CS193P - Lecture 2

iPhone Application Development

Objective-C
Foundation Framework

日本語解説
小嶋 秀樹（宮城大学）
xkozima@myu.ac.jp

Announcements おしらせ

- Enrollment process is almost done 履修登録（選抜）はほぼ終了
- Shooting for end of day Friday 金曜じゅう（に終了）をめざす
- Please drop the class in Axess if you are not enrolled. 選ばれなかったら登録解除してね

Office Hours

- David Jacobs
  - Mondays 4-6pm: Gates 360
- Paul Salzman
  - Some time. Best to try all possible times until you hit it
  - Some place, probably in Gates. Just come by and yell real loud

いつか．会えるまで何度でもトライしてね．

どこか，たぶん Gates（建物）．
来たら大声で呼んでね．

iPhone SDK

履修学生は Developer Program に招待されます．

- Enrolled students will be invited to developer program
  - Login to Program Portal
  - Request a Certificate　証明書をリクエスト
  - Download and install the SDK

デバイス（実機）の UDID が必要です－詳細は後述．

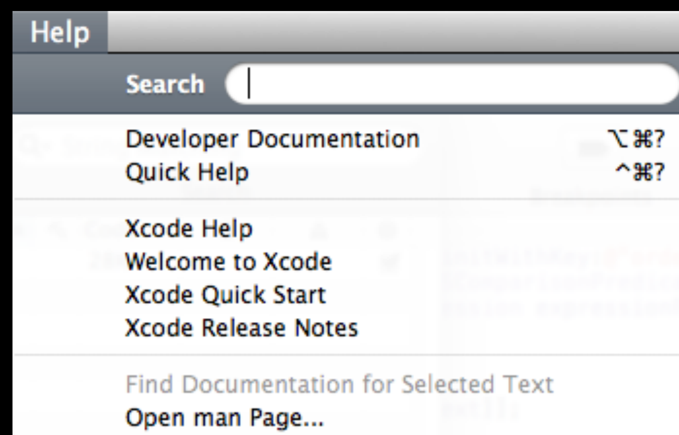
- Will need your Device UDIDs - details to come

聴講生は個別に Developer Program への加入が必要

- Auditors will need to sign up for Developer Program independently
  - Free for Simulator development　シミュレータ上の開発は無料
  - $99 for on-device development　実機上の開発は $99/年

Getting Help ヘルプを受けるには

- The assignment walks you through it 宿題をとおして方法を把握
- Key spots to look ここを見よ
  - API & Conceptual Docs in Xcode
  - Class header files
  - Docs, sample code, tech notes on Apple Developer Connection (ADC) site
    - http://developer.apple.com
    - Dev site uses Google search



Today's Topics 今日の話題

- Questions from Tuesday or Assignments?
- Object Oriented Programming Overview OOP 概論
- Objective-C Language Objective-C 言語
- Common Foundation Classes 一般的な Foundation クラス

Object Basics オブジェクトの基本

OOP Vocabulary OOP 用語

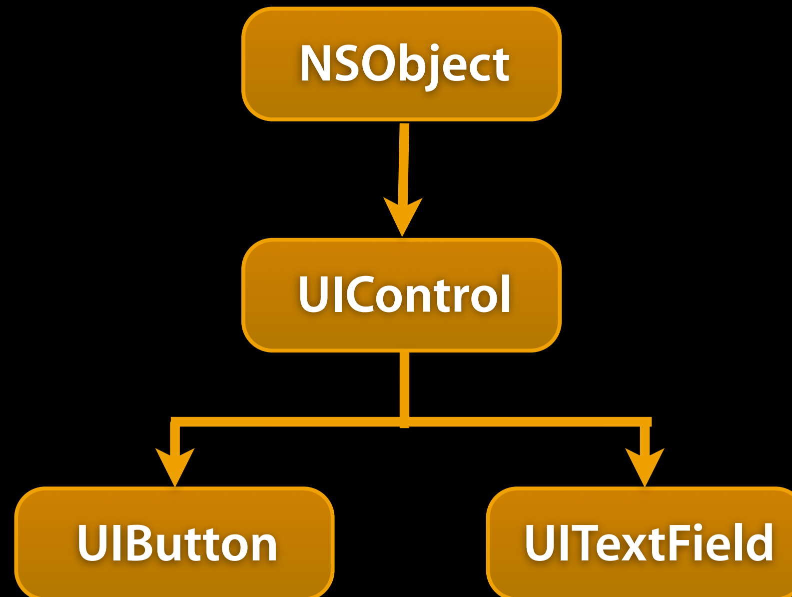
- **Class**: defines the grouping of data and code, the “type” of an object
- **Instance**: a specific allocation of a class
- **Method**: a “function” that an object knows how to perform
- **Instance Variable (or “ivar”)**: a specific piece of data belonging to an object

クラス：データ（インスタンス変数）とコード（メソッド）のまとまりの定義（設計図）．オブジェクトの「型」．
インスタンス：クラスが（メモリ上に）実体化されたもの
メソッド：オブジェクトが実行できる「関数」．
インスタンス変数：オブジェクトに含まれるデータ．

