Generic Collections

Ethan Baker

Collections

- A way to create and manage a group of related objects
- Store data
    - Add items
    - Insert items
- Manage and Manipulate data
    - Delete items
    - Sort items
    - Determine number of items in collection
    - Replace items
    - Search for specific items
- Can grow and shrink dynamically

What Are Generic Collections?

- Generic collections are used to store a specific data type
- When you retrieve an element no need to determine or convert type
- No risk of type mismatching errors
- Minimizes exceptions since errors are given at compile-time
- Faster than non-generic collections

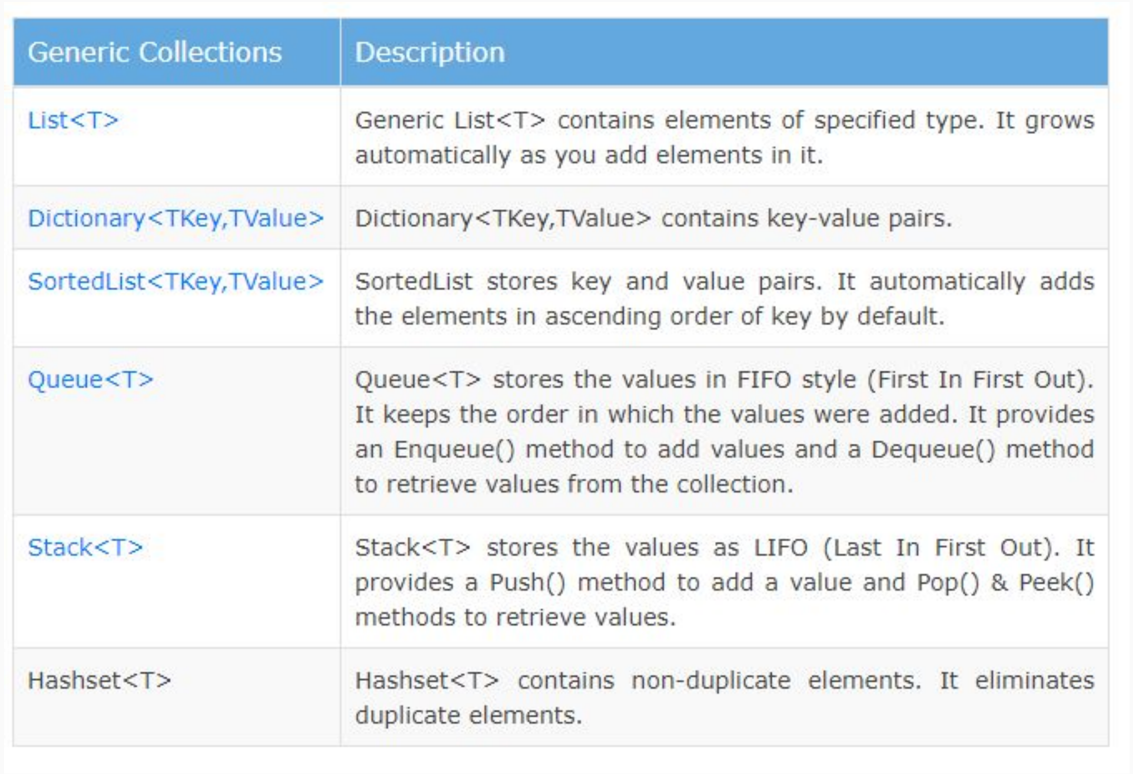
Non-generic vs Generic Collections

Non-generic	Generic
Each element can represent a different type	All elements are one specific type
System.Collections namespace	System.Collections.Generic namespace
Collection size not fixed	Collection size not fixed
Items in collection can be added or removed at runtime	Items in collection can be added or removed at runtime

Non-generic vs Generic Collections cont.

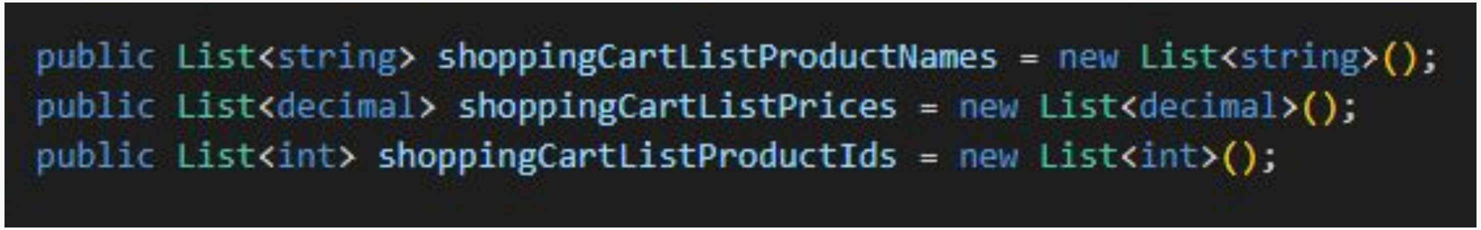
Non-generic	Generic
ArrayList	List<T>
Hashtable	Dictionary<TKey, TValue>
SortedList	SortedList<TKey, TValue>
Queue	Queue<T>
Stack	Stack<T>

