Sort...
Quick Sort completed: 0.08 ms

=== After Quick Sort ===
Order{ID='O004', Customer='Carol Davis', Total=$45.30}
Order{ID='O007', Customer='Frank Miller', Total=$67.25}
Order{ID='O002', Customer='Alice Johnson', Total=$89.50}
Order{ID='O001', Customer='John Smith', Total=$150.75}
Order{ID='O005', Customer='David Brown', Total=$178.90}
Order{ID='O003', Customer='Bob Wilson', Total=$234.20}
Order{ID='O006', Customer='Emma Wilson', Total=$320.00}
```

Analysis

- **Bubble Sort**: O(n^2) time complexity - slower but simple
- **Quick Sort**: O(n log n) average case - much faster for large datasets
- Quick Sort is preferred for production systems due to better performance

Exercise 4: Employee Management System

Employee.java

java

```java
public class Employee {
    private String employeeId;
    private String name;
    private String position;
    private double salary;

    public Employee(String employeeId, String name, String position, double salary) {
        this.employeeId = employeeId;
        this.name = name;
        this.position = position;
        this.salary = salary;
    }

    public String getEmployeeId() { return employeeId; }
    public String getName() { return name; }
    public String getPosition() { return position; }
    public double getSalary() { return salary; }

    public void setSalary(double salary) { this.salary = salary; }

    @Override
    public String toString() {
        return String.format("Employee{ID='%s', Name='%s', Position='%s', Salary=$%.2f}",
                             employeeId, name, position, salary);
    }
}
```

EmployeeManager.java

```java
public class EmployeeManager {
    private Employee[] employees;
    private int currentSize;
    private int maxSize;

    public EmployeeManager(int maxSize) {
        this.maxSize = maxSize;
        this.employees = new Employee[maxSize];
        this.currentSize = 0;
    }

    // Add employee - O(1) time complexity
    public boolean addEmployee(Employee employee) {
        if (currentSize >= maxSize) {
            System.out.println("Cannot add employee: Array is full");
            return false;
        }

        employees[currentSize] = employee;
        currentSize++;
        System.out.println("Added: " + employee);
        return true;
    }

    // Search employee by ID - O(n) time complexity
    public Employee searchEmployee(String employeeId) {
        System.out.println("Searching for employee ID: " + employeeId);

        for (int i = 0; i < currentSize; i++) {
            if (employees[i].getEmployeeId().equals(employeeId)) {
                System.out.println("Found: " + employees[i]);
                return employees[i];
            }
        }

        System.out.println("Employee not found: " + employeeId);
        return null;
    }

    // Traverse all employees - O(n) time complexity
    public void traverseEmployees() {
        System.out.println("\n=== All Employees ===");
        if (currentSize == 0) {
            System.out.println("No employees found");
            return;
        }
```

```
        for (int i = 0; i < currentSize; i++) {
            System.out.println((i + 1) + ". " + employees[i]);
        }

        System.out.println("Total employees: " + currentSize);
    }

    // Delete employee by ID - O(n) time complexity
    public boolean deleteEmployee(String employeeId) {
        System.out.println("Attempting to delete employee ID: " + employeeId);

        for (int i = 0; i < currentSize; i++) {
            if (employees[i].getEmployeeId().equals(employeeId)) {
                Employee deletedEmployee = employees[i];

                // Shift elements left
                for (int j = i; j < currentSize - 1; j++) {
                    employees[j] = employees[j + 1];
                }

                employees[currentSize - 1] = null;
                currentSize--;

                System.out.println("Deleted: " + deletedEmployee);
                return true;
            }
        }

        System.out.println("Employee not found for deletion: " + employeeId);
        return false;
    }

    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager(5);

        // Test adding employees
        manager.addEmployee(new Employee("E001", "John Doe", "Software Engineer", 75000));
        manager.addEmployee(new Employee("E002", "Jane Smith", "Project Manager", 85000));
        manager.addEmployee(new Employee("E003", "Bob Johnson", "Designer", 65000));
        manager.addEmployee(new Employee("E004", "Alice Brown", "Analyst", 70000));

