Programming to a Complex Framework

CS 234

Mari Good

Programming Frameworks

- Programming production applications is complex.
  - Programmers typically develop a set of tools over time that they use to build application. Because they have been used and tested and refined over time, these tools generally increase programmer productivity and increase the quality of code that a programmer writes.
  - C# Programmers often use one or more “frameworks” of classes that have been developed by Microsoft or other 3rd party developers as well.
  - In this lab I’m going to introduce you to a “homegrown” framework that can be used to build “relatively simple” persistent classes like the Product and Customer classes illustrated in your book.

Design Guidelines

- When I created this “framework” I had several design objectives in mind. I wanted to be able to create classes that
  - Illustrated “best practices” for students.
    - Use inheritance, abstract classes and interfaces appropriately.
    - Allow for a realistic amount of code re-use.
    - Throw exceptions on error conditions.
  - I could persist to either an xml file or a database with minimal changes in my code.
    - Use an n-tier architecture.

Design Guidelines

- Functioned in a predictable way and shared as much implementation as possible so I could build UI components that I could re-use or inherit from to create new applications without starting over. All business classes should be derived from the same class and should
  - Validate individual properties and throw exceptions when individual properties are set inappropriately
  - Validate “required” properties for an object before saving it to a data store and throw an exception if all required properties are not provided.
  - Allow a programmer to “undo” changes to properties of an object before saving to the data store.
  - Allow the programmer to call ONE method to save the object to the data store and let the class be smart enough to figure out if it should call create, update or delete.
  - Minimize “trips” to the data store to update an object that hasn’t really changed.

The Framework

- Consists of
  - Data Tier Elements
  - Business Tier Elements
  - “Middle Tier” Elements

The Framework

- Consists of
  - Data Tier Elements
    - 2 Interfaces – one that contains methods for reading from a data store and one that contains methods for writing to a data store
    - An abstract class for working with xml files
    - An abstract class for working with MS SQL Server databases

