Name : Ritik Gupta

There are 6 questions in this assessment.

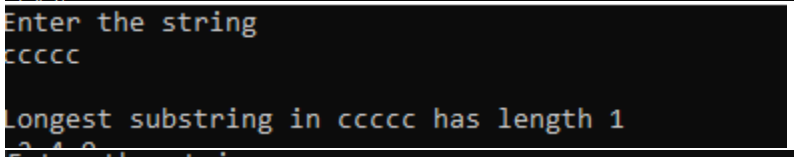
Please zip all relevant files and email back by Monday, June 22, 1700 PST

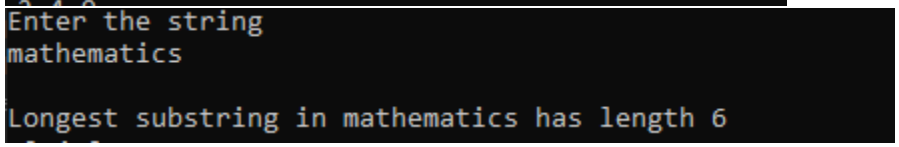
Part I: For the 4 questions below, please document your assumptions, test cases, and most importantly your thought process as you work through the problems. As an example, imagine that not only do you have to solve the problem, but you also have to explain the logic and concepts to a more junior colleague of yours.

1. Question: How do you find the length of the longest substring without repeating characters?
   Input: "pqqkew"
   Output: 3
   Input: "ccccc"
   Output: 1

   Assumptions :
   Given the test case “pqqkew”: the two unique substrings which I determined for this are “pq” and “qkew”. Hence the longest substring has output 4 but not 3.

   Test cases :





String	Longest Substring	Length by algorithm
pqqkew	qkew	4
ccccc	c	1
mathematics	matics	6

All test cases are verified.

Breakdown of logic and algorithm:

To calculate the longest substring in a string with no repeating characters

- Approach was to move forward character by character of a string and check all the characters left behind
- If there is a match of a subsequent character with any character behind, then we break the string using substr function till that matching character and determine length of that substring.
- Now we repeat the process with the new character after the extracted substring
- Take out largest length and display.

For Instance: if the string is "a b d a d f"
Then we will have pointer 1 at first place i.e 'a'. pointer 2 at second place i.e 'b'

1) b will be checked against a. No match then pointer 2 will move to 'd'.
2) pointer 2 will backtrace and check b and a. No match then pointer 2 moves to 'a'.
3) pointer 2 will backtrace and check 'd', 'b' and 'a'. there is a match
4) substring till 4th place a i.e "abd" is taken out. Length is determined.
5) Pointer 1 is changed to now 4th place 'a' (position of match) and pointer 2 to one position after that. i.e 'd'.
   The whole process from 1 repeat but for subsequent characters till end of string

   The substring with largest length wins.

Algorithm:

1) transform string to character array
- initialize counter =0
- initialize lengths to 0
  2) Repeat till end
- outer loop i till end of the character array
- inner loop from j= i-1 decrementing till counter
- check for a match. If match found : l = i – counter(we need l for substring function in c#)
- new length = length of substring from counter till l
- change counter, counter = i
- compare length to keep largest length.
- Break inner loop
- Go to step 2.

If i has traversed the full character array take out the tail end substring with no repeating characters and compare length.

