1) Implementa el siguiente método que ordene el arreglo de productos de forma no descendente de acuerdo al precio del producto, utilizando el algoritmo BucketSort como se vio en clase.

static void ordenarPorPrecio(Producto[] productos)

Especificaciones del algoritmo BucketSort:

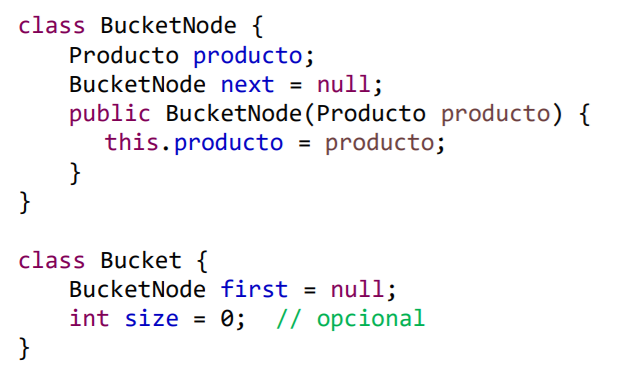
a) Se deberán crear N buckets (arreglo o lista).

b) Cada bucket almacenará una lista enlazada de productos.

c) El número de bucket que se asignará a un producto será de acuerdo a su precio, de forma que a un producto cuyo precio sea el mínimo posible se le asigne el bucket 0, y a un producto cuyo precio sea el máximo posible se le asigna el bucket N – 1.

d) Si el bucket no está vacío, el producto deberá insertarse en una posición de la lista enlazada tal que ésta se mantenga ordenada de forma no descendente por precio, y que garantice que el algoritmo sea estable.

e) Se sugieren las siguientes estructuras para crear los buckets:



import java.util.Arrays;

class Producto {

    private String nombre;

    private double precio;

    private int existencias;

    public static final double PRECIO\_MIN = 3;

    public static final double PRECIO\_MAX = 2000;

    public static final int EXISTENCIAS\_MIN = 5;

    public static final int EXISTENCIAS\_MAX = 50;

    private static int productId = 0;

    public Producto(String n, double p, int e) {

        this.nombre = n;

        this.precio = p;

        this.existencias = e;

    }

    public Producto() {

        this.nombre = String.format("Producto %3d", ++ productId);

        this.precio = PRECIO\_MIN + (PRECIO\_MAX - PRECIO\_MIN) \* Math.random();

        this.existencias = EXISTENCIAS\_MIN + (int) ((EXISTENCIAS\_MAX - EXISTENCIAS\_MIN + 1) \* Math.random());

    }

    public String getNombre() {

        return nombre;

    }

    public double getPrecio() {

        return precio;

    }

    public int getExistencias() {

        return existencias;

    }

    public String toString() {

        return String.format("(%s, $%7.2f, %2d)", nombre, precio, existencias);

    }

}

public class Examen2a{

    static class BucketNode {

        Producto producto;

        BucketNode next = null;

        BucketNode previous = null;

        public BucketNode(Producto producto) {

            this.producto = producto;

        }

    }

    static class Bucket {

        BucketNode first = null;

        BucketNode last = null;

        int size = 0; // opcional

    }

    static void insertion(Bucket bucket){

        BucketNode pivotBucket = null;

        Producto pivotProduct = null;

        BucketNode bucketj = null;

        BucketNode bucketj\_Plus1 = null;

        if(bucket != null){

            if(bucket.first != null){

                pivotBucket = bucket.first.next;

                pivotProduct = (pivotBucket != null) ? pivotBucket.producto : null;

            }

            while(pivotBucket != null){

                bucketj = pivotBucket.previous;

                bucketj\_Plus1 = bucketj.next;

                while(bucketj != null && bucketj.producto.getPrecio() > pivotProduct.getPrecio()){

                    bucketj.next.producto = bucketj.producto;

                    bucketj\_Plus1 = bucketj;

                    bucketj = bucketj.previous;