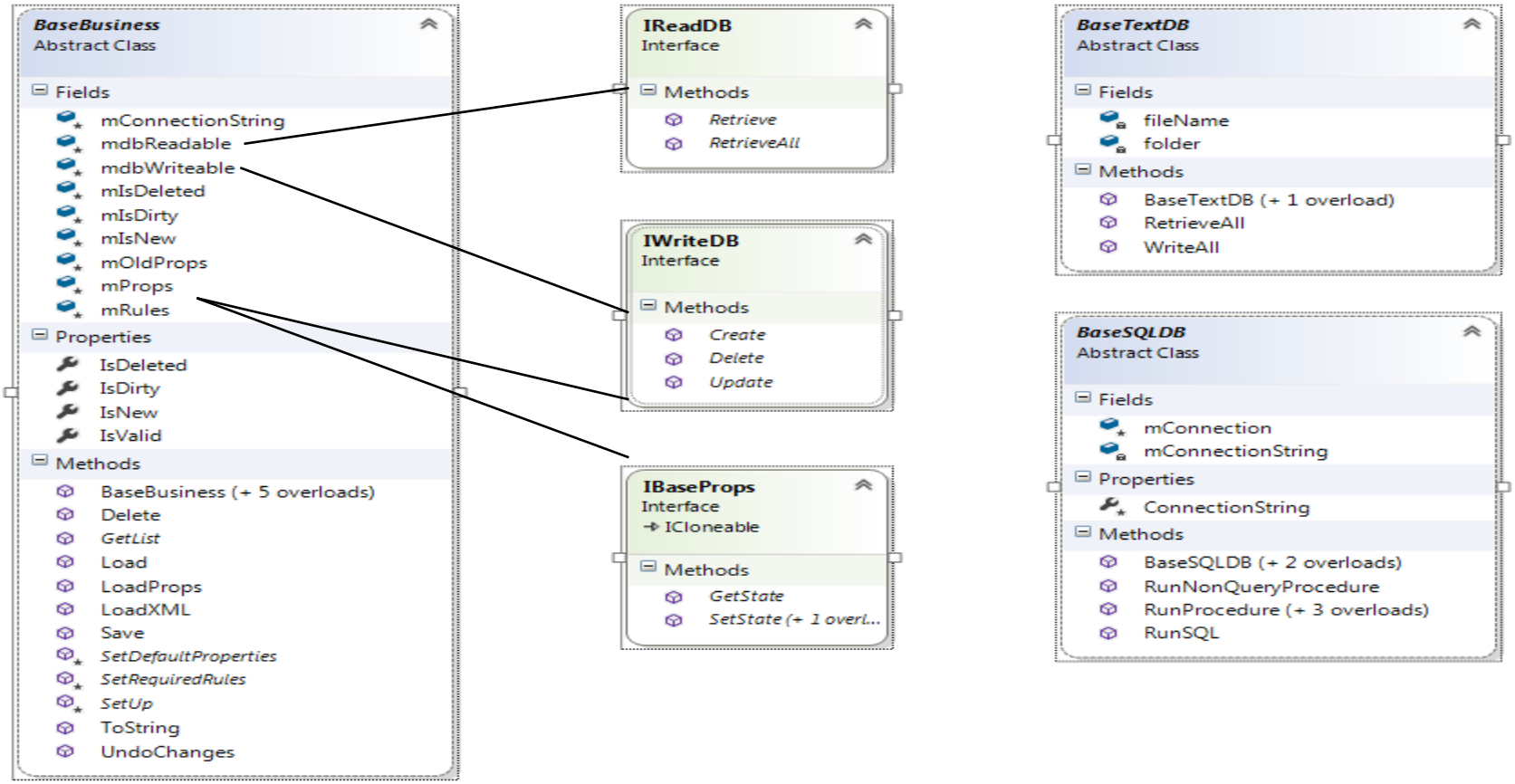
The Framework

- Consists of
  - Business Tier Elements
    - An abstract class that contains most of the logic that implements the design guidelines that were on the previous slides.
      - It includes instance variables for that allow reading and writing to the data store. It calls all of the methods implemented in the interfaces as if the actual class that implements them existed.

The Framework

- Consists of
  - “Middle Tier” Elements
    - Will contain all of the properties of an object and allow for the “conversion” of the properties from the format that the business object works with to the format that the data store works with.

The Framework



The Framework

- Let's start by looking at the BaseBusiness Class
  - Notice the instance variables
    - mdbReadable and mdbWritable. Methods like Load and Save call methods in the corresponding interfaces even though the methods aren't written yet.
    - mProps and mOldProps. Methods like UndoChanges call methods like Clone even though it's not written yet.
  - Notice the abstract methods
    - SetRequiredRules, SetDefaultProperties, Setup. The constructors call these methods, even though they're not written!
  - Any questions about any of the rest of this class?

