1. Movie Library

hackerrank.com/work/tests/1649036/library/hackerrank?copyscrape=true&hide_ai_solvable=true&question=1538642&skill_ids=6c914bbddf85...

Tests > Capgemini Back-End Developer Hiring Test > Library - HackerRank

Hackerrank Questions | My Company Questions | Go back to test

14 QUESTIONS: Status Not Leaked; Ai Protection On; Skills C# (Basic), C# (Intermediate); Type Coding

C#: Movie Library EASY — Coding 30 mins

C#: Movie Library

Recommended Time: 30 mins | Points: 50 | 13 test cases (3 samples)

Skills: C# (Basic)

Coding | EASY | C# | Coding | Inheritance | Interfaces

Languages: C#

A movie library application is being developed. This app has methods to maintain movie listings, with functionality to add movies, remove movies, and search for movies.

Create a new class called *Film* and implement the *IFilm* interface.

- Inside the *Film* class, define the following properties:
  - *Title* (string): the title of the film

© 2023 HackerRank | Privacy policy | Terms of service | API | Support | Status | Academy

Create a new class called *Film* and implement the *IFilm* interface.

- Inside the *Film* class, define the following properties:
  - *Title* (string): the title of the film
  - *Director* (string): the director of the film
  - *Year* (integer): the year the film was released

Create another class called *FilmLibrary* and implement the *IFilmLibrary* interface.

- Inside the *FilmLibrary* class, declare a private field called *_films* of type *List<IFilm>* to store the films.
- Add the following methods to the *FilmLibrary* class:
  - *AddFilm(IFilm film):* adds a film to the film library. It takes an *IFilm* object as a parameter and adds it to the *_films* list
  - *RemoveFilm(string title):* removes a film from the film library based on its title if it is in the *_films*

© 2023 HackerRank | Privacy policy | Terms of service | API | Support | Status | Academy

- *RemoveFilm(string title):* removes a film from the film library based on its title if it is in the *_films*
- *GetFilms():* returns a list of all films in the film library
- *SearchFilms(string query):* searches for films in the film library based on a query string. It returns a list of films whose title or director contains the query
- *GetTotalFilmCount():* returns the total number of films in the film library

Example

There are 2 Film objects, with Title, Director, Year.

© 2023 HackerRank | Privacy policy | Terms of service | API | Support | Status | Academy

2. Real Estate Listing Management

hackerrank.com/work/tests/1649036/library/hackerrank?copyscrape=true&hide_ai_solvable=true&question=1538387&skill_ids=6c914bbd...

Tests > Capgemini Back-End Developer Hiring Test > Library - HackerRank

Hackerrank Questions | My Company Questions | Loading... | Go back to test

C#: Movie Library EASY — Coding 30 mins

C#: Real Estate Listing Management EASY — Coding 30 mins

C#: Real Estate Listing Management

Recommended Time: 30 mins | Points: 50 | 13 test cases (3 samples)

Skills: C# (Basic)

Coding | EASY | Coding | C# | Inheritance | Interfaces

Languages: C#

A customer requested a real estate application to manage their real estate listings. They want to have the ability to add, remove, update, and retrieve listings based on specific criteria.

Create a new class called *RealEstateListing*.

- Add the following properties to the *RealEstateListing* class:

© 2023 HackerRank | Privacy policy | Terms of service | API | Support | Status | Academy

Type Coding ×

C#: Movie Library EASY

Coding 30 mins

C#: Real Estate Listing Management EASY

Coding 30 mins

A customer requested a real estate application to manage their real estate listings. They want to have the ability to add, remove, update, and retrieve listings based on specific criteria.

Create a new class called *RealEstateListing*.

- Add the following properties to the *RealEstateListing* class:
  ID (int)
  Title (string)
  Description (string)
  Price (int)
  Location (string)

Create another class called *RealEstateApp*.

- Declare a private field within the *RealEstateApp* class called *listings*, which is a *List<IRealEstateListing>*.
- Implement the following methods within the *RealEstateApp* class:
  - *AddListing(IRealEstateListing)*: add the listing to *listings*

© 2023 HackerRank Privacy policy Terms of service API Support Status Academy

Coding 30 mins

C#: Real Estate Listing Management EASY

Coding 30 mins

C#: Library Management System MEDIUM

Coding 45 mins

Create another class called *RealEstateApp*.

- Declare a private field within the *RealEstateApp* class called *listings*, which is a *List<IRealEstateListing>*.
- Implement the following methods within the *RealEstateApp* class:
  - *AddListing(IRealEstateListing):* add the listing to *listings*
  - *RemoveListing(int listingID):* if a listing with this *ID* exists, remove it from *listings*
  - *UpdateListing(IRealEstateListing):* update the listing with a matching *ID* to the values in the parameter object
  - *GetListings():* return *listings*
  - *GetListingsByLocation(string location):* return a new list of the listings that match the *location* parameter
  - *GetListingsByPriceRange(int minPrice, int maxPrice):* return a new list of the listings whose price falls within the specified range, inclusive

Example

© 2023 HackerRank Privacy policy Terms of service API Support Status Academy

Coding 35 mins

C#: User Inheritance EASY

Coding 24 mins

C#: Cars Management EASY

Coding 40 mins

Sample Output

```
The most expensive car is Audi A5
The cheapest car is Audi A6
The average price of all cars is 59852
The most expensive model for brand Audi is A5 having price 79105
The most expensive model for brand BMW is 7_Series having price 78000
The most expensive model for brand Cadillac is CTS having price 49772
The most expensive model for brand Isuzu is Amigo having price 65135
The most expensive model for brand Mercury is Cougar having price 47560
The most expensive model for brand Nissan is Altima having price 72154
The most expensive model for brand Pontiac is Bonneville having price 58565
```

Explanation

Here, 12 cars are presented with their details (brand, model, and price). Then, based on that data, the 4 methods are called: *MostExpensiveCar*, *CheapestCar*, *AveragePriceOfCars*, and *MostExpensiveModelForEachBrand*. The result obtained from these 4 function calls is printed to the standard output.

© 2023 HackerRank Privacy policy Terms of service API Support Status Academy

3. Library Management System:

hackerrank.com/test/ep45skjo76l/questions/c3pg6eo4nlr

1h 57m left

ALL

1

2

1. Question 1

A software development team has a request to design a library management system. The system should have features to add and remove books from the library, calculate their total cost, group books by categories, and display the total price for each category.

Define the *Book* class:

- Add the class declaration for the *Book* class.
- Implement the *IBook* interface.
- Add the following properties to the *Book* class:
  - *Id* (integer): the unique identifier of the book
  - *Title* (string): the title of the book
  - *Author* (string): the author of the book
  - *Category* (string): the category or genre of the book
  - *Price* (integer): the price of the book

left

ALL

1

2

Define the *LibrarySystem* class:

- Add the class declaration for the *LibrarySystem* class.
- Implement the *ILibrarySystem* interface.
- Declare a private instance variable *_books* of type *Dictionary<IBook, int>*. This dictionary will store books as keys and their quantities as values.

Implement the methods defined in the *ILibrarySystem* interface within the *LibrarySystem* class.

1. *AddBook(IBook book, int quantity):* adds the specified book to the library system with the given quantity. If the book already exists in the system, the quantity is updated
2. *RemoveBook(IBook book, int quantity):* removes the specified quantity of the book from the library system. If the book's quantity becomes zero or less, it is completely removed from the system
3. *CalculateTotal():* calculates and returns the total value of all books in the library system based on their individual prices and quantities
4. *CategoryTotalPrice():* returns a list of tuples, where each tuple contains a category name and the total price of all books in that category
5. *BooksInfo():* returns a list of tuples, where each tuple contains the book title, its quantity, and its price
6. *CategoryAndAuthorWithCount():* returns a list of tuples, where each tuple contains the book's category, author, and the total quantity of books by that author in that category

Example:

ALL

1

2

Example:
There are 2 Book objects, with *Id, Title, Author, Category, Price,* and *Quantity*.
1 PeterPan JamesMatthewBarrie KidsClassics 193 11
2 TheWizardOfOz FrankBaum KidsClassics 394 3

The output is:
Book Info:
Book Name:PeterPan, Quantity:11, Price:193
Book Name:TheWizardOfOz, Quantity:3, Price:394
Category:KidsClassics, Total Price:3305
Category And Author With Count:
Category:KidsClassics, Author:JamesMatthewBarrie, Count:11
Category:KidsClassics, Author:FrankBaum, Count:3
Total Price: 3305

left

ALL

1

2

Input Format For Custom Testing

The first line contains an integer *n*, the number of books.
Each of the next *n* lines contains 6 space-separated values of book attribute values, *Id Title Author Category Price Quantity*.

Sample Case 1

Sample Input For Custom Testing

```
4
1 Title-1 Author-1 Category-4 206 14
2 Title-2 Author-2 Category-2 527 23
3 Title-3 Author-2 Category-2 734 6
4 Title-4 Author-1 Category-1 58 29
```

Sample Output

1h 28m left

ALL

1

2

```
Book Info:
Book Name:Title-1, Quantity:14,
Price:206
Book Name:Title-2, Quantity:23,
Price:527
Book Name:Title-3, Quantity:6, Price:734
Book Name:Title-4, Quantity:29, Price:58
Category Total Price:
Category:Category-1, Total Price:1682
Category:Category-2, Total Price:16525
Category:Category-4, Total Price:2884
Category And Author With Count:
Category:Category-1, Author:Author-1,
Count:29
Category:Category-2, Author:Author-2,
Count:29
Category:Category-4, Author:Author-1,
Count:14
Total Price: 21091
```

Explanation

There are 4 books to add to the list of books. The provided code stub produces the results of calling the methods.

1h 28m left

ALL

1

2

Sample Case 2

Sample Input For Custom Testing

```
3
1 Title-1 Author-1 Category-3 672 4
2 Title-2 Author-2 Category-4 59 29
3 Title-3 Author-1 Category-3 563 29
```

Sample Output

```
Book Info:
Book Name:Title-1, Quantity:4, Price:672
Book Name:Title-2, Quantity:29, Price:59
Book Name:Title-3, Quantity:29,
Price:563
Category Total Price:
Category:Category-3, Total Price:19015
Category:Category-4, Total Price:1711
Category And Author With Count:
Category:Category-3, Author:Author-1,
Count:33
Category:Category-4, Author:Author-2,
Count:29
Total Price: 20726
```

Explanation

There are 3 books to add to the list. The provided code stub produces the results of calling the methods.

hackerrank.com/test/ep45skjo76l/questions/c3pg6eo4nlr

1h 19m left

ALL

1

2

- Implement the *ILibrarySystem* interface.
- Declare a private instance variable *_books* of type *Dictionary<IBook, int>*. This dictionary will store books as keys and their quantities as values.

Implement the methods defined in the *ILibrarySystem* interface within the *LibrarySystem* class.

1. *AddBook(IBook book, int quantity):* adds the specified book to the library system with the given quantity. If the book already exists in the system, the quantity is updated
2. *RemoveBook(IBook book, int quantity):* removes the specified quantity of the book from the library system. If the book's quantity becomes zero or less, it is completely removed from the system
3. *CalculateTotal():* calculates and returns the total value of all books in the library system based on their individual prices and quantities
4. *CategoryTotalPrice():* returns a list of tuples, where each tuple contains a category name and the total price of all books in that category
5. *BooksInfo():* returns a list of tuples, where each tuple contains the book title, its quantity, and its price
6. *CategoryAndAuthorWithCount():* returns a list of tuples, where each tuple contains the book's category, author, and the total quantity of books by that author in that category

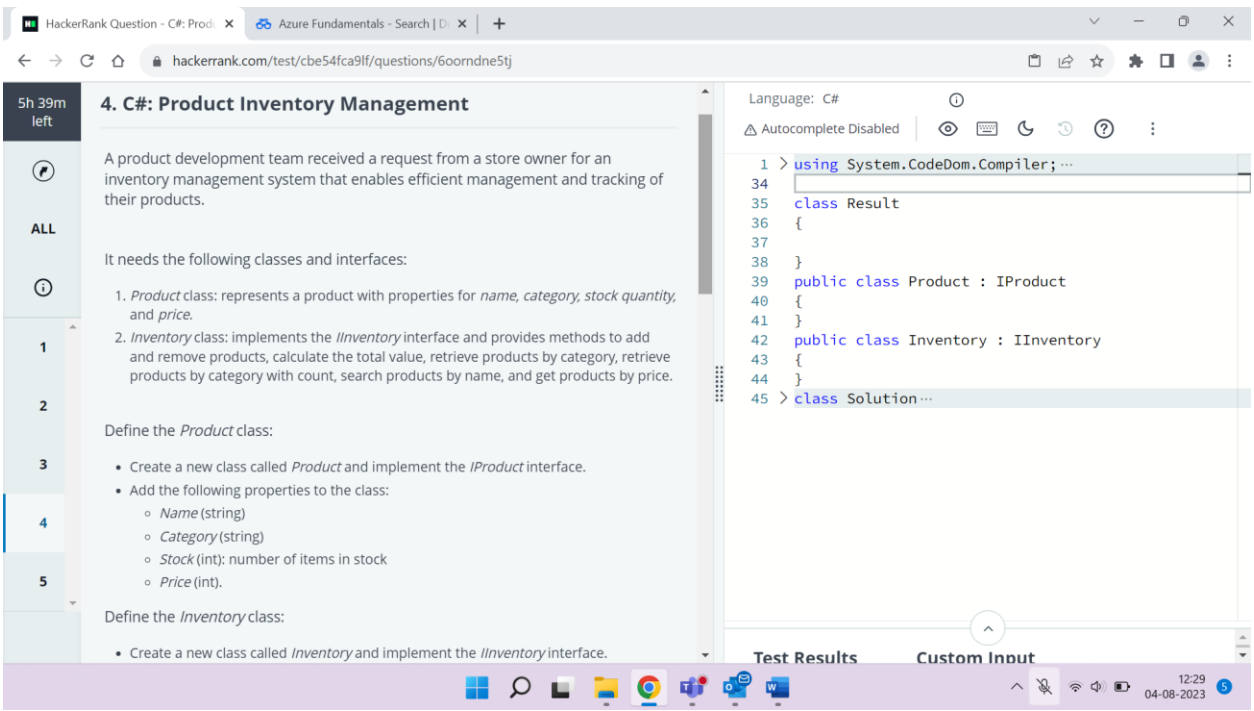
Code:

```
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;

public interface IBook
{
    int Id { get; set; }
    string Title { get; set; }
    string Author { get; set; }
    string Category { get; set; }
    int Price { get; set; }
}

public interface ILibrarySystem
{
    void AddBook(IBook book, int quantity);
    void RemoveBook(IBook book, int quantity);
    int CalculateTotal();
    List<(string, int)> CategoryTotalPrice();
    List<(string, int, int)> BooksInfo();
    List<(string, string, int)> CategoryAndAuthorWithCount();
```

```
}

public class Book : IBook
{
}
public class LibrarySystem : ILibrarySystem
{


}

class Solution
{
    public static void Main(string[] args)
    {
        TextWriter textWriter = new
StreamWriter(@System.Environment.GetEnvironmentVariable("OUTPUT_PATH"), true);

        ILibrarySystem librarySystem = new LibrarySystem();
        int bCount = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 1; i <= bCount; i++)
        {
            var a = Console.ReadLine().Trim().Split(" ");

            IBook e = new Book();
            e.Id = Convert.ToInt32(a[0]);
            e.Title = a[1];
            e.Author = a[2];
            e.Category = a[3];
            e.Price = Convert.ToInt32(a[4]);


            librarySystem.AddBook(e, Convert.ToInt32(a[5]));
        }

        textWriter.WriteLine("Book Info:");

        var booksInfo = librarySystem.BooksInfo();
        foreach (var (title, quantity, price) in booksInfo.OrderBy(a => a.Item1))
        {
            textWriter.WriteLine($"Book Name:{title}, Quantity:{quantity},
Price:{price}");
        }

        textWriter.WriteLine("Category Total Price:");
        var categoryTotalPrice = librarySystem.CategoryTotalPrice();
        foreach (var (category, totalPrice) in categoryTotalPrice.OrderBy(a =>
a.Item1))
        {
            textWriter.WriteLine($"Category:{category}, Total Price:{totalPrice}");
        }

        List<(string, string, int)> categoryAndAuthorWithCount =
librarySystem.CategoryAndAuthorWithCount();
        textWriter.WriteLine("Category And Author With Count:");
        foreach (var (category, author, count) in
categoryAndAuthorWithCount.OrderBy(a => a.Item1))
        {
```

```
            textWriter.WriteLine($"Category:{category}, Author:{author},
Count:{count}");
        }

        int total = librarySystem.CalculateTotal();
        textWriter.WriteLine($"Total Price: {total}");


        textWriter.Flush();
        textWriter.Close();
    }
}
```

