Objective-C

Presented By
Abdelrahman Sayed

Java™ Education and Technology Services

Lecture Two

Objective-C

Agenda

- Controlling instance variable scope.
- More on variables.
- Accessors (setters & getters).
- Inheritance.

Java™ Education and Technology Services

[Controlling Instance Variable Scope]

Objective-C

Controlling Instance Variable Scope

@public

- **Accessible from anywhere**

@protected

- **Accessible only by the class that declared it or by any subclass**

@private

- **Accessible only by the class that declared it**

Controlling Instance Variable Scope

- @protected:
  - ➢This is the default for instance variables defined in the **interface section.**
- @private:
  - ➢This is the default for instance variables defined in the **implementation section**.

Example

```
@interface Printer : NSObject{

    @private

        int pageCounter;
        int tonerLevel;

    @protected

        // other instance variables

}

@end
```

Public variable access

- Accessing a public variable through a pointer to its object is done using

```
obj->varName = value;
```

- You will not do the previous point, because you will Never define a public variable.

[More on variables]

static Keyword

- Makes a class variable.
- Static variable should be defined in the implementation part.
- Static variable needs class getter and class setter.

  Why ??!!

```
static int count;
```

const Keyword

- The const keyword tells the compiler that this variable will not be changed through the program.

- Const variable can’t be modified, so it has to be initialized when it is defined.

```
const int x; // doesn't make sense

const int y = 100; // better
```

self Keyword

- The self keyword is used to refer to an object of the class you are working on.

```
@implementation JETSHelloWorld

-(NSString*) hello{

    return @"Hello Objective-C";

}

-(NSString*) sayHelloObjectiveC{

    return [self hello];

}

@end
```

super Keyword

- Super refers to the parent class of the class that you are working on.

```
@implementation JETSHelloWorld

-(NSString*) sayHello{

    return @"Hello Every Body";

}

@end

@implementation ClassB : JETSHelloWorld

-(NSString*) sayHello{

    return @"Hello iOS";
}

-(NSString*) anyMethod{

    [super sayHello];
    [self sayHello];

}

@end
```

[Accessors]

Setters and Getters

- Setters and getter are almost found in all high level OOP languages.
- They are the official way for accessing class data, which are to be hidden, following the data encapsulation concept.

But ,What about if we have large amounts of instance variables ?

Accessors

- Accessor methods (also referred to as *setters* and *getters).*

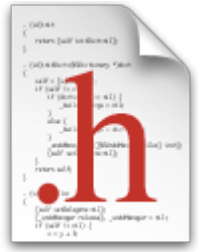
- Objective-C provides a mechanism that automates the creation of accessor methods.

- These are implemented through the use of

  @property and @synthesize.

Defining A Property

@property



- It defines a property.
- Defined within the interface section, outside the curly braces.

```
@property int pro1 , pro2;
```

Synthesize A Property

@synthesize

- It is used to generate the setter and getter methods.
- Used within the implementation section.


```
@implementation JETSHelloWorld

@synthesize pro1 , pro2 ;

@end
```

Accessing Property

Method Access

- The method used to get the value (**the getter method**) has the same name as the property.
  - The getter method for a property called firstName will also be called firstName.
- The method used to set the value (**the setter method**) starts with the word “set” and then uses the capitalized property name.
  - The setter method for a property called firstName will be called setFirstName.

Accessing Property Cont.

- For the property *distance* we can use the methods :
  - The setter method

```
[obj setDistance:value];
```

  - The getter method

```
var = [obj distance];
```

Accessing Property Cont.

Dot Access

- In Objective-C, you can also write the following equivalent expression using the dot operator:

```
obj.pro1;
```

- The general format here is:

```
instanceName.propertyName;
```

Accessing Property Cont.

- You can use a similar syntax to assign values as well:

```
instanceName.propertyName = value;
```

- This is equivalent to writing the following expression:

```
[instanceName setPropertyName : value];
```

[Inheritance]

Defining A Parent

- A parent class can itself have a parent.
- The class that has no parent which is at the top of the hierarchy is known as a root class.
- The root class of all Objective-C classes is NSObject.

- Note: Defining the parent of a class is done in the .h file.

```
@interface ClassName : ParentName
```

Inheritance Overview

- Whenever a new class is defined, the class inherits certain things for example:
  - The (non-private) instance variables and the methods from the parent implicitly become part of the new class definition.
  - Subclass can access these methods and instance variables, as if they were defined directly within the class definition.
- Note that the instance variables that are to be accessed directly by a subclass must be declared in the interface section, and not in the implementation section.