OOP Vocabulary OOP 用語

- Encapsulation カプセル化（インタフェースと内部実装を分離）
  - keep implementation private and separate from interface
- Polymorphism ポリモーフィズム（同じインタフェースをもつ異なるオブジェクト）
  - different objects, same interface
- Inheritance 継承（インヘリタンス）
  - hierarchical organization, share code, customize or extend behaviors 階層構造，コード共有とカスタマイズ・拡張

Inheritance 継承（インヘリタンス）



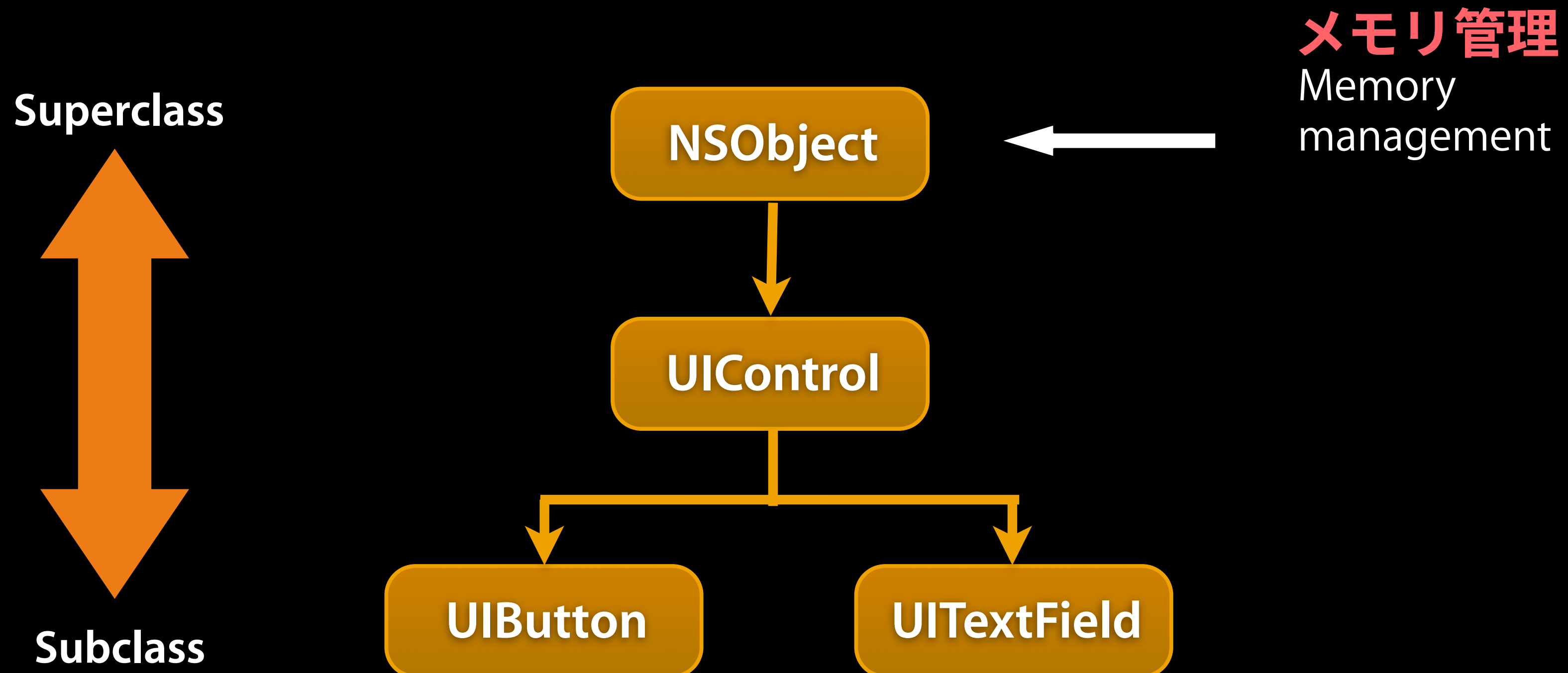
クラス間の階層関係

- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

下位は上位の動作とデータを継承する

下位は上位のメソッドを拡張あるいは上書きしてもよい

Inheritance 継承（インヘリタンス）



クラス間の階層関係

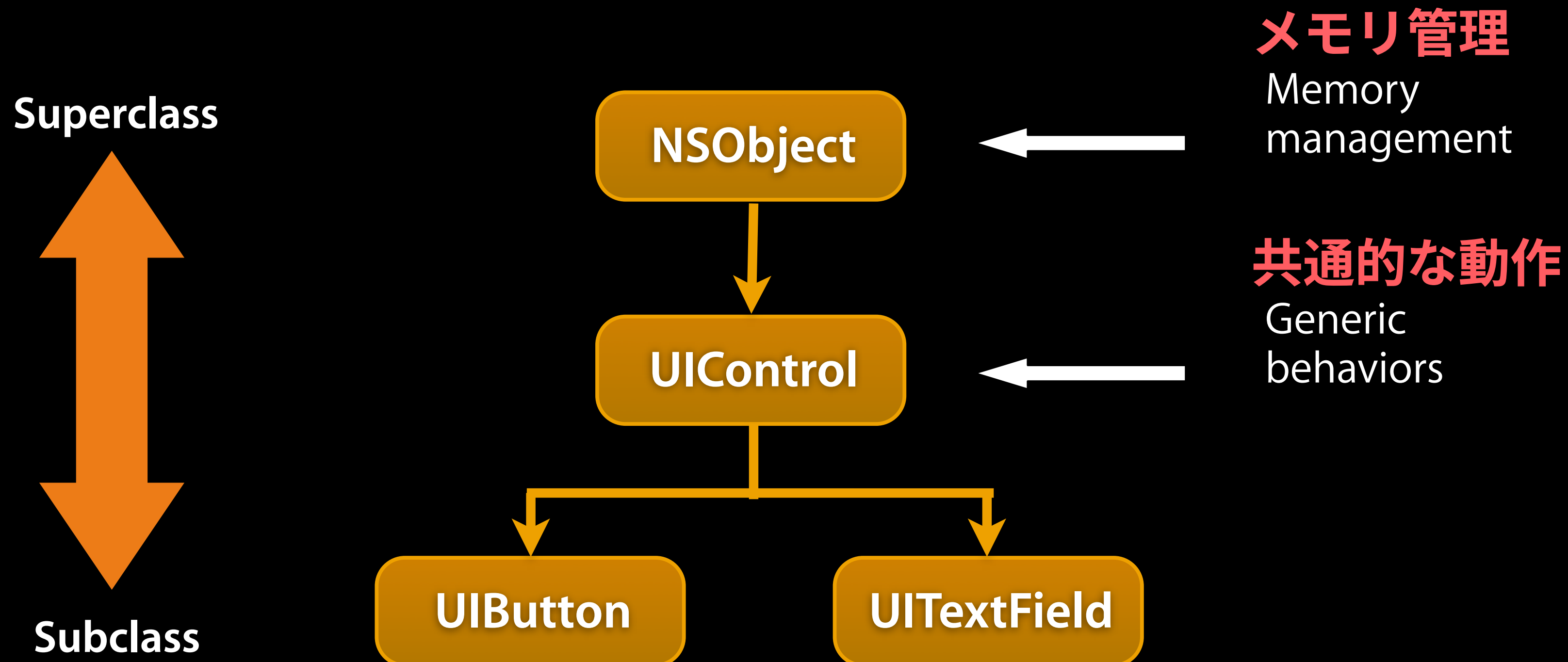
- Hierarchical relation between classes

下位は上位の動作とデータを継承する

- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

下位は上位のメソッドを拡張あるいは上書きしてもよい

Inheritance 継承（インヘリタンス）



