Deakin University

Object Oriented Development

OnTrack Submission

Bucket Sort

Submitted By:
Peter Stacey
pstacey
2020/04/05 11:22

Tutor:
Dipto Pratyaksa

Outcome	**Weight**
Evaluate Code	♦♦♦◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦♦♦◊
Design	♦♦♦◊◊
Justify	♦♦♦♦◊

The task, while providing the basic pseudocode, doesn't provide a lot of guidance and a lot of room is left to evaluate the needs and design a solution. As the task sheet indicates we can use List<T>.Sort (as an example) to complete the final sorting after distributing the accounts into buckets, the task encourages going beyond the learning, to read the Microsoft documentation for C# and the .NET framework. This aligns well with writing code that complies with conventions in the language, especially with the ability to use Linq features that are relevant to the .NET Framework and C# syntax and semantics. My video additionally provides further evidence and critiquing of the quality of the code and result.

April 5, 2020

```csharp
using System;
using System.Collections.Generic;

namespace Task_3._4D
{
    class Program
    {

        static void PrintAccountArray(Account[] accounts)
        {
            foreach (Account account in accounts)
                account.Print();
        }

        public static void Main(string[] args)
        {
            Console.WriteLine("\n**************************************************");
            Console.WriteLine("** TESTING START");
            Console.WriteLine("**************************************************\n");

            Random random = new Random();
            int numberOfAccounts = random.Next(15, 50);

            // Testing REASONABLE Arguments

            Account[] accountsArray = new Account[numberOfAccounts];
            for (int i = 0; i < accountsArray.Length; i++)
            {
                accountsArray[i] = new Account("Jane Doe", Convert.ToDecimal(random.Next(10, 5000)));
            }

            Console.WriteLine("\n**************************************************");
            Console.WriteLine("** Array Order before beginning to sort:");
            Console.WriteLine("**************************************************\n");

            PrintAccountArray(accountsArray);
            AccountSorter.Sort(accountsArray, 5);

            Console.WriteLine("\n**************************************************");
            Console.WriteLine("** Array Order After sorting:");
            Console.WriteLine("**************************************************\n");

            PrintAccountArray(accountsArray);

            List<Account> accountsList = new List<Account>();
            for (int i = 0; i < numberOfAccounts; i++)
            {
                accountsList.Add(new Account("Jane Doe", Convert.ToDecimal(random.Next(10, 5000))));
            }

            Console.WriteLine("\n*************************************************");
            Console.WriteLine("** List Order before beginning to sort:");
            Console.WriteLine("*************************************************\n");

            PrintAccountArray(accountsList.ToArray());
            AccountSorter.Sort(accountsList, 5);

            Console.WriteLine("\n*************************************************");
            Console.WriteLine("** List Order After sorting:");
            Console.WriteLine("*************************************************\n");

            PrintAccountArray(accountsList.ToArray());


            // Testing BAD Arguments

            Console.WriteLine("\n\n*************************************************");
            Console.WriteLine("** Testing Bad Arguments:");
            Console.WriteLine("*************************************************\n");


            Account[] badArray = null;

            try
            {
                AccountSorter.Sort(badArray, 5);  // Null array
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine(ex.Message);
            }

            try
            {
                AccountSorter.Sort(accountsArray, 0); // 0 buckets
            }
            catch (ArgumentOutOfRangeException ex)
            {
                Console.WriteLine(ex.Message);
            }

            List<Account> badList = null;

            try
            {
                AccountSorter.Sort(badList, 5); // Null list
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine(ex.Message);
            }

            try
            {
                AccountSorter.Sort(accountsList, 0); // 0 buckets
            }
            catch (ArgumentOutOfRangeException ex)
            {
                Console.WriteLine(ex.Message);
            }

            Console.WriteLine("\n\n*************************************************");
            Console.WriteLine("** TESTING END");
            Console.WriteLine("*************************************************");
        }
    }
}
```

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace Task_3._4D
{
    static class AccountSorter
    {
        /// <summary>
        /// Returns the maximum account balance from an array of accounts
        /// </summary>
        /// <returns>
        /// The maximum account balance as a decimal
        /// </returns>
        /// <param name="accounts">The array of accounts</param>
        private static decimal MaximumBalance(Account[] accounts)
        {
            return accounts.Max(a => a.Balance);
        }

        /// <summary>
        /// Creates and initializes required list of buckets
        /// </summary>
        /// <returns>
        /// Array of buckets containing a list to store accounts
        /// </returns>
        /// <param name="b">The number of buckets required</param>
        private static List<Account>[] CreateBuckets(int b)
        {
            List<Account>[] buckets = new List<Account>[b];
            for (int i = 0; i < buckets.Length; i++)
            {
                buckets[i] = new List<Account>();
            }
            return buckets;
        }

