Chapter 1

FRAMEWORKS

CHAPTER 1 FRAMEWORKS

The goal of frameworks is to make your business classes reusable in different applications and different mediums. When you write an object that represents a savings account, the Application Kit allows you to display that information in a graphical user interface, the Enterprise Objects Framework allows you to insert it into a relational database, and the WebObjects Framework allows you to display it in an HTML document.

It's crucial to remember that the object is the most powerful representation of your data. For example, the contents of the relational database may represent all the data, but it has none of the behavior of the object.

Goal

To explore the role frameworks play in the development process.

Prerequisites

Participation in at least one software development project that was late and over budget.

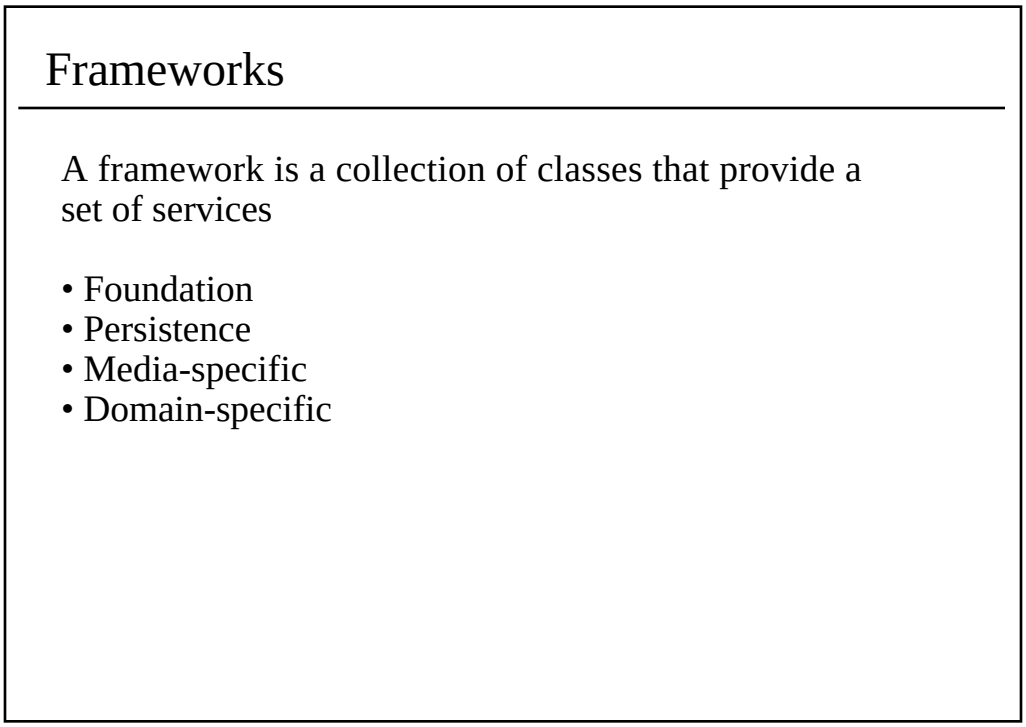
Objective

After this chapter, you'll be able to identify the role of available frameworks.

Reading

The following reference contains more information about frameworks:

/System/Library/Frameworks/FrameworkName.framework/Resources/English.lproj/Documentation/...