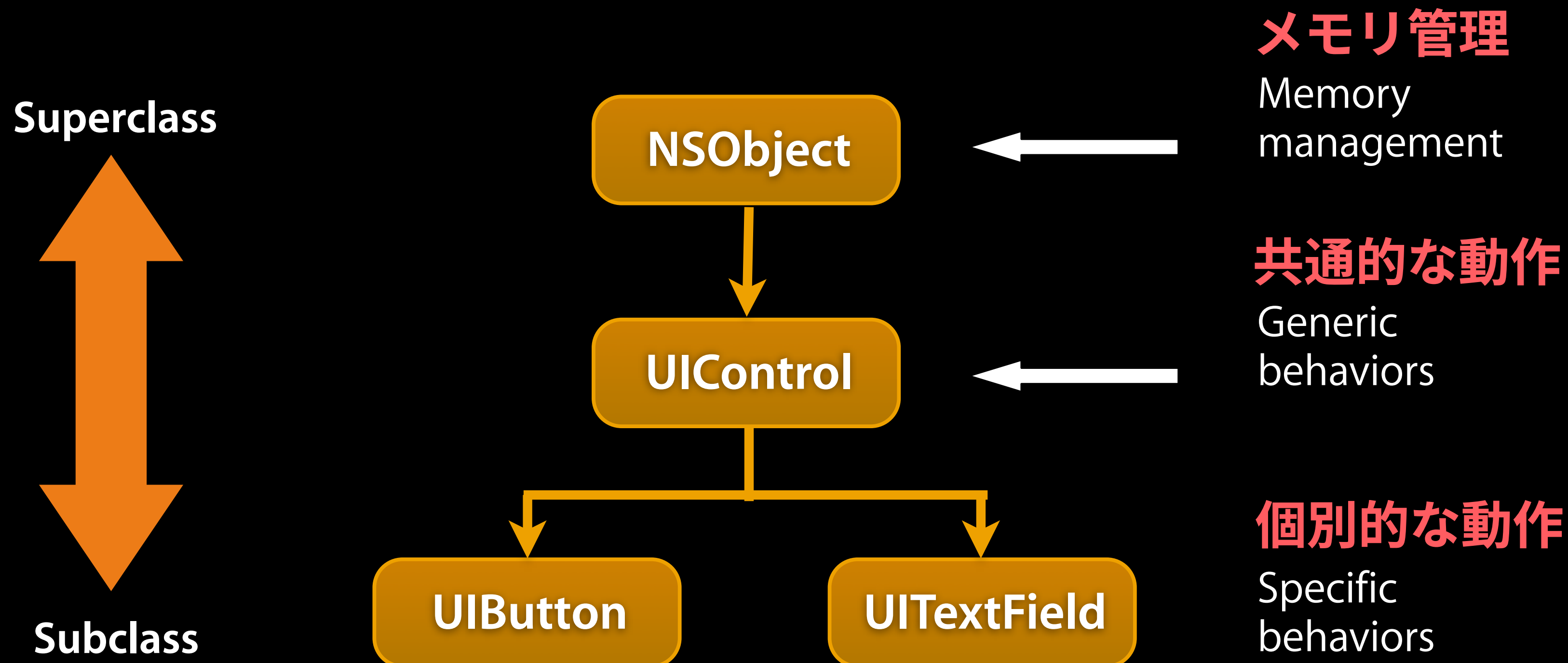
クラス間の階層関係

- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

下位は上位の動作とデータを継承する

下位は上位のメソッドを拡張あるいは上書きしてもよい

Inheritance　継承（インヘリタンス）



クラス間の階層関係

- Hierarchical relation between classes

下位は上位の動作とデータを継承する

- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

下位は上位のメソッドを拡張あるいは上書きしてもよい

More OOP Info? OOP についてもっと詳しく？

- Drop by office hours to talk about basics of OOP 質問に来てね
- Tons of books and articles on OOP 本や論文がいっぱい
- Most Java or C++ book have OOP introductions
- Objective-C 2.0 Programming Language
  - http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC

Objective-C

Objective-C

- Strict superset of C　C のスーパーセット（Cを完全に含む）
  - Mix C with ObjC　C と ObjC をミックスしてよい　さらに C++ も
  - Or even C++ with ObjC (usually referred to as ObjC++)
- A very simple language, but some new syntax　新しい文法も少し
- Single inheritance, classes inherit from one and only one superclass　単一継承（つねに親は１人⇒木構造）
- Protocols define behavior that cross classes　プロトコル：クラス間で共有されるメソッド名の集合
- Dynamic runtime　動的なランタイム
- Loosely typed, if you'd like　弱い型づけ（お好きなら）

Syntax Additions 文法のつけたし

- Small number of additions
- Some new types
  - Anonymous object
  - Class
  - Selectors
- Syntax for defining classes
- Syntax for message expressions

わずかなつけたし

新しい「型」
　無名オブジェクト型（id）
　クラス型
　セレクタ型

クラス定義のための文法
　@interface @implementation

メッセージ式のための文法
　[receiver　message];

Dynamic Runtime 動的なランタイム

- Object creation オブジェクトの生成
  - All objects allocated out of the heap
  - No stack based objects
- Message dispatch メッセージの配達（実行時に配達先を決める）
- Introspection

