**CSCI 2302**

**Generics Chapter**

**Creating Generic Objects Lab**

Contents

[Intro 1](#_Toc86242563)

[Notes 1](#_Toc86242564)

[Advantages 1](#_Toc86242565)

[Learning Goals 4](#_Toc86242566)

[Task 4](#_Toc86242567)

[Bonus Points 5](#_Toc86242568)

[Sample run 5](#_Toc86242569)

[Submit 6](#_Toc86242570)

Intro**:** Wouldn’t it great if we could implement/define a class/ADT/object that was not restricted by a data type, but instantiated with a specific data type? Yes, it would – and Generics allow just that.

Notes: Let’s recap about Java. Java is strongly data typed. This means that in order to have a variable it must declared with a data type and once that data type is declared, it cannot be changed. Generics allow a class to be used by several data types; this is just one of the advantages of generics. Generics have several advantages:

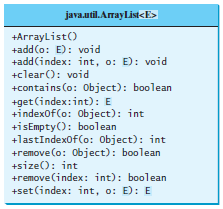
Advantages**:**

* Generics enable you to detect errors at compile time rather than at runtime.
* Generics enable you to reuse code independent of the data types, write a method/class/interface one time and have it applicable to any data types needed.
* Generics eliminate the need to do individual type casting.

Generics are defined with <>. The data type that is specified within the <> is the formal generic type. What the formal generic type is replaced with is the actual concrete type. This replacing is called generic instantiation.

By convention, a single capital letter usually the letter E or the letter T.

Example:



For example:

ArrayList<String> list = new ArrayList<>();

Creates a list for strings.

This specifies that the ArrayList class list can only contain the String data type. If you try to add any other data types, it will cause an error.

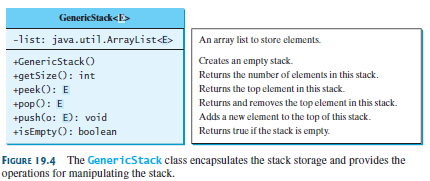
Declaring this with another data type:

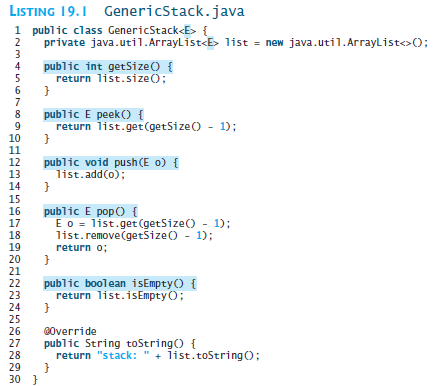
ArrayList<Integer> intList = new ArrayList<>();

This specifies that intList can only contain Integer data types.

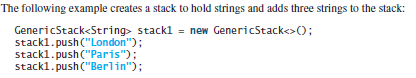
You can use the ArrayList class with different data types. And when you use the methods you do not have to use casting!

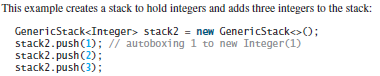
Another example:

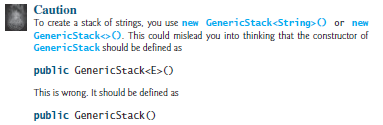




Examples in use:



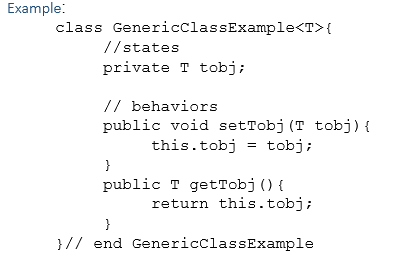




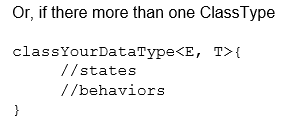
Learning Goals**:** To integrate the Generics concept into how to implement/define a class/object.

Task**:** Complete the steps outlined below:

1. Implement/Define a program, mysfasuserName\_GenericObjects.java, to do the following:
2. Create a Generic object/ADT, GO\_One, with a constructor that accepts one object and an accessor/setter or print method.



1. Create another Generic object/ADT, GO\_Two, with a constructor that accepts two objects and an accessor/setter or print method.



1. In the mysfasuserName\_GenericObjects.java.java instantiate then invoke the accessor and/or print method to display the object to the screen:
   1. A GO\_One with Integer data type
   2. A GO\_One with Double data type
   3. A GO\_One with String data type
   4. A GO\_Two with Integer and String data types
   5. A GO\_Two with Double and String data types
   6. A GO\_Two with String and String data types

Bonus Points: add a static variable to show how many objects of each type were created (hints: static & constructor – remember how you have to access the static variables!).

Sample run**:**

Generic Objects with 1 data type

2302

2302.000

CSCI 2302 Learning Generics!

How many GO\_One objects created? 3 objects

Generic Objects with 2 data types

2302 Generics

2302.000 Generics

Learning Generics

How many GO\_Two objects created? 3 objects

Submit: the GO\_One.java, GO\_Two.java, and mysfasuserName\_GenericObjects.java file(s) in the Dropbox in Brightspace by D2L.