Calculatrice Basique pour Android

« Réalisation d’une calculatrice basique pour Android permettant des opérations arithmétiques simples. »

Par Vasco Valadares Semana – En 2023 – Projet de Cours

Table des matières

1. Introduction

Ce document détaille la conception et l'implémentation du projet "Calculatrice Basique pour Android". L'objectif principal de ce projet était de s'initier au développement d'applications mobiles natives pour la plateforme Android en utilisant le langage Java et l'environnement de développement Android Studio. L'application développée est une calculatrice capable d'effectuer des opérations arithmétiques de base : addition, soustraction, multiplication et division, à travers une interface utilisateur simple et intuitive.

2. Objectifs du Projet

2.1 Contexte :

Ce projet a été entrepris dans le cadre de la première année du BTS SIO (Services Informatiques aux Organisations), spécialité SLAM (Solutions Logicielles et Applications Métiers). Il visait à mettre en application les concepts de base du développement mobile Android.

2.2 Objectifs principaux :

- **Apprentissage des Fondamentaux Android :** Se familiariser avec les concepts clés d'Android tels que les activités, les layouts XML, la gestion des événements et les composants d'interface utilisateur ;
- **Maîtrise de Java pour Android :** Appliquer les connaissances en langage Java dans le contexte spécifique du développement mobile ;
- **Utilisation d'Android Studio :** Découvrir et utiliser l'environnement de développement intégré Android Studio pour la création, le débogage et le déploiement d'applications ;
- **Conception d'Interface Utilisateur Simple :** Proposer une interface utilisateur claire et fonctionnelle pour une calculatrice ;
- **Implémentation de Logique de Calcul :** Intégrer les fonctions pour les calculs arithmétiques de base ;
- **Gestion Élémentaire des Entrées :** Assurer une gestion basique des entrées utilisateur et des erreurs potentielles, comme la division par zéro.

3. Architecture de l'Application

3.1. Structure du Projet

L'application "Calculatrice Basique pour Android" adopte une architecture simple, typique pour des projets de cette envergure sur Android :

- **Activité Principale (MainActivity.java) :** Une seule activité (AppCompatActivity) gère l'ensemble de la logique de l'application et les interactions avec l'interface utilisateur.
- **Layout XML (activity_main.xml) :** La disposition visuelle de l'interface utilisateur (boutons, zones d'affichage) est définie dans ce fichier XML, situé dans le répertoire res/layout.

- **Ressources Android :** Les identifiants des composants d'interface (boutons, TextViews), les chaînes de caractères et autres ressources sont gérés via le système de ressources Android (classe R).

3.2. Choix Technologiques

- **Langage de Programmation :** Java ;
- **Environnement de Développement Intégré (IDE) :** Android Studio ;
- **Version des Outils de Compilation (Build-Tools) :** Android SDK Build-Tools 34 ;
- **Gestion de Version :** Git et GitHub.

4. Structure des Données Internes et Variables Clés

Au sein de la classe MainActivity.java, plusieurs variables membres sont cruciales pour maintenir l'état de la calculatrice et gérer les calculs :

- private TextView historique;
    - **Rôle :** Référence au composant TextView de l'interface utilisateur qui affiche l'historique du calcul en cours (par exemple, "123+45=").
    - **Initialisation :** Dans onCreate(), via findViewById(R.id.historique).
- private TextView calculActuel;
    - **Rôle :** Référence au composant TextView qui affiche le nombre actuellement saisi par l'utilisateur ou le résultat du dernier calcul effectué.
    - **Initialisation :** Dans onCreate(), via findViewById(R.id.calculActuel).
- private double premierChiffre;
    - **Rôle :** Variable de type double utilisée pour stocker la valeur du premier opérande lorsqu'une opération arithmétique binaire (comme l'addition, la soustraction, etc.) est sélectionnée.
- private char operation;
    - **Rôle :** Variable de type char utilisée pour stocker le symbole de l'opération arithmétique sélectionnée par l'utilisateur (par exemple, '+', '-', '*', '/'). Elle est utilisée pour déterminer quel calcul effectuer lorsque l'utilisateur appuie sur le bouton "=".

Ces variables sont initialisées et modifiées au fil des interactions de l'utilisateur avec les boutons de la calculatrice.

5. Fonctionnalités Principales et Implémentation

La logique de la calculatrice est entièrement contenue dans la classe MainActivity.java, principalement à travers les méthodes onCreate, onClicNombre et onClicSpecial.

