Telerik Academy

Objects and Classes

Mobile apps for iPhone & iPad

Learning & Development

http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

Classes and Objects

Modeling Real-world Entities with Objects

Telerik Academy

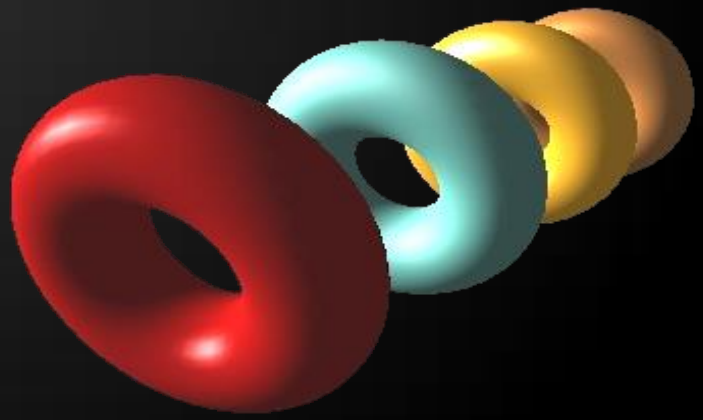
What are Objects?

- Software objects model real-world objects or abstract concepts
  - Examples:
    - bank, account, customer, dog, bicycle, queue
- Real-world objects have states and behaviors
  - Account' states:
    - holder, balance, type
  - Account' behaviors:
    - withdraw, deposit, suspend

Telerik Academy

What are Objects? (2)

- How do software objects implement real-world objects?
  - Use variables/data to implement states
  - Use methods/functions to implement behaviors
- An object is a software bundle of variables and related methods



Objects Represent

- checks
- people
- shopping list

Things from the real world

...

- numbers
- characters
- queues
- arrays

Things from the computer world

What is a Class?

- The formal definition of class:

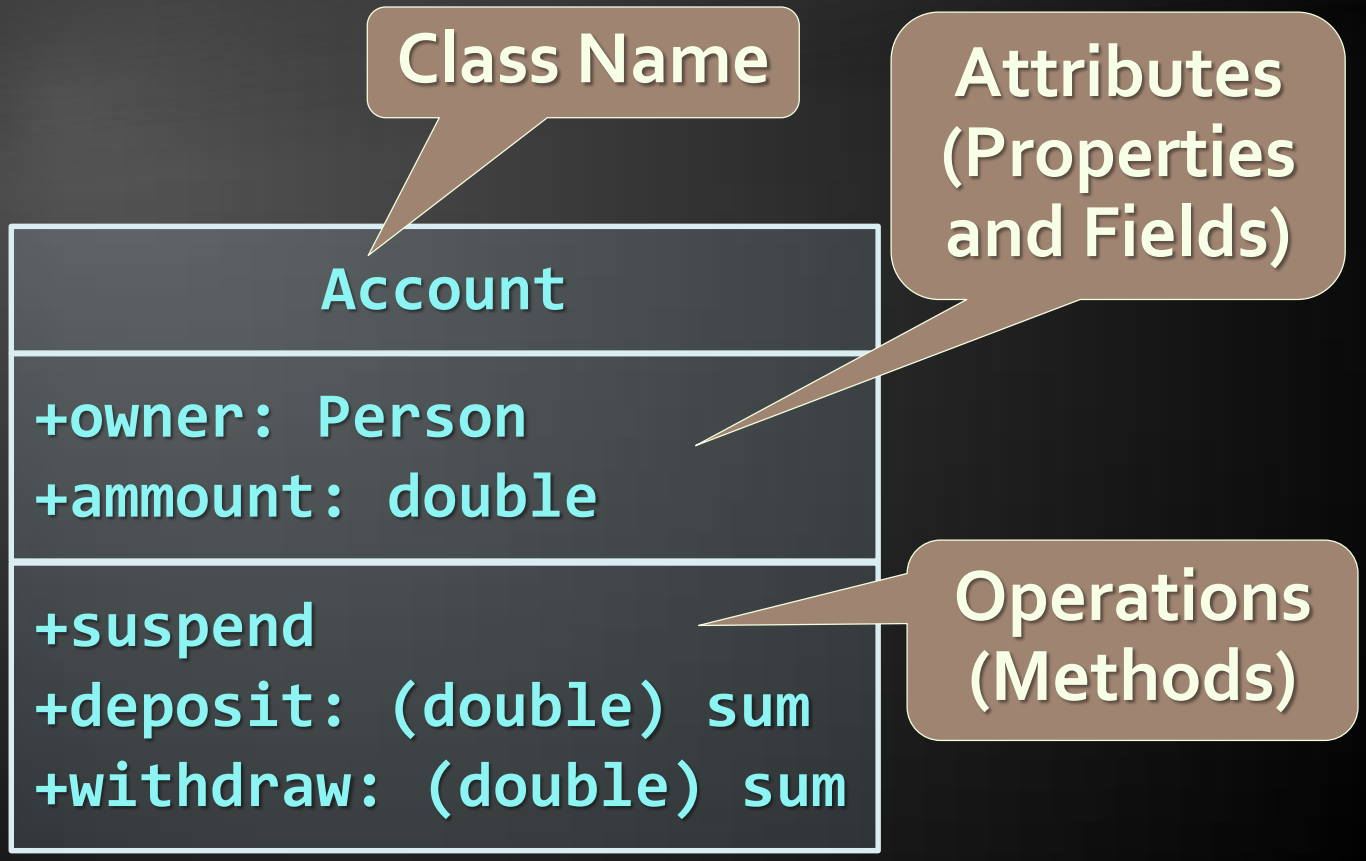
> Classes act as templates from which an instance of an object is created at run time. Classes define the properties of the object and the methods used to control the object's behavior.

