Design Patterns in C# Made Simple

Module 8 Constructing Complex Objects with the Builder Pattern

Zoran Horvat
CEO at Coding Helmet

http://codinghelmet.com
zh@codinghelmet.com
zoranh75

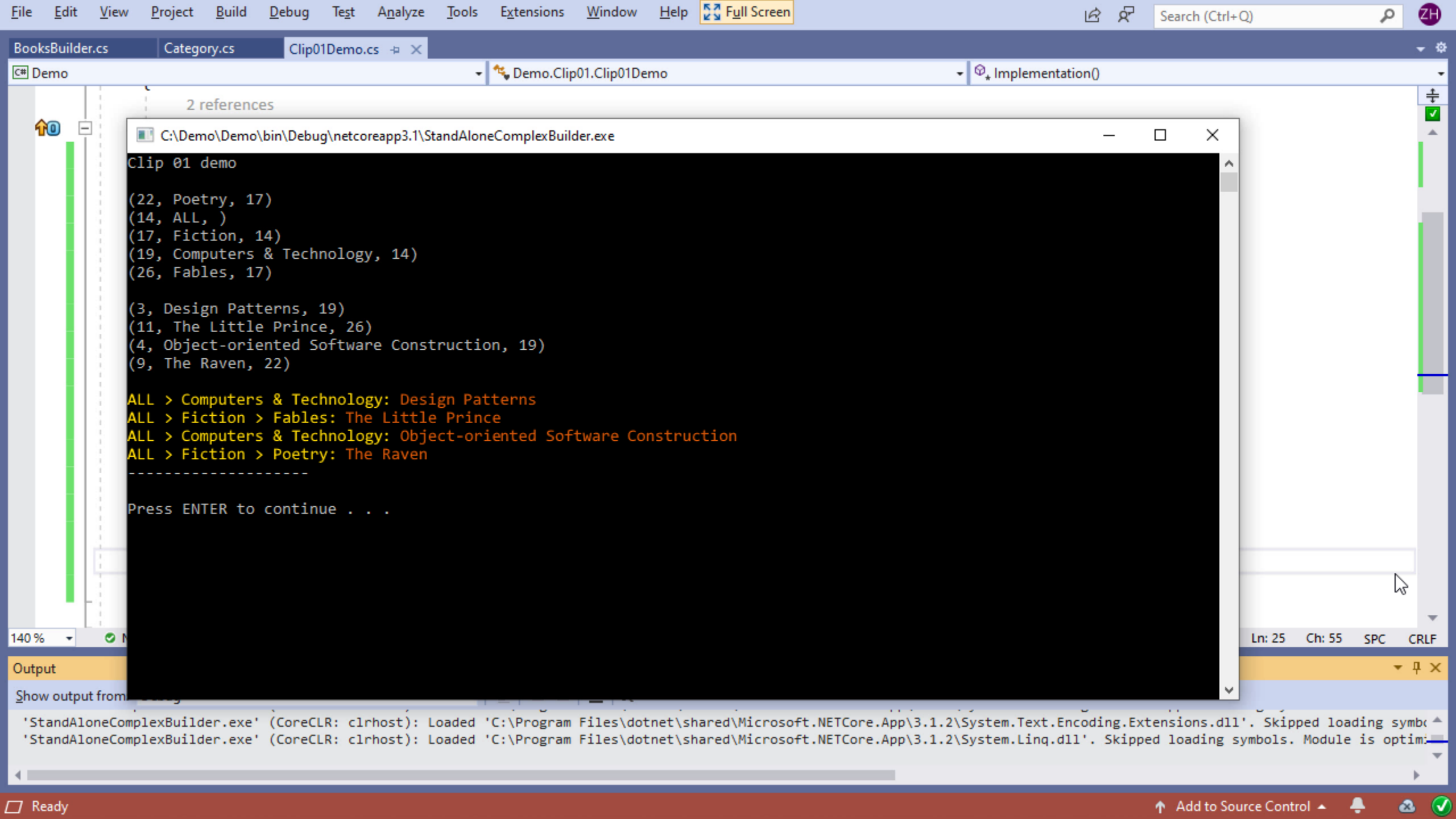
Applying the Builder Pattern

Constructing database connection string

- Connection string builders are common in practice
- The resulting string has complex internal structure

