**Exercise 1: Inventory Management System**

import java.util.HashMap;

import java.util.Map;

class Product {

int productId;

String productName;

int quantity;

double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

}

class InventoryManagement {

private Map<Integer, Product> inventory;

public InventoryManagement() {

inventory = new HashMap<>();

}

public void addProduct(Product product) {

inventory.put(product.productId, product);

}

public void updateProduct(Product product) {

inventory.put(product.productId, product);

}

public void deleteProduct(int productId) {

inventory.remove(productId);

}

public Product getProduct(int productId) {

return inventory.get(productId);

}

public static void main(String[] args) {

InventoryManagement inventory = new InventoryManagement();

Product product1 = new Product(1, "Laptop", 10, 1000.0);

Product product2 = new Product(2, "Mouse", 50, 20.0);

inventory.addProduct(product1);

inventory.addProduct(product2);

System.out.println(inventory.getProduct(1).productName);

inventory.updateProduct(new Product(1, "Laptop", 8, 950.0));

System.out.println(inventory.getProduct(1).quantity);

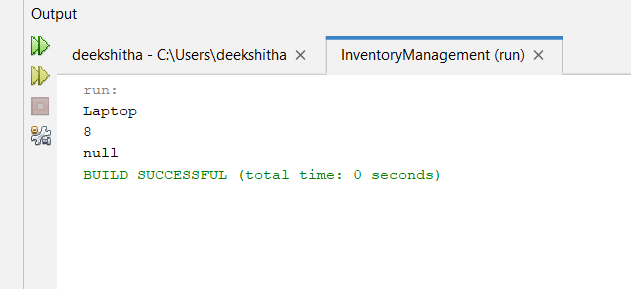
inventory.deleteProduct(2);

System.out.println(inventory.getProduct(2));

}

}

**Output:**

****

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

import java.util.\*;

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

}

class Ecommerce\_Search\_Function {

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

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

if (products[i].productName.equals(name)) {

return i;

}

}

return -1;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int comparison = products[mid].productName.compareTo(name);

if (comparison == 0) {

return mid;

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Mouse", "Electronics"),

new Product(3, "Keyboard", "Electronics"),

new Product(4, "Monitor", "Electronics")

};

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

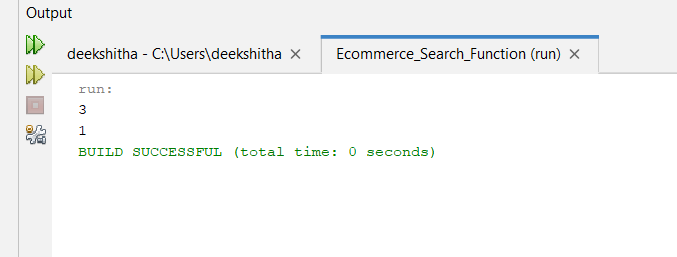
System.out.println(linearSearch(products, "Mouse"));

System.out.println(binarySearch(products, "Laptop"));

}

}

**Output:**

****

**Exercise 3: Sorting Customer Orders**

class Order {

int orderId;

String customerName;

double totalPrice;

Order(int orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

}

class SortingOrders {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

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

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

double pivot = orders[high].totalPrice;

int i = (low - 1);

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

if (orders[j].totalPrice <= 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 main(String[] args) {

Order[] orders = {

new Order(1, "Alice", 250.0),

new Order(2, "Bob", 150.0),

new Order(3, "Charlie", 300.0),

new Order(4, "Dave", 100.0)

};

bubbleSort(orders);

for (Order order : orders) {

System.out.println(order.customerName + " - " + order.totalPrice);

}

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

for (Order order : orders) {

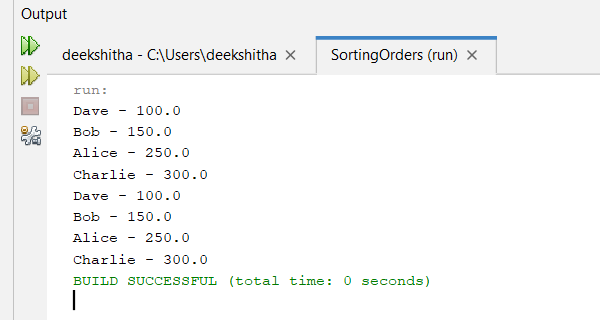
System.out.println(order.customerName + " - " + order.totalPrice);

}

}

}

**Output:**



**Exercise 4: Employee Management System**

class Employee {

int employeeId;

String name;

String position;

double salary;

Employee(int employeeId, String name, String position, double salary) {

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

}

class EmployeeManagement {

private Employee[] employees;

private int size;

public EmployeeManagement(int capacity) {

employees = new Employee[capacity];

size = 0;

}

public void addEmployee(Employee employee) {

if (size < employees.length) {

employees[size++] = employee;

}

}

public Employee searchEmployee(int employeeId) {

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

if (employees[i].employeeId == employeeId) {

return employees[i];

}

}

return null;

}

public void traverseEmployees() {

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

System.out.println(employees[i].name + " - " + employees[i].position);