4.Product Inventory Management



HackerRank Question - C#: Prod | Azure Fundamentals - Search | D

hackerrank.com/test/cbe54fca9lf/questions/6oorndne5tj

5h 25m left

ALL

1 2 3 4 5

Define the *Inventory* class:

- Create a new class called *Inventory* and implement the *IInventory* interface.
- Add a private field *_products* of type *List<IProduct>* to store the products.
- Implement the following methods:
  - *AddProduct(IProduct product):* adds the product to inventory
  - *RemoveProduct(IProduct product):* removes the product from the inventory.
  - *CalculateTotalValue():* calculates and returns the total value of all products in inventory
  - *GetProductsByCategory(string category):* returns a list of products that belong to the specified category
  - *GetProductsByCategoryWithCount():* returns a list of tuples, where each tuple contains a category and the count of products in that category
  - *SearchProductsByName(string name):* returns a list of products whose name contains the search string, *name*
  - *GetAllProductsByCategory():* returns a list of tuples, where each tuple contains a category and a list of products in that category

Example:
There are 3 *Product* objects with *Name, Category, Stock,* and *Price.*
Computer Technology 2 272
Refrigerator HomeAppliances 3 204
WashingMachine HomeAppliances 5 390

Determine the word to be grouped by category from the product list, the word to be searched according to the product name, and the product name to be removed

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

class Result
{

}
public class Product : IProduct
{
}
public class Inventory : IInventory
{
}
class Solution···
```

Test Results Custom Input

12:43 04-08-2023

HackerRank Question - C#: Prod | Azure Fundamentals - Search | D

hackerrank.com/test/cbe54fca9lf/questions/6oorndne5tj

5h 23m left

ALL

1 2 3 4 5

Determine the word to be grouped by category from the product list, the word to be searched according to the product name, and the product name to be removed from the list, respectively.
HomeAppliances Computer Refrigerator

Finally, calculate the total prices of the products. (*Stock*Price*)
Total Value is 2*272 + 3 * 204 + 5*390 equals 3106.
And the output will be:
Home appliances:
Product Name:Refrigerator Category:Home HomeAppliances
Product Name:WashingMachine Category:HomeAppliances
Computer:
Product Name:Computer Category:Technology
Total Value:$3106
Technology:1
HomeAppliances:2
Technology:
Product Name:Computer Price:272
HomeAppliances:
Product Name:Refrigerator Price:204
Product Name:WashingMachine Price:390
New Total Value:$2494

▸ **Input Format For Custom Testing**

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

class Result
{

}
public class Product : IProduct
{
}
public class Inventory : IInventory
{
}
class Solution···
```

Test Results Custom Input

12:44 04-08-2023

HackerRank Question - C#: Prod | Azure Fundamentals - Search | D

hackerrank.com/test/cbe54fca9lf/questions/6oorndne5tj

5h 23m left

ALL

1 2 3 4 5

▼ Input Format For Custom Testing

The first line contains an integer *n*, the number of products.
Each of the next *n* lines contains the product information separated by space (Name Category Stock Price).
The next line contains the string m, which is the category to be searched in the inventory, name and the product name to be deleted.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN                          Function
-----                          --------
4                              number of products n = 4
Product-1 Category-2 33 805    first product name, category,
stock, price = ('Product-1', 'Category-2', 33, 805)
Product-2 Category-1 88 444    second product...
Product-3 Category-1 51 559
Product-4 Category-3 81 315
Category-3 Product-1 Product-1   category to search = 'Category-
3', name to search for = 'Product-1', name to delete = 'Product-
1'
```

Sample Output

```
Category-3:
Product Name:Product-4 Category:Category-3
Product-1:
Product Name:Product-1 Category:Category-2
```

Language: C#

Autocomplete Disabled

```
1  > using System.CodeDom.Compiler; ···
34
35   class Result
36   {
37
38   }
39   public class Product : IProduct
40   {
41   }
42   public class Inventory : IInventory
43   {
44   }
45 > class Solution ···
```

Test Results Custom Input

12:45 04-08-2023

HackerRank Question - C#: Prod | Azure Fundamentals - Search | D

hackerrank.com/test/cbe54fca9lf/questions/6oorndne5tj

5h 22m left

ALL

1 2 3 4 5

Sample Output

```
Category-3:
Product Name:Product-4 Category:Category-3
Product-1:
Product Name:Product-1 Category:Category-2
Total Value:$119661
Category-1:2
Category-2:1
Category-3:1
Category-1:
Product Name:Product-2 Price:444
Product Name:Product-3 Price:559
Category-2:
Product Name:Product-1 Price:805
Category-3:
Product Name:Product-4 Price:315
New Total Value:$93096
```

Explanation

There are 4 products to add. The code stub reads the data, makes the method calls, and generates results.

▼ Sample Case 2

Sample Input For Custom Testing

```
4
Product-1 Category-3 0 590
Product-2 Category-3 18 307
Product-3 Category-2 47 841
```

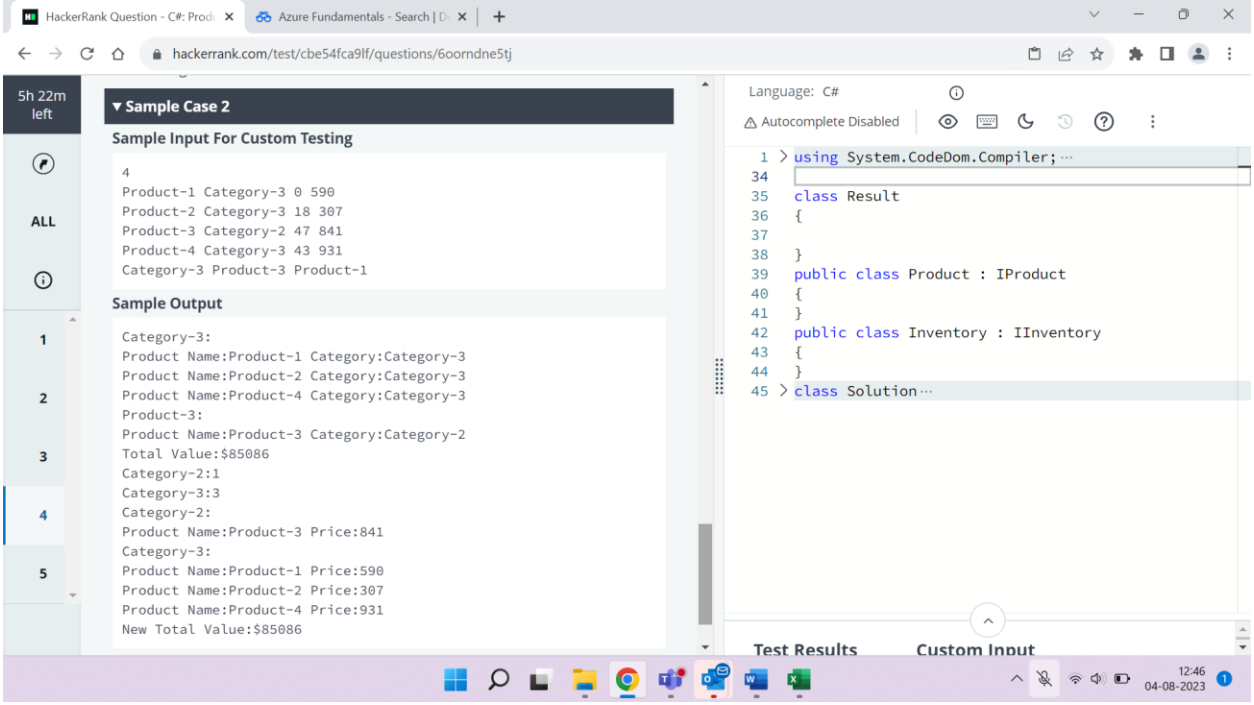
Language: C#

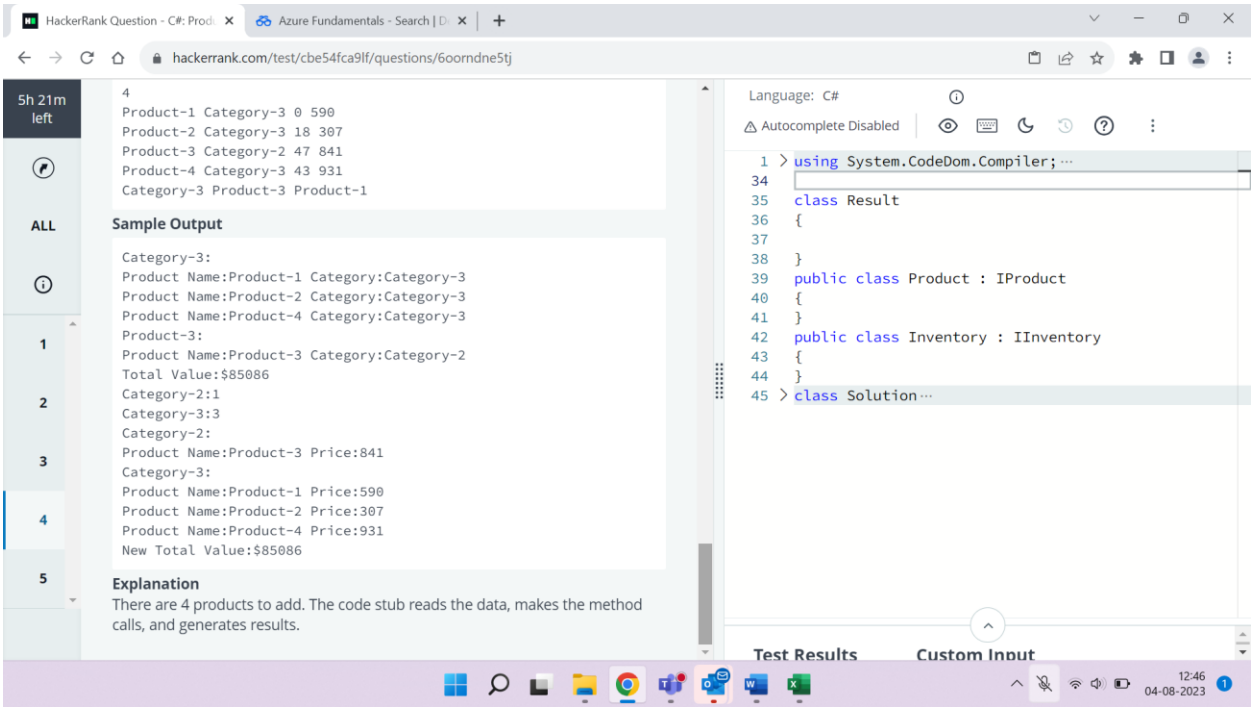
Autocomplete Disabled

```
1  > using System.CodeDom.Compiler; ···
34
35   class Result
36   {
37
38   }
39   public class Product : IProduct
40   {
41   }
42   public class Inventory : IInventory
43   {
44   }
45 > class Solution ···
```

Test Results Custom Input

12:46 04-08-2023





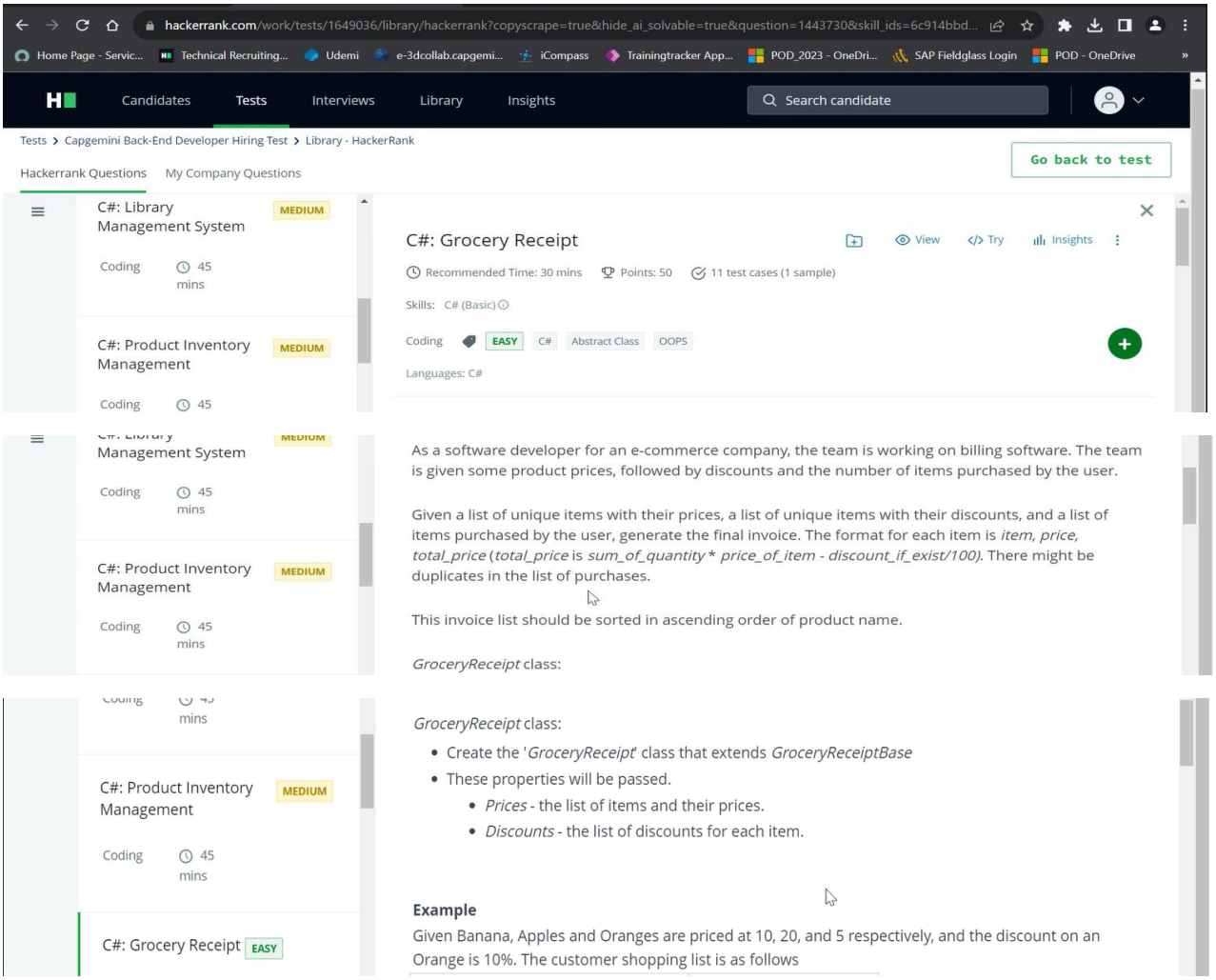

