■ステータスバー非表示

Info.plist に UIStatusBarHidden（boolean） を追加。

■文字列の結合

NSString *object1 = [NSString stringWithCString:"ABC"];
NSString *object2 = [NSString stringWithCString:"abc"];
NSString *object3 = [object1 stringByAppendingString:object2];
NSString *object4 = [NSString stringWithFormat:@"%@%@", object1, object2];

■ NSString -> CString の変換

```
printf("%s\n", [@" あいうえお " cStringUsingEncoding : NSUTF8StringEncoding ]);
```

■文字列表示方法

```
NSString* sampleText = [NSString stringWithUTF8String:"hello world"];
[sampleText drawAtPoint:CGPointMake(50.0, 60.0) withFont:[UIFont systemFontOfSize:30]];
```

■配列

```
myArray = [[NSMutableArray alloc] initWithCapacity:1];
[myArray addObject:ball];
BallClass *ball=[eventArray objectAtIndex:i];
```

■画像表示方法（２ D）

```
UIImage* background;
...
background = [UIImage imageNamed:@"bg.png"];
...
[background drawAtPoint:CGPointMake(0,0)];
```

■プロトコルオブジェクト取得

```
@protocol()
```

■プロトコルに準拠しているかをチェック

conformsToProtocol

```
if ( ! [receiver conformsToProtocol:@protocol(MyXMLSupport)]  ) {
    // オブジェクトが MyXMLSupport プロトコルに準拠していない
    // MyXMLSupport プロトコルで宣言されているメソッドを実装する
    //  レシーバを期待している場合は、これはおそらくエラー
}
```

■型のイントロスペクション

isMemberOfClass: 特定のクラスのインスタンスかどうかをチェックします。

```
if ( [anObject isMemberOfClass:someClass] )
  ...
```

isKindOfClass: 特定のクラスを継承しているどうかをチェックします。

```
if ( [anObject isKindOfClass:someClass] )
  ...
```

■型チェック

オブジェクトの型宣言は、形式プロトコルを含むように拡張することができます。
型宣言では、プロトコル名はタイプ名の後の不等号括弧内に記述します。

```
- (id <Formatting>)formattingService;
id <MyXMLSupport> anObject;
2 つの型を 1 つの宣言で一体化することができます。
Formatter <Formatting> *anObject;
```

プロパティ宣言属性

■ getter=getterName, setter=setterName

getter= と setter= はそれぞれ、プロパティの get および set アクセサの名前を指定します。デフォルトの名前は、キー値コーディングの表記規約に従います（『Key-Value Coding Programming Guide』を参照）。
getter はプロパティの型と一致する型を返し、引数は取りません。setter メソッドはプロパティの型と一致する型の引数を 1 つ取り、void を返します。

■ readonly

プロパティが読み取り専用であることを示します。デフォルトは、読み取り／書き込みが可能です。
ドット構文を使って値を代入しようとすると、コンパイラエラーが発生します。@implementation では getter メソッドだけが必須であり、@synthesize を使う場合は、getter メソッドだけが合成されます。

■ readwrite

プロパティを読み取り／書き込み可能として扱うべきであることを示します。これはデフォルトで適用されます。
@implementation では getter と setter の両方のメソッドが必須であり、@synthesize を使う場合は getter メソッドと setter メソッドが合成されます。

■ assign

setter で単純代入を使用することを指定します。これはデフォルトで適用されます。
このキーワードを使う場合、アプリケーションはガーベジコレクション (GC) を使っていないと、単純代入は適切な振る舞いではなくなるためコンパイラ警告が発生します。GC でないアプリケーションの場合は、オブジェクトへの警告を避けるために、格納方法の属性の 1 つを明示的に指定する必要があります。
GC を使わないアプリケーションで変数が NSCopying プロトコルを採用していると、その状況での assign の使用は通常は適切でないため、この属性に対して警告が発生します。

■ retain

代入時にオブジェクトに対して retain を呼び出す必要があることを指定します。
この属性は Objective-C オブジェクトに対してのみ有効です（Core Foundation オブジェクトに対しては retain は指定できません。詳しくは Core Foundation を参照してください）。

■ copy

代入にオブジェクトのコピーを使用することを指定します。
コピーは、copy メソッドを呼び出すことによって作成されます。この属性はオブジェクト型に対してのみ有効であり、その場合は NSCopying プロトコルを実装する必要があります。

■ nonatomic

合成されるアクセサが非アトミックになるように指定します。
デフォルトでは、合成されるアクセサはすべてアトミックです。これは、マルチスレッド環境でプロパティへの堅牢なアクセスを可能にすることを意図しています。つまり、getter から返される値や setter を通じて設定される値は、ほかのスレッドが同時に実行しているかどうかに関係なく必ず完全に取得または設定されます。詳細については、パフォーマンスとスレッドを参照してください。