Constructing an object graph

- Product may consist of many small objects
- There are many references inside the object graph
- Builders are used to construct a consistent and complete object graph


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
Search (Ctrl+Q)
BooksBuilder.cs Category.cs Clip01Demo.cs
Demo Demo.Clip01.Clip01Demo Implementation()
2 references
C:\Demo\Demo\bin\Debug\netcoreapp3.1\StandAloneComplexBuilder.exe
Clip 01 demo
(22, Poetry, 17)
(14, ALL,)
(17, Fiction, 14)
(19, Computers & Technology, 14)
(26, Fables, 17)
(3, Design Patterns, 19)
(11, The Little Prince, 26)
(4, Object-oriented Software Construction, 19)
(9, The Raven, 22)
ALL > Computers & Technology: Design Patterns
ALL > Fiction > Fables: The Little Prince
ALL > Computers & Technology: Object-oriented Software Construction
ALL > Fiction > Poetry: The Raven
Press ENTER to continue . . .
140 %
Ln: 25 Ch: 55 SPC CRLF
Output
Show output from
'StandAloneComplexBuilder.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.2\System.Text.Encoding.Extensions.dll'. Skipped loading symbo
'StandAloneComplexBuilder.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\3.1.2\System.Linq.dll'. Skipped loading symbols. Module is optim
Ready
Add to Source Control

```
class BooksBuilder
{
  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);
  IEnumerable<Book> Build();
}
```

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords;
  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords;
  IDictionary<int, Category> Categories;

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();

  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();

  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();

  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();

  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

Empty `BookRecords` produces empty `Books` sequence

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();

  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();

  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

Empty `BookRecords` produces empty `Books` sequence

Non-empty `BookRecords` and empty `CategoryRecords` cause failure

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();

  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();

  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

Empty `BookRecords` produces empty `Books` sequence

Non-empty `BookRecords` and empty `CategoryRecords` cause failure

Builder is initialized in a consistent state with all valid defaults

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();
  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();
  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();
  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);
  IEnumerable<Book> Build();
}
```

Empty `BookRecords` produces empty `Books` sequence

Non-empty `BookRecords` and empty `CategoryRecords` cause failure

Builder is initialized in a consistent state with all valid defaults

All components are designed as optional

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();

  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();

  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

Empty **BookRecords** produces empty **Books** sequence

Non-empty **BookRecords** and empty **CategoryRecords** cause failure

Builder is initialized in a consistent state with all valid defaults

All components are designed as optional

```
class BooksBuilder
{
  IEnumerable<(int id, string title, int categoryId)> BookRecords =
    Enumerable.Empty<(int, string, int)>();

  IDictionary<int, (int id, string name, int? parentId)> CategoryRecords =
    new Dictionary<int, (int id, string name, int? parentId)>();

  IDictionary<int, Category> Categories =
    new Dictionary<int, Category>();

  BooksBuilder WithCategories(IEnumerable<(int id, string name, int? parentId)> rows);
  BooksBuilder WithBooks(IEnumerable<(int id, string title, int categoryId)> rows);

  IEnumerable<Book> Build();
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
  ConnectionStringBuilder WithConnectTimeout(int seconds);
  ConnectionStringBuilder WithDefaultConnectTimeout();
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
  ConnectionStringBuilder WithConnectTimeout(int seconds);
  ConnectionStringBuilder WithDefaultConnectTimeout();
  ConnectionStringBuilder WithProvider(string name);
  ConnectionStringBuilder WithDefaultProvider();
}
```

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
  ConnectionStringBuilder WithConnectTimeout(int seconds);
  ConnectionStringBuilder WithDefaultConnectTimeout();
  ConnectionStringBuilder WithProvider(string name);
  ConnectionStringBuilder WithDefaultProvider();
  bool CanBuild();
  string Build();
}
```

