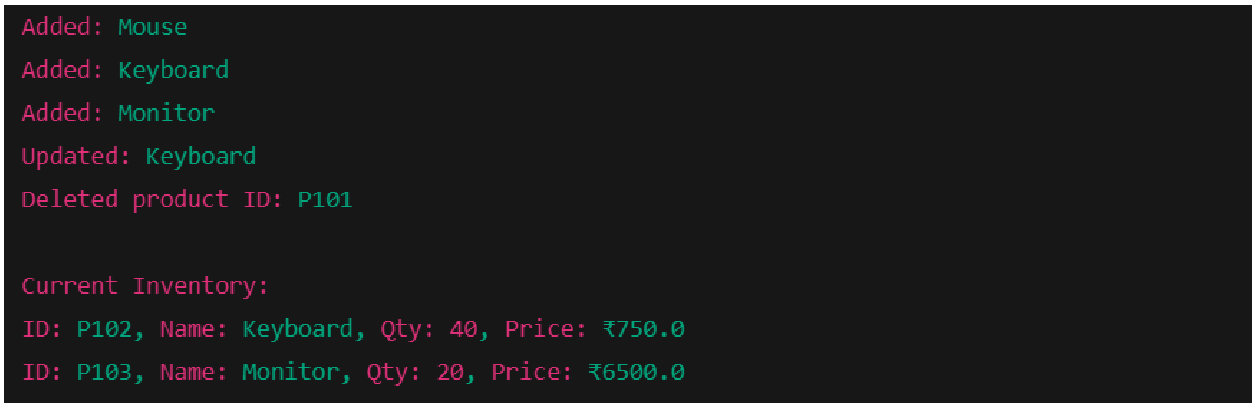
OUTPUTS – SOLID Principles

Exercise 1 :

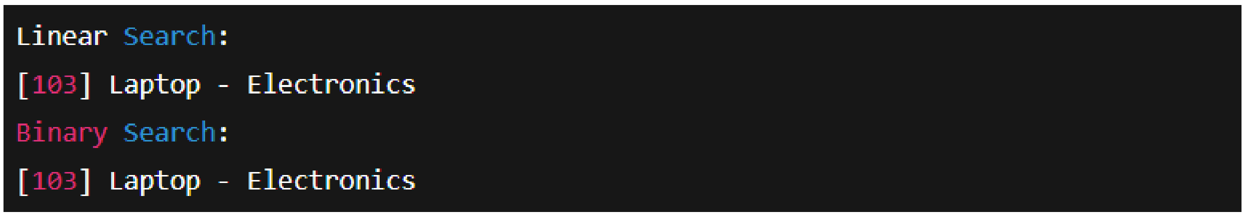


```
Added: Mouse
Added: Keyboard
Added: Monitor
Updated: Keyboard
Deleted product ID: P101

Current Inventory:
ID: P102, Name: Keyboard, Qty: 40, Price: ₹750.0
ID: P103, Name: Monitor, Qty: 20, Price: ₹6500.0
```

Implemented an inventory management system using a *HashMap* to store *Product* objects with *productId* as the key. The *Inventory* class provides methods to add, update, delete, and display products. All operations perform efficiently with an average time complexity of **O(1)** due to the use of a hash map.