「内省」
（オブジェクトの属するクラスを実行時に調べることができる）

すべてのオブジェクトはヒープ領域に作られる

スタック領域（ローカル変数）に置かれるオブジェクトはない

OOP with ObjC

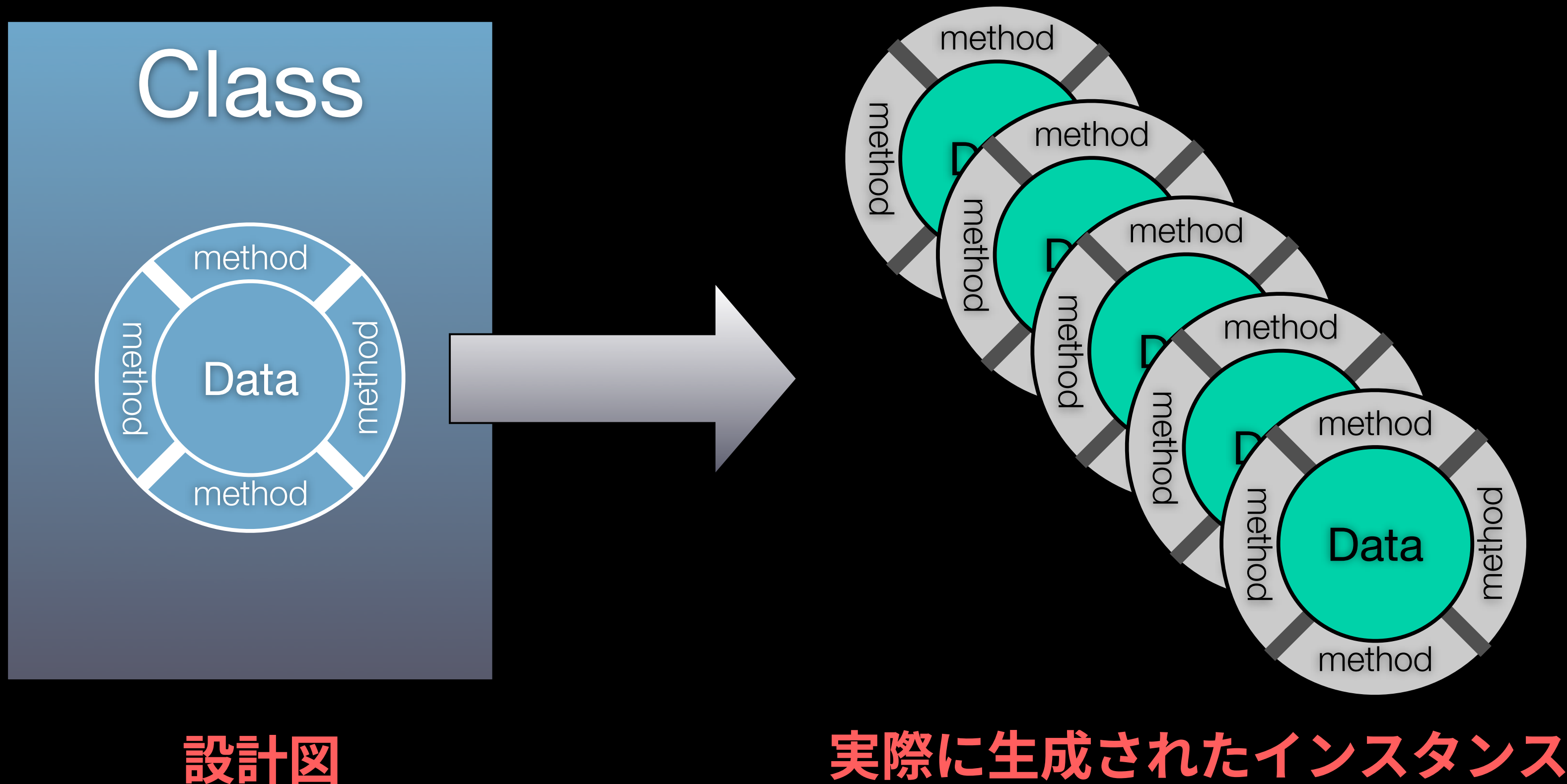
Objective-C による Object Oriented Programming

Classes and Instances クラスとインスタンス

- In Objective-C, classes and instances are both objects
- Class is the blueprint to create instances

どちらも「オブジェクト」

クラスはインスタンスを生成するための「設計図」



設計図

実際に生成されたインスタンス

Classes and Objects クラスとオブジェクト

- Classes declare state and behavior
- State (data) is maintained using instance variables
- Behavior is implemented using methods
- Instance variables typically hidden
  - Accessible only using getter/setter methods

- クラスは「(内部)状態」と「動作」を定義.
- (内部)状態は「インスタンス変数」として保持.
- 動作は「メソッド」として実装.
- インスタンス変数は，通常，隠されていいる.
  getter / setter のみを使ってアクセス可能.

ObjC の視点からみたOOP

OOP From ObjC Perspective

- Everybody has their own spin on OOP 「OOP」の解釈は各人各様
  - Apple is no different Apple も同様
- For the spin on OOP from an ObjC perspective:
  - Read the "Object-Oriented Programming with Objective-C" document
  - http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/OOP_ObjC

ObjeC の視点からみた OOP の解釈：

- "Object-Oriented Programming with Objective-C" を読む？
- "... /OOP ObjC" も読む？

Messaging syntax メッセージの文法

クラスメソッドとインスタンスメソッド

Class and Instance Methods

- Instances respond to instance methods
  - (id)init;
  - (float)height;
  - (void)walk;

（宣言が －で始まる）

インスタンスは
インスタンスメソッドに
応答する

- Classes respond to class methods
  + (id)alloc;
  + (id)person;
  + (Person *)sharedPerson;

（宣言が ＋ で始まる）

クラスは
クラスメソッドに応答する

Message syntax メッセージの文法

受け手

```
[receiver message]
```

引数

```
[receiver message:argument]

[receiver message:arg1 andArg:arg2]
```

Message examples　メッセージの例

人　投票者

```
Person *voter; //assume this exists
```

投票する

```
[voter castBallot];

int theAge = [voter age];
```

```
[voter setAge:21];
```

居留する州

```
if ([voter canLegallyVote]) {
   // do something voter-y
}
```

法的に投票できるか（有権者か）

州への登録　政党　無所属

```
[voter registerForState:@"CA" party:@"Independant"];
```

配偶者

```
NSString *name = [[voter spouse] name];
```

Method definition examples メソッド定義の例

```
Person *voter; //assume this exists
- (void)castBallot;   投票
[voter castBallot];
- (int)age;  年齢（getter)
int theAge = [voter age];
- (void)setAge:(int)age;  年齢（setter)
[voter setAge:21];
- (BOOL)canLegallyVote;   有権者か？
if ([voter canLegallyVote]) {
   // do something voter-y
}
- (void)registerForState:(NSString*)state   州への登録
                   party:(NSString*)party;  政党
[voter registerForState:@"CA" party:@"Independant"];
- (Person*)spouse;  配偶者（たぶん getter)
- (NSString*)name;  名前（たぶん getter)
NSString *name = [[voter spouse] name];
```

