Generic Lists

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR

@deborahkurata | blogs.msmvps.com/deborahk/

Generic List

A strongly typed list of elements that is accessed using a positional index number

Array vs. Generic List

Array	Generic List
Strongly typed	Strongly typed
Fixed length	Expandable
No ability to add or remove elements	Can add, insert, or remove elements
Multi-dimensional	One-dimensional

Generic List

List<T>

List<int>

List<decimal>

List<string>

List<Product>

Overview

- Declaring and Populating a Generic List
- Using Collection Initializers
- Initializing a List of Objects
- Retrieving an Element from a Generic List
- Iterating Through a Generic List
- Types of C# Lists
- FAQ

Declaring a Generic List

```
List<string> colorOptions;
```

- List of what?
- List<T>
  - Where T is the type of elements the list contains

"Red"
"Espresso"
"White"
"Navy"

Initializing a Generic List

```
List<string> colorOptions;
colorOptions = new List<string>();
```

- Reference type
- new keyword

Declaring and Initializing a List

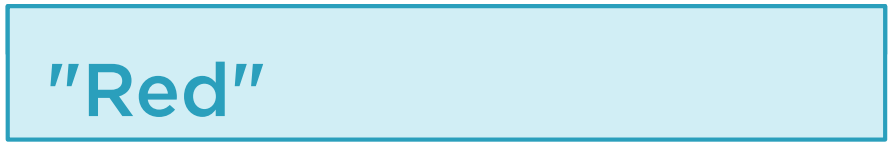
```
List<string> colorOptions;
colorOptions = new List<string>();
```

```
List<string> colorOptions = new List<string>();
```

```
var colorOptions = new List<string>();
```

Populating a List (Add)

```
colorOptions.Add("Red");
```



Populating a List (Add)

```
colorOptions.Add("Red");
colorOptions.Add("Espresso");
colorOptions.Add("White");
colorOptions.Add("Navy");
```

"Red"
"Espresso"
"White"
"Navy"

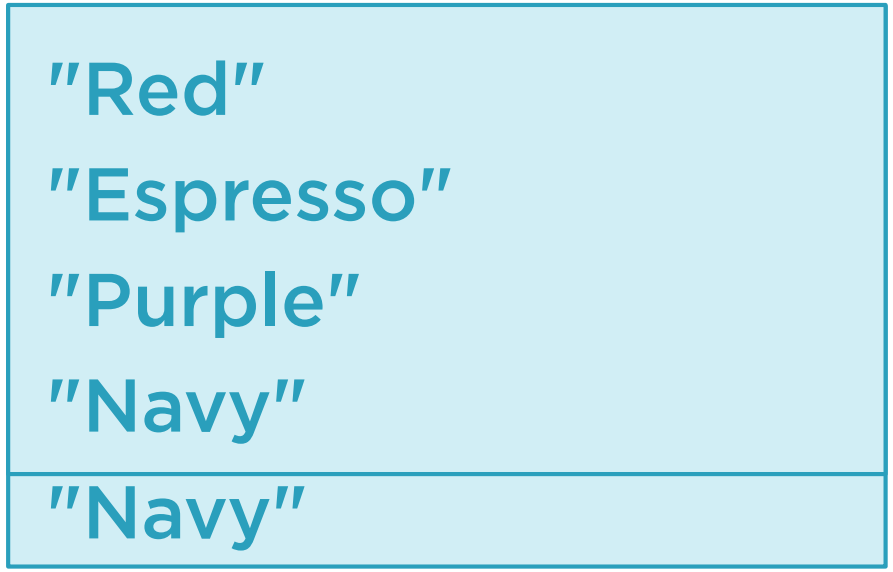
Populating a List (Insert)

```
colorOptions.Insert(2, "Purple");
```

"Red"
"Espresso"
"Purple"
"White"
"Navy"

Removing an Element

```
colorOptions.Remove("White");
```



Generic List Best Practices

Do:

Use generic lists to manage collections

Use Add over Insert where possible

Use a plural variable name for the list

Avoid:

Removing elements where possible

Declaring and Populating a List

```
var colorOptions = new List<string>();
```

```
colorOptions.Add("Red");
colorOptions.Add("Espresso");
colorOptions.Add("White");
colorOptions.Add("Navy");
```