5.1. Interface Utilisateur et Affichage

L'interface utilisateur est définie dans R.layout.activity_main et comprend :

- Deux TextView pour l'affichage : R.id.historique et R.id.calculActuel.
- Des boutons pour les chiffres (de R.id.zero à R.id.neuf).
- Des boutons pour les opérations spéciales et arithmétiques (par exemple, R.id.C, R.id.ADDITION, R.id.EGAL).

La taille du texte dans le TextView calculActuel est ajustée dynamiquement dans la méthode onClicNombre pour une meilleure lisibilité en fonction du nombre de chiffres affichés :

- Taille par défaut : 100sp (pour moins de 7 chiffres).
- 7 chiffres : 90sp.
- 8 chiffres : 80sp.
- 9 chiffres ou plus : 70sp.

5.2. Saisie des Nombres (onClicNombre)

Cette méthode est appelée lorsqu'un bouton numérique (0-9) est pressé.

- **Limitation de saisie :** Empêche la saisie de plus de 9 chiffres dans calculActuel.
- **Mapping ID-Valeur :** Un Dictionary<Integer, Integer> equivalenceId (implémenté avec Hashtable) est utilisé pour faire correspondre l'ID du View (bouton) cliqué à sa valeur numérique.
- **Logique d'Affichage :**
    - Si calculActuel se termine par un ".", le nouveau chiffre est simplement concaténé.
    - Si calculActuel affiche "0" (et n'est pas "0."), le "0" est remplacé par le nouveau chiffre.
    - Sinon, le nouveau chiffre est ajouté à la fin du nombre existant, en préservant la partie entière si le nombre est déjà un entier.

```java
// Extrait de MainActivity.java
public void onClicNombre(View view) {
    if (calculActuel.getText().toString().length() >= 9) return;

    Dictionary<Integer, Integer> equivalenceId = new Hashtable<>();
    equivalenceId.put(R.id.zero, 0);
    equivalenceId.put(R.id.un, 1);
    // ... autres mappages pour 2 à 9 ...
    equivalenceId.put(R.id.neuf, 9);

    if (calculActuel.getText().toString().endsWith(".")) {
        calculActuel.setText(calculActuel.getText().toString() +
Objects.requireNonNull(equivalenceId.get(view.getId())));
    } else if (Float.parseFloat(calculActuel.getText().toString()) == 0f) {

 calculActuel.setText(String.format(Objects.requireNonNull(equivalenceId.get(view.getId())).toString())
);
    } else {
        double ancienneValeur = Double.parseDouble(calculActuel.getText().toString());
        int nouvelleValeur = Objects.requireNonNull(equivalenceId.get(view.getId()));
        String afficher;
        if (ancienneValeur - (int) ancienneValeur > 0) { // Si c'est déjà un décimal
            afficher = String.valueOf(ancienneValeur) + nouvelleValeur;
        } else { // Si c'est un entier
            afficher = String.valueOf((int) ancienneValeur) + nouvelleValeur;
        }
        calculActuel.setText(afficher);
    }

    // Ajustement de la taille du texte
    if (calculActuel.getText().toString().length() == 7) {
        calculActuel.setTextSize(90);
    } else if (calculActuel.getText().toString().length() == 8) {
        calculActuel.setTextSize(80);
    } else if (calculActuel.getText().toString().length() >= 9) {
        calculActuel.setTextSize(70);
    } else {
        calculActuel.setTextSize(100);
    }
}
```

5.3. Gestion des Opérations Spéciales et Calculs (onClicSpecial)

Cette méthode est appelée pour tous les boutons non numériques. Un Dictionary<Integer, String> equivalenceId mappe l'ID du bouton à une chaîne représentant l'action (ex: "C", "/", "="). Un switch est ensuite utilisé pour exécuter l'action appropriée.

5.3.1. Fonctions de Réinitialisation (C, CE) et Correction (DEL)

- **"C" (Clear)** : Réinitialise le TextView calculActuel à "0".
- **"CE" (Clear Entry/All Clear)** : Efface le contenu de historique et réinitialise calculActuel à "0".
- **"DEL" (Delete)** : Supprime le dernier caractère de calculActuel. Si calculActuel devient vide, il est réinitialisé à "0".

```java
// Extrait de MainActivity.java - Cas C, CE, DEL
case "C":
    calculActuel.setText("0");
    break;
case "CE":
    historique.setText("");
    calculActuel.setText("0");
    break;
case "DEL":
    CharSequence ancienneValeur = calculActuel.getText();
    if (ancienneValeur.length() > 0) {
        CharSequence nouvelleValeur = ancienneValeur.subSequence(0, ancienneValeur.length() - 1);
        if (nouvelleValeur.length() == 0) calculActuel.setText("0");
        else calculActuel.setText(nouvelleValeur);
    }
    break;
```