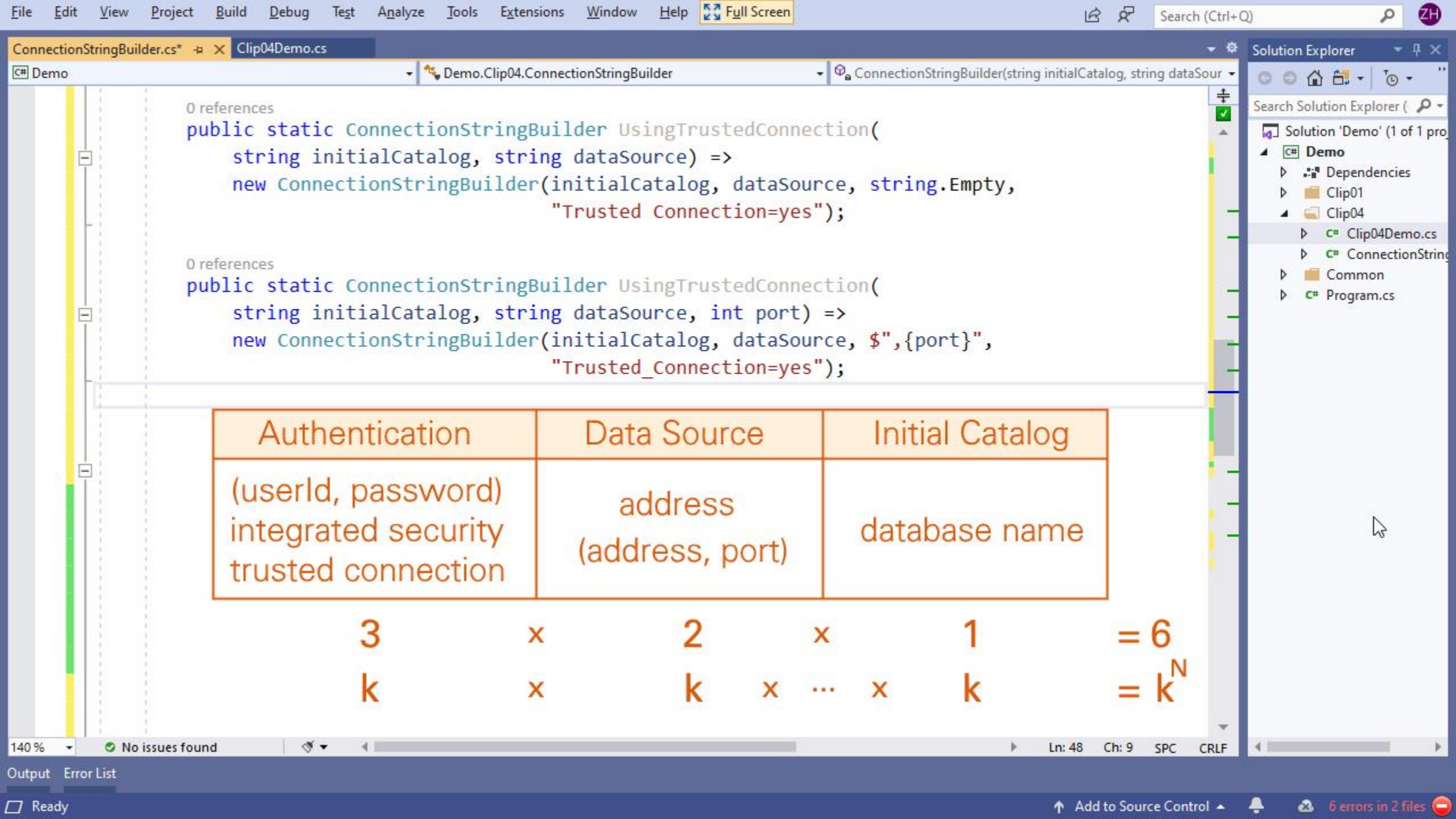
```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
  ConnectionStringBuilder WithConnectTimeout(int seconds);
  ConnectionStringBuilder WithDefaultConnectTimeout();
  ConnectionStringBuilder WithProvider(string name);
  ConnectionStringBuilder WithDefaultProvider();
  bool CanBuild();
  string Build();
}
```

