XcodeによるiPhone/Mac アプリの制作 (4)

Core Data & Table View programming

大切な日付を記録するアプリの制作
create Navigation-based Application

株式会社あいはら 山田 泰司 <taiji@aihara.co.jp>

iPhone シミュレータの言語環境、書式、カレンダーの設定

におけるナビゲーションバーの役割とテーブルビュー

- 上図で矢印の起点をタップしていって、
  言語環境を日本語、書式を日本、カレンダーを和暦にする

大切な日付を記録するアプリ: MemoDates の設計

- 大切な日付の一覧を表示し、
  選択された項目の編集や削除、新規の項目の追加、
  そしてそれらの保存をするアプリ

こういったものを作っていく ☞

iPhone におけるユーザデータの編集、削除、 追加、 保存の方法を示すサンプル

1. 大切な日付は
   タイトル、カテゴリ名、日付、通知フラグ
   からなる属性の集まり。但し、通知フラグは
   現時点では機能的には未使用とする。
   それに、Core Data フレームワークを使う。

2. 大切な日付の編集や削除、追加などの作業の流れ
   についてはナビゲーションバーを使う。

3. 一覧や項目の表示にはテーブルビューを使う。

4. 個々の属性の編集にはそれぞれのビューを使う。

5. 編集時のアンドゥ・リドゥを可能にする。

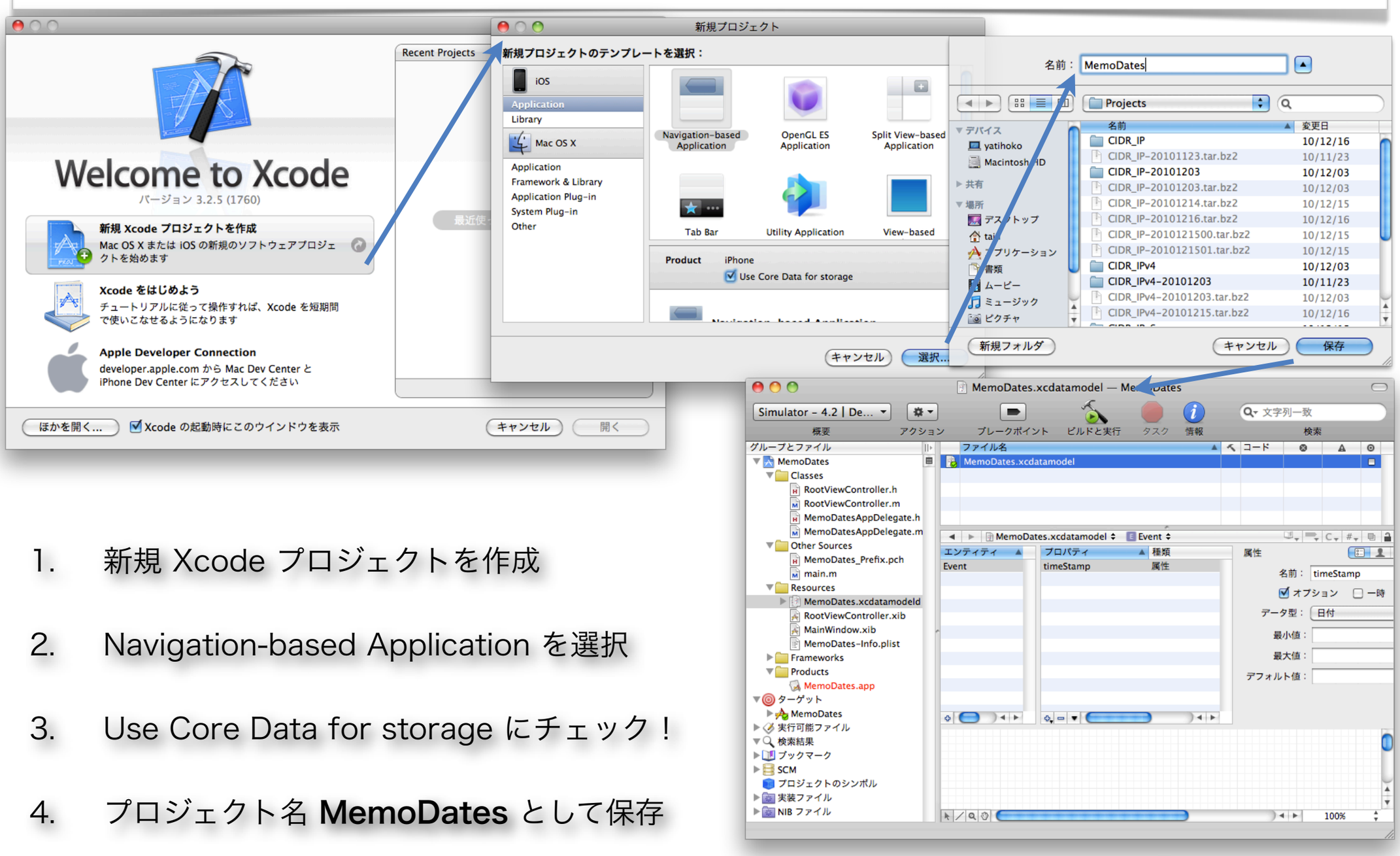
＊今回はコード量が多いですが(P.28〜45)、それは Navigation-based App. + Table View の約束事を細かく書く必要があるせいで、やっている事は簡単。まずは、Core Data とアンドゥマネージャの簡単さと素晴らしさを実感しましょう。

大切な日付を記録するアプリ:
MemoDates の設計

- ナビゲーションバーの役割と
それぞれのビューコントローラー
の関係を整理

MemoDates の作成(1)

create Navigation-based Application + Core Data



1. 新規 Xcode プロジェクトを作成
2. Navigation-based Application を選択
3. Use Core Data for storage にチェック！
4. プロジェクト名 **MemoDates** として保存

MemoDates.xcdatamodeld/**MemoDates.xcdatamodel** の
エンティティ：Event
プロパティ：timeStamp/**データ型「日付」**
に注目！

MemoDates の作成(2)

Navigation-based Application + Core Data テンプレートの動作確認

1. テンプレートのデータモデルのエンティティ Event のプロパティは日付型の timestamp のみ

2. ＋ボタンを押すと、その時刻を timestampキー値とするデータが即座に追加される。

3. アプリケーションを再度起動しても、データが残っている。つまり、保存はされている。

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(0.0)

まずは、**比較元の Core Data なしで保存した MemoDates-CoreData/** を概観

$ less MemoDates-CoreData/Classes/*.[hm]

```
//  MemoDatesAppDelegate.h
#import <UIKit/UIKit.h>

@interface MemoDatesAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UINavigationController *navigationController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;                            // ウインドウ
@property (nonatomic, retain) IBOutlet UINavigationController *navigationController;  // ナビゲーションコントローラ
@end

//  MemoDatesAppDelegate.m
#import "MemoDatesAppDelegate.h"
#import "RootViewController.h"

@implementation MemoDatesAppDelegate
@synthesize window;
@synthesize navigationController;

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {   // アプリが起動したとき
    [self.window addSubview:navigationController.view];     // ナビゲーションコントローラのビューをウインドウに追加
    [self.window makeKeyAndVisible];
    return YES;
}
- (void)applicationWillResignActive:(UIApplication *)application {        // アプリがアクティブ状態を譲るとき
}
- (void)applicationDidEnterBackground:(UIApplication *)application {      // アプリがバックグラウンド状態に遷移したとき
}
- (void)applicationWillEnterForeground:(UIApplication *)application {     // アプリがバックグラウンド状態から復帰するとき
}
- (void)applicationDidBecomeActive:(UIApplication *)application {         // アプリがアクティブ状態に移行したとき
}
- (void)applicationWillTerminate:(UIApplication *)application {           // アプリが終了するとき
}
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application {// アプリがメモリ警告を受けたとき
}
- (void)dealloc {
    [navigationController release];     // ナビゲーションコントローラの参照カウンタを下げる（retain している IBOutlet）
    [window release];                   // ウインドウの参照カウンタを下げる（retain している IBOutlet）
    [super dealloc];
}
@end
```

- ちなみに、Core Data なしで保存した Navigation-based App. を動かしてみると、テンプレートなので「空の」テーブルビューが動作する ☞

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(0.1)

まずは、**比較元の Core Data なしで保存した MemoDates-CoreData/** を概観

$ less MemoDates-CoreData/Classes/*.[hm]

```
//  RootViewController.h
#import <UIKit/UIKit.h>

@interface RootViewController : UITableViewController {     // RootViewController はテーブルビューコントローラのサブクラス
}
@end

//  RootViewController.m
#import "RootViewController.h"

@implementation RootViewController
/*
- (void)viewDidLoad {                              // ビューのメモリへの読み込みが完了したとき
    [super viewDidLoad];
}
*/
/*
- (void)viewWillAppear:(BOOL)animated {            // ビューが表示されようとしているとき
    [super viewWillAppear:animated];
}
*/
/*
- (void)viewDidAppear:(BOOL)animated {             // ビューが表示されたとき
    [super viewDidAppear:animated];
}
*/
/*
- (void)viewWillDisappear:(BOOL)animated {         // ビューが非表示になろうとするとき
    [super viewWillDisappear:animated];
}
*/
/*
- (void)viewDidDisappear:(BOOL)animated {          // ビューが非表示になったとき
    [super viewDidDisappear:animated];
}
*/
/*
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {   // インターフェース方位に自動回転すべきか否か
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {                             // テーブルビューのセクション数
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {     // テーブルビューのセクションにおける行数
    return 0;
}
```

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(0.2)

まずは、**比較元の Core Data なしで保存した MemoDates-CoreData/** を概観

$ less MemoDates-CoreData/Classes/*.[hm]

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {   // 行のテーブルビューセルを返すメソッド
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
    }

          // Configure the cell.                                                          // テーブルビューセルの様相を記述せよ

    return cell;
}
/*
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return YES;
}
*/
/*                                                                                      // 行（セル）の削除または挿入の手続きを行うメソッド
- (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath {
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source.
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationFade];
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert) {
        // Create a new instance of the appropriate class, insert it into the array, and add a new row to the table view.
    }
}
*/
/*                                                                                      // 行（セル）の移動の手続きを行うメソッド
- (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)fromIndexPath toIndexPath:(NSIndexPath *)toIndexPath {
}
*/
/*                                                                                      // 行（セル）の移動が可能かどうかを返答するメソッド
- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *)indexPath {
    return YES;
}
*/
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {       // 行の選択の手続きを行うメソッド
    /*
     <#DetailViewController#> *detailViewController = [[<#DetailViewController#> alloc] initWithNibName:@"<#Nib name#>" bundle:nil];
     // ...
     // Pass the selected object to the new view controller.
     [self.navigationController pushViewController:detailViewController animated:YES];
     [detailViewController release];
    */                                  // コメント中で、詳細ビューコントローラを作成、初期化、そして、ナビゲーションコントローラにプッシュする方法が例示されている。
}
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}
- (void)viewDidUnload {
}
- (void)dealloc {
    [super dealloc];
}
@end
```

// RootViewController.m の続き

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(1)

そして、**Core Data なしで保存した MemoDates-CoreData/** と **Core Data ありで保存した MemoDates+CoreData/** を比較

$ diff -wBbE -ru -U 7 MemoDates-CoreData/ MemoDates+CoreData/ | less

```
--- MemoDates-CoreData/Classes/MemoDatesAppDelegate.h
+++ MemoDates+CoreData/Classes/MemoDatesAppDelegate.h

 #import <UIKit/UIKit.h>
+#import <CoreData/CoreData.h>                                  // Core Data フレームワークを使う

 @interface MemoDatesAppDelegate : NSObject <UIApplicationDelegate> {

     UIWindow *window;
     UINavigationController *navigationController;

+@private                                                       // プライベートのインスタンス変数として、
+    NSManagedObjectContext *managedObjectContext_;            // 管理オブジェクトコンテキスト、
+    NSManagedObjectModel *managedObjectModel_;                // 管理オブジェクトモデル、
+    NSPersistentStoreCoordinator *persistentStoreCoordinator_; // 永続ストアコーディネータ、が追加
 }

 @property (nonatomic, retain) IBOutlet UIWindow *window;
 @property (nonatomic, retain) IBOutlet UINavigationController *navigationController;

+// 管理オブジェクトコンテキスト、管理オブジェクトモデル、永続ストアコーディネータは読み込み専用で retain されたプロパティ
+@property (nonatomic, retain, readonly) NSManagedObjectContext *managedObjectContext;
+@property (nonatomic, retain, readonly) NSManagedObjectModel *managedObjectModel;
+@property (nonatomic, retain, readonly) NSPersistentStoreCoordinator *persistentStoreCoordinator;
+
+- (NSURL *)applicationDocumentsDirectory;                      // アプリのドキュメントのディレクトリ(フォルダ)を返すメソッド
+- (void)saveContext;                                           // コンテキストの保存のためのメソッド

 @end
```

＊差分内容は diff -u (Unified diff.形式)「行頭"-"が削除行、"+"が追加行」「 行頭"---"が比較元ファイル名、"+++"が比較先ファイル名」で表記

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(2)

そして、**Core Data なしで保存した MemoDates-CoreData/** と **Core Data ありで保存した MemoDates+CoreData/** を比較

$ diff -wBbE -ru -U 7 MemoDates-CoreData/ MemoDates+CoreData/ | less

```
--- MemoDates-CoreData/Classes/MemoDatesAppDelegate.m
+++ MemoDates+CoreData/Classes/MemoDatesAppDelegate.m

+- (void)awakeFromNib {                    // UI読み込み後、最初のビューコントローラに自身の管理オブジェクトコンテキストを指定
+    RootViewController *rootViewController = (RootViewController *)[navigationController topViewController];
+    rootViewController.managedObjectContext = self.managedObjectContext;
+}           // アプリケーションデリゲートにおいて、そのナビゲーションコントローラ経由で最初のビューコントローラを取得している

 - (void)applicationDidEnterBackground:(UIApplication *)application {
+    [self saveContext];                            // アプリがバックグラウンド処理へ移行した時に、コンテキストを保存
 }

 - (void)applicationWillTerminate:(UIApplication *)application {
+    [self saveContext];                            // アプリが終了しようとする時に、コンテキストを保存
}

