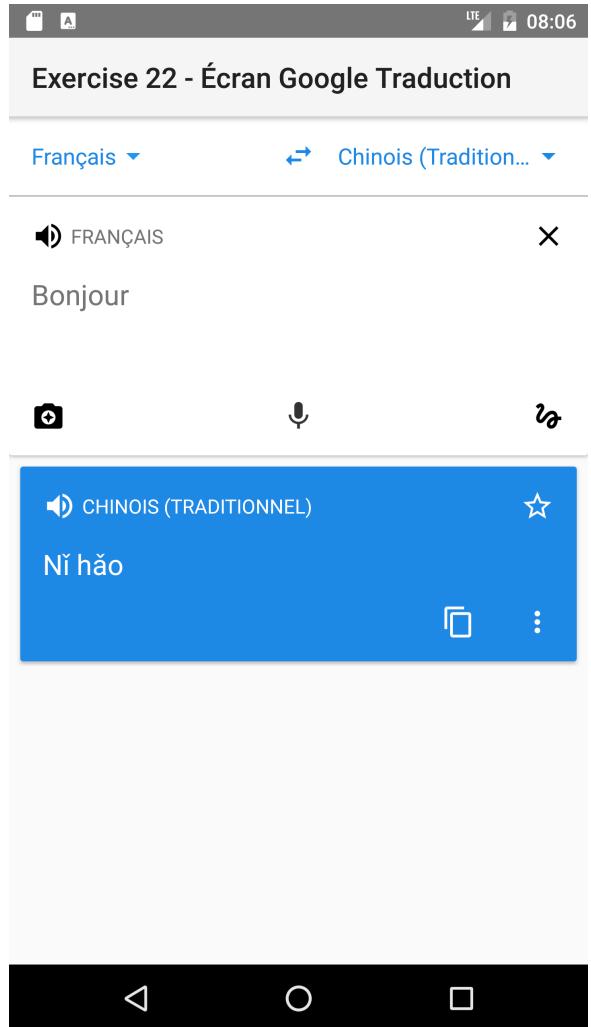


# Persistir les données

Adrian Bracigliano

# Niveau actuel

- Construire des écrans



# Niveau actuel

- ~~Construire des écrans~~
- Naviguer et intéragir avec des écrans



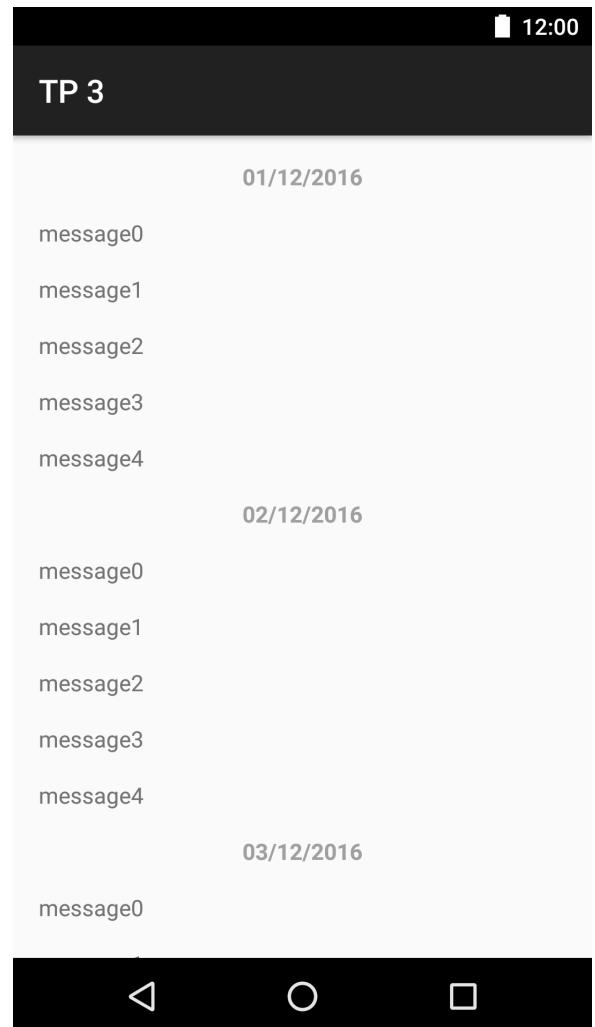
# Niveau actuel

- ~~Construire des écrans~~
- ~~Naviguer et intéragir avec des écrans~~
- Effectuer des traitements longs comme des appels réseaux



# Niveau actuel

- Construire des écrans
- Naviguer et intéragir avec des écrans
- Effectuer des traitements longs
- Utiliser des RecyclerViews



# Objectif

- Ajouter de la mémoire à notre application
- Comment faire survivre les données de l'utilisateur à la destruction d'une Activity et à la destruction de l'application

# Techniques de sauvegarde

- Singleton
- SavedInstanceState
- SharedPreferences
- Base de données SQLite

# Singleton

- Design pattern simple et facile à utiliser
- Permet d'avoir une instance unique d'une classe accessible partout dans l'application
- Permet de sauvegarder n'importe quel type de données
- Les données survivent le temps d'une session (Jusqu'à ce que le process de l'application soit tué par manque de mémoire)

# Singleton - Cas d'utilisation

- Garder temporairement en mémoire des données utilisées à plusieurs endroits dans l'application
- Eviter d'avoir à appeler plusieurs fois les mêmes webservices au cours d'une session
- Eviter d'avoir à passer les mêmes données en intent à plusieurs Activities différentes

# Singleton

```
public class MySingleton {  
  
    // Instance unique  
    private static MySingleton INSTANCE = new MySingleton();  
  
    // Constructeur privé  
    private MySingleton() { }  
  
    // Point d'accès pour l'instance unique  
    public static MySingleton getInstance() {  
        return INSTANCE;  
    }  
}
```

# Singleton

```
public class MySingleton {  
    private static MySingleton INSTANCE = new MySingleton();  
  
    // Données qu'on souhaite persister  
    private ArrayList<Data> list;  
    private Data data;  
  
    private MySingleton() {}  
  
    public static MySingleton getInstance() {  
        return INSTANCE;  
    }  
  
    public ArrayList<Data> getList() {}  
    public void setList(ArrayList<Data> data) {}  
    public Data getData() {}  
    public void setData(Data data) {}  
}
```

```
public class Activity1 extends Activity {  
    private MySingleton mySingleton = MySingleton.getInstance();  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        mySingleton.setData(data);  
    }  
}
```

```
public class Activity1 extends Activity {
    private MySingleton mySingleton = MySingleton.getInstance();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mySingleton.setData(data);
    }
}

public class Activity2 extends Activity {
    private MySingleton mySingleton = MySingleton.getInstance();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Null quand Activity recrée suite à un problème mémoire
        if (mySingleton.getData() != null) {
            showData(mySingleton.getData());
        } else {
            // Retour en arrière ou rechargement des données
        }
    }
}
```

# SavedInstanceState

- Méthode de l'Activity à implémenter
- Permet de sauvegarder n'importe quel type de données
- Données accessibles dans l'Activity uniquement
- Les données survivent à la recréation d'une Activity qui n'a pas été détruite volontairement (Bouton back, fermeture dans applications récentes)

# InstanceState

## SavedInstanceState Cas d'utilisation

- Utilisé par défaut par Android pour sauvegarder et restaurer l'état de chaque View (possédant un id) d'une Activity suite à sa recréation
  - Texte saisi dans un EditText
  - Case cochée d'une CheckBox