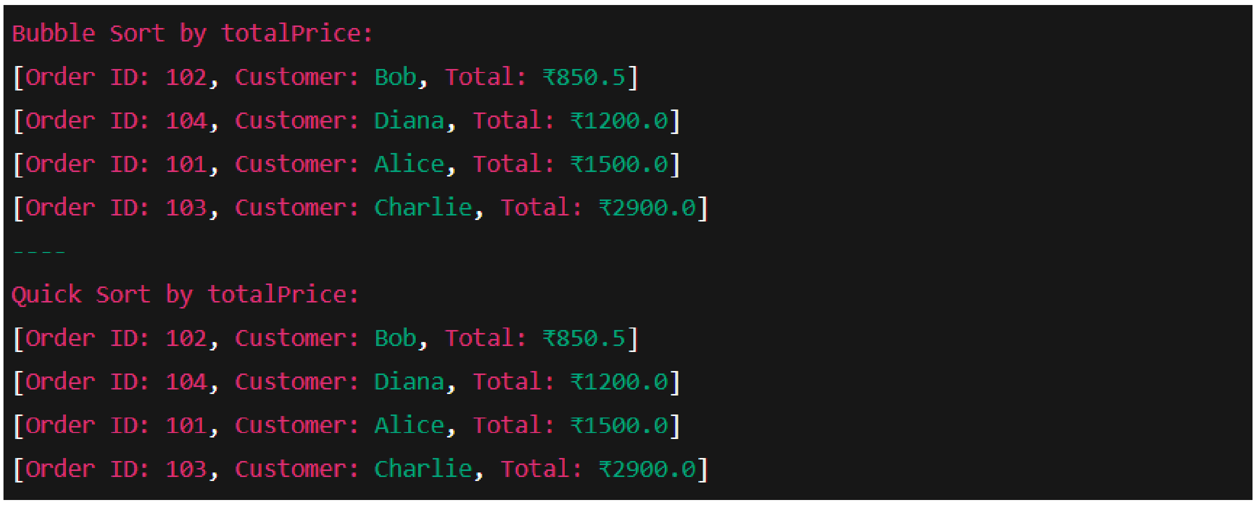
Exercise 2 :



```
Linear Search:
[103] Laptop - Electronics
Binary Search:
[103] Laptop - Electronics
```

Implemented both linear and binary search for *Product* lookup using *productId*. Binary search is more efficient but requires sorted data, making it more suitable for fast e-commerce searches.

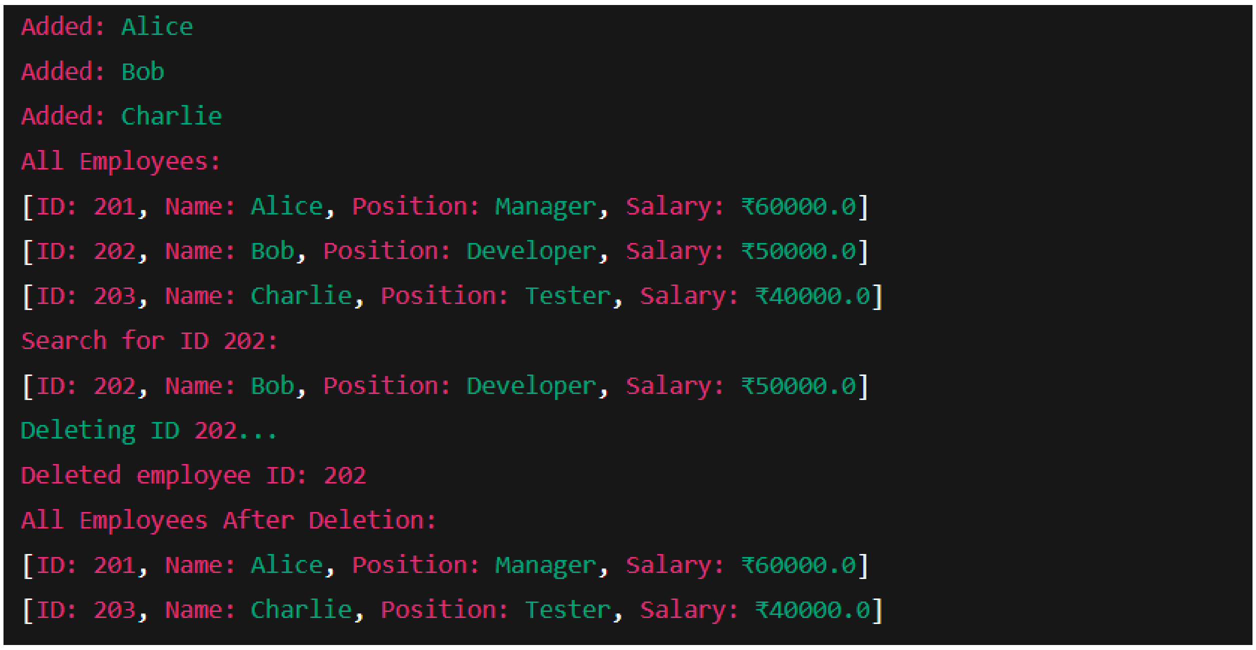
Exercise 3 :



```
Bubble Sort by totalPrice:
[Order ID: 102, Customer: Bob, Total: ₹850.5]
[Order ID: 104, Customer: Diana, Total: ₹1200.0]
[Order ID: 101, Customer: Alice, Total: ₹1500.0]
[Order ID: 103, Customer: Charlie, Total: ₹2900.0]
----
Quick Sort by totalPrice:
[Order ID: 102, Customer: Bob, Total: ₹850.5]
[Order ID: 104, Customer: Diana, Total: ₹1200.0]
[Order ID: 101, Customer: Alice, Total: ₹1500.0]
[Order ID: 103, Customer: Charlie, Total: ₹2900.0]
```