+- (void)saveContext {
+    NSError *error = nil;
+    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
+    if (managedObjectContext != nil) {                                  // 管理オブジェクトコンテキストを取得できたら、
+        if ([managedObjectContext hasChanges] && ![managedObjectContext save:&error]) { // 変更が無ければ保存
+            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+            abort();
+        }
+    }
+}
```

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(3)

```
+- (NSManagedObjectContext *)managedObjectContext {                // 管理オブジェクトコンテキストのゲッターメソッドにて、その初期作成と返値
+    if (managedObjectContext_ != nil) {
+        return managedObjectContext_;                              // 既に管理オブジェクトコンテキストを保持してたら、それを返値
+    }                                                              // さもなくば、
+    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];  // 自身の永続ストアコーディネータを取得して、
+    if (coordinator != nil) {
+        managedObjectContext_ = [[NSManagedObjectContext alloc] init];     // 管理オブジェクトコンテキストを作成、初期化して、
+        [managedObjectContext_ setPersistentStoreCoordinator:coordinator]; // 自身の永続ストアコーディネータを管理オブジェクトコンテキストに設定し、
+    }
+    return managedObjectContext_;                                  // 管理オブジェクトコンテキストを返値
+}

+- (NSManagedObjectModel *)managedObjectModel {                    // 管理オブジェクトモデルのゲッターメソッドにて、その初期作成と返値
+    if (managedObjectModel_ != nil) {
+        return managedObjectModel_;                                // 既に管理オブジェクトモデルを保持してたら、それを返値
+    }                                                              // さもなくば、
+    NSString *modelPath = [[NSBundle mainBundle] pathForResource:@"MemoDates" ofType:@"momd"];       // 主バンドルのリソースにある MemoDates.momd から
+    NSURL *modelURL = [NSURL fileURLWithPath:modelPath];
+    managedObjectModel_ = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];    // 管理オブジェクトモデルを作成、初期化して、
+    return managedObjectModel_;                                    // 管理オブジェクトモデルを返値
+}

+- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {     // 永続ストアコーディネータのゲッターメソッドにて、その初期作成と返値
+    if (persistentStoreCoordinator_ != nil) {
+        return persistentStoreCoordinator_;                        // 既に永続ストアコーディネータを保持していたら、それを返値
+    }                                                              // さもなくば、
+    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:@"MemoDates.sqlite"]; // アプリのドキュメントのディレクトリの
+                                                                                                              // MemoDates.sqlite と
+    NSError *error = nil;
+    persistentStoreCoordinator_ = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self managedObjectModel]];
+                                          // 自身の管理オブジェクトモデルで作成、初期化した永続ストアコーディネータとで、適切な永続ストアコーディネータを作成
+    if (![persistentStoreCoordinator_ addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:nil error:&error]) {
+        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+        abort();
+    }
+    return persistentStoreCoordinator_;                            // 永続ストアコーディネータを返値
+}

+- (NSURL *)applicationDocumentsDirectory {                         // アプリのドキュメントのディレクトリ(フォルダ)を返すメソッド
+    return [[[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask] lastObject];
+}

 - (void)dealloc {
+    [managedObjectContext_ release];               // 管理オブジェクトコンテキストの参照カウンタを下げる
+    [managedObjectModel_ release];                 // 管理オブジェクトモデルの参照カウンタを下げる
+    [persistentStoreCoordinator_ release];         // 永続ストアコーディネータの参照カウンタを下げる
     [navigationController release];
     [window release];
     [super dealloc];
 }
 @end
```

--- MemoDates-CoreData/Classes/MemoDatesAppDelegate.m
+++ MemoDates+CoreData/Classes/MemoDatesAppDelegate.m の続き

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(4)

そして、**Core Data なしで保存した MemoDates-CoreData/** と **Core Data ありで保存した MemoDates+CoreData/** を比較

$ diff -wBbE -ru -U 7 MemoDates-CoreData/ MemoDates+CoreData/ | less

```
--- MemoDates-CoreData/Classes/RootViewController.h
+++ MemoDates+CoreData/Classes/RootViewController.h

 #import <UIKit/UIKit.h>
+#import <CoreData/CoreData.h>                                  // Core Data フレームワークを使う

-@interface RootViewController : UITableViewController {
+@interface RootViewController : UITableViewController <NSFetchedResultsControllerDelegate> {
+                                                               // フェッチ結果コントローラデリゲート形式プロトコルに準拠
+@private                                                       // プライベートのインスタンス変数として、
+    NSFetchedResultsController *fetchedResultsController_;     // フェッチ結果コントローラ、
+    NSManagedObjectContext *managedObjectContext_;             // 管理オブジェクトコンテキスト、が追加
 }
+// 管理オブジェクトコンテキスト、フェッチ結果コントローラは retain されたプロパティ
+@property (nonatomic, retain) NSManagedObjectContext *managedObjectContext;
+@property (nonatomic, retain) NSFetchedResultsController *fetchedResultsController;

 @end
```

<フェッチ結果コントローラデリゲート>の必須のインスタンスメソッド：

controllerWillChangeContent: // 以下の更新の開始を指示

// オブジェクトの追加、削除、更新、移動

controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:

// セクションの追加、削除

controller:didChangeSection:atIndex:forChangeType:

controllerDidChangeContent: // 以上の更新の終了を指示

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(5)

そして、**Core Data なしで保存した MemoDates-CoreData/** と **Core Data ありで保存した MemoDates+CoreData/** を比較

```
--- MemoDates-CoreData/Classes/RootViewController.m
+++ MemoDates+CoreData/Classes/RootViewController.m

+@interface RootViewController ()                                                    // このクラスで、
+- (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath; // このメソッドは絶対に定義すべし、という匿名のカテゴリを用いた宣言
+@end

 @implementation RootViewController

+                                          // アクセサメソッドの定義において、フェッチ結果コントローラ、管理オブジェクトオブジェクトコンテキストのインスタンス変数の明示
+@synthesize fetchedResultsController=fetchedResultsController_, managedObjectContext=managedObjectContext_;

-/*                                                                                  // このメソッドがアンコメントされ、
 - (void)viewDidLoad {
     [super viewDidLoad];
+    self.navigationItem.leftBarButtonItem = self.editButtonItem;                    // ナビゲーションの左のバーボタンに、 自身の編集ボタンを設定
+    UIBarButtonItem *addButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
+                                                                                        target:self action:@selector(insertNewObject)];
+                                                                                    // バーボタンの作成、初期化、ターゲット(self)＆アクション(insertNewObject)の設定
+    self.navigationItem.rightBarButtonItem = addButton;                             // ナビゲーションの右のバーボタンは、このバーボタンを設定
+    [addButton release];                                                            // したので、その参照カウンタを下げる
 }
-*/

-/*                                                                                  // このメソッドがアンコメントされている
 - (void)viewWillAppear:(BOOL)animated {
     [super viewWillAppear:animated];
 }
-*/

+- (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath {        // テーブルビューセルの様相を決定する必須とされたメソッドの定義
+    NSManagedObject *managedObject = [self.fetchedResultsController objectAtIndexPath:indexPath]; // フェッチ結果コントローラ等から得られる管理オブジェクト
+    cell.textLabel.text = [[managedObject valueForKey:@"timeStamp"] description];   // から得られる timestampキー値を印字可能にしてセルのタイトルに設定
+}

+- (void)insertNewObject {
+    // Create a new instance of the entity managed by the fetched results controller.
+    NSManagedObjectContext *context = [self.fetchedResultsController managedObjectContext]; // フェッチ結果コントローラの管理オブジェクトコンテキストと
+    NSEntityDescription *entity = [[self.fetchedResultsController fetchRequest] entity];    // フェッチ結果コントローラのフェッチ要求のエンティティで
+                                                                                    // その管理オブジェクトコンテキスト上でエンティティ名による新規の管理オブジェクトを挿入、それを得る
+    NSManagedObject *newManagedObject = [NSEntityDescription insertNewObjectForEntityForName:[entity name] inManagedObjectContext:context];
+
+    [newManagedObject setValue:[NSDate date] forKey:@"timeStamp"]; // 新規の管理オブジェクトの timestampキー値を [NSDate date], 現在の時刻に設定
+
+    NSError *error = nil;
+    if (![context save:&error]) {                                                                   // 追加したので、保存を試みている。
+        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+        abort();
+    }
+}

+- (void)setEditing:(BOOL)editing animated:(BOOL)animated {                          // 編集モードか否かの設定の為のメソッド
+    [super setEditing:(BOOL)editing animated:(BOOL)animated];                       // 親クラスのメソッド通りに行い、加えて、
+    self.navigationItem.rightBarButtonItem.enabled = !editing;                      // ナビゲーションの右のバーボタンが有効か否かは、非編集モードか否かとする
+}
```

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(6)

```
 - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
-    return 1;                                                          // テーブルビューのセクション数は、
+    return [[self.fetchedResultsController sections] count];           // フェッチ結果コントローラのセクション数としている。
 }

 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
-    return 0;                                                          // テーブルビューのセクションにおける行数は、
+    id <NSFetchedResultsSectionInfo> sectionInfo = [[self.fetchedResultsController sections] objectAtIndex:section];
+    return [sectionInfo numberOfObjects];                              // フェッチ結果コントローラのセクションのオブジェクト数としている。
 }

 - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {     // 行のテーブルビューセルを返すメソッド
     static NSString *CellIdentifier = @"Cell";
     UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
     if (cell == nil) {
         cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
     }
        // Configure the cell.
+    [self configureCell:cell atIndexPath:indexPath];                   // テーブルビューセルの様相を決定する必須とされたメソッドの呼び出し
     return cell;
 }

-/*                                                                     // このメソッドがアンコメントされ、
 - (void)tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath
{
     if (editingStyle == UITableViewCellEditingStyleDelete) {
-        // Delete the row from the data source.
-        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationFade];
+        // Delete the managed object for the given index path
+        NSManagedObjectContext *context = [self.fetchedResultsController managedObjectContext];
+        [context deleteObject:[self.fetchedResultsController objectAtIndexPath:indexPath]];
+        // Save the context.
+        NSError *error = nil;
+        if (![context save:&error]) {
+            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+            abort();
+        }                          // 行（セル）の削除の手続きを、フェッチ結果コントローラから得た管理オブジェクトを対象に管理オブジェクトコンテキスト経由で行っている。
     }
-    else if (editingStyle == UITableViewCellEditingStyleInsert) {      // 行（セル）の挿入の手続きは消されている。
-        // Create a new instance of the appropriate class, insert it into the array, and add a new row to the table view.
-    }
 }
-*/
-/*                                                                     // この行の移動の手続きを行うメソッドは消されている。
-- (void)tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)fromIndexPath toIndexPath:(NSIndexPath *)toIndexPath {
-}
-*/

-/*                                                                     // このメソッドがアンコメントされ、
 - (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *)indexPath {
-    return YES;
+    return NO;                                                         // 行は移動不可であることを返答している。
 }
-*/
```

--- MemoDates-CoreData/Classes/RootViewController.m
+++ MemoDates+CoreData/Classes/RootViewController.m の続き

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(7)

そして、**Core Data なしで保存した MemoDates-CoreData/** と **Core Data ありで保存した MemoDates+CoreData/** を比較

$ diff -wBbE -ru -U 7 MemoDates-CoreData/ MemoDates+CoreData/ | less

```
 - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
+    // Navigation logic may go here -- for example, create and push another view controller. // 行の選択への応答について、コメント内で、
     /*
      <#DetailViewController#> *detailViewController = [[<#DetailViewController#> alloc] initWithNibName:@"<#Nib name#>" bundle:nil];
+     NSManagedObject *selectedObject = [[self fetchedResultsController] objectAtIndexPath:indexPath];        // フェッチ結果コントローラから選択された
      // ...                                                                                                  // 管理オブジェクトを得る方法の説明
      // Pass the selected object to the new view controller.
      [self.navigationController pushViewController:detailViewController animated:YES];
      [detailViewController release];
     */
 }

+- (NSFetchedResultsController *)fetchedResultsController {          // フェッチ結果コントローラのゲッターメソッドにて、フェッチ結果コントローラの初期作成と返値
+    if (fetchedResultsController_ != nil) {
+        return fetchedResultsController_;                           // 既にフェッチ結果コントローラを保持してたら、それを返値
+    }                                                               // さもなくば、
+    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];   // まずは、フェッチ要求を作成、初期化する。
+    NSEntityDescription *entity = [NSEntityDescription entityForName:@"Event" inManagedObjectContext:self.managedObjectContext];
+
+    [fetchRequest setEntity:entity];                                // フェッチ要求に対象となるエンティティ Event を設定
+    [fetchRequest setFetchBatchSize:20];                            // 推奨のfetchBatchSize、0だと無制限
+
+    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"timeStamp" ascending:NO];     // 次にソート記述子を作成、初期化(キー,降順)
+    NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor, nil];                       // ソート記述子はその配列で指定
+
+    [fetchRequest setSortDescriptors:sortDescriptors];              // フェッチ要求にソート記述子の配列を設定
+
+    // Edit the section name key path and cache name if appropriate.
+    // nil for section name key path means "no sections".
+                                                                    // フェッチ要求、管理オブジェクトコンテキスト、セクション名などからフェッチ結果コントローラを作成、初期化
+    NSFetchedResultsController *aFetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest:fetchRequest
+        managedObjectContext:self.managedObjectContext sectionNameKeyPath:nil cacheName:@"Root"];
+    aFetchedResultsController.delegate = self;                      // フェッチ結果コントローラのデリゲート先は自分自身に
+    self.fetchedResultsController = aFetchedResultsController;     // フェッチ結果コントローラを保持
+
+    [aFetchedResultsController release];
+    [fetchRequest release];
+    [sortDescriptor release];
+    [sortDescriptors release];
+
+    NSError *error = nil;
+    if (![fetchedResultsController_ performFetch:&error]) {
+        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+        abort();
+    }
+
+    return fetchedResultsController_;                               // フェッチ結果コントローラを返値
+}
```

--- MemoDates-CoreData/Classes/MemoDatesAppDelegate.m
+++ MemoDates+CoreData/Classes/MemoDatesAppDelegate.m の続き

[参考資料] Core Data なしの Navigation-based App.テンプレートと Core Data ありの Navigation-based App.テンプレートとの比較から学ぶ(8)