Example

```
@interface ClassA : NSObject{

    int x;
}
-(void) initVar;
@end
@implementation ClassA

-(void) initVar{

    x = 100;
}

@end
```

Example Cont.

```
@interface  ClassB : ClassA

-(void) printVar;

@end

@implementation ClassB

-(void) printVar{

    NSLog(@"x= %i" , x );
}

@end
```

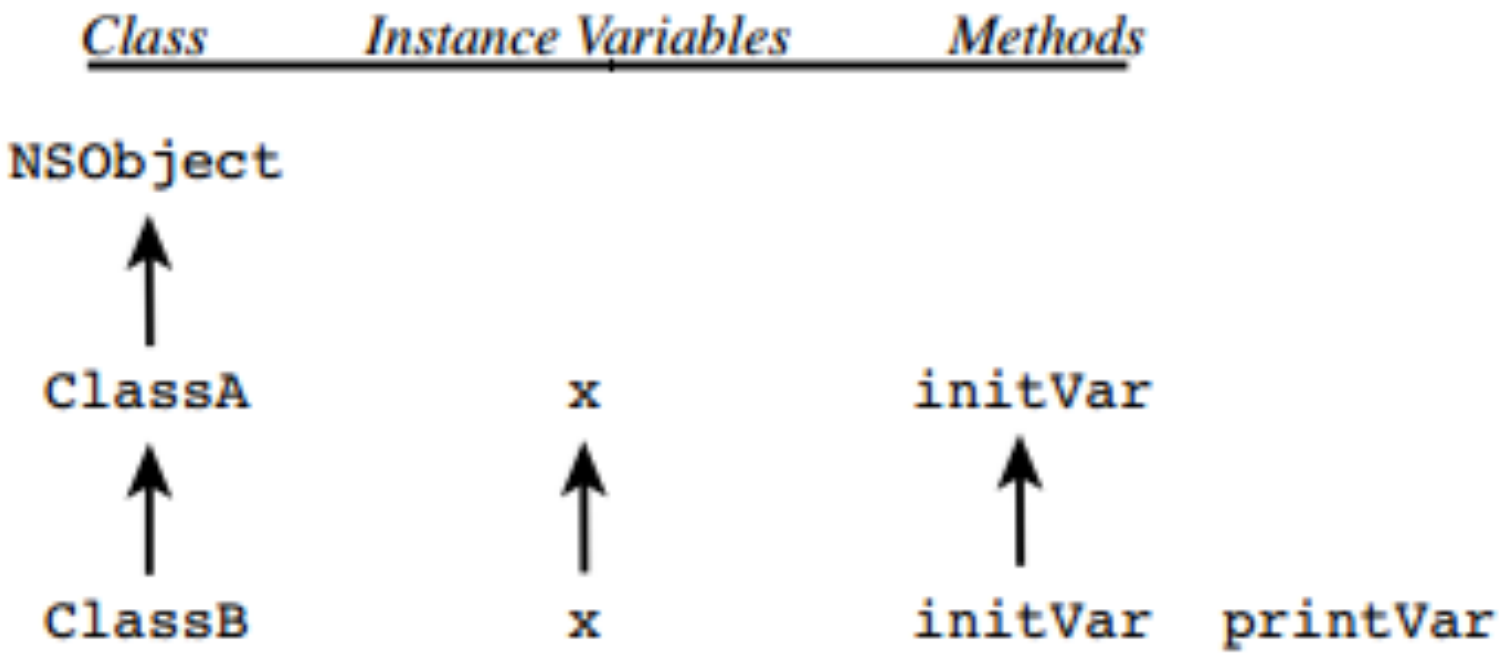
Example Cont.

```
int main(int argc , char * argv[]){

    ClassB *b = [[ClassB alloc] init];
    [b initVar]; // will use inherited method
    [b printVar] // reveal value of x
    return 0;
}
```

Output : x = 100

Example Cont.



Adding New Child Methods

- New methods are added to the child class in the interface section and implemented in the implementation section.
- New methods are able to use
  - ➢ The parent(s) protected and public variables.
  - ➢ Both class/instance methods.

Overriding Parent Methods

- You can't remove or subtract methods through inheritance.
- You can change the behavior of an inherited method by overriding it.
- To override a method you have to define a method with the same:
  - Name
  - Args
  - Return data type

Example

```
@interface ClassA : NSObject{

    int x ; // will be inherited by subclasses

}
-(void) initVar ;
@end
@implementation ClassA

-(void) initVar{

    x = 100;
}

@end
```

Example Cont.

```
@interface  ClassB : ClassA

-(void) initVar;
-(void) printVar;

@end

@implementation ClassB

-(void) initVar { // added method

    x = 200;

}

-(void) printVar{

    NSLog(@"x= %i" , x );
}

@end
```