Return length

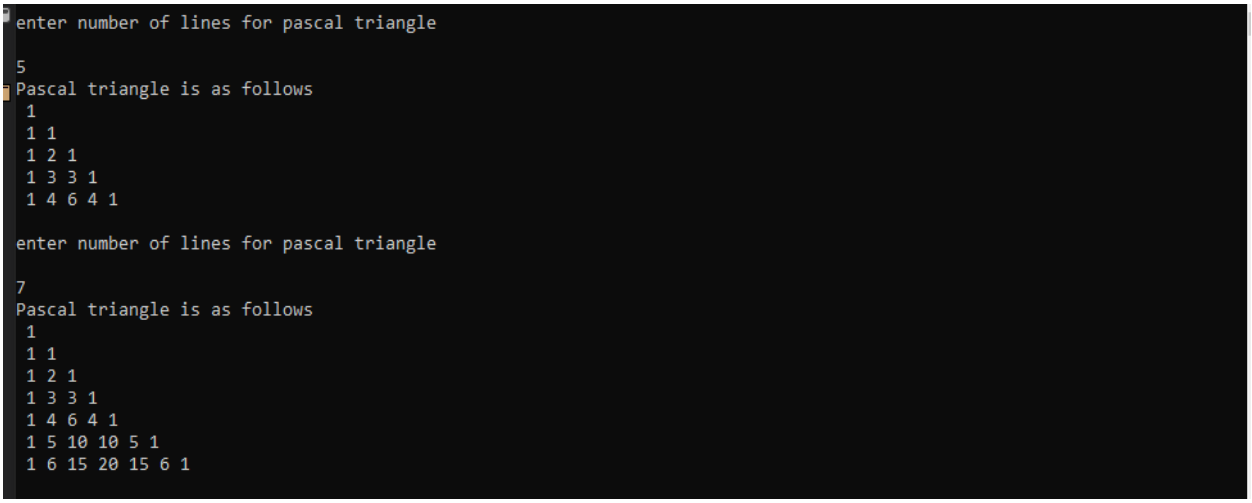
Source Code :

```
public static int LongestSubstring(String s)
        {
            int counter = 0;
           //converting string to character array
            char[] char_array = s.ToCharArray();

            //initializing length
            int length = 0;
            int new_length = 0;
            int i = 1;

            for ( i = 1; i < char_array.Length; i++)
            {  // inner loop decrements till counter
                for (int j = i - 1; j >= counter; j--)
                {
                    // if match is found
                    if (s[i] == s[j])
                    {
  // we need l for substring function to know till what length are  we extracting string\

                        int l = i - counter;
                        // length of substring till counter to l
                        new_length = s.Substring(counter,l ).Length;
 /*for instance if pqqkew is string then we got the substring till  pq
 now new counter is at q of qkew and i will move to k and backtracing will restart*/

                        //new counter
                        counter = i;
                        if (new_length > length)
                        {
                            length = new_length;
                        }

                        break;
                    }

                }

            }
            //tail end substring length match
            // our loop will break at end and would not be able to determine length at
the very end
            if (i == char_array.Length)
            {
                int w = i - counter;
                int last = s.Substring(counter, w).Length;
                if (last > length)
                {
                    length = last;
                }
```

```
        }
        return length;

    }
```

2. Question: How to print Pascal's triangle?
   1
   1 1
   1 2 1
   1 3 3 1
   1 4 6 4 1
   1 5 10 10 5 1

Test Cases:



Above pascal triangles for 5 lines and 7 lines. All test cases are verified

Breakdown of logic and algorithm:

pascal triangle is as :

1
11
121
1331
14641

- As we can see each element below is just an addition of its immediate numbers above with 1 in the corners. I will create an algorithm which does the same

for instance : line 2 and 3

11

121

we can see how 1+1 in line 2 is giving 2 to the line 3 surrounded by one

similarly.

121

1331

we can see 1+1 and 2+1 in line 3 is giving -> 3 and 3 surrounded by 1 in line 4.

Algorithm:

We are solving pascal triangle using multidimensional array as data structure

1) create two dimensional array of dimensions (linesize * linesize)

2) fill elements in array such that corners are 1 and middle elements are addition of immediate upper line numbers

3) for instance, line 3 and 4 of pascals triangle is

121

1331

so array addition for line 3 which is derived from line 2 or array with 2nd line

, array[3][1] =1

[3][2] = [2][0] + [2][1]

[3][3] = [2][1] + [2][2]

