



TP guidé – Création de composants réutilisables

1. Objectif du TP

Dans ce TP, vous allez reprendre l'écran produit construit lors de la séance précédente et l'améliorer.

L'objectif est de transformer une fiche produit simple en un ensemble de composants plus réutilisables.

Vous allez pratiquer :

- le passage d'un **Modifier** en paramètre ;
- le **chaînage** de **modifiers** ;
- la différence entre **padding externe** et **padding interne** ;
- la création de petits composants **réutilisables** ;
- l'ajout d'une action de **clic** ;
- l'ajout de premières informations **d'accessibilité** ;
- l'utilisation de **previews** pour vérifier plusieurs cas.

Nous ne faisons pas encore de gestion d'état réelle. Le **clic** sera transmis sous forme de **callback**, mais il ne modifiera pas encore **isFavorite**. Cela sera vu plus tard dans le bloc **State Management**.

2. Point de départ

Repartez du TP précédent, avec une structure proche de celle-ci.

```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        isAvailable = true,
        isFavorite = false,
        onFavoriteClick = {
            // Pour l'instant, nous ne modifions pas encore l'état.
            // Nous verrons cela dans la séance sur la gestion d'état.
        },
        modifier = modifier.padding(24.dp)
    )
}
```

Et une carte produit proche de celle-ci :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(modifier = modifier) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            ProductTitle(name = name)
            ProductPrice(price = price)
            ProductAvailability(isAvailable = isAvailable)

            FavoriteButton(
                isFavorite = isFavorite,
                onFavoriteClick = onFavoriteClick
            )
        }
    }
}
```



Si votre code n'est pas exactement identique, ce n'est pas grave. L'important est d'avoir une fonction **ProductCard** avec plusieurs paramètres.

3. Étape 1 – Vérifier que chaque composable réutilisable accepte un Modifier

Dans Compose, un composable **réutilisable** doit souvent accepter un paramètre **modifier**. Cela permet au **parent** de **décider** comment placer le composant.

Nous allons donc modifier progressivement les petits composants.

Modifiez **ProductTitle** :

```
@Composable
fun ProductTitle(
    name: String,
    modifier: Modifier = Modifier
) {
    Text(
        text = name,
        modifier = modifier
    )
}
```

Modifiez **ProductPrice** :

```
@Composable
fun ProductPrice(
    price: Double,
    modifier: Modifier = Modifier
) {
    Text(
        text = "$price €",
        modifier = modifier
    )
}
```

Modifiez **ProductAvailability** :

```
@Composable
fun ProductAvailability(
    isAvailable: Boolean,
    modifier: Modifier = Modifier
) {
    Text(
        text = if (isAvailable) {
            "Produit disponible"
        } else {
            "Produit indisponible"
        },
        modifier = modifier
    )
}
```

Ce qu'il faut comprendre : Pour l'instant, le résultat visuel ne change pas forcément.

Mais le code devient meilleur, car chaque petit composant peut maintenant être placé plus facilement par son parent.

4. Étape 2 – Utiliser les **modifiers** depuis le parent

Maintenant que les petits composants acceptent un **modifier**, **ProductCard** peut décider comment les espacer.



Nous allons ajouter un peu d'espace entre les éléments sans changer la logique de chaque composant. Pour rester simple, nous allons utiliser des **padding** sur les composants.

Modifiez le contenu de **ProductCard** :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(modifier = modifier) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            ProductTitle(
                name = name,
                modifier = Modifier.padding(bottom = 8.dp)
            )

            ProductPrice(
                price = price,
                modifier = Modifier.padding(bottom = 8.dp)
            )

            ProductAvailability(
                isAvailable = isAvailable,
                modifier = Modifier.padding(bottom = 12.dp)
            )

            FavoriteButton(
                isFavorite = isFavorite,
                onFavoriteClick = onFavoriteClick
            )
        }
    }
}
```

ProductTitle ne décide pas lui-même qu'il doit avoir un espace en dessous. C'est **ProductCard** qui décide, car c'est **ProductCard** qui connaît l'organisation de la fiche.