```
+- (void)controllerWillChangeContent:(NSFetchedResultsController *)controller {
+    [self.tableView beginUpdates]; // フェッチ結果コントローラからのオブジェクトの追加、削除、更新、移動やセクションの追加、削除による更新の開始をテーブルビューに指示
+}

+- (void)controller:(NSFetchedResultsController *)controller didChangeSection:(id <NSFetchedResultsSectionInfo>)sectionInfo
+           atIndex:(NSUInteger)sectionIndex forChangeType:(NSFetchedResultsChangeType)type { // フェッチ結果コントローラからのセクションの追加、削除を
+    switch(type) {                                                                          // テーブルビューに反映
+        case NSFetchedResultsChangeInsert:
+            [self.tableView insertSections:[NSIndexSet indexSetWithIndex:sectionIndex] withRowAnimation:UITableViewRowAnimationFade];
+            break;
+        case NSFetchedResultsChangeDelete:
+            [self.tableView deleteSections:[NSIndexSet indexSetWithIndex:sectionIndex] withRowAnimation:UITableViewRowAnimationFade];
+            break;
+    }
+}

+- (void)controller:(NSFetchedResultsController *)controller didChangeObject:(id)anObject
+       atIndexPath:(NSIndexPath *)indexPath forChangeType:(NSFetchedResultsChangeType)type
+      newIndexPath:(NSIndexPath *)newIndexPath {                                        // フェッチ結果コントローラからのオブジェクトの追加、削除、更新、移動を
+    UITableView *tableView = self.tableView;
+
+    switch(type) {                                                                      // テーブルビューに反映
+        case NSFetchedResultsChangeInsert:
+            [tableView insertRowsAtIndexPaths:[NSArray arrayWithObject:newIndexPath] withRowAnimation:UITableViewRowAnimationFade];
+            break;
+        case NSFetchedResultsChangeDelete:
+            [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationFade];
+            break;
+        case NSFetchedResultsChangeUpdate:
+            [self configureCell:[tableView cellForRowAtIndexPath:indexPath] atIndexPath:indexPath];
+            break;
+        case NSFetchedResultsChangeMove:
+            [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationFade];
+            [tableView insertRowsAtIndexPaths:[NSArray arrayWithObject:newIndexPath]withRowAnimation:UITableViewRowAnimationFade];
+            break;
+    }
+}

+- (void)controllerDidChangeContent:(NSFetchedResultsController *)controller {
+    [self.tableView endUpdates]; // フェッチ結果コントローラからのオブジェクトの追加、削除、更新、移動やセクションの追加、削除による更新の終了をテーブルビューに指示
+}

 - (void)viewDidUnload {
     // Relinquish ownership of anything that can be recreated in viewDidLoad or on demand.
     // For example: self.myOutlet = nil;
 }

 - (void)dealloc {
+    [fetchedResultsController_ release]; // フェッチ結果コントローラの参照カウンタを下げる
+    [managedObjectContext_ release];     // 管理オブジェクトコンテキストの参照カウンタを下げる
     [super dealloc];
 }

 @end
```

--- MemoDates-CoreData/Classes/MemoDatesAppDelegate.m
+++ MemoDates+CoreData/Classes/MemoDatesAppDelegate.m の続き

＜フェッチ結果コントローラデリゲート＞の必須のインスタンスメソッド：

controllerWillChangeContent: // 以下の更新の開始を指示

// オブジェクトの追加、削除、更新、移動

controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:

// セクションの追加、削除

controller:didChangeSection:atIndex:forChangeType:

controllerDidChangeContent: // 以上の更新の終了を指示

Core Data なしの Navigation-based App.テンプレート と
Core Data ありの Navigation-based App.テンプレート との比較から学び取れた

Core Data とテーブルビューコントローラにおける要求と応答、処理のフロー

フェッチ要求

エンティティ記述：Event、self.managedObjectContext
ソート記述子の配列：sortDescriptor(日付の降順)
管理オブジェクトコンテキスト：self.managedObjectContext

P.16 fetchedResultsController

フェッチ結果コントローラ

フェッチ要求：fetchRequest
管理オブジェクトコンテキスト：self.managedObjectContext
セクション名：なし

P.16 fetchedResultsController

追加・削除

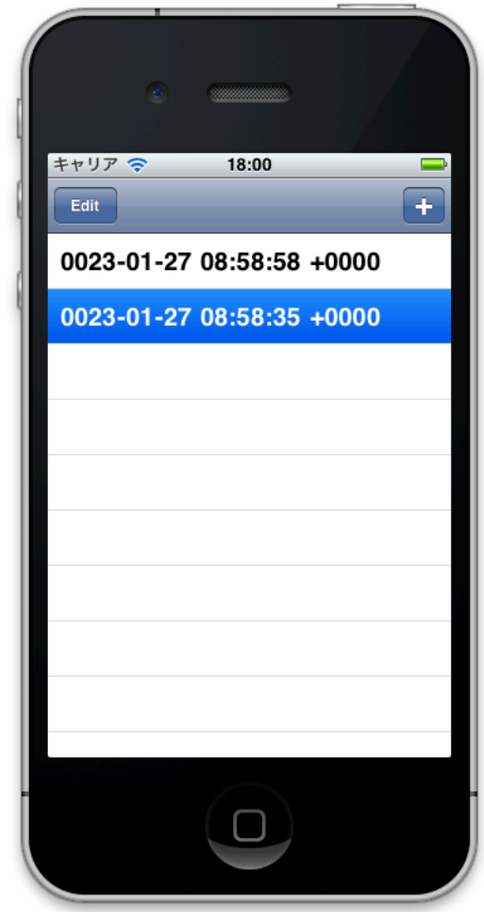
エンティティ記述：Event、NSManagedObjectContext *(追加)
管理オブジェクト：newManagedObject(追加)
管理オブジェクト：self.fetchedResultsController経由(削除)

P.14 insertNewObject, P.15 tableView:commitEditingStyle

保存

管理オブジェクトコンテキスト：NSManagedObjectContext *

P.11, P.14 insertNewObject



管理オブジェクトコンテキスト

永続ストアコーディネータ：self.persistentStoreCoordinator

P.12 managedObjectContext

永続ストアコーディネータ

ストアURL：Documents/MemoDates.sqlite
管理オブジェクトモデル：self.managedObjectModel

P.12 persistentStoreCoordinator

管理オブジェクトモデル

モデルURL：MemoDates.momd/MemoDates.mom
　エンティティ：Event
　　プロパティ：timeStamp/データ型「日付」

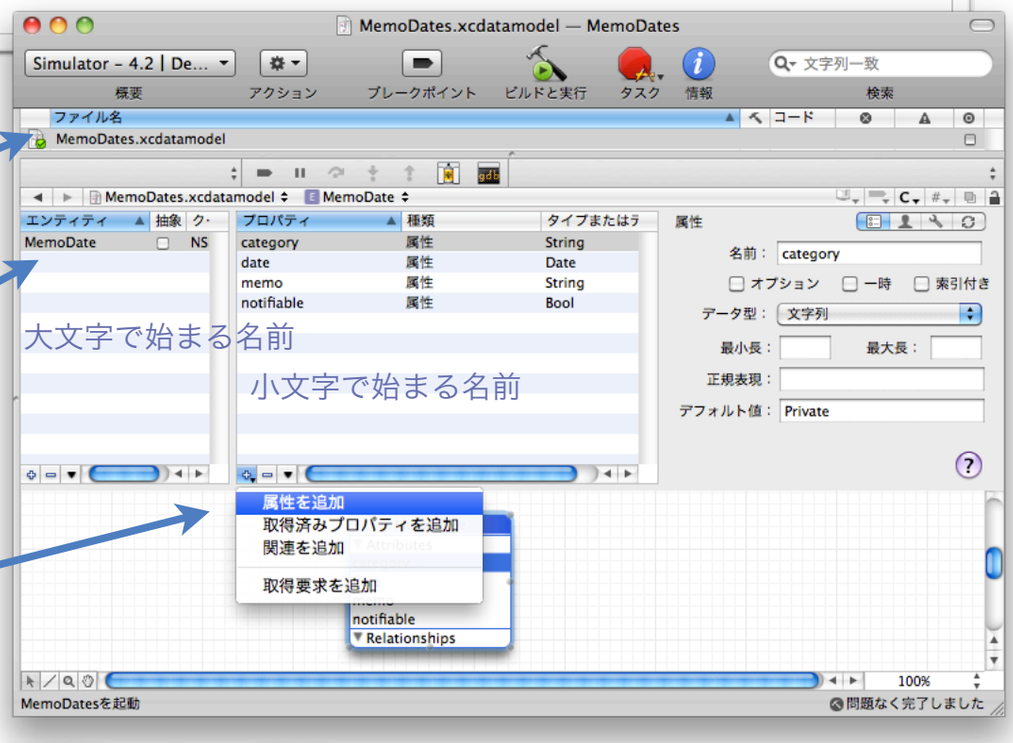
P.12 managedObjectModel

永続ストア

エンティティ：
　オブジェクト：timeStamp/2011-01-27.08:58:58+0000
　オブジェクト：timeStamp/2011-01-27.08:58:35+0000
　　：
　　：
ストア(シミュレータの場合)：
　Library/Application Support/iPhone Simulator/4.2/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents/MemoDates.sqlite

MemoDates の作成(3)

MemoDates.xcdatamodel のモデルブラウザによる編集



1. MemoDates.xcdatamodeld/MemoDates.xcdatamodel を選んで、モデルブラウザを開く

2. エンティティ名を MemoDate に

3. プロパティとして以下の属性を「＋」ボタンのメニューから追加

   (1) プロパティ名：memo、データ型：文字列、オプション：なし

   (2) プロパティ名：category、データ型：文字列、オプション：なし、デフォルト値：Private

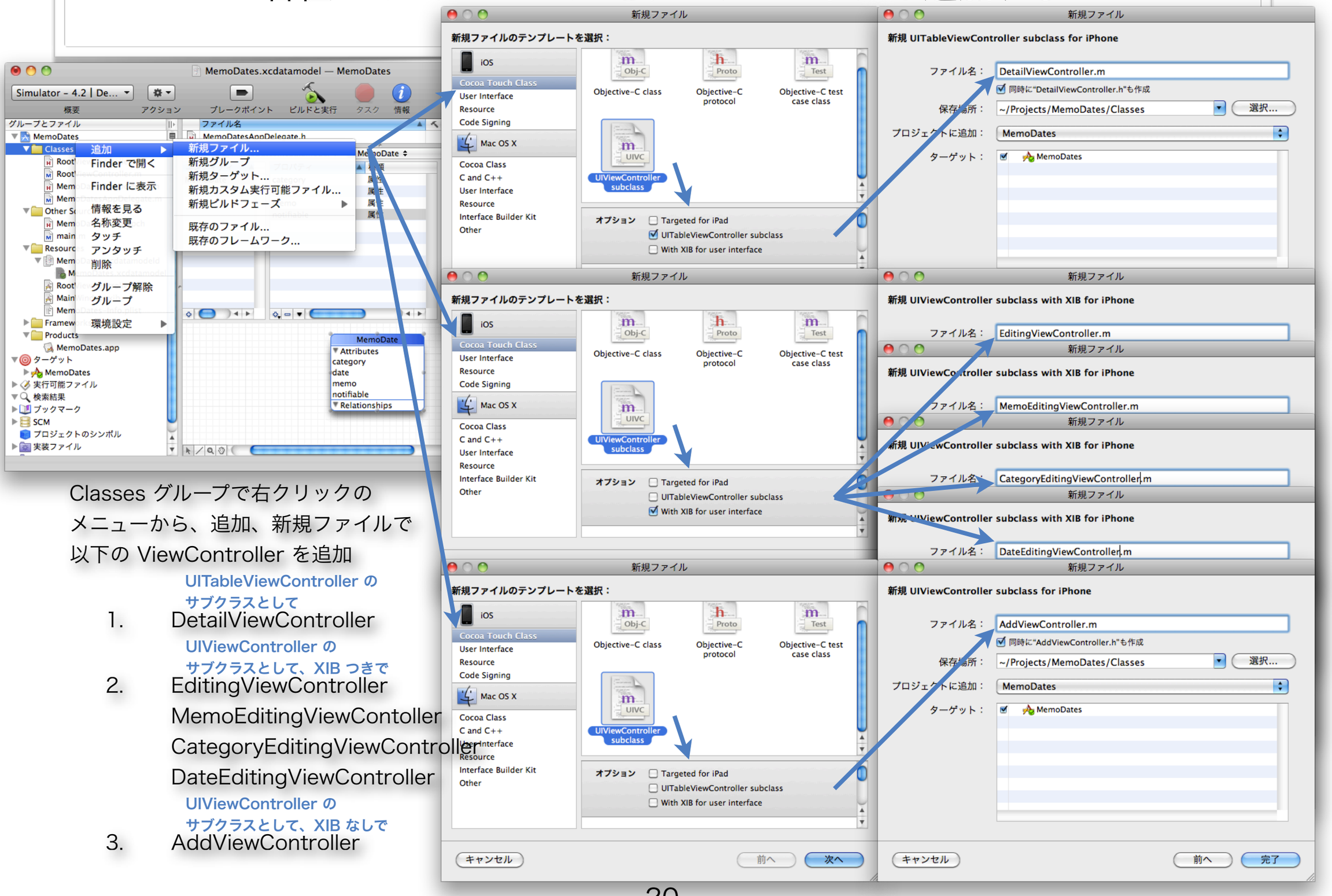
   (3) プロパティ名：date、データ型：日付、オプション：なし

   (4) プロパティ名：notifiable、データ型：真偽値、オプション：なし、デフォルト値：偽

4. ⌘S で保存

MemoDates の作成(4)

各種 ViewController をテンプレートから追加する

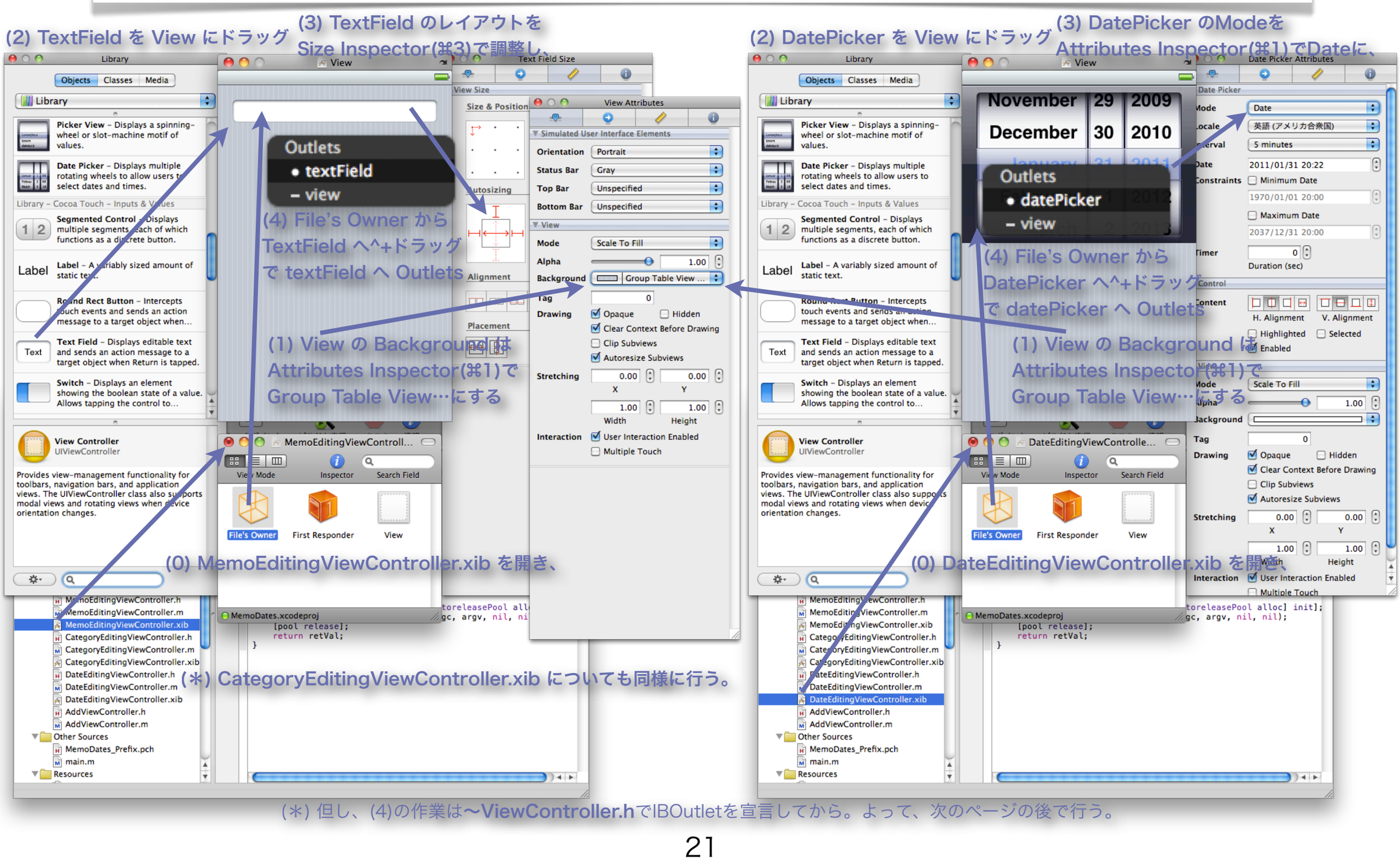


Classes グループで右クリックの
メニューから、追加、新規ファイルで
以下の ViewController を追加

1. DetailViewController
   UITableViewController の
   サブクラスとして
2. EditingViewController
   MemoEditingViewContoller
   CategoryEditingViewController
   DateEditingViewController
   UIViewController の
   サブクラスとして、XIB つきで
3. AddViewController
   UIViewController の
   サブクラスとして、XIB なしで

MemoDates の作成(5)

～EditingViewController.xib の Interface Builder による編集



(＊) 但し、(4)の作業は～**ViewController.h**でIBOutletを宣言してから。よって、次のページの後で行う。

MemoDates の作成(6)

～.[hm] の編集･･･なのだが大変なので **patch コマンド**で一気に編集、ついでに MemoDate-Info.plist も

ターミナル — bash — 80×24

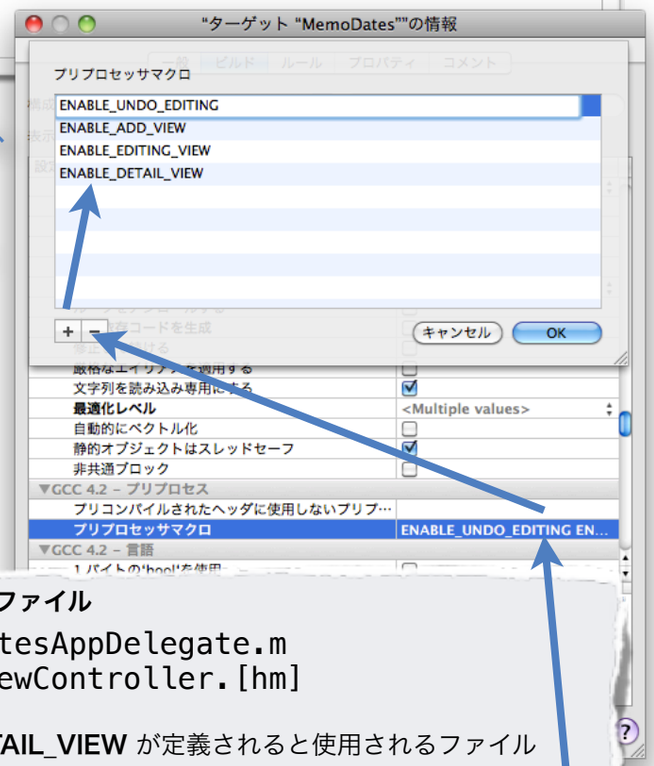
```
~/Projects/MemoDates$ patch -p1 -b -z.org < ../MemoDates.patch
```

編集されたファイル