[3][3] = 1

operation to perform similarly for each line

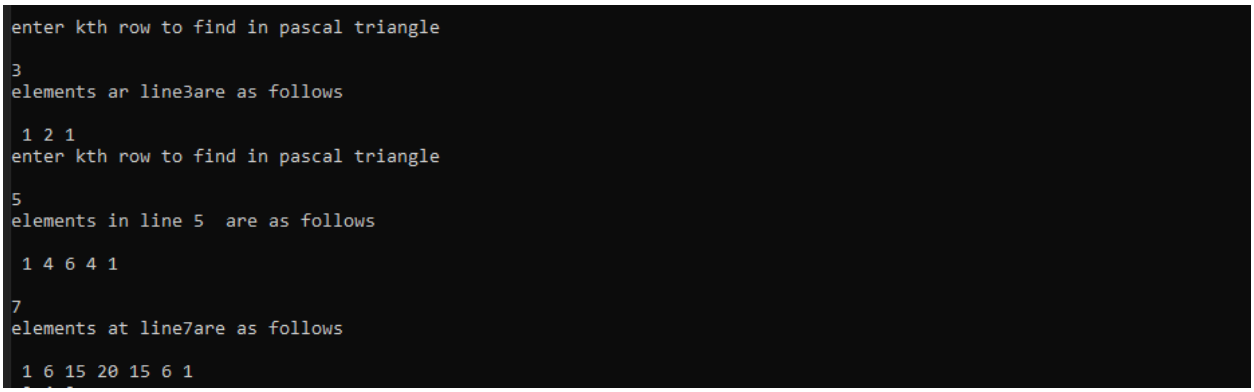
Source Code:

```
    public static void PascalTriangle(int numrows)
 {

     //creating a 2 dimensional array of size numrows
     int[,] array = new int[numrows,numrows];

     for (int i = 0; i < numrows; i++) {
         for (int j = 0; j <= i; j++) {
             /*logic for implementing corners as 1 and middle
             elements as addition of immediate upper elements*/
             if (j == 0 || j == i)
             {
                 array[i, j] = 1;
             }
             else {
             // if its not corners then array[i,j] is addition of upper elements
in the triangles
                 array[i, j] = array[i - 1, j-1] + array[i - 1, j];
             }
         }
```

```
            }
            //printing the array that results in a pascal triangle
            for (int i = 0; i < numrows; i++) {
                for (int j = 0; j <= i; j++) {
                    Console.Write(" " + array[i,j]);
                }
                Console.WriteLine(" ");
            }
            }
```

3. Question: Find the Kth Row of Pascal's Triangle
   Input : k = 4

Test Cases :



Above test cases gave

Elements at line 3 : 1 2 1

Elements at line 5 : 1 4 6 4 1

Elements at line 7 : 1 6 15 20 15 6 1

All test cases verified

Breakdown and Logic:

To find the kth row from pascal triangle:

* I did not want to use the exact code as above and print the last line.
* Made some code and logic changes to accomplish the goal

At every iteration new line is created from its upper line. As new line is addition of its immediate numbers in above line with 1 in corners.
Repeated the iteration till kth line and printed out kth line.

Algorithm :

* created two single dimensional arrays with size k.

Repeat till kth line
* one of them being temporary array to compute the current line for pascals triangle
*Main_array stores the previous line computed. We get values in temporary array from addition of immediate elements in main_array with 1 in the corner

For instance:
If 1 2 1 (line 3) is the main array then temp_array (line 4) is created as 1 in the corners and addition of 1+2 and 2+1 from main array. i.e line 4 -> 1 3 3 1

* copied temporary array to main array for each iteration . : now temp_array becomes main array to solve upcoming line

- Repeated till k times.

If k is reached : print main_array

Source Code:

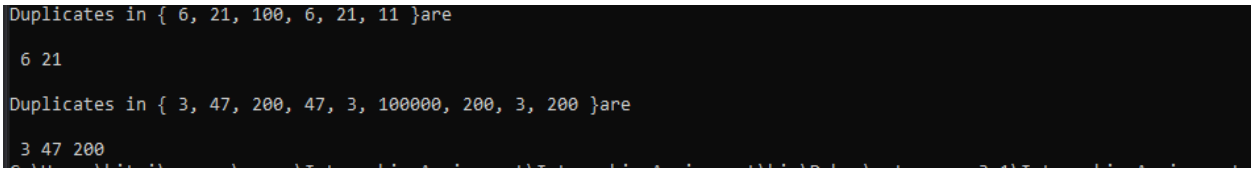
```
public static void FindPascalRow(int k) {

            int[] temp = new int[k];
            int[] main_array = new int[k];

            // outer loop denotes each line
            for (int i = 0; i < k; i++) {
                // inner loop denotes values in a line
                for (int j = 0; j <= i; j++) {
                    // code to convert corner values to 1
                    if (j == 0 || j == i)
                    {
                        temp[j] = 1;
                    }
                    else {
                        // temp array includes values in that line which were computed
from previous line
                        temp[j] = main_array[j-1] + main_array[j];
                    }
                }
                // temp array is copied to main array, to compute new line values from
this main array.
                //temp array now becomes the main array to solve next iteration.
                temp.CopyTo(main_array, 0);
            }
            //printing values of kth line
            for (int i = 0; i < main_array.Length; i++) {
```

```
                Console.Write(" " + main_array[i]);
            }


        }
    }
}
```