C'est une bonne séparation des responsabilités.

5. Étape 3 – Distinguer *padding* externe et *padding* interne

Maintenant, observons deux niveaux de **padding**.

Dans **ProductDemoScreen**, nous plaçons la carte dans l'écran :

```
modifier = modifier.padding(24.dp)
```

Dans **ProductCard**, nous plaçons le contenu à l'intérieur de la carte :

```
Column(
    modifier = Modifier.padding(16.dp)
)
```

Ces deux **padding**s n'ont pas le même rôle.

- Le **premier** éloigne la carte des **bords** de l'écran.
- Le **second** éloigne le contenu des **bords** de la **carte**.

Gardez **ProductDemoScreen** ainsi :



```
@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        isAvailable = true,
        isFavorite = false,
        onFavoriteClick = {
            // ...
        },
        modifier = modifier.padding(24.dp)
    )
}
```

Et gardez dans **ProductCard** :

```
Surface(modifier = modifier) {
    Column(
        modifier = Modifier.padding(16.dp)
    ) {
        // contenu
    }
}
```

Ce qu'il faut comprendre :

- Le parent contrôle l'espace autour.
- Le composant contrôle l'espace interne nécessaire à sa lisibilité.

C'est une règle simple, mais très importante pour écrire des composants réutilisables.

6. Étape 4 – Ajouter **fillMaxWidth**

Actuellement, selon notre code, la carte peut prendre seulement la **taille de son contenu**. Nous allons demander à la carte d'occuper **toute la largeur disponible**.

Pour cela, on utilise **fillMaxWidth()**.

Modifiez l'appel de **ProductCard** dans **ProductDemoScreen** :

```
modifier = modifier
    .padding(24.dp)
    .fillMaxWidth()
```

Import à ajouter si nécessaire : `import androidx.compose.foundation.layout.fillMaxWidth`

Ce qu'il faut comprendre : **ProductDemoScreen** décide que la carte doit prendre toute la largeur disponible après le **padding externe**.

La carte elle-même **reste réutilisable** : ailleurs, on pourrait l'utiliser sans **fillMaxWidth**.

7. Étape 5 – Observer l'ordre des modifieurs

Nous allons comparer deux chaînes de **modifieurs**.

Version A :

```
modifier = modifier
    .padding(24.dp)
    .fillMaxWidth()
```

Version B :



```
modifier = modifier
    .fillMaxWidth()
    .padding(24.dp)
```

Ces deux versions peuvent produire un résultat différent. L'ordre indique l'ordre des transformations.

Testez d'abord la version B, puis testez la version A. Il faut être conscient que l'ordre a un effet.

Pour notre TP, gardez ensuite cette version :

```
modifier = modifier
    .fillMaxWidth()
    .padding(24.dp)
```

Cette version est facile à lire : l'écran donne à la carte **toute la largeur**, puis garde un espace avec les bords.

8. Étape 6 – Rendre la carte cliquable

Nous allons maintenant permettre à la carte de **signaler un clic**.

Attention : nous ne faisons pas encore de **navigation**. Nous ne faisons pas encore **d'état** avancé.

Nous ajoutons seulement un **callback** : `onProductClick: () -> Unit`

- La carte dira : « *on a cliqué sur moi* ».
- Le parent décidera plus tard quoi faire.

Modifiez la signature de **ProductCard** :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onProductClick: () -> Unit,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(
        color = Color.Magenta,
        modifier = modifier.clickable(onClick = onProductClick)
    ) {
        Column(
            modifier = Modifier.padding(16.dp)
        ) {
            // le reste du code ne change pas ...
        }
    }
}
```

Import à ajouter si nécessaire : `import androidx.compose.foundation.clickable`

Puis modifiez **ProductDemoScreen** pour ajouter **onProductClick** au-dessus de **onFavoriteClick** :

