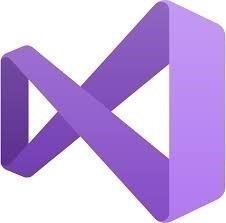
**CS 131C# - Beginner**

**HOP10 – Generic Types and Collections**

03/01/2020 Developed by Kim Nguyen

Center for Information Assurance (CIAE) @City University of Seattle (CityU)



**Before You Start**

* Version numbers may not match with the most current version at the time of writing. If given the option to choose between stable release (long-term support) or most recent, please choose the stable release rather than beta-testing version.
* This tutorial targets Windows users and MacOS users.
* There might be subtle discrepancies along the steps. Please use your best judgement while going through this cookbook style tutorial to complete each step.
* For your working directory, use your course number. This tutorial may use a different course number as an example.
* The directory path shown in screenshots may be different from yours.
* If you are not sure what to do or confused with any steps:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Understand Generic types.
* Write programs using Generic Types.
* Familiarized with Collections.
* Practiced and programed using Array List and Hash Table

**Resources**

* C# Tutorials | W3Schools.com- <https://www.w3schools.com/cs/default.asp>
* C# Tutorials | Geeks for Geeks - [https://www.geeksforgeeks.org/](https://www.geeksforgeeks.org/c-sharp-generics-introduction/)
* C# Tutorials | tutorials.com- [https://www.tutorialspoint.com/csharp/](https://www.tutorialspoint.com/csharp/csharp_strings.htm)

**Generics**

Previously, we have learned and practiced on different data types, classes and objects. We know that if a class has specified data type, other data type cannot be accepted. However, OOP has a work around, providing flexibility and simplicity for programs. Beside all the data types you have learned in the past weeks, this week, we will learn about Generic Data Type.

Generics were added to version 2.0 of the C# language. Generics allow you to define the specification of the data type of programming elements in a class or a method, until it is actually used in the program. In other words, generics allow you to write a class or method that can work with any data type.

You write the specifications for the class or the method, with substitute parameters for data types. When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.

Let's practice!

**Create a project**

1. Open Visual Studio.
2. File > New > Project
3. Select Console App (.NET Core), click Next
4. Type “GenericType” in the Project name and save it in the following locations:

**If you are an online student:**

Save it here > CS131-Spring-2020\**ON**\FirstnameLastname/Module10/Generic-Collection

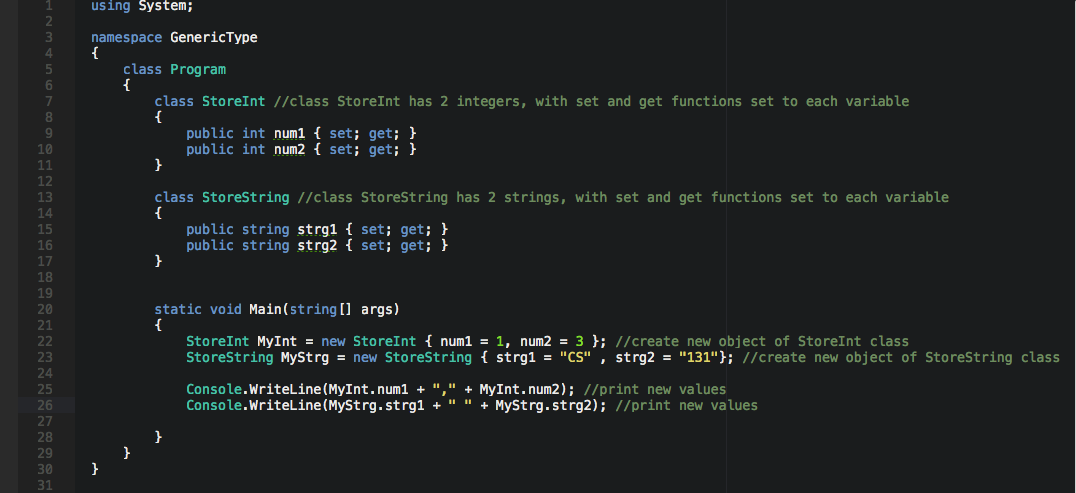
**If you are an onsite student:**

Save it here > CS131-Spring-2020\**IN**\FirstnameLastname/Module10/ Generic-Collection

You don't have to type the following code. However, do have a look. In the screenshot, there are two classes StoreInt and StoreString that takes 2 different types of int and string, respectively.

This means that for StoreInt if we pass any data type that's not integer, it will not be accepted.

Similarly, for StoreString if we pass any data type that's not string, it will not be accepted.



Result: 

Although StoreInt and StoreString do exact same things, I still have to re-create the classes, because they use different data types.

What if next time, instead of string and integer, I want to use float, or boolean or any other data type? We will keep having to create more similar classes, only to change the data type. This can quickly create a long, hard to read and cumbersome program. So, let’s use Generic Type to fix this.