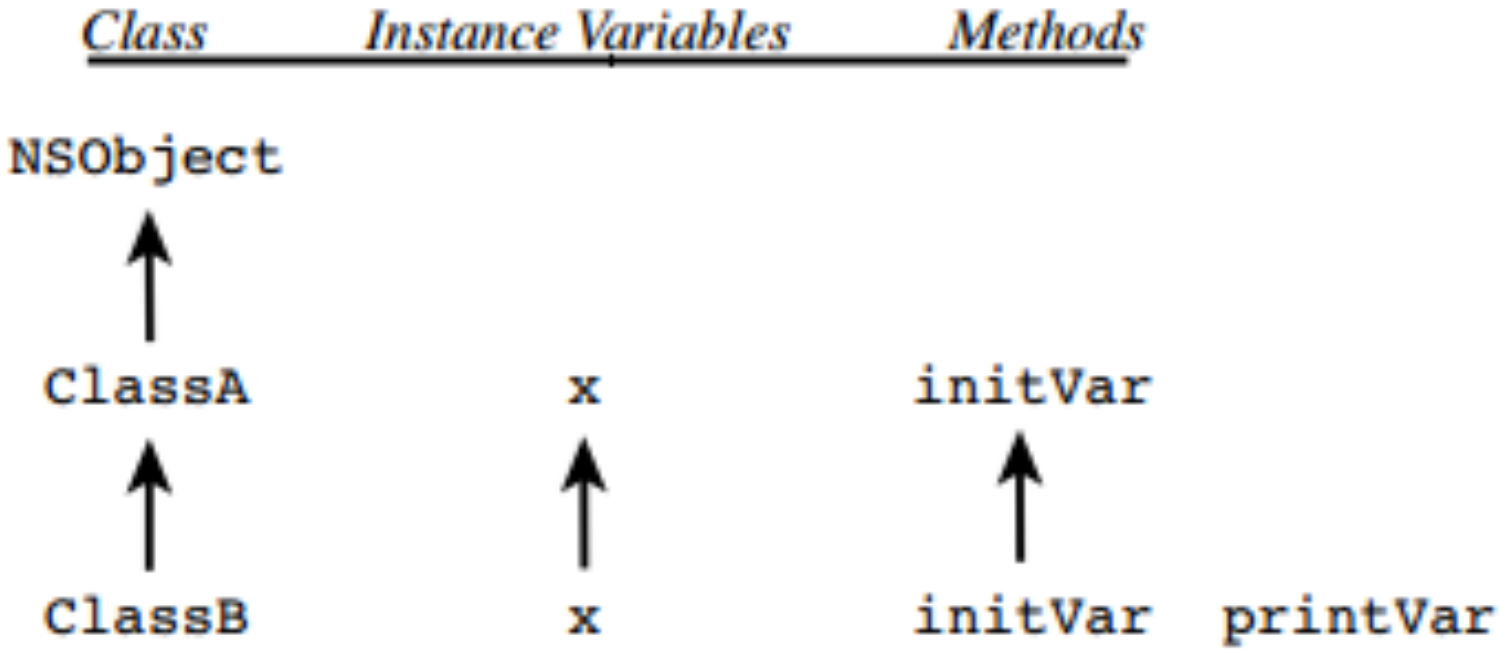
Example Cont.

```
int main(int argc , char * argv[]){

    ClassB *b = [[ClassB alloc] init];
    [b initVar]; // uses overriding method in B
    [b printVar] // reveal value of x
    return 0;
}
```

Output: x = 200

Example Cont.



Overriding the initVar method

Which Method Is Selected ?

- If you have methods in different classes with the same name, the correct method is chosen based on *the class of the receiver* of the message.

```
int main(int argc , char * argv[]){

    ClassA *a = [[ClassA alloc] init];
    ClassB *b = [[ClassB alloc] init];
    [a initVar]; // uses ClassA method
    [b initVar]; // uses overriding ClassB method
    [b printVar] // reveal value of x
    return 0;
}
```

Adding New Variables

- As mentioned before extending a class results in passing variables from parent to child, except variables defined as private.
- Extending a class enables us to add new variables in the interface section to the child class.

[Lab Exercise]

1.Counter

- Create a program that counts the number of instances created from it's class

- Hint: Override the init method

- Try using new method and state what you see ??!!

2.Rectangle

- Create a program that calculates the area of a given rectangle using properties
- The rectangle’s dimension should be obtained from the user.

3.Square

- Create Square class that inherit from Rectangle class and override the area method.

4.Simple calculator

- Create a simple calculator that performs addition, subtraction, multiplication, and division on two numbers entered by user.

- The sequence of application should be as the following:

4.Simple calculator Cont.

Enter the first num :

Enter the second num :

Choose operation you want

1) Add
2) Sub
3) Mul
4) Div

If select add it will display num1 + num 2 = result

If select sub it will display num1 - num 2 = result

And so on

After display the result show this message :

1)Enter new 2 number

2)Exit

5.Friends application

- Class Friend (fid , name , age , phone , email)

// property class

- Class FriendManager
  - -(void) addFriend :(Friend*) friend;
  - -(void) deleteFriend : (int) friendId;
  - -(NSMuatableArray*) getAllFriends;
  - and NSMutableArray to store friends on it

5.Friends application Cont.

- In main function
  - Make two objects of friend and one object from FriendManager.
  - Add two friend objects to FriendManager.
  - Print all friends.
  - Delete one friend.
  - Print all friends again.