Definition by Google

Classes

- Classes provide the structure for objects
  - Define their prototype, act as template
- Classes define:
  - Set of attributes
    - Represented by variables and properties
    - Hold their state
  - Set of actions (behavior)
    - Represented by methods
- A class defines the methods and types of data associated with an object

Classes – Example

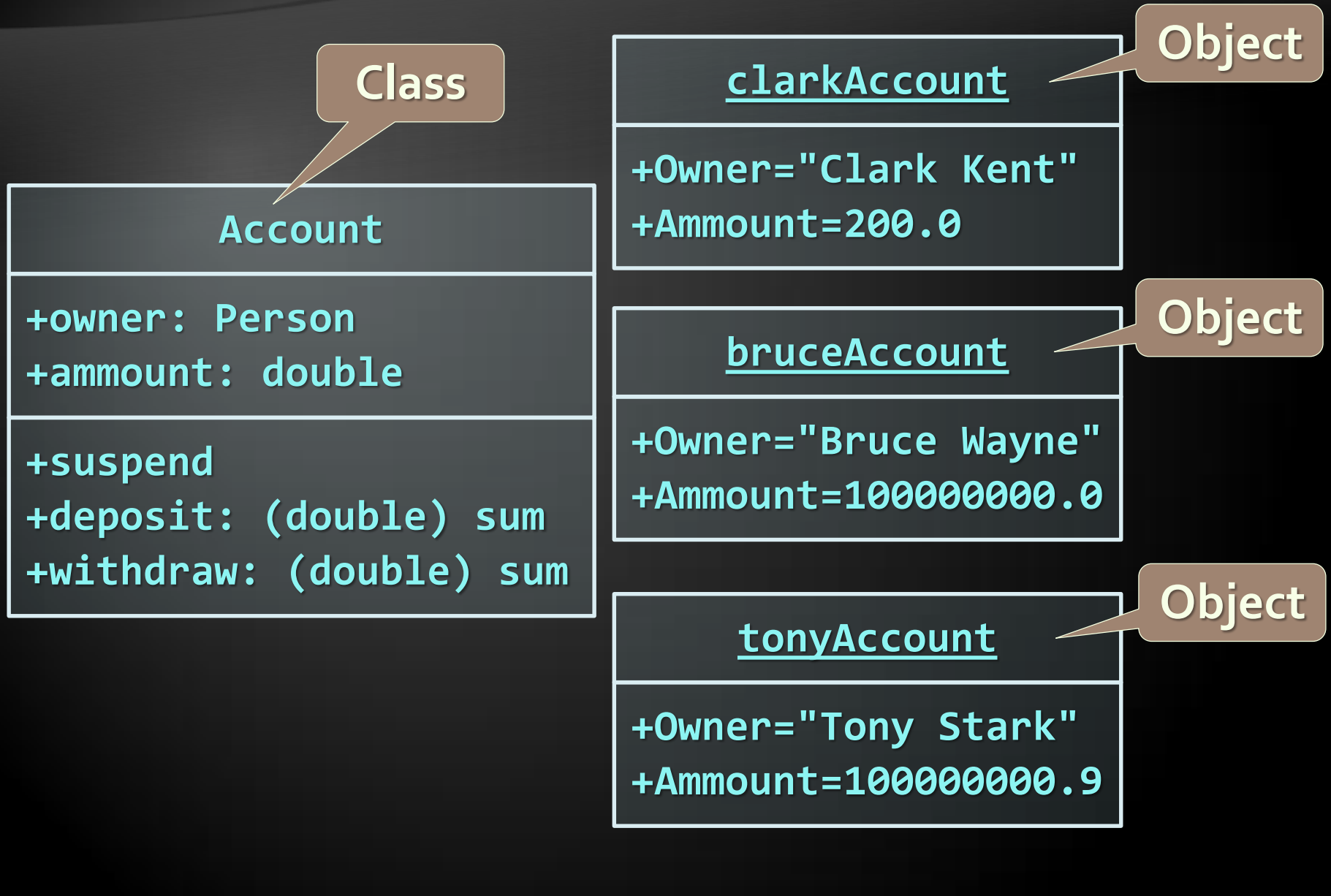


Objects

- An object is a concrete instance of a particular class
- Creating an object from a class is called instantiation
- Objects have state
  - Set of values associated to their attributes
- Example:
  - Class: Account
  - Objects: Ivan's account, Peter's account

Telerik Academy

Objects – Example



Telerik Academy

Object Types and App Memory

App Memory

- Every MAC OS X/iOS application has two places to hold the values of the app (variables and stuff)
  - The Stack and the Heap
- The stack is a fixed-sized stack data structure that holds primitive types and object pointers
  - Only the address, not the object itself
- The heap is the place where all objects live
  - Their addresses are stored on the stack

Primitive Object Types

- Some objects in Obj-C are passed by value
  - They live on the stack and are destroyed when out of scope
  - Their value is copied when passed as a parameter to a method, or when assigned to another object
  - NSInteger, NSUInteger, CGFloat, etc…
  - char, int, float, double, etc…

Telerik Academy

Reference Object Types

- Other objects are passed by reference
  - They live in the Heap, and only their address in the Heap is passed
    - Not copied, only their addresses (references)
  - NSObject, NSArray, NSString, etc…
  - Instances of custom classes are also stored on the heap

Telerik Academy