Terminology 用語

- Message expression メッセージ式

  [receiver method: argument]

  引数

- Message メッセージ

  [receiver method: argument]

- Selector セレクタ

  [receiver method: argument]

- Method メソッド

  The code selected by a message

  メッセージによって選択されるプログラムコード

Dot Syntax ドット記法

- Objective-C 2.0 introduced dot syntax ObjC 2.0 で導入
- Convenient shorthand for invoking accessor methods

```
float height = [person height];
float height = person.height;
```

アクセサ（getter/setter）を呼び出すための略記法

```
[person setHeight:newHeight];
person.height = newHeight;
```

（上と下は同じ意味）

- Follows the dots... ドットをたどる…

```
[[person child] setHeight:newHeight];
// exactly the same as
person.child.height = newHeight;
```

（上と下は同じ意味）

getter setter

Objective-C Types

Objective-C のデータ型

動的な型づけ，静的な型づけ

Dynamic and static typing

- Dynamically-typed object 動的に型づけられるオブジェクト

```
id anObject
```

  - Just id
  - Not id * (unless you really, really mean it...)

- Statically-typed object 静的に型づけられたオブジェクト

```
Person *anObject
```

（人クラス）　（ポインタになっていることに注意）

- Objective-C provides compile-time, not runtime, type checking
- Objective-C always uses dynamic binding

ObjC はコンパイル時に型チェック（実行時ではない）
ObjC は実行時にバインディング（実行するメソッドの同定）

The null object pointer 空オブジェクト nil

- Test for nil explicitly 明示的に nil かどうかをテスト

```
if (person == nil) return;
```

- Or implicitly または暗黙的に…

```
if (!person) return;
```

- Can use in assignments and as arguments if expected 代入にも引数にも使える

```
person = nil;
[button setTarget: nil];
```

- Sending a message to nil? nil へメッセージを送ると…

```
person = nil;
[person castBallot];
```

BOOL typedef BOOL 型定義

- When ObjC was developed, C had no boolean type (C99 introduced one) ObjC 開発当時，C には boolean 型がなかった
- ObjC uses a typedef to define BOOL as a type

```
BOOL flag = NO;
```

- Macros included for initialization and comparison: YES and NO

```
if (flag == YES)
if (flag)
if (!flag)
if (flag != YES)
flag = YES;
flag = 1;
```

ObjC では BOOL 型を定義
値は YES (1), NO (0)

セレクタはメソッドを（名前によって）同定する

Selectors identify methods by name

- A selector has type SEL　セレクタ型

```
SEL action = [button action];
[button setAction:@selector(start:)];
```

- Conceptually similar to function pointer　概念的には関数へのポインタに似ている

- Selectors include the name and all colons, for example:

```
-(void)setName:(NSString *)name age:(int)age;
```

would have a selector:　このメソッド宣言のセレクタは：

```
SEL sel = @selector(setName:age:);
```

Working with selectors セレクタの扱い

- You can determine if an object responds to a given selector

```
id obj;
SEL sel = @selector(start:);
if ([obj respondsToSelector:sel]) {
  [obj performSelector:sel withObject:self]
}
```

与えられたセレクタに
オブジェクトが応答できるか？

- This sort of introspection and dynamic messaging underlies many Cocoa design patterns

```
-(void)setTarget:(id)target;
-(void)setAction:(SEL)action;
```

このような「内省」や
動的なメッセージ配達が
Cocoa デザインパターンの
根底にある

Working with Classes クラスの扱い

Class Introspection クラスの内省

- You can ask an object about its class

あるオブジェクトに
直属するクラスを尋ねる

```
Class myClass = [myObject class];
NSLog(@"My class is %@", [myObject className]);
```

- Testing for general class membership (subclasses included):

```
if ([myObject isKindOfClass:[UIControl class]]) {
  // something
}
```

あるオブジェクトが
あるクラスの下位に位置するか

- Testing for specific class membership (subclasses excluded):

```
if ([myObject isMemberOfClass:[NSString class]]) {
  // something string specific
}
```

あるオブジェクトが
あるクラスの直属であるか

Working with Objects オブジェクトの扱い

Identity versus Equality

同一性と同値性

- Identity—testing equality of the pointer values

```
if (object1 == object2) {
  NSLog(@"Same exact object instance");
}
```

全く同一のインスタンス（ポインタ比較）

- Equality—testing object attributes

```
if ([object1 isEqual: object2]) {
  NSLog(@"Logically equivalent, but may
         be different object instances");
}
```