```
Classes/MemoDatesAppDelegate.m
Classes/RootViewController.h
Classes/RootViewController.m
Classes/DetailViewController.h
Classes/DetailViewController.m
Classes/EditingViewController.h
Classes/EditingViewController.m
Classes/MemoEditingViewController.h
Classes/MemoEditingViewController.m
Classes/CategoryEditingViewController.h
Classes/CategoryEditingViewController.m
Classes/DateEditingViewController.h
Classes/DateEditingViewController.m
Classes/AddViewController.h
Classes/AddViewController.m
MemoDates-Info.plist
```

MemoDates.patch ファイルをProjects ディレクトリに取得し、
ターミナルで
$ cd ~/Projects/MemoDates
でカレントディレクトリを MemoDates ディレクトリに移行し、
このように patch コマンドを実行する

完成！…というのでは唐突すぎるので、下記のように、
(プリプロセッサ)マクロをいくつか定義していくと、
段階的に機能が追加されていくようになっています



マクロ **ENABLE_DETAIL_VIEW** が定義されると挙動が変わるファイル

```
Classes/MemoDatesAppDelegate.m
Classes/RootViewController.m
```

マクロ **ENABLE_EDITING_VIEW** が定義されると挙動が変わるファイル

```
Classes/DetailViewController.m
```

マクロ **ENABLE_ADD_VIEW** が定義されると挙動が変わるファイル

```
Classes/RootViewController.[hm]
```

マクロ **ENABLE_UNDO_EDITING** が定義されると挙動が変わるファイル

```
Classes/DetailViewController.[hm]
Classes/MemoEditingViewController.m
Classes/CategoryEditingViewController.m
Classes/DateEditingViewController.m
```

右のファイル→
の追加にあたり、
←左のファイル
の修正を行う事と等価

はじめから使用されるファイル

```
Classes/MemoDatesAppDelegate.m
Classes/RootViewController.[hm]
```

マクロ **ENABLE_DETAIL_VIEW** が定義されると使用されるファイル

```
Classes/DetailViewController.[hm]
```

マクロ **ENABLE_EDITING_VIEW** が定義されると使用されるファイル

```
Classes/EditingViewController.[hm]
Classes/MemoEditingViewController.[hm]
Classes/CategoryEditingViewController.[hm]
Classes/DateEditingViewController.[hm]
```

マクロ **ENABLE_ADD_VIEW** が定義されると使用されるファイル

```
Classes/AddViewController.[hm]
```

(プリプロセッサ)マクロを定義するには、
ターゲット MemoDates の情報(⌘I)のビルドタブにて、
GCC - プリプロセスのプリプロセッサマクロをダブルクリックで編集、
+ボタンで追加して、定義したいものを入力

MemoDates の作成(7)

⌘R で RootViewController の挙動を確認

MemoDates の作成(8)

マクロ **ENABLE_DETAIL_VIEW** を定義したのち

⌘R で RootViewController と DetailViewController の挙動を確認

MemoDates の作成(9)

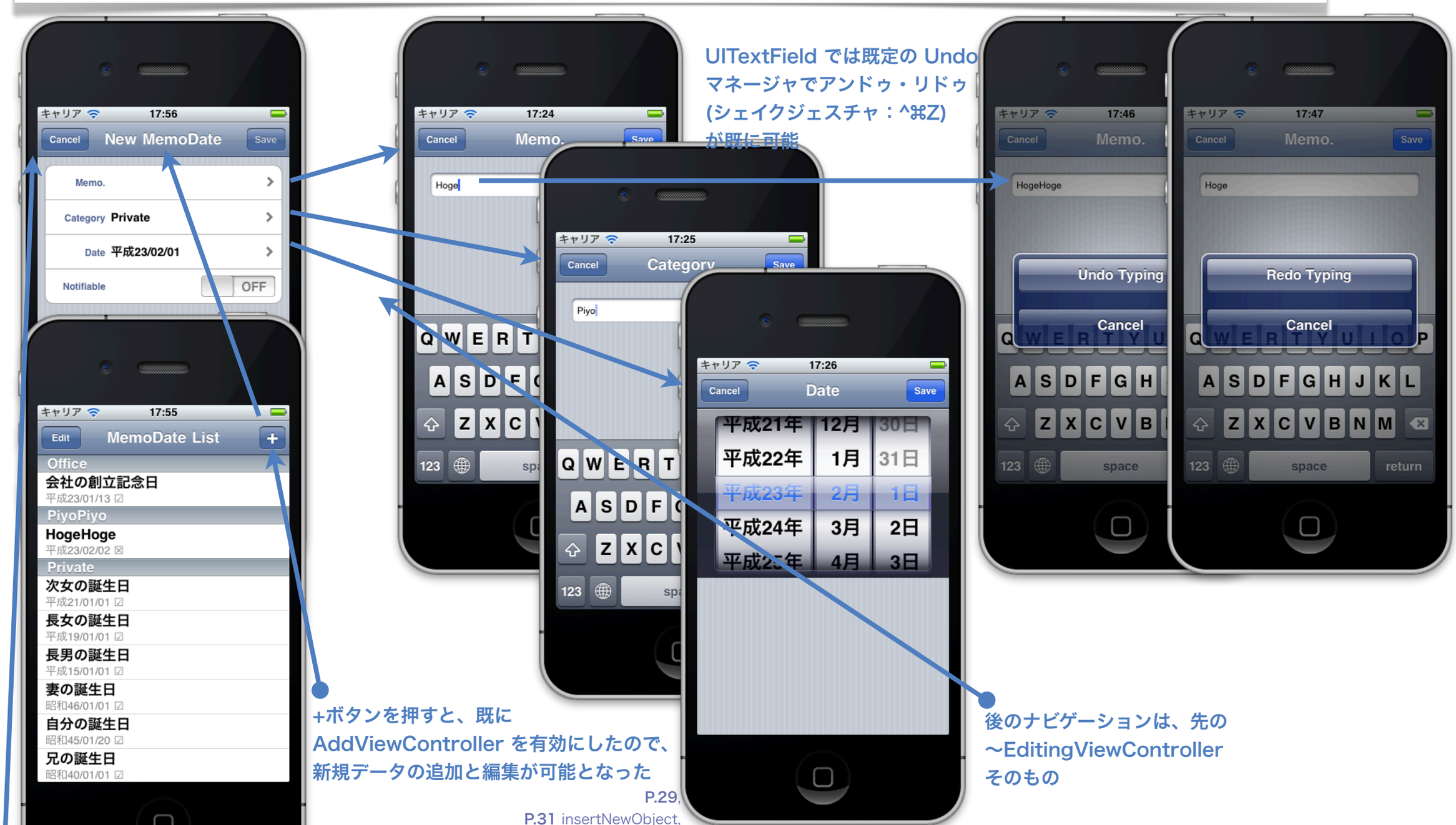
マクロ **ENABLE_EDITING_VIEW** を定義したのち

⌘R で DetailViewController と ～EditingViewController の挙動を確認

MemoDates の作成(10)

マクロ **ENABLE_ADD_VIEW** を定義したのち

⌘R で RootViewController と AddViewController の挙動を確認



UITextField では既定の Undo マネージャでアンドゥ・リドゥ (シェイクジェスチャ：^⌘Z) が既に可能

ここでアンドゥ・リドゥ (シェイクジェスチャ：^⌘Z) をしても何も起こらないことに留意

+ボタンを押すと、既に AddViewController を有効にしたので、新規データの追加と編集が可能となった

P.29,
P.31 insertNewObject,
P.32 addViewController:didFinishWithSave:,
P.45 cancel:, save:

後のナビゲーションは、先の〜EditingViewController そのもの

MemoDates の作成(11)

マクロ **ENABLE_UNDO_EDITING** を定義したのち

⌘R で DetailViewController と ～EditingViewController の挙動を確認

P.36 setEditing:animated:,
P.38, P.45 viewDidLoad, viewDidUnload

Undo マネージャを有効にしたので
すべての編集に関するアンドゥ・リドゥ
(シェイクジェスチャ：^⌘Z)
が可能になった

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(1)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/MemoDatesAppDelegate.m
+++ MemoDates/Classes/MemoDatesAppDelegate.m
         NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
         abort();
     }
                                                    // ここは、永続ストアコーディネータのゲッターメソッド persistentStoreCoordinator の末尾の方
+#if !defined(ENABLE_DETAIL_VIEW)                   // このマクロが定義されていなかったら有効になる、ダミーデータ群を作成する箇所
+  NSEntityDescription *entity = [NSEntityDescription entityForName:@"MemoDate" inManagedObjectContext:self.managedObjectContext];
+  struct {
+    NSString *memo, *category;
+    NSString *date;
+    BOOL notifiable;
+  } fake_data_set[] = {
+    { @"父の誕生日", nil, @"1935/06/11", YES },
+    { @"母の誕生日", nil, @"1939/10/16", NO },
+    { @"姉の誕生日", nil, @"1961/01/01", YES },
+    { @"兄の誕生日", nil, @"1965/01/01", YES },
+    { @"自分の誕生日", nil, @"1970/05/20", YES },
+    { @"妻の誕生日", nil, @"1971/01/01", YES },
+    { @"長男の誕生日", nil, @"2003/01/01", YES },
+    { @"長女の誕生日", nil, @"2007/01/01", YES },
+    { @"次女の誕生日", nil, @"2009/01/01", YES },
+    { @"会社の創立記念日", @"Office", @"2011/01/13", YES },
+  };
+  NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
+  NSCalendar *calendar = [[NSCalendar alloc] initWithCalendarIdentifier:NSGregorianCalendar];
+  [dateFormatter setCalendar:calendar];
+  [calendar release];
+  [dateFormatter setDateFormat:@"yyyy/MM/DD"];
+  size_t i;
+  for (i=0; i<sizeof(fake_data_set)/sizeof(fake_data_set[0]); i++) {
+    NSManagedObject *newManagedObject = [NSEntityDescription insertNewObjectForEntityForName:[entity name]
+                                         inManagedObjectContext:self.managedObjectContext];          // 新規の管理オブジェクトの追加
+    [newManagedObject setValue:fake_data_set[i].memo forKey:@"memo"];                                // memoキー値の設定
+    if (fake_data_set[i].category) [newManagedObject setValue:fake_data_set[i].category forKey:@"category"];   // categoryキー値の設定
+    [newManagedObject setValue:[dateFormatter dateFromString:fake_data_set[i].date] forKey:@"date"];  // dateキー値の設定
+    [newManagedObject setValue:[NSNumber numberWithBool:fake_data_set[i].notifiable] forKey:@"notifiable"];  // notifiableキー値の設定
+  }
+  [dateFormatter release];
+  [self.managedObjectContext save:NULL];
+#endif
     return persistentStoreCoordinator_;
 }
```