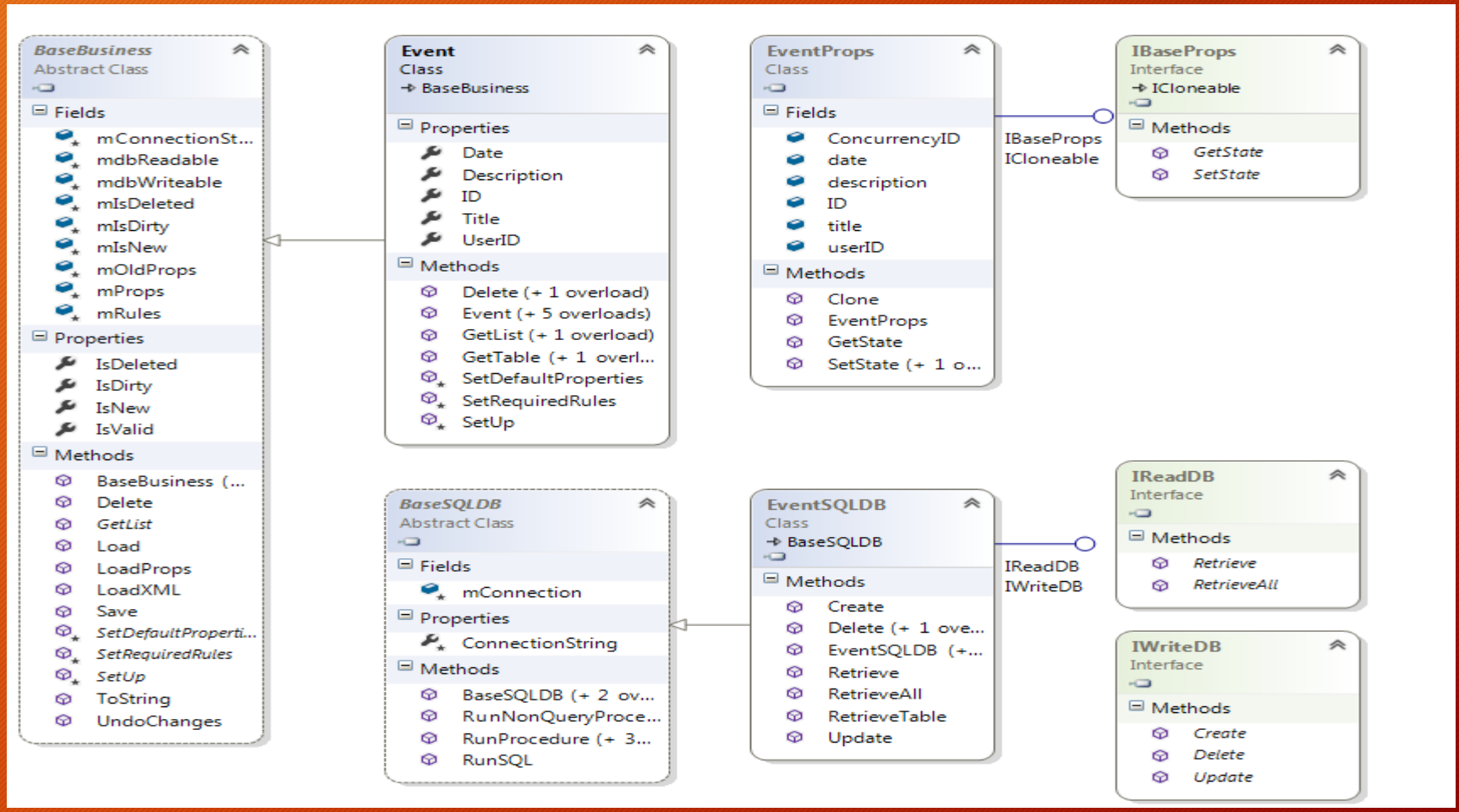
The Framework

- We've already talked about the interfaces but
  - IReadDB
  - IWriteDB
  - IBaseProps
- Anything interesting in the 2 Data Tier classes?
  - BaseTextDB
  - BaseSQLDB

The Event Class(es)

- Consists of
  - Data Tier Elements – EventSQLDB
    - It extends BaseSQLDB and implements IReadDB and IWriteDB.
    - It contains implementations of 2 read methods and 3 write methods required by the interfaces. I calls lots of methods in the base class to do the implementation. Notice all of the alias(es) for sql related class names.
  - Business Tier Elements - Event
    - Extends BaseBusiness and must implement the 3 abstract methods. It contains property procedures (including validation on the setters) that the base class couldn't know about. It also contains constructors which essentially call the constructor on the base class.
    - I added a method that returns a DataTable as an illustration.
  - "Middle Tier" Elements – Event Props
    - EventProps – contains instance variables for each of the properties of the object. It implements ICloneable and IBaseProps. It must contains an implementation of Clone, GetState and SetState.

The Event Class(es)



The Event Class(es)

- Before we begin we should create the EventCalendar database on your machines
  - Find the sql script on moodle. Execute it as we did last week to create the database
  - Look at the structure of the tables. Questions?
  - All of my db classes will used stored procedures rather than embedding sql code in my applications.
    - What is a stored procedure?
    - Why use a stored procedure rather than embedding sql?
    - For each stored procedure
      - Look at the code.
      - Execute it and make sure that it works.