Frameworks

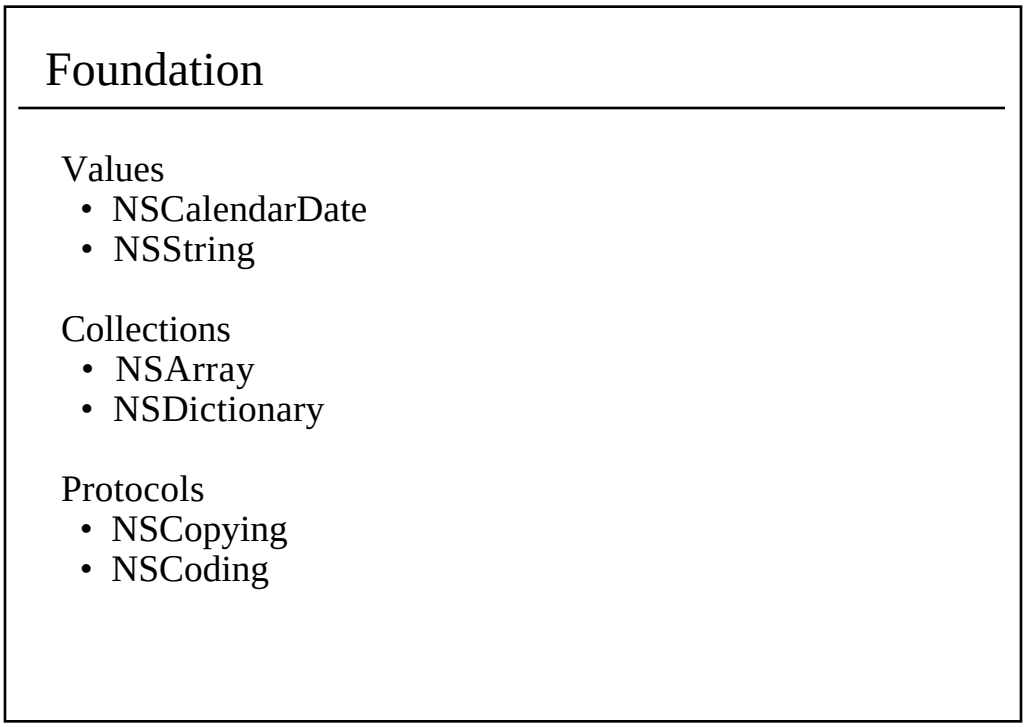
A framework is a collection of classes that provide a set of services. There are several types of frameworks.

Foundation frameworks provide classes to represent values and collections. Values are objects like dates and strings. Collections are objects like arrays, sets, and dictionaries.

Persistence frameworks give objects the ability to live longer than the process that created them, often by storing the data in a relational database. Enterprise Objects is a persistence framework.

Media-specific frameworks deal with presenting data in a specific representation. They primarily deal with input and output. The Application Kit is a framework for displaying objects in a graphic user interface, and relaying input from users back to the model objects. WebObjects is a framework for getting objects in and out of HTML documents.

Domain-specific frameworks are developed by corporations or consultants. These frameworks represent business specific concepts. For example, a large investment bank might develop a framework of classes that represent stocks, bonds, trades, options, and futures.



Foundation

The Foundation Framework defines a base layer of Objective-C classes for all the other frameworks. In addition to providing a set of useful primitive object classes, it introduces several paradigms that define functionality not covered by the Objective-C language. Foundation was designed to accomplish the following goals:

» Provide a set of basic utility classes

» Make software development easier by introducing consistent conventions for things such as object ownership

» Support Unicode strings, object persistence, and object distribution

» Provide a level of operating system independence, enhancing application portability

Persistence

Imagine that you create an object in some application:

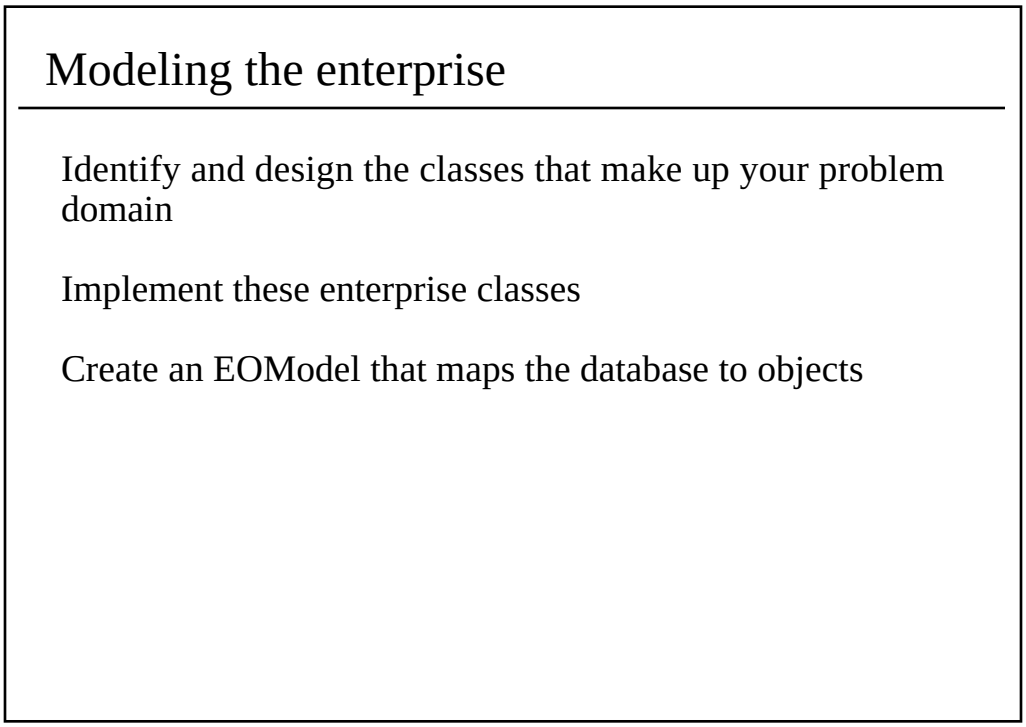
```
myEmployee = [[Employee alloc]init];
```

What happens to that object when the program ends? It's gone. The object is simply a piece of memory. When the application terminates, all the objects in its memory are destroyed.

Many of the objects you create have important data that you want to stay around after the program exits. The ability of an object to exist for longer than the life of a single application is called **persistence**. There are three general methods of creating persistence:

- Put the object in a database
- Put the object in a file
- Copy the object into the memory of another process

Enterprise Objects is a persistence framework that stores object data in a database. Foundation includes support for storing object data in files or distributing objects across the network.



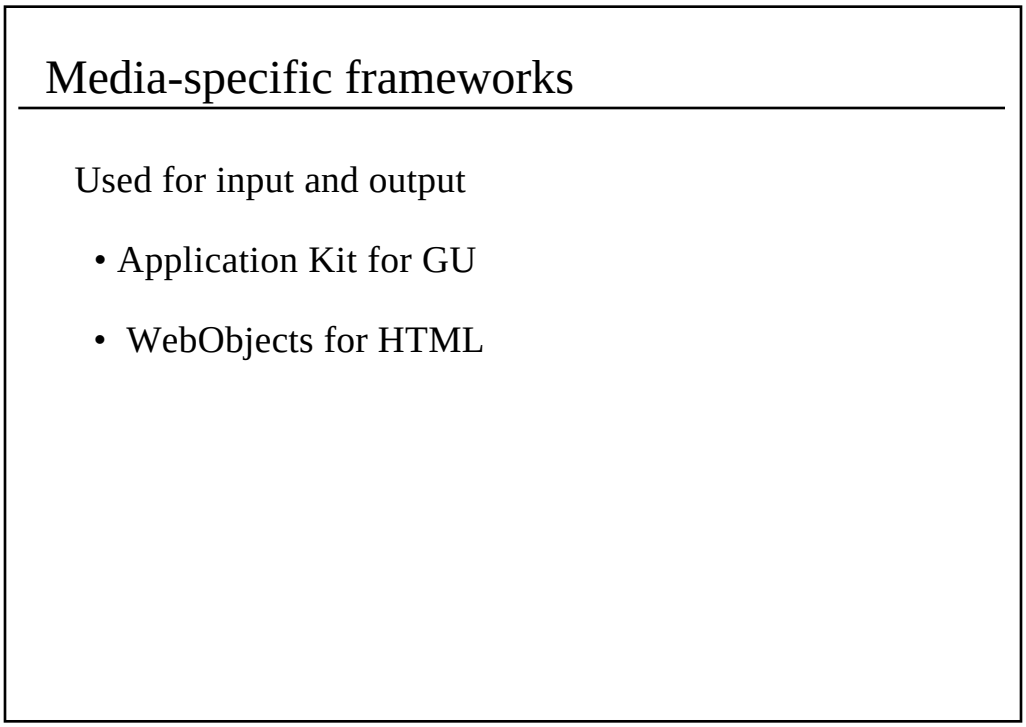
Modeling the enterprise

The first step in building your business objects is to identify the classes in your problem domain. Apple refers to these business classes as **enterprise classes**. One of the goals of object-oriented programming is to make sure there is only one version of these classes.