＊差分内容は diff -u (Unified diff.形式)「行頭"-"が削除行、"+"が追加行」「 行頭"---"が比較元ファイル名、"+++"が比較先ファイル名」で表記

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(2)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/RootViewController.h
+++ MemoDates/Classes/RootViewController.h

 #import <UIKit/UIKit.h>
 #import <CoreData/CoreData.h>
+#if defined(ENABLE_ADD_VIEW)                                    // このマクロが定義されていたら、
+#import "AddViewController.h"
+#endif

-@interface RootViewController : UITableViewController <NSFetchedResultsControllerDelegate> {
+@interface RootViewController : UITableViewController <NSFetchedResultsControllerDelegate
+#if defined(ENABLE_ADD_VIEW)                                    // このマクロが定義されていたら、
+, AddViewControllerDelegate                                     // AddViewControllerDelegate 形式プロトコルに準拠
+#endif
+> {
 @private
     NSFetchedResultsController *fetchedResultsController_;
     NSManagedObjectContext *managedObjectContext_;
+#if defined(ENABLE_ADD_VIEW)                                    // このマクロが定義されていたら、
+    NSManagedObjectContext *addingManagedObjectContext_;        // 別の追加用の管理オブジェクトコンテキストを使う
+#endif
 }

+#if defined(ENABLE_ADD_VIEW)          // 追加用の管理オブジェクトコンテキストも retain されたプロパティ
+@property (nonatomic, retain) NSManagedObjectContext *addingManagedObjectContext;
+#endif
 @property (nonatomic, retain) NSManagedObjectContext *managedObjectContext;
 @property (nonatomic, retain) NSFetchedResultsController *fetchedResultsController;

 @end
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(3)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

```
--- MemoDates+CoreData/Classes/RootViewController.m
+++ MemoDates/Classes/RootViewController.m

 #import "RootViewController.h"
+#if defined(ENABLE_DETAIL_VIEW)                                    // このマクロが定義されていたら、
+#import "DetailViewController.h"
+#endif
+#if defined(ENABLE_ADD_VIEW)                                       // このマクロが定義されていたら、
+#import "AddViewController.h"
+#endif

 @interface RootViewController ()
 - (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath;
 @end


 @implementation RootViewController

 @synthesize fetchedResultsController=fetchedResultsController_, managedObjectContext=managedObjectContext_;
+#if defined(ENABLE_ADD_VIEW)                                       // このマクロが定義されていたら、
+@synthesize addingManagedObjectContext=addingManagedObjectContext_;// アクセサメソッドの定義で追加用の管理オブジェクトコンテキストのインスタンス変数を明示
+#endif

 #pragma mark -
 #pragma mark View lifecycle

 - (void)viewDidLoad {
     [super viewDidLoad];
+    self.title = @"MemoDate List";                                 // タイトルの設定
     self.navigationItem.leftBarButtonItem = self.editButtonItem;
     UIBarButtonItem *addButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemAdd target:self
                                                                                action:@selector(insertNewObject)];
     self.navigationItem.rightBarButtonItem = addButton;
     [addButton release];
 }


 // Implement viewWillAppear: to do additional setup before the view is presented.
 - (void)viewWillAppear:(BOOL)animated {
+    [self.tableView reloadData];                                   // 自身のテーブルビューにデータの再読込を指示
     [super viewWillAppear:animated];
 }

 - (void)configureCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath {        // テーブルビューセルの様相を決定する必須とされたメソッド
     NSManagedObject *managedObject = [self.fetchedResultsController objectAtIndexPath:indexPath];
-    cell.textLabel.text = [[managedObject valueForKey:@"timeStamp"] description];   // テンプレートにおける timeStampキー値の代わりに、
+    cell.textLabel.text = [[managedObject valueForKey:@"memo"] description];        // MemoDate における memoキー値をセルのタイトルに表示
+                                                                   // MemoDate における date, notifiableキー値をセルのサブタイトルに表示
+    cell.detailTextLabel.text = [NSString stringWithFormat:@"%@ %@", [NSDateFormatter localizedStringFromDate:[managedObject valueForKey:@"date"]
+                                                              dateStyle:NSDateFormatterMediumStyle timeStyle:NSDateFormatterNoStyle],
+                                [[managedObject valueForKey:@"notifiable"] boolValue] ? @"☑" : @"☒" ];
 }
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(4)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較
$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
 - (void)insertNewObject {
+#if !defined(ENABLE_ADD_VIEW)    // このマクロが定義されていなかったら有効になる、ダミーデータを作成する箇所
     // Create a new instance of the entity managed by the fetched results controller.
     NSManagedObjectContext *context = [self.fetchedResultsController managedObjectContext];
     NSEntityDescription *entity = [[self.fetchedResultsController fetchRequest] entity];
     NSManagedObject *newManagedObject = [NSEntityDescription insertNewObjectForEntityForName:[entity name] inManagedObjectContext:context];

-    [newManagedObject setValue:[NSDate date] forKey:@"timeStamp"];                 // テンプレートにおける timeStampキー値を [NSDate date] の代わりに、
+    [newManagedObject setValue:@"Hoge" forKey:@"memo"];                            // MemoDate における memoキー値を @"Hoge"
+    [newManagedObject setValue:@"Piyo" forKey:@"category"];                        // MemoDate における categoryキー値を @"Piyo"
+    [newManagedObject setValue:[NSDate date] forKey:@"date"];                      // MemoDate における dateキー値を [NSDate date], 現在の時刻
+    [newManagedObject setValue:[NSNumber numberWithBool:YES] forKey:@"notifiable"]; // MemoDate における notifiableキー値を NSNumber に変換した YES


     NSError *error = nil;
     if (![context save:&error]) {
         NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
         abort();
     }
+#else                           // 上記のマクロが定義されていたら有効になる、新規の管理オブジェクトを挿入し、AddViewController を表示する箇所
+  NSManagedObjectContext *context = [[NSManagedObjectContext alloc] init]; // 追加用の管理オブジェクトコンテキストを作成、初期化
+  // フェッチ結果コントローラから得られる管理オブジェクトコンテキストから得られる永続ストアコーディネータを、追加用の管理オブジェクトコンテキストに設定
+  [context setPersistentStoreCoordinator:[[self.fetchedResultsController managedObjectContext] persistentStoreCoordinator]];
+  NSEntityDescription *entity = [[self.fetchedResultsController fetchRequest] entity];      // フェッチ結果コントローラのフェッチ要求のエンティティ
+
+  AddViewController *addViewController = [[AddViewController alloc] initWithStyle:UITableViewStyleGrouped];  // AddViewController の作成、初期化
+  addViewController.delegate = self;     // AddViewController のデリゲート変数に自分自身を設定、後述の AddViewControllerDelegate プロトコルで使用
+  // 追加用の管理オブジェクトコンテキスト上でエンティティ名による新規の管理オブジェクトを挿入、それを AddViewController の選択されたオブジェクトに設定
+  addViewController.selectedObject = [NSEntityDescription insertNewObjectForEntityForName:[entity name] inManagedObjectContext:context];
+  [addViewController.selectedObject setValue:[NSDate date] forKey:@"date"]; // 新規の管理オブジェクトの timestampキー値を [NSDate date], 現在の時刻に設定
+  self.addingManagedObjectContext = context;      // 追加用の管理オブジェクトコンテキストをセッターメソッドにより自ら保持(retain)
+  [context release];
+
+  // AddViewController 用のナビゲーションコントローラの作成、初期化
+  UINavigationController *navController = [[UINavigationController alloc] initWithRootViewController:addViewController];
+  // 自身のナビゲーションコントローラにモーダルビューコントローラとして AddViewController 用のナビゲーションコントローラの表示を指示
+  [self.navigationController presentModalViewController:navController animated:YES];
+
+  [addViewController release];
+  [navController release];
+#endif
 }
```

--- MemoDates+CoreData/Classes/RootViewController.m
+++ MemoDates/Classes/RootViewController.m の続き

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(5)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/RootViewController.m
+++ MemoDates/Classes/RootViewController.m  の続き
```

```
+#if defined(ENABLE_ADD_VIEW)                                                    // このマクロが定義されていたら、
+- (void)addViewController:(AddViewController *)controller didFinishWithSave:(BOOL)save {    // AddViewControllerDelegate 形式プロトコルメソッドの定義
+
+  if (save) {
+    NSNotificationCenter *dnc = [NSNotificationCenter defaultCenter];
+    [dnc addObserver:self selector:@selector(addControllerContextDidSave:) name:NSManagedObjectContextDidSaveNotification
+                              object:self.addingManagedObjectContext];       // 既定の通知センターに保存の完了のための監視者として自身のメソッドを追加
+
+    NSError *error;
+    if (![self.addingManagedObjectContext save:&error]) {                     // 追加用の管理オブジェクトコンテキストで、保存を試みている
+      NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+      abort();
+    }
+    [dnc removeObserver:self name:NSManagedObjectContextDidSaveNotification object:self.addingManagedObjectContext];  // 監視者を削除
+  }
+  self.addingManagedObjectContext = nil;
+
+  [self dismissModalViewControllerAnimated:YES];                  // モーダルビューコントローラ(AddViewController 用のナビゲーションコントローラ)を終了
+}

+- (void)addControllerContextDidSave:(NSNotification*)saveNotification {    // 保存が完了すると、上記で追加した既定の通知センターから呼ばれるメソッド
+
+  NSManagedObjectContext *context = [self.fetchedResultsController managedObjectContext];
+  [context mergeChangesFromContextDidSaveNotification:saveNotification];    // 管理オブジェクトコンテキストに追加用の管理オブジェクトコンテキストでの変更をマージ
+}
+#endif

 - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {

     static NSString *CellIdentifier = @"Cell";

     UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
     if (cell == nil) {
-      cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault  reuseIdentifier:CellIdentifier] autorelease];
+      cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle reuseIdentifier:CellIdentifier] autorelease];
     }                                                                                  // テーブルビューセルのスタイルの変更