# SavedInstanceState

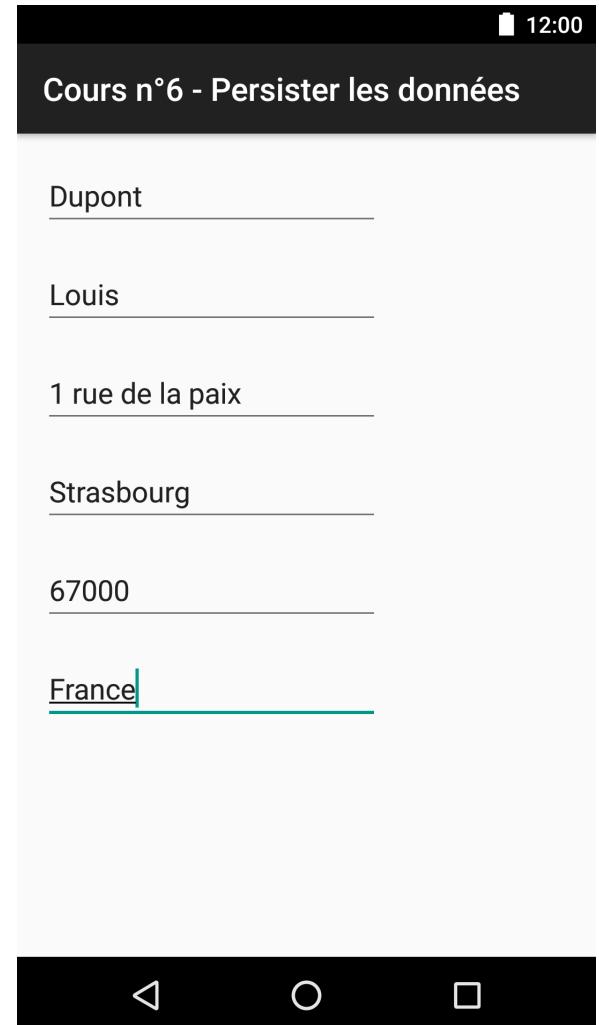
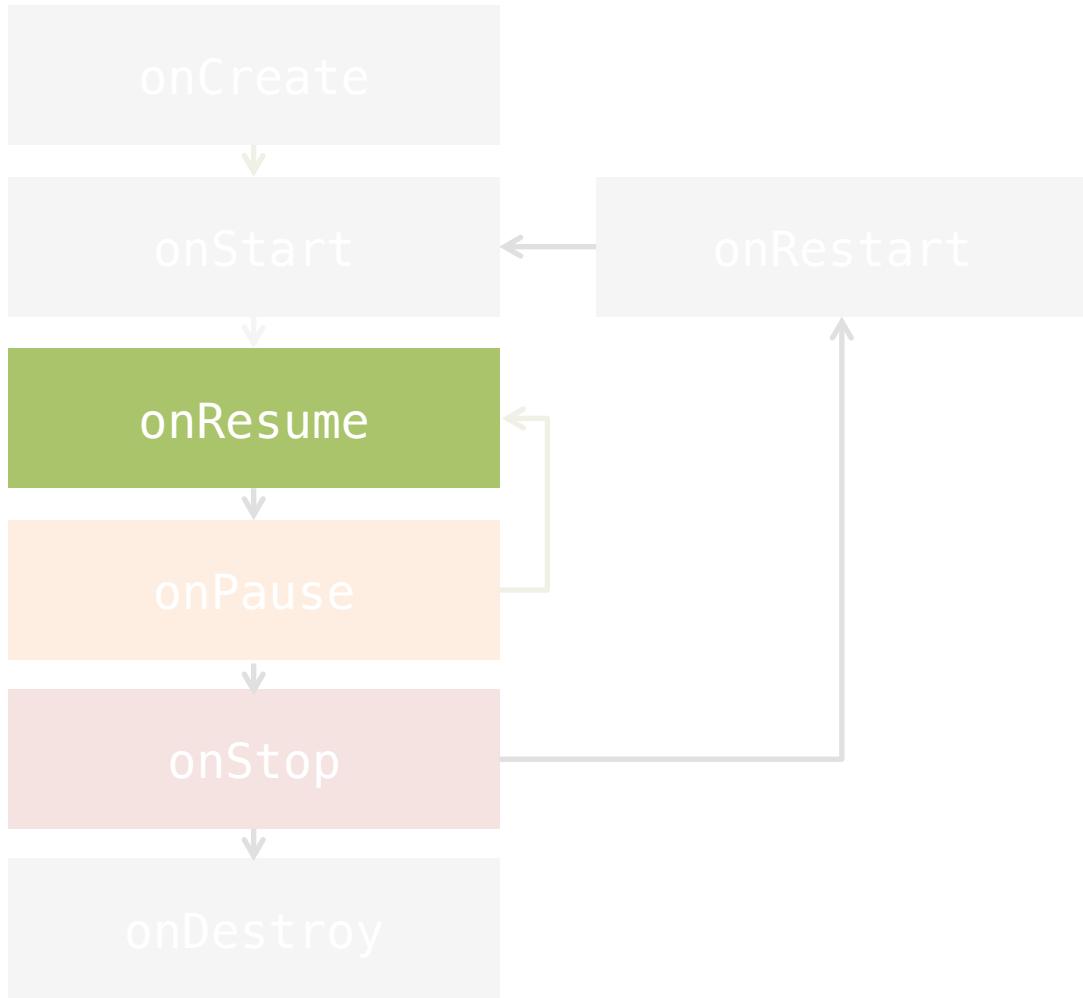
## Cas d'utilisation