論理的に同値のインスタンス２つ
（ただし同一インスタンスかは不明）

-description　description というメソッド

- NSObject implements -description　NSObject で実装（つまり何にでも使える）

```
- (NSString *)description;
```

- Objects represented in format strings using %@　%@ とすると…
- When an object appears in a format string, it is asked for its description　…description を書き出す

```
[NSString stringWithFormat: @"The answer is: %@", myObject];
```

- You can log an object's description with:　ログに書き出すことも可（コンソールに出力）

```
NSLog([anObject description]);
```

- Your custom subclasses can override description to return more specific information

あなたがカスタム化したサブクラスは，親クラスのdescription メソッドを上書きして，より詳しい情報を返すこともできる．

Foundation Classes

Foundation Framework

- Value and collection classes
- User defaults
- Archiving
- Notifications
- Undo manager
- Tasks, timers, threads
- File system, pipes, I/O, bundles

- 数，文字列，配列，辞書など
- アプリ個人設定など
- アーカイブなど
- 通知など
- アンドゥなど
- タスク，タイマ，スレッドなど
- 入出力など

NSObject

- Root class　ルート（最上位）クラス
- Implements many basics　オブジェクトの基本機能を実装
  - Memory management　メモリ管理
  - Introspection　内省
  - Object equality　同値性の判定

NSString

- General-purpose Unicode string support ユニコード文字列
  - Unicode is a coding system which represents all of the world's languages ユニコードとは世界中の言語を表現するコード体系
- Consistently used throughout Cocoa Touch instead of "char *"
- Without doubt the most commonly used class
- Easy to support any language in the world with Cocoa

  - char * に代って Cocoa Touch 全般で一貫して使われる．
  - 疑いもなく最もよく使われるクラス．
  - Cocoa では世界中のどの言語も簡単にサポート．

String Constants

- In C constant strings are　C の文字列定数（char *）

```
"simple"
```

- In ObjC, constant strings are　ObjC の文字列定数

```
@"just as simple"
```

- Constant strings are NSString instances　文字列定数は NSString のインスタンス

```
NSString *aString = @"Hello World!";
```

Format Strings 文字列の書式付生成

- Similar to printf, but with %@ added for objects printf と似ているが オブジェクトのために %@ を追加

```
NSString *aString = @"Johnny";
NSString *log = [NSString stringWithFormat: @"It's '%@'", aString];
```

log would be set to `It's 'Johnny'`

- Also used for logging 動作ログを残すためにも使われる

```
NSLog(@"I am a %@, I have %d items", [array className], [array count]);
```

would log something like:

```
I am a NSArray, I have 5 items
```

NSString

既存の文字列を要求（入力）し，
変更を加えた新しい文字列をつくる（出力）

- Often ask an existing string for a new string with modifications

```
- (NSString *)stringByAppendingString:(NSString *)string;
- (NSString *)stringByAppendingFormat:(NSString *)string;
- (NSString *)stringByDeletingPathComponent;
```

Append = （末尾に）追加

- Example:

```
NSString *myString = @"Hello";
NSString *fullString;
fullString = [myString stringByAppendingString:@" world!"];
```

fullString would be set to `Hello world!`

NSString

- Common NSString methods

一般的な NSString のメソッド

```
- (BOOL)isEqualToString:(NSString *)string;
- (BOOL)hasPrefix:(NSString *)string;
- (int)intValue;
- (double)doubleValue;
```

prefix = 接頭辞
（先頭の部分文字列）

- Example:

```
NSString *myString = @”Hello”;
NSString *otherString = @”449”;
if ([myString hasPrefix:@”He”]) {
      // will make it here
}
if ([otherString intValue] > 500) {
      // won’t make it here
}
```

NSMutableString 可変（可変長）文字列

- NSMutableString subclasses NSString NSString のサブクラス
- Allows a string to be modified 文字列の内容を変更できる
- Common NSMutableString methods

```
+ (id)string;
- (void)appendString:(NSString *)string;
- (void)appendFormat:(NSString *)format, ...;
```

```
NSMutableString *newString = [NSMutableString string];
[newString appendString:@”Hi”];
[newString appendFormat:@”, my favorite number is: %d”,
   [self favoriteNumber]];
```

Collections データの集まり

- Array - ordered collection of objects 配列：順序づけられた…
- Dictionary - collection of key-value pairs 辞書：K-Vペアの集まり
- Set - unordered collection of unique objects 集合：順序なし重複なし
- Common enumeration mechanism 列挙
- Immutable and mutable versions 変更不可 / 変更可能
  - Immutable collections can be shared without side effect
  - Prevents unexpected changes
  - Mutable objects typically carry a performance overhead

- 変更不可の「集まり」は副作用なく共有できる
- （よって）不意の改変を避けることができる
- 変更可能なオブジェクトは（便利だが）パフォーマンスが悪い

NSArray 配列

- Common NSArray methods

```
+ arrayWithObjects:(id)firstObj, ...;  // nil terminated!!!
- (unsigned)count;
- (id)objectAtIndex:(unsigned)index;
- (unsigned)indexOfObject:(id)object;
```

末尾に nil を!!

