Module 1 Day 8

Collections, Part 2

What makes an application?

- Program Data
  - ✓ Variables & .NET Data Types
  - ✓ Arrays
  - ❖ More Collections (list, dictionary, stack, queue)
  - ❖ Classes and objects (OOP)
- Program Logic
  - ✓ Statements and expressions
  - ✓ Conditional logic (if)
  - ✓ Repeating logic (for, foreach, do, while)
  - ✓ Methods (functions / procedures)
  - ❖ Classes and objects (OOP)
  - ❑ Frameworks (MVC)
- Input / Output
  - User
    - ✓ Console read / write
    - ❑ HTML / CSS
    - ❑ Front-end frameworks (HTML / CSS / JavaScript)
  - Storage
    - ❑ File I/O
    - ❑ Relational database
    - ❑ APIs

Arrays & Lists

- Accessed by Index [n]
- Iterated by foreach
- Index is always an **integer**, and always starts at 0
- What if I want to lookup state names by their state code?
- What if I want to lookup city by zip code?

```
List<string> stateCodes = new List<string>()
{ "AL", "AK", "AR", "AZ", "CA", "CO", "CT", "DE" }; // etc
List<string> stateNames = new List<string>()
{ "Alabama", "Alaska", "Arkansas", "Arizona",
    "California", "Colorado", "Connecticut", "Deleware"};   // etc
```

Dictionary

- Known as an ***Associative Array***
- Every item is a ***Key-Value Pair***
- Key can be any type; Value can be any type (same or different)

```
// Dictionary<TKey, TValue> name = new Dictionary<TKey, TValue>();
```

- Dictionaries are accessed using the **Key**
  - **Cannot** be accessed by index
- The Key must be unique (Values may be duplicated)

Create a Dictionary

- Declare – Instantiate / Allocate - Initialize

```
// Create a dictionary that associates state codes with state names
Dictionary<string, string> statecodes = new Dictionary<string, string>()
{
    {"AL", "Alabama" },
    {"AK", "Alaska" },
    {"AZ", "Arizona" },
    {"AR", "Arkansas" },
    {"CA", "California" },
    {"CO", "Colorado" },
    {"CT", "Connecticut" },
    {"DE", "Delaware" },        // etc
};
```

Using a Dictionary

- Access elements using [*key*]

```
string stateName = stateCodes["CO"];
```

- Check for existence using ContainsKey

```
if (stateCodes.ContainsKey("CO"))
{
```

- Add a Dictionary entry using Add

```
// Add another state key-value pair
stateCodes.Add("WY", "Wyoming");
```

- Assigning using bracket notation adds or updates the entry

```
// If the OH key already exists, Update it. If not, Add it
stateCodes["OH"] = "Ohio";
```

- Remove an entry using Remove

```
// Remove an existing entry
stateCodes.Remove("DE");
```