Primitive and Reference Types

Live Demo

Telerik Academy

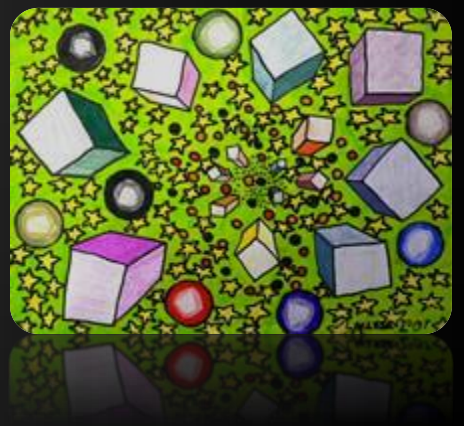
Classes in Objective-C

Using Classes and their Class Members

Telerik Academy

Classes in Objective-C

- Classes – basic units that compose programs
- Implementation is encapsulated (hidden)
- Classes in Objective-C can contain:
  - Fields (member variables)
  - Properties
  - Methods



- Every class in Objective-C has two files
  - Public interface file (the .h file)
  - Implementation file (the .m file)

Fields

- Fields are data members of a class
  - Can be variables and constants (read-only)
  - All fields are private (they can be accessed only from the implementation of the class)
- Accessing a field doesn't invoke any actions of the object
  - Just accesses its value
- Most of the cases they are hidden from the world
  - They live only in the implementation part

Accessing Fields

- Constant fields can be only read
- Variable fields can be read and modified
- Usually properties are used instead of directly accessing variable fields

Telerik Academy

Creating Classes in Objective-C

- Classes in Objective-C consists of two separate files - ClassName.h and ClassName.m
  - ClassName.h contains the public interface of the class
    - Members that are accessible from other objects
  - ClassName.m contains the implementations of the public interface and private members

Telerik Academy

Creating Classes

Live Demo

Properties

- Properties look like fields
  - Have name and type
  - Can contain code, executed when accessed
  - Declared using the @property directive

Properties

- Usually used for encapsulation
  - They control the access to the data fields
  - They validate the given input values
  - Can contain more complex logic
    - Like parsing or converting data

