Hands On Exercises - Generics

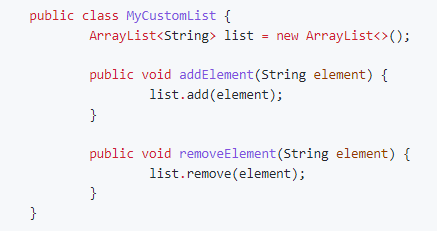
**Setup Instructions:**

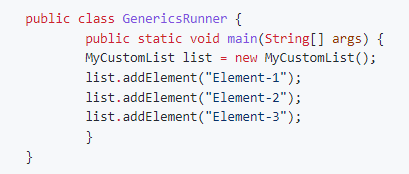
1. Use your Java Project “FullStackDay5”.
2. Create your classes in the package specified.
3. Define your classes as given below for each assignment.

Assignment 01

Let's look at a scenario where want to write a wrapper class around the ArrayList data structure, maybe to do some better custom error-checking and stuff. For now, we will just look at basic wrapper functionality, the error checking intent is just an excuse!

Create the following classes in package “**generics**”.



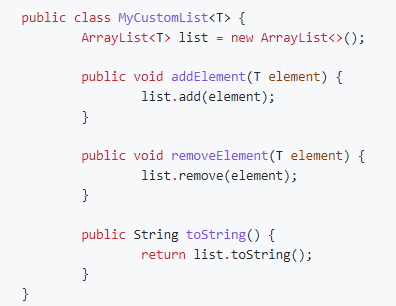


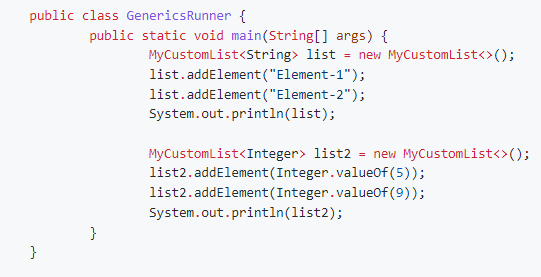
The MyCustomList class is a wrapper for ArrayList of Strings. Insertion and deletion of elements into this data structure is straightforward.

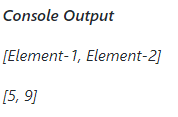
Let's say I would want to create MyCustomList for other types. Should we write additional wrapper classes MyCustomList and so on?

Let's look at an example:

Refactor **MyCustomList.java and GenericsRunner.java** as below:







The identifier T in the definition of the Generic class MyCustomList<T> is a placeholder for the actual type of the container. It is this placeholder that truly converts the MyCustomList class into a template.

The naming convention for these type placeholders is: \* Always use UpperCase letters of the alphabet (such as T, or S), or \* intuitive words such as TYPE

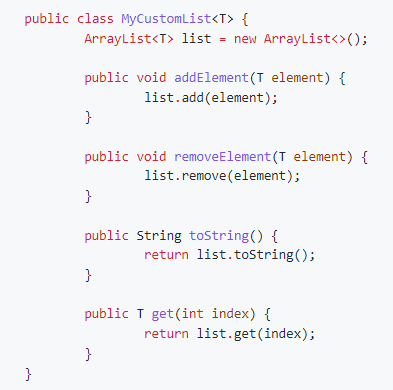
At the time of actual instantiation of MyCustomList inside GenericsRunner.main, this placeholder is substituted by the actual type:

* When MyCustomList<String> list is created, T is substituted by String
* When MyCustomList<Integer> list2 is created, T is substituted by Integer

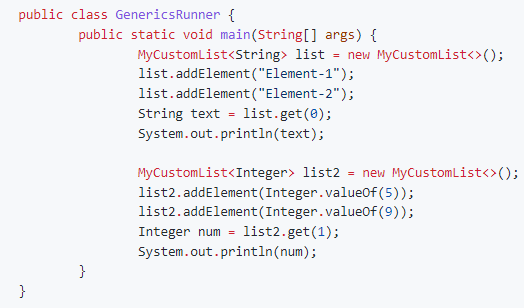
Assignment 02

Write a method get inside the generic class MyCustomList, which returns the element at a particular index (passed as argument) in its list storage.