     // Configure the cell.
     [self configureCell:cell atIndexPath:indexPath];

     return cell;
 }

+- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {       // セクションのヘッダのタイトルを返すメソッド
+  return [[self.fetchedResultsController.sections objectAtIndex:section] name];                    // フェッチ結果コントローラのセクションの名前を返す
+}
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(6)

```
 - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {        // 行の選択への応答
     // Navigation logic may go here -- for example, create and push another view controller.
-    /*                                                                                              // ここをアンコメントし、ほぼ説明通りに、
-     <#DetailViewController#> *detailViewController = [[<#DetailViewController#> alloc] initWithNibName:@"<#Nib name#>" bundle:nil]; // これを
+#if defined(ENABLE_DETAIL_VIEW)                                                                     // このマクロが定義されていたら、
+     DetailViewController *detailViewController = [[DetailViewController alloc] initWithStyle:UITableViewStyleGrouped];       // 作成、初期化
      NSManagedObject *selectedObject = [[self fetchedResultsController] objectAtIndexPath:indexPath];
+     detailViewController.selectedObject = selectedObject;                                          // 加えて、選択された管理オブジェクトを設定して、
      // ...
      // Pass the selected object to the new view controller.
      [self.navigationController pushViewController:detailViewController animated:YES];              // 自身のナビゲーションコントローラにプッシュ
      [detailViewController release];
-     */
+#endif
 }

 - (NSFetchedResultsController *)fetchedResultsController {                                          // フェッチ結果コントローラのゲッターメソッド
     NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
-    NSEntityDescription *entity = [NSEntityDescription entityForName:@"Event" inManagedObjectContext:self.managedObjectContext]; // エンティティ名を
+    NSEntityDescription *entity = [NSEntityDescription entityForName:@"MemoDate" inManagedObjectContext:self.managedObjectContext]; // MemoDate に
     [fetchRequest setEntity:entity];
     [fetchRequest setFetchBatchSize:20];

-    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"timeStamp" ascending:NO];     // ソート記述子を timeStampキー値の降順
-    NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor, nil];                       // の配列ではなく、
+    NSSortDescriptor *categorySortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"category" ascending:YES];      // categoryキー値の昇順と
+    NSSortDescriptor *dateSortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"date" ascending:NO];               // dateキー値の降順
+    NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:categorySortDescriptor, dateSortDescriptor, nil];     // の配列とする

     [fetchRequest setSortDescriptors:sortDescriptors];

     // Edit the section name key path and cache name if appropriate.
     // nil for section name key path means "no sections".
-    NSFetchedResultsController *aFetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest:fetchRequest
-        managedObjectContext:self.managedObjectContext sectionNameKeyPath:nil cacheName:@"Root"];             // フェッチ結果コントローラのセクション名を
+    NSFetchedResultsController *aFetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest:fetchRequest
+        managedObjectContext:self.managedObjectContext sectionNameKeyPath:@"category" cacheName:@"Root"];     // categoryキー値に変更
     aFetchedResultsController.delegate = self;
     self.fetchedResultsController = aFetchedResultsController;

     [aFetchedResultsController release];
     [fetchRequest release];
-    [sortDescriptor release];
+    [categorySortDescriptor release];
+    [dateSortDescriptor release];
     [sortDescriptors release];

     NSError *error = nil;
     if (![fetchedResultsController_ performFetch:&error]) {
```

```
 - (void)viewDidUnload {
     // Relinquish ownership of anything that can be recreated in viewDidLoad or on demand.
     // For example: self.myOutlet = nil;
+  self.fetchedResultsController = nil;
 }

 - (void)dealloc {
     [fetchedResultsController_ release];
     [managedObjectContext_ release];
+#if defined(ENABLE_ADD_VIEW)
+    [addingManagedObjectContext_ release];
+#endif
     [super dealloc];
 }
```

--- MemoDates+CoreData/Classes/RootViewController.m
+++ MemoDates/Classes/RootViewController.m の続き

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(7)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較
$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/DetailViewController.h
+++ MemoDates/Classes/DetailViewController.h

 #import <UIKit/UIKit.h>


 @interface DetailViewController : UITableViewController {
+  UISwitch *notifiableSwitch;         // notifiableキー値のためのスイッチのインスタンス変数の宣言
+  NSManagedObject *selectedObject;    // 選択された管理オブジェクトのインスタンス変数の宣言
+#if defined(ENABLE_UNDO_EDITING)      // このマクロが定義されていたら、
+  NSUndoManager *undoManager;         // アンドゥマネージャのインスタンス変数の宣言
+#endif
 }

+// これらは retain されたプロパティ
+@property (nonatomic, retain) UISwitch *notifiableSwitch;
+@property (nonatomic, retain) NSManagedObject *selectedObject;
+#if defined(ENABLE_UNDO_EDITING)      // このマクロが定義されていたら、
+@property (nonatomic, retain) NSUndoManager *undoManager;
+
+- (void)setUpUndoManager;             // アンドゥマネージャのセットアップのためのメソッド
+- (void)cleanUpUndoManager;           // アンドゥマネージャのクリーンアップのためのメソッド
+#endif
+
 @end
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(8)

```
--- MemoDates+CoreData/Classes/DetailViewController.m
+++ MemoDates/Classes/DetailViewController.m

 #import "DetailViewController.h"
+#if defined(ENABLE_EDITING_VIEW)                    // このマクロが定義されていたら、
+#import "MemoEditingViewController.h"
+#import "CategoryEditingViewController.h"
+#import "DateEditingViewController.h"
+#endif

 @implementation DetailViewController

+@synthesize notifiableSwitch;                       // notifiableキー値のためのスイッチのアクセサメソッドの定義
+@synthesize selectedObject;                         // 選択された管理オブジェクトのアクセサメソッドの定義
+#if defined(ENABLE_UNDO_EDITING)                    // このマクロが定義されていたら、
+@synthesize undoManager;                            // アンドゥマネージャのアクセサメソッドの定義
+#endif

+- (void)updateRightBarButtonItemState {             // 右のバーボタンの状態の更新するメソッド(私的に利用)
+                                        // 自身のナビゲーションアイテムの右のバーボタンが有効か否かは、選択された管理オブジェクトが更新において有効か否かに設定
+  self.navigationItem.rightBarButtonItem.enabled = [selectedObject validateForUpdate:NULL];
+}

-/*                                                  // このメソッドをアンコメントして、
 - (void)viewDidLoad {                              // ビューのメモリへの読み込みが完了したとき
     [super viewDidLoad];

     // Uncommon the following line to display an Edit button in the navigation bar for this view controller.
-    // self.navigationItem.rightBarButtonItem = self.editButtonItem;
+    self.navigationItem.rightBarButtonItem = self.editButtonItem;  // ナビゲーションの右のバーボタンに、自身の編集ボタンを設定
+    self.tableView.allowsSelectionDuringEditing = YES;             // 編集モード時のセルの選択を許可
 }
-*/

-/*                                                  // このメソッドをアンコメントして、
 - (void)viewWillAppear:(BOOL)animated {            // ビューが表示されようとしているとき
+  [self.tableView reloadData];                      // 自身のテーブルビューにデータの再読み込みを指示
+  [self updateRightBarButtonItemState];             // 右のバーボタンアイテムの状態の更新
     [super viewWillAppear:animated];
 }
-*/
-/*                                                  // このメソッドをアンコメントして、
 - (void)viewDidAppear:(BOOL)animated {             // ビューが表示されたとき
     [super viewDidAppear:animated];
+    [self becomeFirstResponder];                    // 自身がファーストレスポンダを取得する、この場合、後のアンドゥマネージャの動作のための準備
 }
-*/
-/*                                                  // このメソッドをアンコメントして、
 - (void)viewWillDisappear:(BOOL)animated {
     [super viewWillDisappear:animated];
+    [self resignFirstResponder];                    // 自身はファーストレスポンダを棄権する、この場合、先のアンドゥマネージャの動作のための準備に対する後始末
 }
-*/
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(9)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較
$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
+- (void)setEditing:(BOOL)editing animated:(BOOL)animated {                  // 編集モードか否かの設定の為のメソッド
+  [super setEditing:editing animated:animated];                             // 親クラスのメソッド通りに行い、加えて、
+
+  [self.navigationItem setHidesBackButton:editing animated:animated];       // 戻るボタンは編集モードなら隠す
+  [self.tableView reloadData];                                              // 編集モードが変化したわけなので、
+                                                                            // 自身のテーブルビューにデータの再読み込みを指示
+  if (editing) {                                                            // 編集モードなら、
+#if defined(ENABLE_UNDO_EDITING)                                            // このマクロが定義されていたら、
+    [self setUpUndoManager];                                                // アンドゥマネージャをセットアップ
+#endif
+  }
+  else {                                                                    // 非編集モードなら、
+#if defined(ENABLE_UNDO_EDITING)                                            // このマクロが定義されていたら、
+    [self cleanUpUndoManager];                                              // アンドゥマネージャをクリーンアップ
+#endif
+    NSError *error;
+    if (![selectedObject.managedObjectContext save:&error]) {       // 選択された管理オブジェクトの管理オブジェクトコンテキスト上で、保存を試みる
+      NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
+      abort();
+    }
+  }
+}

 - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {        // テーブルビューのセクション数は、
     // Return the number of sections.
-    return <#number of sections#>;
+    return 1;                                                               // 1
 }


 - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section { // テーブルビューのセクションにおける行数は、
     // Return the number of rows in the section.
-    return <#number of rows in section#>;
+    return 4;                                                               // memo, category, date, notifiable キー値の 4
 }
```

--- MemoDates+CoreData/Classes/DetailViewController.m
+++ MemoDates/Classes/DetailViewController.m の続き

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(10)

```
// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {     // 行のテーブルビューセルを返すメソッド

    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
-       cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];       // スタイルの
+       cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue2 reuseIdentifier:CellIdentifier] autorelease];       // 変更のみ
+       cell.editingAccessoryType = UITableViewCellAccessoryDisclosureIndicator;    // セルの編集モードにおけるアクセサリタイプをディスクロージャインジケータに
    }

    // Configure the cell...                                                                    // テーブルビューセルの様相を記述
+  switch (indexPath.row) {
+    case 0:                                                                                   // ０行目なら、
+      cell.textLabel.text = @"Memo.";                                                         // textLabel.textを設定
+      cell.detailTextLabel.text = [[selectedObject valueForKey:@"memo"] description];         // detailTextLabel.text を memoキー値に設定
+      break;
+    case 1:                                                                                   // １行目なら、
+      cell.textLabel.text = @"Category";                                                      // textLabel.text を設定
+      cell.detailTextLabel.text = [[selectedObject valueForKey:@"category"] description];     // detailTextLabel.text を categoryキー値に設定
+      break;
+    case 2:                                                                                   // ２行目なら、
+      cell.textLabel.text = @"Date";                                                          // textLabel.text を設定
+                                                                                              // detailTextLabel.text を dateキー値に設定
+      cell.detailTextLabel.text = [NSDateFormatter localizedStringFromDate:[selectedObject valueForKey:@"date"]
+                                                                 dateStyle:NSDateFormatterMediumStyle timeStyle:NSDateFormatterNoStyle];
+      break;
+    case 3:                                                                                   // ３行目なら、
+      cell.selectionStyle = UITableViewCellSelectionStyleNone;                                // セルの選択についてのスタイルを無しにして、
+      cell.textLabel.text = @"Notifiable";                                                    // textLabel.text を設定
+      self.notifiableSwitch.on = [[selectedObject valueForKey:@"notifiable"] boolValue];      // notifiable 用スイッチの on を notifiableキー値に設定
+      self.notifiableSwitch.enabled = self.editing;                                           // notifiable 用スイッチの有効か否かを、編集モードか否かに設定
+      cell.accessoryView = cell.editingAccessoryView = self.notifiableSwitch;                 // セルの(編集モード)アクセサリビューを notifiable 用スイッチ
+      break;
+  }

    return cell;
 }
+- (NSIndexPath *)tableView:(UITableView *)tableView willSelectRowAtIndexPath:(NSIndexPath *)indexPath
+{
+  return self.editing ? indexPath : nil;          // 編集モードなら行の選択に対応、さもなくば、選択は不可
+}
+- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath
+{
+  return UITableViewCellEditingStyleNone;         // 編集モードのスタイルを無しに(スタイルにはこの他に、削除と追加がある)
+}
+- (BOOL)tableView:(UITableView *)tableView shouldIndentWhileEditingRowAtIndexPath:(NSIndexPath *)indexPath
+{
+  return NO;                                      // 編集モードで行のインデントはしない
+}
```

--- MemoDates+CoreData/Classes/DetailViewController.m
+++ MemoDates/Classes/DetailViewController.m の続き