```
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;
```

```
public interface IProduct
{
    string Name { get; set; }
    string Category { get; set; }
    int Stock { get; set; }
    int Price { get; set; }
}

public interface IInventory
{
    void AddProduct(IProduct product);
    void RemoveProduct(IProduct product);
    int CalculateTotalValue();
    List<IProduct> GetProductsByCategory(string category);
    List<IProduct> SearchProductsByName(string name);
    List<(string, int)> GetProductsByCategoryWithCount();
    List<(string, List<IProduct>)> GetAllProductsByCategory();
}


class Result
{

}
public class Product : IProduct
{
}
public class Inventory : IInventory
{
}
class Solution
{
    public static void Main(string[] args)
    {
        TextWriter textWriter = new StreamWriter(@System.Environment.GetEnvironmentVa
riable("OUTPUT_PATH"), true);

        IInventory inventory = new Inventory();
        int pCount = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 1; i <= pCount; i++)
        {
            var a = Console.ReadLine().Trim().Split(" ");
            Product e = new Product();
            e.Name = a[0];
            e.Category = a[1];
            e.Stock = Convert.ToInt32(a[2]);
            e.Price = Convert.ToInt32(a[3]);
            inventory.AddProduct(e);
        }
        var b = Console.ReadLine().Trim().Split(" ");
        var randomCategoryName = b[0];
```

```
        var randomProductName = b[1];
        string productName = b[2];

        var getProductsByCategory = inventory.GetProductsByCategory(randomCategoryNam
e);

        textWriter.WriteLine($"{randomCategoryName}:");
        foreach (var product in getProductsByCategory.OrderBy(a=>a.Name))
        {
            textWriter.WriteLine($"Product Name:{product.Name} Category:{product.Cate
gory}");
        }

        var searchProductsByName = inventory.SearchProductsByName(randomProductName);
        textWriter.WriteLine($"{randomProductName}:");
        foreach (var product in searchProductsByName.OrderBy(a=>a.Name))
        {
            textWriter.WriteLine($"Product Name:{product.Name} Category:{product.Cate
gory}");
        }
        textWriter.WriteLine("Total Value:$" + inventory.CalculateTotalValue());

        var getProductsByCategoryWithCount = inventory.GetProductsByCategoryWithCount
();
        foreach (var item in getProductsByCategoryWithCount.OrderBy(a=>a.Item1))
        {
            textWriter.WriteLine($"{item.Item1}:{item.Item2}");
        }

        var getAllProductsByCategory = inventory.GetAllProductsByCategory();
        foreach (var item in getAllProductsByCategory.OrderBy(a=>a.Item1))
        {
            textWriter.WriteLine($"{item.Item1}:");
            foreach (var item2 in item.Item2)
            {
                textWriter.WriteLine($"Product Name:{item2.Name} Price:{item2.Price}"
);
            }
        }




        var productsToDelete = inventory.SearchProductsByName(productName);

        foreach (var product in productsToDelete)
        {
                inventory.RemoveProduct(product);
        }
        textWriter.WriteLine("New Total Value:$" + inventory.CalculateTotalValue());
```

```
        textWriter.Flush();
        textWriter.Close();
    }
}
```

5. Grocery Receipt

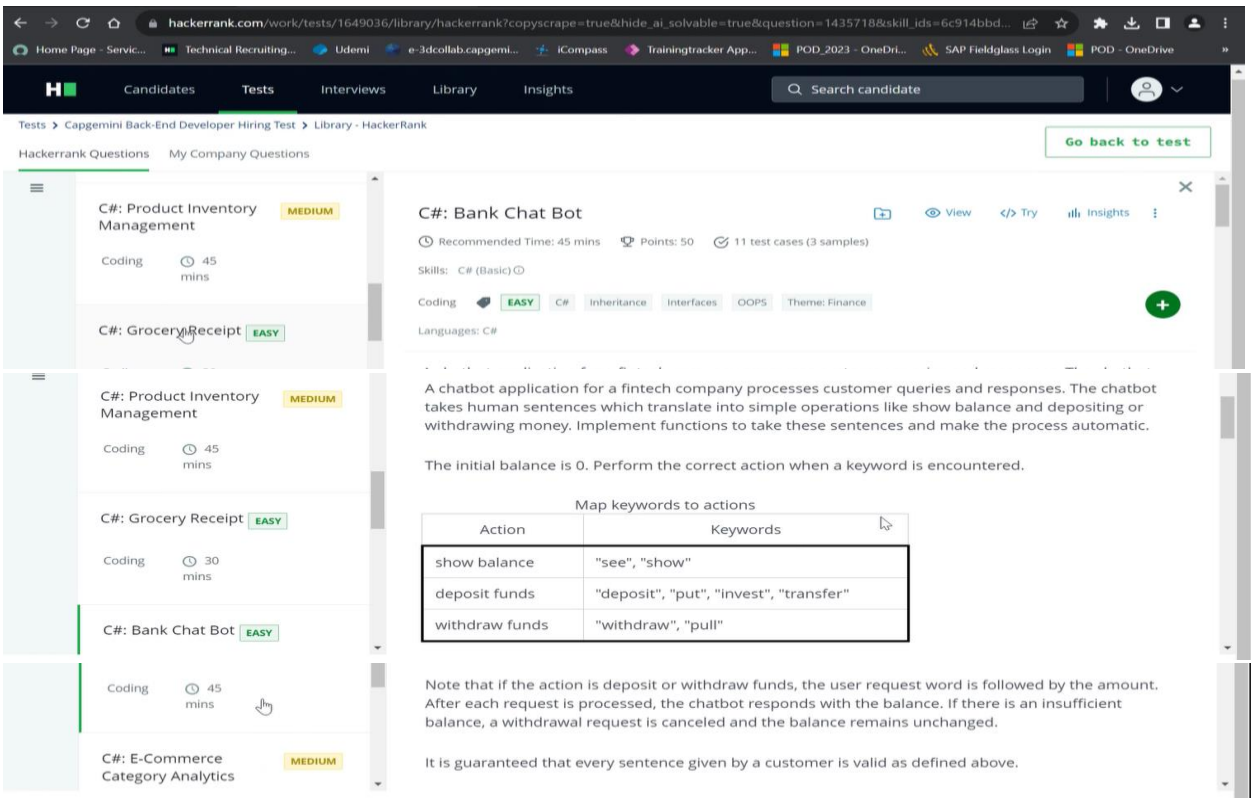



```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
```

```
abstract class GroceryReceiptBase2
{
    public Dictionary<string, decimal> Prices { get; set; }
    public Dictionary<string, int> Discounts { get; set; }


    public GroceryReceiptBase2(Dictionary<string, decimal> prices, Dictionary<string, int> discounts)
    {
        Prices = prices;
        Discounts = discounts;
    }


    public abstract IEnumerable<(string fruit, decimal price, decimal total)>
Calculate(List<Tuple<string, int>> shoppingList);
}


class GroceryReceipt2: GroceryReceiptBase2
{


}


class Solution2
{
    public static void Main(string[] args)
    {
        List<Tuple<string, int>> boughtItems = new List<Tuple<string, int>>();
        Dictionary<string, decimal> prices = new Dictionary<string, decimal>();
        Dictionary<string, int> discounts = new Dictionary<string, int>();


        int n = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 0; i < n; i++)
        {
            var a = Console.ReadLine().Trim().Split(' ');
            prices.Add(a[0], Convert.ToDecimal(a[1]));
        }


        int m = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 0; i < m; i++)
        {
            var a = Console.ReadLine().Trim().Split(' ');
            discounts.Add(a[0], Convert.ToInt32(a[1]));
        }


        int b = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 0; i < b; i++)
        {
            var a = Console.ReadLine().Trim().Split(' ');
            boughtItems.Add(new Tuple<string, int>(a[0], Convert.ToInt32(a[1])));
        }


        var g = new GroceryReceipt2(prices, discounts);
        var result = g.Calculate(boughtItems);


        foreach (var x in result)
        {
            Console.WriteLine($"{x.fruit} {x.price.ToString("0.0", new NumberFormatInfo() {
NumberDecimalSeparator = "." })} {x.total.ToString("0.0", new NumberFormatInfo() { NumberDecimalSeparator
= "." })}");
```

```
        }
    }
}
```

6. Bank Chat Bot



C#: Bank Chat Bot · Coding

QUESTION DESCRIPTION

A chatbot application for a fintech company processes customer queries and responses. The chatbot takes

human sentences which translate into simple operations like show balance and depositing or withdrawing money. Implement functions to take these sentences and make the process automatic.

The initial balance is 0. Perform the correct action when a keyword is encountered.

Map keywords to actions

Action	Keywords
show balance	"see", "show"
deposit funds	"deposit", "put", "invest", "transfer"
withdraw funds	"withdraw", "pull"

Note that if the action is deposit or withdraw funds, the user request word is followed by the amount. After each request is processed, the chatbot responds with the balance. If there is an insufficient balance, a withdrawal request is canceled and the balance remains unchanged.

It is guaranteed that every sentence given by a customer is valid as defined above.

BankOperations class:

- Implement the IBankAccountOperation interface.
- The following methods will be used to create output:
  - *Deposit* method takes as a parameter decimal *d* and adds it to the accounts.
  - *Withdraw* method takes as a parameter decimal *d* and subtracts it from the accounts.
  - *ProcessOperation* method takes as a parameter the string message and returns the balance.

Example

Commands	Output
I want to see my balance	0
I want to see money in my account	0
I want to withdraw 200	0
I want to pull 100	0
I want to deposit 500	500
I want to invest 600	1100
I want to transfer 100 to my account	1200
I want to deposit 500 dollars	1700
Pull 100 dollars	1600
Deposit 200	1800

The initial balance is 0, and no withdrawals are honored unless there is a sufficient balance.

Function Description

Create the BankOperations class and implement functions according to the IBankAccountOperation interface to process messages.

▼ Input Format For Custom Testing

The first line contains an integer n, the number of lines.

Each of the next *n* lines contains the sentence by a customer that needs to be processed.

▼ Sample Case 0

Sample Input

```
STDIN                       FUNCTION
-----                       --------
5                           number of requests = 5
Deposit 2396
I want to transfer 3017 dollars to my account
I want to pull 2300 dollars
I want to deposit 3980 dollars
I want to transfer 2811 dollars to my account
```

Sample Output

```
2396
5413
3113
```

```
7093
9904
```

Explanation

Initial balance is 0

Deposit 2396, balance 2396

Deposit 3017, balance 5413

Withdraw 2300, balance 3113

Deposit 3980, balance 7093

Deposit 2811, balance 9904

Sample Case 1

Sample Input

```
STDIN                                FUNCTION
-----                                --------
5
I want to invest 4410
I want
to
withdr
aw 161
dollar
sI
want
to
withdr
aw 443
I
want
to
pul
l
427
2
dol
lar
s
Pul
l
375
```

Sample Output

```
4410
4249
3806
3806
3431
```

Explanation

The balance is zero initially.

Deposit 4410, then make 4 withdrawals.

```
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;

public interface IBankAccountOperation
{
    /*
        Sample operations expected to be proceeded
        Values may be different
        -I want to see my balance
        -I want to see money in my account
        -I want to withdraw 200
        -I want to pull 100
        -I want to deposit 500
        -I want to invest 600
        -I want to transfer 100 to my account
        -I want to deposit 500 dollars
        -Pull 100 dollars
        -Deposit 200
    */
    void Deposit(decimal  d);
    void Withdraw(decimal d);
    //returns the current balance after process.
    decimal ProcessOperation(string message);
}

//Create BankOperations class by implementing IBankAccountOperation
 interface
class BankOperations:IBankAccountOperation
```

```
{

}

class Solution
{
    public static void Main(string[] args)
    {
        TextWriter textWriter = new StreamWriter(@System.Environmen
t.GetEnvironmentVariable("OUTPUT_PATH"), true);

        string k = Console.ReadLine();
        int n = Convert.ToInt32(k);
        List<string> inputs = new List<string>();
        for (int i = 0; i < n; i++)
        {
            inputs.Add(Console.ReadLine());
        }
        BankOperations opt = new BankOperations();
        foreach (var item in inputs)
        {
            textWriter.WriteLine(opt.ProcessOperation(item));
        }

        textWriter.Flush();
        textWriter.Close();
    }
}
```

7. E-Commerce Category Analytics:

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 17m left

ALL

1 2 3 4 5

1. C#: E-Commerce Category Analytics

An e-commerce company requests the following reports about sales and inventory.

- The category that has the most products.
- List products that belong to more than one category.
- The sum of product prices within each category.
- The category that has the highest sum of product prices.

Implement the classes shown.

- Create a class called *Category* that implements the *ICategory* interface.
  - This class should have the following properties:
    - *Id* (int)
    - *Name* (string)
    - *Products* (List of *IProduct*)
  - Create a constructor that
    - takes in an *Id* and *Name* as arguments
    - initializes the *Id* and *Name* properties
    - initializes the *Products* property as an empty list
  - Create a method called *AddProduct* that
    - takes in an *IProduct* object as an argument

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */


class Solution···
```

Test Results Custom Input

11:51 04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 16m left

ALL

1 2 3 4 5

  - Create a method called *AddProduct* that
    - takes in an *IProduct* object as an argument
    - adds the product to the *Products* list
- Create a class called *Product* that implements the *IProduct* interface.
  - This class should have the following properties:
    - *Id* (int)
    - *Name* (string)
    - *Price* (decimal)
  - Create a constructor that
    - takes in an *Id*, *Name*, and *Price* as arguments
    - initializes the *Id*, *Name*, and *Price* properties
- Create a class called *Company* that implements the *ICompany* interface.
  - This class should have the following properties:
    - *Id* (int)
    - *Name* (string)
    - *Categories* (List of *ICategory*)
  - Create a constructor that
    - takes in an *Id* and *Name* as arguments
    - initializes the *Id* and *Name* properties
    - initializes the *Categories* property as an empty list
  - Create a method called *AddCategory* that
    - takes in an *ICategory* object as an argument
    - adds the category to the Categories list
  - Create a method called *GetTopCategoryNameByProductCount* that
    - returns the name of the category with the most products

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */


class Solution···
```

Test Results Custom Input

11:52 04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 13m left

ALL

1 2 3 4 5

- Create a method called *GetTopCategoryNameByProductCount* that
  - returns the name of the category with the most products
- Create a method called *GetProductsBelongsToMultipleCategory* that
  - returns a list of products that belong to multiple categories
  - retains the order that products were added
- Create a method called *GetTopCategoryBySumOfProductPrices* that
  - returns the name and total value of the category with the highest sum of product prices
- Create a method called *GetCategoriesWithSumOfTheProductPrices* that
  - returns a list of tuples that contains the category and the sum of the product prices for each category
  - retains the order that products were added

Example

There are two categories *[Category1, Category2]* and 6 products with *Name*, *Price*, and *Category* as shown.

Name	Price	Category
Product1	10	Category1
Product2	15	Category1, Category2
Product3	20	Category1, Category2
Product4	10	Category1, Category2

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */


class Solution···
```

Test Results Custom Input

11:54 04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 12m left

ALL

1 2 3 4 5

Example

There are two categories *[Category1, Category2]* and 6 products with *Name*, *Price*, and *Category* as shown.