4. Question: How do you find duplicate numbers in an array if it contains multiple duplicates?
   Input : [6, 21, 100, 6, 21, 11]
   Output : 6 21

   Input : [3, 47, 200, 47, 3, 100000, 200, 3, 200]
   Output : 3 47 200

Test cases:



All test cases are verified

Breakdown and logic:
I wanted to complete this in O(n) complexity, so I used Dictionary as a data structure to solve this problem.

Dictionary stores the values in key and value format such that for each key there is a value associated. Value can be accessed using that key.

Algorithm

- Created a Dictionary
- Iterated through each element in the array.
- If that element is not the key in the dictionary, then that element is made a key with value associated with 1.

- On Iteration, if element in array is already a key in the dictionary then we will increment that key's value such that value is incremented by 1.
- After all elements are iterated print all those keys whose values are greater than 1.

\

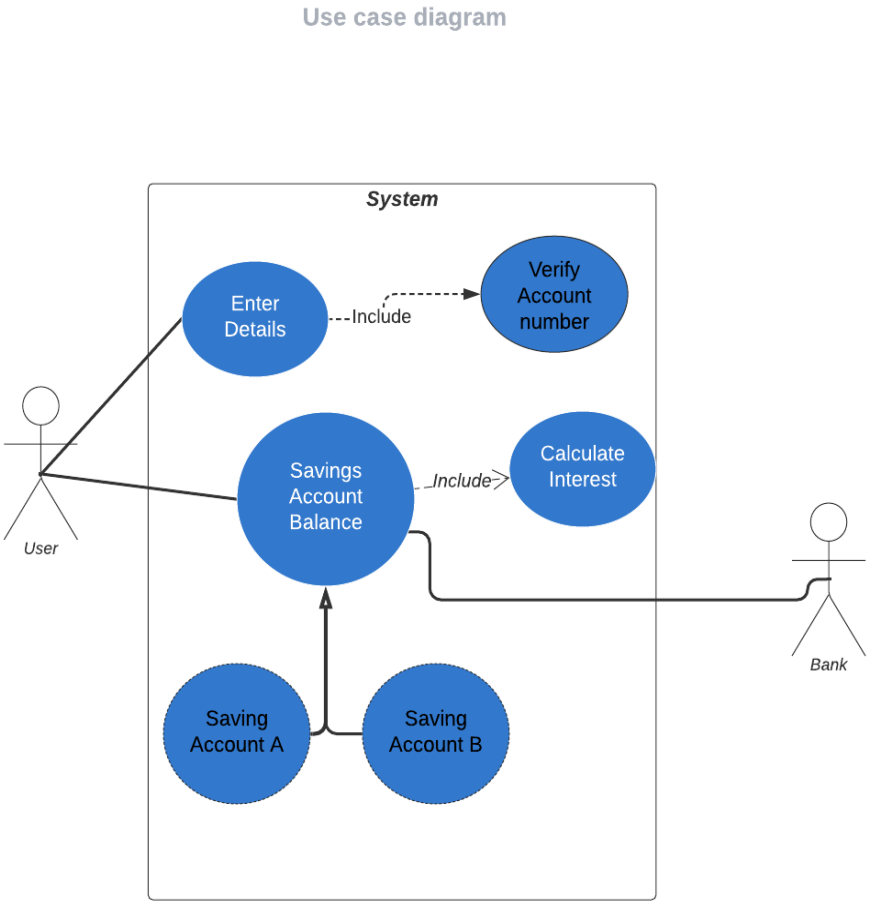
Source Code

```
public static void ContainsDuplicate(int[] arr) {


//created a dictionary with key and values as integers
            Dictionary<int, int> myDict = new Dictionary<int, int>();
            //initialized value to 1
            int value = 1;
            // iterated through each element of the array
            for (int i = 0; i < arr.Length; i++) {
                 /*if dictionary does not have that element has
                the key push that element as the key with value as 1*/
                if (!myDict.ContainsKey(arr[i]))
                {
                    myDict.Add(arr[i], value);
                }
                else {
                    // else if key as already present then increment value of 1 that
denotes count of that element
                    myDict[arr[i]] = value + 1;
                }

            }

            //print keys with values greater than 1
            foreach (KeyValuePair<int, int> item in myDict)
            {
                if (item.Value > 1)
                {
                    Console.Write(" " + item.Key);
                }
            }

           }
```

