;B_ToDo_SettingUp.rtfd;;¬ Previous Section ;D_ToDo_Subclass1.rtfd;;¬ Next Section

4. To Do Tutorial

Creating the Model Class (ToDoItem)

The ToDoItem class provides the model objects for the To Do application. Its instance variables hold the data that defines tasks that should be done or appointments that have to be kept. Its methods allow access to this data. In addition, it provides functions that perform helpful calculations with that data. ToDoItem thus encapsulates both data *and* behavior that goes beyond accessing data.

Since ToDoItem is a model class, it has no user-interface duties and so the expedient course is to create the class without using Interface Builder. We first add the class to the project; Project Builder helps out by generating template source-code files.

1 Add the ToDoItem class to the project.

Select Classes in the project browser.
Choose New In Project from the File menu.
In the New File In ToDo panel, type ªToDoItemº in the Name field.
Make sure the ªCreate headerº switch is checked.
Click the OK button.

TD_CreatingModelClass.eps ¬

As you've done before with Travel Advisor, start by declaring instance variables and methods in the header file, **ToDoItem.h**.

2 Declare ToDoItem's instance variables and methods.

Type the instance variables as shown at right.

Indicate the protocols adopted by this class.

```
@interface ToDoItem:NSObject<NSCoding, NSCopying>
{
    NSCalendarDate *day;
    NSString *itemName;
    NSString *notes;
    NSTimer *itemTimer;
    long secsUntilDue;
    long secsUntilNotif;
    ToDoItemStatus itemStatus;
}
```

You are adopting the NSCopying protocol in addition to the NSCoding protocol because you are going to implement a method that makes ªsnapshotº copies of ToDoItem instances.

Instance Variable	**What it Holds**
TableHeadRule.eps ¬	
day	The day (a date resolved to 12:00 AM) of the to-do item
TableRule.eps ¬	
itemName	The name of the to-do item (the content's of a document text field)
419488_TableRule.eps ¬	
notes	The contents of the inspector's Notes display; this could be any information related to the to-do item, such as an agenda to discuss at a meeting.
528181_TableRule.eps ¬	
itemTimer	A timer for notification messages.
650634_TableRule.eps ¬	
secsUntilDue	The seconds after **day** at which the item comes due
762569_TableRule.eps ¬	
secsUntilNotif	The seconds after **day** at which a notification is sent (before **secsUntilDue**)
885413_TableRule.eps ¬	
itemStatus	Either ªincomplete,º ªcomplete,º or ªdeferToNextDayº

3 Define enum constants for use in ToDoItem's methods.

Define these constants before the **@interface** directive.

```
typedef enum _ToDoItemStatus {
    incomplete=0,
    complete,
    deferToNextDay
} ToDoItemStatus;

enum {
    minInSecs = 60,
    hrInSecs = (minInSecs * 60),
    dayInSecs = (hrInSecs * 24),
    weekInSecs = (dayInSecs * 7)
};
```

The first set of constants are values for the **itemStatus** instance variable. The second set of constants are for convenience and clarity in the methods that deal with temporal values.

4 Declare two time-conversion functions.

```
BOOL ConvertSecondsToTime(long secs, int *hour, int *minute);
long ConvertTimeToSeconds(int hr, int min, BOOL flag);
```

These functions provide computational services to clients of this class, converting time in seconds to hours and minutes (as required by the user interface), and back again to seconds (as stored by ToDoItem).

Type the method declarations shown below.

```
- (id)initWithName:(NSString *)name andDate:(NSCalendarDate *)date;
- (void)dealloc;
- (BOOL)isEqual:(id)anObject;
- (id)copyWithZone:(NSZone *)zone;
- (id)initWithCoder:(NSCoder *)coder;
- (void)encodeWithCoder:(NSCoder *)coder;
- (void)setDay:(NSCalendarDate *)newDay;
- (NSCalendarDate *)day;
- (void)setItemName:(NSString *)newName;
- (NSString *)itemName;
- (void)setNotes:(NSString *)notes;
- (NSString *)notes;
- (void)setItemTimer:(NSTimer *)aTimer;
- (NSTimer *)itemTimer;
- (void)setSecsUntilDue:(long)secs;
- (long)secsUntilDue;
- (void)setSecsUntilNotif:(long)secs;
- (long)secsUntilNotif;
- (void)setItemStatus:(ToDoItemStatus)newStatus;
- (ToDoItemStatus)itemStatus;
```

Most of these declarations are for accessor methods. You know what to do.

5 Implement accessor methods.

Open ToDoItem.m in the code editor.
Implement methods that get and set the values of ToDoItem's instance variables.
Implement the setItemTimer: method as shown below.

```
- (void)setItemTimer:(NSTimer *)aTimer
{
    if (itemTimer) {
```

```
        [itemTimer invalidate];
        [itemTimer autorelease];
    }
    itemTimer = [aTimer retain];
}
```

The **setItemTimer**: method is slightly different from the other ªsetº accessor methods. It sends **invalidate** to **itemTimer** to disable the timer before it autoreleases it.

> Timers (instances of NSTimer) are always associated with a run loop (an instance of NSRunLoop). See ``Tick Tock Brrrring: Run Loops and Timers'' ;ToDoConcepts.rtfd;linkMarkername TickTockBrrrring:RunLoopsandTimers;¬ for more on timers and run loops.