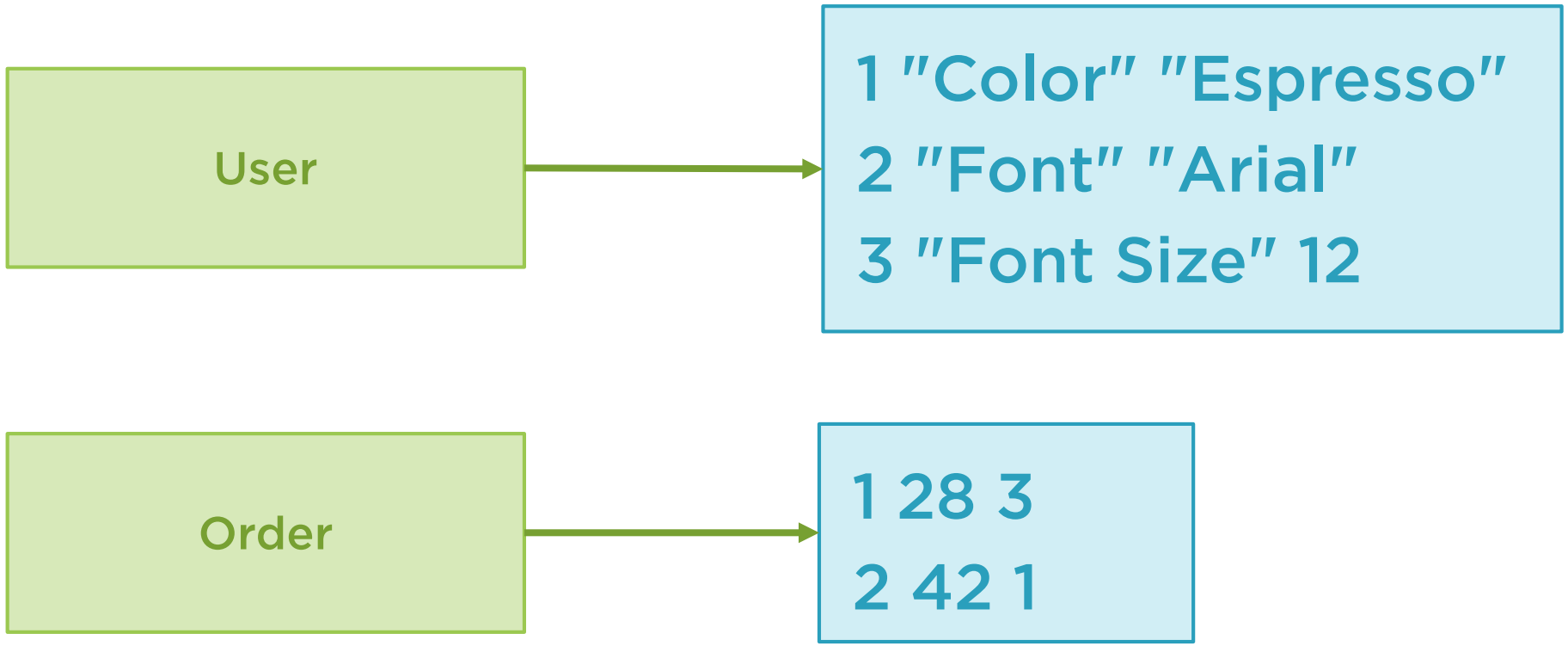
Collection Initializers

```
var colorOptions = new List<string>();
```

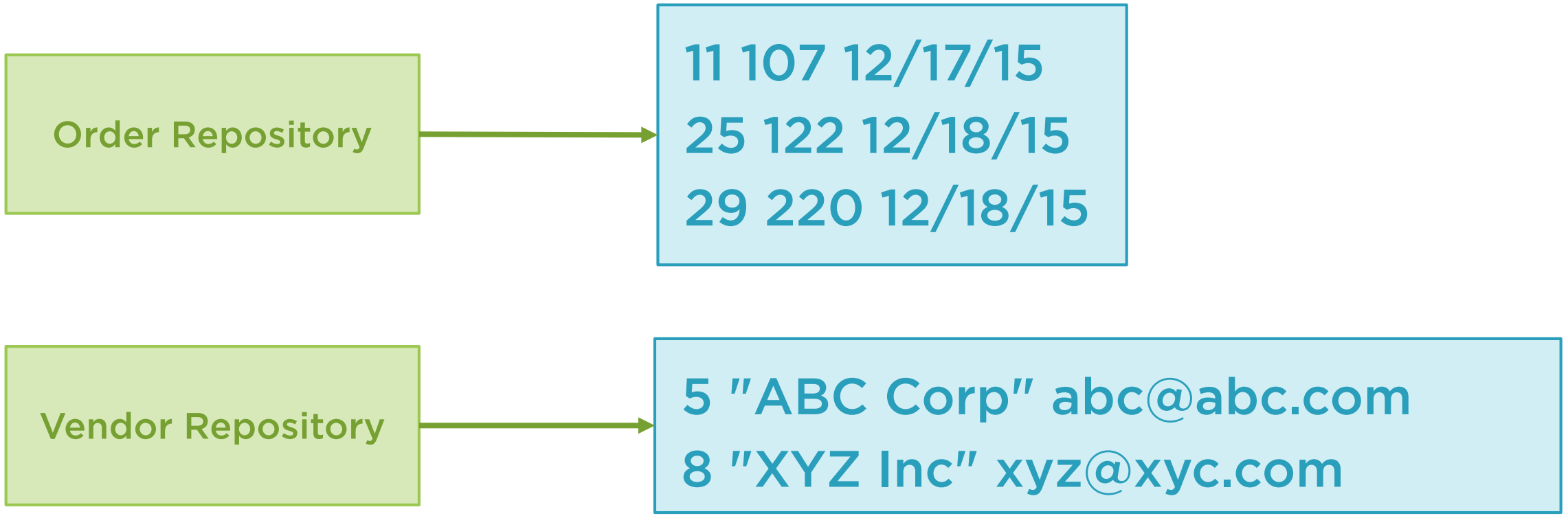
```
colorOptions.Add("Red");
colorOptions.Add("Espresso");
colorOptions.Add("White");
colorOptions.Add("Navy");
```

```
var colorOptions = new List<string>() {"Red","Espresso","White","Navy"};
```

