

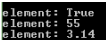
Generics

Categories of Collection Classes

- Collection classes can be separated into two categories, non-generic and generic
- Generic classes allow the programmer to specify the type of object in the collection as a parameter.
- Generic collections are homogeneous and "type safe"
- Non-generic collections can store heterogeneous objects

Example of Non-Generic Collection

```
ArrayList al = new ArrayList();  
al.Add(true);  
al.Add(55);  
al.Add(3.14);  
foreach(Object o in al)  
{  
    Console.WriteLine("element: {0}", o);  
}
```

produces: 

Non-Generic Collections

- See System.Collections Namespace
- Commonly used classes
 - ArrayList
 - Queue
 - SortedList
 - Stack

Issues with Non-Generic Collections

- Performance
 - boxing: store value type within a reference variable (to a System.Object variable)
 - unboxing: restore value type from a reference variable
 - value types are generally found on the stack and reference types are stored in the heap
 - Non-Generic collections store data as System.Object types

Issues with Non-Generic Collections

- Type Safety
 - You must unbox into an appropriate data type
 - Generates an exception if this isn't the case
 - Lack of strong typing can lead to bugs.

Generics

- Type of object in collection specified with "type parameter", <T>
- These are listed in the System.Collections.Generic Namespace
- Commonly used classes
 - List<T>
 - Queue<T>
 - Stack<T>
 - Dictionary<TKey, TValue>
 - LinkedList<T>

Generics

- Example

```
// list of ints
// int is substituted for T
// note call to constructor
List<int> l = new List<int>();
```

Generics vs Non-Generics

- Notice that in some cases (Queue, Stack), the same class name has both generic and non-generic versions
- Use the generic version for this class. Only in rare cases would you need the type flexibility that non-generics give you.
