CHAPITRE 5 : GESTION DES EVENEMENTS ET INTENTIONS

1. Gérer les évènements d'une vue

Pour gérer les événements d'une vue sous Android, vous pouvez utiliser les écouteurs d'événements (event listeners) appropriés pour le type d'événement que vous souhaitez gérer. Voici quelques exemples d'écouteurs d'événements couramment utilisés :

1. OnClickListener : Cet écouteur permet de gérer les événements de clic sur une vue. Vous pouvez l'attacher à un bouton, une image, ou toute autre vue qui accepte les clics.

java
Copier

```java
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Code à exécuter lors du clic sur le bouton
    }
});
```

2. OnLongClickListener : Cet écouteur permet de gérer les événements de clic long sur une vue. Il est similaire à OnClickListener, mais il est déclenché lorsque l'utilisateur maintient le clic pendant une certaine durée.

java
Copier

```java
TextView textView = findViewById(R.id.textView);
textView.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        // Code à exécuter lors du clic long sur le TextView
        return true; // Retourne true si l'événement est consommé, false sinon
    }
});
```

3. TextWatcher : Cet écouteur permet de gérer les modifications de texte dans un EditText. Il est utile lorsque vous souhaitez réagir en temps réel aux changements de texte.

java
Copier

```java
EditText editText = findViewById(R.id.editText);
editText.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after)
{
        // Code à exécuter avant la modification du texte
    }

    @Override
```

```
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        // Code à exécuter pendant la modification du texte
    }

    @Override
    public void afterTextChanged(Editable s) {
        // Code à exécuter après la modification du texte
    }
});
```

Ce ne sont que quelques exemples d'écouteurs d'événements. Android propose une large gamme d'écouteurs pour gérer différents types d'événements tels que les gestes tactiles, les mouvements, les changements de focus et bien plus encore. Vous pouvez consulter la documentation officielle d'Android pour en savoir plus sur les différents écouteurs d'événements disponibles et comment les utiliser dans votre application.

2- Afficher des messages de journalisation (Logs) et d'information (Toasts) Passer d'une activité à une autre

Pour afficher des messages de journalisation (logs) et des informations (toasts) dans une application Android, ainsi que pour passer d'une activité à une autre, voici les étapes à suivre :

1. Afficher des messages de journalisation (logs) :

Pour afficher des messages de journalisation dans votre application Android, vous pouvez utiliser la méthode `Log.d()` (ou d'autres niveaux de journalisation tels que `Log.i()`, `Log.e()`, etc.). Voici un exemple :

java
Copier

```
import android.util.Log;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.d(TAG, "Message de journalisation (log) de niveau DEBUG");
        Log.i(TAG, "Message de journalisation (log) de niveau INFO");
        Log.e(TAG, "Message de journalisation (log) de niveau ERREUR");
    }
}
```

Dans cet exemple, nous utilisons `Log.d(TAG, message)` pour afficher un message de journalisation de niveau DEBUG. Vous pouvez également utiliser `Log.i()` pour les informations, `Log.e()` pour les erreurs, etc. `TAG` est une

```
étiquette qui vous aide à identifier les messages de journalisation dans vo
s logs.

Pour voir les messages de journalisation, vous pouvez utiliser l'outil "Log
cat" dans Android Studio. Les messages de journalisation seront affichés da
ns la console Logcat pendant l'exécution de l'application.
```

2. Afficher des informations (toasts) :

Pour afficher des informations sous forme de toasts dans votre application Android, vous pouvez utiliser la classe `Toast`. Voici un exemple :

java
Copier

```
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toast.makeText(this, "Message de toast", Toast.LENGTH_SHORT).show()
;
    }
}
```

```
Dans cet exemple, nous utilisons `Toast.makeText(context, message, duration
)` pour créer un toast avec un message spécifié et une durée d'affichage. V
ous pouvez utiliser `Toast.LENGTH_SHORT` pour une durée courte ou `Toast.LE
NGTH_LONG` pour une durée plus longue.

Le toast sera affiché à l'écran pour informer l'utilisateur. Assurez-vous d
'appeler `show()` pour afficher le toast.
```

3. Passer d'une activité à une autre :

Pour passer d'une activité à une autre dans votre application Android, vous pouvez utiliser un `Intent` pour démarrer une nouvelle activité. Voici un exemple :

java
Copier

```
import android.content.Intent;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        // Lorsque vous souhaitez passer à une autre activité, par exemple
lors d'un clic sur un bouton
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, SecondActivit
y.class);
                startActivity(intent);
            }
        });
    }
}
```

Dans cet exemple, nous utilisons un `Intent` pour démarrer une nouvelle act ivité (`SecondActivity`). Vous devez spécifier la classe de l'activité de d estination dans le constructeur de l'intent. Ensuite, nous utilisons `start Activity(intent)` pour démarrer l'activité.

Assurez-vous d'avoir déclaré vos activités dans le fichier `AndroidManifest .xml` de votre application pour qu'elles soient reconnues.

Vous pouvez également passer des données supplémentaires à la nouvelle acti vité en ajoutant des extras à l'intent, en utilisant `intent.putExtra(key, value)`. Les données peuvent être récupérées dans l'activité de destination à l'aide de `getIntent().getXXXExtra(key)`.

En utilisant ces étapes, vous pouvez afficher des messages de journalisation (logs) et d'information (toasts) dans votre application Android, ainsi que passer d'une activité à une autre pour naviguer entre différentes parties de votre application.