        /// <summary>
        /// Distributes accounts into buckets from the array of accounts
        /// </summary>
        /// <param name="accounts">The array of accounts to distribute</param>
        /// <param name="buckets">The array of buckets to distribute into</param>
        private static void DistributeAccounts(Account[] accounts, List<Account>[]
         ↪  buckets)
        {
            decimal maximum = MaximumBalance(accounts);
            foreach (Account account in accounts)
            {
                int bucket = (int)(Math.Floor(buckets.Length * account.Balance /
                 ↪  maximum));
                if (bucket == buckets.Length)
                    bucket -= 1;
                buckets[bucket].Add(account);
            }
        }

        /// <summary>
        /// Sorts the accounts in each bucket by account balance
        /// </summary>
        /// <param name="buckets">The buckets holding accounts</param>
        private static void SortBuckets(List<Account>[] buckets)
        {
            for (int i = 0; i < buckets.Length; i++)
            {
                buckets[i] = buckets[i].OrderBy(a => a.Balance).ToList();
            }
        }

        /// <summary>
        /// Sorts an array of accounts by their account balance from
        /// smallest to largest
        /// </summary>
        /// <param name="accounts">The array of accounts to sort</param>
        /// <param name="b">The number of buckets to use</param>
        /// <exception cref="System.NullReferenceException">Thrown
        /// if the accounts array is null</exception>
        /// <exception cref="System.ArgumentOutOfRangeException">Thrown
        /// if the number of buckets is 0 or less</exception>
        public static void Sort(Account[] accounts, int b)
        {
            if (accounts == null)
            {
                throw new NullReferenceException("Accounts cannot be null");
            }

            if (b <= 1)
            {
                throw new ArgumentOutOfRangeException("At least 2 buckets needed");
            }

            List<Account>[] buckets = CreateBuckets(b);
            DistributeAccounts(accounts, buckets);
            SortBuckets(buckets);

            // Write the accounts in the buckets back into the original
            // accounts array. Idx tracks the position in the
            // original accounts array to write to
            int idx = 0;
            for (int i = 0; i < buckets.Length; i++)
            {
                foreach (Account account in buckets[i])
                {
                    accounts[idx] = account;
                    idx++;
                }
            }
        }

        /// <summary>
        /// Sorts a list of accounts by their account balance from
        /// smallest to largest
        /// </summary>
        /// <param name="accounts">The list of accounts to sort</param>
        /// <param name="b">The number of buckets to use</param>
        /// <exception cref="System.NullReferenceException">Thrown
        /// if the accounts list is null</exception>
        /// <exception cref="System.ArgumentOutOfRangeException">Thrown
        /// if the number of buckets is 0 or less</exception>
        public static void Sort(List<Account> accounts, int b)
        {
            if (accounts == null)
            {
                throw new NullReferenceException("Accounts cannot be null");
            }

            Account[] accountsArray = accounts.ToArray();
            Sort(accountsArray, b);

            // Write the accountsArray back into the accounts list.
            // Cannot simply call .ToList() as order is not guaranteed.
            for (int i = 0; i < accounts.Count; i++)
            {
                accounts[i] = accountsArray[i];
            }
        }
    }
}
```

```
using System;

namespace Task_3._4D
{
    /// <summary>
    /// A bank account class to hold the account name and balance details
    /// </summary>
    class Account
    {
        // Instance variables
        private String _name;
        private decimal _balance;

        // Read-only properties
        public String Name { get => _name; }
        public decimal Balance { get => _balance; }


        /// <summary>
        /// Class constructor
        /// </summary>
        /// <param name="name">The name string for the account</param>
        /// <param name="balance">The decimal balance of the account</param>
        public Account(String name, decimal balance = 0)
        {
            _name = name;
            if (balance <= 0)
                return;
            _balance = balance;
        }

        /// <summary>
        /// Deposits money into the account
        /// </summary>
        /// <returns>
        /// Boolean whether the deposit was successful (true) or not (false)
        /// </returns>
        /// <param name="amount">The decimal amount to add to the balance</param>
        public Boolean Deposit(decimal amount)
        {
            if (amount <= 0)
                return false;

            _balance += amount;
            return true;
        }

        /// <summary>
        /// Withdraws money from the account (with no overdraw protection currently)
        /// </summary>
        /// <returns>
        /// Boolean whether the withdrawal was successful (true) or not (false)
        /// </returns>
        /// <param name="amount">The amount to subtract from the balance</param>
        public Boolean Withdraw(decimal amount)
        {
            if ((amount <= 0) || (amount > _balance))
                return false;

            _balance -= amount;
            return true;
        }

        /// <summary>
        /// Outputs the account name and current balance as a string
        /// </summary>
        public void Print()
        {
            Console.WriteLine("Account Name: {0}, Balance: {1}",
                _name, _balance.ToString("C"));
        }
    }
}
```