

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



```
1  using System;
2  using System.Collections.Generic;
3
4  namespace Task_3._4D
5  {
6      class Program
7      {
8
9          static void PrintAccountArray(Account[] accounts)
10         {
11             foreach (Account account in accounts)
12                 account.Print();
13         }
14
15         public static void Main(string[] args)
16         {
17             Console.WriteLine("\n*****"
18                               ↪ );
19             Console.WriteLine("** TESTING START");
20             Console.WriteLine("*****\n"
21                               ↪ );
22
23             Random random = new Random();
24             int numberOfAccounts = random.Next(15, 50);
25
26             // Testing REASONABLE Arguments
27
28             Account[] accountsArray = new Account[numberOfAccounts];
29             for (int i = 0; i < accountsArray.Length; i++)
30             {
31                 accountsArray[i] = new Account("Jane Doe",
32                               ↪ Convert.ToDecimal(random.Next(10, 5000)));
33             }
34
35             Console.WriteLine("\n*****"
36                               ↪ );
37             Console.WriteLine("** Array Order before beginning to sort:");
38             Console.WriteLine("*****\n"
39                               ↪ );
40
41             PrintAccountArray(accountsArray);
42             AccountSorter.Sort(accountsArray, 5);
43
44             Console.WriteLine("\n*****"
45                               ↪ );
46             Console.WriteLine("** Array Order After sorting:");
47             Console.WriteLine("*****\n"
48                               ↪ );
49
50             PrintAccountArray(accountsArray);
51
52             List<Account> accountsList = new List<Account>();
53             for (int i = 0; i < numberOfAccounts; i++)
```

```
47     {
48         accountsList.Add(new Account("Jane Doe",
49             ↪ Convert.ToDecimal(random.Next(10, 5000))));
50     }
51
52     Console.WriteLine("\n*****"
53     ↪ );
54
55     Console.WriteLine("** List Order before beginning to sort:");
56     Console.WriteLine("*****\n"
57     ↪ );
58
59     PrintAccountArray(accountsList.ToArray());
60     AccountSorter.Sort(accountsList, 5);
61
62     Console.WriteLine("\n*****"
63     ↪ );
64
65     Console.WriteLine("** List Order After sorting:");
66     Console.WriteLine("*****\n"
67     ↪ );
68
69     PrintAccountArray(accountsList.ToArray());
70
71     // Testing BAD Arguments
72
73     Console.WriteLine("\n\n*****"
74     ↪ *");
75
76     Console.WriteLine("** Testing Bad Arguments:");
77     Console.WriteLine("*****\n"
78     ↪ );
79
80     Account[] badArray = null;
81
82     try
83     {
84         AccountSorter.Sort(badArray, 5); // Null array
85     }
86     catch (NullReferenceException ex)
87     {
88         Console.WriteLine(ex.Message);
89     }
90
91     try
92     {
93         AccountSorter.Sort(accountsArray, 0); // 0 buckets
94     }
95     catch (ArgumentOutOfRangeException ex)
96     {
97         Console.WriteLine(ex.Message);
98     }
99
100     List<Account> badList = null;
```

```
93
94     try
95     {
96         AccountSorter.Sort(badList, 5); // Null list
97     }
98     catch (NullReferenceException ex)
99     {
100         Console.WriteLine(ex.Message);
101     }
102
103     try
104     {
105         AccountSorter.Sort(accountsList, 0); // 0 buckets
106     }
107     catch (ArgumentOutOfRangeException ex)
108     {
109         Console.WriteLine(ex.Message);
110     }
111
112     Console.WriteLine("\n\n*****
    ↳ *");
113     Console.WriteLine("** TESTING END");
114     Console.WriteLine("*****");
115 }
116 }
117 }
```

```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4
5  namespace Task_3._4D
6  {
7      static class AccountSorter
8      {
9          /// <summary>
10         /// Returns the maximum account balance from an array of accounts
11         /// </summary>
12         /// <returns>
13         /// The maximum account balance as a decimal
14         /// </returns>
15         /// <param name="accounts">The array of accounts</param>
16         private static decimal MaximumBalance(Account[] accounts)
17         {
18             return accounts.Max(a => a.Balance);
19         }
20
21         /// <summary>
22         /// Creates and initializes required list of buckets
23         /// </summary>
24         /// <returns>
25         /// Array of buckets containing a list to store accounts
26         /// </returns>
27         /// <param name="b">The number of buckets required</param>
28         private static List<Account>[] CreateBuckets(int b)
29         {
30             List<Account>[] buckets = new List<Account>[b];
31             for (int i = 0; i < buckets.Length; i++)
32             {
33                 buckets[i] = new List<Account>();
34             }
35             return buckets;
36         }
37
38         /// <summary>
39         /// Distributes accounts into buckets from the array of accounts
40         /// </summary>
41         /// <param name="accounts">The array of accounts to distribute</param>
42         /// <param name="buckets">The array of buckets to distribute into</param>
43         private static void DistributeAccounts(Account[] accounts, List<Account>[]
44         ↪ buckets)
45         {
46             decimal maximum = MaximumBalance(accounts);
47             foreach (Account account in accounts)