Writing a good set of enterprise classes is a very difficult task. You will probably have to redesign, implement, and test them a few times before they're perfect. Each time you revisit your enterprise classes, you'll find your understanding of the business processes they model is deeper.

The good news is that once you've accomplished the difficult task of creating a good set of enterprise classes, you can reuse them in a wide variety of applications. Because all your applications use the same set of business classes, maintenance of the source code is much easier.

After building your enterprise classes, you generally want to store the data from them in a relational database. The mapping from business classes to a relational database is called an **EOModel**. The EOModel is also reusable.

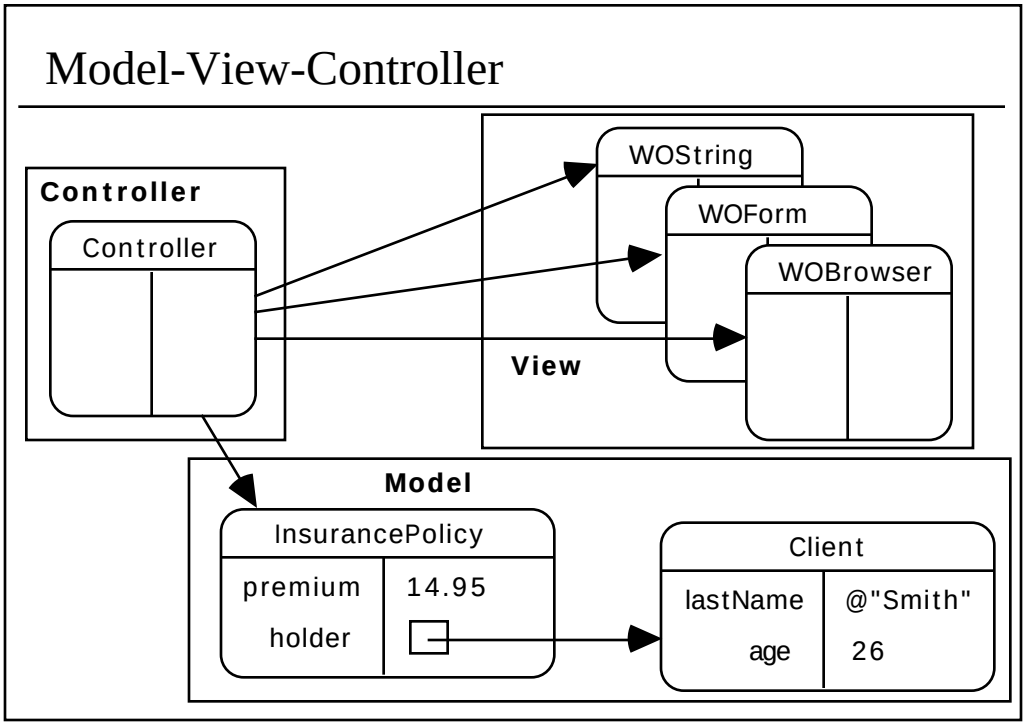


Media-specific frameworks

The Application Kit helps you display your business objects in a graphical user interface. All classes from the Application Kit are prefixed with the letters NS.

The WebObjects Framework helps integrate your business objects into web documents. All WebObjects classes start with the letters WO.

You should only have to write your business objects once. The business objects model your problem domain and should contain no information about databases, HTML, or graphical user interfaces.