```
onProductClick = {
    // Plus tard : ouvrir le détail du produit
},
onFavoriteClick = {
    // Pour l'instant, nous ne modifions pas encore l'état.
    // Nous verrons cela dans la séance sur la gestion d'état.
},
```

Nous avons maintenant deux événements différents :

- **onProductClick** : clic sur la **fiche produit** ;



- **onFavoriteClick** : clic sur le bouton favori.
- La carte ne décide pas de la navigation.
- Le bouton ne décide pas de la logique favori.
- Ils signalent simplement des événements.

9. Étape 7 – Rendre l'intention du clic plus claire

Actuellement, la carte est cliquable, mais le code ne précise pas clairement son rôle.

Nous pouvons indiquer que cette zone cliquable se comporte comme un **bouton**.

Modifiez la partie *Surface* :

```
Surface(  
    color = Color.LightGray,  
    modifier = modifier.clickable(  
        role = Role.Button,  
        onClick = onProductClick  
    )  
) {  
    Column(  
        modifier = Modifier.padding(16.dp)  
    ) {  
        // contenu  
    }  
}
```

Import à ajouter si nécessaire : `import androidx.compose.ui.semantics.Role`

- Un élément personnalisé cliquable peut être plus **compréhensible** si son rôle est indiqué.
- Ici, on dit que **la carte se comporte comme un bouton** : l'utilisateur peut l'activer.

10. Étape 8 – Ajouter une description d'accessibilité simple

Nous allons maintenant ajouter une **information sémantique** à la carte.

L'objectif est que l'action soit **compréhensible** même pour un utilisateur qui ne perçoit pas l'interface visuellement.

Pour cela, on peut utiliser **semantics**. Modifiez la *Surface* comme cela :

```
Surface(  
    modifier = modifier.clickable(  
        role = Role.Button,  
        onClick = onProductClick  
    )  
    .semantics {  
        contentDescription = "Ouvrir le détail du produit $name"  
    }  
) {  
    // contenue  
}
```

Imports à ajouter si nécessaire :

```
import androidx.compose.ui.semantics.contentDescription  
import androidx.compose.ui.semantics.semantics
```

- **contentDescription** fournit une **description lisible** par les **services d'accessibilité**.
- Il ne faut pas tout décrire inutilement, mais lorsqu'un élément personnalisé est cliquable, il est important que son action soit compréhensible.
- Ici, la description explique l'action, par exemple : *Ouvrir le détail du produit « Sac à dos »*



11. Étape 9 – Extraire un composant *ProductCardContent*

ProductCard commence à avoir deux responsabilités :

1. gérer le conteneur cliquable ;
2. organiser le contenu de la carte.

Pour améliorer la lisibilité, nous allons extraire le contenu dans un composable séparé.

Créez un nouveau composable :

```
@Composable
fun ProductCardContent(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier.padding(16.dp)
    ) {
        ProductTitle(
            name = name,
            modifier = Modifier.padding(bottom = 8.dp)
        )

        ProductPrice(
            price = price,
            modifier = Modifier.padding(bottom = 8.dp)
        )

        ProductAvailability(
            isAvailable = isAvailable,
            modifier = Modifier.padding(bottom = 12.dp)
        )

        FavoriteButton(
            isFavorite = isFavorite,
            onFavoriteClick = onFavoriteClick
        )
    }
}
```

Puis modifiez **ProductCard** :

```
@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onProductClick: () -> Unit,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(
        modifier = modifier
        .clickable(
            role = Role.Button,
            onClick = onProductClick
        )
        .semantics {
            contentDescription = "Ouvrir le détail du produit $name"
        }
    ) {
        ProductCardContent(
            name = name,
            price = price,
            isAvailable = isAvailable,
            isFavorite = isFavorite,
            onFavoriteClick = onFavoriteClick
        )
    }
}
```



- **ProductCard** gère maintenant le **conteneur général** et le **clic** sur la carte.
- **ProductCardContent** gère l'organisation interne.

Ce découpage rend le code plus lisible et plus facile à faire évoluer.

12. Étape 10 – Améliorer *FavoriteButton*

Le bouton **favori** reçoit déjà une **donnée** et un **événement**.

Nous pouvons maintenant lui ajouter un **modifier**, comme pour les autres composants.

Modifiez **FavoriteButton** :

