Université de Paris **Programmation des composants mobiles**
M1 **Année 2021-2022**

TP n° 1

Prise en main, première application Kotlin

IMPORTANT

Inscrivez-vous sur le Moodle de U de Paris `moodle.u-paris.fr` Programmation des composants mobiles (Android)

Avant de faire ce TP vous avez besoin d'un Android Studio fonctionnel (`Android studio`+émulateur ou `Android studio`+smartphone ou tablette — sur une machine des salles 2031 ou 2032 ou sur votre ordinateur personnel).

Pour l'installation sur les machines de l'UFR suivez notre Guide d'installation ("TP0").

Pour ceux/celles qui ne peuvent pas accéder à moodle, exceptionnellement, le transparents du premier cours se trouvent sur `www.irif.fr/~zielonka/Enseignement/Android/2021/COURS/01/android1.pdf` Et les TP00 et TP01 se trouvent sur `www.irif.fr/ zielonka/Enseignement/Android/TP/00/TP00.pdf` et `www.irif.fr/ zielonka/Enseignement/Android/TP/01/TP01.pdf`.

Ce TP s'appuie principalement sur le cours 1.

Documentation de Kotlin se trouve `kotlinlang.org/docs/home.html`

1 La première "vraie" application

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

1.1 Hello

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

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 déclaré comme `data class`. `data class` est une classe composée uniquement de donnée. La méthode `toString()` de `data class` est automatiquement redéfinie par Kotlin.

1.3 distance

Écrire la fonction `distance` qui prend comme argument deux points et calcule la distance entre eux. On suppose que la distance entre les point `p` et `q` est $|p.x - q.x| + |p.y - q.y|$ (donc c'est un `Int`).

1.4 classe Rectangle

La classe `Rectangle` représente un rectangle sur le plan dont les sommets ont les coordonnées entières. Un tel rectangle est définie de faço,n 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 `arryOf<T>( )` fournie par kotlin, qui prend en paramètre les objets de la classe `T` que nous voulons mettre dans `Array`. La fonction `arrayOf()` prend un nombre variable d'argument et le type `T` peut dans la plupart de cas être déduit par compilateur. Par exemple `arrayOf("ala","tala","mala")` retourne un `Array` de trois `String`s.

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.

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 $<= w$.

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 le paramètre de `filter` une lambda expression appropriée.

Pour tester afficher 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`.

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`.