48             {
49                 int bucket = (int)(Math.Floor(buckets.Length * account.Balance /
50                 ↪ maximum));
51                 if (bucket == buckets.Length)
52                     bucket -= 1;
53                 buckets[bucket].Add(account);
54             }
55         }
56     }
57 }
```

```
52     }
53 }
54
55 /// <summary>
56 /// Sorts the accounts in each bucket by account balance
57 /// </summary>
58 /// <param name="buckets">The buckets holding accounts</param>
59 private static void SortBuckets(List<Account>[] buckets)
60 {
61     for (int i = 0; i < buckets.Length; i++)
62     {
63         buckets[i] = buckets[i].OrderBy(a => a.Balance).ToList();
64     }
65 }
66
67 /// <summary>
68 /// Sorts an array of accounts by their account balance from
69 /// smallest to largest
70 /// </summary>
71 /// <param name="accounts">The array of accounts to sort</param>
72 /// <param name="b">The number of buckets to use</param>
73 /// <exception cref="System.NullReferenceException">Thrown
74 /// if the accounts array is null</exception>
75 /// <exception cref="System.ArgumentOutOfRangeException">Thrown
76 /// if the number of buckets is 0 or less</exception>
77 public static void Sort(Account[] accounts, int b)
78 {
79     if (accounts == null)
80     {
81         throw new NullReferenceException("Accounts cannot be null");
82     }
83
84     if (b <= 1)
85     {
86         throw new ArgumentOutOfRangeException("At least 2 buckets needed");
87     }
88
89     List<Account>[] buckets = CreateBuckets(b);
90     DistributeAccounts(accounts, buckets);
91     SortBuckets(buckets);
92
93     // Write the accounts in the buckets back into the original
94 // accounts array. Idx tracks the position in the
95 // original accounts array to write to
96     int idx = 0;
97     for (int i = 0; i < buckets.Length; i++)
98     {
99         foreach (Account account in buckets[i])
100         {
101             accounts[idx] = account;
102             idx++;
103         }
104     }
```

```
105     }
106
107     /// <summary>
108     /// Sorts a list of accounts by their account balance from
109     /// smallest to largest
110     /// </summary>
111     /// <param name="accounts">The list of accounts to sort</param>
112     /// <param name="b">The number of buckets to use</param>
113     /// <exception cref="System.NullReferenceException">Thrown
114     /// if the accounts list is null</exception>
115     /// <exception cref="System.ArgumentOutOfRangeException">Thrown
116     /// if the number of buckets is 0 or less</exception>
117     public static void Sort(List<Account> accounts, int b)
118     {
119         if (accounts == null)
120         {
121             throw new NullReferenceException("Accounts cannot be null");
122         }
123
124         Account[] accountsArray = accounts.ToArray();
125         Sort(accountsArray, b);
126
127         // Write the accountsArray back into the accounts list.
128         // Cannot simply call .ToList() as order is not guaranteed.
129         for (int i = 0; i < accounts.Count; i++)
130         {
131             accounts[i] = accountsArray[i];
132         }
133     }
134 }
135 }
```

```
1  using System;
2
3  namespace Task_3._4D
4  {
5      /// <summary>
6      /// A bank account class to hold the account name and balance details
7      /// </summary>
8      class Account
9      {
10         // Instance variables
11         private String _name;
12         private decimal _balance;
13
14         // Read-only properties
15         public String Name { get => _name; }
16         public decimal Balance { get => _balance; }
17
18
19         /// <summary>
20         /// Class constructor
21         /// </summary>
22         /// <param name="name">The name string for the account</param>
23         /// <param name="balance">The decimal balance of the account</param>
24         public Account(String name, decimal balance = 0)
25         {
26             _name = name;
27             if (balance <= 0)
28                 return;
29             _balance = balance;
30         }
31
32         /// <summary>
33         /// Deposits money into the account
34         /// </summary>
35         /// <returns>
36         /// Boolean whether the deposit was successful (true) or not (false)
37         /// </returns>
38         /// <param name="amount">The decimal amount to add to the balance</param>
39         public Boolean Deposit(decimal amount)
40         {
41             if (amount <= 0)
42                 return false;
43
44             _balance += amount;
45             return true;
46         }
47
48         /// <summary>
49         /// Withdraws money from the account (with no overdraw protection currently)
50         /// </summary>
51         /// <returns>
52         /// Boolean whether the withdrawal was successful (true) or not (false)
53         /// </returns>
```



```
54     /// <param name="amount">The amount to subtract from the balance</param>
55     public Boolean Withdraw(decimal amount)
56     {
57         if ((amount <= 0) || (amount > _balance))
58             return false;
59
60         _balance -= amount;
61         return true;
62     }
63
64     /// <summary>
65     /// Outputs the account name and current balance as a string
66     /// </summary>
67     public void Print()
68     {
69         Console.WriteLine("Account Name: {0}, Balance: {1}",
70             _name, _balance.ToString("C"));
71     }
72 }
73 }
```