TP 4

À la fin de ce TP :

- Faire une archive contenant les projets Xcode des exercices
- Envoyer l'archive à ahumiliere@captaintrain.com avec l'objet : `[DANT] TP 4 – Prénom Nom`
- Si le TP est fait à plusieurs, préciser les noms et adresses mail de chacun

Le TP se base sur un nouveau projet Xcode à créer, de type « Single View Application ».

Exercice 1

- Dans Interface Builder, ajouter un Label pour afficher l'heure courante.
- Lui appliquer un style, le positionner avec des contraintes Autolayout et définir une valeur initiale à **00:00**.
- Ajouter un IBOutlet vers le label dans la classe `ViewController`.

Exercice 2 : NSDateFormatter

- Ajouter une classe `Clock` et définir une méthode renvoyant une instance de `NSDate` correspondant à la date du moment (voir la documentation de `NSDate`).
- Les propriétés Swift existent sours forme de getter (*computed properties*). Remplace la méthode `currentTime` par une *computed property*.

```
var currentTime: NSDate {
   return NSDate()
}
```

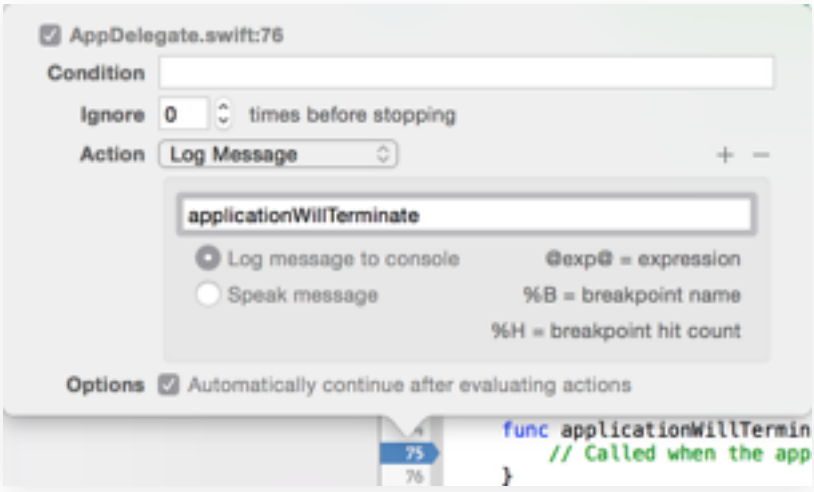
- Déclarer une propriété `clock` dans la classe `ViewController` et lui affecter par défaut une instance de `Clock`.
- Mettre à jour `viewDidLoad` pour afficher la date dans le label en utilisant la propriété `currentTime` de `Clock`.
- Lancer l'application et remarquer que `NSDate` est difficilement lisible.
- Utiliser `NSDateFormatter` pour afficher la date dans un format facilement lisible par un humain.

```
let formatter = NSDateFormatter()
formatter.timeStyle = .ShortStyle
timeLabel.text = formatter.stringFromDate(clock.currentTime)
```

Exercice 3 : AppDelegate

- Dans le simulateur, mettre l'application en Background (⇧⌘**H**). Attendre jusqu'à ce que l'heure change dans la barre de statut. Ouvrir l'application en cliquant sur l'icône. L'heure n'est pas à jour.
- Afficher le menu Multitasking (deux fois ⇧⌘**H**). Glisser vers le haut pour fermer l'application. La relancer. L'heure est maintenant correcte. L'heure courante est calculée dans la méthode `viewDidLoad`. La première fois que le `ViewController` est affiché pendant la vie de l'application.

- Ajouter un print dans `viewDidLoad`. En surveillant la console, lancer l'application, la mettre en background, la rouvrir, la killer, la rouvrir.
- La classe `ViewController` hérite de `UIViewController` (voir la documentation). Essayer d'overrider la méthode `viewWillAppear` et y transposer le contenu de la méthode `viewDidLoad`.
- En surveillant la console, lancer l'application, la mettre en background, la rouvrir. En déduire à quel moment est appelée la méthode `viewWillAppear`. Ce n'est pas non plus la méthode appropriée.
- Regarder le contenu de la classe `AppDelegate`. Se documenter sur le protocole `UIApplicationDelegate`.
- Au lieu d'ajouter un prince dans toutes les méthodes de AppDelegate, utilise Xcode pour ajouter des breakpoints qui continuent automatiquement après avoir affiché un message dans les logs.



- Reproduire le scénario précédent et observer la console.
- Déterminer quelle méthode de `AppDelegate` est la plus appropriée pour ce qu'on veut faire (mettre à jour l'heure quand l'application s'affiche).
- `AppDelegate` n'est clairement pas adapté pour effectuer des opérations sur les views (cf. pattern MVC).

Exercice 4 : NSNotificationCenter

- Etudier la documentation de la classe `NSNotificationCenter` et notamment les méthodes `defaultCenter` et `addObserver:selector:name:object:`.
- Enregistrer `ViewController` comme un observer de la notification `UIApplicationWillEnterForegroundNotification` dans `viewDidLoad`.

```
NSNotificationCenter.defaultCenter().addObserver(self,
    selector: "updateTimeLabel",
    name: UIApplicationWillEnterForegroundNotification,
    object: nil)
```

- Implementer la méthode `updateTimeLabel` pour mettre à jour le Label.
- Réécrire `viewWillAppear` pour utiliser la méthode `updateTimeLabel`.
- Lancer l'application et reproduire le scenario précédent. Observer que l'heure se met à jour quand on rouvre l'application.

- Quand on enregistre une classe comme observer, il faut la désenregistrer quand l'application est quittée. Implémenter la méthode `deinit` :

```
deinit {
   NSNotificationCenter.defaultCenter().removeObserver(self)
}
```

Exercice 5 : NSTimer

- On va maintenant chercher à faire que l'application soit *en permanence* à l'heure. Nous allons utiliser `NSTimer`.
- Ajouter une nouvelle propriété sur `ViewController` pour un `NSTimer`. Le timer sera déclaré en optionnel car l'initialiseur `ViewController` ne va pas initialiser la propriété.
- Etudier la documentation de `NSTimer`, notamment la méthode `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:`.
- Remplacer l'enregistrement en tant qu'observer dans `viewDidLoad` au profit de la création d'un `NSTimer` qui appellera `updateTimeLabel` toutes les secondes.
- Modifier le style du `NSDateFormatter` pour que le label affiche les secondes (choisir le style approprié).
- Modifier l'implémentation de `deinit` pour s'adapter à l'usage du `NSTimer`.

```
deinit {
   if let timer = self.timer {
      timer.invalidate()
   }
}
```

- Lancer l'application

Exercice 6

- Apple fournit aux développeurs les *iOS Human Interface Guidelines* ou *HIG* qui décrivent les bonnes pratiques en terme d'expérience utilisateur.
- Il est recommandé de ne pas masquer la barre de statut, mais la décision est laissée au libre choix du développeur. Ici, nous allons la masquer pour éviter le doublon avec notre label.
- Il est possible de la masquer pour un seul controller avec la méthode `prefersStatusBarHidden` ou de le faire pour toute l'application dans le fichier Info.plist : `Status bar is initially hidden` à `YES` et `View controller-based status bar appearance` à `NO`.

- Explain the significance of an app's main property list file for configuration.
- Lancer l'application et observer le résultat.
- Changer l'orientation de l'appareil dans le simulateur (⌘→ ou ⌘←) et observer comment l'interface s'adapte.
- Il est possible de choisir quelles orientations sont disponibles pour l'application dans ce même Info.plist.