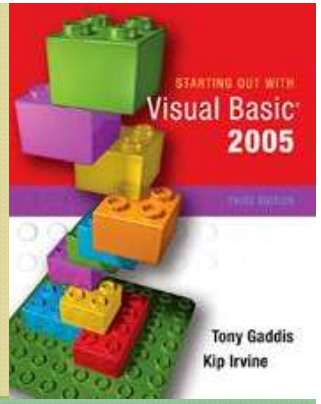
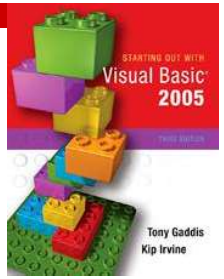


12.3 Collections

A Collection Holds a Group of Items

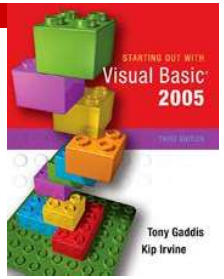
It Automatically Expands and Shrinks in Size to Accommodate the Items Added to It, and Allows Items to Be Stored With Associated Key Values, Which May Be Used in Searches

Collections



- A *collection* is similar to an array
- A single unit that contains several items
- Can access items in a collection by numeric index
- A collection's indices begin at one, not zero
- Collections automatically expand and shrink as items are added and removed
- The items stored in a collection do not have to be of the same type

A Collection is a Class



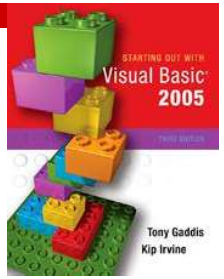

```
Dim customers As Collection
customers = New Collection()

    ' Or alternatively

Dim customers As New Collection()
```

- New collections are instantiations of the *Collection Class*
- The Collection Class provides various methods and properties for use with individual collections

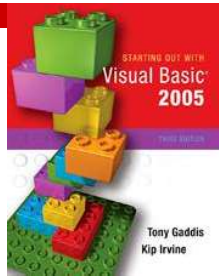
Adding Items to a Collection




```
Object.Add(Item [, Key] [, Before] [,After])
```

- *Add* is a method of the *Collection class*
- *Object* is the collection that *Item* is added to
- *Item* can be an object, variable, or value
- *Key* is a unique value optionally used to identify a member of the collection
- *Before* or *After* is used to specify where a new item should be placed in the collection
- Default is to insert at the end of the collection

Uses of Before and After



- Add custData with key "Smith" before the item with key "Thomas“

```
customers.Add(custData, "Smith", "Thomas")
```

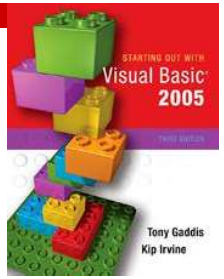
- Add custData with key "Smith" after the item with key "Reece“

```
customers.Add(custData, "Smith",, "Reece")
```

- Add custData after 3rd item in collection

```
customers.Add(custData,,,3)
```

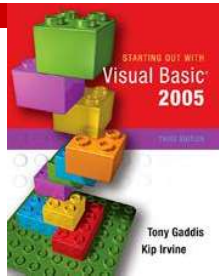
Add Method Exceptions



- An exception can occur when adding to a collection so *Try-Catch* should be used
  - Cannot add member with same key as another member of the collection
  - If a key or index is specified for a Before or After, the value must exist

```
Try
   customers.Add(custData, "Smith")
Catch ex as ArgumentException
   MessageBox.Show(ex.Message)
End Try
```

Accessing Item by Their Indices



- Can access an item in a collection using an index value
- Index value can be used in two ways:
  - Using the collection's *Item* method

    ***Object*.Item(*Index*)**

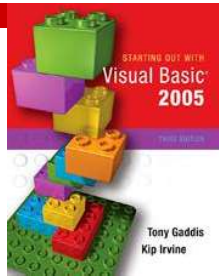
  - Or specify the index as is done with an array

    Object(Index)

- Get value at index 3 of names collection by:

  names.Item(3) –or- names(3)

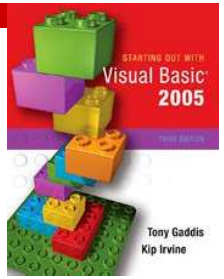
IndexOutOfRange Exception



- If an invalid index is encountered, an index out of range exception will occur
- Should use *Try-Catch* to trap such messages

```
Dim custData as Customer
Try
    custData = Ctype(customers.Item(5), Customer)
Catch ex as IndexOutOfRangeException
    MessageBox.Show(ex.Message)
End Try
```