Initializing a List of Objects



Initializing a List of Objects (cont)



Declaring, Initializing, and Populating a List

```
var vendors = new List<Vendor>();
```

```
var vendor = new Vendor() {VendorId=5,CompanyName="ABC Corp",Email="abc@abc.com"};
vendors.Add(vendor);

vendor = new Vendor() {VendorId = 8,CompanyName = "XYZ Inc",Email = "xyz@xyz.com"};
vendors.Add(vendor);
```

```
vendors.Add(new Vendor()
                {VendorId = 5, CompanyName = "ABC Inc", Email = "abc@abc.com"});
vendors.Add(new Vendor()
                {VendorId = 8, CompanyName = "XYZ Inc", Email = "xyz@xyz.com"});
```

Retrieving an Element from a List

```
vendors[1]
```

0	5 "ABC Corp" abc@abc.com
1	8 "XYZ Inc" xyz@xyz.com
2	24 "EFG Ltd" efg@efg.com

Retrieving List Element Best Practices

Do:

Take care when referencing elements by index

Will generate a runtime exception

Avoid:

Retrieving elements by index when you need all elements

Iterate through instead

Iterating Through a Generic List

"Red"
"Espresso"
"White"
"Navy"

5 "ABC Corp" abc@abc.com
8 "XYZ Inc" xyz@xyz.com
24 "EFG Ltd" efg@efg.com

`foreach`

`for`

Iterating a List

`foreach`	`for`
Quick and easy	Complex but flexible
Iterate all elements	Iterate all or a subset of elements
Element is read-only But the element's properties are editable	Element is read/write

Common C# Lists by Namespace

System

- Array

System.Collections (.NET 1)

- ArrayList

System.Collections.Generic (.NET 2+)

- List<T>
- LinkedList<T>
- Queue<T>
- Stack<T>

Selecting an Appropriate List

Array

- Multiple dimensions
- Small performance benefit with large fixed number of elements

ArrayList

- If < .NET 2

List<T>

- Use most often

Selecting an Appropriate List (cont)

LinkedList<T>

- Linked to element before it and after it in sequence
- Insert/remove elements in middle of list

Queue<T>

- Discard element after retrieval
- Access elements in same order as they were added

Stack<T>

- Discard element after retrieval
- Access last added element first

Selecting an Appropriate List (cont)

System.Collections.ObjectModel

- Appropriate for a reusable library
- ReadOnlyCollection
- ObservableCollection

System.Collections.Specialized

- Specialty collections
- StringCollection

System.Collections.Concurrent

- Thread-safe list classes

Selecting a List Best Practices

Do:

Use List<T> unless some other list better meets the requirements

Avoid:

.NET 1 lists in the System.Collections namespace

Such as ArrayList

Frequently Asked Questions

- When is it appropriate to use a generic list?
  - Any time the application needs to manage a list of things.
- What are the key differences between an array and a generic list?
  - An array is fixed length and can have multiple dimensions.
  - A generic list can be any length and provides methods to easily add, insert, or remove elements from the list.

Frequently Asked Questions (cont)

- **What is the difference between `foreach` and `for` when iterating through a list?**
  - `foreach` provides simple syntax for iterating all elements in a list.
  - `for` provides more complex but flexible syntax for iterating all or any subset of elements in a list.
    - Plus the iterated elements are editable.
- **When using `foreach` on a list of objects, is the iterated object editable?**
  - The object instance is not editable.
  - But the object properties are editable.

Summary

Declaring and Populating a Generic List

Using Collection Initializers

Initializing a List of Objects

Retrieving an Element from a Generic List

Iterating Through a Generic List

Types of C# Lists