```
@Composable
fun FavoriteButton(
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Button(
        onClick = onFavoriteClick,
        modifier = modifier
    ) {
        Text(
            text = if (isFavorite) {
                "★ Favori"
            } else {
                "☆ Ajouter aux favoris"
            }
        )
    }
}
```

Puis dans **ProductCardContent**, appelez-le ainsi :

```
FavoriteButton(
    isFavorite = isFavorite,
    onFavoriteClick = onFavoriteClick,
    modifier = Modifier.fillMaxWidth()
)
```

- Le bouton devient lui aussi plus **réutilisable**.
- Ici, **ProductCardContent** décide que dans une **fiche** produit, le bouton **favori** doit prendre toute la largeur.
- Mais dans un **autre contexte**, on pourrait utiliser le même bouton sans **fillMaxWidth**.

13. Étape 11 – Créer plusieurs *previews*

Comme on l'a déjà vu, les **previews** servent à vérifier plusieurs états visuels. Nous allons créer (plutôt modifier car normalement ils existent déjà) deux **previews** :

- produit **disponible**, non **favori** ;
- produit **indisponible**, **favori**.



```
@Preview(showBackground = true)
@Composable
fun ProductCardAvailablePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Sac à dos",
            price = 49.99,
            isAvailable = true,
            isFavorite = false,
            onProductClick = {},
            onFavoriteClick = {},
            modifier = Modifier
                .fillMaxWidth()
                .padding(24.dp)
        )
    }
}
```

Et :

```
@Preview(showBackground = true)
@Composable
fun ProductCardUnavailableFavoritePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Casque audio",
            price = 89.99,
            isAvailable = false,
            isFavorite = true,
            onProductClick = {},
            onFavoriteClick = {},
            modifier = Modifier
                .fillMaxWidth()
                .padding(24.dp)
        )
    }
}
```

Adaptez **GreetingCardTheme** au nom réel du thème de votre projet.

- Un composant réutilisable doit être vérifié dans plusieurs situations.
- Ne testez pas uniquement le cas « parfait ».

Une carte produit peut être :

- disponible ou indisponible ;
- favorite ou non favorite ;
- courte ou longue ;
- intégrée dans un écran large ou étroit.

14. Étape 12 – Code complet indicatif

Voici une version complète possible du code du TP.

```
package com.example.greetingcard

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Button
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Surface
```



```
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.semantics.Role
import androidx.compose.ui.semantics.contentDescription
import androidx.compose.ui.semantics.semantics
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import com.example.greetingcard.ui.theme.GreetingCardTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            GreetingCardTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    ProductDemoScreen(
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}

@Composable
fun ProductDemoScreen(modifier: Modifier = Modifier) {
    ProductCard(
        name = "Sac à dos",
        price = 49.99,
        isAvailable = true,
        isFavorite = false,
        onProductClick = {
            // Plus tard : ouvrir le détail du produit
        },
        onFavoriteClick = {
            // Pour l'instant, nous ne modifions pas encore l'état.
            // Nous verrons cela dans la séance sur la gestion d'état.
        },
        modifier = modifier
            .fillMaxWidth()
            .padding(12.dp)
    )
}

@Composable
fun ProductCard(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onProductClick: () -> Unit,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Surface(
        modifier = modifier
            .clickable(
                role = Role.Button,
                onClick = onProductClick
            )
            .semantics {
                contentDescription = "Ouvrir le détail du produit $name"
            }
    ) {
        ProductCardContent(
            name = name,
            price = price,
            isAvailable = isAvailable,
            isFavorite = isFavorite,
            onFavoriteClick = onFavoriteClick
        )
    }
}

@Composable
fun ProductCardContent(
    name: String,
    price: Double,
    isAvailable: Boolean,
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
)
```



```
) {
    Column(
        modifier = modifier.padding(16.dp)
    ) {
        ProductTitle(
            name = name,
            modifier = Modifier.padding(bottom = 8.dp)
        )

        ProductPrice(
            price = price,
            modifier = Modifier.padding(bottom = 8.dp)
        )

        ProductAvailability(
            isAvailable = isAvailable,
            modifier = Modifier.padding(bottom = 12.dp)
        )