Examples of Generic Collections



Generic Collections	Description
List<T>	Generic List<T> contains elements of specified type. It grows automatically as you add elements in it.
Dictionary<TKey,TValue>	Dictionary<TKey,TValue> contains key-value pairs.
SortedList<TKey,TValue>	SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default.
Queue<T>	Queue<T> stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection.
Stack<T>	Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values.
Hashset<T>	Hashset<T> contains non-duplicate elements. It eliminates duplicate elements.

Example: List<T>



```
public List<string> shoppingCartListProductNames = new List<string>();
public List<decimal> shoppingCartListPrices = new List<decimal>();
public List<int> shoppingCartListProductIds = new List<int>();
```

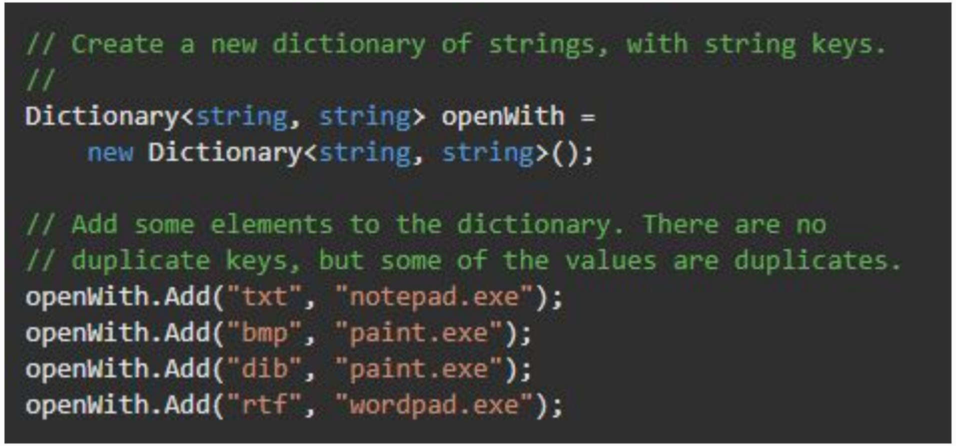
Generic lists to keep track of different types of data needed for a shopping cart.

- Properties
  - Count - get the number of elements
- Methods
  - Add(T) - add to end of list
  - Remove(T) - remove first occurrence
  - Clear() - remove all elements
  - Sort() - sorts elements

Example: Dictionary<TKey, TValue>

- Properties
  - Count - get the number of key/value pairs
  - Keys - get a collection of the keys
  - Values - get collection of the values

- Methods
  - Add(TKey, TValue) - add to dictionary
  - Remove(Tkey) - remove value with the specified key
  - Clear() - remove all elements
  - ContainsValue(TValue) - determines if the dictionary contains a specified value



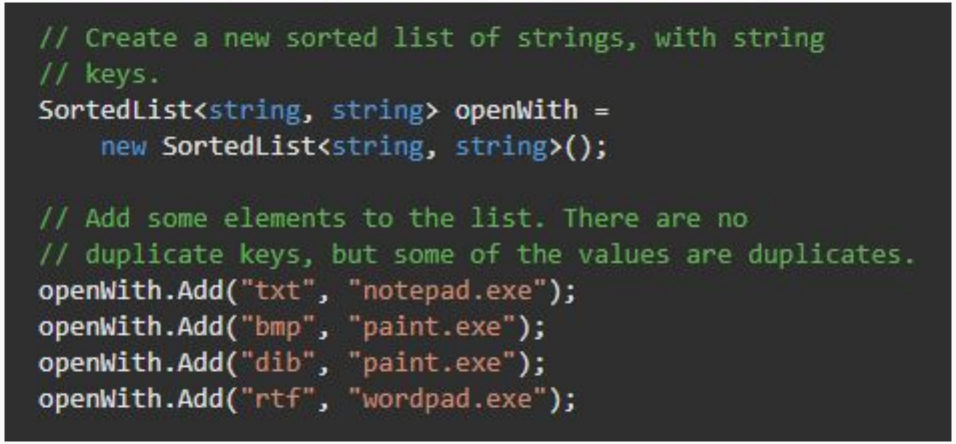
```
// Create a new dictionary of strings, with string keys.
//
Dictionary<string, string> openWith =
    new Dictionary<string, string>();

// Add some elements to the dictionary. There are no
// duplicate keys, but some of the values are duplicates.
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");
```