                }

                bucketj\_Plus1.producto = pivotProduct;

                pivotBucket = pivotBucket.next;

                pivotProduct = (pivotBucket != null) ? pivotBucket.producto : null;

            }

        }

    }

    static void ordenarPorPrecio(Producto[] productos){

        Bucket bucketList[] = new Bucket[(int)(Producto.PRECIO\_MAX-Producto.PRECIO\_MIN+1)];

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

            bucketList[i] = new Bucket();

        }

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

            BucketNode nodo = new BucketNode(productos[i]);

            int code = (int)(nodo.producto.getPrecio()-Producto.PRECIO\_MIN);

            if(bucketList[code].size == 0){

                bucketList[code].first = nodo;

                bucketList[code].last = nodo;

            }else{

                nodo.previous = bucketList[code].last;

                bucketList[code].last.next = nodo;

                bucketList[code].last = nodo;

            }

            bucketList[code].size++;

        }

        int index = 0;

        for(Bucket bucket : bucketList){

            insertion(bucket);

            while(bucket.first != null){

                productos[index] = bucket.first.producto;

                bucket.first = bucket.first.next;

                index++;

            }

        }

    }

    static boolean isProductSorted(Producto productos[]){

        if(productos != null){

            double pastPrice = productos[0].getPrecio();

            for(int i = 1; i<productos.length;i++){

                if(pastPrice > productos[i].getPrecio()){

                    return false;

                }

            }

            return true;

        }

        return false;

    }

    public static void main(String[] args){

        /\*Producto p1 = new Producto("Producto 1", 22, 32);

        Producto p2 = new Producto("Producto 2", 10, 6);

        Producto p3 = new Producto("Producto 3", 15, 6);

        Producto p4 = new Producto("Producto 4", 200, 30);

        Producto p5 = new Producto("Producto 5", 190, 13);

        Producto p6 = new Producto("Producto 6", 22, 25);

        Producto productList[] = {p1,p2,p3,p4,p5,p6};

        ordenarPorPrecio(productList);

        System.out.println(Arrays.toString(productList));

        System.out.println(isProductSorted(productList));

        \*/

        for(int n = 100\_000; n<=500\_000; n+=100\_000){

            Producto productList[] = new Producto[n];

            for(int i = 0; i<n; i++){

                productList[i] = new Producto();

            }

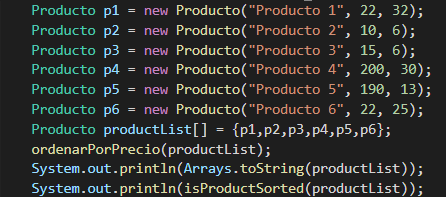
            ordenarPorPrecio(productList);

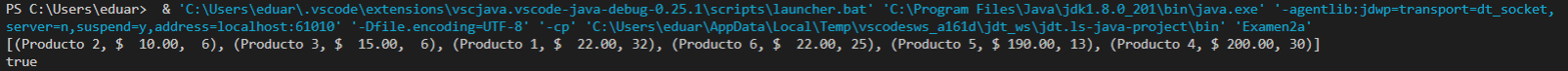
            System.out.println(isProductSorted(productList));

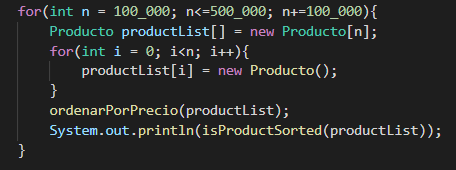
        }

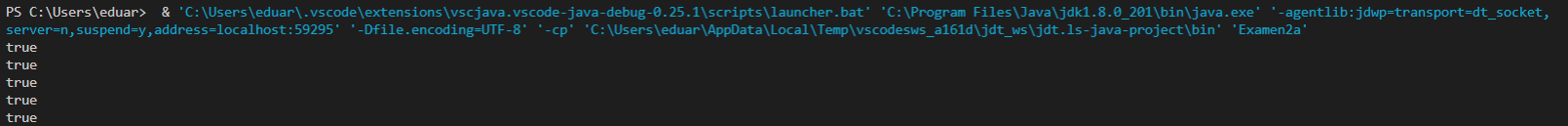
    }

}









2) Implementa el siguiente método que ordene el arreglo de productos de forma no descendente de acuerdo a las existencias del producto, utilizando el algoritmo Conteo como se vio en clase.