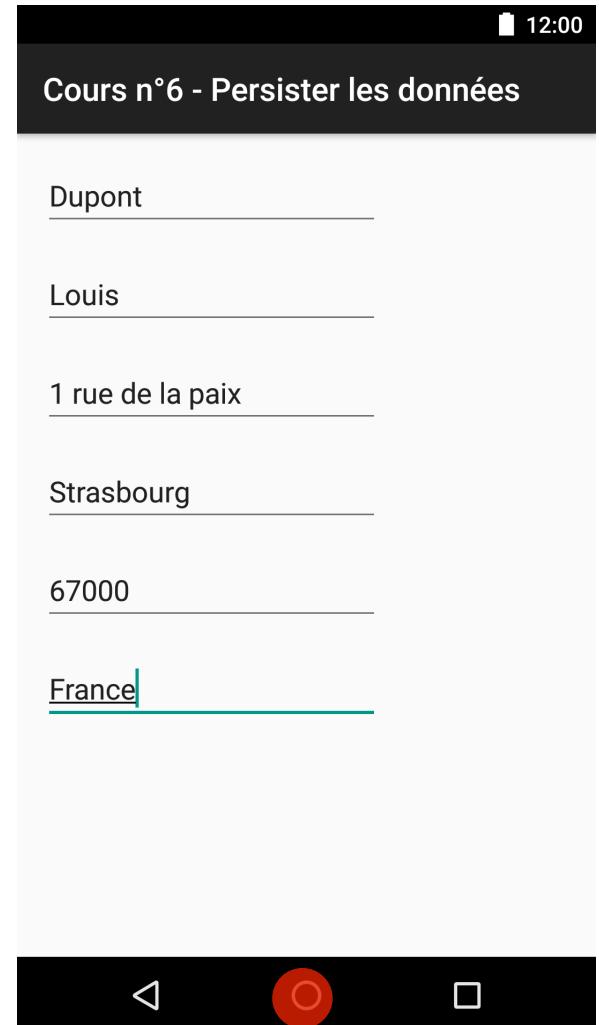
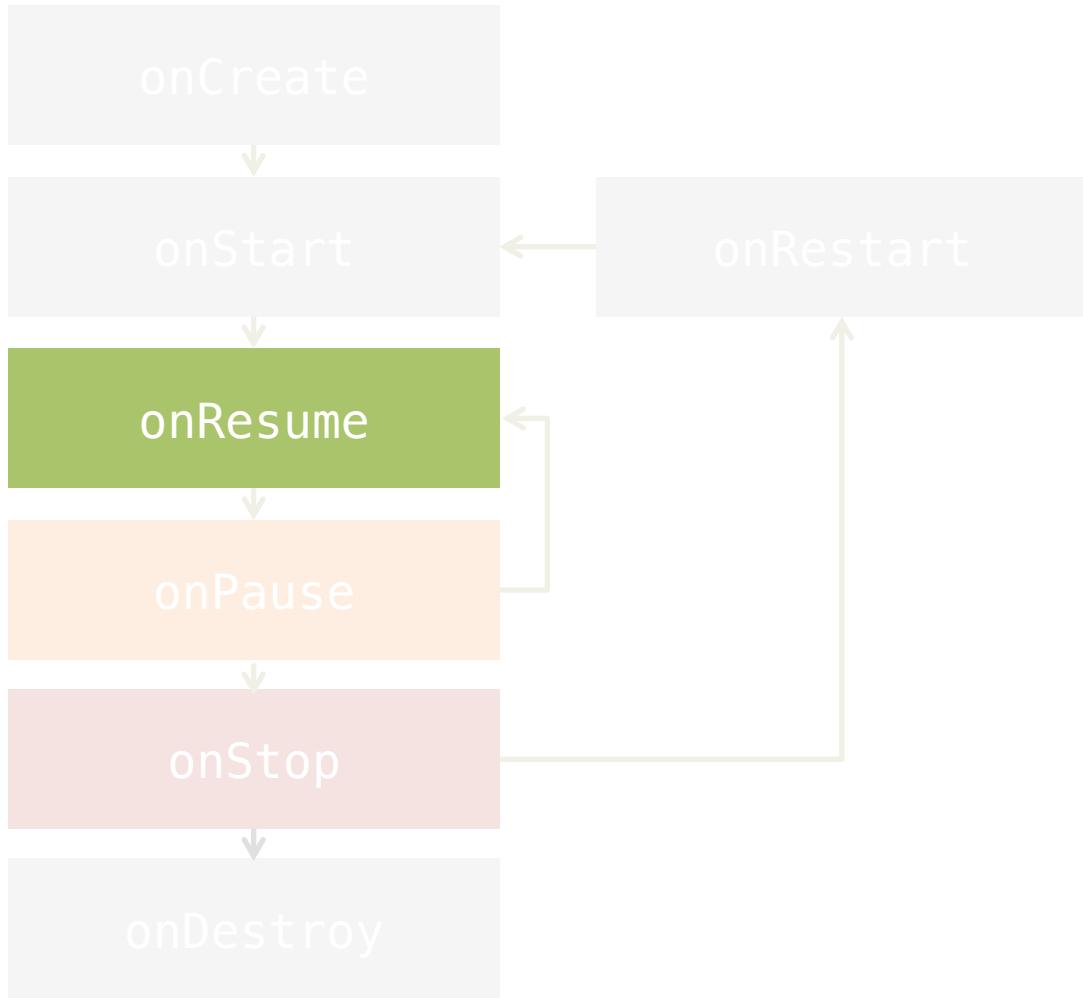
- Permettre à l'utilisateur de retrouver l'application comme il l'avait laissée
- Eviter d'avoir à appeler un webservice pour récupérer des données à chaque recréation d'Activity

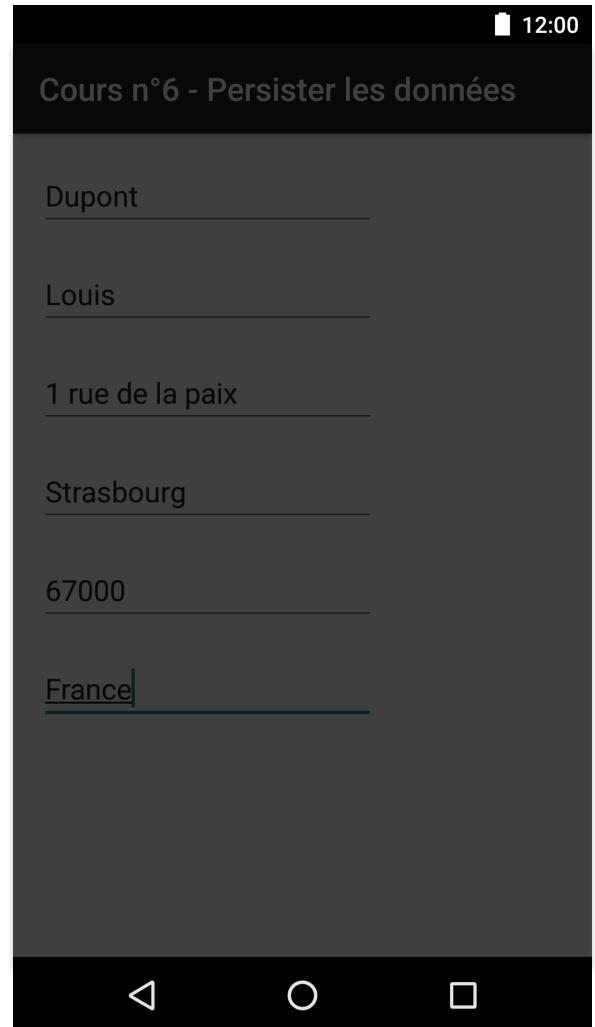
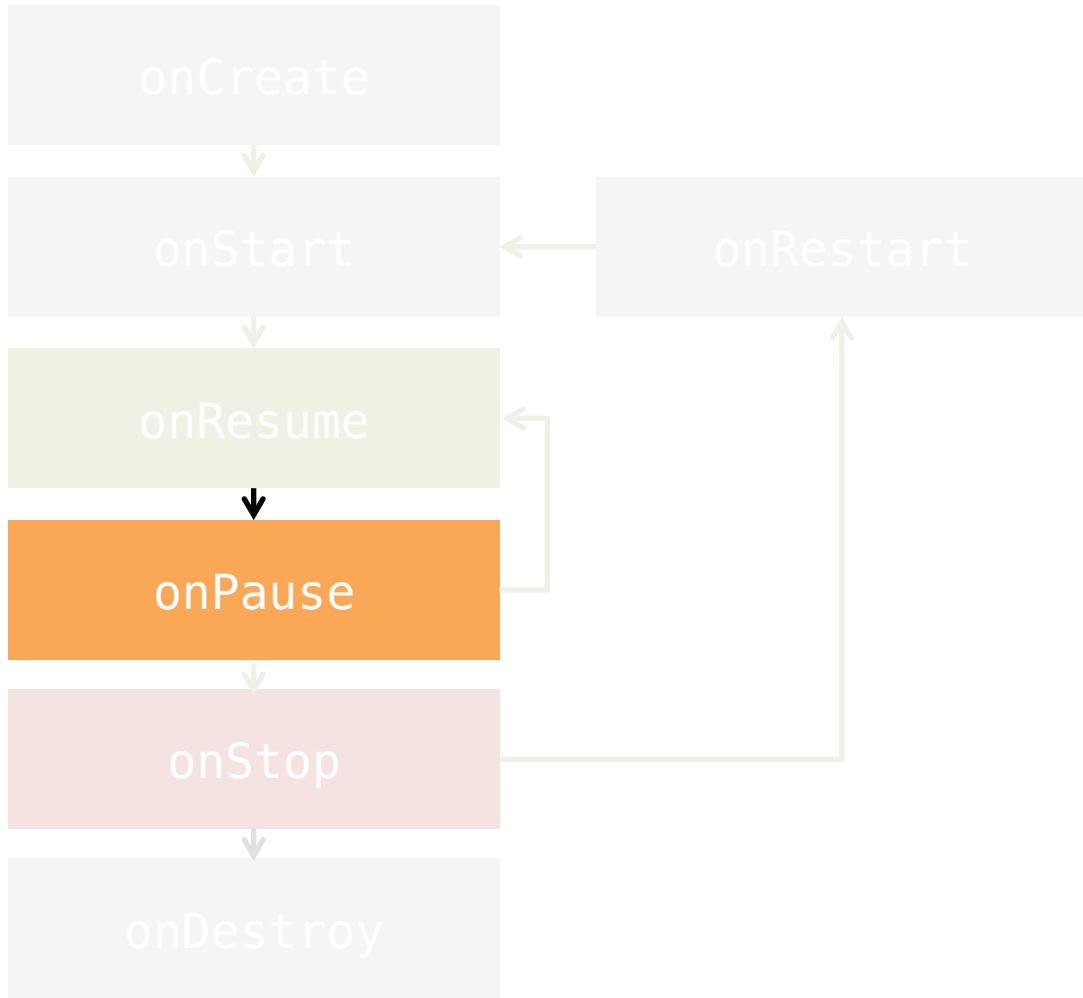
```
public class MyActivity extends Activity {  
    /**  
     * Placer dans le Bundle outState les données pour  
     * sauvegarder l'état de l'Activity  
     */  
    @Override  
    public void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
    }  
}
```

