**C# Vanket**

**What is Var**

**var**. An **implicitly** typed local variable is strongly typed just as if you had declared the type yourself, but the compiler determines the type. The following two declarations of i are functionally equivalent:

var i = 10; // implicitly typed

int i = 10; //explicitly typed

It means that the type of the local being declared will be inferred by the compiler

**What is reflection?**

**Reflection** is the ability of inspecting an assemblies' metadata at runtime.  It is used to find all types in an assembly and/or dynamically invoke methods in an assembly.

 With Reflection, we can dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties.There are several uses of reflection.

**1.** When you drag and drop a button on a win forms or an asp.net application. The properties window uses reflection to show all the properties of the Button class. So,reflection is extensivley used by IDE or a UI designers.  
  
**2.** Late binding can be achieved by using reflection. You can use reflection to dynamically create an instance of a type, about which we don't have any information at compile time. So, reflection enables you to use code that is not available at compile time.

**Boxing?**

Boxing is the process of converting a [value type](https://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** (Reference Type)

**Un-Boxing?**

Un-Boxing is the process of converting type **object**  back to a value type.

**What is Generics?**

**Classes, structures, interfaces, and methods that have placeholders(Type Parameters) for one or more of the types that they store or use.**

**Benefits...**

**Type Safety.**

**Performance (No un-boxing or boxing).**

**Reuse.**

All generic collection classes including List are strongly typed. This means   
  if you have created a List of type Customer, only objects of type Customer   
   can be added to the list.

introduced in C# 2.0. Generics allow us to **design classes and methods decoupled from the data types**. Generic classes are extensively used by collection classes available in System.Collections.Generic namespace

The problem with *ArrayList* and all the other .NET v1.0 collections is that they operate on type ***object*** which requires Boxing and Un-Boxing object type items in the collection

**Generic Collections Fixed that...**

A generic collection is strongly typed (**type safe**), meaning that you can only put one **type** of object into it which means better performance because there is no overhead from boxing and un-boxing

# System.Collections.Generic Namespace

The System.Collections.Generic namespace contains interfaces and classes that define generic collections, which allow users to create strongly typed collections that provide better type safety and performance than non-generic strongly typed collections.

[**Dictionary<TKey, TValue>**](https://msdn.microsoft.com/en-us/library/xfhwa508(v=vs.110).aspx)

**1.** A dictionary is a collection of (key, value) pairs.  
**2.** Dictionary class is present in System.Collections.Generic namespace.  
**3.** When creating a dictionary, we need to specify the type for key and value.  
**4.** Dictionary provides fast lookups for values using keys.  
**5.** Keys in the dictionary must be unique.**Dictionary.**

Fast lookups are critical. Dictionaries provide fast lookups, based on keys,  
to get values. With them, we use keys and values of any type, including int and string.

[**SortedDictionary<TKey, TValue>**](https://msdn.microsoft.com/en-us/library/f7fta44c(v=vs.110).aspx)

**SortedDictionary** keeps its keys always sorted. It allows you to avoid sorting the keys on your own. Its lookup performance is slower than Dictionary. It has advantages if you require a sorted lookup table in memory.

[**LinkedList<T>**](https://msdn.microsoft.com/en-us/library/he2s3bh7(v=vs.110).aspx)

**LinkedList** allows fast inserts and removes. It implements a linked list. Each object is separately allocated. Certain operations do not require the whole collection to be copied. In many common cases LinkedList hinders performance.

[**List<T>**](https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx)

**List.** An array does not dynamically resize. A List does. With it, we do not need to manage the size on our own. This type is ideal for linear collections not accessed by keys.

A List class can be used to create a collection of any type. For example, we can create a list of Integers, Strings and even complex types. The objects stored in the list can be accessed by index. Unlike arrays, lists can grow in size automatically. This class also provides methods to search, sort, and manipulate lists.

**Find()** method of the List class loops thru each object in the list until a match is found. So, if you want to lookup a value using a key, dictionary is better for performance over list. So, use dictionary when you know the collection will be primarily used for lookups.

