МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«Харківський політехнічний інститут»

Кафедра «Програмної інженерії та інформаційних технологій управління»

Iндивiдуальне домашне завдання

з предмету «Кросплатформенне програмування»

Виконав:

Студент групи КН-36а

Кулик В.В.

Перевірив:  
 Марченко I.I.

Харків 2018

|  |  |
| --- | --- |
| AddNullException.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  class AddNullException : Exception  {  public AddNullException(string message) : base(message)  {  }  public AddNullException(string message, Exception ex)  : base(message, ex)  {  }  public AddNullException() : base()  {  }  }  }  ArrayListIterator.cs  using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  class ArrayListIterator<T> : IEnumerator<T> where T : IName<T>  {  Container<T> parent;  private int position;  public ArrayListIterator(Container<T> parent)  {  this.parent = parent;  position = -1;  }  public bool MoveNext()  {  if (position != parent.Repository.Length)  {  position++;  }  return position < parent.Repository.Length;  }  object IEnumerator.Current  {  get  {  if (position == -1 || position == parent.Repository.Length)  {  throw new InvalidOperationException();  }  return parent[position];  }  }  T IEnumerator<T>.Current  {  get  {  if (position == -1 || position == parent.Repository.Length)  {  throw new InvalidOperationException();  }  return parent[position];  }  }  public void Reset()  {  position = -1;  }  public void Dispose() { }  }  }  Container.cs  using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Runtime.Serialization.Formatters.Binary;  namespace Music  {  [Serializable]  class Container<T> : IEnumerable<T> where T : IName<T>  {  private T[] repository = new T[0];  public T[] Repository  {  get  {  return repository;  }  set  {  repository = value;  }  }  public decimal Sum { get; set; }  public void PriceChanging(decimal price)  {  Sum -= price;  }  public void PriceChanged(decimal price)  {  Sum += price;  }  public T this[int index]  {  get  {  if (index < 0 || index > repository.Length)  {  throw new OutOfRangeException("The index is outside the array.");  }  return repository[index];  }  set  {  repository[index] = value;  }  }  public T this[string name]  {  get  {  foreach (T p in repository)  {  if (p.Name == name)  {  return p;  }  }  return default(T);  }  set  {  for (int i = 0; i < repository.Length; i++)  {  if (repository[i].Name == name)  {  repository[i] = value;  }  }  }  }  public void Add(params T[] arg)  {  foreach (T p in arg)  {  if (p == null)  {  throw new AddNullException("A null object was found in the array.");  }  }  foreach (T a in arg)  {  (a as Product).Editing += PriceChanging;  (a as Product).Edited += PriceChanged;  Sum += (a as Product).Price;  int length = repository.Length + 1;  T[] newRepository = new T[length];  for (var i = 0; i < repository.Length; i++)  {  if (repository[i] != null) newRepository[i] = repository[i];  }  newRepository[length - 1] = a;  repository = newRepository;  }  }  public void Remove(int index)  {  (this[index] as Product).Editing -= PriceChanging;  (this[index] as Product).Edited -= PriceChanged;  Sum -= (this[index] as Product).Price;  if (index >= repository.Length || index < 0)  {  throw new OutOfRangeException("Removal is not possible. The index is outside the array.");  }  T[] newRepository = new T[repository.Length - 1];  for (int i = 0; i < repository.Length; i++)  {  if (i < index)  {  newRepository[i] = repository[i];  }  else if (i > index)  {  newRepository[i - 1] = repository[i];  }  }  repository = newRepository;  }  public void Sort()  {  //Sort((x, y) => { return x.CompareTo(y); });  for (int i = 0; i < repository.Length; i++)  {  for (int j = i + 1; j < repository.Length; j++)  {  if ((repository[i] as Product).Price > (repository[i] as Product).Price)  {  var temp = repository[i];  repository[i] = repository[j];  repository[j] = temp;  }  }  }  }  public void Sort(Compare<T> del)  {  //Array.Sort(repository, (T x, T y) => { return del(x, y); });  for (int i = 0; i < repository.Length; i++)  {  for (int j = i + 1; j < repository.Length; j++)  {  if (del(repository[i], repository[j]) > 0)  {  var obj = repository[i];  repository[i] = repository[j];  repository[j] = obj;  }  }  }  }  public T Find(Find<T> del)  {  foreach (T obj in this)  {  if (del(obj))  {  return obj;  }  }  return default(T);  }  public T[] FindAll(Find<T> del)  {  T[] main = new T[0];  foreach (T obj in this)  {  if (del(obj))  {  T[] temp = new T[main.Length + 1];  for (var i = 0; i < main.Length; i++)  {  temp[i] = main[i];  }  temp[temp.Length - 1] = obj;  main = temp;  }  }  return main;  }  public override string ToString()  {  StringBuilder result = new StringBuilder("");  foreach (T a in repository)  {  result.Append(a);  result.Append("\n");  }  return result.ToString();  }  public IEnumerable Reverse()  {  for (int i = repository.Length - 1; i >= 0; i--)  {  yield return repository[i];  }  }  public IEnumerable FindByString(string s)  {  for (int i = 0; i < repository.Length; i++)  {  if (repository[i].Name.Contains(s))  {  yield return repository[i];  }  }  }  public IEnumerable InOrder()  {  T[] temp = new T[repository.Length];    //temp = repository.OrderBy(x => x.Name).ToArray();  for (int i = 0; i < repository.Length; i++)  {  temp[i] = repository[i];  }  for (int i = 0; i < temp.Length; i++)  {  for (int j = i + 1; j < temp.Length; j++)  {  if ((temp[i] as Product).Price > (temp[j] as Product).Price)  {  var obj = temp[i];  temp[i] = temp[j];  temp[j] = obj;  }  }  }  for (int i = 0; i < temp.Length; i++)  {  yield return temp[i];  }  }  public IEnumerator<T> GetEnumerator()  {  return new ArrayListIterator<T>(this);  }  IEnumerator IEnumerable.GetEnumerator()  {  return GetEnumerator();  }  }  }  LinkedListIterator.cs  using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  class LinkedListIterator<T> : IEnumerator<T> where T : IName<T>  {  LinkedListContainer<T> parent;  private int position;  public LinkedListIterator(LinkedListContainer<T> parent)  {  this.parent = parent;  position = -1;  }  public bool MoveNext()  {  if (position != parent.Count)  {  position++;  }  return position < parent.Count;  }  object IEnumerator.Current  {  get  {  if (position == -1 || position == parent.Count) throw new InvalidOperationException();  return parent[position];  }  }  T IEnumerator<T>.Current  {  get  {  if (position == -1 || position == parent.Count) throw new InvalidOperationException();  return parent[position];  }  }  public void Reset()  {  position = -1;  }  public void Dispose() { }  }  }  LinkedListNode.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  [Serializable]  class LinkedListNode<T> where T: IName<T>  {  public LinkedListNode<T> Next { get; set; }  public LinkedListNode<T> Previous { get; set; }  public T Data { get; set; }  }  }  NothingFoundException.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  class NothingFoundException : Exception  {  public NothingFoundException(string message) : base(message)  {  }  public NothingFoundException(string message, Exception ex)  : base(message, ex)  {  }  public NothingFoundException() : base()  {  }  }  }  OutOfRange.Exception.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  class OutOfRangeException : Exception  {  public OutOfRangeException(string message) : base(message)  {  }  public OutOfRangeException(string message, Exception ex)  : base(message, ex)  {  }  public OutOfRangeException() : base()  {  }  }  }  Product.