        FavoriteButton(
            isFavorite = isFavorite,
            onFavoriteClick = onFavoriteClick,
            modifier = Modifier.fillMaxWidth()
        )
    }
}

@Composable
fun ProductTitle(
    name: String,
    modifier: Modifier = Modifier
) {
    Text(
        text = name,
        modifier = modifier
    )
}

@Composable
fun ProductPrice(
    price: Double,
    modifier: Modifier = Modifier
) {
    Text(
        text = "$price €",
        modifier = modifier
    )
}

@Composable
fun ProductAvailability(
    isAvailable: Boolean,
    modifier: Modifier = Modifier
) {
    Text(
        text = if (isAvailable) {
            "Produit disponible"
        } else {
            "Produit indisponible"
        },
        modifier = modifier
    )
}

@Composable
fun FavoriteButton(
    isFavorite: Boolean,
    onFavoriteClick: () -> Unit,
    modifier: Modifier = Modifier
) {
    Button(
        onClick = onFavoriteClick,
        modifier = modifier
    ) {
        Text(
            text = if (isFavorite) {
                "★ Favori"
            } else {
                "★ Ajouter aux favoris"
            }
        )
    }
}
}
```



```
@Preview(showBackground = true)
@Composable
fun ProductDemoScreenPreview() {
    GreetingCardTheme {
        ProductDemoScreen()
    }
}

@Preview(showBackground = true)
@Composable
fun ProductCardAvailablePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Sac à dos",
            price = 49.99,
            isAvailable = true,
            isFavorite = false,
            onProductClick = {},
            onFavoriteClick = {},
            modifier = Modifier
                .fillMaxWidth()
                .padding(24.dp)
        )
    }
}

@Preview(showBackground = true)
@Composable
fun ProductCardUnavailableFavoritePreview() {
    GreetingCardTheme {
        ProductCard(
            name = "Casque audio",
            price = 89.99,
            isAvailable = false,
            isFavorite = true,
            onProductClick = {},
            onFavoriteClick = {},
            modifier = Modifier
                .fillMaxWidth()
                .padding(24.dp)
        )
    }
}
```

Le nom du thème dépend de votre projet. Remplacez ***GreetingCardTheme*** par le nom réellement généré dans votre application.

15. Questions de compréhension (auto-évaluation)

Répondez brièvement aux questions suivantes.

1. À quoi sert un *Modifier* en Compose ?
2. Pourquoi est-il recommandé de faire accepter un *modifier* aux composables réutilisables ?
3. Quelle est la différence entre le padding appliqué dans *ProductDemoScreen* et celui appliqué dans *ProductCardContent* ?
4. Pourquoi l'ordre des modifieurs peut-il changer le résultat visuel ou interactif ?
5. Pourquoi *ProductTitle* ne doit-il pas imposer lui-même un padding externe ?
6. Quel est le rôle de *onProductClick* ?
7. Pourquoi le clic sur la carte ne fait-il pas encore de navigation ?
8. À quoi sert *Role.Button* dans notre carte cliquable ?
9. À quoi sert *contentDescription* ?



10. Pourquoi est-il utile de créer plusieurs previews d'un même composant ?

16. Ce qu'il faut retenir

- Dans cette séance, vous avez transformé une fiche produit simple en composants plus réutilisables.
- Vous devez surtout retenir que *Modifier* est un outil central en Compose. Il permet de modifier la taille, l'espace, l'apparence, le comportement et certaines informations d'accessibilité d'un composant.
- Un composable réutilisable doit souvent accepter un paramètre :
- modifier: Modifier = Modifier
- Le parent décide généralement du placement externe.
- Le composant décide de son contenu et, si nécessaire, de son espace interne.
- L'ordre des modifieurs est important et doit être lu comme une suite de transformations.
- Enfin, un composant personnalisé interactif doit être compréhensible, y compris pour les outils d'accessibilité.

Dans la séance suivante, nous entrerons dans le bloc **Compose UI** avec les composants fondamentaux comme *Text*, *Button*, *Icon*, *Image*, *Card* et *TextField*.