Encapsulating Data

Properties Encapsulate an Object’s Values

> Declare Public Properties for Exposed Data

> Use Accessor Methods to Get or Set Property Values

> Dot Syntax Is a Concise Alternative to Accessor Method Calls

> Most Properties Are Backed by Instance Variables

> You Can Customize Synthesized Instance Variable Names

> You Can Define Instance Variables without Properties

> Access Instance Variables Directly from Initializer Methods

```
- (id)init {
    self = [super init];

    if (self) {
        // initialize instance variables here
    }

    return self;
}
```

An init method should assign self to the result of calling the superclass’s initialization method before doing its own initialization. A superclass may fail to initialize the object correctly and return nil so you should always check to make sure self is not nil before performing your own initialization.

By calling [super init] as the first line in the method, an object is initialized from its root class down through each subclass init implementation in order. Figure 3-1 shows the process for initializing an XYZShoutingPerson obje

> The Designated Initializer is the Primary Initialization Method
If an object declares one or more initialization methods, you should

decide which method is the designated initializer.
This is often the method that offers the most options for initialization (such as the method with the most arguments), and is called by other methods you write for convenience.
You should also typically override init to call your designated initializer with suitable default values.

> You Can Implement Custom Accessor Methods

> Properties Are Atomic by Default

Manage the Object Graph through Ownership and Responsibility

> Avoid Strong Reference Cycles

> Use Strong and Weak Declarations to Manage Ownership

> Use Unsafe Unretained References for Some Classes

> Copy Properties Maintain Their Own Copies