Name	Price	Category
Product1	10	Category1
Product2	15	Category1, Category2
Product3	20	Category1, Category2
Product4	10	Category1, Category2
Product5	5	Category2
Product6	*30*	Category2

Output
Top category:Category2
Common products:
Product2
Product3
Product4
Most valuable category:Category2 80.0
Category1 55.0

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */


class Solution···
```

Test Results Custom Input

11:56 04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 11m left

Category1 has 4 products, Category2 has 5 products. Product2, Product3, Product4 are common products. The sum of the prices for Category1 is 55, Category2 is 80.

▾ Input Format For Custom Testing

The first line contains an integer *n*, the number of product lines in the input.
Each of the next *n* lines contains the product information (Id, Name, Price).
The next line contains an integer *m*, the number of category lines in the input.
Each of the next *m* lines contains the category information (Id, Name).
The next line contains an integer *k*, the number of products added to a category.
Each of the next *k* lines contains the Category Id and Product Id comma separated.

▾ Sample Case 0

Sample Input

```
STDIN                    FUNCTION
-----                    --------
6                  →     n = 6
1 Product1 96      →     [ProductId, ProductName, ProductPrice]
= [[1, "Product1", 96],
2 Product2 15
[2, "Product2", 15],
3 Product3 145
[3, "Product3", 145],
4 Product4 164
[4, "Product4", 164],
```

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */


class Solution···
```

Test Results Custom Input

11:57
04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 10m left

▾ Sample Case 0

Sample Input

```
STDIN                    FUNCTION
-----                    --------
6                  →     n = 6
1 Product1 96      →     [ProductId, ProductName, ProductPrice]
= [[1, "Product1", 96],
2 Product2 15
[2, "Product2", 15],
3 Product3 145
[3, "Product3", 145],
4 Product4 164
[4, "Product4", 164],
5 Product5 26
[5, "Product5", 26],
6 Product6 195
[6, "Product6", 195]]
3                  →     m = 3
1 Category1        →     [CategoryId, CategoryName] = [[1,
"Category1"], [2, "Category2"],
2 Category2                                            [3,
"Category3"]]
3 Category3
9                  →     k = 9
1 1                →     [CategoryId, ProductId] = [[1,1], [1,
2], [1, 4], [1, 5], [1, 6], [2, 2],
1 2                                                    [3, 3], [3,
4], [3, 5]]
```

Language: C#

Autocomplete Disabled

```
using System.CodeDom.Compiler;···

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */


class Solution···
```

Test Results Custom Input

11:57
04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 10m left

ALL

1 2 3 4 5

Sample Output

```
Top category:Category1
Common products:
Product2
Product4
Product5
Most valuable category:Category1 496.0
Category1 496.0
Category2 15.0
Category3 335.0
```

Explanation

Categories and their products are

Category 1 [1, 2, 4, 5, 6]

Category 2 [2]

Category 3 [3, 4, 5].

Products in multiple categories, in the order they first appeared are

2 [1, 2]

4 [1, 3]

5 [1, 3]

The values of each category are

Category 1 96 + 15 + 164 + 26 + 195 = 496

Category 2 15

Category 3 164 + 26 + 195 =385

▸ **Sample Case 1**

Language: C#

Autocomplete Disabled

```
1  > using System.CodeDom.Compiler; ···
41
42   /*
43    * Create Category,Product and Company classes by implementing
44    * ICategory,
45    * IProcduct,
46    * ICompany interfaces.
47    */
48
49
50 > class Solution ···
```

Test Results Custom Input

11:58 04-08-2023

HackerRank Question - C#: E-Co

hackerrank.com/test/cbe54fca9lf/questions/eqjqnq5c2c3

6h 8m left

ALL

1 2 3 4 5

▾ **Sample Case 1**

Sample Input

```
STDIN                    FUNCTION
-----                    --------
7                  →     n = 7
1 Product1 35      →     [ProductId, ProductName, ProductPrice] =
[[1, "Product1", 35],
2 Product2 175
[2, "Product2", 175],
3 Product3 117
[3, "Product3", 117],
4 Product4 30
[4, "Product4", 30],
5 Product5 198
[5, "Product5", 198],
6 Product6 171
[6, "Product1", 171],
7 Product7 123
[7, "Product7", 123]]
3                  →     m = 3
1 Category1        →     [CategoryId, CategoryName] = [[1,
"Category1"], [2, "Category2"],
2 Category2                                            [3,
"Category3"]]
3 Category3
13                 →     k = 13
1 2                →     [CategoryId, ProductId] = [[1,2], [1,
4], [1, 5], [1, 6], [2, 1], [2, 2]
```

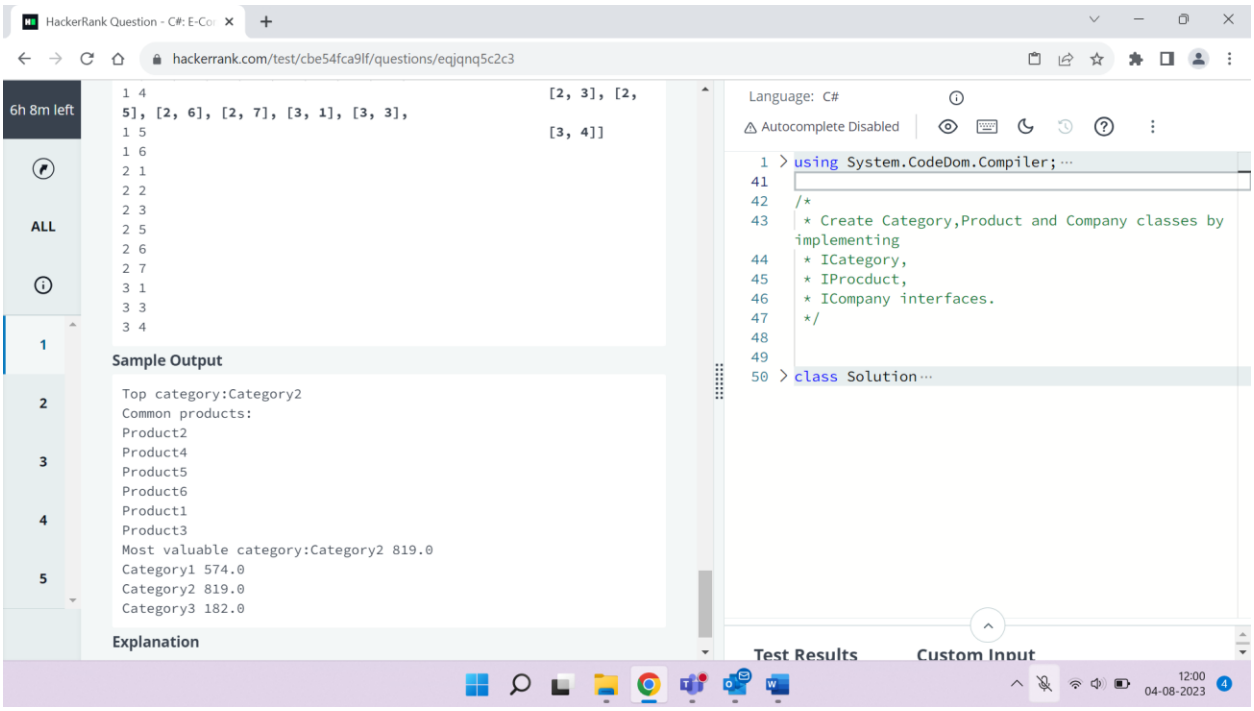
Language: C#

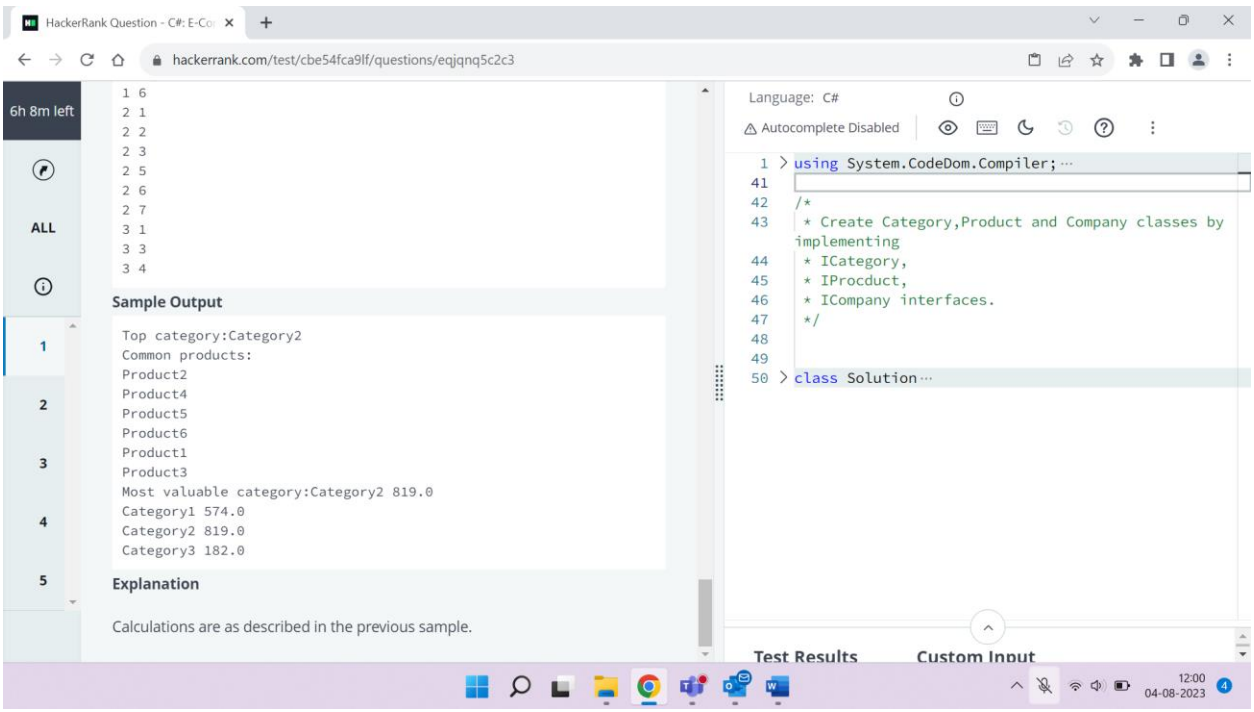
Autocomplete Disabled

```
1  > using System.CodeDom.Compiler; ···
41
42   /*
43    * Create Category,Product and Company classes by implementing
44    * ICategory,
45    * IProcduct,
46    * ICompany interfaces.
47    */
48
49
50 > class Solution ···
```

Test Results Custom Input

12:00 04-08-2023






```
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Globalization;
```

```
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;

interface ICategory
{
    int Id { get; set; }
    string Name { get; set; }
    List<IProduct> Products { get; set; }
    void AddProduct(IProduct product);
}
interface IProduct
{
    int Id { get; set; }
    string Name { get; set; }
    decimal Price { get; set; }
}

interface ICompany
{
    //The top category name by product count
    string GetTopCategoryNameByProductCount();
    //The products are added to more than one category
    List<IProduct> GetProductsBelongsToMultipleCategory();
    //Top category according to the total price of its products
    (string categoryName, decimal totalValue) GetTopCategoryBySumOf
ProductPrices();
    //The list of categories with the sum of the prices of the prod
ucts
    List<(ICategory category, decimal totalValue)> GetCategoriesWit
hSumOfTheProductPrices();
}

/*
 * Create Category,Product and Company classes by implementing
 * ICategory,
 * IProcduct,
 * ICompany interfaces.
 */
```

```
class Solution
{
    public static void Main(string[] args)
    {
        TextWriter textWriter = new StreamWriter(@System.Environmen
t.GetEnvironmentVariable("OUTPUT_PATH"), true);
        var company = new Company(1, "Company 1");
        List<IProduct> products = new List<IProduct>();
        List<ICategory> categories = new List<ICategory>();

        int n = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 1; i <= n; i++)
        {
            var a = Console.ReadLine().Trim().Split(" ");
            products.Add(new Product(Convert.ToInt32(a[0]), a[1], C
onvert.ToInt32(a[2])));
        }

        int m = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 1; i <= m; i++)
        {
            var a = Console.ReadLine().Trim().Split(" ");
            categories.Add(new Category(Convert.ToInt32(a[0]), a[1]
));
        }

        int p = Convert.ToInt32(Console.ReadLine().Trim());
        for (int i = 1; i <= p; i++)
        {
            var a = Console.ReadLine().Trim().Split(" ");
            var c = categories.FirstOrDefault(x=>x.Id == Convert.To
Int32(a[0]));
            var pp = products.FirstOrDefault(x=>x.Id == Convert.ToI
nt32(a[1]));
            if(c!= null && pp!= null){
                c.AddProduct(pp);
            }

        }

        foreach (var item in categories)
```

```
        {
            company.AddCategory(item);
        }
        var topCategory = company.GetTopCategoryNameByProductCount(
);
        var commonProducts = company.GetProductsBelongsToMultipleCa
tegory();
        var mostValuableCategory = company.GetTopCategoryBySumOfPro
ductPrices();
        var categoriesWithTotalPrices = company.GetCategoriesWithSu
mOfTheProductPrices();

        textWriter.WriteLine("Top category:" + topCategory);
        textWriter.WriteLine("Common products:");
        foreach (var item in commonProducts)
        {
            textWriter.WriteLine(item.Name);
        }
        textWriter.WriteLine("Most valuable category:" + mostValuab
leCategory.categoryName
            + " " + mostValuableCategory.totalValue.ToString("0.0",
 new NumberFormatInfo() { NumberDecimalSeparator = "." }));
        foreach (var item in categoriesWithTotalPrices)
        {
            textWriter.WriteLine(item.category.Name + " " + item.to
talValue.ToString("0.0", new NumberFormatInfo() { NumberDecimalSepa
rator = "." }));
        }


        textWriter.Flush();
        textWriter.Close();
    }
}
```

9. Employee Inheritance:

HackerRank Question - C#: Empl

hackerrank.com/test/cbe54fca9lf/questions/e4obiocdh7c

6h 6m left

ALL

1 2 3 4 5

2. C#: Employee Inheritance

Implement inheritance as described below.