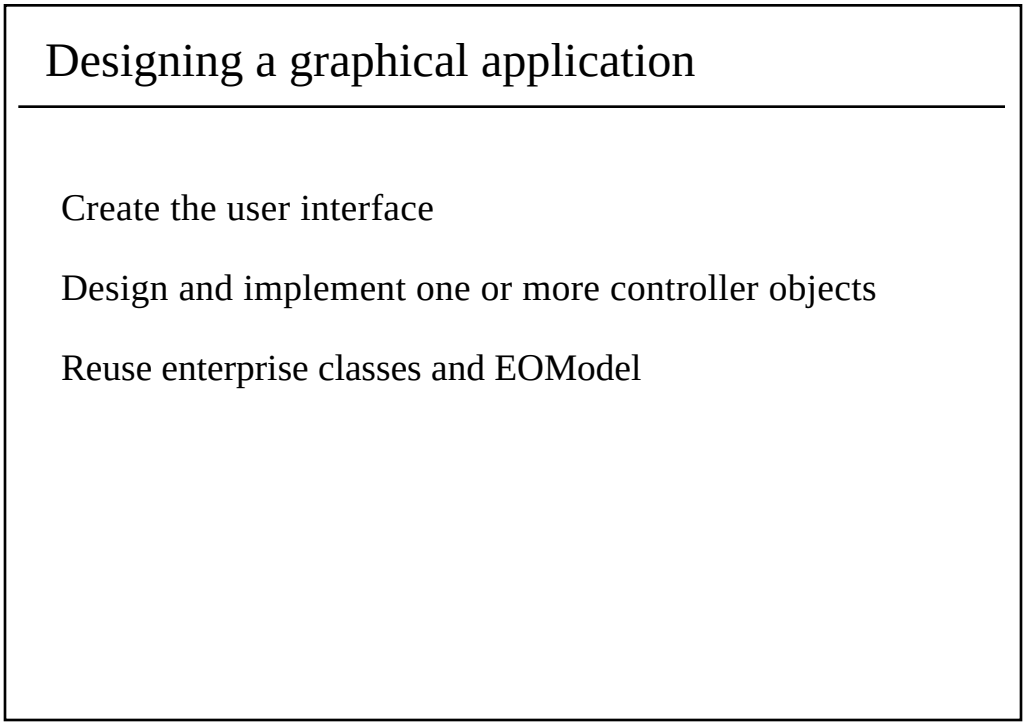
Model-View-Controller

Once you have modeled your enterprise, it's time to start writing applications. Applications typically have three parts:

- Model: The model consists of all your enterprise objects, including their behavior and inter-relationships. The model is completely reusable in any application written for your enterprise.
- View: The view is the screen onto which your model is projected. This is the representation of your data, whether in a traditional graphical application or a web-based application. The objects that make up a view are reusable in any application, and often provided by Apple or third parties. The Application Kit is made up almost entirely of view objects.
- Controller: Controllers synchronize the model and the view. Logic specific to an application, rather than to the problem domain as a whole, goes in controller objects. Therefore, they are the least reusable.

Model-View-Controller is an example of a design pattern. A **design pattern** is a strategy for solving problems that occur frequently. A good framework uses a few design patterns consistently, making it intuitive to use.

Model-View-Controller is one of several patterns used throughout the frameworks. The most important part of this class is to introduce you to these design patterns so the frameworks seem consistent and intuitive as you use them. You've already seen one other design pattern in Chapter 2—target/action.