1) Type the following code in your Program.cs:



2) Run your program:



By using Generic Type, we now able to use the same class for different data types, cutting the program much shorter.

3) Edit your Program.cs to match the following screenshot:



We have added another parameter, indicating that StoreAnything can take in two parameters of any types (Please note that T and U are just names, they can be changed to anything)

Inside Main(), we passed in two parameters that of two different types and it works fine, because we have declared above that StoreAnything takes in two parameters of any types.

4) Run your program again:



**CHALLENGE:**

Edit StroreAnything to take in 3 parameters of Generic type. Keep the same values for the first two parameters, for the third one, set to char type with the value of '!'

Print result on the screen. Expected:

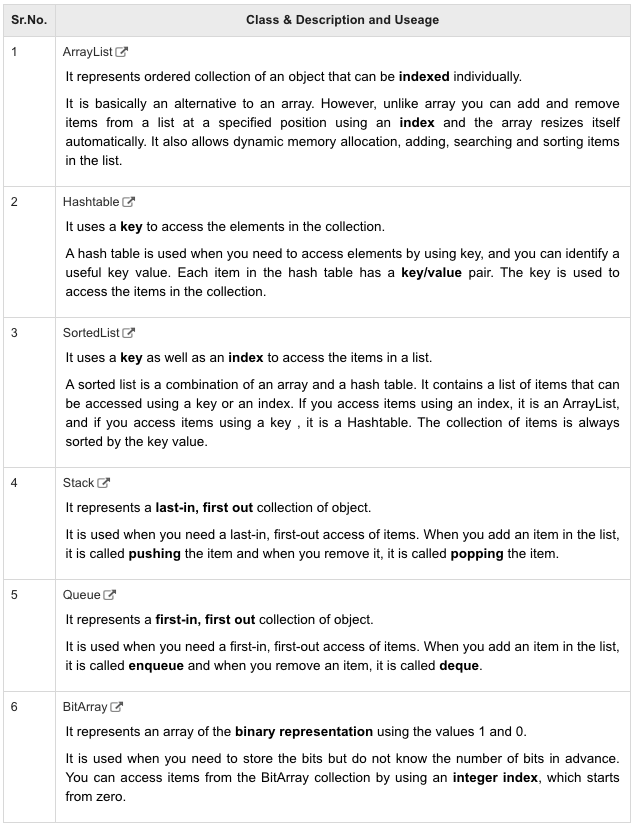


**Collections:**

Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types in C#.

The following are the various commonly used classes of the System.Collection namespace. Click the following links to check their detail.



Let's practice the first 2 collection types

**Create a project**

1. Open Visual Studio.
2. File > New > Project
3. Select Console App (.NET Core), click Next
4. Type “Arraylist” in the Project name and save it in the following locations:

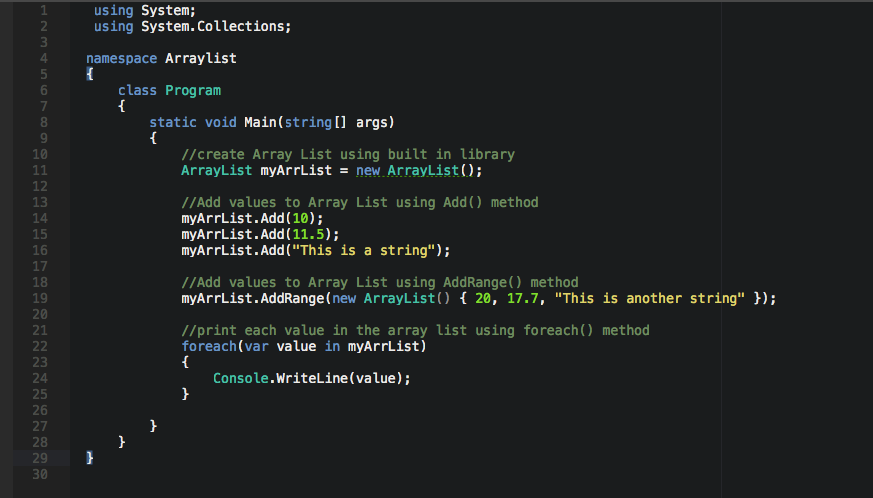
**If you are an online student:**

Save it here > CS131-Spring-2020\**ON**\FirstnameLastname/Module10/Generic-Collection/Collections

**If you are an onsite student:**

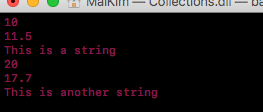
Save it here > CS131-Spring-2020\**IN**\FirstnameLastname/Module10/ Generic-Collection/Collections

1) Type the following code in your Program.cs:

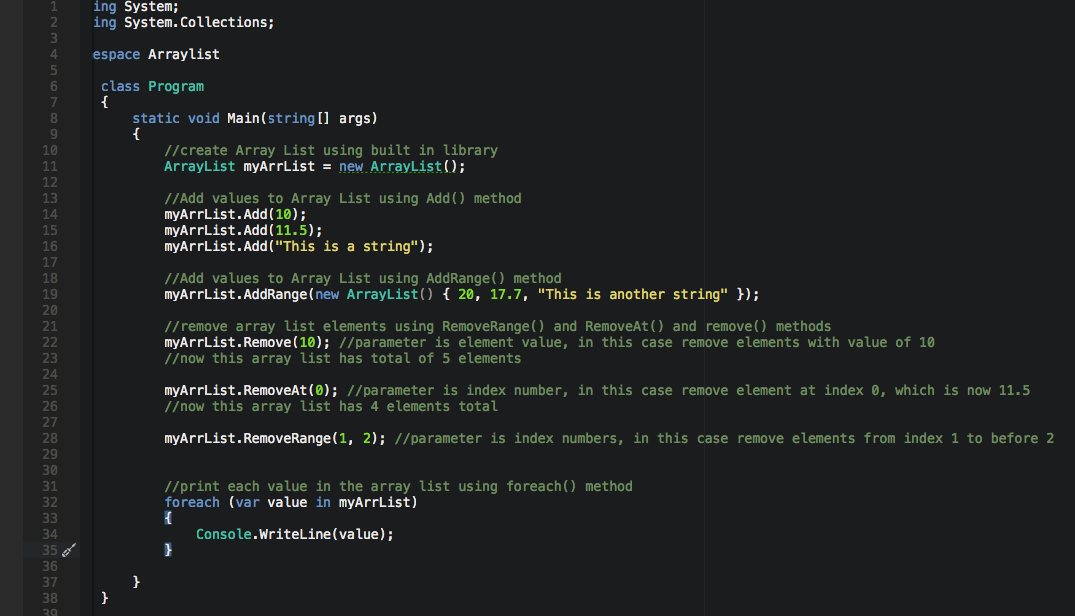


As you can see, more convenient than an array, you can add and remove items from a list at a specified position using an index and **the array resizes itself automatically**.

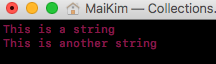
2) Run your program:



3) Update your code to match following screenshot:



4) Run your program:



**Create a project**

1)Open Visual Studio.

2)File > New > Project

3)Select Console App (.NET Core), click Next

4)Type “HashTable” in the Project name and save it in the following locations:

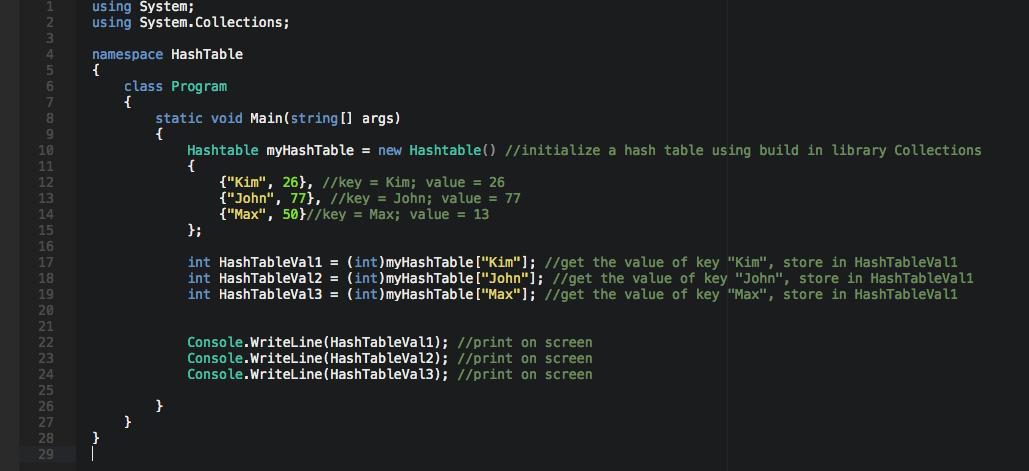
**If you are an online student:**

Save it here > CS131-Spring-2020\**ON**\FirstnameLastname/Module10/Generic-Collection/Collections

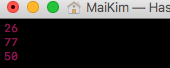
**If you are an onsite student:**

Save it here > CS131-Spring-2020\**IN**\FirstnameLastname/Module10/ Generic-Collection/Collections

1) Type the following code in Program.cs:



2) Run your program:



By using Hash Table, we are able to cut the run time shorter, imagine if we have 1000 elements in a list, to look for age of 'Kim' or 'John' or 'Max' we would have to look through each element. This is slow and not efficient. Instead, we can use Hash Table, to look for the element using its key, knowing all the names is stored under key "name", every time we want to find names, we can just hit that key, instead of going though all kinds of elements.

No Challenge for this section! 😊

**Push your work to GitHub**

**Commit changes**

1. Click on the **Home** button > **Changes**
2. Type commit message
3. Select **Commit All and Push**

**Create a pull request**

1. Go to your fork page on GitHub website
2. Near the top left side, change the active branch to your new branch
3. Click on the "New Pull Request" button next to the branch name.