

## TP n° 1

### Prise en main, première application Kotlin

#### Important things

- On fera appel à chaque TP. N'oubliez pas de soumettre les justificatifs d'absence.
- Inscrivez-vous sur le Moodle de U Paris Cité <https://moodle.u-paris.fr/course/view.php?id=10712> Programmation des composants mobiles
- Pour tous les TPs vous avez besoin d'un Android Studio fonctionnel sur votre ordinateur portable. Si ce n'est pas encore fait allez vite sur <https://developer.android.com/studio/> et faites l'installation.
- Si vous n'avez pas d'ordinateur portable compatible parlez de toute urgence à l'enseignant.
- Vous devez avoir installé la dernière version (Dolphin 2021.3.1), ne la mettez plus à jour jusqu'à l'examen et la soutenance du projet!
- Pour le prochain TP il est très conseillé d'apporter aussi un téléphone ou une tablette sous Android et un câble USB pour le connecter à votre ordinateur portable. Surtout si votre ordinateur n'est pas très puissant.

#### Ressources en ligne

- Cours et TP sur Moodle (pour cette séance vous utiliserez Cours 1 et TP 1)
- Ressources sur <https://developer.android.com/>
- Documentation de Kotlin sur <https://kotlinlang.org/docs/home.html>
- Un million d'autres sites, à utiliser avec précaution

## 1 Une application Kotlin

**Remarque** Android Studio n'est pas vraiment destiné à développer les programmes Kotlin en ligne de commande. On le fera aujourd'hui quand même. Le Studio sera lent (surtout si c'est la première compilation quand Gradle télécharge et installe des tonnes de bibliothèques), soyez patients.

### 1.1 Hello

Dans Android Studio, créez l'application TP01. Suivez les instructions données en cours 1 pour créer une application à exécuter sur un terminal.

Pour voir si votre application est correctement générée dans AndroidStudio mettre dans la fonction `main()` l'affichage de "Hello" et essayer de l'exécuter.

### 1.2 Classe Point

Définir la classe `Point` qui correspond à un point sur le plan avec les coordonnées entières `x` et `y`. La classe `Point` sera déclarée comme `data class`, c'est-à-dire une classe composée uniquement de données. La méthode `toString()` de `data class` est automatiquement redéfinie par Kotlin.

### 1.3 classe Rectangle

La classe `Rectangle` représente un rectangle sur le plan dont les sommets ont les coordonnées entières. Un tel rectangle est définie de façon unique par deux de ses sommets, le sommet `p` en bas à gauche et `q` en haut à droite.

Écrire la classe `Rectangle` avec un constructeur qui prend comme paramètres les points `p` et `q`. Les valeurs par défaut pour les deux points sont respectivement  $(0,0)$  et  $(1,1)$ . Redéfinir la méthode `toString()` de `Rectangle` pour qu'elle retourne `"p=$p q=$q"`.

Définir dans `main()` un `Array` composée de plusieurs rectangles. Pour définir les rectangles utilisez aussi les valeurs par défaut (un rectangles avec `p` et `q` par défaut, un rectangle avec la valeur `p` par défaut et `q` défini à la création de rectangle, un rectangle avec la valeur `q` par défaut et `p` défini à la création de rectangle, etc.).

**Remarque.** Pour définir un `Array` vous utiliserez la fonction `arrayOf<T>()` fournie par Kotlin, qui prend en paramètre les objets de la classe `T` que nous voulons mettre dans `Array`. Par exemple `arrayOf("ala", "tala", "mala")` retourne un `Array` de trois `Strings`.

Le bloc `init{ }` d'une classe s'exécute quand on appelle un constructeur. Ajouter dans `Rectangle` le bloc `init` dans lequel on vérifie si `p.x < q.x && p.y < q.y` (c'est-à-dire que les points `p` et `q` sont situés comme indiqué au début de l'exercice). Si ce n'est pas le cas alors lancer l'exception `IllegalArgumentException()` (comme en java, avec `throw` mais sans `new`).

Ajouter dans `main()` la création d'un rectangle qui provoquera lancement de l'exception `IllegalArgumentException()`. Ajouter `try catch` pour intercepter l'exception.

Définir la classe `Carre` dérivée de `Rectangle`. Le constructeur de `Carre` prend deux paramètres, le point `p` comme dans le constructeur de `Rectangle` et `l : Int` – la longueur d'un côté du carré. dans le bloc `init{ }` vous vérifiez si `l > 0`, sinon vous lancez l'exception `IllegalArgumentException`. Insérez un ou deux carrés dans votre tableau.

Dans la classe `Rectangle` ajouter la méthode `surface` qui retourne la surface de rectangle. Dans `main` afficher la surface de chaque rectangle qui est dans `Array` défini précédemment.

**Indication.** Si `r` est un rectangle alors `"${r.surface()}"` est un `String` qui contient le résultat de la méthode `surface()` suivi par des espaces.

Soit `w` une valeur telle que `Array` contient aussi bien des rectangle dont la surface est  $> w$  que des rectangles dont la surface est  $\leq w$ . Créer dans `main()` une liste qui contient tous les rectangles de `Array` dont la surface est  $> w$ .

**Indication.** Pour construire cette liste vous devez utiliser la méthode `filter` de `Array` (en fait `filter()` est défini pour chaque `Collection`).

Si nous avons `Array<T>` qui contient les objets de la classe `T` alors le paramètre de `filter` est de type

`T -> Boolean`

c'est-à-dire une fonction qui prend un objet `T` en paramètre et retourne un booléen. `filter` applique cette fonction à chaque élément de `Array` et retourne la liste de tous les objets `T` pour lesquels la fonction donne la valeur `true`. Vous devez passer comme paramètre de `filter` une lambda-expression appropriée. Pour tester affichez les surfaces de tous les rectangles de la liste obtenue par `filter()`.

Ajouter dans la classe `Rectangle` la méthode `intersects()` qui prend un paramètre un autre rectangle `r` et retourne `true` si `r` intersecte le rectangle `this`.

Dans `main` afficher tous les couples de rectangles de `Array` qui s'intersectent. Vous devez examiner chaque couple de rectangles une seule fois, et ne jamais afficher l'information peu utile qu'un rectangle s'intersecte avec soi-même.

**Indication.** Si `rect` est un `Array` alors, sans surprise, `rect[i]` est *i*-ème élément de `rect`.