Create an abstract class Employee that has the following:

1. A member variable *department* [string]
2. A member variable *name* [string]
3. A member variable *location* [string]
4. A member variable *isOnVacation* [boolean] (default status value is false)
5. A constructor function that takes 3 parameters and assigns them to *department*, *name*, and *location* respectively
6. A member function *GetDepartment()* that returns the value *department*
7. A member function *GetName()* that returns the value *name*
8. A member function *GetLocation()* that returns the value *location*
9. A member function *GetStatus()* that returns the value *isOnVacation*
10. A member function *SwitchStatus()* that toggles the *isOnVacation* value

Create a class FinanceEmployee that inherits from the above class Employee. It has the following:

Language: C#

Autocomplete Disabled

```
using System; ···
    class Employee {}
    class FinanceEmployee {}
    class MarketingEmployee {}
    class Solution ···
```

Test Results Custom Input

12:02 04-08-2023

HackerRank Question - C#: Empl

hackerrank.com/test/cbe54fca9lf/questions/e4obiocdh7c

6h 6m left

ALL

1 2 3 4 5

Create a class FinanceEmployee that inherits from the above class Employee. It has the following:

1. A constructor function that takes 2 parameters, *name* and *location*. It calls the base class constructor with *department* value 'Finance', *name*, and *location* respectively.

Create a class MarketingEmployee that inherits from the above class Employee. It has the following:

1. A constructor function that takes 2 parameters, *name* and *location*. It calls the base class constructor with *department* value 'Marketing', *name*, and *location* respectively.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the function calls. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the stubbed code.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
Sam Seattle
Jane Miami
```

Sample Output

Language: C#

Autocomplete Disabled

```
using System; ···
    class Employee {}
    class FinanceEmployee {}
    class MarketingEmployee {}
    class Solution ···
```

Test Results Custom Input

12:02 04-08-2023

HackerRank Question - C#: Empl

hackerrank.com/test/cbe54fca9lf/questions/e4obiocdh7c

6h 3m left

ALL

1 2 3 4 5

Input Format For Custom Testing

The first line contains 2 space-separated strings for building the FinanceEmployee object, where first is the name of the employee and second is the location of the employee respectively.
The second line contains 2 space-separated strings for building the MarketingEmployee object, where first is the name of the employee and second is the location of the employee respectively.

Sample Case 0

Sample Input For Custom Testing

```
Sam Seattle
Jane Miami
```

Sample Output

```
FinanceEmployee info: Department - Finance, Name - Sam,
Location - Seattle
Sam is not on vacation
Switching
Sam is on vacation
Switching
Sam is not on vacation
MarketingEmployee info: Department - Marketing, Name - Jane,
Location - Miami
Jane is not on vacation
Switching
Jane is on vacation
```

Language: C#

Autocomplete Disabled

```
using System;···
    class Employee {}
    class FinanceEmployee {}
    class MarketingEmployee {}
    class Solution···
```

Test Results Custom Input

12:05 04-08-2023

HackerRank Question - C#: Empl

hackerrank.com/test/cbe54fca9lf/questions/e4obiocdh7c

6h 2m left

ALL

1 2 3 4 5

Sample Case 0

Sample Input For Custom Testing

```
Sam Seattle
Jane Miami
```

Sample Output

```
FinanceEmployee info: Department - Finance, Name - Sam, Location
- Seattle
Sam is not on vacation
Switching
Sam is on vacation
Switching
Sam is not on vacation
MarketingEmployee info: Department - Marketing, Name - Jane,
Location - Miami
Jane is not on vacation
Switching
Jane is on vacation
Switching
Jane is not on vacation
```

Explanation

First, an employee object is created with *department* "Finance", *name* "Sam", *location* "Seattle", and *isOnVacation* false. Then, all 4 functions are called: *GetDepartment*, *GetName*, *GetLocation*, and *GetStatus*. The result is printed to the standard output.

Then, the function *SwitchStatus* is called, followed by *GetStatus*, and the result is

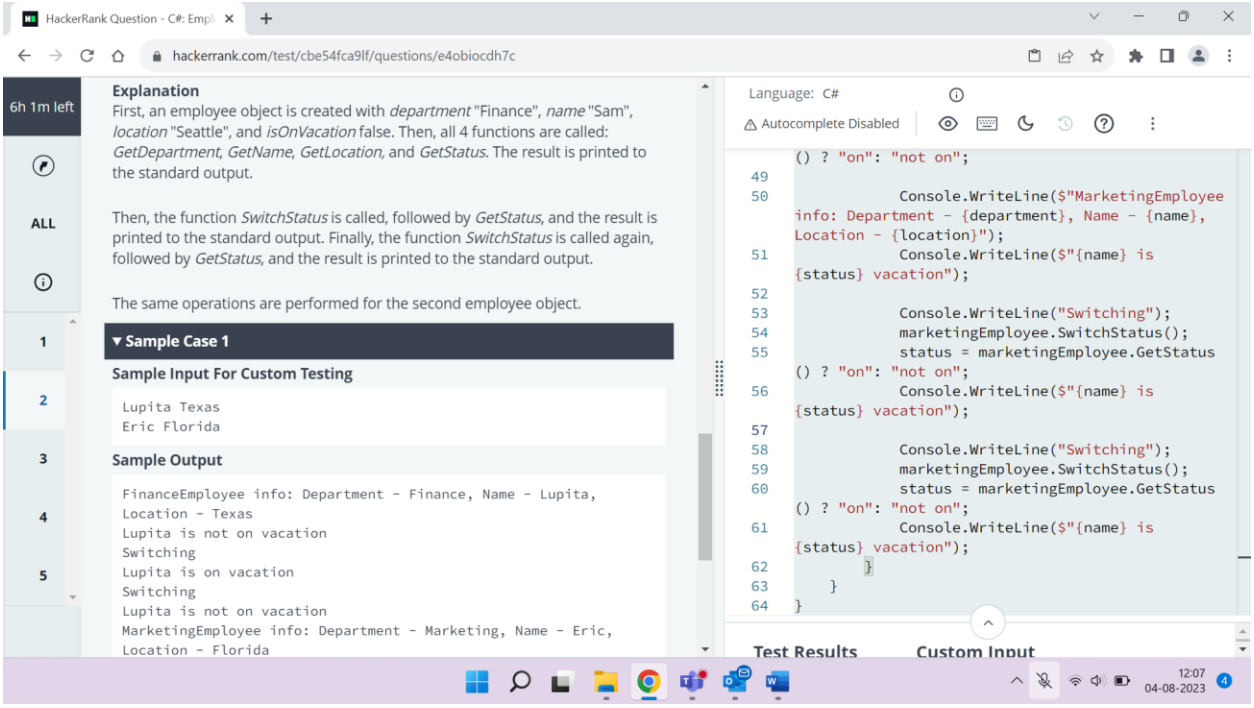
Language: C#

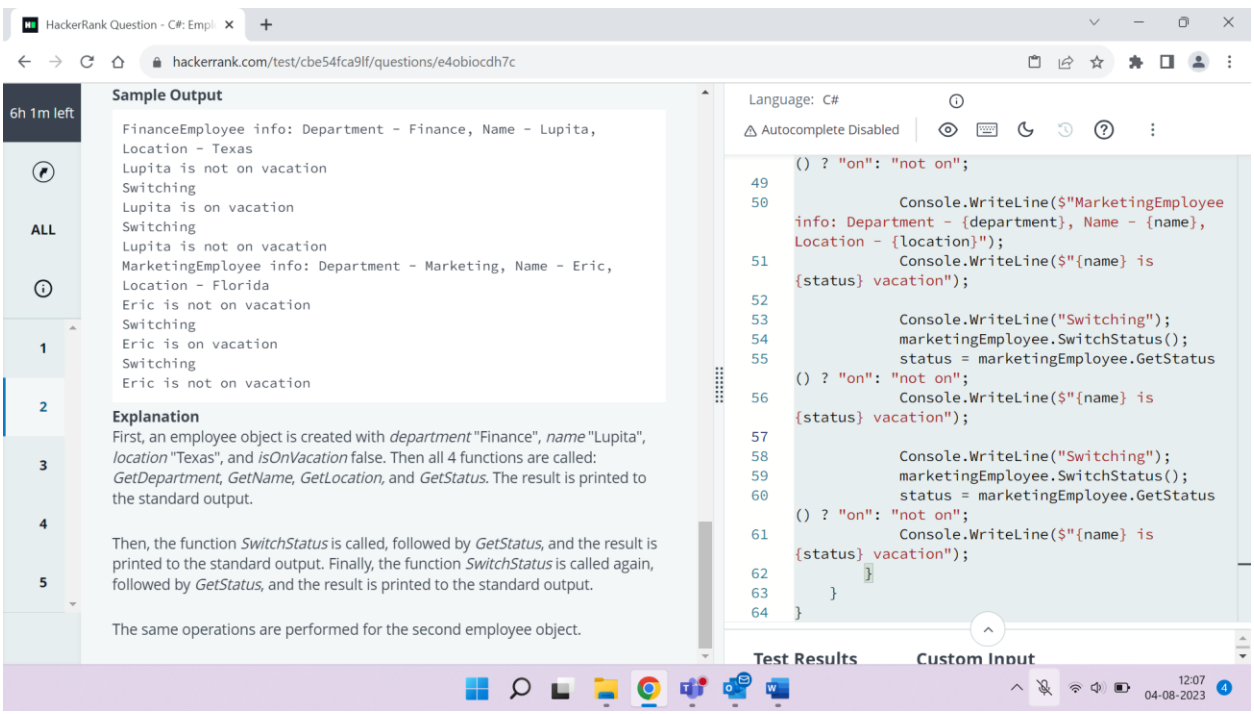
Autocomplete Disabled

```
() ? "on": "not on";

            Console.WriteLine($"MarketingEmployee
info: Department - {department}, Name - {name},
Location - {location}");
            Console.WriteLine($"{name} is
{status} vacation");

            Console.WriteLine("Switching");
            marketingEmployee.SwitchStatus();
            status = marketingEmployee.GetStatus
() ? "on": "not on";
            Console.WriteLine($"{name} is
{status} vacation");

            Console.WriteLine("Switching");
            marketingEmployee.SwitchStatus();
            status = marketingEmployee.GetStatus
() ? "on": "not on";
            Console.WriteLine($"{name} is
{status} vacation");
        }
    }
}
```

Test Results Custom Input

12:06 04-08-2023





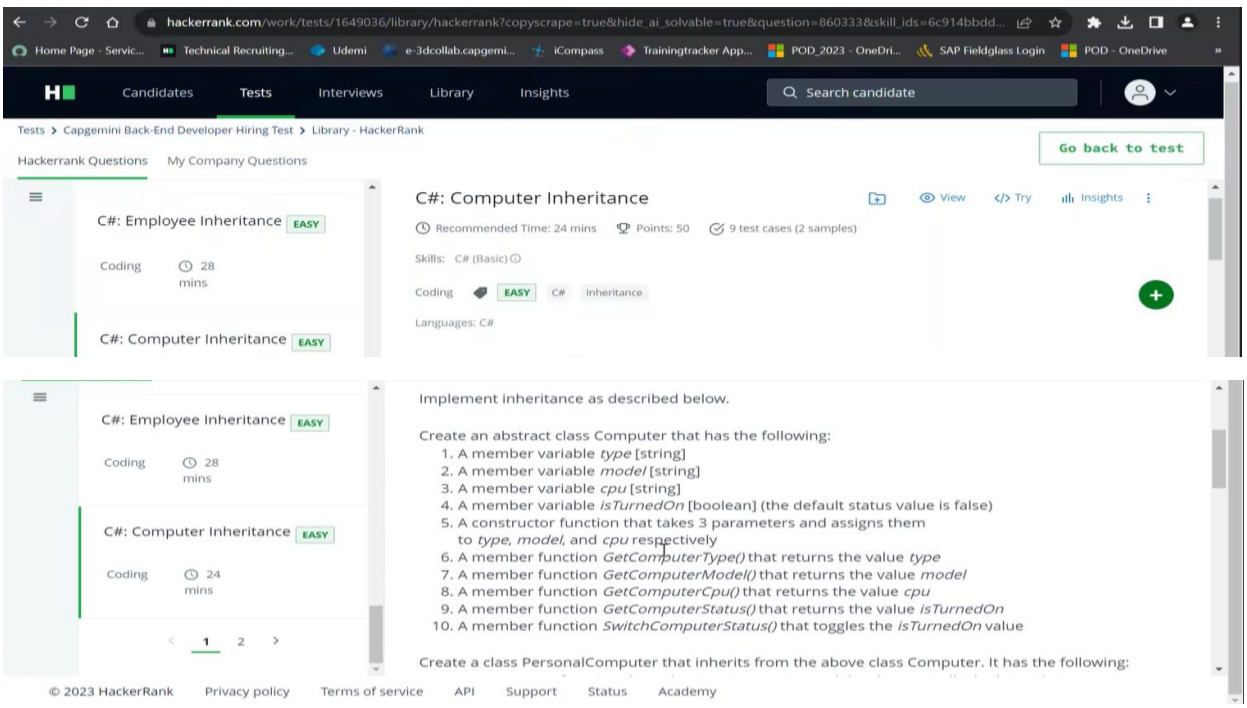

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Solution
```

```
{

    public abstract class Employee
    {
        protected string department;
        protected string name;
        protected string location ;
        protected bool isOnVacation=false;
        public Employee(string department,string name,string locati
on)
        {
            this.department=department;
            this.name=name;
            this.location=location;
        }
        public abstract string GetDepartment();

        public abstract string GetName();

        public abstract string GetLocation();

        public abstract bool GetStatus();

        public abstract void SwitchStatus();

    }
    class FinanceEmployee:Employee
    {
    }
    class MarketingEmployee:Employee
    {
    }

    class Solution
    {
        static void Main()
        {
            Type baseType = typeof(Employee);
            if (!baseType.IsAbstract)
                throw new Exception($"{baseType.Name} type should b
e abstract");

            string str = Console.ReadLine();
```

```
            string[] strArr = str.Split(' ');
            Employee financeEmployee = new FinanceEmployee(strArr[0
], strArr[1]);

            var department = financeEmployee.GetDepartment();
            var name = financeEmployee.GetName();
            var location = financeEmployee.GetLocation();
            var status = financeEmployee.GetStatus() ? "on": "not o
n";

            Console.WriteLine($"FinanceEmployee info: Department -
{department}, Name - {name}, Location - {location}");
            Console.WriteLine($"{name} is {status} vacation");

            Console.WriteLine("Switching");
            financeEmployee.SwitchStatus();
            status = financeEmployee.GetStatus() ? "on": "not on";
            Console.WriteLine($"{name} is {status} vacation");

            Console.WriteLine("Switching");
            financeEmployee.SwitchStatus();
            status = financeEmployee.GetStatus() ? "on": "not on";
            Console.WriteLine($"{name} is {status} vacation");

            str = Console.ReadLine();
            strArr = str.Split(' ');
            Employee marketingEmployee = new MarketingEmployee(strA
rr[0], strArr[1]);

            department = marketingEmployee.GetDepartment();
            name = marketingEmployee.GetName();
            location = marketingEmployee.GetLocation();
            status = marketingEmployee.GetStatus() ? "on": "not on"
;

            Console.WriteLine($"MarketingEmployee info: Department
- {department}, Name - {name}, Location - {location}");
            Console.WriteLine($"{name} is {status} vacation");

            Console.WriteLine("Switching");
            marketingEmployee.SwitchStatus();
            status = marketingEmployee.GetStatus() ? "on": "not on"
;
```

```
            Console.WriteLine($"{name} is {status} vacation");

            Console.WriteLine("Switching");
            marketingEmployee.SwitchStatus();
            status = marketingEmployee.GetStatus() ? "on": "not on"
;
            Console.WriteLine($"{name} is {status} vacation");
        }
    }
}
```

10. Computer Inheritance:



11. Authentication State

HackerRank Question - C#: Auth

hackerrank.com/test/cbe54fca9lf/questions/4rci26qmclc

6h 0m left

ALL

1 2 3 4 5

3. C#: Authentication State

Implement 3 functions to allow users to register, log in, and log out of a website. See the table for messages to return.