*5.3.2. Sélection des Opérateurs Arithmétiques (+, -, *, /)*

Lorsque l'utilisateur appuie sur un bouton d'opérateur arithmétique :

- Une vérification est faite pour éviter une opération si calculActuel est "0" (pour éviter "0/" par exemple).
- Le caractère de l'opération est stocké dans la variable operation.
- La valeur actuelle de calculActuel est convertie en double et stockée dans premierChiffre.
- historique est mis à jour pour afficher premierChiffre suivi du symbole de l'opérateur.
- calculActuel est réinitialisé à "0" pour permettre la saisie du second opérande.

```java
// Extrait de MainActivity.java - Cas Opérateur (exemple avec '/')
case "/":
    if (calculActuel.getText().equals("0")) return; // Évite la division par zéro à ce stade
    operation = '/';
    premierChiffre = Double.parseDouble(calculActuel.getText().toString()); // Utiliser
Double.parseDouble
    historique.setText(calculActuel.getText() + "/");
    calculActuel.setText("0");
    break;
// Logique similaire pour '-', '+', '*'
```

5.3.3. Exécution du Calcul (=)

Lorsque le bouton "=" est pressé :

- Une variable resultat de type double est initialisée.

- Un switch interne basé sur la variable operation effectue le calcul arithmétique approprié entre premierChiffre et la valeur actuelle de calculActuel (convertie en double).
- **Gestion de la division par zéro :** Le code actuel ne gère pas explicitement une ArithmeticException si l'utilisateur tente de diviser par zéro lors de l'appui sur "=". Cela pourrait entraîner un crash.
- Le TextView historique est mis à jour pour afficher l'opération complète (ex: "12+5=").
- Le resultat est affiché dans calculActuel. Si le résultat est un entier (pas de partie décimale), il est affiché sans le ".0". Sinon, le nombre double complet est affiché.
- La variable operation est réinitialisée à un espace (' ') pour indiquer qu'aucune opération n'est en attente.

```java
// Extrait de MainActivity.java - Cas Égal
case "=":
    double resultat = 0;
    double secondChiffre = Double.parseDouble(calculActuel.getText().toString()); // Utiliser
Double.parseDouble

    switch (operation) {
        case '+':
            resultat = premierChiffre + secondChiffre;
            break;
        case '-':
            resultat = premierChiffre - secondChiffre;
            break;
        case '*':
            resultat = premierChiffre * secondChiffre;
            break;
        case '/':
            if (secondChiffre == 0) {
                // Gérer l'erreur de division par zéro ici (ex: afficher "Erreur")
                calculActuel.setText("Erreur");
                historique.setText("");
                operation = ' ';
                return;
            }
            resultat = premierChiffre / secondChiffre;
            break;
        default: // Aucune opération sélectionnée
            historique.setText(calculActuel.getText().toString() + "=");
            return;
    }

    historique.setText(historique.getText().toString() + calculActuel.getText().toString() + "=");

    if (resultat == (int) resultat) { // Vérification plus robuste pour entier
        calculActuel.setText(String.format(Integer.toString((int) resultat)));
    } else {
        calculActuel.setText(String.format(Double.toString(resultat)));
    }
    operation = ' '; // Réinitialiser l'opération
    break;
```

5.3.4. Gestion du Point Décimal (.)

- Lorsque le bouton "." est pressé, un point est ajouté à la fin du contenu de calculActuel.

- **Amélioration possible :** Le code actuel ne vérifie pas si un point décimal existe déjà dans calculActuel avant d'en ajouter un nouveau, ce qui pourrait conduire à des nombres invalides comme "1.2.3".

```
// Extrait de MainActivity.java - Cas Point
case ".":
    // Amélioration : if (!calculActuel.getText().toString().contains(".")) {
    calculActuel.setText(calculActuel.getText() + ".");
    // }
    break;
```

6. Points Particuliers et Solutions Techniques

- **Utilisation de Dictionary (Hashtable) :** Pour mapper les IDs des boutons à leurs valeurs ou actions correspondantes, simplifiant la logique dans les méthodes onClicNombre et onClicSpecial en permettant l'usage de switch au lieu de multiples if-else.
- **Ajustement Dynamique de la Taille du Texte :** La taille du texte dans le champ d'affichage principal (calculActuel) est réduite lorsque le nombre de chiffres augmente, assurant que les nombres restent visibles sans déborder.
- **Gestion de l'État Simplifiée :** L'état du calcul (premier opérande, opération en cours) est géré par des variables de classe simples (premierChiffre, operation), ce qui est suffisant pour une calculatrice basique.
- **Formatage du Résultat :** Le résultat des calculs est formaté pour s'afficher comme un entier si la partie décimale est nulle, améliorant la lisibilité.
- **Limitation de la Longueur de Saisie :** La saisie est limitée à 9 chiffres pour éviter les problèmes d'affichage et de gestion de très grands nombres dans cette version basique.