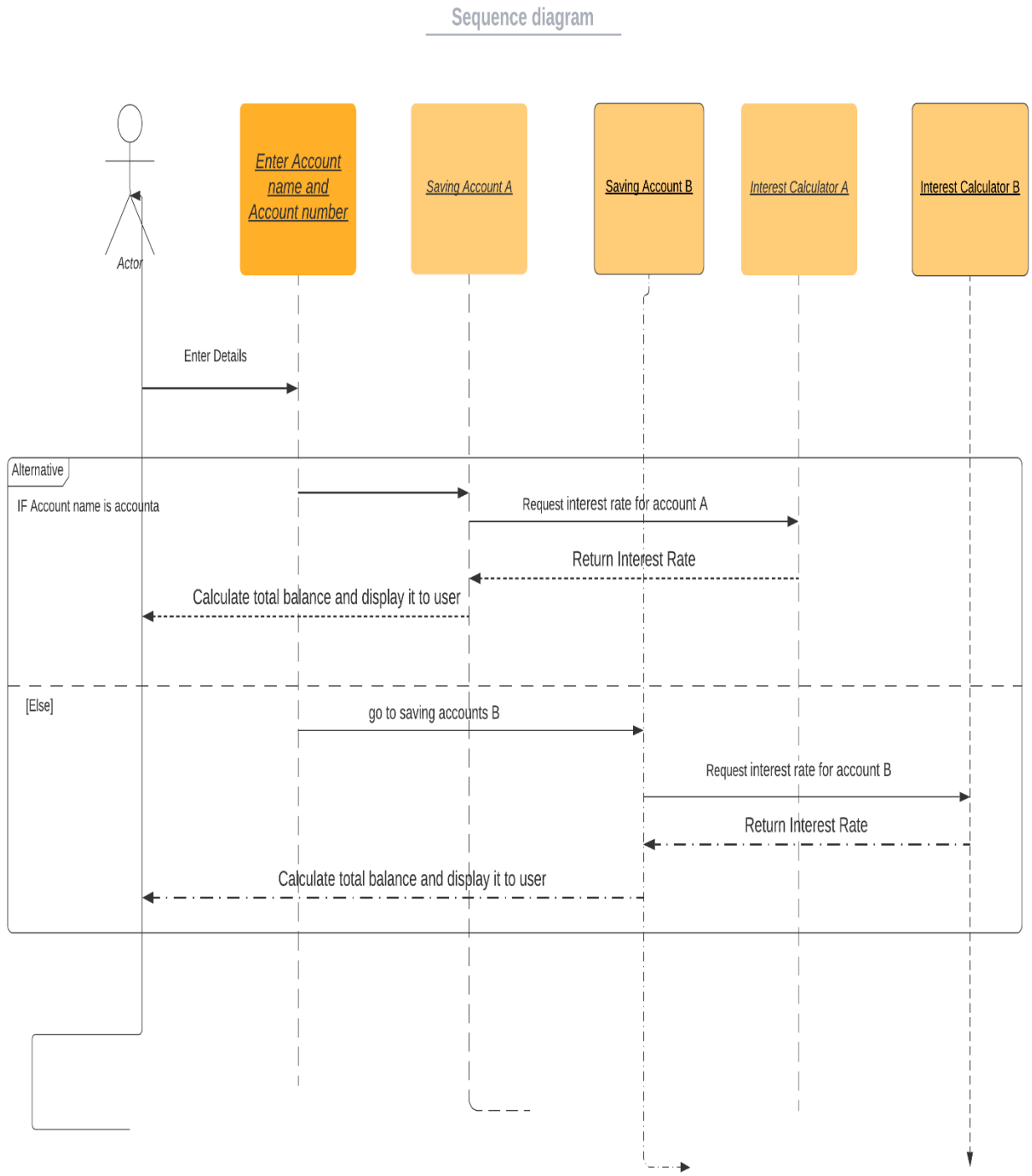
Part II:

