Coding test .NET

Please write a small program for each of the exercises

**1. Denomination routine**

An ATM has three cartridges for different denominations:

* 10 EUR cartridge
* 50 EUR cartridge
* 100 EUR cartridge

Now we want to pay out following amounts from the ATM:

* 30 EUR
* 50 EUR
* 60 EUR
* 80 EUR
* 140 EUR
* 230 EUR
* 370 EUR
* 610 EUR
* 980 EUR

Write a program which will calculate for each payout the possible combinations which the ATM can pay out.

For example, for 100 EUR the available payout denominations would be:

* 10 x 10 EUR
* 1 x 50 EUR + 5 x 10 EUR
* 2 x 50 EUR
* 1 x 100 EUR

**My solution:** The following program will calculate all the possible combinations which the ATM can payout desired Amount.

**Code:**

using System;

using System.Collections;

using System.Collections.Generic;

namespace ConsoleApp3

{

class ClassATMDenominationCalculator

{

public static void Main()

{

int[] euroSizes = new int[] { 100, 50, 10 }; // the denomination you want to find.

int[,] resultArray = new int[1000, 3]; //the total EURO payment combinations

int payOutAmount = 0;

Console.WriteLine("Enter the Payout Amount ");

payOutAmount = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Entered Payout amount is " + payOutAmount);

// int total = 980; // Test data

int divHundred = payOutAmount / 100; //2

int divFifty = payOutAmount / 50; //4

int divTen = payOutAmount / 10; //28

int RowCount = 0;

Console.WriteLine("The possible combinations of Payout is\n ");

for (int i = divHundred; i >= 0; i--)

{

int newtotal = payOutAmount - divHundred \* 100;

int RowIndex = (newtotal / 50);

for (int j = 0; j < RowIndex + 1; j++)

{

string result100 = "", result50 = "", result10 = "";

resultArray[RowCount, 0] = divHundred;

resultArray[RowCount, 1] = j;

resultArray[RowCount, 2] = (newtotal - (j \* 50)) / 10;

if (resultArray[RowCount, 0] > 0)

{

result100 = resultArray[RowCount, 0] + " \* 100 EUR +";

}

if (resultArray[RowCount, 1] > 0)

{

result50 = resultArray[RowCount, 1] + " \* 50 EUR +";

}

if (resultArray[RowCount, 2] > 0)

{

result10 = resultArray[RowCount, 2] + " \* 10 EUR";

}

string resultFinal = result100 + result50 + result10;

Console.WriteLine(resultFinal.Trim('+'));

RowCount++;

}

divHundred--;

}

}

}

}

**Output:** Below is the output for payout amounts

* 30 EUR
* 50 EUR
* 60 EUR
* 80 EUR
* 140 EUR
* 230 EUR

**Likewise, we can give any amount in the console input and fetch the desired output:**

Enter the Payout Amount

30

Entered Payout amount is 30

The possible combinations of Payout is

3 \* 10 EUR

Enter the Payout Amount

50

Entered Payout amount is 50

The possible combinations of Payout is

5 \* 10 EUR

1 \* 50 EUR

Enter the Payout Amount

60

Entered Payout amount is 60

The possible combinations of Payout is

6 \* 10 EUR

1 \* 50 EUR +1 \* 10 EUR

Enter the Payout Amount

80

Entered Payout amount is 80

The possible combinations of Payout is

8 \* 10 EUR

1 \* 50 EUR +3 \* 10 EUR

Enter the Payout Amount

140

Entered Payout amount is 140

The possible combinations of Payout is

1 \* 100 EUR +4 \* 10 EUR

14 \* 10 EUR

1 \* 50 EUR +9 \* 10 EUR

2 \* 50 EUR +4 \* 10 EUR

Enter the Payout Amount

230

Entered Payout amount is 230

The possible combinations of Payout is

2 \* 100 EUR +3 \* 10 EUR

1 \* 100 EUR +13 \* 10 EUR

1 \* 100 EUR +1 \* 50 EUR +8 \* 10 EUR

1 \* 100 EUR +2 \* 50 EUR +3 \* 10 EUR

23 \* 10 EUR

1 \* 50 EUR +18 \* 10 EUR

2 \* 50 EUR +13 \* 10 EUR

3 \* 50 EUR +8 \* 10 EUR

4 \* 50 EUR +3 \* 10 EUR

**2. REST server**

A small REST server with good performance for simple customer management has two functions:

* POST customers

Request:

[

{

firstName: 'Aaaa',

lastName: 'Bbbb',

age: 20,

id: 5

},

{

firstName: 'Bbbb',

lastName: 'Cccc',

age: 24,

id: 6

}

]

Multiple customers can be sent in one request.

The server validates every customer of the request:

* checks that every field is supplied
* validates that the age is above 18
* validates that the ID has not been used before

The server then adds each customer as an object to an internal array – the customers will not be appended to the array but instead it will be inserted at a position so that the customers are sorted by last name and then first name WITHOUT using any available sorting functionality (an example for the inserting is in the Appendix).

