# Objective C Classes and Methods

* **Class is created across two different files:**
  + .h file - This is the public interface to the class. This is where you publish methods and properties that you want other classes to know about.
  + .m file -
* **Header file Syntax**
  + #import "AnyHeaderFile.h"
  + // @interface means you are creating a class here.
  + // In objective C you always have to derive from something. If you don’t have anything in particular you want to derive from then just derive from NSObject which is the base class in objective c. is
  + // NSObject contains all NSString, NSArray etc.
  + @interface ClassName: SuperClass
  + // define public properties
  + // define public methods
  + @end
* **Implementation file syntax**
  + #import "YourClassHeader.h"
  + // All the private properties and methods are defined here. This is a class extension.
  + @interface ClassName ()
  + // define private properties
  + // define private methods
  + @end
  + // Now you implement your class here
  + @implementation ClassName { // All instance variables can go here }
  + // implement methods
  + @end
* **Methods**
  + **Syntax**
    - **+/-**
      * **You add + or - to the start.**
      * **- is for instance methods.**
      * **Instance methods are used with instances of that class and hence can use the properties of the class.**
      * **+ is for class methods.**
      * **This is used on class and cannot call the properties of the class at this point of time.**
    - **return type**
      * **what is the method going to return**
    - **method names**
    - **param type**
    - **param name**
    - +/- (return\_type) method\_name\_one:(param\_type) param\_name method\_name\_two:(param\_type) param\_name
* **Example**
  + **Implementation File**
    - #import "DateCalculator.h"
    - @implementation DateCalculator {
    - float \_hisAge; // \_ is added because it is a convention and apple uses this too
    - }
    - - (void)setHisAge:(float)hisAge {
    - \_hisAge = hisAge;
    - }
    - - (float)hisAge {
    - return \_hisAge;
    - }
    - - (BOOL)shouldHeDateIfHerAgeIs:(float)herAge {
    - float minAgeToDate = \_hisAge/2 + 7;
    - return herAge > minAgeToDate;
    - }
    - @end
  + **Header File**
    - #import <Foundation/Foundation.h>
    - @implementation DateCalculator : NSObject
    - - (void)setHisAge:(float)hisAge;
    - - (float)hisAge;
    - - (BOOL)shouldHeDateIfHerAgeIs:(float)herAge;
    - @end
* **Calling a Method**
  + **Allocate memory**
  + **Call a method(init) to set up the object.**
  + ClassName \*myObject = [[ClassName alloc] init];
  + [myObject doIt];
  + [myObject doItWithA:a];
* **As described earlier in this chapter, memory is allocated dynamically for an Objective-C object. The first step in creating an object is to make sure enough memory is allocated not only for the properties defined by an object’s class, but also the properties defined on each of the superclasses in its inheritance chain.**
* **The NSObject root class provides a class method, alloc, which handles this process for you:**

|  |
| --- |
| * **+ (id)alloc;** |

* **Notice that the return type of this method is id. This is a special keyword used in Objective-C to mean “some kind of object.” It is a pointer to an object, like (NSObject \*), but is special in that it doesn’t use an asterisk. It’s described in more detail later in this chapter, in Objective-C Is a Dynamic Language.**
* **The alloc method has one other important task, which is to clear out the memory allocated for the object’s properties by setting them to zero. This avoids the usual problem of memory containing garbage from whatever was stored before, but is not enough to initialize an object completely.**
* **You need to combine a call to alloc with a call to init, another NSObject method:**

|  |
| --- |
| * **- (id)init;** |

* **The init method is used by a class to make sure its properties have suitable initial values at creation, and is covered in more detail in the next chapter.**
* **Note that init also returns an id.**
* **If one method returns an object pointer, it’s possible to nest the call to that method as the receiver in a call to another method, thereby combining multiple message calls in one statement. The correct way to allocate and initialize an object is to nest the alloc call inside the call to init, like this:**

|  |
| --- |
| * **NSObject \*newObject = [[NSObject alloc] init];** |

* **This example sets the newObject variable to point to a newly created NSObject instance.**
* **The innermost call is carried out first, so the NSObject class is sent the alloc method, which returns a newly allocated NSObject instance. This returned object is then used as the receiver of the init message, which itself returns the object back to be assigned to the newObject pointer, as shown in Figure 2-5.**
* **Figure 2-5  Nesting the alloc and init message**
  + 