5. Use the following UML Diagrams to document the source code below.

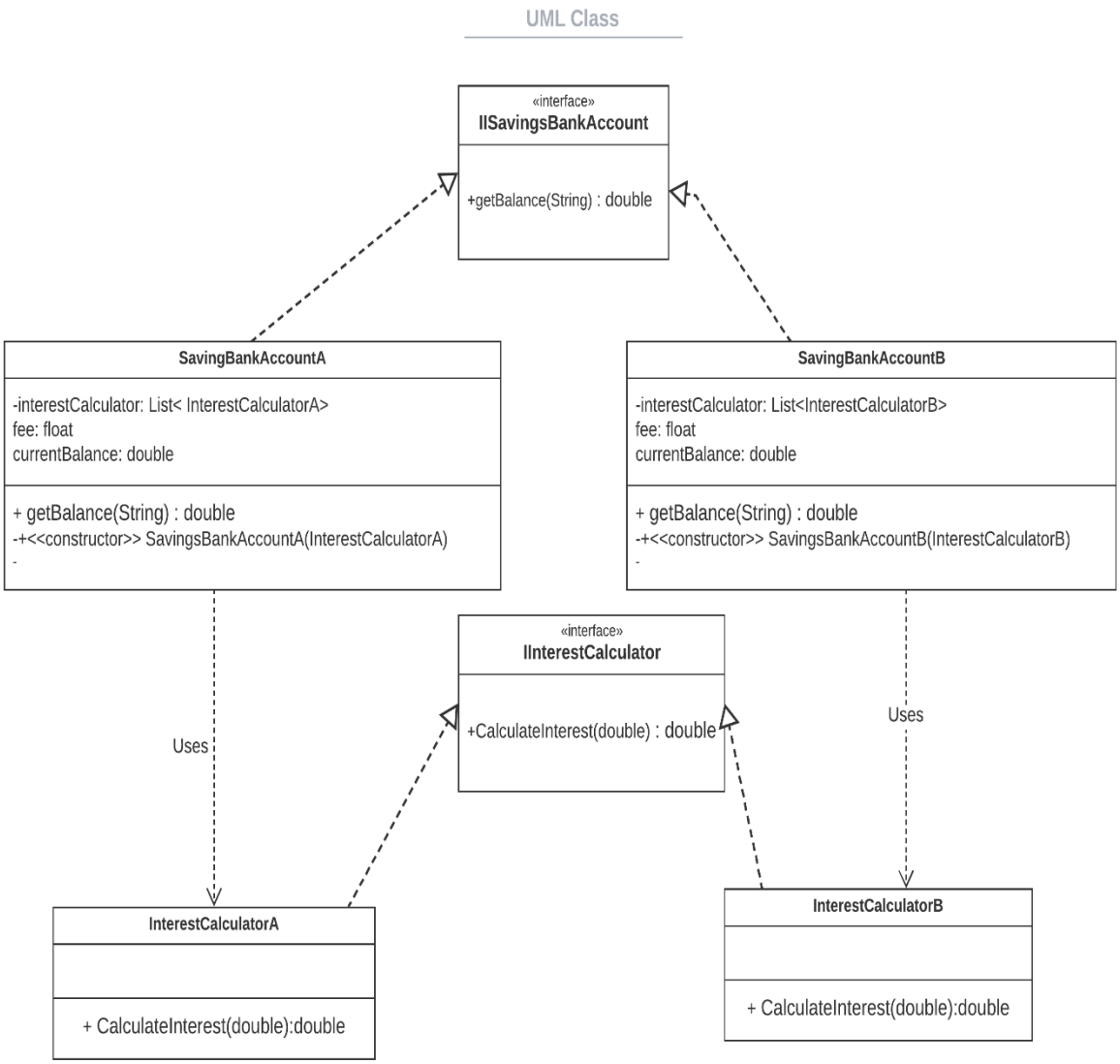
- Use Case



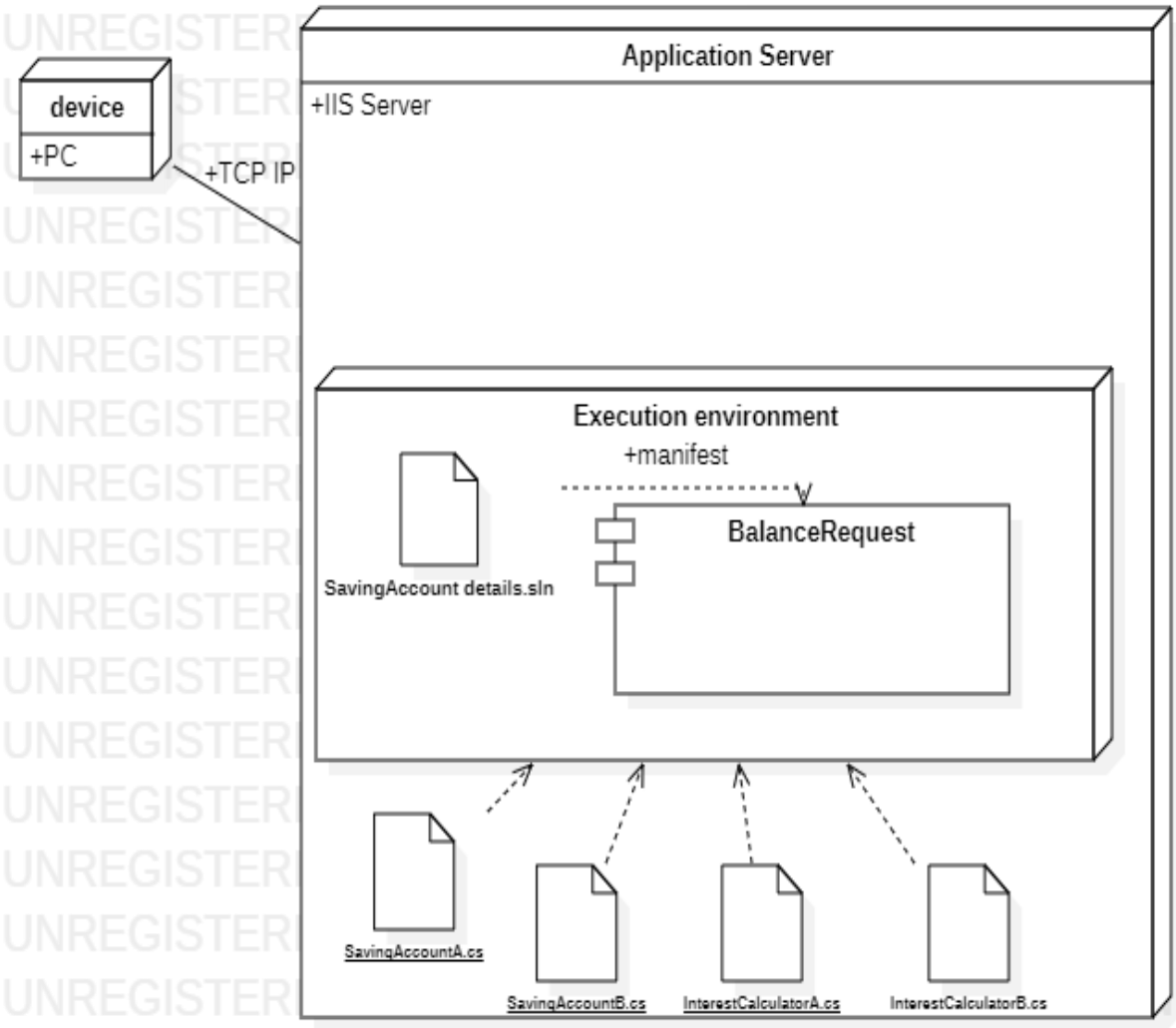
- Sequence


Sequence diagram
Actor
Enter Account name and Account number
Saving Account A
Saving Account B
Interest Calculator A
Interest Calculator B
Enter Details
Alternative
IF Account name is accounta
Request interest rate for account A
Return Interest Rate
Calculate total balance and display it to user
[Else]
go to saving accounts B
Request interest rate for account B
Return Interest Rate
Calculate total balance and display it to user

- Class


UML Class
«interface»
IISavingsBankAccount
+getBalance(String) : double
SavingBankAccountA
-interestCalculator: List< InterestCalculatorA>
fee: float
currentBalance: double
+ getBalance(String) : double
-+<<constructor>> SavingsBankAccountA(InterestCalculatorA)
-
SavingBankAccountB
-interestCalculator: List<InterestCalculatorB>
fee: float
currentBalance: double
+ getBalance(String) : double
-+<<constructor>> SavingsBankAccountB(InterestCalculatorB)
-
«interface»
IInterestCalculator
+CalculateInterest(double) : double
Uses
Uses
InterestCalculatorA
+ CalculateInterest(double):double
InterestCalculatorB
+ CalculateInterest(double):double