static void ordenarPorExistencias(Producto[] productos)

Especificaciones del algoritmo Conteo:

a) El tamaño del arreglo de conteos estará en función de la diferencia entre el número mínimo y máximo de existencias posibles.

b) Recuerda que el algoritmo debe implementarse de manera que sea estable. Para ello, tendrás que crear un arreglo temporal de productos, pero el arreglo recibido deberá contener el ordenamiento definitivo.

import java.util.Arrays;

class Producto {

    private String nombre;

    private double precio;

    private int existencias;

    public static final double PRECIO\_MIN = 3;

    public static final double PRECIO\_MAX = 2000;

    public static final int EXISTENCIAS\_MIN = 5;

    public static final int EXISTENCIAS\_MAX = 50;

    private static int productId = 0;

    public Producto(String n, double p, int e) {

        this.nombre = n;

        this.precio = p;

        this.existencias = e;

    }

    public Producto() {

        this.nombre = String.format("Producto %3d", ++ productId);

        this.precio = PRECIO\_MIN + (PRECIO\_MAX - PRECIO\_MIN) \* Math.random();

        this.existencias = EXISTENCIAS\_MIN + (int) ((EXISTENCIAS\_MAX - EXISTENCIAS\_MIN + 1) \* Math.random());

    }

    public String getNombre() {

        return nombre;

    }

    public double getPrecio() {

        return precio;

    }

    public int getExistencias() {

        return existencias;

    }

    public String toString() {

        return String.format("(%s, $%7.2f, %2d)", nombre, precio, existencias);

    }

}

public class Examen2b {

    static void ordenarPorExistencias(Producto[] productos){

        int conteo[] = new int[Producto.EXISTENCIAS\_MAX-Producto.EXISTENCIAS\_MIN+1];

        for(Producto producto:productos){

            conteo[producto.getExistencias()-Producto.EXISTENCIAS\_MIN]++;

        }

        for(int i = 1; i< conteo.length;i++){

            conteo[i] += conteo[i-1];

        }

        Producto productosTemp[] = new Producto[productos.length];

        for(int i = productos.length-1 ; i>=0 ; i--){

            Producto x = productos[i];

            int xnum = x.getExistencias()-Producto.EXISTENCIAS\_MIN;

            int j = conteo[xnum] - 1;

            productosTemp[j] = x;

            conteo[xnum]--;

        }

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

            productos[i] = productosTemp[i];

        }

    }

    static boolean isProductSorted(Producto productos[]){

        if(productos != null){

            double pastPrice = productos[0].getExistencias();

            for(int i = 1; i<productos.length;i++){

                if(pastPrice > productos[i].getExistencias()){

                    return false;

                }

            }

            return true;

        }

        return false;

    }

    public static void main(String[] args){

        Producto p1 = new Producto("Producto 1", 22, 32);

        Producto p2 = new Producto("Producto 2", 10, 6);

        Producto p3 = new Producto("Producto 3", 15, 6);

        Producto p4 = new Producto("Producto 4", 200, 30);

        Producto p5 = new Producto("Producto 5", 190, 13);

        Producto p6 = new Producto("Producto 6", 22, 25);

        Producto productList[] = {p1,p2,p3,p4,p5,p6};

        ordenarPorExistencias(productList);

        System.out.println(Arrays.toString(productList));

        System.out.println(isProductSorted(productList));

        /\*

        for(int n = 100\_000; n<=500\_000; n+=100\_000){

            Producto productList[] = new Producto[n];

            for(int i = 0; i<n; i++){

                productList[i] = new Producto();

            }

            ordenarPorExistencias(productList);

            System.out.println(isProductSorted(productList));

        }\*/

    }

}