7. Interface Utilisateur (Description et Interactions)

7.1. Vue Générale de l'Interface

L'interface utilisateur de la calculatrice est conçue pour être familière et simple d'utilisation. Elle comprend typiquement :

- **Zone d'Affichage de l'Historique :** Un TextView en haut (R.id.historique) affichant la séquence de l'opération (ex: "123 + 45 =").
- **Zone d'Affichage Principale :** Un TextView plus grand (R.id.calculActuel) affichant le nombre en cours de saisie ou le résultat.
- **Pavé Numérique :** Des boutons pour les chiffres de 0 à 9.

- **Boutons d'Opérations :** Des boutons pour l'addition (R.id.ADDITION), la soustraction (R.id.SOUSTRACTION), la multiplication (R.id.MULTIPLICATION), et la division (R.id.DIVISER).
- **Boutons de Fonction :**
    - R.id.C (Clear)
    - R.id.CE (Clear Entry/All Clear)
    - R.id.DEL (Delete)
    - R.id.point (Point décimal)
    - R.id.EGAL (Égal)

7.2. Interactions Utilisateur

1. L'utilisateur saisit le premier nombre en cliquant sur les boutons numériques. Le nombre apparaît dans le champ calculActuel.
2. L'utilisateur clique sur un bouton d'opérateur (+, -, *, /). Le nombre saisi passe dans premierChiffre, l'opérateur est stocké, et l'affichage historique est mis à jour. calculActuel est réinitialisé à "0".
3. L'utilisateur saisit le second nombre.
4. L'utilisateur clique sur le bouton "=". Le calcul est effectué en utilisant premierChiffre, l'opérateur stocké, et le nombre actuel dans calculActuel. Le résultat s'affiche dans calculActuel et l'opération complète dans historique.
5. Les boutons C, CE, DEL permettent de corriger les erreurs de saisie ou de réinitialiser le calcul.
6. Le bouton «. » permet d'insérer un point décimal.

8. Perspectives d'Évolution et Améliorations Futures

Bien que fonctionnelle pour les opérations de base, l'application offre plusieurs pistes d'amélioration :

- **Gestion des Erreurs Robuste :** Améliorer la gestion d'erreurs de la calculatrice avec l'implémentation de « try/catch » et l'affichage de messages comment « Erreur », ou autre ;
- **Fonctionnalités Avancées :** Ajouter des opérations scientifiques comme le pourcentage (%), la racine carrée (√), l'exponentiation (x^y), intégrer des fonctions de mémoire (M+, M-, MR, MC) ;
- **Historique des Calculs :** Permettre de visualiser une liste des calculs précédents, avec la possibilité de réutiliser des résultats ou des opérations ;
- **Interface Utilisateur Améliorée :** Proposer des thèmes (clair/sombre) et améliorer l'esthétique générale et l'ergonomie ;
- **Gestion des Nombres Négatifs :** Permettre la saisie de nombres négatifs (par exemple, via un bouton "+/-") ;

- **Priorité des Opérateurs :** Pour des calculs plus complexes (ex: "2+3*4"), implémenter la logique de priorité des opérations (PEMDAS : parenthèse, exposant, multiplication et division, addition et soustraction) au lieu d'une évaluation strictement séquentielle ;
- **Tests Unitaires :** Rédiger des tests unitaires avec JUnit pour valider la logique de calcul et prévenir les régressions lors d'évolutions futures ;
- **Refactoring du Code :** Envisager de séparer la logique de calcul de l'activité UI (par exemple, dans une classe de modèle ou un utilitaire) pour une meilleure maintenabilité et testabilité à mesure que la complexité augmente.

9. Conclusion

Le projet "Calculatrice Basique pour Android" a constitué une introduction au développement d'applications mobiles avec Java et Android Studio. Il m’a permis de mettre en pratique les concepts fondamentaux de la plateforme Android, de la conception de l'interface utilisateur à l'implémentation de la logique métier pour des opérations arithmétiques simples. L'utilisation de structures de données comme Hashtable pour la gestion des événements et l'adaptation dynamique de l'affichage sont des exemples des solutions techniques mises en œuvre.