|  |  |  |
| --- | --- | --- |
|  | **PROTOCOLS** | **SUBCLASSING** |
| **WHAT?** | * A protocol is the same thing as an interface in Java. * A protocol specifies what methods a class will implement. | * 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. |
| **WHEN TO USE?** | * **Use protocols to declare a set of methods that a class must implement.** | * Subclassing is more useful if you want to alter the behaviour of only certain instances, and retain the original method for others. |
| **ADVANTAGES** | * To declare methods that others are expected to implement. * To declare the interface to an object while concealing its class. * To capture similarities among classes that are not hierarchically related. | * Generally use subclasses on third-party and private frameworks rather than a category. * Subclassing is a better option for customization |
| **DISADVANTAGE** | * Protocols are generally reserved for specific patterns, such as the delegation pattern. | * Subclassing can only widen an interface, you can't narrow it. This leads to leaky abstractions. |
| **SYNTAX** | In protocol.h   |  | | --- | | @protocol ProtocolName | | // list of methods and properties | | @end |   In interface file .h  @interface class:NSObject <protocolName>  {  }  @end | @interface subclassname:classname  {  //instance variables that subclassname has but classname lacks go here  }  //methods that subclassname has and classname may or may not will go here. If both have it, subclassname’s implementation override’s classnames’s as long as an instance of subclassname is references .  @end |