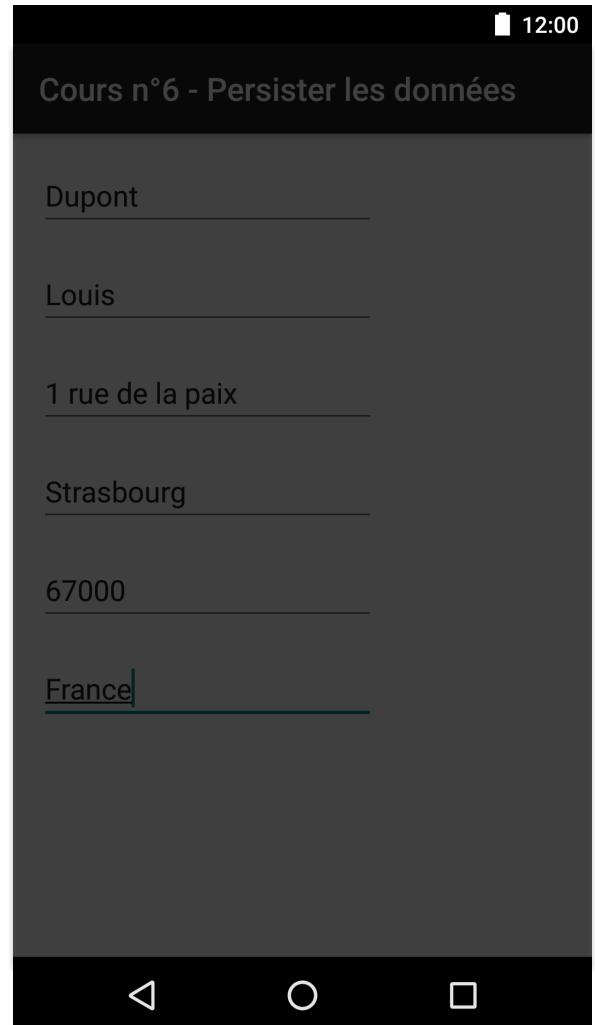
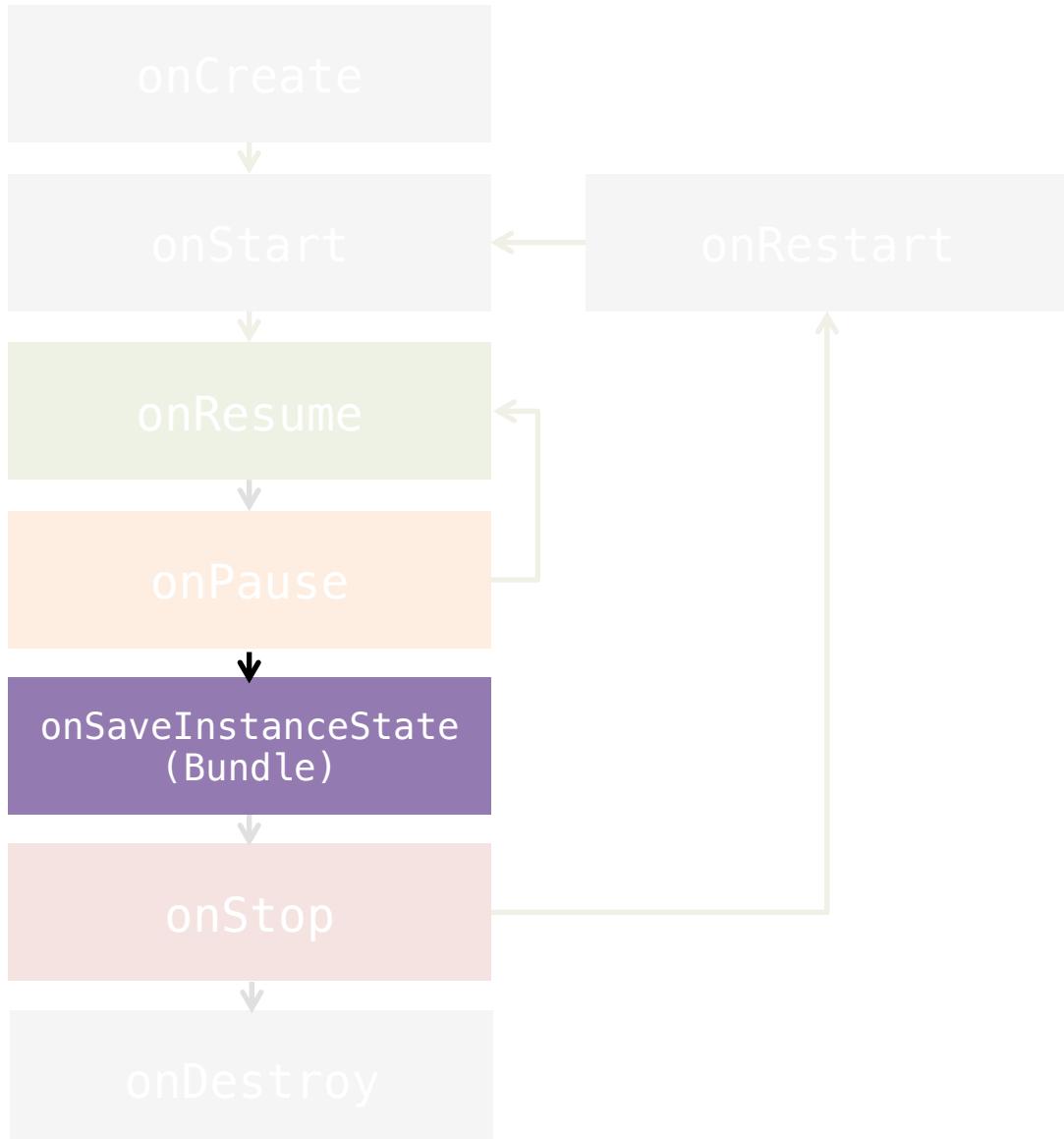
```
public class MyActivity extends Activity {  
    /**  
     * Placer dans le Bundle outState les données pour  
     * sauvegarder l'état de l'Activity  
     */  
    @Override  
    public void onSaveInstanceState(Bundle outState) {  
        // Sauvegarde l'état de chaque View du layout  
        // possédant un id  
        super.onSaveInstanceState(outState);  
    }  
}
```