The server also persists the array so it will be still available after a restart of the server.

* GET customers

Returns the array of customers with all fields

Write the server and a small simulator which can send several requests for POST customers and GET customers in parallel to the server.

For that program it is not allowed to use any sorting mechanism like array.sort().

The simulated POST customers requests have following requirements:

* Each request should contain at least 2 different customers
* Age should be randomized between 10 and 90
* ID should be increasing sequentially.
* The first names and last names of the Appendix should be used in random combinations

**Appendix:**

**Data:**

First names: Last names:

Leia Liberty

Sadie Ray

Jose Harrison

Sara Ronan

Frank Drew

Dewey Powell

Tomas Larsen

Joel Chan

Lukas Anderson

Carlos Lane

**Example for the inserting mechanism:**

Array in server:

[

{ lastName: 'Aaaa', firstName: 'Aaaa', age: 20, id: 3 },

{ lastName: 'Aaaa', firstName: 'Bbbb', age: 56, id: 2 },

{ lastName: 'Cccc', firstName: 'Aaaa', age: 32, id: 5 },

{ lastName: 'Cccc', firstName: 'Bbbb', age: 50, id: 1 },

{ lastName: 'Dddd', firstName: 'Aaaa', age: 70, id: 4 },

]

Request POST customers:

[{ lastName: 'Bbbb', firstName: 'Bbbb', age: 26, id: 6 }]

Array after insert:

[

{ lastName: 'Aaaa', firstName: 'Aaaa', age: 20, id: 3 },

{ lastName: 'Aaaa', firstName: 'Bbbb', age: 56, id: 2 },

{ lastName: 'Bbbb', firstName: 'Bbbb', age: 26, id: 6 },

{ lastName: 'Cccc', firstName: 'Aaaa', age: 32, id: 5 },

{ lastName: 'Cccc', firstName: 'Bbbb', age: 50, id: 1 },

{ lastName: 'Dddd', firstName: 'Aaaa', age: 70, id: 4 },

]

Request POST customers:

[{ lastName: 'Bbbb', firstName: 'Aaaa', age: 28, id: 7 }]

Array after insert:

[

{ lastName: 'Aaaa', firstName: 'Aaaa', age: 20, id: 3 },

{ lastName: 'Aaaa', firstName: 'Bbbb', age: 56, id: 2 },

{ lastName: 'Bbbb', firstName: 'Aaaa', age: 28, id: 7 },

{ lastName: 'Bbbb', firstName: 'Bbbb', age: 26, id: 6 },

{ lastName: 'Cccc', firstName: 'Aaaa', age: 32, id: 5 },

{ lastName: 'Cccc', firstName: 'Bbbb', age: 50, id: 1 },

{ lastName: 'Dddd', firstName: 'Aaaa', age: 70, id: 4 },

]

**My solution:** Please find the below program to create a REST service and console application to consume the service.

**REST API for GetCustomers() and PostCustomer(Customer[] newcustomers) methods.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace SampleProgramTest.Controllers

{

public class CustomerController : ApiController

{

List<Customer> customers = new List<Customer>

{

new Customer { firstName = "Sadie", lastName = "Ray", age = 20 ,id=1},

new Customer { firstName = "Jose", lastName = "Ronan", age = 25 ,id=2 },

new Customer { firstName = "Frank", lastName = "Larsen", age = 30 ,id=3 },

new Customer { firstName = "Joel", lastName = "Harrison", age = 22 ,id=4},

new Customer { firstName = "Dewey", lastName = "Powell", age = 80 ,id=5 },

new Customer { firstName = "Tomas", lastName = "Lane", age = 60 ,id=6 },

new Customer { firstName = "Leia", lastName = "Liberty", age = 45 ,id=7},

new Customer { firstName = "Dewey", lastName = "Anderson", age = 54 ,id=8 },

new Customer { firstName = "Lukas", lastName = "Chan", age = 80 ,id=9 },

new Customer { firstName = "Carlos", lastName = "Lane", age = 72 ,id=10 },

new Customer { firstName = "Sara", lastName = "Lane", age = 29 ,id=11 },

};

// GET: api/Customer

public Customer[] GetCustomers()

{

Customer c = new Customer();

return c.SortCustomer(customers);

}

// GET: api/Customer/5

public string Get(int id)

{

return "value";

}

// POST: api/Customer

public Customer[] PostCustomer(Customer[] newcustomers)

{

Customer cust = new Customer();

foreach (var c in newcustomers)

{

bool isValid=cust.ValidateCustomer(c, customers);

if (isValid)

{

customers.Add(new Customer()

{

firstName = c.firstName,

lastName = c.lastName,

age = c.age,

id = c.id,

});

}

}

return cust.SortCustomer(customers);

}

// PUT: api/Customer/5

public void Put(int id, [FromBody]string value)

