# Exercises: Generics

Problems for exercises and homework for the ["CSharp Advanced" course @ Software University](https://softuni.bg/courses/csharp-advanced).

You can check your solutions here: <https://judge.softuni.bg/Contests/1475/Generics-Exercises>

## Generic Box of String

Create a generic class **Box** that can be initialized with any type and store the value. Override **ToString()** method method and print the type and stored value in format {class full name: value}. On the first line, you will get **n** - the number of strings to read from the console. On the next **n** lines, you will get the actual strings. For each of them create a box and call its **ToString()** method to print its data on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  life in a box  box in a life | System.String: life in a box  System.String: box in a life |

## Generic Box of Integer

Use the description of the previous problem but now, test your generic box with Integers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  7  123  42 | System.Int32: 7  System.Int32: 123  System.Int32: 42 |

## Generic Swap Method Strings

Create a generic method that receives a list, containing any type of data and swaps the elements at two given indexes.

As in the previous problems read **n** number of boxes of type String and add them to the list. On the next line, however you will receive a swap command consisting of two indexes. Use the method you've created to swap the elements that correspond to the given indexes and print each element in the list.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Pesho  Gosho  Swap me with Pesho  0 2 | System.String: Swap me with Pesho  System.String: Gosho  System.String: Pesho |

## Generic Swap Method Integers

Use the description of the previous problem but now, test your list of generic boxes with Integers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  7  123  42  0 2 | System.Int32: 42  System.Int32: 123  System.Int32: 7 |

## Generic Count Method Strings

Create a **method** that receives as an argument a **list of any type that can be compared** and an **element of the given type**. The method should **return the count of the elements that are greater than the value of the given element**. **Modify your Box class** to support **comparison by value** of the stored data.

On the first line, you will receive **n** - the number of elements to add to the list. On the next **n** lines, you will receive the actual elements. On the last line, you will get the value of the element to which you need to compare every element in the list.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  aa  aaa  bb  aa | 2 |

## Generic Count Method Doubles

Use the description of the previous problem but now, test your list of generic boxes with **doubles**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  7.13  123.22  42.78  7.55 | 2 |

## Tuple

There is something, really annoying in C#. It is called a [**Tuple**](https://msdn.microsoft.com/en-us/library/system.tuple(v=vs.110).aspx). It is a class, which may store a few objects but let’s focus on the type of Tuple which contains two objects. The first one is “**item1**” and the second one is “**item2**”. It is kind of like a **KeyValuePair** except – it **simply has items,** which are **neither key nor value**. The annoyance is coming from the fact, that you have no idea what these objects are holding. The class name is telling you nothing, the methods, which it has – also. So, let’s say for some reason we would like to try to implement it by ourselves.

The task: Create a class “**Tuple**”, which is holding two objects. Like we said, the first one, will be “**item1**” and the second one - “**item2**”. The tricky part here is to make the class hold generics. This means, that when you create a new object of class - “**Tuple**”, there should be a way to explicitly, specify both the items’ type separately.

### Input

The input consists of three lines:

* The first one is holding a person name and an address. They are separated by space(s). Your task is to collect them in the tuple and print them on the console. Format of the input:

**<first name> <last name>>** **<address>**

* The second line holds a **name** of a personand the **amount of beer** (int) he can drink. Format:

**<name> <liters of beer>**

* The last line will hold an **Integer** and a **Double**. Format:

**<Integer> <Double>**

### Output

* Print the tuples’ items in format: {**item1**} -> {**item2**}

### Constraints

Use the good practices we have learned. Create the class and make it have getters and setters for its class variables. The input will be valid, no need to check it explicitly!

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sofka Tripova Stolipinovo  Az 2  23 21.23212321 | Sofka Tripova -> Stolipinovo  Az -> 2   1. -> 21.23212321 |

## Threeuple

Create a Class **Threeuple**. Its name is telling us, that it will hold no longer, just a pair of objects. The task is simple, our **Threeuple** should **hold three objects**. Make it have getters and setters. You can even extend the previous class

### Input

The input consists of three lines:

* The first one is holding a name, an address and a town. Format of the input:

**<<first name> <last name>> <address> <town>**

* The second line is holding a name, beer liters, and a Boolean variable with value - **drunk** or **not**. Format:

**<name> <liters of beer> <drunk or not>**

* The last line will hold a name, a bank balance (double) and a bank name. Format:

**<name> <account balance> <bank name>**

### Output

* Print the Threeuples’ objects in format: {**firstElement**} -> {**secondElement**} -> {**thirdElement**}

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sofka Tripova Stolipinovo Plovdiv  MitkoShtaigata 18 drunk  SashoKompota 0.10 NkqfaBanka | Sofka Tripova -> Stolipinovo -> Plovdiv  MitkoShtaigata -> 18 -> True  SashoKompota -> 0.1 -> NkqfaBanka |
| Ivan Ivanov Tepeto Plovdiv  Mitko 18 not  Sasho 0.10 NGB | Ivan Ivanov -> Tepeto -> Plovdiv  Mitko -> 18 -> False  Sasho -> 0.1 -> NGB |

### Note

You may extend your previous solution.