- NSNotFound returned for index if not found

```
NSArray *array = [NSArray arrayWithObjects:@"Red", @"Blue",
@"Green", nil];

if ([array indexOfObject:@"Purple"] == NSNotFound) {
      NSLog (@"No color purple");
}
```

見つからなかった場合に
インデックスとして
NSNotFound が返る

- Be careful of the nil termination!!!

末尾に nil を置くことに注意
（arrayWithObjects メソッド）

NSMutableArray 可変配列

NSArray のサブクラス
だから NSArray（の機能）
のすべてを使える

- NSMutableArray subclasses NSArray
- So, everything in NSArray
- Common NSMutableArray Methods

```
+ (NSMutableArray *)array;
- (void)addObject:(id)object;
- (void)removeObject:(id)object;
- (void)removeAllObjects;
- (void)insertObject:(id)object atIndex:(unsigned)index;
```

```
NSMutableArray *array = [NSMutableArray array];
[array addObject:@”Red”];
[array addObject:@”Green”];
[array addObject:@”Blue”];
[array removeObjectAtIndex:1];
```

NSDictionary 辞書

- Common NSDictionary methods

```
+ dictionaryWithObjectsAndKeys: (id)firstObject, ...;

- (unsigned)count;

- (id)objectForKey:(id)key;
```

与えられたキーに対応するオブジェクトがなければ nil を返す

- nil returned if no object found for given key

```
NSDictionary *colors = [NSDictionary
      dictionaryWithObjectsAndKeys:@"Red", @"Color 1",
      @"Green", @"Color 2", @"Blue", @"Color 3", nil];

NSString *firstColor = [colors objectForKey:@"Color 1"];


if ([colors objectForKey:@"Color 8"]) {

    // won't make it here

}
```

NSMutableDictionary 可変辞書

- NSMutableDictionary subclasses NSDictionary
- Common NSMutableDictionary methods

NSDictionary のサブクラス

```
+ (NSMutableDictionary *)dictionary;
- (void)setObject:(id)object forKey:(id)key;
- (void)removeObjectForKey:(id)key;
- (void)removeAllObjects;
```

```
NSMutableDictionary *colors = [NSMutableDictionary dictionary];

[colors setObject:@"Orange" forKey:@"HighlightColor"];
```

NSSet 集合

- Unordered collection of objects オブジェクトの順序なしの「集まり」（重複もなし）
- Common NSSet methods

```
+ setWithObjects:(id)firstObj, ...;    // nil terminated
- (unsigned)count;
- (BOOL)containsObject:(id)object;
```

NSMutableSet 可変集合

- NSMutableSet subclasses NSSet NSSet のサブクラス
- Common NSMutableSet methods

```
+ (NSMutableSet *)set;
- (void)addObject:(id)object;
- (void)removeObject:(id)object;
- (void)removeAllObjects;
- (void)intersectSet:(NSSet *)otherSet;
- (void)minusSet:(NSSet *)otherSet;
```

Enumeration 列挙

- Consistent way of enumerating over objects in collections
- Use with NSArray, NSDictionary, NSSet, etc.

```
NSArray *array = ... ; // assume an array of People objects

// old school
Person *person;
int count = [array count];
for (i = 0; i < count; i++) {
    person = [array objectAtIndex:i];
    NSLog([person description]);
}

// new school
for (Person *person in array) {
   NSLog([person description]);
}
```

「集まり」なら何でも

```
// new school
Person *p;
for (p in array) {
   NSLog(...);
}
```

としてもよい

数（整数・実数）

NSNumber

ObjC では，普通，標準的な C の数（整数・実数など）を使うことが多い

- In Objective-C, you typically use standard C number types
- NSNumber is used to wrap C number types as objects
- Subclass of NSValue
- No mutable equivalent!
- Common NSNumber methods

NSNumber は C の数を
オブジェクトに包んだもの

NSMutableNumber はない

```
+ (NSNumber *)numberWithInt:(int)value;
+ (NSNumber *)numberWithDouble:(double)value;
- (int)intValue;
- (double)doubleValue;
```

Other Classes その他のクラス

- NSData / NSMutableData バイトデータ
  - Arbitrary sets of bytes
- NSDate / NSCalendarDate 時刻と日付
  - Times and dates

Getting some objects

メモリ管理について
学ぶまでの間は…

- Until we talk about memory management:
  - Use class factory methods
    - NSString's +stringWithFormat:
    - NSArray's +array
    - NSDictionary's +dictionary
  - Or any method that returns an object except alloc/init or copy.

クラスファクトリメソッドを使って
オブジェクトを生成しよう

もしくは，
alloc/init/copy 以外の
オブジェクトを返すメソッドを使って
オブジェクトを生成しよう

More ObjC Info?

- http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC
- Concepts in Objective C are applicable to any other OOP language

ObjC における諸概念は
他の OOP 言語（C++, Java など）にも適用できる

Questions?