- Users will register with their email, password, and location information.
  - Users can register from any location.
- There is a list of allowed locations.
- The user logs in with an email address and password.
  - Users can log in only from an allowed location.
- If the username and password are correct, the login attempt fails when any of the following exists:
  - the user is logged in already
  - the user is logged in already from different a location
  - the user attempts to log in from a location that is not allowed
  - the user is blocked
- The user is blocked after 3 consecutive failed login attempts.
  - The count is reset to 0 upon successful login.

The messages that should be returned according to status are shown.

Register	Login	Logout
User1@email.com	User1@email.com is not	User1@email.com is

Language: C#

Autocomplete Disabled

```
35  * These strings can be copied and pasted to avoid typing errors.
36  * User1@email.com should be replaced with the correct user email.
37  *
38  * User1@email.com registered successfully!
39  * User1@email.com is not registered!
40  * User1@email.com is not logged in!
41  * User1@email.com is already registered!
42  * User1@email.com logged in successfully!
43  * User1@email.com logged out successfully!
44  * User1@email.com is already logged in!
45  * User1@email.com is already logged in from another location!
46  * User1@email.com is blocked!
47  * User1@email.com is not allowed to login from this location!
48  * User1@email.com password is incorrect!
49  */
50
51  class User{
52
53  }
54  class ApplicationAuthState{
```

Test Results Custom Input

12:08 04-08-2023

HackerRank Question - C#: Auth

hackerrank.com/test/cbe54fca9lf/questions/4rci26qmclc

5h 57m left

ALL

1 2 3 4 5

The messages that should be returned according to status are shown.

Register	Login	Logout
User1@email.com registered successfully!	User1@email.com is not registered!	User1@email.com is not logged in!
User1@email.com is already registered!	User1@email.com logged in successfully!	User1@email.com logged out successfully!
	User1@email.com is already logged in!	
	User1@email.com is already logged in from another location!	
	User1@email.com is blocked!	
	User1@email.com is not allowed to login from this location!	
	User1@email.com password is incorrect!	

Language: C#

Autocomplete Disabled

```
35  * These strings can be copied and pasted to avoid typing errors.
36  * User1@email.com should be replaced with the correct user email.
37  *
38  * User1@email.com registered successfully!
39  * User1@email.com is not registered!
40  * User1@email.com is not logged in!
41  * User1@email.com is already registered!
42  * User1@email.com logged in successfully!
43  * User1@email.com logged out successfully!
44  * User1@email.com is already logged in!
45  * User1@email.com is already logged in from another location!
46  * User1@email.com is blocked!
47  * User1@email.com is not allowed to login from this location!
48  * User1@email.com password is incorrect!
49  */
50
51  class User{
52
53  }
54  class ApplicationAuthState{
```

Test Results Custom Input

12:11 04-08-2023

HackerRank Question - C#: Auth

hackerrank.com/test/cbe54fca9lf/questions/4rci26qmclc

5h 57m left

ALL

1 2 3 4 5

Example

Given the following allowed locations = ["*Location1*", "*Location2*"] and commands:

Commands	Output
Register User1@email.com with location Location1	*User1@email.com registered successfully!*
Login User1@email.com from Location1	*User1@email.com logged in successfully!*
Login User1@email.com from Location2	*User1@email.com is already logged in from another location!*
Login User2@email.com without registration.	*User2@email.com is not registered!*

Function Description

- Create the *User* class and implement the *IUser* interface.
  - This class should have:
    - a constructor which takes id, email, password, and location.
    - *Id, Email, Password, Location*, and *IncorrectAttempt* properties.
      - These properties are initialized with the passed values by the

Language: C#

Autocomplete Disabled

```
35  * These strings can be copied and pasted to avoid typing errors.
36  * User1@email.com should be replaced with the correct user email.
37  *
38  * User1@email.com registered successfully!
39  * User1@email.com is not registered!
40  * User1@email.com is not logged in!
41  * User1@email.com is already registered!
42  * User1@email.com logged in successfully!
43  * User1@email.com logged out successfully!
44  * User1@email.com is already logged in!
45  * User1@email.com is already logged in from another location!
46  * User1@email.com is blocked!
47  * User1@email.com is not allowed to login from this location!
48  * User1@email.com password is incorrect!
49  */
50
51  class User{
52
53  }
54  class ApplicationAuthState{
```

Test Results Custom Input

12:11 04-08-2023

HackerRank Question - C#: Auth

hackerrank.com/test/cbe54fca9lf/questions/4rci26qmclc

5h 56m left

ALL

1 2 3 4 5

- These properties are initialized with the passed values by the constructor.
- *IncorrectAttempt* is initialized to 0.
- Create the *ApplicationAuthState* class and implement the IApplicationAuthState interface.
  - This class should have:
    - a constructor which
      - takes a list of allowed locations
      - initializes the *AllowedLocations* property with the provided list
      - initializes the *RegisteredUsers* and *UsersLoggedIn* properties to new instances of List<IUser>
    - the public *Register* method -
      - The method takes a parameter user of type *IUser* and returns a string.
      - It verifies
        - that the user is not yet registered
      - If the user is not yet registered
        - add the user to *RegisteredUsers*
        - return the success message
      - If the user is already registered
        - return the appropriate message
    - the public *Login* method -
      - The method takes a parameter user of type *IUser* and returns a string.
      - It verifies
        - that the provided email is registered
        - that the user does not have too many failed login attempts
        - that the password is correct

Language: C#

Autocomplete Disabled

```
35  * These strings can be copied and pasted to avoid typing errors.
36  * User1@email.com should be replaced with the correct user email.
37  *
38  * User1@email.com registered successfully!
39  * User1@email.com is not registered!
40  * User1@email.com is not logged in!
41  * User1@email.com is already registered!
42  * User1@email.com logged in successfully!
43  * User1@email.com logged out successfully!
44  * User1@email.com is already logged in!
45  * User1@email.com is already logged in from another location!
46  * User1@email.com is blocked!
47  * User1@email.com is not allowed to login from this location!
48  * User1@email.com password is incorrect!
49  */
50
51  class User{
52
53  }
54  class ApplicationAuthState{
```

Test Results Custom Input

12:12 04-08-2023

HackerRank Question - C#: Auth

hackerrank.com/test/cbe54fca9lf/questions/4rci26qmclc

5h 52m left

ALL

1 2 3 4 5

- that the password is correct
- that the user is not logged in already
- If all tests pass
  - add the user to the *UsersLoggedIn* list
  - reset the *IncorrectAttempt* property to 0
  - return the success message
- Otherwise
  - increment the *IncorrectAttempt* property
  - return the appropriate message
- the public *Logout* method -
  - The method takes a parameter user of type *IUser* and returns a string
  - It verifies
    - that the user is logged in
  - If the user is logged in, remove the user from the *UsersLoggedIn* list and return the success message.
  - If the user is not logged in, return the appropriate message.

▸ Input Format for Custom Testing

▾ Sample Case 0

Sample Input

```
1
location4
4
```

Language: C#

Autocomplete Disabled

```
 * These strings can be copied and pasted to avoid typing errors.
 * User1@email.com should be replaced with the correct user email.
 *
 * User1@email.com registered successfully!
 * User1@email.com is not registered!
 * User1@email.com is not logged in!
 * User1@email.com is already registered!
 * User1@email.com logged in successfully!
 * User1@email.com logged out successfully!
 * User1@email.com is already logged in!
 * User1@email.com is already logged in from another location!
 * User1@email.com is blocked!
 * User1@email.com is not allowed to login from this location!
 * User1@email.com password is incorrect!
 */

class User{

}
class ApplicationAuthState{
```

Test Results Custom Input

12:16 04-08-2023

HackerRank Question - C#: Auth

hackerrank.com/test/cbe54fca9lf/questions/4rci26qmclc

5h 51m left

ALL

1 2 3 4 5

▾ Input Format for Custom Testing

The first line contains an integer *n*, the number of allowed locations.
Each of the next *n* lines contains the allowed location name.
The next line contains an integer *m*, the number of users.
Each of the next *m* lines contains the (*Id, Email, Password, Location*) of the user information.
The next line contains an integer *k*, the number of operations.
Each of the next *k* lines contains the Function name and User index separated by ":"

▾ Sample Case 0

Sample Input

```
1
location4
4
16,user16@email.com,14165,location16
8,user8@email.com,89680,location8
17,user17@email.com,26883,location17
16,user16@email.com,36862,location16
10
Register:2
Register:2
Login:1
Register:2
Login:0
Login:0
```

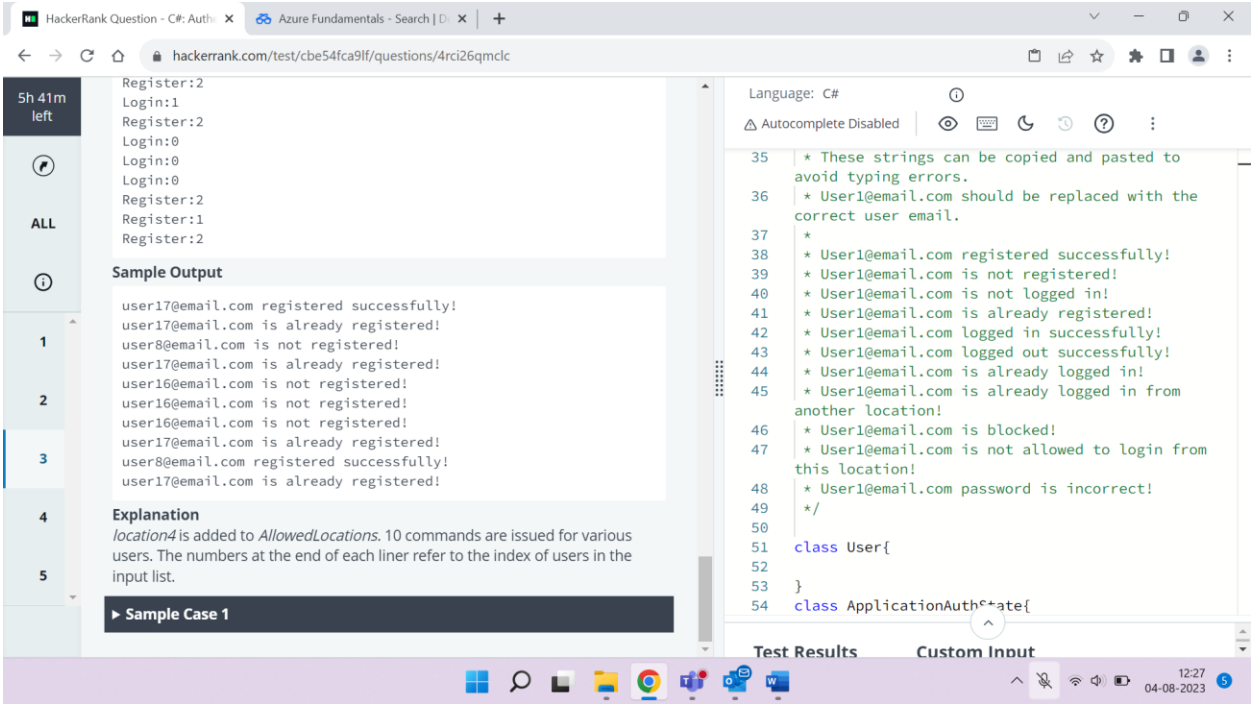
Language: C#

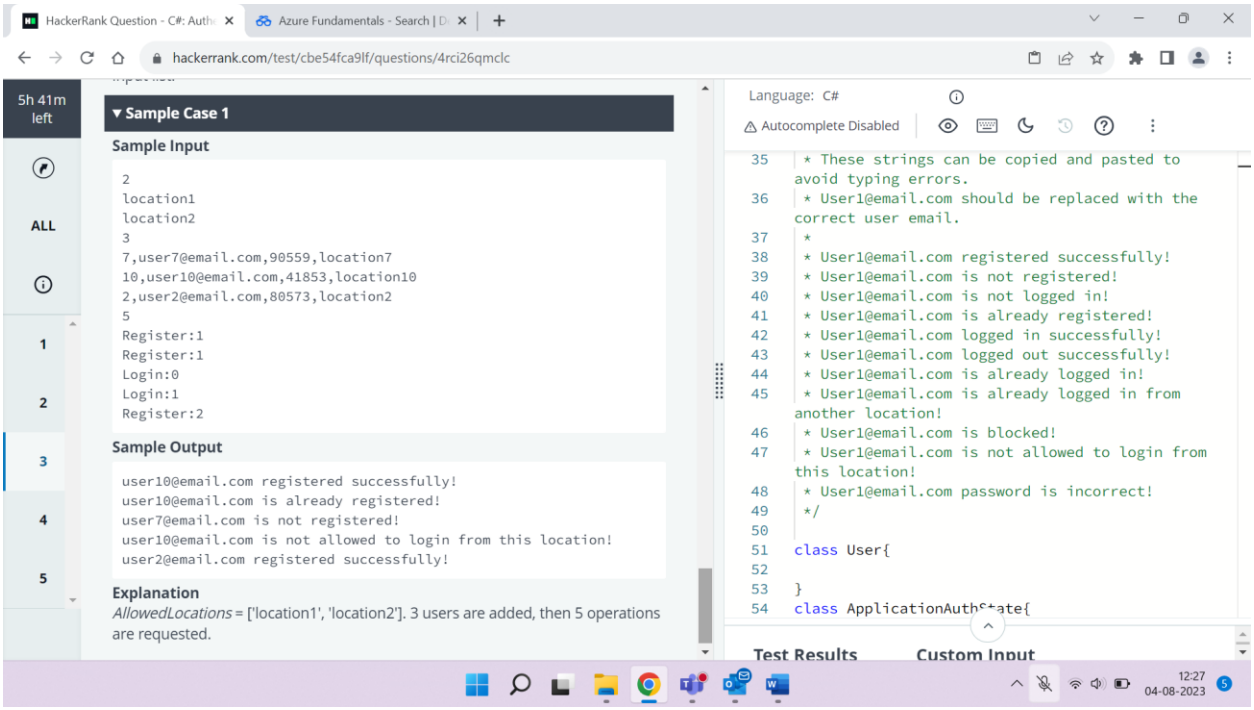
Autocomplete Disabled

```
 * These strings can be copied and pasted to avoid typing errors.
 * User1@email.com should be replaced with the correct user email.
 *
 * User1@email.com registered successfully!
 * User1@email.com is not registered!
 * User1@email.com is not logged in!
 * User1@email.com is already registered!
 * User1@email.com logged in successfully!
 * User1@email.com logged out successfully!
 * User1@email.com is already logged in!
 * User1@email.com is already logged in from another location!
 * User1@email.com is blocked!
 * User1@email.com is not allowed to login from this location!
 * User1@email.com password is incorrect!
 */

class User{

}
class ApplicationAuthState{
```