1. Transférer des données entre les activités

Pour transférer des données entre les activités dans une application Android, vous pouvez utiliser les Intent et les extras. Voici comment procéder :

1. Dans l'activité source (l'activité à partir de laquelle vous voulez transférer les données), créez un objet Intent et utilisez la méthode `putExtra()` pour ajouter les données à transférer. Par exemple :

java
Copier

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
intent.putExtra("nom", "John");
intent.putExtra("age", 25);
startActivity(intent);
```

Dans cet exemple, nous ajoutons deux extras à l'intent : le nom et l'âge. Les extras sont identifiés par des clés (dans ce cas, "nom" et "age") et peuvent contenir différents types de données (chaînes, entiers, booléens, etc.).

2. Dans l'activité de destination (la deuxième activité où vous voulez récupérer les données), vous pouvez récupérer les extras à l'aide de la méthode `getIntent().getXXXExtra()`. Par exemple :

java
Copier

```
Intent intent = getIntent();
String nom = intent.getStringExtra("nom");
int age = intent.getIntExtra("age", 0);
```

Dans cet exemple, nous utilisons `getStringExtra()` pour récupérer la valeur de l'extra "nom" en tant que chaîne, et `getIntExtra()` pour récupérer la valeur de l'extra "age" en tant qu'entier. La méthode `getIntExtra()` prend également un deuxième argument facultatif qui représente une valeur par défaut au cas où l'extra ne serait pas présent.

3. Vous pouvez maintenant utiliser les données récupérées dans l'activité de destination comme bon vous semble. Par exemple, vous pouvez les afficher dans des TextView, les utiliser pour effectuer des calculs, etc.

Il est important de noter que les données transférées via les extras sont limitées en taille. Si vous avez besoin de transférer des objets plus complexes ou de grandes quantités de données, vous pouvez envisager d'autres méthodes telles que le stockage dans une base de données ou l'utilisation de classes de gestion de l'état (ViewModels).
De plus, assurez-vous de bien déclarer les deux activités dans votre fichier AndroidManifest.xml pour qu'elles soient reconnues et accessibles.
En utilisant ces étapes, vous pouvez facilement transférer des données entre les activités de votre application Android.

1. Détecter et résoudre des exceptions

La détection et la résolution des exceptions sont des aspects importants du développement d'applications Android. Voici quelques étapes à suivre pour détecter et résoudre des exceptions :

1. Comprendre les exceptions : Familiarisez-vous avec les types d'exceptions courantes dans le langage Java et dans le développement Android, tels que `NullPointerException`, `IllegalArgumentException`, `IOException`, etc. Comprenez les causes possibles de ces exceptions et les actions nécessaires pour les résoudre.
2. Utilisez des blocs try-catch : Entourez le code susceptible de générer une exception avec des blocs try-catch. Cela permet de capturer les

exceptions et de prendre des mesures appropriées pour les résoudre ou les gérer. Voici un exemple :

java
Copier

```
try {
    // Code susceptible de générer une exception
} catch (Exception e) {
    // Gestion de l'exception
    Log.e(TAG, "Une exception s'est produite : " + e.getMessage());
}
```

Dans cet exemple, le code pouvant générer une exception est placé à l'intérieur du bloc `try`. Si une exception se produit, elle est capturée par le bloc `catch`. Vous pouvez ensuite effectuer des actions spécifiques pour gérer l'exception, comme afficher un message d'erreur ou enregistrer les détails de l'exception dans les logs.

3. Utilisez des clauses finally : Utilisez la clause `finally` pour exécuter du code qui doit être exécuté, qu'une exception soit levée ou non. Par exemple :

java
Copier

```
try {
    // Code susceptible de générer une exception
} catch (Exception e) {
    // Gestion de l'exception
} finally {
    // Code à exécuter, qu'une exception se produise ou non
}
```

Le bloc `finally` est généralement utilisé pour effectuer des opérations de nettoyage, comme la fermeture de ressources ou la libération de mémoire.

4. Utilisez des logs pour déboguer : Utilisez la classe `Log` pour enregistrer des informations de débogage dans les logs lorsqu'une exception se produit. Cela peut vous aider à identifier la cause de l'exception et à résoudre le problème. Par exemple :

java
Copier

```
catch (Exception e) {
    Log.e(TAG, "Une exception s'est produite : " + e.getMessage());
}
```

Assurez-vous d'inclure des informations utiles dans les logs, telles que le nom de la classe, la méthode et le message d'erreur de l'exception.

5. Utilisez des outils de débogage : Android Studio dispose d'outils de débogage puissants qui vous permettent de parcourir le code pas à pas, de surveiller les valeurs des variables et d'analyser les traces d'exécution pour détecter les exceptions. Utilisez le débogueur

intégré pour identifier les erreurs et comprendre le flux d'exécution de votre application.

6. Lire la documentation et rechercher des solutions : Consultez la documentation Android officielle et d'autres ressources en ligne pour en savoir plus sur les exceptions spécifiques et les meilleures pratiques pour les résoudre. Recherchez également des solutions sur les forums et les communautés de développeurs, car il y a de fortes chances que d'autres aient déjà rencontré des problèmes similaires et trouvé des solutions.

En suivant ces étapes, vous serez en mesure de détecter et de résoudre des exceptions dans votre application Android, ce qui contribuera à améliorer la stabilité et la fiabilité de votre application.

Conclusion

La gestion des événements et des intentions est un aspect essentiel du développement d'applications Android. Les événements, tels que les clics sur les boutons ou les mouvements de l'utilisateur, déclenchent des actions dans l'application, tandis que les intentions permettent de démarrer des composants tels que des activités ou des services.