Properties

- Every property has two components called accessors
  - Getter
    - Called when the property is requested

```
double x = shape.x;
```

  - Setter
    - Called when the property value is changing

```
shape.x = 4.5;
```

Properties (2)

- Properties can be either:
  - Read-only
  - Write-only
  - Read-write
  - By default they are read-write

```
@interface
@property (readonly) int size;
@property int capacity;
@end
```

Telerik Academy

Properties

Live Demo

Telerik Academy

Properties: Getters and Setters

- Properties provide a getter and a setter methods
  - Used to execute some code when assigning a value or getting the value of the property
  - By default:
    - The setter method is called `setPropertyName`
    - The getter method is with the name of the property

```
@property int capacity;

-(int) capacity{ … }
-(void) setCapacity { … }
```

Telerik Academy

Properties: Getters and Setters

Live demo

Telerik Academy

Instantiating Objects

Telerik Academy

Instantiating Objects

- Objective-C provides two ways of instantiating objects
  - Using init methods

```
Person *p = [[Person alloc] init];
```

```
Person *p = [[Person alloc] initWithFirstname: @"Peter"];
```

  - Using factory methods

```
Person *p = [Person person];
```

```
Person *p = [Person personWithFirstname: @"Peter"];
```

Telerik Academy

Instantiating Objects

- Objective-C provides two ways of instantiating objects
  - Using init methods

```
Person *p = [[Person alloc] init];
```

```
Person *p = [[Person alloc] initWithFirstname: @"Peter"];
```

  - Using factory methods

```
Person *p = [Person person];
```

```
Person *p = [Person personWithFirstname: @"Peter"];
```

Both are equally used and in best case both should be present

Telerik Academy

Init Methods

Init Methods

- Init methods are used to create objects
  - They are regular methods that have a special meaning in modern MAC/iOS apps
    - ARC checks for methods with prefix "init" and treat their result differently

Telerik Academy

Init Methods Template

- Init methods have a special template to follow:

```
-(id) init
{
  self = [super init];
  if (self)
  {
    //the parent is instantiated properly and can
    //continue instance specific stuff..
  }
  return self;
}
```

Telerik Academy

Multiple Init Methods

- ARC checks for methods with prefix "init"
  - All of the following are valid init methods:

```
@interface Person: NSObject




@ends
```

Telerik Academy

Multiple Init Methods

- ARC checks for methods with prefix "init"
  - All of the following are valid init methods:

```
@interface Person: NSObject
-(id) init;



@ends
```

The default init method

Produces objects without parameters

Telerik Academy

Multiple Init Methods

- ARC checks for methods with prefix "init"
  - All of the following are valid init methods:

```
@interface Person: NSObject
-(id) init;
-(id) initWithFullname: (NSString *) fullname;


@ends
```

init method that takes a single parameter

Handles the role of "method overloading"

Telerik Academy

Multiple Init Methods

- ARC checks for methods with prefix "init"
  - All of the following are valid init methods:

```
@interface Person: NSObject

-(id) init;

-(id) initWithFullname: (NSString *) fullname;

-(id) initWithFirstname: (NSString *) fname
            andLastName: (NSString *) lname;

@ends
```

init method that takes two parameters

The same as the others

Telerik Academy

Multiple Init Methods

Live Demo

Telerik Academy

Factory Methods

Factory Methods

- Factory methods are no more than hidden init methods
  - They are class methods (the message is sent to the class, instead of to a concrete object)

Factory Methods (2)

- By concept, factory methods' identifiers start with the name of the class, i.e.

```
@interface Person: NSObject
+(id) person;
+(id) personWithFirstname: (NSString *) fname;
@end
```

```
@implementation
+(id) personWithFirstname: (NSString *) fname
{
  Person *p = [[Person alloc] init];
  p.firstname = fname;
  return p;
}
@end
```

Telerik Academy

Factory Methods

Live Demo

Telerik Academy

Objects and Classes

Questions?

http://academy.telerik.com

Homework

1. Create classes for an event manager app:
   - Each event has title, category, description, date and list of guests (strings)
   - Event manager can:
     - Create event
     - List all events
     - List events by category
     - Sort events by either date or title

Homework (2)

2. Create an iPhone application using the event manager. Create two views:
   - One for listing events
   - One for creating events
   - Use the class from the previous exercise
   - Research about creating iPhone apps
   - Research about Seques and transitions between different views