A generic collection is strongly type safe. Which type of data you want to put in generic type, this information you have to provide at compile time. It means you can only put one type of object into it. This eliminates type mismatches at runtime. Another benefit of type safety is that, it provides better performance than non-generic collection.  
  
**Let us take one example and understand what problem may occur in ArrayList.**

Example

ArrayList   myList = new ArrayList ();  
            myList.Add(1);  
            myList.Add(2);  
            myList.Add(3);  
            myList.Add("4");  
            foreach (Object obj in myList)  
            {  
                int number = (int)obj;  
            }

The above code will compile fine, but in foreach loop it will throw an exception because the 4 is a string and not an integer. An ArrayList has no compile-time information about what’s in it. We could have accidentally added a string type “4” to the list. That’s why we get the type mismatch exception.  
  
If you use generic List<T> in place of ArrayList, then you will get compile time error.  
The .NET Framework class library contains several generic collection classes in the System.Collections.Generic namespace. If you want to use these generic collections you must use System.Collections.Generic namespace in your program.  
  
**Let us take some example that how to add different types in list.**

* **Generic list for storing integer type**

List<int> intList = new List<int> ();  
intList.Add(250);

* **Generic list for storing character type**

List<char> charList = new List<char> ();  
charList.Add('B');

* **Generic list for storing string type**

List<string> stringList = new List<string> ();  
stringList.Add("Nishant");  
  
**Equivalent Generic Types**

|  |  |
| --- | --- |
| **Type** | **Generic Type** |
| ArrayList | List<> |
| Queue | Queue<> |
| Stack | Stack<> |
| Hashtable | Dictionary<> |
| SortedList | SortedList<> |
| ListDictionary | Dictionary<> |
| OrderedDictionary | Dictionary<> |
| SortedDictionary | SortedDictionary<> |
| NameValueCollection | Dictionary<> |
| DictionaryEntry | NameValuePair<> |
| StringCollection | List<String> |
| StringDictionary | Dictionary<String> |

**Let us take one example that uses generic list.**

using System;  
using System.Collections.Generic;  
  
namespace ConsoleApplication1  
{  
    public class Employee  
    {  
        public int EmpID { get; set; }  
        public string EmpName { get; set; }  
        public double EmpSalary { get; set; }       
     
        public Employee()  
        {  
            EmpID = 0;  
            EmpName = "N/A";  
            EmpSalary = 0.0;    
        }  
  
        public Employee(int ID, string Name ,double salary)  
        {  
            EmpID = ID;  
            EmpName = Name;  
            EmpSalary = salary;             
        }  
         
        public  List<Employee> GetEmployees()  
        {  
            List<Employee> empList = new List<Employee>();  
            empList.Add(new Employee(1, "Raj    ", 2000));  
            empList.Add(new Employee(2, "Sheetal", 1000));  
            empList.Add(new Employee(3, "Ramesh", 2500));  
            empList.Add(new Employee(4, "Shaheen", 3000));  
  
            return empList;  
        }  
    }  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Employee obj = new Employee();             
            Console.WriteLine("EmployeeID \t EmployeeName \t EmployeeSalary");  
            foreach(Employee emp in obj.GetEmployees())  
            {  
              Console.WriteLine(" {0} \t\t {1} \t\t {2}", emp.EmpID, emp.EmpName,    
              emp.EmpSalary );    
            }              
            Console.ReadLine();  
        }  
    }  
}

**Output:**  
  
EmployeeID       EmployeeName    EmployeeSalary  
    1                       Raj                             2000  
    2                       Sheetal                       1000  
    3                       Ramesh                      2500  
    4                       Shaheen                      3000

Generic SortedList Class

The generic List class is same as the ArrayList, but it is type-safe collection.  
  
C# has a generic and non-generic SortedList. Generic SortedList stores the data in key-value pair in ascending order of key by default.  
  
SortedList<int,string> listObj = new SortedList<int,string>();  
  
Generic SortedList implements IDictionary<TKey,TValue> & ICollection<KeyValuePair<TKey,TValue>>  interfaces. So you can access the item by using key and index both.

Example

using System;  
using System.Collections.Generic;  
  
namespace ConsoleApplication1  
{     
    class Program  
    {  
        static void Main(string[] args)  
        {  
            SortedList<int, string> myList = new SortedList<int, string>();  
            myList.Add(30, "Thirty");  
            myList.Add(40, "Fourty");  
            myList.Add(10, "Ten");  
            myList.Add(50, "Fifty");  
            myList.Add(20, "Twenty");  
            foreach (KeyValuePair<int, string> item in myList)  
            {  
                Console.WriteLine("Key: {0}, Value: {1}", item.Key, item.Value);  
            }                         
            Console.ReadLine();  
        }  
    }  
}

Execute the above program, you will get the sorted result according to the key.  
  
**Output:**  
Key: 10, Value: Ten  
Key: 20, Value: Twenty  
Key: 30, Value: Thirty  
Key: 40, Value: Fourty  
Key: 50, Value: Fifty

Generic Dictionary Class

The generic Dictionary class is similar to the Hashtable, ListDictionary, and HybridDictionary classes. It stores the data in key-value format.  
  
Dictionary<TKey, TValue> is a generic collection where TKey denotes the type of key and TValue is the type of TValue.  
  
**Initialization of Dictionary**  
  
Dictionary<int, string> dictObj = new Dictionary<int, string>();  
  
Here types of key and value are integer and string respectively. You can specify these types according to your needs.

Example

using System;  
using System.Collections.Generic;  
  
namespace ConsoleApplication1  
{     
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Dictionary<int, string> dicObj = new Dictionary<int, string>();  
            dicObj.Add(1, "One");  
            dicObj.Add(2, "Two");  
            dicObj.Add(3, "Three");  
            dicObj.Add(4,"Four");  
  
            foreach (KeyValuePair<int, string> item in dicObj)  
            {  
                Console.WriteLine("Key: {0}, Value: {1}", item.Key, item.Value);  
            }             
            Console.ReadLine();  
        }  
    }  
}

You can also initialize dictionary collection using initializer syntax with keys and values as shown below.

Dictionary<int, string> dicObj = new Dictionary<int, string>()  
{  
        {1,"One"},  
        {2,"Two"},  
        {3,"Three"},  
        {4,"Four"}  
};

Access Individual Element from the dictionary  
  
string value = dicObj[1];  
Console.WriteLine(value);  
  
Please note that if key is not present, then run time exception KeyNotFoundException will occur.  
  
**Important Properties and methods of Generic Dictionary**

|  |  |
| --- | --- |
| **Property** | **Description** |
| Count | Gets the total number of items exists in the Dictionary. |
| IsReadOnly | Returns a boolean indicating whether the Dictionary<TKey,TValue> is read-only. |
| Keys | Returns collection of keys of Dictionary<TKey,TValue>. |
| Values | Returns collection of values in Dictionary<TKey,TValue>. |

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void Add(TKey key, TValue value) | Add key-value pairs in Dictionary<TKey, TValue> collection. |
| void Remove(TKey) | Removes the element with the specified key. |
| bool ContainsKey(TKey key) | It checks whether the value exists or not. |
| bool ContainsValue(TValue value) | It checks whether the value exists or not. |
| void Clear() | Removes all the elements from generic Dictionary. |