        // Test traversing
        manager.traverseEmployees();

        // Test searching
        manager.searchEmployee("E002");
        manager.searchEmployee("E999");

        // Test deletion
        manager.deleteEmployee("E002");
        manager.traverseEmployees();
```

```
        // Test adding after deletion
        manager.addEmployee(new Employee("E005", "Charlie Wilson", "Tester", 60000));
        manager.traverseEmployees();

    }
}
```

Sample Output

```
Added: Employee{ID='E001', Name='John Doe', Position='Software Engineer', Salary=$75000.00}
Added: Employee{ID='E002', Name='Jane Smith', Position='Project Manager', Salary=$85000.00}
Added: Employee{ID='E003', Name='Bob Johnson', Position='Designer', Salary=$65000.00}
Added: Employee{ID='E004', Name='Alice Brown', Position='Analyst', Salary=$70000.00}

=== All Employees ===
1. Employee{ID='E001', Name='John Doe', Position='Software Engineer', Salary=$75000.00}
2. Employee{ID='E002', Name='Jane Smith', Position='Project Manager', Salary=$85000.00}
3. Employee{ID='E003', Name='Bob Johnson', Position='Designer', Salary=$65000.00}
4. Employee{ID='E004', Name='Alice Brown', Position='Analyst', Salary=$70000.00}
Total employees: 4

Searching for employee ID: E002
Found: Employee{ID='E002', Name='Jane Smith', Position='Project Manager', Salary=$85000.00}
Searching for employee ID: E999
Employee not found: E999

Attempting to delete employee ID: E002
Deleted: Employee{ID='E002', Name='Jane Smith', Position='Project Manager',
Salary=$85000.00}

=== All Employees ===
1. Employee{ID='E001', Name='John Doe', Position='Software Engineer', Salary=$75000.00}
2. Employee{ID='E003', Name='Bob Johnson', Position='Designer', Salary=$65000.00}
3. Employee{ID='E004', Name='Alice Brown', Position='Analyst', Salary=$70000.00}
Total employees: 3

Added: Employee{ID='E005', Name='Charlie Wilson', Position='Tester', Salary=$60000.00}

=== All Employees ===
1. Employee{ID='E001', Name='John Doe', Position='Software Engineer', Salary=$75000.00}
2. Employee{ID='E003', Name='Bob Johnson', Position='Designer', Salary=$65000.00}
3. Employee{ID='E004', Name='Alice Brown', Position='Analyst', Salary=$70000.00}
4. Employee{ID='E005', Name='Charlie Wilson', Position='Tester', Salary=$60000.00}
Total employees: 4
```

Analysis

- **Add**: O(1) - Direct insertion at end
- **Search**: O(n) - May need to check every element
- **Traverse**: O(n) - Must visit every element
- **Delete**: O(n) - Need to find element + shift remaining elements

Exercise 5: Task Management System

Task.java

```java
public class Task {
    private String taskId;
    private String taskName;
    private String status;

    public Task(String taskId, String taskName, String status) {
        this.taskId = taskId;
        this.taskName = taskName;
        this.status = status;
    }

    public String getTaskId() { return taskId; }
    public String getTaskName() { return taskName; }
    public String getStatus() { return status; }

    public void setStatus(String status) { this.status = status; }

    @Override
    public String toString() {
        return String.format("Task{ID='%s', Name='%s', Status='%s'}",
                             taskId, taskName, status);
    }
}
```

TaskNode.java

```java
public class TaskNode {
    Task task;
    TaskNode next;

    public TaskNode(Task task) {
        this.task = task;
        this.next = null;
    }
}
```

TaskLinkedList.java

java

```
public class
TaskLinkedList { private
TaskNode head; private
int size;

public TaskLinkedList() {
    this.head = null;
    this.size = 0;
}

// Add task at the beginning - O(1) time complexity
public void addTask(Task task) {
    TaskNode newNode = new TaskNode(task);
    newNode.next = head;
    head = newNode;
    size++;
    System.out.println("Added: " + task);
}