Implemented both Bubble Sort and Quick Sort to sort *Order* objects by *totalPrice*. While Bubble Sort is simple, Quick Sort is significantly faster and more efficient for real-world applications, especially with large numbers of orders.

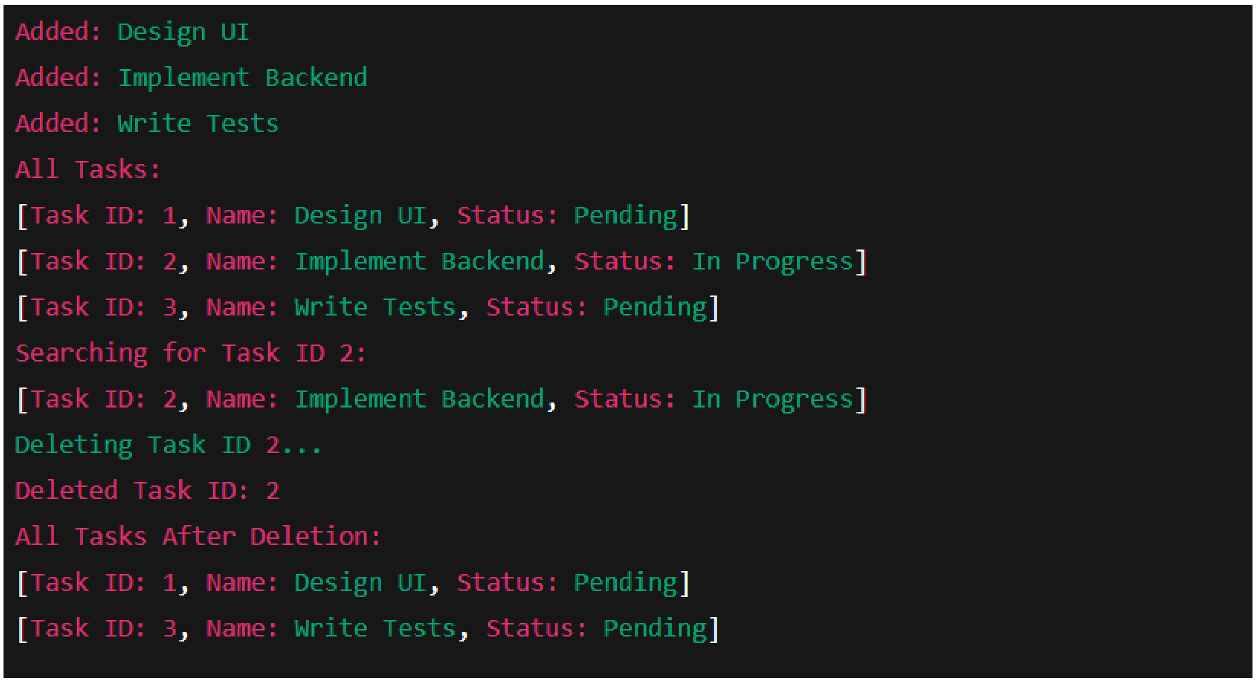
Exercise 4 :



```
Added: Alice
Added: Bob
Added: Charlie
All Employees:
[ID: 201, Name: Alice, Position: Manager, Salary: ₹60000.0]
[ID: 202, Name: Bob, Position: Developer, Salary: ₹50000.0]
[ID: 203, Name: Charlie, Position: Tester, Salary: ₹40000.0]
Search for ID 202:
[ID: 202, Name: Bob, Position: Developer, Salary: ₹50000.0]
Deleting ID 202...
Deleted employee ID: 202
All Employees After Deletion:
[ID: 201, Name: Alice, Position: Manager, Salary: ₹60000.0]
[ID: 203, Name: Charlie, Position: Tester, Salary: ₹40000.0]
```