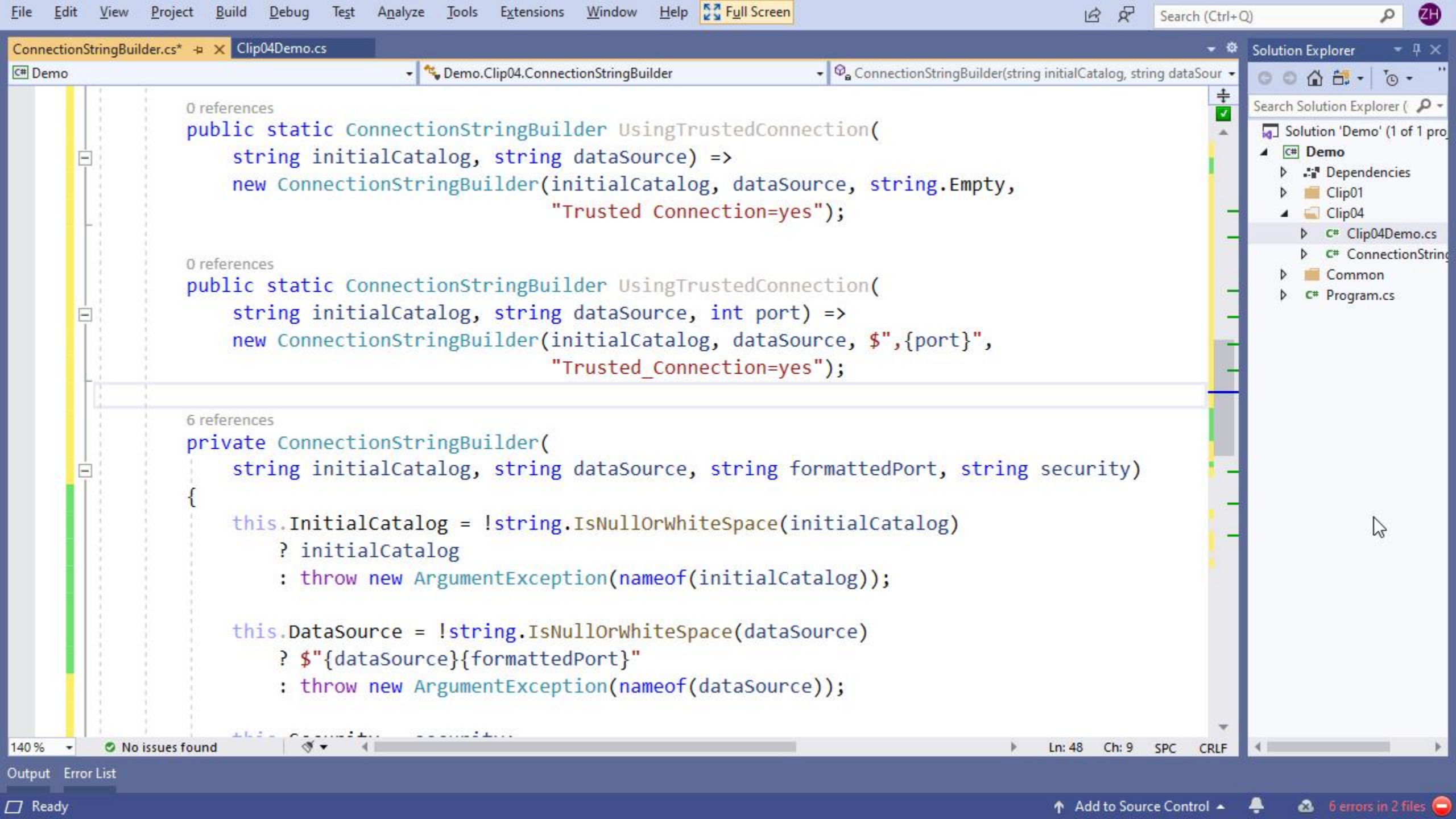
```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;

  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
  ConnectionStringBuilder WithConnectTimeout(int seconds);
  ConnectionStringBuilder WithDefaultConnectTimeout();
  ConnectionStringBuilder WithProvider(string name);
  ConnectionStringBuilder WithDefaultProvider();
  bool CanBuild();
  string Build();
}
```