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Music  {  [Serializable]  public class Product : IName<Product>, IName  {  public event PriceChangingHadler Editing;  public event PriceChangingHadler Edited;  protected string name;  protected decimal price;  public Product()  {  this.Name = "";  this.Price = 0;  }  public Product(string name, decimal price)  {  this.Name = name;  this.Price = price;  }  public string Name  {  get  {  return name;  }  set  {  name = value;  }  }  public decimal Price  {  get  {  return price;  }  set  {  if (price < 0) throw new ZeroPriceException("The price can`t be a negative value.");  if (Editing != null) Editing(price);  price = value;  if (Edited!= null) Edited(price);  }  }  public int CompareTo(object obj)  {  return Name.CompareTo((obj as IName).Name);  }  public int CompareTo(Product obj)  {  return Name.CompareTo(obj.Name);  }  public override string ToString()  {  return string.Format($"Name:{Name},Price:{Price}");  }  }  }  Program.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Runtime.Serialization.Formatters.Binary;  using System.IO;  namespace Music  {  public class Program  {  public static void Main()  {  //ArrayList  Product elgr = new ElectroGuitar("SPG900", 2000, "FANNDEC", 6, "rosewood", "metal", "H-S-H");  Product drm1 = new Drum("VMX-2218B/C250", 89500, "PEARL", 18, "maple");  Product drm2 = new Drum("VMX-2216B/C280", 28540, "TAMA", 16, "bubinga");  Product drm3 = new Drum("VMX-2168B/C270", 8940, "PEARL", 18, "maple");  Product drm4 = new Drum("TBB2216S", 7585, "TAMA", 16, "bubinga");  Product drm5 = new Drum("VMX-2218B/C280", 9995, "PEARL", 18, "maple");  Product drm6 = new Drum("TBB2168S", 6980, "TAMA", 18, "bubinga");  Container<Product> con = new Container<Product>();  con.Add(elgr, drm1, drm2, drm3, drm4, drm5, drm6);  //con.Remove(1);  Compare<Product> del1 = new Compare<Product>((Product a, Product b) => { return a.Name.CompareTo(b.Name); });  Compare<Product> del2 = new Compare<Product>((Product a, Product b) => { return a.Price.CompareTo(b.Price); });  con.Sort(del2);  Find<Product> del3 = new Find<Product>((Product a) => { return(a.Price > 5000 && a.Price < 10000); });  var filter1 = from obj in con.Repository where obj.Price == (from temp in con.Repository select (temp.Price)).Max() select obj; //нахождение объекта с максимальной ценой  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Max: " + filter1.FirstOrDefault());  var filter2 = con.Where(obj => obj.Price == (con.Min(x => x.Price))); //нахождение объекта с минимальной ценой  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Min: " + filter2.FirstOrDefault());  var filterCategory = from obj in con group obj.Price by obj.GetType().Name into g select new { Catagory = g.Key, Value = g.Average().ToString("0.00") };  //средняя цена по каждой категории (категория - название соответствующего класса (тип объекта))  foreach (var obj in filterCategory)  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine(obj);  }    var filter3 = (con.Average(x => x.Price)).ToString("0.00"); //средняя цена всех товаров  Console.WriteLine("Avg: " + filter3);  //Сериализация  BinaryFormatter formatter = new BinaryFormatter();  using (FileStream fs = new FileStream("container.dat", FileMode.OpenOrCreate))  {  formatter.Serialize(fs, con);  Console.ForegroundColor = ConsoleColor.Yellow;  Console.WriteLine("Object was serialized.");  }  //Десериализация  using (FileStream fs = new FileStream("container.dat", FileMode.OpenOrCreate))  {  Container<Product> deserilizeCon = (Container<Product>)formatter.Deserialize(fs);  Console.ForegroundColor = ConsoleColor.Yellow;  foreach (var obj in deserilizeCon)  {  Console.WriteLine(obj);  }  }  //Console.WriteLine(con.FindAll(del3)[0]);  //LinkedList  Product item1 = new Product("Guitar", 3500);  Product item2 = new Product("Violin", 4000);  Product item3 = new Product("Harp", 18999);  Product item4 = new Product("Guitar", 8045);  LinkedListContainer<Product> list = new LinkedListContainer<Product>();  list.Add(item1);  list.Add(item2);  list.Add(item3);  list.Add(item4);  Compare<Product> del4 = new Compare<Product>((Product a, Product b) => { return a.Name.CompareTo(b.Name); });  Compare<Product> del5 = new Compare<Product>((Product