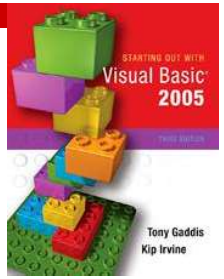
The Count Property



- The *Count* property of a collection gives the number of current items in the collection

```
Dim intX As Integer
For intX = 1 To names.Count
    lstNames.Items.Add(names(intX).ToString())
Next intX
```

Storing Objects in a Collection



- Storing an object, without a key

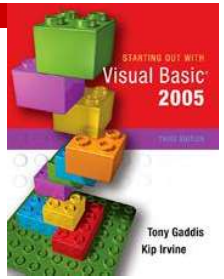
```
Dim studentCollection As New Collection()

studentCollection.Add(studentData)
```

- Storing an object, with a key

```
studentCollection.Add(studentData, _
                      studentData.IdNumber)
```

Searching for an Item by Key Value

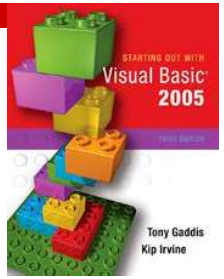


- Item method can be used to retrieve an item with a specific index

```
Object.Item(Expression)
```

- If Expression is a string, it is used as a key to search for the matching member
- If Expression is numeric, it is used as an index value for the item
- If no item is found (via key or index), an exception occurs

Retrieving Item Examples



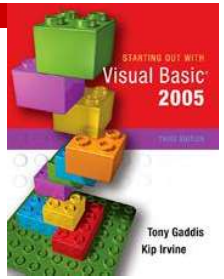
- Find studentCollection item with key 49812
  - If Option Strict on, must cast result to Student

```
Dim s as Student
s = Ctype(studentCollection.Item("49812"), Student)
```

- Retrieve all members by index and display LastName property in a message box

```
Dim i as Integer
Dim s as Student
For I = 1 to studentCollection.Count
    s = Ctype(studentCollection.Item(i), Student)
    MessageBox.Show(s.LastName)
Next i
```

Using References Versus Copies



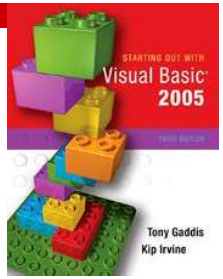
- When an Item in a collection is a fundamental VB data type, only a copy is retrieved
  - This code does not change the item at index 1

```
Dim n as Integer
n = CType(numbers(1), Integer)
n = 0
```

- The Item in this collection is an object so:
  - A reference is returned instead of a copy
  - LastName of object in collection is changed

```
Dim s as Student
s = CType(studentCollection.Item("49812"), Student)
s.LastName = "Griffin"
```

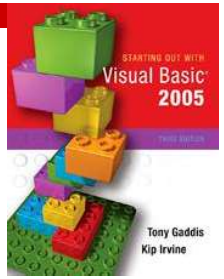
For Each Loop with a Collection



- Can use a For Each loop to read members of a collection
  - Eliminates the counter variable required to use a For…Next
  - Also no need to compare to Count property

```
Dim s As Student
For Each s In studentCollection
      MessageBox.Show(s.LastName)
Next s
```

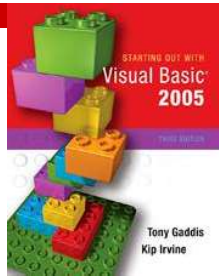
Removing Members of a Collection




```
Object.Remove(Expression)
```

- *Remove* is a method of the *Collection class*
- *Object* is collection *Member* removed from
- *Expression* can be
  - Numeric and interpreted as an index
  - Or a string and interpreted as a key value
  - Exception thrown if *Expression* not found

Removing Member Examples



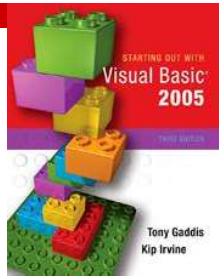
- Verify “49812” is a key value in collection, then remove member with this key value

```
If studentCollection.Contains(“49812”) Then
   studentCollection.Remove("49812")
End If
```

- Verify index location 7 exists in collection, then remove member at this index location

```
Dim intIndex As Integer = 7
If intIndex > 0 _
   and intIndex <=studentCollection.Count Then
      studentCollection.Remove(intIndex)
End If
```

Working with Collections



- Since a collection is an instance of a class
  - Procedures accept collections as arguments
  - Functions can return a collection
  - Follow same guidelines as any class object
- Parallel collections work like parallel arrays
  - Can use index to relate parallel collections just as we did with arrays
  - Or can use key values to relate collections