Algorithms Data Structures

**Exercise 1: Inventory Management System**

**CODE**

import java.util.\*;

class Productnew{

    private int productId;

    private String productName;

    private String category;

    private static Productnew products[]=new Productnew[100];

    private static int index=0;

    public Productnew(int productId,String productName,String category)

    {

        this.productId=productId;

        this.productName=productName;

        this.category=category;

    }

    public static void addProduct(Productnew product)

    {

        products[index]=product;

        index++;

    }

    public int getProductId()

    {

        return productId;

    }

    public static Productnew linearSearch(int productId)

    {

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

        {

            if(products[i].productId==productId)

            return products[i];

        }

        return null;

    }

    public static Productnew binarySearch(int productId)

    {

        int low=0;

        int high=index-1;

        while(low<=high)

        {

            int mid=(low+high)/2;

            if(products[mid].productId==productId)

            return products[mid];

            else if(productId>mid)

            {

                low=mid+1;

            }

            else

            {

                high=mid-1;

            }

        }

        return null;

    }

    public static Productnew[] getAllProducts()

    {

        return products;

    }

    @Override

    public String toString()

    {

        return "Id:   "+productId+"    Name :"+productName+"        category :"+category;

    }

    public static int getIndex()

    {

        return index;

    }

}

public class ECP {

    public static void main(String[] args) {

        Productnew p4=new Productnew(4, "Tea", "Dairy Product");

        Productnew p2=new Productnew(2, "Paneer","Dairy Product" );

        Productnew p1=new Productnew(1, "Milk", "Dairy Product");

        Productnew p3=new Productnew(3, "Coffee", "Dairy Product");

        System.out.println(p1);

        System.out.println(p2);

        System.out.println(p3);

        System.out.println(p4);

        Productnew.addProduct(p4);

        Productnew.addProduct(p2);

        Productnew.addProduct(p3);

        Productnew.addProduct(p1);

        Arrays.sort(Productnew.getAllProducts(),0,Productnew.getIndex(),(a1,a2)->Integer.compare(a1.getProductId(), a2.getProductId()));//

        //Earlier the sort algo was working on the empty blocs and default values were read as i have used 4 out of 100 space of the array,specify the range for sorting

        System.out.println("\n"+"-----AFTER SORTING-------"+"\n");

        for (int i=0;i<Productnew.getIndex();i++) {

            System.out.println(Productnew.getAllProducts()[i]);

        }

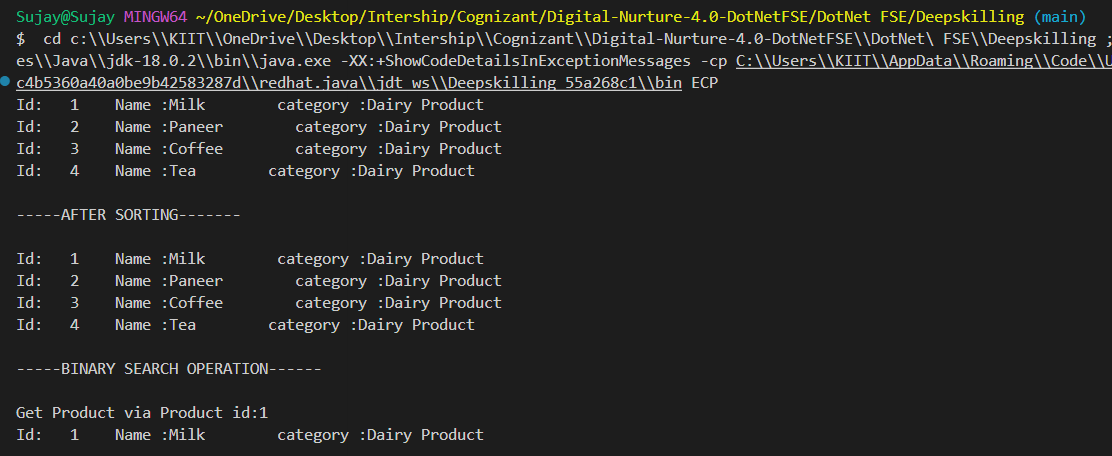
        System.out.println("\n"+"-----BINARY SEARCH OPERATION------"+"\n");

        System.out.println("Get Product via Product id: "+1+"\n"+Productnew.binarySearch(1));

    }

}

**OUTPUT**



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

**CODE**

import java.util.\*;

class Product {

    private int productId, quantity;

    private String productName;

    private double price;

    private static Map<Integer, Product> store = new HashMap<>();

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

        this.productId = productId;

        this.price = price;

        this.productName = productName;

        this.quantity = quantity;

    }

    public static String addProduct(Product newProduct) {

        int oldStoreSize = store.size();

        store.put(newProduct.productId, newProduct);

        int newStoreSize = store.size();

        return newStoreSize > oldStoreSize ? "Product Added Successfully" : "Failed Add operation";

