***ASSIGNMENT 1***

***1pi11is402***

***Name:ARPITHA PAI .K***

***Subclassing an Objective-C Class***

Every object you create in your Cocoa application descends from the 'NSObject' foundation class. The NSObject class identifies properties and methods which apply to all objects. The NSObject class is divided into smaller groups of objects, called subclasses. Objects in these subclasses not only conform to the protocol of NSObject, they are also defined more precisely by the methods that govern their subclass. Every object class inherits from the superclasses above it in the object hierarchy, and also declares the methods which make it a unique class.

A subclass inherits methods and instance variables from its superclass.

A subclass can override a method it inherits from the class it is based on. The class it is based on is called its superclass. If subclassname overrides a method named methodname it inherits from classname but it'd like to use the classname variation on methodname instead of its own, it can do something like:

Code:

[super methodname];

Subclassing is just a subclass of another class and it inherits behavior and members of itsparentclass.  
  
. Where does the description method come from? Every class has exactly one superclass — except for the root class of the entire hierarchy:NSObject. That means, at minimum, every class inherits from NSObject. NSObject implements a method named description.

Sending the description message to an NSObject returns an NSString containing information about that instance. By default, that string is the object’s class and its address in memory. A subclass of NSObject, like NSString, will override this method to return something that does a better job describing an instance of that subclass. For NSString, description just returns the string itself since that is the best way to describe an NSString instance.

***Protocols an Objective-C Class***

Objective-C allows you to define protocols, which declare the methods expected to be used for a particular situation

The basic syntax to define a protocol looks like this:

|  |
| --- |
| @protocol ProtocolName |
| // list of methods and properties |
| @end |

Protocols can include declarations for both instance methods and class methods, as well as properties.

Protocols are a way of enforcing certain methods to be utilized, regardless of the actual class the object is part of, thus ensuring that a certain form of method template is implemented. Basically, a way of declaring an interface to an object while concealing it's class. This is why we write a protocol in a similar way to how we would write an interface declaration.