Going with the Flow

Dave Fancher

@davefancher | davefancher.com

Pipelining

Pipeline Data Flow

```
Console.WriteLine(
    Encoding.UTF8.GetString(
        new byte [] { 72, 101, 108, 108, 111 }
    )
);
```

Nested Method Calls

```
var bytes = new byte [] { 72, 101, 108, 108, 111 };
var decodedString = Encoding.UTF8.GetString(bytes);
Console.WriteLine(decodedString);
```

Intermediate Variables

```
[| 72uy; 101uy; 108uy; 108uy; 111uy |]
|> Encoding.UTF8.GetString
|> Console.WriteLine
```

Pipelining (F#)

Method Chains ~= Pipelines

Method chaining is an architectural pattern which must be deliberately designed into types being chained

```
Enumerable
    .Range(1, 100)
    .Where(i => i % 2 == 0)
    .Reverse();
```

Example

IEnumerable<T>

StringBuilder

```
...public StringBuilder Append(sbyte value);
public StringBuilder Append(char value);
public StringBuilder Append(short value);
public StringBuilder Append(byte value);
public StringBuilder Append(bool value);
public StringBuilder Append(int value);
public StringBuilder Append(float value);
public StringBuilder Append(decimal value);
...public StringBuilder Append(uint value);
public StringBuilder Append(object value);
...public StringBuilder Append(char[] value);
...public StringBuilder Append(ulong value);
...public StringBuilder Append(ushort value);
public StringBuilder Append(double value);
public StringBuilder Append(long value);
...public StringBuilder Append(string value);
...public StringBuilder Append(char* value, int valueCount);
public StringBuilder Append(char value, int repeatCount);
...public StringBuilder Append(string value, int startIndex, int count);
...public StringBuilder Append(char[] value, int startIndex, int charCount);
public StringBuilder AppendFormat(string format, params object[] args);
public StringBuilder AppendFormat(string format, object arg0);
public StringBuilder AppendFormat(IFormatProvider provider, string format, object arg0);
public StringBuilder AppendFormat(IFormatProvider provider, string format, params object[] args);
public StringBuilder AppendFormat(string format, object arg0, object arg1);
public StringBuilder AppendFormat(IFormatProvider provider, string format, object arg0, object arg1);
public StringBuilder AppendFormat(string format, object arg0, object arg1, object arg2);
public StringBuilder AppendFormat(IFormatProvider provider, string format, object arg0, object arg1, object arg2);
```

StringBuilder Examples

Traditional

```
var sb = new StringBuilder("ABC", 50);
sb.Append(new char[] { 'D', 'E', 'F' });
sb.AppendFormat("GHI{0}{1}", 'J', 'k');
sb.Insert(0, "Alphabet: ");
sb.Replace('k', 'K');
var str = sb.ToString();
```

Method Chain

```
var str =
    new StringBuilder("ABC", 50)
        .Append(new char[] { 'D', 'E', 'F' })
        .AppendFormat("GHI{0}{1}", 'J', 'k')
        .Insert(0, "Alphabet: ")
        .Replace('k', 'K')
        .ToString();
```

Global Chaining

Method chaining tends to break down when working across disparate types

Partial Function Application

```
public int Add (int x, int y) => x + y;

Add(5, 10);
```

Non-Curried Method

```
public Func<int, int> Add (int x) => y => x + y;

Add(5)(10);
```

Curried Method

```
new[] { 2, 4, 6, 8 }.Select(Add(5));
```

Review

- Pipelining allows data to flow between functions
- Method chaining is the OO version of pipelining
- Types must be designed with chaining in mind
- Higher-order extension methods enable global method chaining
- Partial function application improves composability

Course Review

- C#'s built-in functional features improve:
  - Predictability
  - Maintainability
  - Testability
- Controlling side effects through immutability and scope
- Preferring expressions over statements
- Pipelining through higher-order extension methods

Functional Programming in C#

Dave Fancher

@davefancher | davefancher.com