# Exercises: Generics

[Judge](https://judge.softuni.org/Contests/1527)

## Generic Box

Create a **generic class Box** that can store any type. **Override** the **toString()** method to print the type and the value of the stored data in the format "**{class full name}: {value}**"**.**

Use the class that you've created and test it with the class **java.lang.String**. 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 | java.lang.String: life in a box  java.lang.String: box in a life |
| 1  I am a programmer | java.lang.String: I am a programmer |

## 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 | java.lang.Integer: 7  java.lang.Integer: 123  java.lang.Integer: 42 |
| 5  12  13  14  15  16 | java.lang.Integer: 12  java.lang.Integer: 13  java.lang.Integer: 14  java.lang.Integer: 15  java.lang.Integer: 16 |

## 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 corresponding to the given indexes and **print** **each** element in the list.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Peter  George  Swap me with Peter  0 2 | java.lang.String: Swap me with Peter  java.lang.String: George  java.lang.String: Peter |
| 2  Uni  Soft  0 1 | java.lang.String: Soft  java.lang.String: Uni |

## 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 | java.lang.Integer: 42  java.lang.Integer: 123  java.lang.Integer: 7 |
| 5  12  13  14  15  16  3 4 | java.lang.Integer: 12  java.lang.Integer: 13  java.lang.Integer: 14  java.lang.Integer: 16  java.lang.Integer: 15 |

## 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 elements that are greater than the value of the given element**.

**Modify your Box class** to support **comparing by the value** of the data stored.

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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| 3  aa  aaa  bb  aa | 2 | 6  a  b  c  d  e  f g | 0 |

## 

## Generic Count Method Doubles

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

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| 3  7.13  123.22  42.78  7.55 | 2 | 1  1231542.123  1 | 1 |

## Custom List

Create a generic data structure that can store **any type** that can be **compared.** Implement functions:

* **void add(T element)**
* **T remove(int index)**
* **boolean contains(T element)**
* **void swap(int index, int index)**
* **int countGreaterThan(T element)**
* **T getMax()**
* **T getMin()**

Create a command interpreter that reads commands and modifies the custom list that you have created. Implement the commands:

* **Add {element}** - Adds the given element to the end of the list.
* **Remove {index}** - Removes the element at the given index.
* **Contains {element}** - Prints if the list contains the given element (**true** or **false**)**.**
* **Swap {index1} {index2}** - Swaps the elements at the given indexes.
* **Greater {element}** - Counts the elements that are greater than the given element and prints their count
* **Max** - Prints the maximum element in the list.
* **Min** - Prints the minimum element in the list.
* **Print** - Prints all elements in the list, each on a separate line.
* **END** - stops the reading of commands.

**Note**: For the **Judge tests,** use **String** as **T.**

**Examples**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| Add aa  Add bb  Add cc  Max  Min  Greater aa  Swap 0 2  Contains aa  Print  END | cc  aa  2  true  cc  bb  aa | Add Peter  Add George  Swap 0 0  Swap 1 1  Swap 0 1  Swap 1 0  Swap 0 1  Print  END | George Peter |

## Custom List Sorter

Extend the previous problem by creating an additional **Sorter class**. It should have a single static **method** **sort()** which can sort objects of type **CustomList** containing any type that can be compared. **Extend the command list** to support one additional command **Sort**:

* **Sort** - Sort the elements in the list in ascending order.

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| Add cc  Add bb  Add aa  Sort  Print  END | aa  bb  cc | Add Peter  Add George  Max  Sort  Print  END | Peter  George  Peter |

## \*Custom List Iterator

For the print command, you have probably used a **for** a loop. Extend your custom list class by making it implement **Iterable.** This should allow you to iterate your list in a **foreach** statement.

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| Add aa  Add bb  Add cc  Max  Min  Greater aa  Swap 0 2  Contains aa  Print  END | cc  aa  2  true  cc  bb  aa | Add Peter  Add George  Swap 0 0  Swap 1 1  Swap 0 1  Swap 1 0  Swap 0 1  Print  END | George  Peter |

## 10.\*Tuple

*There is something really annoying in the C# - language. It is called a* ***Tuple****. It is a class that contains two objects. The first one is* ***item1,*** *and the second one is* ***item2****. It is kind of like a* ***Map.Entry*** *except - it* ***simply has items*** *that are* ***neither key nor value****. The annoyance comes from the fact that you have no idea what these objects hold. The class name is telling you nothing, the methods which it has – too. So let's say we could try to implement it in Java, just for practicing generics.*

Create a class **Tuple**, which is holding two objects. As we said, the first one will be an **item1,** and the second one - an **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's name and an address. They are separated by space. Your task is to collect them in the tuple and print them on the console. Format:

"**{first name} {last name}** **{address}**"

* The second line holds a **person's name** and the **amount of beer** 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** |
| Sam Johnson Sofia  John 2  23 21.23212321 | Sam Johnson -> Sofia  John -> 2  23 -> 21.23212321 |
| Svetlin Nakov Tarnovo  Nakov 300  25 24.355555 | Svetlin Nakov -> Tarnovo  Nakov -> 300  25 -> 24.355555 |

## 11.\*Threeuple

Now you are aware of a Class, which is probably a bad practice to use. Anyway, it is a nice example of using generics. Our next task is to create another Tuple. This time, our task is harder.

Create a Class **Threeuple**. Its name tells us that it will no longer hold 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 holds a name, an address, and a town. Format of the input:

"**{first name} {last name}** **{address} {town}**"

* The second line holds a name, beer liters, and a **Boolean variable** - 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** |
| Sofia Jackson Izgrev Burgas  Max 18 drunk  Alex 0.10 DSK | Sofia Jackson -> Izgrev -> Burgas  Max -> 18 -> true  Alex -> 0.1 -> DSK |
| Peter Johnson Tepeto Plovdiv  Sam 18 not  Alex 0.10 NGB | Peter Johnson -> Tepeto -> Plovdiv  Sam -> 18 -> false  Alex -> 0.1 -> NGB |