This builder starts off
in an inconsistent state

```
class ConnectionStringBuilder
{
  string DataSource;
  string InitialCatalog;
  string Security;
  bool IsSecurityValid;
  string ConnectTimeoutSegment;
  string ProviderSegment;
  ConnectionStringBuilder WithDataSource(string address);
  ConnectionStringBuilder WithDataSource(string address, int portNumber);
  ConnectionStringBuilder WithInitialCatalog(string initialCatalog);
  ConnectionStringBuilder WithCredentials(string userId, string password);
  ConnectionStringBuilder UseIntegratedSecurity();
  ConnectionStringBuilder UseTrustedConnection();
  ConnectionStringBuilder WithConnectTimeout(int seconds);
  ConnectionStringBuilder WithDefaultConnectTimeout();
  ConnectionStringBuilder WithProvider(string name);
  ConnectionStringBuilder WithDefaultProvider();
  bool CanBuild();
  string Build();
}
```


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
Search (Ctrl+Q)
ZH
ConnectionStringBuilder.cs*
Clip04Demo.cs
Demo
Demo.Clip04.ConnectionStringBuilder
ConnectionStringBuilder(string initialCatalog, string dataSour
0 references
public static ConnectionStringBuilder UsingTrustedConnection(
string initialCatalog, string dataSource) =>
new ConnectionStringBuilder(initialCatalog, dataSource, string.Empty,
"Trusted Connection=yes");
0 references
public static ConnectionStringBuilder UsingTrustedConnection(
string initialCatalog, string dataSource, int port) =>
new ConnectionStringBuilder(initialCatalog, dataSource, $",{port}",
"Trusted_Connection=yes");
Authentication
Data Source
Initial Catalog
(userId, password)
integrated security
trusted connection
address
(address, port)
database name
3 x 2 x 1 = 6
k x k x ... x k = k^N
Solution Explorer
Search Solution Explorer
Solution 'Demo' (1 of 1 pro
Demo
Dependencies
Clip01
Clip04
Clip04Demo.cs
ConnectionString
Common
Program.cs
140 %
No issues found
Ln: 48
Ch: 9
SPC
CRLF
Output
Error List
Ready
Add to Source Control
6 errors in 2 files


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
Search (Ctrl+Q)
ConnectionStringBuilder.cs* Clip04Demo.cs
Demo
Demo.Clip04.ConnectionStringBuilder
ConnectionStringBuilder(string initialCatalog, string dataSour
0 references
public static ConnectionStringBuilder UsingTrustedConnection(
string initialCatalog, string dataSource) =>
new ConnectionStringBuilder(initialCatalog, dataSource, string.Empty,
"Trusted Connection=yes");
0 references
public static ConnectionStringBuilder UsingTrustedConnection(
string initialCatalog, string dataSource, int port) =>
new ConnectionStringBuilder(initialCatalog, dataSource, $",{port}",
"Trusted_Connection=yes");
6 references
private ConnectionStringBuilder(
string initialCatalog, string dataSource, string formattedPort, string security)
{
this.InitialCatalog = !string.IsNullOrWhiteSpace(initialCatalog)
? initialCatalog
: throw new ArgumentException(nameof(initialCatalog));
this.DataSource = !string.IsNullOrWhiteSpace(dataSource)
? $"{dataSource}{formattedPort}"
: throw new ArgumentException(nameof(dataSource));
Solution Explorer
Search Solution Explorer
Solution 'Demo' (1 of 1 pro
Demo
Dependencies
Clip01
Clip04
Clip04Demo.cs
ConnectionString
Common
Program.cs
140 %
No issues found
Ln: 48 Ch: 9 SPC CRLF
Output Error List
Ready
Add to Source Control
6 errors in 2 files

Summary

Constructing a complex object graph with a Builder

- There may be a complicated process
- Or, an algorithm to create an object graph
- That process can be encapsulated in a builder

Summary

Object consistency principle and the Builder

- Initial state should be consistent
- Transitions should lead to consistent states
- No reference to an impossible object
- Otherwise, we risk failure

Summary

Constructing a consistent builder

- Receive mandatory components through the constructor
- Initialize optional components to valid defaults
- Applicable to simpler cases
- Not applicable to many mandatory components
- Not applicable to complex components

Summary

Consistent complex builder

- Form a chain of related interfaces
- Each interface responsible for one component
- Each interface grants access to the next
- Consumer always observes consistent builder object
- The `Build()` method will never fail

Summary

Immutable builder design

- Builders accumulate components
- It is common to construct mutable builders
- Some designs require a deeply immutable builder

Designing an immutable builder

- Applicable to parallel execution
- Avoids aliasing bugs
- Immutable collections are of great help
- Immutable collections are very efficient