a, Product b) => { return a.Price.CompareTo(b.Price); });  list.Sort(del5);  Find<Product> del7 = new Find<Product>((Product a) => { return (a.Price > 3000 && a.Name == "Guitar"); });  var filter4 = list.Aggregate((x,y) => x.Price > y.Price ? x : y); //нахождение объекта с максимальной ценой  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Max: " + filter4);  var filter5 = list.Aggregate((x, y) => x.Price < y.Price ? x : y); //нахождение объекта с минимальной ценой  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Min: " + filter5);  var filterCategory2 = from obj in list group obj.Price by obj.GetType().Name into g select new { Catagory = g.Key, Value = g.Average().ToString("0.00") };  //средняя цена по каждой категории (категория - название соответствующего класса (тип объекта))  foreach (var obj in filterCategory2)  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine(obj);  }  var filter6 = list.Average(x => x.Price).ToString("0.00"); //средняя цена всех товаров  Console.WriteLine("Avg: " + filter6);  //Сериализация  BinaryFormatter formatter1 = new BinaryFormatter();  using (FileStream fs = new FileStream("container.dat", FileMode.OpenOrCreate))  {  formatter1.Serialize(fs, list);  Console.ForegroundColor = ConsoleColor.Yellow;  Console.WriteLine("Object was serialized.");  }  //Десериализация  using (FileStream fs = new FileStream("container.dat", FileMode.OpenOrCreate))  {  LinkedListContainer<Product> deserilizeCon = (LinkedListContainer<Product>)formatter1.Deserialize(fs);  Console.ForegroundColor = ConsoleColor.Yellow;  foreach (var obj in deserilizeCon)  {  Console.WriteLine(obj);  }  }  //Console.WriteLine(list.FindAll(del7)[1]);  Console.ReadKey();  }  }  } | Delegates.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  public delegate int Compare<T>(T a, T b);  public delegate bool Find<T>(T a);  public delegate void PriceChangingHadler(decimal sum);  }  DigitalPiano.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Music  {  [Serializable]  public class DigitalPiano : Instrument  {  private int numberOfKeys;  private string typeOfMechanics;  public DigitalPiano()  {    }  public DigitalPiano(string name, decimal price, string manufacturer, int numberOfKeys, string typeOfMechanics):base(name, price, manufacturer)  {  this.numberOfKeys = numberOfKeys;  this.typeOfMechanics = typeOfMechanics;  }  public string TypeOfMechanics  {  get  {  return typeOfMechanics;  }  set  {  typeOfMechanics = value;  }  }  public int NumberOfKeys  {  get  {  return NumberOfKeys;  }  set  {  if (numberOfKeys < 0) throw new ZeroPriceException("The number of keys can`t be a negative value.");  numberOfKeys = value;  }  }  public override string ToString()  {  return string.Format($"Name:{name},Price:{price},manufacturer:{manufacturer},numberOfKeys:{numberOfKeys},typeOfMechanics:{typeOfMechanics}");  }  }  }  Drum.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Music  {  [Serializable]  public class Drum : Instrument  {  private int depth;  private string material;  public Drum(string name, decimal price, string manufacturer, int depth, string material):base(name,price,manufacturer)  {  this.depth = depth;  this.material = material;  }  public Drum()  {    }  public string Material  {  get  {  return material;  }  set  {  material = value;  }  }  public int Depth  {  get  {  return depth;  }  set  {  if (depth < 0) throw new ZeroPriceException("The depth can`t be a negative value.");  depth = value;  }  }  public override string ToString()  {  return string.Format($"Name:{name},Price:{price},manufacturer:{manufacturer},depth:{depth},material:{material}");  }  }  }  ElectroGuitar.