アプリケーション
プログラミング指南

小林　博久　ー　@khirohis

最低限読んどけ

- iOS Reference Library - 日本語版

  http://developer.apple.com/jp/devcenter/ios/library/japanese.html

- Objective-C 2.0, Cocoaメモリ管理
- HIG, アプリケーションプログラミングガイド

なにを重視するか

- ゲーム：スピード、リソース節約
- ウェブ：セキュリティ、応答速度
- アプリ：落ちない、OSとの協調

小林メソッド

- 今日のために作りました名付けました
- 自分のプログラミングスタイルについて紹介するよ（恥ずかしいけど）
- この後の酒のつまみにでも

コーディングスタイルは統一しよう

- Apple のサンプルに準拠したほうがいいかな
- google-styleguide なんかも参考に
- しかし他人のソースをいじるときはそのスタイルに合わせてあげよう

ソースは決まった順番で記述

- 定数、マクロ定義
- 無名カテゴリ（@property, private method）
- クラスメソッド
- init, dealloc
- アクセッサ
- パブリックメソッド
- デリゲート、データソース
- プライベートメソッド

コードに意味を込めて

- たとえば ++ と += 1 なんかを使い分けたり
- 比較演算は定数的なものを後ろにするとか、日本語的に

コメントは書くな（ウソ）

- 適切な処理単位で関数分けして長い名前を付けて
- テンポラリの変数をガンガン使って長い名前を付けて
- 逆に TODO: はガンガン書こう

ロジック、データ構造はなるったけシンプルに

- キリのいいタイミングでちょいちょいリファクタリング
- 結局拡張するにしても無駄にならないor 無駄が少ない
- もちろん無理の無い程度に拡張性は含めて

property をガンガン使おう

- オブジェクト指向的メリット
- オブジェクト管理を容易に
- 遅延評価でメモリマネージメントを容易に
- お約束を守ることにより比較的容易に安全・強固なアプリケーションが書ける

約束０：オブジェクトの扱い

- 単にリファレンスカウンタの付いたメモリブロックへのポインタだろjk
- 変数に代入するのもメッセージパッシングするのも”参照渡し”を念頭に

約束１：クラス変数、インスタンス変数の命名

- まずはプレフィックス or サフィックス付きで
- 前または後ろにアンスコが一般的かな
- 小林メソッド的には前にアンスコ二つ

約束２：オブジェクトはとにかく retain で

- ただし delegate など相互参照するオブジェクトは assign
- NSString ぐらい copy で…

  いやいやいや

約束３：直接アクセス厳禁！

- 必ずドット演算子で
- 所詮はメソッド呼び出し、パフォーマンスが重要な局面では適宜最適化を

約束４：※ただし init, dealloc は除く

- init, dealloc, アクセッサでは直接アクセス
- 基底クラスにアクセスするようなケースはしょうがないね
- ここばっかりは retain, release を慎重に

ありがちな間違い

- init, アクセッサで retain 忘れ
- dealloc で release 忘れ
- init - alloc 後 autorelease (release) 忘れ
- Mutable からの copy 忘れ

property の書き方をいくつか

- 基本パターン

.h

```
NSHoge *__hage;
@property (nonatomic, retain) NSHoge *hage;
```

.m

```
@synthesyze hage = __hage;
```

外部からは readonly

- 無名カテゴリで

  .h

  NSHoge *__hage;

  @property (nonatomic, retain, readonly) NSHoge *hage;

  .m

  @interface kuma ()

  @property (nonatomic, retain) NSHoge *hage;

  @end

  @synthesyze hage = __hage;

その他

- getter または setter どちらかだけカスタマイズ

  @synthesyze しときつつどっちか記述してしまう

- getter または setter しか使わせたくない

  @dynamic でどっちかだけしか記述しない

property に慣れたら

- dynamic を織り交ぜていこう

特に遅延評価できるようにするとメモリ警告への対応が容易になるよ！

最低限やっとけ

- ビルド → Build and Analyze

  ※静的アナライザを実行 でもいいんだけど

- 実行 → パフォーマンスツールを使って実行 → Leaks
- 全View で ハードウェア → メモリ警告をシミュレート

ちなみに

- こんかい使ったプレゼンテーションアプリケーション、そのうち App Sotre に並ぶかも、よろしく！