Implemented an employee management system using arrays. Supported operations include adding, searching, traversing, and deleting employees. Array-based storage gives fast access but is limited in flexibility.

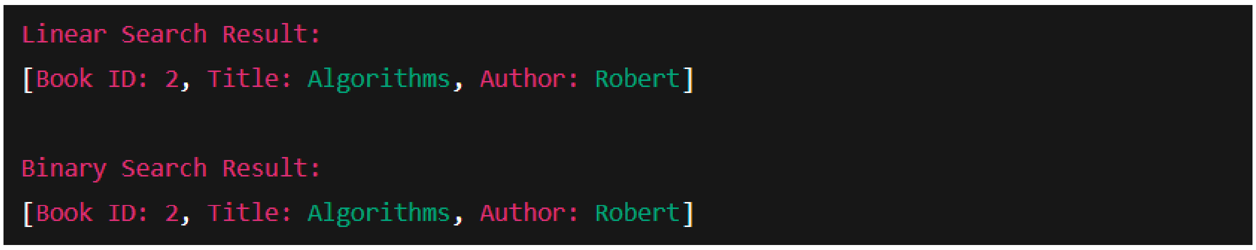
Exercise 5 :



```
Added: Design UI
Added: Implement Backend
Added: Write Tests
All Tasks:
[Task ID: 1, Name: Design UI, Status: Pending]
[Task ID: 2, Name: Implement Backend, Status: In Progress]
[Task ID: 3, Name: Write Tests, Status: Pending]
Searching for Task ID 2:
[Task ID: 2, Name: Implement Backend, Status: In Progress]
Deleting Task ID 2...
Deleted Task ID: 2
All Tasks After Deletion:
[Task ID: 1, Name: Design UI, Status: Pending]
[Task ID: 3, Name: Write Tests, Status: Pending]
```

Used a singly linked list to manage tasks dynamically. Supported adding, searching, traversing, and deleting tasks. Linked lists are flexible and efficient for dynamic task systems, especially when frequent modifications are needed.

Exercise 6 :

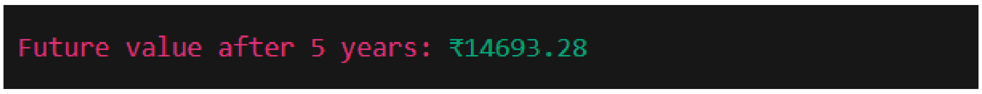


```
Linear Search Result:
[Book ID: 2, Title: Algorithms, Author: Robert]

Binary Search Result:
[Book ID: 2, Title: Algorithms, Author: Robert]
```

Created a library management system that supports searching books by title using both linear and binary search. Binary search is faster but requires sorted data, while linear search is simpler and works on any data.

Exercise 7:



```
Future value after 5 years: ₹14693.28
```

Implemented a recursive method to forecast future financial values based on past growth. The approach uses recursion to apply compound interest year by year. It’s simple and effective for small time frames, but should be optimized or made iterative for large values of n to avoid performance issues.