Example: SortedList<TKey, TValue>

- Properties
  - Count - get the number of key/value pairs
  - Keys - get a collection of the keys
  - Values - get collection of the values

- Methods
  - Add(TKey, TValue) - add to dictionary
  - Remove(Tkey) - remove value with the specified key
  - Clear() - remove all elements
  - ContainsValue(TValue) - determines if the dictionary contains a specified value



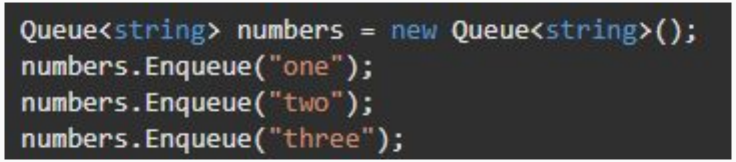
```
// Create a new sorted list of strings, with string
// keys.
SortedList<string, string> openWith =
    new SortedList<string, string>();

// Add some elements to the list. There are no
// duplicate keys, but some of the values are duplicates.
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");
```

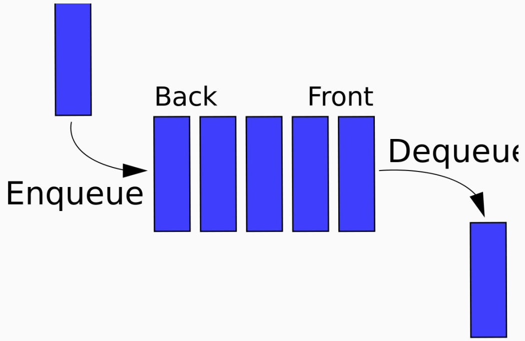
Example: Queue<T>

First-in, first-out collection of objects

- Properties
  - Count - get the number of elements
- Methods
  - Enque(T) - add to end of Queue
  - Deque() - remove and returns object at front of Queue
  - Peek() - returns object at the front of the Queue without removing it
  - Clear() - remove all elements



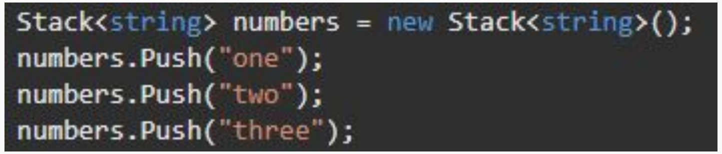
```
Queue<string> numbers = new Queue<string>();
numbers.Enqueue("one");
numbers.Enqueue("two");
numbers.Enqueue("three");
```



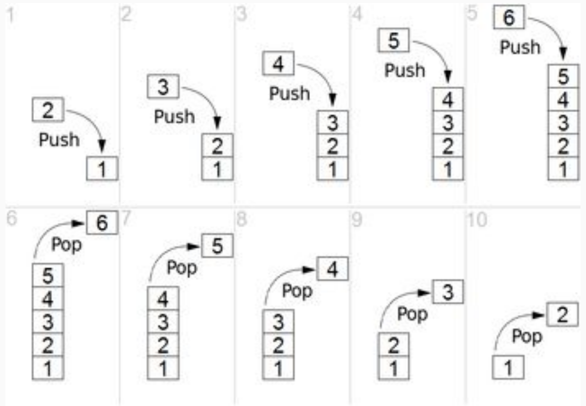
Example: Stack<T>

Last-in, first-out collection of objects

- Properties
  - Count - get the number of elements
- Methods
  - Push(T) - add object to top of Stack
  - Pop() - remove and returns object from the top of the Stack
  - Peek() - returns object at the top of the Stack without removing it
  - Clear() - remove all elements



```
Stack<string> numbers = new Stack<string>();
numbers.Push("one");
numbers.Push("two");
numbers.Push("three");
```



Resources

https://www.c-sharpcorner.com/UploadFile/736bf5/collection-in-C-Sharp

https://www.tutorialsteacher.com/csharp/csharp-collection

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections

https://en.wikipedia.org/wiki/Queue_(abstract_data_type)

https://en.wikipedia.org/wiki/Stack_(abstract_data_type)