#### **MyCustomList.java**



**GenericsRunner.java**



**Console Output**

Element-1

9

We have defined a method MyCustomList<T>.get whose return type is generic as well. The return type has the same placeholder T as the template in the definition of MyCustomList<T>.

* For MyCustomList<String> list, list.get returns a String
* For MyCustomList<Integer> list2, list2.get returns an Integer

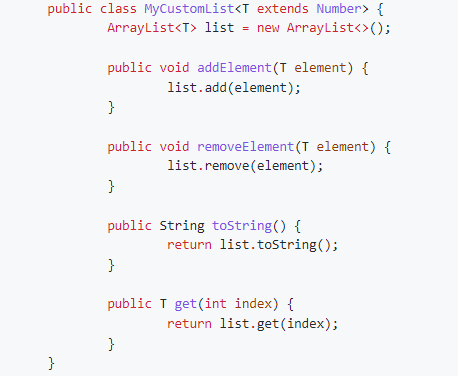
#### **Implementing Type Restrictions on Generics**

We saw above that we could use MyCustomList<T> to be instantiated into data structures for storing Strings as well as for Integers.

What if we wanted to to use MyCustomList<T> purely for storing numeric values?

##### **Generic Type Restrictions**

**MyCustomList.java**



**GenericsRunner.java**



**Console Output**

5

9

When we specify T extends Number as the type, we can use all the methods in the API of class Number are available for use.

Assignment 03

We can create generic methods as well. Let's look at a few examples:

##### **Generic Method**

**GenericsRunner.java**



**Console Output**

[A, B, C, A, B, C]

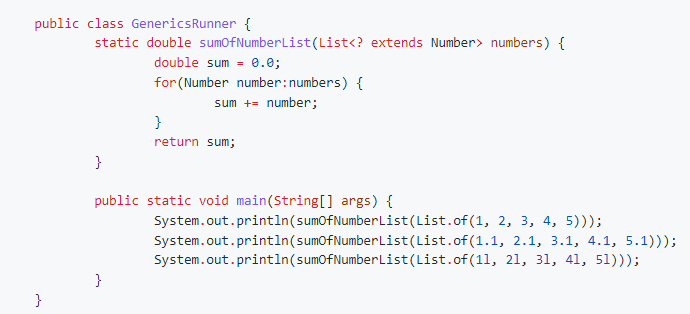
[1, 2, 3, 1, 2, 3]

Assignment 04

#### **Generics And Wild-Cards**

You can use wild card with generics too - ? extends Number

**GenericsRunner.java**



**Console Output**

15.0

15.5

15.0

The symbol ? in the definition of the method static double sumOfNumberList(List<? extends Number> numbers) is the ****wild-card**** symbol. It denotes the fact that in order to be a valid argument to sumOfNumberList, numbers can be a List of any elements, so long as all of them are of type sub-classed from Number.

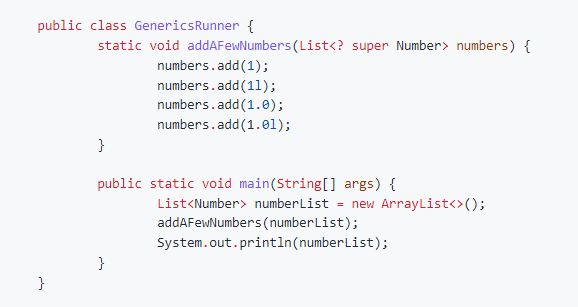
* This includes Integer, Long, Short, Byte, Float and Double.
* It also includes their primitive type counterparts, since they can be converted implicitly to their Wrapper class counterparts.
* Of course, all these elements of List numbers need to be of a homogeneous type.

#### **Restricted Heterogeneous Lists**

The generic wildcard we saw in the previous section is referred to as a ****Upper-Bounded Wild-Card****. It can be used to specify homogeneous types with a restriction. There is another category of wild-cards called ****Lower-Bounded Wild-Card****, which can be used with create ****Heterogeneous**** types of elements , within the restriction. Here is an example.

##### **More wild-cards**

**GenericsRunner.java**



**Console Output**

[1, 1, 1.0. 1.0]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*