In this application, you want client objects to be able to copy your ToDoItem objects and test them for equality. You must define this behavior yourself.

6 Implement the isEqual: method.

```
- (BOOL)isEqual:(id)anObject
{
    if ([anObj isKindOfClass:[ToDoItem class]] &&
        [itemName isEqualToString:[anObj itemName]] &&
        [day isEqualToDate:[anObj day]])
        return YES;
    else
        return NO;
}
```

The default implementation of **isEqual:** (in NSObject) is based on pointer equality. However, ToDoItem has a different basis for equality; any two ToDoItem objects for the same calendar day and having the same item name are considered equal. The implementation of **isEqual:** overrides NSObject to make these tests. (Note

that it invokes NSString's and NSDate's own **isEqual...** methods for the specific tests.)

938726_TableRule.eps ¬***Before You Go On***

There is a specific as well as a general need for the **isEqual:** override. In the To Do application, an NSArray contains a day's ToDoItems. To access them, other objects in the application invoke several NSArray methods that, in turn, invoke the **isEqual:** method of each object in the array.
256075_TableRule.eps ¬

7 Implement the copyWithZone: method.

```
- (id)copyWithZone:(NSZone *)zone
{
    ToDoItem *newobj = [[ToDoItem alloc] initWithName:itemName
        andDate:day];
    [newobj setNotes:notes];
    [newobj setItemStatus:itemStatus];
    [newobj setSecsUntilDue:secsUntilDue];
    [newobj setSecsUntilNotif:secsUntilNotif];

    return newobj;
}
```

This implementation of the **copyWithZone:** protocol method makes a copy of a ToDoItem instance that is an independent replicate of the original (**self**). It does this by allocating a new ToDoItem object and initializing it with the essential instance variables held by **self**. Copying is often implemented for *value* objectsÐ objects that represent attributes such as numbers, dates, and to-do items.

> Copies of objects can be either deep or shallow. In deep copies (like ToDoItem's) every copied instance variable is an independent replicate, including the values referenced by pointers. In shallow copies, pointers are copied but the referenced objects are the same. For more on this topic, see the description of the NSCopying protocol in the Foundation reference documentation.

The next method you'll implementÐ**description**Ðassists you and other developers in debugging the To Do application with **gdb**. When you enter the **po** (print object) command in **gdb** with a ToDoItem as the argument, this **description** method is invoked and essential debugging information is printed.

8 Implement the description method.

```
- (NSString *)description
{
    NSString *desc = [NSString stringWithFormat:@"%@\n\tName: %@\n\tDate: %@\n\tNotes:
%@\n\tCompleted: %@\n\tSecs Until Due: %d\n\tSecs Until Notif: %d",
            [super description],
            [self itemName],
            [self day],
            [self notes],
            (([self itemStatus]==complete)?@"Yes":@"No"),
            [self secsUntilDue],
            [self secsUntilNotif]];

    return (desc);
}
```

9 Implement ToDoItem's initializing and deallocation methods.

Here are some things to remember as you implement **initWithName:andDate:** and **dealloc**:

SquareBullet.eps ¬ If the first argument of **initWithName:andDate:** (the item name) is not a valid string, return **nil**. If the second argument (the date) is **nil**, set the related instance variable to some reasonable value (such as today's date). Also, be sure to invoke **super**'s **init** method.

609779_SquareBullet.eps ¬ The instance variables to initialize are **day**, **itemName**, **notes**, and **itemStatus** (to ªincompleteº).

727869_SquareBullet.eps ¬ In **dealloc**, release those object instance variables initialized in **initWithName:andDate:** plus any object instance variables that were initialized later. Also invalidate any timer before you release it.

10 Implement ToDoItem's archiving and unarchiving methods.

When you implement **encodeWithCoder:** and **initWithCoder:**, keep the following in mind:

954074_SquareBullet.eps ¬ Encode and decode instance variables in the same order.

65074_SquareBullet.eps ¬ Copy the object instance variables after you decode them.

173937_SquareBullet.eps ¬ You don't need to archive the **itemTimer** instance variable since timers are re-set when a document is opened.

The final step in creating the ToDoItem class is to implement the functions that furnish ªvalue-addedº behavior.

11 Implement ToDoItem's time-conversion functions.

```
long ConvertTimeToSeconds(int hr, int min, BOOL flag)          /* 1 */
{
    if (flag) { /* PM */
        if (hr >= 1 && hr < 12)
            hr += 12;
    } else {
        if (hr == 12)
            hr = 0;
    }
    return ((hr * hrInSecs) + (min * minInSecs));
}
```

```
BOOL ConvertSecondsToTime(long secs, int *hour, int *minute)   /* 2 */
{
    int hr=0;
    BOOL pm=NO;

    if (secs) {
        hr = secs / hrInSecs;
        if (hr > 12) {
            *hour = (hr -= 12);
            pm = YES;
        } else {
            pm = NO;
            if (hr == 0)
                hr = 12;
            *hour = hr;
        }
        *minute = ((secs%hrInSecs) / minInSecs);
    }
    return pm;
}
```

1. This expression, as well as others in these two methods, uses the **enum** constants for time-values-as seconds that you defined earlier.
2. The **ConvertSecondsToTime()** function uses indirection as a means for returning multiple values and directly returns a Boolean to indicate AM or PM.