DEVELOPPEMENT ANDROID AVEC KOTLIN

INTRODUCTION

Le développement d'applications Android avec Kotlin a gagné en popularité grâce à sa concision, sa sécurité et sa lisibilité. Dans cet exposé, nous allons explorer en détail certaines fonctionnalités clés du développement Android en utilisant Kotlin, en mettant l'accent sur les interfaces, les génériques, les parcelables, la toolbar et le context de l'application.

LES INTERFACES

En Android, les interfaces sont utilisées pour définir un contrat que les classes doivent implémenter. Par exemple, l'interface `OnClickListener` est utilisée pour gérer les événements de clic sur les vues. Elle peut contenir du code mais pas d'état donc une interface ne peut pas être instancier elle définie un comportement.

Exemple: Un Contrat de Location

Supposons que vous ayez un contrat de location (l'interface) qui spécifie que toute personne qui le signe doit effectuer certaines actions, comme payer le loyer à temps.

```
interface Locataire {
    fun payerLoyer()
}

class Personne : Locataire {
    override fun payerLoyer() {
        println("Loyer payé à temps.")
    }
}
```

Ici, la classe Personne a signé le contrat (implémenté l'interface) en promettant de payer le loyer.

LES GENIRICS

Les génériques permettent de créer des classes, interfaces, et méthodes qui peuvent fonctionner avec différents types de données.

Dans le contexte Android, vous pouvez les rencontrer dans des classes utilitaires, des adapteurs, etc. Ils sont utiliser si on:

- Applique un même comportement sur différentes class
- Pour stocker les éléments dans différents liste
- Trie les éléments sur n'importe quel type

AVANTAGES

- Gérer un nouveau type sans casser tout
- Plus de vérification à la compilation

Exemple:

Imaginez une boîte qui peut contenir différents types d'objets. Vous n'avez pas besoin d'une boîte spécifique pour chaque type d'objet.

Générique au niveau d'une méthode

```
fun <T> affiche(eleme :Array<T>){
    print("le contenu de la liste est"$elem)
}
fun main(){
    val entier = 2
    val string = "mamadou"
    affiche(entier)
    affiche(string)
}
```

Génériques au niveau d'une class

```
class BoitePolyvalente<T>(val contenu: T) {
    fun afficherContenu() {
        println("Contenu de la boîte : $contenu")
    }
}

fun main() {
    val boiteString = BoitePolyvalente("Bonjour !")
    val boiteInt = BoitePolyvalente(42)

    boiteString.afficherContenu()
    boiteInt.afficherContenu()
}
```

Ici, Boite Polyvalente est une boîte qui peut contenir n'importe quel type d'objet.

LES PARCELABLE

L'interface `Parcelable` est utilisée pour permettre à un objet d'être sérialisé facilement pour le passage d'objets complexes entre différentes composantes de l'application, comme les activités et les fragments.

Cela peut améliorer les performances par rapport à la sérialisation classique

La sérialisation procéder qui permet de transformer en objet une instance de classe pour la transférer ver une autre partie du programme donc un objet est parcelable si on peut le transférer vers:

- Une Activity

  Une autre application
- Utilisation: pour cela il faut:
- -Indiquer les données qui seront envoyées
- -Indiquer l‘ ordre dans lequel ils seront reçus

- Exemple: Imaginons une classe Person que nous voulons passer entre les activités :
- Activity 1:

```
data class Contact(val nom: String, val numero: String) : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readString() ?: "",
        parcel.readString() ?: ""
    )

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(nom)
        parcel.writeString(numero)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<Contact> {
        override fun createFromParcel(parcel: Parcel): Contact {
            return Contact(parcel)
        }

        override fun newArray(size: Int): Array<Contact?> {
            return arrayOfNulls(size)
        }
    }
}
```

TOOLBAR EN ANDROID

- La `Toolbar` est une barre d'outils qui peut être placée en haut d'une activité ou d'un fragment. Elle remplace généralement l'ActionBar plus ancienne et offre plus de flexibilité et de personnalisation. Vous pouvez y ajouter des éléments tels que des boutons, des logos, des titres, etc. Elle permet d'ajouter plus d'action supplémentaires pour l'utilisateur

  Appellation: ActionBar, AppBar, Toolbar

- Dans le xml:

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar" />
```

Dans votre activité, vous pouvez l'initialiser :

```
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar

class MyActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val toolbar: Toolbar = findViewById(R.id.my_toolbar)
        setSupportActionBar(toolbar)
    }
}
```

LE CONTEXT

Le `Context` est une classe fondamentale dans Android qui fournit des informations sur l'environnement d'exécution de l'application.

- Il est nécessaire pour effectuer de nombreuses opérations, comme l'accès aux ressources, le lancement d'activités, la création de services, etc.
- Il existe plusieurs types de contextes, tels que `Activity`, `Service`, `Application`, etc.

- Dans une application de messagerie, par exemple :

```
class Messagerie(val context: Context) {
    fun envoyerMessage(destinataire: String, message: String) {
        // Utilisation du Context pour envoyer un message
        val intent = Intent(context, MessageActivity::class.java)
        intent.putExtra("DESTINATAIRE", destinataire)
        intent.putExtra("MESSAGE", message)
        context.startActivity(intent)
    }
}
```

Ici, la classe Messagerie utilise le Context pour démarrer une nouvelle activité et envoyer un message à un destinataire spécifié

CONCLUSION

En résumé, ces concepts sont tous liés au développement Android et sont souvent utilisés ensemble pour créer des applications robustes et flexibles. Les interfaces définissent des contrats, les génériques apportent de la flexibilité, Parcelable facilite la transmission d'objets, la Toolbar offre une interface utilisateur riche, et le Context est essentiel pour interagir avec l'environnement d'exécution de l'application.