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Music  {  [Serializable]  public class ElectroGuitar : Guitar  {    private string typeOfPickups;  public ElectroGuitar(string name, decimal price, string manufacturer, int numberOfStrings, string guitarNeck, string typeOfStrings, string typeOfPickups) : base(name, price, manufacturer, numberOfStrings, guitarNeck, typeOfStrings)  {  this.typeOfPickups = typeOfPickups;  }  public ElectroGuitar()  {    }  public string TypeOfPickups  {  get  {  return typeOfPickups;  }  set  {  typeOfPickups = value;  }  }  public override string ToString()  {  return string.Format($"Name:{name},Price:{price},manufacturer:{manufacturer},guitarNeck:{guitarNeck},numberOfStrings:{numberOfStrings},typeOfStrings:{typeOfStrings},typeOfPickups:{typeOfPickups}");  }  }  }  Guitar.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Music  {  [Serializable]  public class Guitar : Instrument  {  protected string guitarNeck;  protected int numberOfStrings;  protected string typeOfStrings;  public Guitar(string name, decimal price, string manufacturer, int numberOfStrings, string guitarNeck, string typeOfStrings):base(name, price, manufacturer)  {  this.guitarNeck = guitarNeck;  this.numberOfStrings = numberOfStrings;  this.typeOfStrings = typeOfStrings;  }  public Guitar()  {    }  public int NumberOfStrings  {  get  {  return numberOfStrings;  }  set  {  if (numberOfStrings < 0) throw new ZeroPriceException("The number Of strings can`t be a negative value.");  numberOfStrings = value;  }  }  public string TypeOfStrings  {  get  {  return typeOfStrings;  }  set  {  typeOfStrings = value;  }  }  public string GuitarNeck  {  get  {  return guitarNeck;  }  set  {  guitarNeck = value;  }  }  public override string ToString()  {  return string.Format($"Name:{name},Price:{price},manufacturer:{manufacturer},guitarNeck:{guitarNeck},numberOfStrings:{numberOfStrings},typeOfStrings:{typeOfStrings}");  }  }  }  IName.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  interface IName : IComparable  {  string Name { get; set; }  }  }  IName2.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  interface IName<T> : IComparable<T>  {  string Name { get; set; }  }  }  Instrument.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Music  {  [Serializable]  public class Instrument : Product  {  protected string manufacturer;  public Instrument(string name, decimal price, string manufacturer):base(name, price)  {  this.manufacturer = manufacturer;  }  public Instrument()  {    }  public string Manufacturer  {  get  {  return manufacturer;  }  set  {  manufacturer = value;  }  }  public override string ToString()  {  return string.Format($"Name:{name},Price:{price},manufacturer:{manufacturer}");  }  }  }  LinkedListContainer.cs  using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  [Serializable]  class LinkedListContainer<T> : IEnumerable<T> where T: IName<T>  {  public event PriceChangingHadler Added;  LinkedListNode<T> start;  LinkedListNode<T> end;  public uint Count { get; private set; }  public decimal Sum { get; set; }  public void PriceChanging(decimal price)  {  Sum -= price;  }  public void PriceChanged(decimal price)  {  Sum += price;  }  public T this[int index]  {  get  {  if (index < 0 || index >= Count)  {  throw new OutOfRangeException("The index is outside the list.");  }  var temp = start;  for (int i = 0; i < index; i++)  {  temp = temp.Next;  }  return temp.Data;  }  set  {  if (index < 0 || index >= Count)  {  throw new OutOfRangeException("Specifying is not possible. The index is outside the list.");  }  var temp = start;  for (int i = 0; i < index; i++)  {  temp = temp.Next;  }  temp.Data = value;  }  }  public T this[string name]  {  get  {  var temp = start;  for (int i = 0; i < Count; i++)  {  if (temp.Data.Name == name)  {  return temp.Data;  }  temp = temp.Next;  }  return default(T);  }  set  {  var temp = start;  for (int i = 0; i < Count; i++)  {  if (temp.Data.Name == name)  {  temp.Data = value;  }  temp = temp.Next;  }  }  }  public void Add(T obj)  {  if (obj == null)  {  throw new AddNullException("Adding is not possible. Object is null.");  }  LinkedListNode<T> node = new