**1. Contains() function -** Use this function to check if an item exists in the list. This method returns true if the items exists, else false.  
  
**2. Exists() function -**Use this function, to check if an item exists in the list based on a condition. This method returns true if the items exists, else false.  
  
**3. Find() function -**This method searches for an element that matches the conditions defined by the specified lambda expression and returns the first matching item from the list.  
  
**4. FindLast() function -**This method searches for an element that matches the conditions defined by the specified lambda expression and returns the Last matching item from the list.  
  
**5. FindAll() function -**This method returns all the items from the list that match the conditions specified by the lambda expression.   
  
**6. FindIndex() function -**This method returns the index of the first item, that matches the condition specified by the lambda expression. There are 2 other overloads of this method which allows us to specify the range of elements to search, with in the list.  
  
**7. FindLastIndex() function -**This method returns the index of the last item, that matches the condition specified by the lambda expression. There are 2 other overloads of this method which allows us to specify the range of elements to search, with in the list.  
  
**8. Convert an array to a List -**Use ToList() method  
  
**9. Convert a list to an array -**Use ToArray() method  
  
**10. Convert a List to a Dictionary -**Use ToDictionary() method

[**SortedList<TKey, TValue>**](https://msdn.microsoft.com/en-us/library/ms132319(v=vs.110).aspx)

**SortedList** stores elements in an ordered way. It can be quickly searched with binary search. It has worse lookup performance than Dictionary collections.

[**Queue<T>**](https://msdn.microsoft.com/en-us/library/7977ey2c(v=vs.110).aspx)

**Queue** is a FIFO collection. It processes elements in a first-in, first-out order. To restate, it handles the elements that it received longest ago first.

To add items to the end of the queue, use **Enqueue**() method.  
To remove an item that is present at the beginning of the queue, use **Dequeue**() method.  
A foreach loop iterates thru the items in the queue, but will not remove them from the queue.  
To check if an item, exists in the queue, use **Contains**() method.  
  
**What is the difference between Dequeue() and Peek() methods?**  
**Dequeue**() method removes and returns the item at the beginning of the queue, where as**Peek**() returns the item at the beginning of the queue, without removing it.

[**Stack<T>**](https://msdn.microsoft.com/en-us/library/3278tedw(v=vs.110).aspx)

**Stack** is a LIFO collection. It provides a powerful and simple last-in-first-out data structure.

To insert an item at the top of the stack, use **Push**() method.  
  
To remove and return the item that is present at the top of the stack, use **Pop**() method.

To check if an item exists in the stack, use **Contains**() method.

**What is the difference between Pop() and Peek() methods?**  
Pop() method removes and returns the item at the top of the stack, where as Peek() returns the item at the top of the stack, without removing it.

**Generic Interface**

 Generics give us flexibility and reuse in our classes, methods, and interfaces.

### Why should you override ToString() method?

**Difference between Convert.ToString() and ToString()?**

 Convert.ToString() handles null, while ToString() doesn't, and throws a NULL Reference exception.

**Why should you override Equals() method?**

**What is the difference between == operator and the .Equals() Virtual Method?**

 By default **"=="** operator checks for **reference equality** and **.Equals()** method checks for **value equal**

**What is the difference between Reference equality and value equality?**

Reference equality means that two variables living on the stack are pointing/referring to the same object on the Heap.

CustomerService.Customer c1 = new CustomerService.Customer();

c1.Id = 1;

c1.Name = "Eric";

c1.City = "Denver";

c1.BirthDay = new DateTime(1962, 10, 10);

CustomerService.Customer c2 = c1;

**C1 does (==) equal C2.**

If the two variables living on the Stack are pointing at two different instances of the same object living on the Heap the reference equality is false even if the two instances contain identical data.

CustomerService.Customer c1 = new CustomerService.Customer();

c1.Id = 1;

c1.Name = "Eric";

c1.City = "Denver";

c1.BirthDay = new DateTime(1962, 10, 10);

CustomerService.Customer c2 = new CustomerService.Customer();

c1.Id = 1;

c1.Name = "Eric";

c1.City = "Denver";

c1.BirthDay = new DateTime(1962, 10, 10);

**C1 does not (!=) equal C2 because C1 and C2 are pointing/referring to two different object instances with different memory locations (Identities).**

**So...**

If two objects have reference equality, then they also have value equality, but value equality does not guarantee reference equality.