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(11)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
+#if defined(ENABLE_UNDO_EDITING)                                   // このマクロが定義されていたら、
+- (void)setUpUndoManager {                                        // アンドゥマネージャのセットアップのためのメソッド
+  if (selectedObject.managedObjectContext.undoManager == nil) {
+    NSUndoManager *anUndoManager = [[NSUndoManager alloc] init];   // アンドゥマネージャを作成、初期化
+    [anUndoManager setLevelsOfUndo:4];                             // levelsOfUndo を設定、0だと無制限
+    self.undoManager = anUndoManager;                              // アンドゥマネージャを自身で保持
+    [anUndoManager release];
+
+    selectedObject.managedObjectContext.undoManager = undoManager; // 選択された管理オブジェクトの管理オブジェクトコンテキストのアンドゥマネージャに設定
+  }
+  NSUndoManager *anUndoManager = selectedObject.managedObjectContext.undoManager;
+
+  NSNotificationCenter *dnc = [NSNotificationCenter defaultCenter];        // 既定の通知センターにアンドゥ・リドゥ完了の監視者として自身のメソッドを追加
+  [dnc addObserver:self selector:@selector(undoManagerDidUndo:) name:NSUndoManagerDidUndoChangeNotification object:anUndoManager];
+  [dnc addObserver:self selector:@selector(undoManagerDidRedo:) name:NSUndoManagerDidRedoChangeNotification object:anUndoManager];
+}
+- (void)cleanUpUndoManager {     // アンドゥマネージャのクリーンアップのためのメソッド
+  [[NSNotificationCenter defaultCenter] removeObserver:self];      // 自身のすべての監視者を削除
+
+  if (selectedObject.managedObjectContext.undoManager == undoManager) {    // 保持されているアンドゥマネージャを nil に設定つまり解放
+    selectedObject.managedObjectContext.undoManager = nil;
+    self.undoManager = nil;
+  }
+}
+- (NSUndoManager *)undoManager {                                   // アンドゥマネージャのゲッターメソッド
+  return selectedObject.managedObjectContext.undoManager;
+}
+- (void)undoManagerDidUndo:(NSNotification *)notification {        // 既定の通知センターから呼ばれるメソッド
+  [self.tableView reloadData];                                     // テーブルビューにデータの再読み込みを指示
+  [self updateRightBarButtonItemState];                            // 右のバーボタンの状態の更新
+}
+- (void)undoManagerDidRedo:(NSNotification *)notification {        // 既定の通知センターから呼ばれるメソッド
+  [self.tableView reloadData];                                     // テーブルビューにデータの再読み込みを指示
+  [self updateRightBarButtonItemState];                            // 右のバーボタンの状態の更新
+}
+
+- (BOOL)canBecomeFirstResponder {                                  // アンドゥマネージャを利用するにはファーストレスポンダになれる必要がある
+  return YES;
+}
+#endif
```

--- MemoDates+CoreData/Classes/DetailViewController.m
+++ MemoDates/Classes/DetailViewController.m の続き

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(12)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
 - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
     // Navigation logic may go here. Create and push another view controller.
     /*
     <#DetailViewController#> *detailViewController = [[<#DetailViewController#> alloc] initWithNibName:@"<#Nib name#>" bundle:nil];
      // ...
      // Pass the selected object to the new view controller.
     [self.navigationController pushViewController:detailViewController animated:YES];
     [detailViewController release];
     */
+#if defined(ENABLE_EDITING_VIEW) // このマクロが定義されていたら、
+  switch (indexPath.row) {
+    case 0: {                                                                    // ０行目なら、MemoEditingViewController を作成、初期化
+        MemoEditingViewController *controller = [[MemoEditingViewController alloc] initWithNibName:@"MemoEditingViewController" bundle:nil];
+        controller.editedObject = selectedObject;                                // 選択された管理オブジェクトが、その編集される管理オブジェクト
+        [self.navigationController pushViewController:controller animated:YES];     // 自身のナビゲーションコントローラにプッシュ
+        [controller release];
+      }
+      break;
+    case 1: {                                                                    // １行目なら、MemoEditingViewController を作成、初期化
+        CategoryEditingViewController *controller = [[CategoryEditingViewController alloc] initWithNibName:@"CategoryEditingViewController" bundle:nil];
+        controller.editedObject = selectedObject;                                // 選択された管理オブジェクトが、その編集される管理オブジェクト
+        [self.navigationController pushViewController:controller animated:YES];     // 自身のナビゲーションコントローラにプッシュ
+        [controller release];
+      }
+      break;
+    case 2: {                                                                    // ２行目なら、MemoEditingViewController を作成、初期化
+        DateEditingViewController *controller = [[DateEditingViewController alloc] initWithNibName:@"DateEditingViewController" bundle:nil];
+        controller.editedObject = selectedObject;                                // 選択された管理オブジェクトが、その編集される管理オブジェクト
+        [self.navigationController pushViewController:controller animated:YES];     // 自身のナビゲーションコントローラにプッシュ
+        [controller release];
+      }
+      break;
+  }
+#endif
 }
```

```
--- MemoDates+CoreData/Classes/DetailViewController.m
+++ MemoDates/Classes/DetailViewController.m  の続き
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(13)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
+- (IBAction)notifiableSwitchValueChanged:(UISwitch *)sender                    // notifiableキー値のためのスイッチの値が変わったら呼ばれるメソッド
+{
+#if defined(ENABLE_UNDO_EDITING)                                              // このマクロが定義されていたら、
+  [undoManager setActionName:[NSString stringWithFormat:@"Notifiable"]];      // アンドゥマネージャにアクション名の設定
+#endif
+  [selectedObject setValue:[NSNumber numberWithBool:sender.on] forKey:@"notifiable"];       // 選択された管理オブジェクトの notifiableキー値を変更
+}

+- (UISwitch *)notifiableSwitch {                                              // notifiableキー値のためのスイッチのゲッターメソッド
+  if (notifiableSwitch == nil) {
+    notifiableSwitch = [[UISwitch alloc] initWithFrame:CGRectMake(0, 0, 0, 0)];            // スイッチを作成、初期化、および、アクションの設定
+    [notifiableSwitch addTarget:self action:@selector(notifiableSwitchValueChanged:) forControlEvents:UIControlEventValueChanged];
+  }
+  return notifiableSwitch;
+}

- (void)viewDidUnload {
+  self.notifiableSwitch = nil;                                                // notifiableキー値のためのスイッチを解放
 }

 - (void)dealloc {
+#if defined(ENABLE_UNDO_EDITING)                                              // このマクロが定義されていたら、
+  [undoManager release];                                                      // アンドゥマネージャの参照カウンタを下げる
+#endif
+  [notifiableSwitch release];                                                 // notifiableキー値のためのスイッチの参照カウンタを下げる
+  [selectedObject release];                                                   // 選択された管理オブジェクトの参照カウンタを下げる
     [super dealloc];
 }
```

--- MemoDates+CoreData/Classes/DetailViewController.m
+++ MemoDates/Classes/DetailViewController.m の続き

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(14)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

```
--- MemoDates+CoreData/Classes/EditingViewController.h
+++ MemoDates/Classes/EditingViewController.h

 #import <UIKit/UIKit.h>

 @interface EditingViewController : UIViewController {
+  NSManagedObject *editedObject;                                  // 編集される管理オブジェクトのためのインスタンス変数
 }

+@property (nonatomic, retain) NSManagedObject *editedObject;     // 編集される管理オブジェクトは retain されるプロパティ
+
+- (IBAction)cancel:(id)sender;                                   // 編集の取消のためのメソッドの宣言
+- (IBAction)save:(id)sender;                                     // 編集の保存のためのメソッドの宣言
+
 @end

--- MemoDates+CoreData/Classes/EditingViewController.m
+++ MemoDates/Classes/EditingViewController.m

 #import "EditingViewController.h"

 @implementation EditingViewController

+@synthesize editedObject;

-/*                                                                // このメソッドをアンコメントし、
 - (void)viewDidLoad {
     [super viewDidLoad];
+
+  UIBarButtonItem *saveButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemSave
+                                 target:self action:@selector(save:)];     // バーボタンの作成、初期化、ターゲット(self)＆アクション(save:)の設定
+  self.navigationItem.rightBarButtonItem = saveButton;                     // ナビゲーションの右のバーボタンに、このバーボタンを設定
+  [saveButton release];                                                    // したので、その参照カウンタを下げる
+
+  UIBarButtonItem *cancelButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemCancel
+                                  target:self action:@selector(cancel:)]; // バーボタンの作成、初期化、ターゲット(self)＆アクション(cancel:)の設定
+  self.navigationItem.leftBarButtonItem = cancelButton;                    // ナビゲーションの左のバーボタンに、このバーボタンを設定
+  [cancelButton release];                                                  // したので、その参照カウンタを下げる
 }
-*/

+- (IBAction)cancel:(id)sender {                                   // 編集の取消のためのメソッドの定義
+  [self.navigationController popViewControllerAnimated:YES];     // ナビゲーションコントローラにトップのビューの除去を指示
+}
+- (IBAction)save:(id)sender {                                     // 編集の保存のためのメソッドの定義
+  [self.navigationController popViewControllerAnimated:YES];     // ナビゲーションコントローラにトップのビューの除去を指示
+}

 - (void)dealloc {
+    [editedObject release];                                       // 編集される管理オブジェクトの参照カウンタを下げる
     [super dealloc];
 }
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(15)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/MemoEditingViewController.h
+++ MemoDates/Classes/MemoEditingViewController.h

 #import <UIKit/UIKit.h>
+#import "EditingViewController.h"

-@interface MemoEditingViewController : UIViewController {
+@interface MemoEditingViewController : EditingViewController {      // EditingViewController を継承
+  IBOutlet UITextField *textField;                                  // テキストフィールドのアウトレット
 }

 @end

--- MemoDates+CoreData/Classes/MemoEditingViewController.m
+++ MemoDates/Classes/MemoEditingViewController.m

-/*                                                                  // このメソッドをアンコメントし、
 - (void)viewDidLoad {
     [super viewDidLoad];
+    self.title = @"Memo.";                                         // タイトルを設定
 }

+- (void)viewWillAppear:(BOOL)animated {                            // このメソッドを追加し、
+  [super viewWillAppear:animated];                                 // 親のクラスのメソッド通りに行い、
+  textField.text = [editedObject valueForKey:@"memo"];             // 編集される管理オブジェクトの memoキー値をテキストフィールドに表示
+  [textField becomeFirstResponder];                                // ソフトウェアキーボードが表示されるよう、ファーストレスポンダになる
+}
-*/

+- (IBAction)cancel:(id)sender {                                    // 編集の取消のためのメソッドの定義
+  [super cancel:sender];                                           // 親のクラスのメソッド通りに行う(トップのビューが除去)
+}
+- (IBAction)save:(id)sender {                                      // 編集の保存のためのメソッドの定義
+#if defined(ENABLE_UNDO_EDITING)                                   // このマクロが定義されていたら、
+  NSUndoManager *undoManager = [[editedObject managedObjectContext] undoManager];
+  [undoManager setActionName:[NSString stringWithFormat:@"%@", self.title]];       // アンドゥマネージャにアクション名の設定
+#endif
+  [editedObject setValue:textField.text forKey:@"memo"];           // テキストフィールドの文字列を編集される管理オブジェクトの memoキー値に設定
+  [super save:sender];                                             // 親のクラスのメソッド通りに行う(トップのビューが除去)
+}
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(16)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/CategoryEditingViewController.h
+++ MemoDates/Classes/CategoryEditingViewController.h

 #import <UIKit/UIKit.h>
+#import "EditingViewController.h"

-@interface CategoryEditingViewController : UIViewController {
+@interface CategoryEditingViewController : EditingViewController {  // EditingViewController を継承
+  IBOutlet UITextField *textField;                                  // テキストフィールドのアウトレット
 }

 @end

--- MemoDates+CoreData/Classes/CategoryEditingViewController.m
+++ MemoDates/Classes/CategoryEditingViewController.m

-/*                                                                  // このメソッドをアンコメントし、
 - (void)viewDidLoad {
     [super viewDidLoad];
+    self.title = @"Category";                                      // タイトルを設定
 }

+- (void)viewWillAppear:(BOOL)animated {                            // このメソッドを追加し、
+  [super viewWillAppear:animated];                                 // 親のクラスのメソッド通りに行い、
+  textField.text = [editedObject valueForKey:@"category"];         // 編集される管理オブジェクトの categoryキー値をテキストフィールドに表示
+  [textField becomeFirstResponder];                                // ソフトウェアキーボードが表示されるよう、ファーストレスポンダになる
+}
-*/

+- (IBAction)cancel:(id)sender {                                    // 編集の取消のためのメソッドの定義
+  [super cancel:sender];                                           // 親のクラスのメソッド通りに行う(トップのビューが除去)
+}
+- (IBAction)save:(id)sender {                                      // 編集の保存のためのメソッドの定義
+#if defined(ENABLE_UNDO_EDITING)                                   // このマクロが定義されていたら、
+  NSUndoManager *undoManager = [[editedObject managedObjectContext] undoManager];
+  [undoManager setActionName:[NSString stringWithFormat:@"%@", self.title]];       // アンドゥマネージャにアクション名の設定
+#endif
+  [editedObject setValue:textField.text forKey:@"category"];       // テキストフィールドの文字列を編集される管理オブジェクトの categoryキー値に設定
+  [super save:sender];                                             // 親のクラスのメソッド通りに行う(トップのビューが除去)
+}
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(17)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/Classes/DateEditingViewController.h
+++ MemoDates/Classes/DateEditingViewController.h

 #import <UIKit/UIKit.h>
+#import "EditingViewController.h"

-@interface DateEditingViewController : UIViewController {
+@interface DateEditingViewController : EditingViewController {      // EditingViewController を継承
+  IBOutlet UIDatePicker *datePicker;                                // 日付ピッカーのアウトレット
 }

 @end

--- MemoDates+CoreData/Classes/DateEditingViewController.m
+++ MemoDates/Classes/DateEditingViewController.m

-/*                                                                  // このメソッドをアンコメントし、
 - (void)viewDidLoad {
     [super viewDidLoad];
+    self.title = @"Date";                                          // タイトルを設定
}

+- (void)viewWillAppear:(BOOL)animated {                            // このメソッドを追加し、
+  [super viewWillAppear:animated];                                 // 親のクラスのメソッド通りに行い、
+  datePicker.date = [editedObject valueForKey:@"date"];            // 編集される管理オブジェクトの dateキー値を日付ピッカーに設定
+ }
-*/

+- (IBAction)cancel:(id)sender {                                    // 編集の取消のためのメソッドの定義
+  [super cancel:sender];                                           // 親のクラスのメソッド通りに行う(トップのビューが除去)
+}
+- (IBAction)save:(id)sender {                                      // 編集の保存のためのメソッドの定義
+#if defined(ENABLE_UNDO_EDITING)                                   // このマクロが定義されていたら、
+  NSUndoManager *undoManager = [[editedObject managedObjectContext] undoManager];
+  [undoManager setActionName:[NSString stringWithFormat:@"%@", self.title]];       // アンドゥマネージャにアクション名の設定
+#endif
+  [editedObject setValue:datePicker.date forKey:@"date"];          // 日付ピッカーの日付を編集される管理オブジェクトの dateキー値に設定
+  [super save:sender];                                             // 親のクラスのメソッド通りに行う(トップのビューが除去)
+}
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(18)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

```
--- MemoDates+CoreData/Classes/AddViewController.h
+++ MemoDates/Classes/AddViewController.h

 #import <UIKit/UIKit.h>
+#import "DetailViewController.h"

+@protocol AddViewControllerDelegate;                                  // AddViewControllerDelegate 形式プロトコル名の宣言

-@interface AddViewController : UIViewController {
+@interface AddViewController : DetailViewController {                 // DetailViewController を継承
+   id <AddViewControllerDelegate> delegate;                           // AddViewControllerDelegate 形式プロトコル型インスタンス変数の委譲の宣言
 }

+@property (nonatomic, assign) id <AddViewControllerDelegate> delegate;      // 委譲は assign されるプロパティ
 @end
+@protocol AddViewControllerDelegate                                   // AddViewControllerDelegate 形式プロトコルの宣言
+- (void)addViewController:(AddViewController *)controller didFinishWithSave:(BOOL)save;     // 形式プロトコルのメソッドの宣言
+@end

--- MemoDates+CoreData/Classes/AddViewController.m
+++ MemoDates/Classes/AddViewController.m

+@synthesize delegate;                                                 // 委譲のアクセサメソッドの定義