    }

    public static Product removeProduct(int productId) {

        return store.remove(productId);

    }

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

        Product updatedProduct = store.get(productId);

        if (updatedProduct == null) return null;

        if (price >= 0) updatedProduct.price = price;

        if (quantity >= 0) updatedProduct.quantity = quantity;

        if (productName != null && !productName.equals(updatedProduct.productName)) {

            updatedProduct.productName = productName;

        }

        return updatedProduct;

    }

    public String getProductName() { return productName; }

    public double getProductPrice() { return price; }

    public int getProductQuantity() { return quantity; }

    @Override//overide the pre-defined toString method of "Object class--root of class hierchy"

    public String toString()//default return value is class name with hashcode-which a unique identifier

    {

        return "ID: " + productId + ", Name: " + productName + ", Price: " + price + ", Qty: " + quantity;

       // default return value Inventory@2c7b84de

    }

}

public class IMS {

    public static void main(String[] args) {

        Product product = new Product(1, "rice", 10000, 20.23);

        System.out.println(product);

        String status = Product.addProduct(product);

        System.out.println(status);

        Product updated = Product.updateProduct(1, "basmati rice", -1, -1); // -1 for no update

        if (updated != null) {

            System.out.println(updated);

        } else {

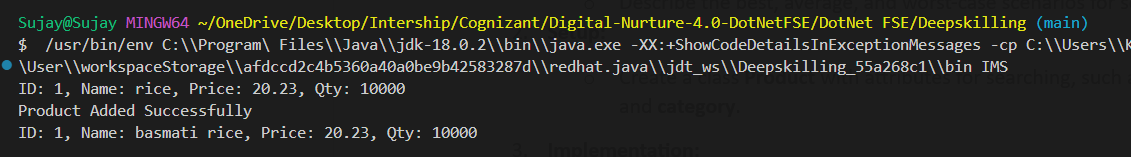
            System.out.println("Product not found for update.");

        }

    }

}

**OUTPUT**

****

**Exercise 3: Sorting Customer Orders**

**CODE**

class Order{

    private int orderId;

    private String customerName;

    private double totalPrice;

    private static Order orders[]=new Order[10];

    public static int index=0;

    Order(int orderId,String customerName,double totalPrice)

    {

        this.orderId=orderId;

        this.customerName=customerName;

        this.totalPrice=totalPrice;

    }

    public static void bubbleSort()

    {

       for(int i=0;i<index;i++)

       {

        for(int j=0;j<index-i-1;j++)

        {

            if(orders[j].totalPrice>orders[j+1].totalPrice)

            {

              Order tempOd=orders[j];

              orders[j]=orders[j+1];

              orders[j+1]=tempOd;

            }

        }

       }

    }

    public static void addOrder(Order order)

    {

        orders[index]=order;

        index++;

    }

    public static int partition(Order orders[],int low,int high)

    {

        int i=low-1;

        Order pivot=orders[high];

        for(int j=low;j<=high-1;j++)

        {

            if(orders[j].totalPrice<pivot.totalPrice)

            {

                i++;

                swap(orders, i, j);

            }

        }

        swap(orders, i+1, high);

        return i+1;

    }

    public static void swap(Order orders[],int i,int j)

    {

        Order temp=orders[i];

        orders[i]=orders[j];

        orders[j]=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);

        }

    }

    public String getCustomerName()

    {

        return customerName;

    }

    public double getPrice()

    {

        return totalPrice;

    }

    public int getOrderId()

    {

        return orderId;

    }

    public static Order[] getAllOrders()

    {

        return orders;

    }

    public static int getIndex()

    {

        return index;

    }

    @Override

    public String toString()

    {

        return "ID : "+orderId+" Customer Name :"+customerName+" Total Price : "+totalPrice;

    }

}

public class SCO {

    public static void main(String[] args) {

        Order o1=new Order(1, "Sujay", 1220.98);

        Order o2=new Order(2, "Shreya", 1200.98);

        Order o3=new Order(3, "Rishav", 3220.98);

        Order o4=new Order(4, "John", 1000.98);

        Order o5=new Order(5, "Noah", 1120.98);

        Order.addOrder(o1);

        Order.addOrder(o2);

        Order.addOrder(o3);

        Order.addOrder(o4);

        Order.addOrder(o5);

        for(int i=0;i<Order.getIndex();i++)

        {

           System.out.println(Order.getAllOrders()[i]);

        }

        System.out.println("\n"+"-----After Sorting based on Total Price--------"+"\n");

        Order.quickSort(Order.getAllOrders(),0,Order.getIndex()-1);

        for(int i=0;i<Order.getIndex();i++)

        {

           System.out.println(Order.getAllOrders()[i]);

        }

    }

}

**OUTPUT**