Test Results Custom Input

12:17 04-08-2023





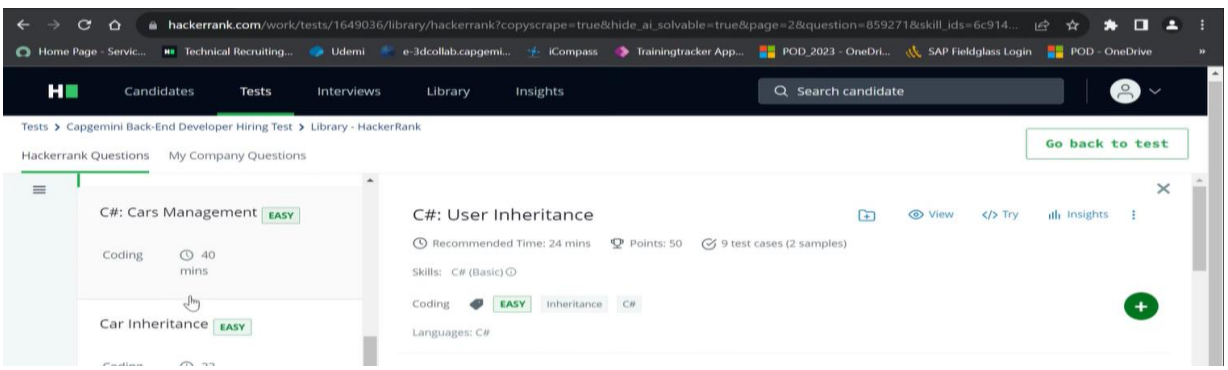

```
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Globalization;
```

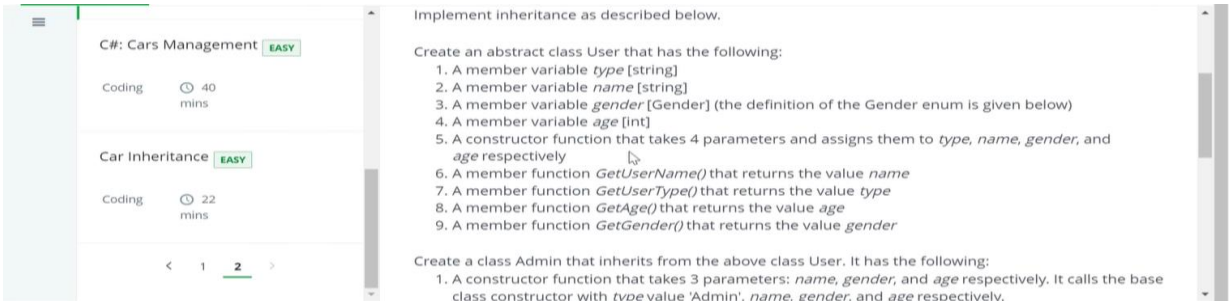
```
using System.IO;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;

interface IUser
{
    public int Id { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public int IncorrectAttempt { get; set; }
    public string Location { get; set; }
}

interface IApplicationAuthState
{
    public List<IUser> RegisteredUsers { get; set; }
    public List<IUser> UsersLoggedIn { get; set; }
    public List<string> AllowedLocations { get; set; }
    string Register(IUser user);
    string Login(IUser user);
    string Logout(IUser user);
}

/*
 * These strings can be copied and pasted to avoid typing errors.
 * User1@email.com should be replaced with the correct user email.
 *
 * User1@email.com registered successfully!
 * User1@email.com is not registered!
 * User1@email.com is not logged in!
 * User1@email.com is already registered!
 * User1@email.com logged in successfully!
 * User1@email.com logged out successfully!
 * User1@email.com is already logged in!
 * User1@email.com is already logged in from another location!
 * User1@email.com is blocked!
 * User1@email.com is not allowed to login from this location!
 * User1@email.com password is incorrect!
 */
```

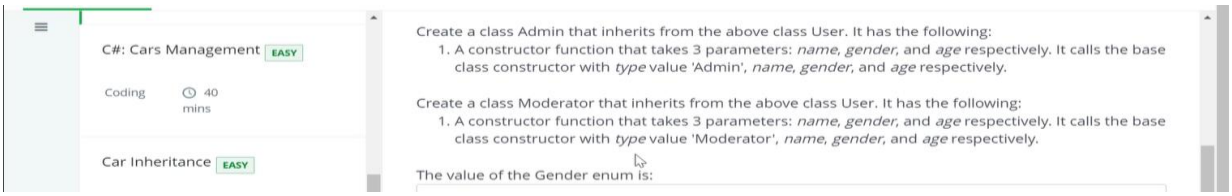
```
class User{

}
class ApplicationAuthState{

}

class Solution
{
    public static void Main(string[] args)
    {
        TextWriter textWriter = new StreamWriter(@System.Environmen
t.GetEnvironmentVariable("OUTPUT_PATH"), true);
        List<IUser> users = new List<IUser>();
        List<string> allowedLocations = new List<string>();
        int allowedLocationCount = Convert.ToInt32(Console.ReadLine
().Trim());
        for (int i = 0; i < allowedLocationCount; i++)
        {
            var a = Console.ReadLine().Trim();
           allowedLocations.Add(a);
        }
        ApplicationAuthState authState = new ApplicationAuthState(a
llowedLocations);

        int usersCount = Convert.ToInt32(Console.ReadLine().Trim())
;
        for (int i = 0; i < usersCount; i++)
        {
            var a = Console.ReadLine().Trim().Split(',');
            users.Add(new User(Convert.ToInt32(a[0]), a[1], a[2], a
[3]));
        }

        int actionCount = Convert.ToInt32(Console.ReadLine().Trim()
);
        for (int i = 0; i < actionCount; i++)
        {
            var a = Console.ReadLine().Trim().Split(':');
            var uIndex = Convert.ToInt32(a[1]);
            if(a[0] == "Register"){
```

```
                textWriter.WriteLine(authState.Register(users[uInde
x]));
            } else if(a[0] == "Login"){
                textWriter.WriteLine(authState.Login(users[uIndex])
);
            } else if (a[0] == "Logout"){
                textWriter.WriteLine(authState.Logout(users[uIndex]
));
            }
        }

        textWriter.Flush();
        textWriter.Close();
    }
}
```

12.User Inheritance







C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

1 2

The value of the Gender enum is:

```
public enum Gender
{
   Male,
   Female,
   Other
}
```

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the function calls. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the stubbed code.

▶ **Input Format For Custom Testing**

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

1 2

▼ **Input Format For Custom Testing**

The first line contains 3 space-separated strings for building the Admin object, where first is the name of the user, second is the gender of the user, and third is the age of the user respectively.
The second line contains 3 space-separated strings for building the Moderator object, where first is the name of the user, second is the gender of the user, and third is the age of the user respectively.

▼ **Sample Case 0**

Sample Input For Custom Testing

```
Oscar Male 23
Abel Female 36
```

Sample Output

```
Type of user Oscar is Admin
Age of user Oscar is 23
Gender of user Oscar is Male
```

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

1 2

Sample Input For Custom Testing

```
Oscar Male 23
Abel Female 36
```

Sample Output

```
Type of user Oscar is Admin
Age of user Oscar is 23
Gender of user Oscar is Male
Type of user Abel is Moderator
Age of user Abel is 36
Gender of user Abel is Female
```

Explanation

First, a user is created with *type* 'Admin', *name* 'Oscar', *gender* 'Male', and *age* 23. Then, all 4 functions are called: *GetUserName, GetUserType, GetAge,* and *GetGender*. The result is printed to the standard output. The same process occurs for the second user.

▶ **Sample Case 1**

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

1 2

Sample Input For Custom Testing

```
Ace Male 29
Ali Female 36
```

Sample Output

```
Type of user Ace is Admin
Age of user Ace is 29
Gender of user Ace is Male
Type of user Ali is Moderator
Age of user Ali is 36
Gender of user Ali is Female
```

Explanation

First, a user is created with *type* 'Admin', *name* 'Ace', *gender* 'Male', and *age* 29. Then, all 4 functions are called: *GetUserName, GetUserType, GetAge,* and *GetGender*. The result is printed to the standard output. The same process occurs for the second user.

13. Cars Management

hackerrank.com/work/tests/1649036/library/hackerrank?copyscrape=true&hide_ai_solvable=true&page=2&question=858037&skill_ids=6c914...

Home Page - Servic... | Technical Recruiting... | Udemi | e-3dcollab.capgemi... | iCompass | Trainingtracker App... | POD_2023 - OneDri... | SAP Fieldglass Login | POD - OneDrive

Candidates | Tests | Interviews | Library | Insights | Search candidate

Tests > Capgemini Back-End Developer Hiring Test > Library - HackerRank

Hackerrank Questions | My Company Questions | Go back to test

Coding 35 mins

C#: User Inheritance EASY

Coding 24 mins

C#: Cars Management

View | Try | Insights

Recommended Time: 40 mins | Points: 50 | 12 test cases (2 samples)

Skills: C# (Basic)

Coding EASY C# LINQ

Languages: C#

Coding 35 mins

C#: User Inheritance EASY

Coding 24 mins

C#: Cars Management EASY

Given a list of data, implement the following 4 methods using LINQ in a class that allows for managing cars:

1. *MostExpensiveCar* - returns the most expensive car in the car list.
2. *CheapestCar* - returns the cheapest car in the car list.
3. *AveragePriceOfCars* - returns the average price of all cars in the car list. The average price is rounded to the nearest whole number.
4. *MostExpensiveModelForEachBrand* - calculates the most expensive car for each unique brand and returns the results as a **Dictionary<string, Car>** sorted by key, where Key[string] is a unique brand, and Value[Car] is the car with the maximum price in this brand.

Here is the description of the Car class:

Coding 35 mins

C#: User Inheritance EASY

Coding 24 mins

C#: Cars Management EASY

Coding 40 mins

Here is the description of the Car class:

- *Brand* [string] - the brand of the car.
- *Model* [string] - the model of the car.
- *Price* [int] - the price of the car

Your implementation of the class will be tested by a stubbed code on several input files. The input file contains parameters for the function calls (i.e., the car data). The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the stubbed code.

▶ **Input Format For Custom Testing**

▶ **Sample Case 0**

▶ **Sample Case 1**

C#: User Inheritance EASY

Coding 24 mins

C#: Cars Management EASY

Coding 40 mins

▼ **Input Format For Custom Testing**

The first line contains an integer, *n*, denoting the number of cars on which operations have to be performed.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains space-separated strings, such that the first of them is the brand of the car, the second is the model of the car, and third is the price of the car, respectively.

▶ **Sample Case 0**

▶ **Sample Case 1**

Coding 35 mins

C#: User Inheritance EASY

Coding 24 mins

C#: Cars Management EASY

Coding 40 mins

▼ Sample Case 0

Sample Input For Custom Testing

```
12
Audi A6 45447
BMW 5_Series 49198
Mercury Cougar 47560
Audi A5 79105
Cadillac CTS 49772
Audi Q7 52927
Nissan Altima 72154
Isuzu Amigo 65135
BMW 7_Series 78000
Pontiac Bonneville 58565
Nissan Quest 55357
BMW X5 65000
```

Sample Output

```
The most expensive car is Audi A5
```

14. Car Inheritance

hackerrank.com/work/tests/1649036/library/hackerrank?copyscrape=true&hide_ai_solvable=true&page=2&question=317734&skill_ids=6c914...

Home Page - Servic... | Technical Recruiting... | Udemi | e-3dcollab.capgemi... | iCompass | Trainingtracker App... | POD_2023 - OneDri... | SAP Fieldglass Login | POD - OneDrive

Candidates | Tests | Interviews | Library | Insights | Search candidate

Tests > Capgemini Back-End Developer Hiring Test > Library - HackerRank

Hackerrank Questions | My Company Questions | Go back to test

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Car Inheritance

View | Try | Insights

Recommended Time: 22 mins | Points: 50 | 9 test cases (3 samples)

Skills: C# (Basic), C++ (Intermediate), Java (Basic)

Coding | EASY | Polymorphism | Abstraction | Java | C++ | C#

Languages: C++, C++14, C#, Java 7, Java 8

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

< 1 2 >

Build on an abstract class and initialize an instance of each class with a variable. The program will then test the implementation by retrieving the stored data.

The locked code in the editor does the following:

1. Declares an abstract class named Car with the implementations for `getIsSedan()` and `getSeats()` methods and an abstract method named `getMileage()`.
2. Creates *WagonR*, *HondaCity*, or *InnovaCrysta* object based on input (0 for WagonR, 1 for HondaCity and 2 for InnovaCrysta).
3. Calls the `getIsSedan()`, `getSeats()`, and `getMileage()` methods on the object.

UML Class Diagram

Car

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

< 1 2 >

UML Class Diagram

Car

isSedan: Boolean
seats: String

+ Car(isSedan: Boolean, seats: String)
+ getIsSedan(): Boolean
+ getSeats(): String
+ getMileage(): String

WagonR | HondaCity | InnovaCrysta

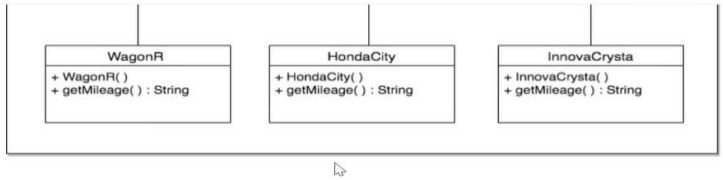
C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

< 1 2 >



The details for each car are provided below:

1. WagonR is not a sedan and has 4 seats.
2. HondaCity is a sedan and has 4 seats.
3. InnovaCrysta is not a sedan and has 6 seats.

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

< 1 2 >

Function Description

Complete the code in the editor below to implement the following:

1. Create **classes** named `WagonR`, `HondaCity`, and `InnovaCrysta` that all inherit from the `Car` class.
2. Each class must have a **constructor** that receives one integer argument representing the mileage of the car.
3. Each class must implement a `getMileage()` **method** which returns a string in the form of `'<mileage> kmpl'` where `<mileage>` is the value provided to the constructor.

Constraints

- 0 ≤ type of car ≤ 2
- 5 ≤ mileage ≤ 30

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

< 1 2 >

▼ Input Format For Custom Testing

The first line contains an integer that describes the type of car to instantiate.
The second line contains an integer, the mileage of the car.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
0     →   type of car to instantiate = 0 (WagonR)
22    →   mileage = 22
```

Sample Output

```
A WagonR is not Sedan, is 4-seater, and has a mileage of around 22 kmpl.
```

C#: Cars Management EASY

Coding 40 mins

Car Inheritance EASY

Coding 22 mins

< 1 2 >

0 → type of car to instantiate = 0 (WagonR)
22 → mileage = 22

Sample Output

A WagonR is not Sedan, is 4-seater, and has a mileage of around 22 kmpl.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
1     →   type of car to instantiate = 1 (HondaCity)
12    →   mileage = 12
```

Sample Output

```
A HondaCity is Sedan, is 4-seater, and has a mileage of around 12 kmpl.
```

© 2023 HackerRank Privacy policy Terms of service API Support Status Academy

15 User Management:

hackerrank.com/test/ep45skjo76l/questions/b2pnasdpnb5

48m left

ALL

2

2. Question 2

The manage users page in a company administrative portal contains a user list. The admin can add new users, update current users, and save all changes by pressing the save button. Given one list that contains updated and added users, count updated and added users.