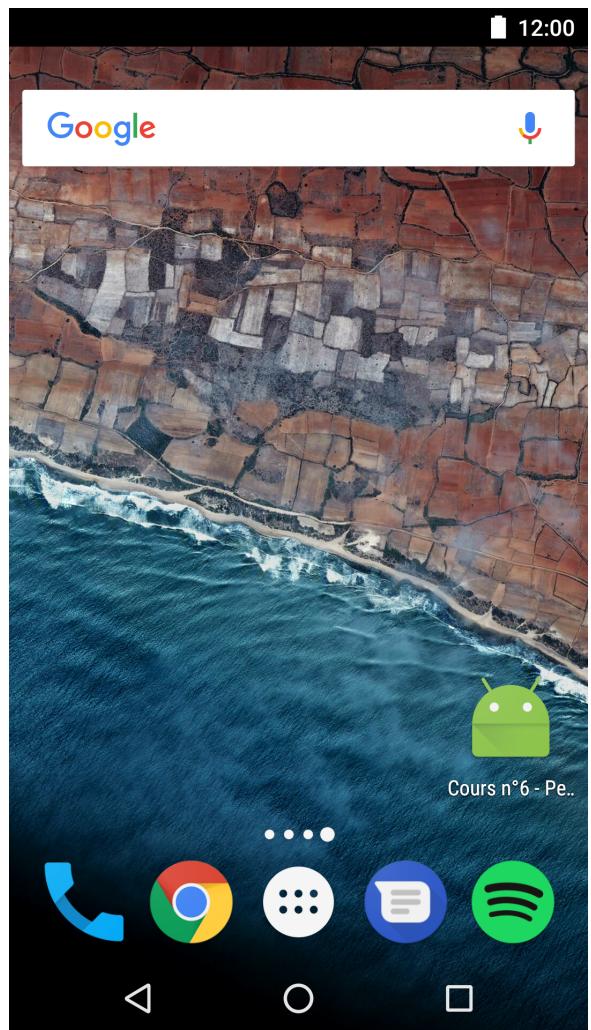
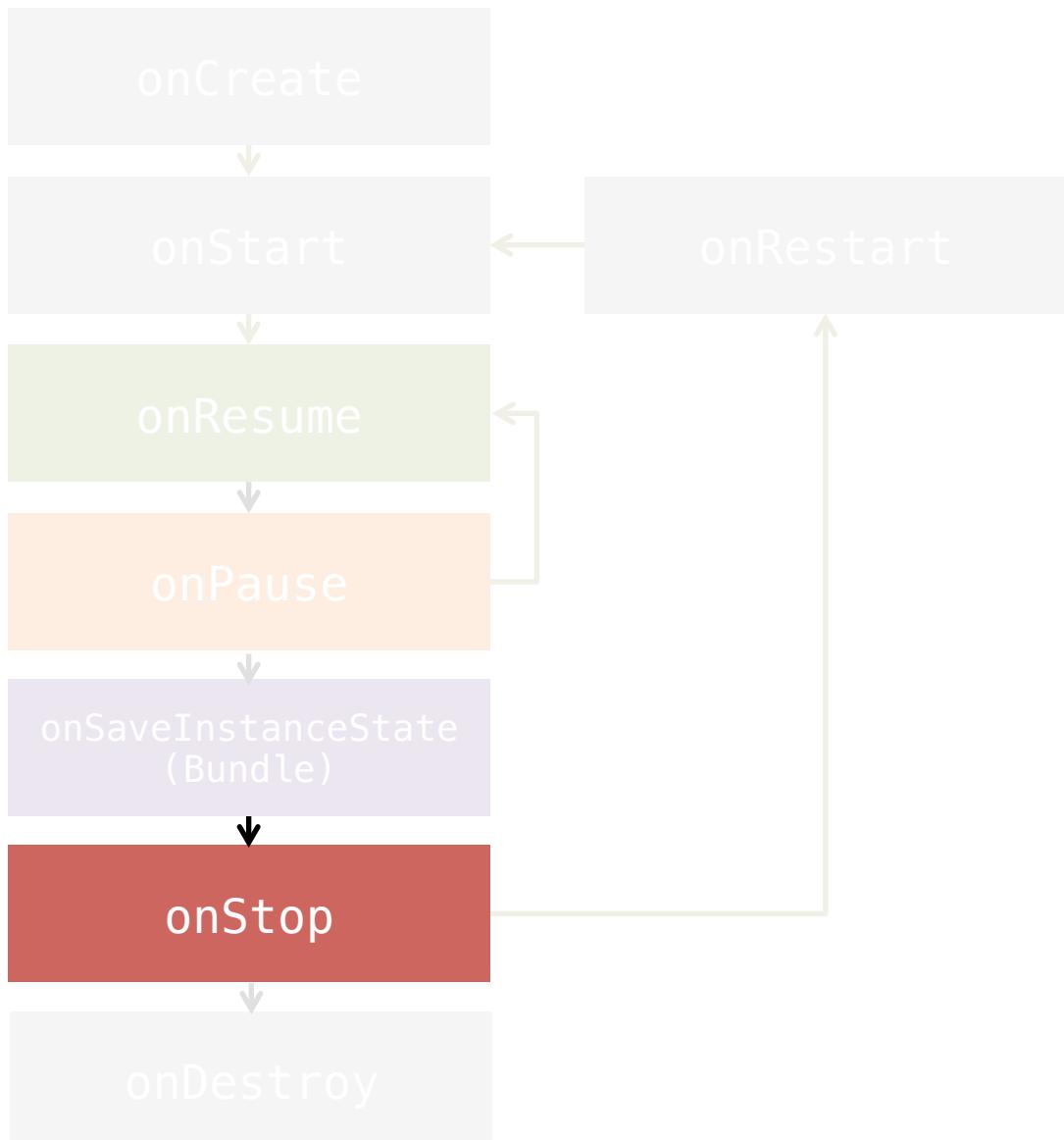
```
public class MyActivity extends Activity {  
    /**  
     * @param savedInstanceState Bundle de onSaveInstanceState()  
     */  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
    }  
  
    /**  
     * @param savedInstanceState Bundle de onRestoreInstanceState()  
     */  
    @Override  
    protected void onRestoreInstanceState(Bundle savedInstanceState) {  
        super.onRestoreInstanceState(savedInstanceState);  
    }  
  
    /**  
     * Placer dans le Bundle outState les données pour  
     * sauvegarder l'état de l'Activity  
     */  
    @Override  
    public void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
    }  
}
```

