Lab: Generics

Problems for exercises and homework for the "Java OOP Advanced" course @ SoftUni.

You can check your solutions here: https://judge.softuni.bg/Contests/521/Generics-Lab .

Part I: Generics

1. Jar of T

Create a class **Jar<>** that can store **anything**.

It should have two public methods:

- **`void add(element)`**
- **`element remove()`**

Adding should add on **top** of its contents. Remove should get the **topmost** element.

Examples

```
Jar<Pickle> jarOfPickles = new Jar<>();

jarOfPickles.add(new Pickle());
jarOfPickles.add(new Pickle());

Pickle pickle = jarOfPickles.remove();
```

Hints

Use the syntax **`Jar<T>`** to create a generic class

2. Generic Array Creator

Create a class **`ArrayCreator`** with a method and a single overload to it:

- **`static T[] create(int length, T item)`**
- **`static T[] create(Class<T>, int length, T item)`**

The method should return an array with the given length and every element should be set to the given default item.

Examples

```
String[] strings = ArrayCreator.create(10, "none");
Integer[] integers = ArrayCreator.create(Integer.class, 10, 0);
```

Part II: Type Parameter Bounds

3. Generic Scale

Create a class **Scale<T>** that holds two elements - left and right. The scale should receive the elements through its single constructor:

- **`Scale(T left, T right)`**

The scale should have a single method:

- **`T getHeavier()`**

The greater of the two elements is heavier. The method should return **`null`** if elements are equal.

Examples

```
Scale<String> stringScale = new Scale<>("a", "c");
System.out.println(stringScale.getHeavier());

Scale<Integer> integerScale = new Scale<>(1, 2);
System.out.println(integerScale.getHeavier());
```

4. List Utilities

Create a class **`ListUtils`** that you will use through several other exercises:

The class should have two static methods:

- **`T getMin(List<T> list)`**
- **`T getMax(List<T> list)`**

The methods should throw **`IllegalArgumentException`** if an empty list is passed.

Examples

```
List<Integer> integers = new ArrayList<>();
Collections.addAll(integers, 1, 2, 18, 2, -1);

Integer maxInteger = ListUtils.getMax(integers);

List<String> strings = new ArrayList<>();
Collections.addAll(strings, "a", "b", "c");

String minString = ListUtils.getMin(strings);
```

5. Null Finder

Add a method to your **`ListUtils`** class that finds the index of every **`null`** element in a given list with the method:

- **`static List<Integer> getNullIndices(List<> list)`**

Add the appropriate **generic syntax** to the signature. The method should work with any **`List<>`**.

Examples

```
List<Integer> integers = new ArrayList<>();
Collections.addAll(integers, 1, 2, null, 2, null);

List<Integer> IntegerNulls = ListUtils.getNullIndices(integers);

List<String> strings = new ArrayList<>();
Collections.addAll(strings, "a", null, "c");

List<Integer> StringNulls = ListUtils.getNullIndices(integers);
```

6. Generic Flat Method

In **ListUtils**, create a generic static method that flattens a **List<List<>>** into a resulting **List<>**

- **void flatten(List<> destination, List<List<>> source)**

Add the appropriate generic syntax to the signature. The method should work with any **List<>**.

Examples

```
List<Integer> integers = new ArrayList<>();
Collections.addAll(integers, 1,2,3);

List<Double> doubles = new ArrayList<>();
Collections.addAll(doubles, 1.2, 3.2);

List<List<? extends Number>> jagged = new ArrayList<>();
Collections.addAll(jagged, integers, doubles);

List<Number> dest = new ArrayList<>();
ListUtils.flatten(dest, jagged);
```

7. Generic Add All Method

In **ListUtils**, create a generic static method that **adds all elements from a given source** list **to a given destination** list with the static method:

- **void addAll(List<> destination, List<> source)**

Add the appropriate generic syntax to the signature. The method should work with any **List<>**.

Examples

```
List<Integer> integers = new ArrayList<>();
Collections.addAll(integers, 1, 2, null, 2, null);

List<Double> doubles = new ArrayList<>();
Collections.addAll(doubles, 1.2, 3.2, 5.5);

List<Number> destination = new ArrayList<>();

ListUtils.addAll(destination, integers);
ListUtils.addAll(destination, doubles);
```