{

}

// DELETE: api/Customer/5

public void Delete(int id)

{

}

}

public class Customer

{

public string lastName { get; set; }

public string firstName { get; set; }

public int age { get; set; }

public int id { get; set; }

public Customer[] SortCustomer(List<Customer> customers)

{

Customer[] customerArray = customers.ToArray();

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

{

for (int j = 0; j < customerArray.Length - 1; j++)

{

if (customerArray[j].lastName.CompareTo(customerArray[j + 1].lastName) > 0)

{

var temp = customerArray[j];

customerArray[j] = customerArray[j + 1];

customerArray[j + 1] = temp;

}

else if (customerArray[j].lastName.CompareTo(customerArray[j + 1].lastName) == 0)

if (customerArray[j].firstName.CompareTo(customerArray[j + 1].firstName) > 0)

{

var temp = customerArray[j];

customerArray[j] = customerArray[j + 1];

customerArray[j + 1] = temp;

}

}

}

return customerArray;

}

public bool ValidateCustomer(Customer newCustomer,List<Customer> customers)

{

bool isValid=true;

//checks all the required fields

if(newCustomer.firstName=="" || newCustomer.firstName == "" || newCustomer.age.ToString() == "" || newCustomer.id.ToString() == "")

isValid = false;

//check age is above 18

if(newCustomer.age < 18)

isValid = false;

if (newCustomer.id > 0)

{

foreach (var c in customers)

{

if (newCustomer.id == c.id)

isValid = false;

}

}

return isValid;

}

}

}

**Console Application to consume the REST API:**

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Net.Http.Headers;

namespace ConsoleApp2

{

class Program

{

static void Main(string[] args)

{

ConsumeAPI cAPI = new ConsumeAPI();

cAPI.GetAllCustomers(); //Get Customers

cAPI.PostCustomer(); //Add Customers

}

}

class ConsumeAPI

{

public void GetAllCustomers()

{

using (var client = new WebClient()) //WebClient

{

client.Headers.Add("Content-Type:application/json"); //Content-Type

client.Headers.Add("Accept:application/json");

var result = client.DownloadString("https://localhost:44394/api/customer"); //URI

Console.WriteLine(Environment.NewLine + result);

}

}

public void PostCustomer() //Adding Event

{

using (var client = new WebClient())

{

List<Customer> customerList = new List<Customer> {

new Customer { firstName = "Lukas", lastName = "Lane", age = 10 ,id=12},

new Customer { firstName = "Jose", lastName = "Lane", age = 25 ,id=13 },

};

client.Headers.Add("Content-Type:application/json");

client.Headers.Add("Accept:application/json");

var result = client.UploadString("https://localhost:44394/api/customer", JsonConvert.SerializeObject(customerList.ToArray()));

Console.WriteLine(result);

}

}

}

}

public class Customer

{

public string firstName { get; set; }

public string lastName { get; set; }

public int age { get; set; }

public int id { get; set; }

}

**Sample Output:**

[

{"lastName":"Anderson","firstName":"Dewey","age":54,"id":8},

{"lastName":"Chan","firstName":"Lukas","age":80,"id":9},

{"lastName":"Harrison","firstName":"Joel","age":22,"id":4},

{"lastName":"Lane","firstName":"Carlos","age":72,"id":10},

{"lastName":"Lane","firstName":"Sara","age":29,"id":11},

{"lastName":"Lane","firstName":"Tomas","age":60,"id":6},

{"lastName":"Larsen","firstName":"Frank","age":30,"id":3},

{"lastName":"Liberty","firstName":"Leia","age":45,"id":7},

{"lastName":"Powell","firstName":"Dewey","age":80,"id":5},

{"lastName":"Ray","firstName":"Sadie","age":20,"id":1},

{"lastName":"Ronan","firstName":"Jose","age":25,"id":2}

]

[

{"lastName":"Anderson","firstName":"Dewey","age":54,"id":8},

{"lastName":"Chan","firstName":"Lukas","age":80,"id":9},

{"lastName":"Harrison","firstName":"Joel","age":22,"id":4},

{"lastName":"Lane","firstName":"Carlos","age":72,"id":10},

{"lastName":"Lane","firstName":"Jose","age":25,"id":13},

{"lastName":"Lane","firstName":"Sara","age":29,"id":11},

{"lastName":"Lane","firstName":"Tomas","age":60,"id":6},

{"lastName":"Larsen","firstName":"Frank","age":30,"id":3},

{"lastName":"Liberty","firstName":"Leia","age":45,"id":7},

{"lastName":"Powell","firstName":"Dewey","age":80,"id":5},

{"lastName":"Ray","firstName":"Sadie","age":20,"id":1},

{"lastName":"Ronan","firstName":"Jose","age":25,"id":2}

]

**The solution file for the ATM payout calculator is zipped and attached below.**



**The REST service and consuming console application are zipped and attached below.**