**What is the difference between object reference variables and objects?**

* Reference variable is a pointer/reference to a object and is sitting on the STACK.
* Object is a bunch of attributes, methods etc (**state and behavior**) sitting on the HEAP
* A reference Stack variable points to the object location on the Heap.

**What is an object reference variable?**

Object reference variables, stay on the stack and are pointers to actual objects on the heap.

**Why should you override Equals() method on our classes.**

**When comparing two instances of our class the == operator will compare reference equality. (Good, No problem)**

**However the Equals() method should compare value equality between are two object references but can't because the base implementation of the Equals() method knows nothing about what values properties in our class it should compare.**

**If we override the base Equals() method we can implement our own equality check based on the property values we choose.**

**What is an  enumeration (Enum)?**

A distinct type that consists of a set of named constants called the enumerator list.

# What are specialized collections?

# System.Collections.Specialized Namespace

Specialized collections are collections with highly specific purposes.

|  |  |  |
| --- | --- | --- |
|  | **Class** | **Description** |
| Public class | [CollectionsUtil](https://msdn.microsoft.com/en-us/library/system.collections.specialized.collectionsutil(v=vs.110).aspx) | Creates collections that ignore the case in strings. |
| Public class | [ListDictionary](https://msdn.microsoft.com/en-us/library/system.collections.specialized.listdictionary(v=vs.110).aspx) | Implements **IDictionary** using a singly linked list. Recommended for collections that typically include fewer than 10 items. |
| Public class | [NameObjectCollectionBase](https://msdn.microsoft.com/en-us/library/system.collections.specialized.nameobjectcollectionbase(v=vs.110).aspx) | Provides the **abstract** base class for a collection of associated [String](https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx) keys and [Object](https://msdn.microsoft.com/en-us/library/system.object(v=vs.110).aspx) values that can be accessed either with the key or with the index. |
| Public class | [NameObjectCollectionBase.KeysCollection](https://msdn.microsoft.com/en-us/library/system.collections.specialized.nameobjectcollectionbase.keyscollection(v=vs.110).aspx) | Represents a collection of the [String](https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx) keys of a collection. |
| Public class | [NameValueCollection](https://msdn.microsoft.com/en-us/library/system.collections.specialized.namevaluecollection(v=vs.110).aspx) | Represents a collection of associated [String](https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx) keys and [String](https://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx) values that can be accessed either with the key or with the index. |
| Public class Supported by Portable Class Library | [NotifyCollectionChangedEventArgs](https://msdn.microsoft.com/en-us/library/system.collections.specialized.notifycollectionchangedeventargs(v=vs.110).aspx) | Provides data for the [CollectionChanged](https://msdn.microsoft.com/en-us/library/system.collections.specialized.inotifycollectionchanged.collectionchanged(v=vs.110).aspx) event. |
| Public class | [OrderedDictionary](https://msdn.microsoft.com/en-us/library/system.collections.specialized.ordereddictionary(v=vs.110).aspx) | Represents a collection of key/value pairs that are accessible by the key or index. |
| Public class | [StringCollection](https://msdn.microsoft.com/en-us/library/system.collections.specialized.stringcollection(v=vs.110).aspx) | Represents a collection of strings. |
| Public class | [StringDictionary](https://msdn.microsoft.com/en-us/library/system.collections.specialized.stringdictionary(v=vs.110).aspx) | Implements a hash table with the key and the value strongly typed to be strings rather than objects. |
| Public class | [StringEnumerator](https://msdn.microsoft.com/en-us/library/system.collections.specialized.stringenumerator(v=vs.110).aspx) | Supports a simple iteration over a [StringCollection](https://msdn.microsoft.com/en-us/library/system.collections.specialized.stringcollection(v=vs.110).aspx). |