As a software developer for this company, implement the backend side of this user management page. Given two user lists, one is the current database and the other a list of possible changes. The first value in a record is its user ID. If this value is 0, it is a new user to insert into the database. If there is another user ID, compare all of the user's attributes with what is in the database. If there is any difference, the record is updated.

The implementation should return two lists. The first is a list of records to update. The second is a list of records to insert.

Example:

List 1 (Current Users in DB)	List 2 (Users List from UI)
1 User1	1 User4
2 User2	0 User5

Language: C#

Autocomplete Ready

```
37
38 static class UserManager
39 {
40         /*
41      * Complete the 'CompareUsers' function below.
42      *
43      * The function is expected to return two
   lists which are updated and inserted.
44      * The function accepts two lists
   usersListInDB and newUsersList as parameter.
45      */
46     public static (List<User> updated, List<User>
   inserted) CompareUsers(List<User> usersListInDB,
   List<User> newUsersList)
47     {
48
49     }
50 }
51
52
53
54
55
56 > class Solution···
```

Test Results Custom Input

48m left

ALL

2

Example:

List 1 (Current Users in DB)	List 2 (Users List from UI)
1 User1	1 User4
2 User2	0 User5
	2 User2

User ID 1 changes the username from 'User1' to 'User4'.
In the second line, the user ID is 0. A new user is created with the username 'User5'.
User ID 2 matches what is in the database, so no changes are made.
Return two lists, ['1 User1'] and ['0 User5'].

Function Description
Implement the *CompareUsers* function as described.

CompareUsers has the following parameters:
List<User> usersListInDB: the current database
List<User> newUsersList: the new transactions

Return
List<User> updated
List<User> inserted

Language: C#

Autocomplete Ready

```
37
38 static class UserManager
39 {
40         /*
41      * Complete the 'CompareUsers' function below.
42      *
43      * The function is expected to return two
   lists which are updated and inserted.
44      * The function accepts two lists
   usersListInDB and newUsersList as parameter.
45      */
46     public static (List<User> updated, List<User>
   inserted) CompareUsers(List<User> usersListInDB,
   List<User> newUsersList)
47     {
48
49     }
50 }
51
52
53
54
55
56 > class Solution···
```

Test Results Custom Input

hackerrank.com/test/ep45skjo76l/questions/b2pnasdpnb5

47m left

ALL

2

▾ Input Format For Custom Testing

The first line contains an integer *n*, the count of users in the database.
Each of the next *n* lines contains the comma-separated user information of one user.
The next line contains an integer *m*, the number of users to compare.
Each of the next *m* lines contains the comma-separated user information to process.

▾ Sample Case 0

Sample Input For Custom Testing

```
4
1,0,First0,Last0,40,1993.05.23,email0@gmail.com,Male,Country0,City0,Address0,ZipCode0,PhoneNumber0,Department0,Role0,2016.07.18,656,Active
2,7984,First1,Last1,50,1997.01.05,email1@gmail.com,Female,Country1,City1,Address1,ZipCode1,PhoneNumber1,Department1,Role1,2018.10.08,778,Inactive
3,5968,First2,Last2,0,1999.03.20,email2@gmail.com,Male,Country2,City2,Address2,ZipCode2,PhoneNumber2,Department2,Role2,2019.10.16,734,Active
4,3952,First3,Last3,10,1989.04.02,email3@gmail.com,Female,Country3,City3,Address3,ZipCode3,PhoneNumber3,Department3,Role3,2016.11.05,858,Inactive
2
1,0,First0,Last0,40,1993.05.09,email0@gmail.com,Male,Country0,City0,Address0,ZipCode0,PhoneNumber0,Department0,Role0,2017.02.24,176,Active
```

Language: C#

Autocomplete Ready

```
static class UserManager
{
        /*
      * Complete the 'CompareUsers' function below.
      *
      * The function is expected to return two lists which are updated and inserted.
      * The function accepts two lists usersListInDB and newUsersList as parameter.
      */
      public static (List<User> updated, List<User> inserted) CompareUsers(List<User> usersListInDB, List<User> newUsersList)
      {

      }
}
class Solution ···
```

Test Results **Custom Input**

hackerrank.com/test/ep45skjo76l/questions/b2pnasdpnb5

47m left

ALL

2

```
3,5968,First2,Last2,0,1999.03.20,email2@gmail.com,Male,Country2,City2,Address2,ZipCode2,PhoneNumber2,Department2,Role2,2019.10.16,734,Active
4,3952,First3,Last3,10,1989.04.02,email3@gmail.com,Female,Country3,City3,Address3,ZipCode3,PhoneNumber3,Department3,Role3,2016.11.05,858,Inactive
2
1,0,First0,Last0,40,1993.05.09,email0@gmail.com,Male,Country0,City0,Address0,ZipCode0,PhoneNumber0,Department0,Role0,2017.02.24,176,Active
2,7984,First1,Last1,50,1997.01.05,email1@gmail.com,Female,Country1,City1,Address1,ZipCode1,PhoneNumber1,Department1,Role1,2018.10.08,778,Inactive
```

Sample Output

```
Updated Users:1
Inserted Users:0
```

Explanation

There are 4 users in the database initially, and there are 2 users to compare.

- The date value "1993.05.23" is updated to "1993.05.09" in the first line. Add the record to the *updated* list.
- The user in the second line matches the user in the database, so no update or insertion occurs.

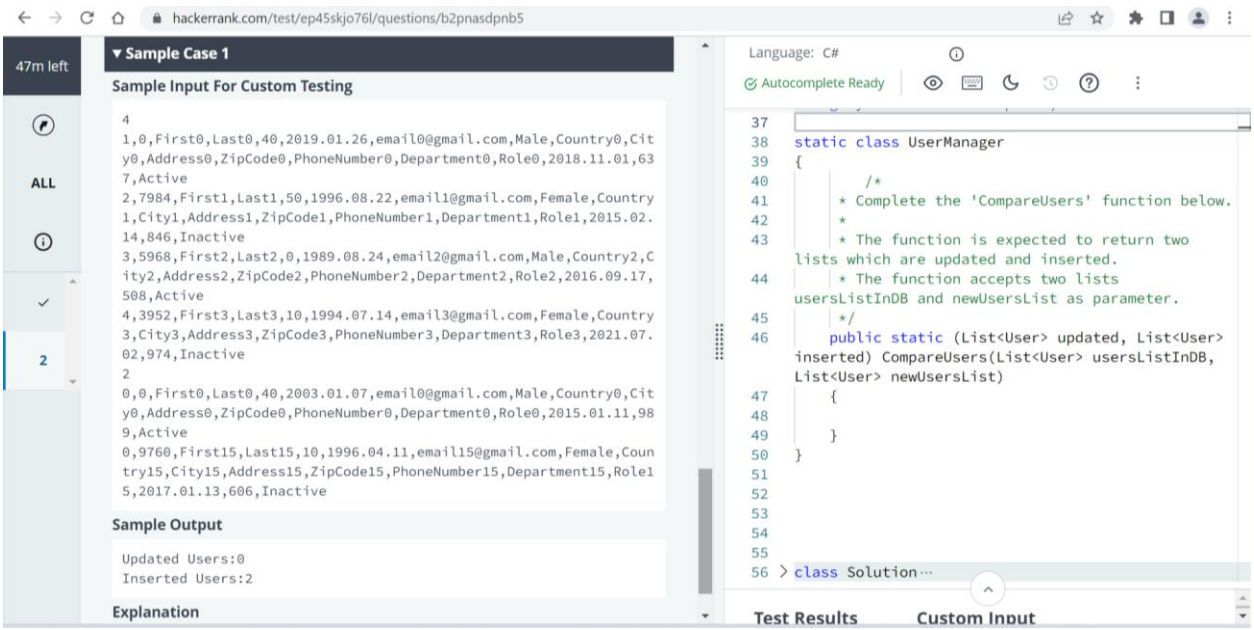
▸ Sample Case 1

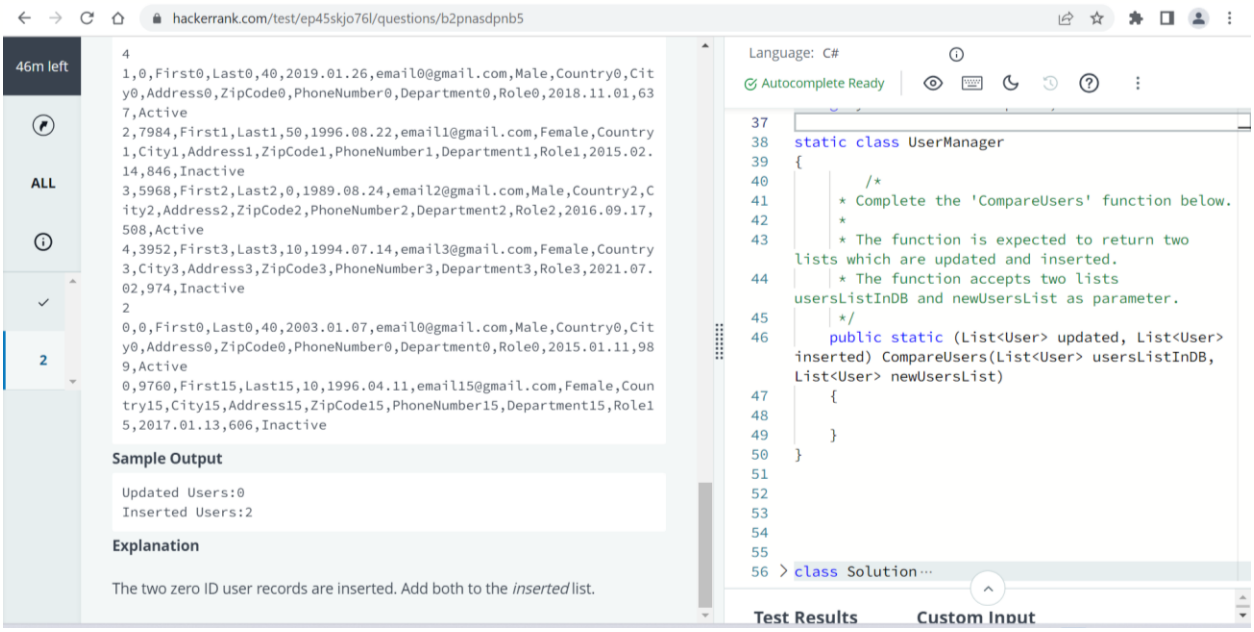
Language: C#

Autocomplete Ready

```
static class UserManager
{
        /*
      * Complete the 'CompareUsers' function below.
      *
      * The function is expected to return two lists which are updated and inserted.
      * The function accepts two lists usersListInDB and newUsersList as parameter.
      */
      public static (List<User> updated, List<User> inserted) CompareUsers(List<User> usersListInDB, List<User> newUsersList)
      {

      }
}
class Solution ···
```

Test Results **Custom Input**





Code:

```
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Globalization;
using System.IO;
using System.Linq;
```

```
using System.Reflection;
using System.Runtime.Serialization;
using System.Text.RegularExpressions;
using System.Text;
using System;


class User
{
    public int Id { get; set; }
    public string IdentityNumber { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
    public DateOnly? BirthDate { get; set; }
    public string Email { get; set; }
    public string Gender { get; set; }
    public string Country { get; set; }
    public string City { get; set; }
    public string Address { get; set; }
    public string ZipCode { get; set; }
    public string PhoneNumber { get; set; }
    public string Department { get; set; }
    public string Roles { get; set; }
    public DateOnly? JoinDate { get; set; }
    public decimal Credit { get; set; }
    public string Status { get; set; }
}

static class UserManager
{
        /*
     * Complete the 'CompareUsers' function below.
     *
     * The function is expected to return two lists which are updat
ed and inserted.
     * The function accepts two lists usersListInDB and newUsersLis
t as parameter.
     */

    }
}
```

```
class Solution
{
    public static void Main(string[] args)
    {
        TextWriter textWriter = new StreamWriter(@System.Environmen
t.GetEnvironmentVariable("OUTPUT_PATH"), true);
        List<User> usersListInDB = new List<User>();
        List<User> newUsersList = new List<User>();

        int userInDbCount = Convert.ToInt32(Console.ReadLine().Trim
());
        for (int i = 0; i < userInDbCount; i++)
        {
            var u = Console.ReadLine().Trim().Split(',');
            var usr = new User()
            {
                Id = string.IsNullOrEmpty(u[0]) ? 0 : Convert.ToInt
32(u[0]),
                IdentityNumber = u[1],
                FirstName = u[2],
                LastName = u[3],
                Age = string.IsNullOrEmpty(u[4]) ? 0 : Convert.ToIn
t32(u[4]),
                BirthDate = string.IsNullOrEmpty(u[5]) ? null : new
 DateOnly(Convert.ToInt32(u[5].Split('.')[0]), Convert.ToInt32(u[5]
.Split('.')[1]), Convert.ToInt32(u[5].Split('.')[2])),
                Email = u[6],
                Gender = u[7],
                Country = u[8],
                City = u[9],
                Address = u[10],
                ZipCode = u[11],
                PhoneNumber = u[12],
                Department = u[13],
                Roles = u[14],
                JoinDate = string.IsNullOrEmpty(u[15]) ? null : new
 DateOnly(Convert.ToInt32(u[15].Split('.')[0]), Convert.ToInt32(u[1
5].Split('.')[1]), Convert.ToInt32(u[15].Split('.')[2])),
```

```
                Credit = string.IsNullOrEmpty(u[16]) ? 0 : Convert.
ToDecimal(u[16]),
                Status = u[17]
            };
            usersListInDB.Add(usr);
        }
        int newUsersCount = Convert.ToInt32(Console.ReadLine().Trim
());
        for (int i = 0; i < newUsersCount; i++)
        {
            var u = Console.ReadLine().Trim().Split(',');
            var usr = new User()
            {
                Id = string.IsNullOrEmpty(u[0]) ? 0 : Convert.ToInt
32(u[0]),
                IdentityNumber = u[1],
                FirstName = u[2],
                LastName = u[3],
                Age = string.IsNullOrEmpty(u[4]) ? 0 : Convert.ToIn
t32(u[4]),
                BirthDate = string.IsNullOrEmpty(u[5]) ? null : new
 DateOnly(Convert.ToInt32(u[5].Split('.')[0]), Convert.ToInt32(u[5]
.Split('.')[1]), Convert.ToInt32(u[5].Split('.')[2])),
                Email = u[6],
                Gender = u[7],
                Country = u[8],
                City = u[9],
                Address = u[10],
                ZipCode = u[11],
                PhoneNumber = u[12],
                Department = u[13],
                Roles = u[14],
                JoinDate = string.IsNullOrEmpty(u[15]) ? null : new
 DateOnly(Convert.ToInt32(u[15].Split('.')[0]), Convert.ToInt32(u[1
5].Split('.')[1]), Convert.ToInt32(u[15].Split('.')[2])),
                Credit = string.IsNullOrEmpty(u[16]) ? 0 : Convert.
ToDecimal(u[16]),
                Status = u[17]
            };
            newUsersList.Add(usr);
        }
```

```
        var (updated, inserted) = UserManager.CompareUsers(usersLis
tInDB, newUsersList);
        textWriter.WriteLine("Updated Users:" + updated.Count);
        textWriter.WriteLine("Inserted Users:" + inserted.Count);
        textWriter.Flush();
        textWriter.Close();
    }
}
```