- Deployment


Application Server
+IIS Server
device
+PC
+TCP IP
Execution environment
+manifest
BalanceRequest
SavingAccount details.sln
SavingAccountA.cs
SavingAccountB.cs
InterestCalculatorA.cs
InterestCalculatorB.cs

Program.cs

```
public static void Main(string[] args)
{
            Console.WriteLine("Enter Savings Bank Account Name: ");
            string savingsbankAccountName = Console.ReadLine();

            Console.WriteLine("Enter Savings Bank Account Number: ");
            string savingsbankAccountNumber = Console.ReadLine();

            if (savingsbankAccountName.ToLower().Equals("savingsbankaccounta"))
            {
                IInterestCalculator<InterestCalculatorB> interestCalculatorB = new
InterestCalculatorB();
                SavingsBankAccountB savingsBankAccountB = new
SavingsBankAccountB(interestCalculatorB);
                Console.WriteLine($"Balance for {savingsbankAccountName} is
{savingsBankAccountB.GetBalance(savingsbankAccountNumber)}");
            }
            else
            {
                IInterestCalculator<InterestCalculatorA> interestCalculatorB = new
InterestCalculatorA();
                SavingsBankAccountA savingsBankAccountA = new
SavingsBankAccountA(interestCalculatorB);
                Console.WriteLine($"Balance for {savingsbankAccountName} is
{savingsBankAccountA.GetBalance(savingsbankAccountNumber)}");

            }
        }
```

SavingsBankAccountA.cs

```
public class SavingsBankAccountA : ISavingsBankAccount<InterestCalculatorA>
    {
        private IInterestCalculator<InterestCalculatorA> interestCalculator;
        public SavingsBankAccountA(IInterestCalculator<InterestCalculatorA>
interestCalculator)
        {
            this.interestCalculator = interestCalculator;
        }

        public double GetBalance(string accountNumber)
        {
            float fee = 5;
            double currentBalance = 2000;

            double actualBalance = currentBalance +
this.interestCalculator.CalculateInterest(currentBalance) - fee;
            return actualBalance;
        }


    }
```

IInterestCalculator.cs

```
public interface IInterestCalculator<T>
    {
        public double CalculateInterest(double amount);
    }
```

InterestCalculatorA.cs

```
public class InterestCalculatorA : IInterestCalculator<InterestCalculatorA>
    {
        public double CalculateInterest(double amount)
        {
            return amount * 0.2;
        }
    }
```

ISavingsBankAccount.cs

```
public interface ISavingsBankAccount<T>
    {
        public double GetBalance(string accountNumber);
    }
```

InterestCalculatorB.cs

```
public class InterestCalculatorB : IInterestCalculator<InterestCalculatorB>
    {
        public double CalculateInterest(double amount)
        {
            return amount * 500;
        }
    }
```

SavingsBankAccountB.cs

```
public class SavingsBankAccountB : ISavingsBankAccount<InterestCalculatorB>
    {
        private IInterestCalculator<InterestCalculatorB> interestCalculator;
        public SavingsBankAccountB(IInterestCalculator<InterestCalculatorB>
interestCalculator)
        {
            this.interestCalculator = interestCalculator;
        }

        public double GetBalance(string accountNumber)
        {
            float fee = 1000;
            double currentBalance = 10000;

            double actualBalance = currentBalance +
this.interestCalculator.CalculateInterest(currentBalance) - fee;
            return actualBalance;
```

```
        }

    }
```

6. Come up with test cases for **InterestCalculatorB.cs** and **SavingsBankAccountB.cs**

Test Scenario	Test Case	Test Data	Test Response
Check valid account number and name	Check response	Account name = “savingsaccounta”	Success

Test Scenario	Test Case	Test Data	Test Response
Check Calculated Balance	Verify correct balance	Account name = “Savingsaccounta”	5009000

InterestCalculatorB.cs

Test Scenario	Test Case	Test Data	Test Response
Verify interest rate	Determine if interest rate add up is correct	Accountname = “Savingsaccounta” Balance = 1000 Balance = 10000 Balance = 100000	 500000 5000000 50000000