Stage 4.2 - Key-Paths and Collection Operators

Scene 1: On-Set

Key-Value coding extends beyond simple keys to *key-paths*. Key-paths are used to follow a path of property relationships for setting and accessing values. As with direct property referencing, the syntax uses the "." operator to create a path to be traversed.

<KEYNOTE SLIDE>
As an example, let's assume we're doing HR at a pet supply company:

We have a **WorkerObj** class that stores employee information and, since the company takes an interest in its employee's pets, **WorkerObj** has a "pet" property of type **petInfo**.

```
@interface PetInfo : NSObject
@property (nonatomic,retain) NSString *petname;
@property (assign) int age;
@end
```

```
@interface WorkerObj : NSObject
@property (nonatomic,retain) NSString *name;
@property (assign) int salary;
@property (nonatomic,retain) PetInfo *pet;
@end
```

Assuming we have a WorkerObj instance, **w1**; using Key-Paths we can get the employee's pet's name using the -valueForKeyPath: method with the path "pet.petname"

```
NSString *petname = [w1 valueForKeyPath:@"pet.petname"]
```

We can set a value using the same keypath with -setValueForKeyPath

```
[w1 setValue:@"Claws" ForKeyPath:@"pet.petname"]
```

in this simple case, the KVC calls are equivalent to using the property syntax w1.pet.petname

NSString *petname = w1.pet.petname

w1.pet.petname = @"Claws";

</KEYNOTE SLIDE>

We'll experiment with these **PetInfo** and **WorkerObj** classes in a moment, but first I'll give you a practical example of how keypath can be useful.

String keys are prone to typos, so it's a good idea to use #defines or constants for your keys.
For example:
<KEYNOTE bold type appears on screen, normal type is read**>**
rather than repeatedly using an NSString "someKey":
dict[@"someKey"]
it's better to use a #define:
#define SOME_KEY @"someKey"
then use the #define:
dict[SOME_KEY]
Another option is to set a constant nsstring variable:
const NSString *kSomeKey = @"someKey"
and use that variable as the key:
dict[kSomeKey]
This will allow us to leverage compiler auto-completion and error checking.

The need to to access values in a nested dictionary would seem to break our scheme of using key constants:
dict[@"subDict1"][@"valueX"]
dict[@"subDict2"][@"valueZ"]

However, using key-paths, we can still use #defines or nsstring constants for nested values:
#define VALUE_X @"subDict1.valueX"
#define VALUE_Z @"subDict2.valueZ"

[dict valueForKeyPath:VALUE_X]
[dict valueForKeyPath:VALUE_Z]

</KEYNOTE>

<u>Scene 2: Screencast</u>

We've implemented the classes from our pet example and initialized three instances: W1,W2 and W3. These three WorkerObj instances have also been populated with sample data.
(go through them)

Example 1:
First we'll use a couple simple key path expressions on W1 that accesses a values:

Example 2:
Change Frank's pet's age
You could get the same results using property dot-syntax, but there are differences in what is possible using the two approaches. For one, since key-paths are strings, they can, like method selectors, be constructed at runtime. Also, since its KVC, the system is more robust. For example, if "petname" happens to be an instance variable rather than a property, "pet.petname" will still work as a keypath but referencing "pet.petname" directly will generate an error.

// example 3
Keys and key-paths are even more powerful when used with collection types: we'll create an NSArray called "employees" which will contain all three WorkerObj instances.

The key "name" for employees gives us an array of employee names without the bother of iterating over the array.
// example 4
Similarly, the keypath "pet.petname" is a convenient way to both iterate over an array and traverse properties to create a new array.

Even more interesting are the keypath collection operators.
// example 5

// example 6
If we log the keypath "@sum.salary" we see that a sum is computed over the salary properties of all the employees.

// example 7

Logging “@max.pet.age” shows us that the oldest pet owned by any employee is 5 years old.

<KEYNOTE SLIDE bold type on slide>

A few of the common collection operators are:
@avg - which calculates an arithmetic average
@max - gives the largest value in the collection
@min - finds the minimum value
@sum - gives you the sum total over the collection

</KEYNOTE>

Code Challenge

1. Write a keypath expression that will calculate the average salary for the WorkerObjs in the employees array.

Ans:
[employees valueForKeyPath:@"@avg.salary"];

2. Write a keypath expression which will set the pet age for WorkerObj instance w2 to 12.

Ans:
[w2 setValue:@12 forKeyPath:@"pet.age"];