}

}

public void deleteEmployee(int employeeId) {

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

if (employees[i].employeeId == employeeId) {

employees[i] = employees[--size];

employees[size] = null;

break;

}

}

}

public static void main(String[] args) {

EmployeeManagement management = new EmployeeManagement(10);

Employee emp1 = new Employee(1, "Alice", "Manager", 50000.0);

Employee emp2 = new Employee(2, "Bob", "Developer", 40000.0);

management.addEmployee(emp1);

management.addEmployee(emp2);

management.traverseEmployees();

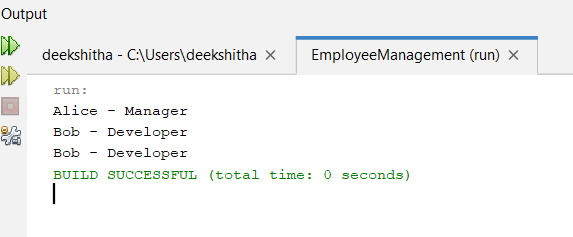
management.deleteEmployee(1);

management.traverseEmployees();

}

}

**Output:**



**Exercise 5: Task Management System**

class Task {

int taskId;

String taskName;

String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

}

class Node {

Task task;

Node next;

Node(Task task) {

this.task = task;

this.next = null;

}

}

class TaskManagement {

private Node head;

public TaskManagement() {

head = null;

}

public void addTask(Task task) {

Node newNode = new Node(task);

newNode.next = head;

head = newNode;

}

public Task searchTask(int taskId) {

Node current = head;

while (current != null) {

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

return current.task;

}

current = current.next;

}

return null;

}

public void traverseTasks() {

Node current = head;

while (current != null) {

System.out.println(current.task.taskName + " - " + current.task.status);

current = current.next;

}

}

public void deleteTask(int taskId) {

Node current = head;

Node prev = null;

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

prev = current;

current = current.next;

}

if (current != null) {

if (prev != null) {

prev.next = current.next;

} else {

head = current.next;

}

}

}

public static void main(String[] args) {

TaskManagement management = new TaskManagement();

Task task1 = new Task(1, "Design Database", "Pending");

Task task2 = new Task(2, "Develop API", "In Progress");

management.addTask(task1);

management.addTask(task2);

management.traverseTasks();

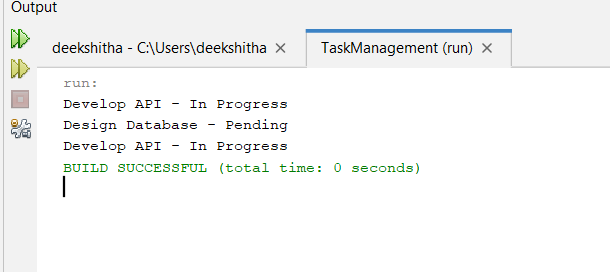
management.deleteTask(1);

management.traverseTasks();

}

}

**Output:**



**Exercise 6: Library Management System**

class Book {

int bookId;

String title;

String author;

Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

}

class LibraryManagement {

public static int linearSearch(Book[] books, String title) {

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

if (books[i].title.equals(title)) {

return i;

}

}

return -1;

}

public static int binarySearch(Book[] books, String title) {

int left = 0;

int right = books.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int comparison = books[mid].title.compareTo(title);

if (comparison == 0) {

return mid;

} else if (comparison < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

public static void main(String[] args) {

Book[] books = {

new Book(1, "1984", "George Orwell"),

new Book(2, "The Great Gatsby", "F. Scott Fitzgerald"),

new Book(3, "To Kill a Mockingbird", "Harper Lee")

};

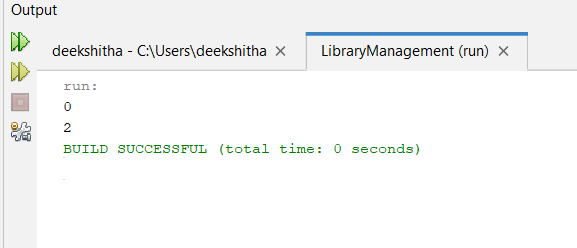
System.out.println(linearSearch(books, "1984"));

System.out.println(binarySearch(books, "To Kill a Mockingbird"));

}

}

**Output:**



**Exercise 7: Financial Forecasting**

public class FinancialForecasting {

public static double forecast(double[] pastData, int year) {

if (year < pastData.length) {

return pastData[year];

}

double last = forecast(pastData, year - 1);

double secondLast = forecast(pastData, year - 2);

double growthRate = last / secondLast;

return last \* growthRate;

}

public static void main(String[] args) {

double[] pastData = {1000.0, 1100.0, 1210.0};

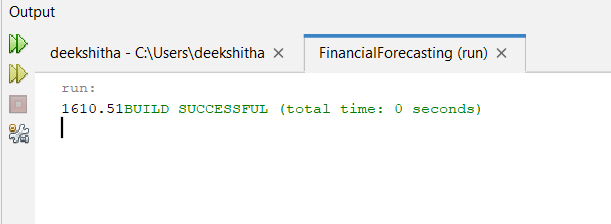
int targetYear = 5;

System.out.printf("%.2f", forecast(pastData, targetYear));

}

}

**Output:**

****