# Lab: Exceptions and Error Handling Lab

Problems for exercise and homework for the ["C# OOP" course @ SoftUni"](https://softuni.bg/trainings/3585/csharp-oop-february-2022).

You can check your solutions here: <https://judge.softuni.org/Contests/3324/Exceptions-and-Error-Handling-Lab>

## Square Root

Write a program that reads an integer **number** and **calculates** and **prints** its **square** **root**.

* If the number is negative, print "**Invalid number.**"
* In all cases finally, print "**Goodbye.**"

Use **try-catch-finally**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9 | 3  Goodbye. |
| -1 | Invalid number.  Goodbye. |

## Enter Numbers

Write a method **ReadNumber(int start, int end)** that enters an integer number in a given range (**start…end**), **excluding** the numbers **start** and **end**. If an **invalid number** or a **non-number** text is entered, the method should **throw an exception**. Using this method write a program that enters **10 numbers**: **a1, a2, … a10, such that 1 < a1 < … < a10 < 100**. If the user enters an invalid number, continue while there are 10 valid numbers entered. Print the array elements, separated with **comma and space “**, **”**.

### Hints

* When the entered input holds a non-integer value, print “**Invalid Number!**”
* When the entered input holds an integer that is out of range, print "**Your number is not in range {currentNumber} - 100!**"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  3  4  5  6  7  8  9  10  11 | 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 |
| 1  2  1  3  p  4  5  6  7  8  9  10  11 | Your number is not in range (1 - 100)  Your number is not in range (1 – 100)  Invalid Number!  2, 3, 4, 5, 6, 7, 8, 9, 10, 11 |

## Cards

Create a class **Card** to hold a card’s face and suit.

* Valid card faces are: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A
* Valid card suits are: S (♠), H (♥), D (♦), C (♣)

Both face and suit are expected as an uppercase string. The class also needs to have a toString() method that prints the card’s face and suit as a string in the following format:

**"[{face}{suit}]" – example: [A♠] [5♣] [10♦]**

Use the following UTF code literals to represent the suits:

* \u2660 – Spades (♠)
* \u2665 – Hearts (♥)
* \u2666 – Diamonds (♦)
* \u2663 – Clubs (♣)

Write a program that takes a deck of cards as a string array and prints them as a sequence of cards (space separated). Print an exception message **"Invalid card!"** when an invalid card definition is passed as input.

### Input

* A single line with the faces and suits of the cards in the format:

**"{face} {suit}, {face} {suit}, …"**

### Output

* As output, print on the console the list of cards as strings, separated by space.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| A S, 10 D, K H, 2 C | [A♠] [10♦] [K♥] [2♣] |
| 5 C, 10 D, king C, K C, Q heart, Q H | Invalid card!  Invalid card!  [5♣] [10♦] [K♣] [Q♥] |

### Hints

Write a method CreateCard(face, suit), which creates a card face and card suit and returns a Card object. The method should throw an exception if invalid data are given in its arguments. Later, you can catch the exception and print an error message.

## Sum of Integers

You will receive a sequence of **elements of different types**, separated by **space**. Your task is to calculate the sum of all valid integer numbers in the input. Try to add each element of the array to the sum and **write** **messages** for the possible **exceptions** while processing the element:

* If you receive an **element**, which is **not in the correct format** **(FormatException)**:  
  **"The element '{element}' is in wrong format!"**
* If you receive an **element**, which is **out of the integer** **type range (OverflowException)**:  
  **"The element '{element}' is out of range!"**

After each processed element add the following message:

**"Element '{element}' processed - current sum: {sum}"**

At the end print the total sum of all integers:

**"The total sum of all integers is: {sum}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2147483649 2 3.4 5 invalid 24 -4  2147483647 2 3.4 5 invalid 24 -4 | The element '2147483649' is out of range!  Element '2147483649' processed - current sum: 0  Element '2' processed - current sum: 2  The element '3.4' is in wrong format!  Element '3.4' processed - current sum: 2  Element '5' processed - current sum: 7  The element 'invalid' is in wrong format!  Element 'invalid' processed - current sum: 7  Element '24' processed - current sum: 31  Element '-4' processed - current sum: 27  The total sum of all integers is: 27 |
| 9876 string 10 -2147483649 -8 3 4.86555 8 | Element '9876' processed - current sum: 9876  The element 'string' is in wrong format!  Element 'string' processed - current sum: 9876  Element '10' processed - current sum: 9886  The element '-2147483649' is out of range!  Element '-2147483649' processed - current sum: 9886  Element '-8' processed - current sum: 9878  Element '3' processed - current sum: 9881  The element '4.86555' is in wrong format!  Element '4.86555' processed - current sum: 9881  Element '8' processed - current sum: 9889  The total sum of all integers is: 9889 |

## Play Catch

You will receive on the **first** line an **array** of **integers**. After that you will receive **commands**, which should **manipulate** the array:

* **"Replace {index} {element}"** – **Replace** the element at the given **index** with the given **element**.
* **"Print {startIndex} {endIndex}"** – **Print** the elements from the **start** index to the **end** index **inclusive**.
* **"Show {index}"** – **Print** the element at the **index**.

You have the task to **rewrite** the **messages** from the **exceptions** which can be **produced** from your **program**:

* If you receive an **index**, which does **not** **exist** in the **array** print:  
  **"**The index does not exist!**"**
* If you receive a **variable**, which is of **invalid** **type**:  
  **"**The variable is not in the correct format!**"**

When you catch **3** exceptions – **stop** the **input** and **print** the **elements** of the array separated with **"**, **"**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **1 2 3 4 5**  Replace 1 9  Replace 6 3  Show 3  Show peter  Show 6 | The index does not exist!  4  The variable is not in the correct format!  The index does not exist!  1, 9, 3, 4, 5 |
| 1 2 3 4 5  Replace 3 9  Print 1 4  Print -3 12  Print 1 5  Show 3  Show 12.3 | 2, 3, 9, 5  The index does not exist!  The index does not exist!  9  The variable is not in the correct format!  1, 2, 3, 9, 5 |

### Constraints

* The **elements** of the array will be in **integers** in the interval **[-2147483648…2147483647]**
* You will always receive a **valid** string for the **first** part of the **command**, but the **parameters** might be **invalid**
* In the “**Print**”command always be true **startIndex <= endIndex**
* You will always **receive** at least **3** exceptions

## Money Transactions

You will receive on the **first** line a collection of bank accounts, consisting of an **account number (integer)** and its **balance (double)**, in the following format:

**"{account number}-{account balance},{account number}-{account balance},…"**

After that, until the **"End"** command, you will receive **commands**, which should **manipulate** the given account`s balance:

* **"Deposit {account number} {sum}"** – **Add** the given sum to the given **account`s balance**.
* **"Withdraw {account number} {sum}"** – **Subtract** the given sum from the **account`s balance**.

Print the following **messages** from the **exceptions** which can be **produced** from your **program**:

* If you receive an invalid command:  
  **"Invalid command!"**
* If you receive an account, which does not exist:  
  **"**Invalid account!**"**
* If you receive the "Withdraw" command with the sum, which is bigger than the balance:  
  **"**Insufficient balance!**"**

In all cases, after each received command, print the message:

**"Enter another command"**

After each successful operation print, the new balance is formatted to the second integer after the decimal point:

**"Account {account number} has new balance: {balance}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1-45.67,2-3256.09,3-97.34  Deposit 1 50  Withdraw 3 100  End | Account 1 has new balance: 95.67  Enter another command  Insufficient balance!  Enter another command |
| 1473653-97.34,44643345-2347.90  Withdraw 1473653 150.50  Deposit 44643345 200  Block 1473653 30  Deposit 1 30  End | Insufficient balance!  Enter another command  Account 44643345 has new balance: 2547.90  Enter another command  Invalid command!  Enter another command  Invalid account!  Enter another command |

### Constraints

* The types of the **elements** of the given command line will be **valid**
* You will always **receive** **3** **elements** in each command line