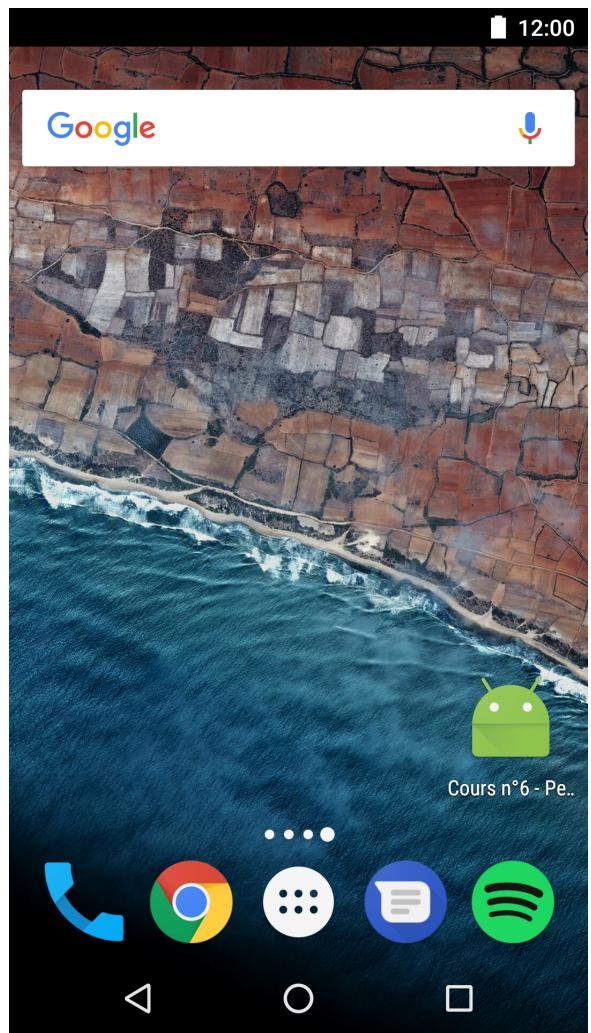
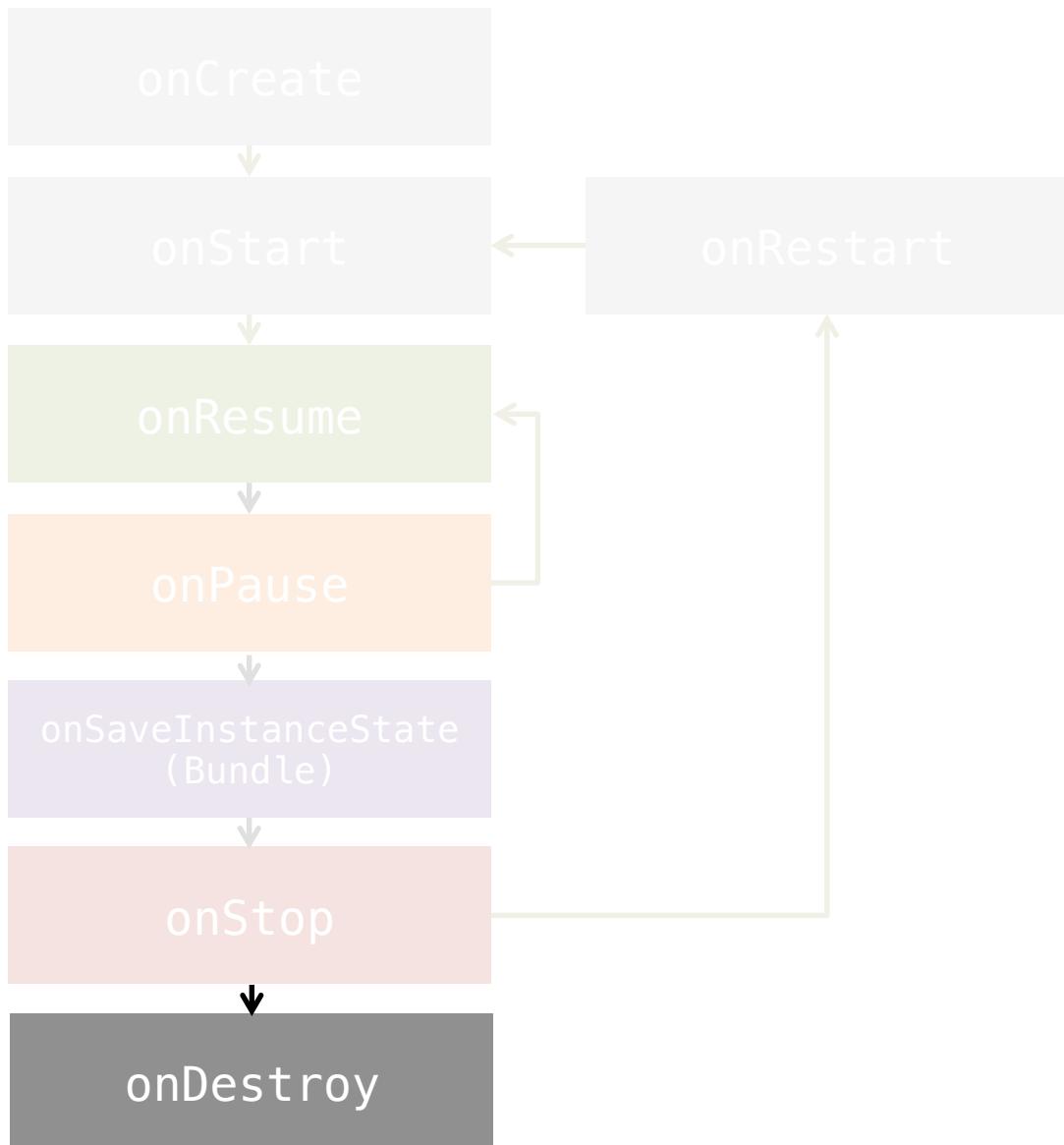