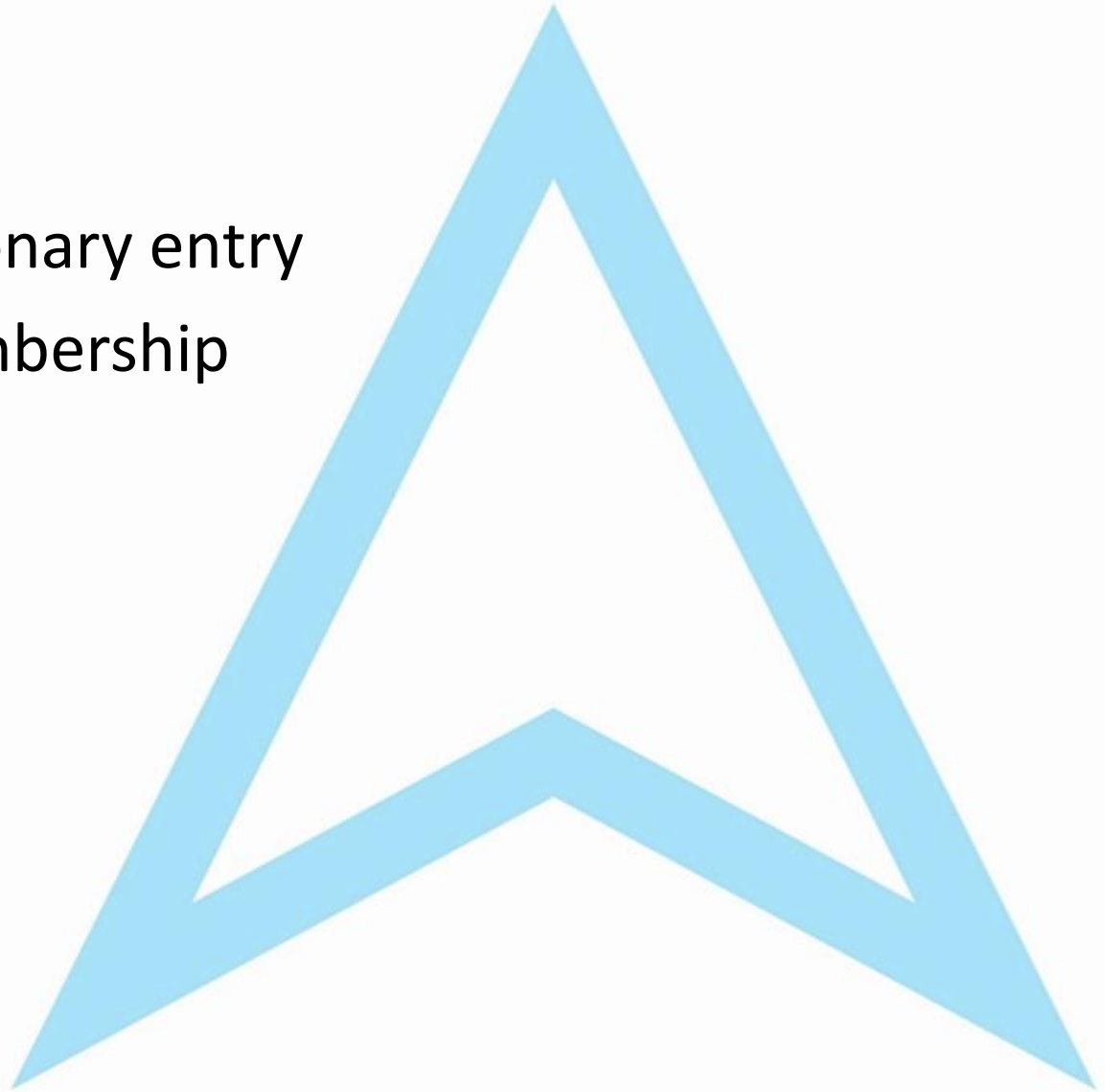
Iterating a Dictionary

- foreach works, but returns a **KeyValuePair**
- From the KeyValuePair, you can get to the **Key** or the **Value**

```
foreach (KeyValuePair<string, string> entry in stateCodes)
{
    Console.WriteLine("State Code: {0}, {1}", entry.Key, entry.Value);
}
```

HashSet

- Stores unique values of any type
- Similar to the “Key” side of a dictionary entry
- Very fast access to determine membership



HashSet Methods

```
// Create and populate a new HashSet that contains existing user names
HashSet<string> userNames = new HashSet<string>()
    {"BettyA", "JoeB", "MichaelQ", "JoeD" };

// A new user selects a name
string newUserName = "JoeB";

// See if the user name already exists
if (!userNames.Contains(newUserName))
{
    // It does not exist, so we can add it
    userNames.Add(newUserName);
}
```

HashSet Methods

- foreach
- Remove(valueToRemove)
- myHashSet.UnionWith(anotherHashSet)
- myHashSet.IntersectWith(anotherHashSet)