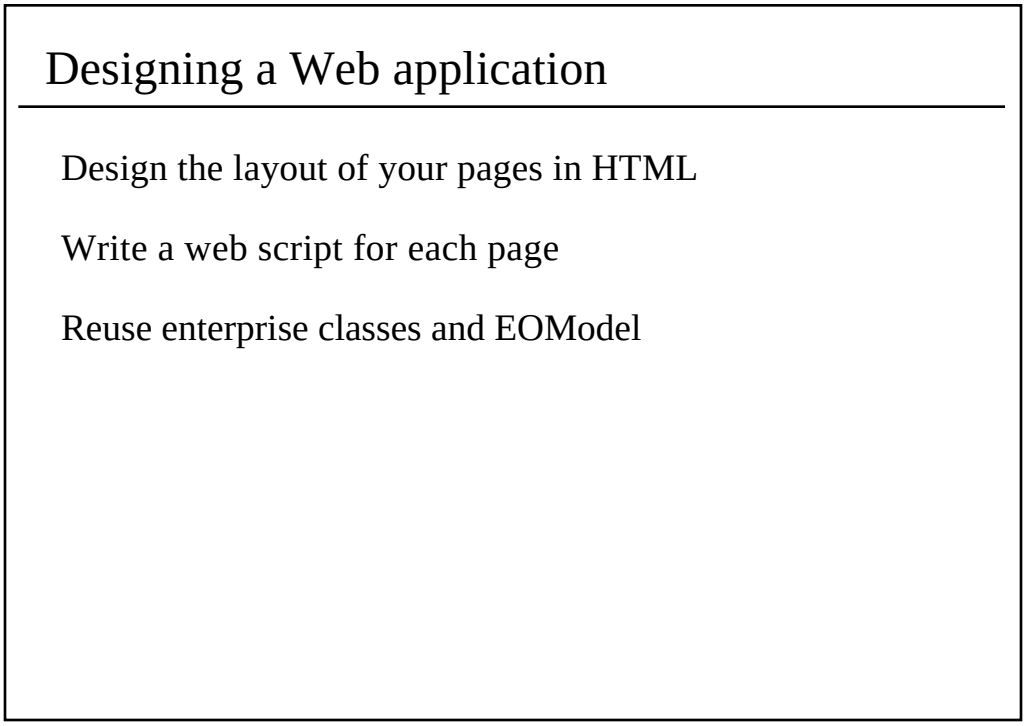


Designing a graphical application

When you write an application with a graphical user interface, the three steps correspond very naturally to the Model-View-Controller design pattern.

- Create a user interface for the application. Generally, you do this using Interface Builder and classes from the Application Kit. This step creates the View portion of your application.
- Design and implement one or more controller objects to coordinate the information in your view with your enterprise objects. These objects represent the Controller of your application.
- Reuse enterprise classes and their mapping to the database, if any. The Model of your problem domain should be reusable for any application.

Notice that the only classes you have to write from scratch are the controllers. The objects used to create the view are provided by Apple, and someone at your enterprise has created a good object model of the problem domain.



Designing a Web Application

Designing a web site using WebObjects also uses the Model-View-Controller pattern:

- Design the layout of your pages in HTML using WebObjects tags and reusable components. This is the View of your web application.
- Write a web script for each page to coordinate information in the view with the model objects running on the server. The scripts function as the Controller for your web application.
- Reuse enterprise classes and their mapping to the database, if any. The Model of your problem domain should be reusable for any application.

Again, you can reuse components when creating your View, and the Model is reusable for any type of application, not just an application with a traditional graphical user interface. The only truly application-specific coding is in creating the Controller.

Important ideas from this section

- A framework is a collection of classes that provide a set of services
- There are four types of frameworks:

  Foundation

  Persistence

  Media-specific

  Domain-specific

- Persistence is the ability for an object to last longer than the process that created it
- Persistence is usually achieved by:

  Saving objects in a database

  Saving objects in a file

  Copying objects into the memory of another process

- The mapping of data from a relational database to objects is called an EOModel
- Apple's two media-specific frameworks:

  Application Kit

  WebObjects

- Both media-specific frameworks use the Model-View-Controller design pattern

REVIEW FRAMEWORKS AND THE ENTERPRISE OBJECT

1. Name the framework each of the following classes is part of:
   - EODatabaseChannel
   - WOSubmitButton
   - NSTextField
   - NSCalendarDate
   - NSArray
2. Define persistence.
3. What is the function of an EOModel?
4. Explain the Model-View-Controller design pattern in your own words.
5. How do Apple's technologies make creating applications easier using the Model-View-Controller design pattern?