Atelier 2 – Les expressions Lambda / la programmation fonctionnelle

Dans le code expliquant l’utilisation du SDK de Spotify, on retrouve une structure de code étrange :

.setEventCallback(playerState -> {  
 **final** Track track = playerState.**track**;  
 **if** (track != **null**) {  
 Log.*d*(**"MainActivity"**, track.**name** + **" by "** + track.**artist**.**name**);  
 }  
});

Il s’agit d’une expression lambda, caractérisée par la formulation :

paramètre -> expression / retour

ou

( paramètre1, paramètre2 ) -> expression / retour

Une expression lambda sert à permet d’écrire du code plus court, c’est une fonction anonyme ( pas de type de retour explicite, pas de modificateur d’accès, pas de nom ) . Plus précisément…

* C’est un raccourci syntaxique qui permet de définir une méthode exactement à l’endroit où elle est utilisée, ce raccourci est particulièrement utile quand le traitement à faire le sera une seule fois, il n’y a donc par d’avantage à coder une méthode « normale ».
* **Permet de remplacer une classe qui met en œuvre une interface fonctionnelle**

EX 1. : on veut gérer le clic d’un bouton

monBouton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

System.out.println("clic");

}

})

monBouton.addActionListener(event -> System.out.println("clic"));

boutton.setOnClickListener(source-> valeur +champText.getTExt());



EX 2 La méthode forEach de la classe Vector (api 24 ou plus pour l’utiliser )

* Bon exemple car cette méthode prend en paramètre un objet mettant en oeuvre une interface fonctionnelle ( Consumer )

data.forEach((n) -> System.out.println(n));

Site très complet : <https://www.jmdoudoux.fr/java/dej/chap-lambdas.htm>