Home Test

**Problem Solving**

1. Create console application project
   1. Complete implementation for RandomFactory<T> class.

The purpose of this class to give control to users to get random values of “T” type.

Pay attention that after indexing, the users of this object can remove elements from “T” collection, so there can be a case that the index is not preserved in continuous order.

* 1. Code:

1. public class RandomFactory<T>
2. {
3. private Dictionary<int, T> indexDict;
4. public RandomFactory(List<T> collectionToRandom)
5. {
6. this.TotalRecords = collectionToRandom.Count;
7. this.SetIndexToAllItems(collectionToRandom);
8. }
9. public int TotalRecords { get; private set; }
10. private void SetIndexToAllItems(List<T> collectionToRandom)
11. {
12. //implement
13. }
14. public KeyValuePair<int,T> GetRandom()
15. {
16. //implement
17. }
18. public void RemoveItem(int index)
19. {
20. this.indexDict.Remove(index);
21. //implement
22. }
23. }

public class RandomFactory<T>



[This Photo](https://proofmart.com/product/github-logo-png-hd-images-transparent-background-free-download/) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

Available on GitHub

{

private Dictionary<int, T> indexDict;

public RandomFactory(List<T> collectionToRandom)

{

this.TotalRecords = collectionToRandom.Count;

this.SetIndexToAllItems(collectionToRandom);

}

public int TotalRecords { get; private set; }

private void SetIndexToAllItems(List<T> collectionToRandom)

{

indexDict = new Dictionary<int, T>();

for (int i = 0; i < TotalRecords; i++)

indexDict.Add(i, collectionToRandom[i]);

}

public KeyValuePair<int, T> GetRandom()

{

Random rnd = new Random();

int i = -1;

while (!indexDict.ContainsKey(i))

i = rnd.Next(0, TotalRecords); //generate a random index within the range until we find an existing one

return new KeyValuePair<int, T>(i, indexDict[i]);

}

public void RemoveItem(int index)

{

this.indexDict.Remove(index);

}

}

**OOB**

1. Create console application project
2. Create ContactBL class .
3. Create BaseExecutor abstract class that implements interface IExecute, with Execute method that receives int argument and returns string.
4. Create ExecutorFactory that receive int argument and return implementation for BaseExecutor by logic given below. (each implementation must be a class):
   1. If int is 1 then implementation return “option 1”
   2. If int is 2 then implementation return “option 2”

4.3 If int is 3 then the two implementations above returned.

5. In ContactBL add function ExecuteAgeLogic that receives int and returns the value by calling ExecutorFactory.

6. The outputted string for each implementation must begin with a new line (except the first one)

interface IExecute



[This Photo](https://proofmart.com/product/github-logo-png-hd-images-transparent-background-free-download/) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

Available on GitHub

{

string Execute(int a);

}

abstract class BaseExecutor : IExecute

{

public virtual string Execute(int a)

{

return "default";

}

}

class Option1 : BaseExecutor

{

public override string Execute(int a)

{

return "option 1";

}

}

class Option2 : BaseExecutor

{

public override string Execute(int a)

{

return "option 2";

}

}

class Option3 : BaseExecutor

{

public override string Execute(int a)

{

return "option 1\noption 2";

}

}

public class ContactBL

{

string ExecutorFactory(int a)

{

BaseExecutor obj;

if (a == 1)

{

obj = new Option1();

}

else if (a == 2)

{

obj = new Option2();

}

else if (a == 3)

{

obj = new Option3();

}

else

return "non of option";

return obj.Execute(1);

}

public string ExecuteAgeLogic(int a)

{

return ExecutorFactory(a);

}

**Algorithms**

1. Write a method with complexity of 0(n), that receives unordered array of integers.

And returns the start and end index of the range that has the maximum summary amount.

Example:

[3, -1, 4, -8, 2] - > start = 0, end = 2 (3, -1, 4)

[-5, -3, -2, -7] -> start = 2, end =2 (-2)

static (int, int) MaxSumRange(int[] a)



[This Photo](https://proofmart.com/product/github-logo-png-hd-images-transparent-background-free-download/) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

Available on GitHub

{

if (a.Length == 0)

return (0, 0);

int maxSum = a.Min();

int maxRangeStart = 0;

int maxRangeEnd = 0;

int currentSum = 0;

int currentStart = 0;

for (int i = 0; i < a.Length; i++)

{

int currentEnd = i;

if (currentSum <= 0)

{

currentStart = currentEnd;

currentSum = a[i];

}

else

currentSum += a[i];

if (currentSum >= maxSum)

{

maxSum = currentSum;

maxRangeStart = currentStart;

maxRangeEnd = currentEnd;

}

}

return (maxRangeStart, maxRangeEnd);

}