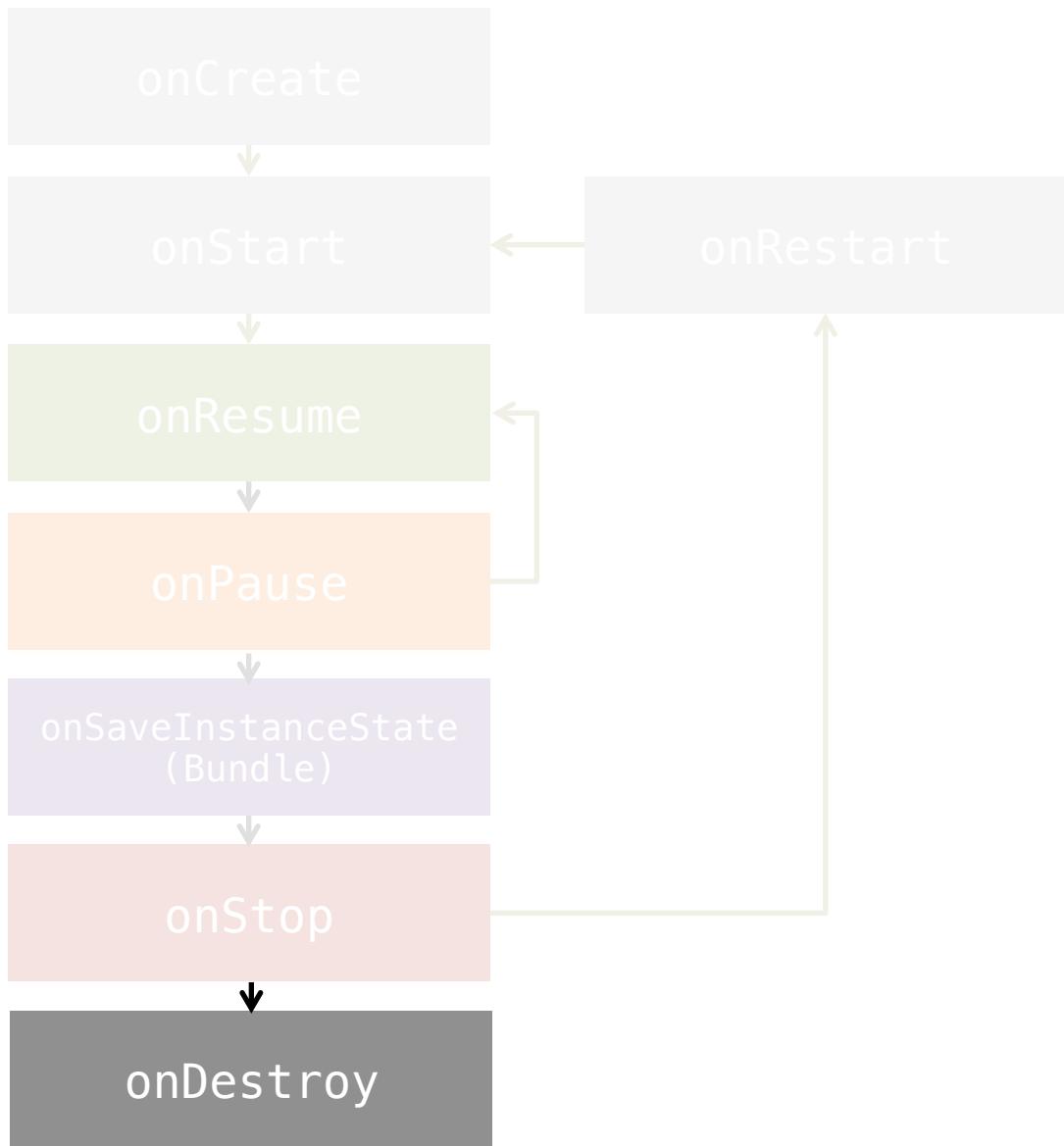


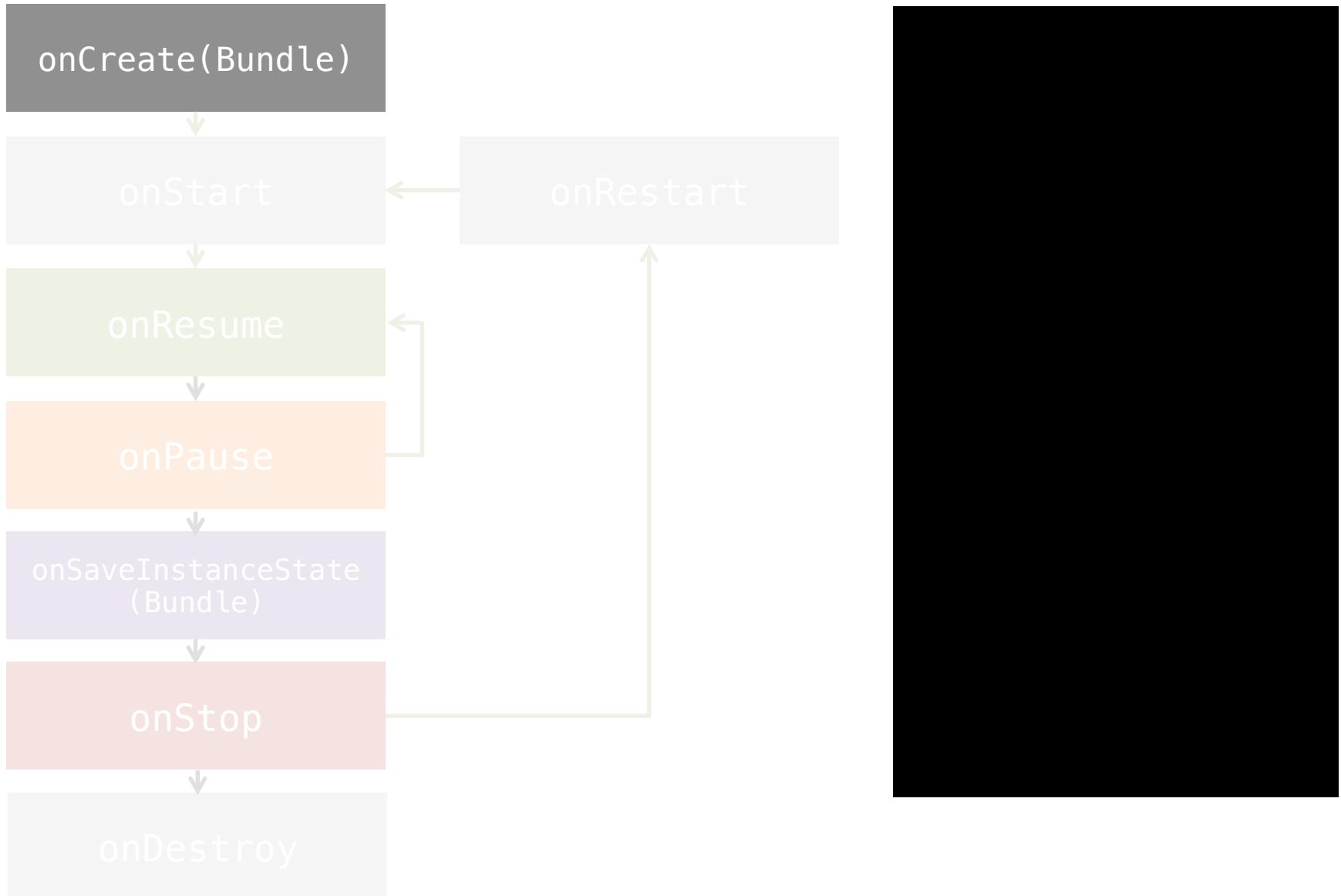