-/*                                                                    // このメソッドをアンコメントして、
 - (void)viewDidLoad {
     [super viewDidLoad];
+   self.title = @"New MemoDate";                                      // タイトルを設定
+   self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemCancel
+         target:self action:@selector(cancel:)] autorelease];         // ナビゲーションの左のバーボタンの作成、初期化、ターゲット(self)＆アクション(cancel:)の設定
+   self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemSave
+         target:self action:@selector(save:)] autorelease];           // ナビゲーションの右のバーボタンの作成、初期化、ターゲット(self)＆アクション(save:)の設定
+
+#if defined(ENABLE_UNDO_EDITING)                                      // このマクロが定義されていたら、
+   [self setUpUndoManager];                                           // アンドゥマネージャをセットアップ
+#endif
+   self.editing = YES;                                                // 始めから編集モードに
 }
-*/
 - (void)viewDidUnload {
     [super viewDidUnload];
+#if defined(ENABLE_UNDO_EDITING)                                      // このマクロが定義されていたら、
+   [self cleanUpUndoManager];                                         // アンドゥマネージャをクリーンアップ
+#endif
 }
+- (IBAction)cancel:(id)sender {
+   [delegate addViewController:self didFinishWithSave:NO];            // 形式プロトコルのメソッドの呼び出し
+}
+- (IBAction)save:(id)sender {
+   [delegate addViewController:self didFinishWithSave:YES];           // 形式プロトコルのメソッドの呼び出し
+}
```

Core Data ありの Navigation-based App. テンプレートから完成した MemoDates へのパッチで学ぶ(19)

つまり、P.21 の段階の **MemoDates+CoreData/** と**完成した MemoDates** を比較

$ diff -wBbE -ru -U 7 MemoDates+CoreData/ MemoDates/ | less

```
--- MemoDates+CoreData/MemoDates-Info.plist
+++ MemoDates/MemoDates-Info.plist
      <key>CFBundleDevelopmentRegion</key>
      <string>English</string>
      <key>CFBundleDisplayName</key>
      <string>${PRODUCT_NAME}</string>
      <key>CFBundleExecutable</key>
      <string>${EXECUTABLE_NAME}</string>
      <key>CFBundleIconFile</key>
-     <string></string>
+     <string>MemoDatesIcon</string>                                  <!-- アイコンファイル名 -->
      <key>CFBundleIdentifier</key>
-     <string>com.yourcompany.${PRODUCT_NAME:rfc1034identifier}</string>
+     <string>com.mac.taiji-yamada.${PRODUCT_NAME:rfc1034identifier}</string>     <!-- バンドルの識別子 -->
      <key>CFBundleInfoDictionaryVersion</key>
      <string>6.0</string>
      <key>CFBundleName</key>
      <string>${PRODUCT_NAME}</string>
      <key>CFBundlePackageType</key>
      <string>APPL</string>
      <key>CFBundleSignature</key>
      <string>????</string>
      <key>CFBundleVersion</key>
      <string>1.0</string>
      <key>LSRequiresIPhoneOS</key>
      <true/>
      <key>NSMainNibFile</key>
      <string>MainWindow</string>
+     <key>UIApplicationExitsOnSuspend</key>                          <!-- サスペンド時の終了を -->
+     <true/>                                                         <!-- 真に -->
 </dict>
 </plist>
```

Core Data ありの Navigation-based App.テンプレート＋各種ViewControllerテンプレートから完成させた
MemoDates App. での

Core Data とテーブルビューコントローラにおける要求と応答、処理のフロー

フェッチ要求

エンティティ記述：MemoDate、self.managedObjectContext
ソート記述子の配列：categorySortDescriptor, dateSortDescriptor
管理オブジェクトコンテキスト：self.managedObjectContext

P.33 fetchedResultsController

フェッチ結果コントローラ

フェッチ要求：fetchRequest
管理オブジェクトコンテキスト：self.managedObjectContext
セクション名：category

P.33 fetchedResultsController

追加・削除

エンティティ記述：MemoDate、 NSManagedObjectContext *(追加)
管理オブジェクト：addViewController.selectedObject(追加)
管理オブジェクト：self.fetchedResultsController経由(削除)

P.31 insertNewObject, P.15 tableView:commitEditingStyle

保存

管理オブジェクトコンテキスト：NSManagedObjectContext *

P.11, P.36 setEditing*, P.32 addViewController:didFinishWithSave

管理オブジェクトコンテキスト

永続ストアコーディネータ：self.persistentStoreCoordinator

P.12 managedObjectContext
P.31 insertNewObject

永続ストアコーディネータ

ストアURL：Documents/MemoDates.sqlite
管理オブジェクトモデル：self.managedObjectModel

P.12 persistentStoreCoordinator

管理オブジェクトモデル

モデルURL：MemoDates.momd/MemoDates.mom
　　エンティティ：MemoDate
　　　　プロパティ：memo/データ型「文字列」
　　　　プロパティ：date/データ型「日付」
　　　　プロパティ：category/データ型「文字列」、規定値「Private」
　　　　プロパティ：notifiable/データ型「論理値」、規定値「偽」

P.12 managedObjectModel

永続ストア

エンティティ：
　　オブジェクト：memo/自分の誕生日、date/1970-05-20、category/Private、notifiable/NO
　　オブジェクト：memo/長女の誕生日、date/2007-01-01、category/Private、notifiable/YES
　　オブジェクト：memo/会社の創立記念日、date/2011-01-13、category/Office、notifiable/NO
　　：
ストア(シミュレータの場合)：
　　Library/Application Support/iPhone Simulator/4.2/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Documents/MemoDates.sqlite

プログラミング言語 Objective-C

主な各種ディレクティブ、型、記法(1)

`@protocol Delegate;`	形式プロトコル名の宣言のディレクティブ
`@interface ClassName : SuperClass <Protocol> {`	クラスのインターフェースの宣言開始のディレクティブ
`  double value;`	インスタンス変数の宣言、倍精度型
`  id <Delegate> delegate;`	インスタンス変数の宣言、形式プロトコル型
`}`	
`+ classWithValue:(double)aValue;`	クラスメソッドの宣言（簡易コンストラクタ）
`- initWithValue:(double)aValue;`	インスタンスメソッドの宣言（指定イニシャライザ）
`- (double)value;`	インスタンスメソッドの宣言（ゲッターメソッド）
`- (void)setValue:(double)aValue;`	インスタンスメソッドの宣言（セッターメソッド）
`@property double value;`	アクセサ（インスタンス）メソッドの宣言、上記２行と等価
`@property (nonatomic, assign) id <Delegate> delegate;`	アクセサ（インスタンス）メソッドの宣言、括弧内はプロパティ宣言属性（既定はassign）
`@end`	宣言または定義終了のディレクティブ
`@protocol Delegate`	形式プロトコルのメソッドの宣言開始のディレクティブ
`- (void)doSomething;`	形式プロトコルのメソッドの宣言
`@end`	宣言または定義終了のディレクティブ
`@implementation ClassName`	クラスのインターフェースの定義開始のディレクティブ
`+ classWithValue:(double)aValue`	クラスメソッドの定義（簡易コンストラクタ）
`{`	
`  return [[[self alloc] initWithValue:aValue]`	
`          autorelease];`	
`}`	
`- initWithValue:(double)aValue {`	インスタンスメソッドの定義（指定イニシャライザ）
`  if ((self = [super init]))`	
`    value = aValue;`	
`  return self;`	
`}`	
`- (double)value`	インスタンスメソッドの定義（ゲッターメソッド）
`{`	
`  return value;`	
`}`	
`- (void)setValue:(double)aValue`	インスタンスメソッドの定義（セッターメソッド）
`{`	
`  value = aValue;`	
`}`	
`@synthesize value;`	アクセサ（インスタンス）メソッドの定義、上記２ブロックと同等（正確にはロックが掛かる）
`@dynamic value;`	アクセサメソッドを直接または動的に提供することをコンパイラに指示（synthesizeしない）
`@synthesize delegate;`	アクセサ（インスタンス）メソッドの定義、上記２ブロックと同等
`@end`	宣言または定義終了のディレクティブ

プログラミング言語 Objective-C

主な各種ディレクティブ、型、記法(2)

`@interface ClassName : SuperClass <Protocol> {`
`@private`
`  NSObject *object_;`
`}`
`- (NSObject *)object;`
`- (void)setObject:(NSObject *)anObject;`
`@property (nonatomic, retain) NSObject *object;`
`@end`
	クラスのインターフェースの宣言開始のディレクティブ
	以下のインスタンス変数のこのクラスに限定するディレクティブ
	インスタンス変数の宣言、NSObject *型
	
	インスタンスメソッドの宣言（ゲッターメソッド）
	インスタンスメソッドの宣言（セッターメソッド）
	アクセサ（インスタンス）メソッドの宣言、上記２行と等価、括弧内はプロパティ宣言属性
	宣言または定義終了のディレクティブ

`@interface ClassName ()`
`- (void)doSomethingWith:(id)target;`
`@end`
 	
`@implementation ClassName`
`- (NSObject *)object`
`{`
`  return object_;`
`}`
`- (void)setObject:(NSObject *)anObject`
`{`
`  [anObject retain];`
`  [object_ release];`
`  object_ = anObject;`
`}`
`@synthesize object=object_;`
`- (void)dealloc`
`{`
`  [object_ release];`
`  [super dealloc];`
`}`
 	
`- (void)doSomethingWith:(id)target {}`
`@end`
	クラス拡張の宣言開始、括弧内が空（匿名のカテゴリ、インスタンス変数は宣言不可）
	拡張（インスタンス）メソッドの宣言
	宣言または定義終了のディレクティブ
	
	クラスのインターフェースの定義開始のディレクティブ
	インスタンスメソッドの定義（ゲッターメソッド）
	
	インスタンスメソッドの定義（セッターメソッド）
	
	新しいオブジェクトの所有権を取得
	古いオブジェクトの所有権を放棄
	新たに代入する
	
	アクセサ（インスタンス）メソッドの定義、上記２ブロックと同等（正確にはロックが掛かる）
	親クラスのデストラクタのオーバーライド
	
	オブジェクトの所有権を放棄
	親クラスのデストラクタ
	
	拡張（インスタンス）メソッドの定義、これがないとコンパイル時にエラーになる
	宣言または定義終了のディレクティブ

プログラミング言語 Objective-C

主な各種ディレクティブ、型、記法(3)

id	オブジェクトへのポインタ型
nil	NULLオブジェクトポインタ、(id)NULL
BOOL	真偽型、char型
NO	偽値、(BOOL)0
YES	真値、(BOOL)1
#import	二度読み防止済み#includeディレクティブ
self	受信側オブジェクトへのポインタ変数
super	継承されている受信側オブジェクトへのポインタ変数
@"文字列"	NSString型オブジェクト定数ディレクティブ
@"文字列*1*" @"文字列*2*" … @"文字列*n*"	NSString型オブジェクト定数ディレクティブ、*n*個のディレクティブで指定された文字列を結合
[*receiver message*]	メッセージ式、セレクタ＝メソッド名は *message*
[*string* isEqual:*@"文字列"*]	単一引数をもつメッセージ式、セレクタは isEqual:
[NSString stringWithCstring:*buffer* encoding:*stringEncoding*]	複数引数をもつメッセージ式、セレクタは stringWithCstring:encoding:
[string withFormat:*@"%@=%g\n", @"文字列", 100.0*]	可変引数をもつメッセージ式、セレクタは withFormat:
receiver.value	ドット記法、アクセサのゲッターメソッドの呼び出し、[*receiver value*]と等価
receiver.value = 100.0	ドット記法、アクセサのセッターメソッドの呼び出し、[*receiver* set*Value*:*100.0*]と等価
@selector(*method_name*)	コンパイル済みセレクタを返すディレクティブ

まとめ

Core Data & Table View Programming

1. 大切な日付を記録するアプリを制作した
2. Core Data エンティティのモデルブラウザによる編集
3. NSManagedObject のキー値の取得と設定
   NSNumber が必要となる型だけは注意
4. Navigation-based Application + Table View はコード量が多し！
   しかし、狭い画面におけるUIのあり様として仕方が無いのかも…
5. アンドゥ・リドゥがアンドゥマネージャによって簡単にサポート！
6. Interface Builder によらない UI の生成@ DetailViewController
7. 形式プロトコルの提供と利用を行った@ AddViewController

参考文献

Core Data & Table View programming

1. Apple Inc., “iOS Core Data チュートリアル,” 2010. 本稿で紹介したテンプレートのエンティティのプロパティが「現在の時刻」から Core Location による「現在の位置情報」に変わっただけで、技術的な見地では作成できるアプリの程度はテンプレートと同質。しかも、iOS において特有のフェッチ結果コントローラの解説がなく残念。それらを踏まえれば参考になるチュートリアル

2. Apple Inc., “iPhoneヒューマンインターフェイスガイドライン,” 2010. ナビゲーションバーやテーブルビューにおける、iPhoneのようなデバイス特有のヒューマンインターフェースのあり方をわかりやすく説明した文書

3. Apple Inc., “iOS View Controllerプログラミングガイド,” 2010. ナビゲーションコントローラ、テーブルビューコントローラ、モーダルビューコントローラ他ビューコントローラー全般の利用場面を指南する文書

4. Apple Inc., “iOS Table Viewプログラミングガイド,” 2010. iPhoneのようなデバイスにおいて極めて重要なテーブルビューのプログラミングについての紹介

5. Apple Inc., “Cocoaメモリ管理プログラミングガイド,” 2009. iOSでは必須のメモリ管理と自動解放プールやアクセサメソッドにおける基礎。一方、Mac OS X v10.5 以降では Apple Inc., “Garbage Collection Programming Guide,” 2010 も参照のこと

その他の参考文献とキーワード

other iOS/Mac programming & keywords — 本講で扱えなかった話題

1. Apple Inc., “iOS Scroll View プログラミングガイド,” 2010. 本講で扱ったビューコントローラから漏れてしまったビューコントローラ。

2. Apple Inc., “Blocks プログラミングトピックス,” 2010. プログラミング言語Cの拡張として、手続きにスタック及びヒープ記憶域の結合を実現する言語仕様「Blocks拡張」を紹介。cf. Grand Central Dispatch/スレッドプール、C++0x のラムダ

3. Apple Inc., “Security Overview,” “Secure Coding Guide,” “Certificate, Key, and Trust Services Programming Guide,” 2010. 電子情報保全に関するプログラミングガイド。

- ローカル通知とプッシュ通知, Event Kit, オーディオセッション/Core Audio, AV Foundation, 位置情報知覚, OpenGL(ES), Core Animation, アクセシビリティ, iPod ライブラリアクセス, Address Book, ドキュメントインタラクション/Quick Look, Bonjour, WebKit, Printing/PDF, Game Kit, Store Kit, iAd, … Predicate/Core Data, Concurrency/Threading, vDSP API, OpenCL, Camera/Display/other devices, … Microsoft C++/CLI(.NET/Mono) …