LinkedListNode<T>();  node.Data = obj;  if (start == null)  {  start = node;  end = node;  }  else  {  end.Next = node;  node.Previous = end;  end = node;  }  (obj as Product).Editing += PriceChanging;  (obj as Product).Edited += PriceChanged;  Sum += (obj as Product).Price;  Count++;  }  public void Remove(int index)  {  if (index < 0 || index >= Count)  {  throw new OutOfRangeException("Specifying is not possible. The index is outside the list.");  }  else if (index == 0) // Если индекс равен начальному объекту в списке  {  if (start == null)  {  throw new NullReferenceException();  }  else  {  if (start.Next != null)  {  start.Next.Previous = null;  }  start = start.Next;  Count--;  }  }  else if (index == Count - 1) // Если индекс равен последнему объекту в списке  {  if (end == null)  {  throw new NullReferenceException();  }  else  {  if (end.Previous != null)  {  end.Previous.Next = null;  }  end = end.Previous;  Count--;  }  }  else  {  var temp = start;  for (int i = 0; i < index; i++)  {  temp = start.Next;  }  temp.Previous.Next = temp.Next;  temp.Next.Previous = temp.Previous;  Count--;  }  (this[index] as Product).Editing -= PriceChanging;  (this[index] as Product).Edited -= PriceChanged;  Sum -= (this[index] as Product).Price;  }  public void Sort()  {  T[] temp = new T[Count];  for (int i = 0; i < Count; i++)  {  temp[i] = this[i];  }  for (int i = 0; i < temp.Length; i++)  {  for (int j = i + 1; j < temp.Length; j++)  {  if ((temp[i] as Product).Price > (temp[i] as Product).Price)  {  var obj = temp[i];  temp[i] = temp[j];  temp[j] = obj;  }  }  }  for (int i = 0; i < Count; i++)  {  this[i] = temp[i];  }  }  public void Sort(Compare<T> del)  {  T[] temp = new T[Count];  for (var i = 0; i < Count; i++)  {  temp[i] = this[i];  }  for (int i = 0; i < temp.Length; i++)  {  for (int j = i + 1; j < temp.Length; j++)  {  if (del(temp[i], temp[j]) > 0)  {  var obj = temp[i];  temp[i] = temp[j];  temp[j] = obj;  }  }  }  //Array.Sort(temp, (T x, T y) => { return del(x, y); });  for (int i = 0; i < Count; i++)  {  this[i] = temp[i];  }  }  public T Find(Find<T> del)  {  foreach (T obj in this)  {  if (del(obj)) return obj;  }  return default(T);  }  public T[] FindAll(Find<T> del)  {  T[] main = new T[0];  foreach (T obj in this)  {  if (del(obj))  {  T[] temp = new T[main.Length + 1];  for (var i = 0; i < main.Length; i++)  {  temp[i] = main[i];  }  temp[temp.Length - 1] = obj;  main = temp;  }  }  return main;  }  public override string ToString()  {  StringBuilder result = new StringBuilder("");  var temp = start;  for (int i = 0; i < Count; i++)  {  result.Append(temp.Data);  temp = temp.Next;  result.Append("\n");  }  return result.ToString();  }  public IEnumerable Reverse()  {  for (int i = (int)Count - 1; i >= 0; i--)  {  yield return this[i];  }  }  public IEnumerable FindByString(string s)  {  for (int i = 0; i < Count; i++)  {  if (this[i].Name.Contains(s))  {  yield return this[i];  }  }  }  public IEnumerable InOrder()  {  T[] temp = new T[Count];  var node = start;  for (int i = 0; i < Count; i++)  {  temp[i] = node.Data;  node = node.Next;  }  //temp = temp.OrderBy(x => x.Name).ToArray();  for (int i = 0; i < temp.Length; i++)  {  for (int j = i + 1; j < temp.Length; j++)  {  if ((temp[i] as Product).Price > (temp[j] as Product).Price)  {  var obj = temp[i];  temp[i] = temp[j];  temp[j] = obj;  }  }  }  for (int i = 0; i < Count; i++)  {  yield return temp[i];  }  }  IEnumerator IEnumerable.GetEnumerator()  {  return GetEnumerator();  }  public IEnumerator<T> GetEnumerator()  {  return new LinkedListIterator<T>(this);  }  }  }  ZeroPriceException.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Music  {  class ZeroPriceException : Exception  {  public ZeroPriceException(string message) : base(message)  {  }  public ZeroPriceException(string message, Exception ex)  : base(message, ex)  {  }  public ZeroPriceException() : base()  {  }  }  } |