// Search for task by ID - O(n) time complexity
public Task searchTask(String taskId) {
    System.out.println("Searching for task ID: " + taskId);
    TaskNode current = head;
    int position = 0;

    while (current != null) {
        if (current.task.getTaskId().equals(taskId)) {
            System.out.println("Found at position " + position + ": " + current.task);
            return current.task;
        }
        current = current.next;
        position++;
    }

    System.out.println("Task not found: " + taskId);
    return null;
}

// Traverse all tasks - O(n) time complexity
public void traverseTasks() {
    System.out.println("\n=== All Tasks ===");
    if (head == null) {
        System.out.println("No tasks found");
        return;
    }
```

```
    TaskNode current = head;
    int position = 0;

    while (current != null) {
        System.out.println((position + 1) + ". " + current.task);

        current = current.next;
        position++;
    }
    System.out.println("Total tasks: " + size);
}

// Delete task by ID - O(n) time complexity
public boolean deleteTask(String taskId) {
    System.out.println("Attempting to delete task ID: " + taskId);

    if (head == null) {
        System.out.println("List is empty");
        return false;
    }

    if (head.task.getTaskId().equals(taskId)) {
        Task deletedTask = head.task;
        head = head.next;
        size--;
        System.out.println("Deleted: " + deletedTask);
        return true;
    }

    TaskNode current = head;
    while (current.next != null) {
        if (current.next.task.getTaskId().equals(taskId)) {
            Task deletedTask = current.next.task;
            current.next = current.next.next;
            size--;
            System.out.println("Deleted: " + deletedTask);
            return true;
        }
        current = current.next;
    }

    System.out.println("Task not found for deletion: " + taskId);
    return false;
}

public static void main(String[] args) {
    TaskLinkedList taskList = new TaskLinkedList();

    // Test adding tasks
    taskList.addTask(new Task("T001", "Design Database", "In Progress"));
    taskList.addTask(new Task("T002", "Write Unit Tests", "Pending"));
    taskList.addTask(new Task("T003", "Deploy Application", "Not Started"));
```

```
        taskList.addTask(new Task("T004", "Code Review", "Completed"));

        // Test traversing
        taskList.traverseTasks();

        // Test searching
        taskList.searchTask("T002");
        taskList.searchTask("T999");

        // Test deletion
        taskList.deleteTask("T001");
        taskList.traverseTasks();

        // Test adding after deletion
        taskList.addTask(new Task("T005", "Documentation", "In Progress"));
        taskList.traverseTasks();
    }
}
```

Sample Output

```
Added: Task{ID='T001', Name='Design Database', Status='In Progress'}
Added: Task{ID='T002', Name='Write Unit Tests', Status='Pending'}
Added: Task{ID='T003', Name='Deploy Application', Status='Not Started'}
Added: Task{ID='T004', Name='Code Review', Status='Completed'}

=== All Tasks ===
1. Task{ID='T004', Name='Code Review', Status='Completed'}
2. Task{ID='T003', Name='Deploy Application', Status='Not Started'}
3. Task{ID='T002', Name='Write Unit Tests', Status='Pending'}
4. Task{ID='T001', Name='Design Database', Status='In Progress'}
Total tasks: 4

Searching for task ID: T002
Found at position 2: Task{ID='T002', Name='Write Unit Tests', Status='Pending'}
Searching for task ID: T999
Task not found: T999

Attempting to delete task ID: T001
Deleted: Task{ID='T001', Name='Design Database', Status='In Progress'}

=== All Tasks ===
1. Task{ID='T004', Name='Code Review', Status='Completed'}
2. Task{ID='T003', Name='Deploy Application', Status='Not Started'}
3. Task{ID='T002', Name='Write Unit Tests', Status='Pending'}
Total tasks: 3

Added: Task{ID='T005', Name='Documentation', Status='In Progress'}

=== All Tasks ===
1. Task{ID='T005', Name='Documentation', Status='In Progress'}
2. Task{ID='T004', Name='Code Review', Status='Completed'}
3. Task{ID='T003
```