The Event Class(es)

- Look at the implementation of each class from a really high level. What are the methods? How much code in each method comes from the base class? What kind of code is specific to each class? Start with
  - Business Tier Elements - Event
  - “Middle Tier” Elements – EventProps
  - Data Tier Elements – EventSQLDB

The Event Class(es)

- Look at the DETAILED implementation of the database related classes. Start with
  - BaseSQLDB class – Pay attention to how I use the ADO.NET classes and the "utility methods" I provide
  - EventSQLDB class – At this point you should be able to read my code and have a general idea of what it does. You should also be able to make educated guesses about what you'd have to change when you copy and paste to make a ProductSQLDB or CustomerSQLDB class. ☺
- Look at the unit tests
  - Notice SetUp. What am I doing here and why?
  - Notice the test that illustrate concurrency "issues"
  - Make sure all of the tests run on your machine. You'll have to change the dataSource which contains the database connection string information.

Lab 6

- Asks you to create 2 sets of classes using the framework.
  - Create a new version of the MMABooks database with the script that I've provided. The Products table has an autonumber primary key. Both tables have ConcurrencyId fields.
    - Write each stored procedure using the Event stored procedures as a model. Test each procedure as you go.
    - I'll get you started with one or 2 for the Products table.

Lab 6

- Asks you to create 2 sets of classes using the framework.
  - Create a new solution – ProductCustomerStuff and put it in the same folder as the FrameworkLibrary folder
    - Create a class library first – ProductCustomerClasses
    - Create 3 more class libraries – ProductCustomerProps, ProductCustomerDB and ProductCustomerTests
    - Add the ToolsCSharp project to the solution too
    - Make sure that you install the Nunit NuGet Package to the test project.
    - Make sure that all of the other projects know about ToolsCSharp
    - Make sure that the DB project knows about the Props project
    - Make sure that the Classes project knows about the Props project and the DB project

Lab 6

- The Product Class(es)
  - "Middle Tier" Elements – Do this one first
    - ProductProps – contains instance variables for each of the properties of the object. It implements ICloneable and IBaseProps. It must contains an implementation of Clone, GetState and SetState.
    - Let's talk about what the properties of a product should be.
    - Write the headings for all of the required methods in each and hardcode a return value so that the code will compile. Rebuild the solution.
    - Write one method at a time, testing as you go. It's hard to test the version of SetState that uses a data reader without connecting to the database. We'll leave the testing of that until after we've written Retrieve in the db class.

Lab 6

- The Product Class(es)
  - Data Tier Elements – Do this one second
    - Add the ProductSQLDB class. extends BaseSQLDB and implements IReadDB and IWriteDB. It must contains an implementation of Retrieve, Create, Update and Delete.
    - Write the headings for all of the required methods in each and hardcode a return value so that the code will compile. Rebuild the solution.
    - Write one DB method at a time beginning with Retrieve, testing as you go. We didn't test this version of SetState so our testing might indicate a problem there that we'll have to correct.

Lab 6

- The Product Class(es)
  - Business Tier Elements – Do this one last
    - Product – extends BaseBusiness and must implement the 3 abstract methods. It contains property procedures (including validation on the setters) that the base class couldn't know about. It also contains constructors which essentially call the constructor on the base class. Write the headings for all of the required methods in each and hardcode a return value so that the code will compile. Rebuild the solution.
    - Write one method at a time, testing as you go.

Lab 6

- The Customer Class(es) - Now repeat the same process.
  - "Middle Tier" Elements – Do this one first
    - Write and Test Clone, GetState, SetState
  - Data Tier Elements – Do this one second
    - Write your stored procedures first
    - Write and Test Retrieve, Create, Update, Delete
  - Business Tier Elements – Do this one last
    - Mimic the tests I wrote for the Event class and the tests we just wrote for the Product class.

What's Next

- You'll have at least one class period to "just program" your Customer class(es)
- Entity Framework and LINQ
  - Chapters 23 and 24 in the text. Reading Quiz 7.
  - We'll start with Entity Framework and use LINQ to build an application using Entity Framework generated classes.
  - Skim chapter 23 focusing on 724-740. It won't make a lot of sense until you use it but you should get the general idea.
  - We'll walk through most of chapter 24 together in class using the MMABooks database from the text and the example application in the book. That's lab 7.
- Quiz 2 is an optional exercise.