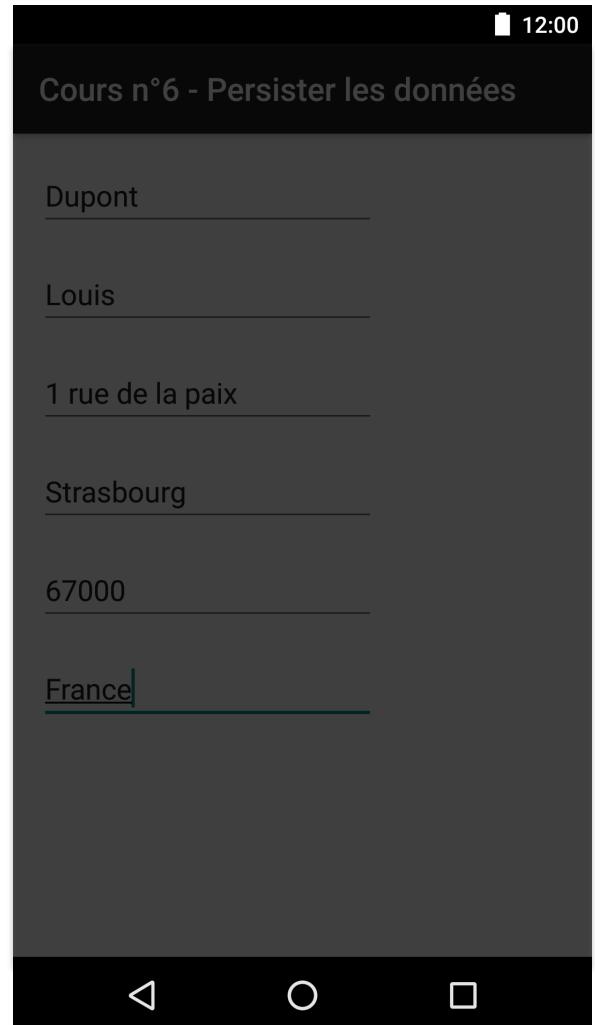
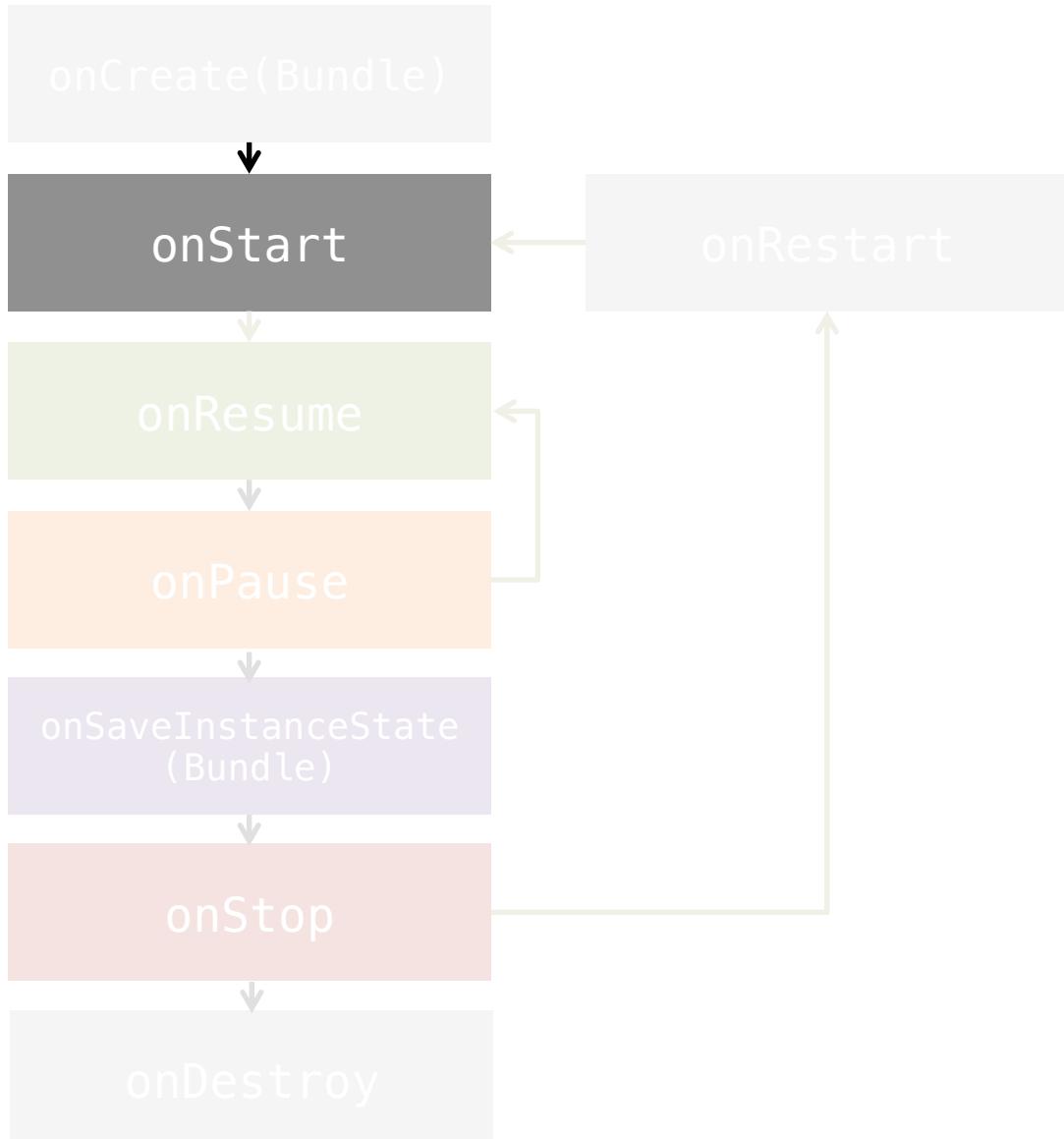


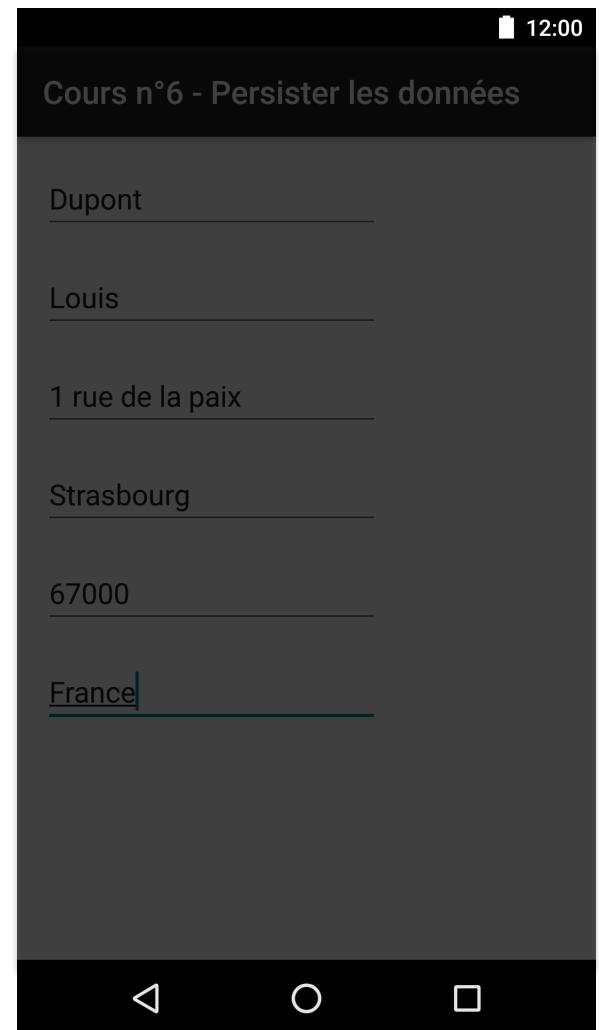
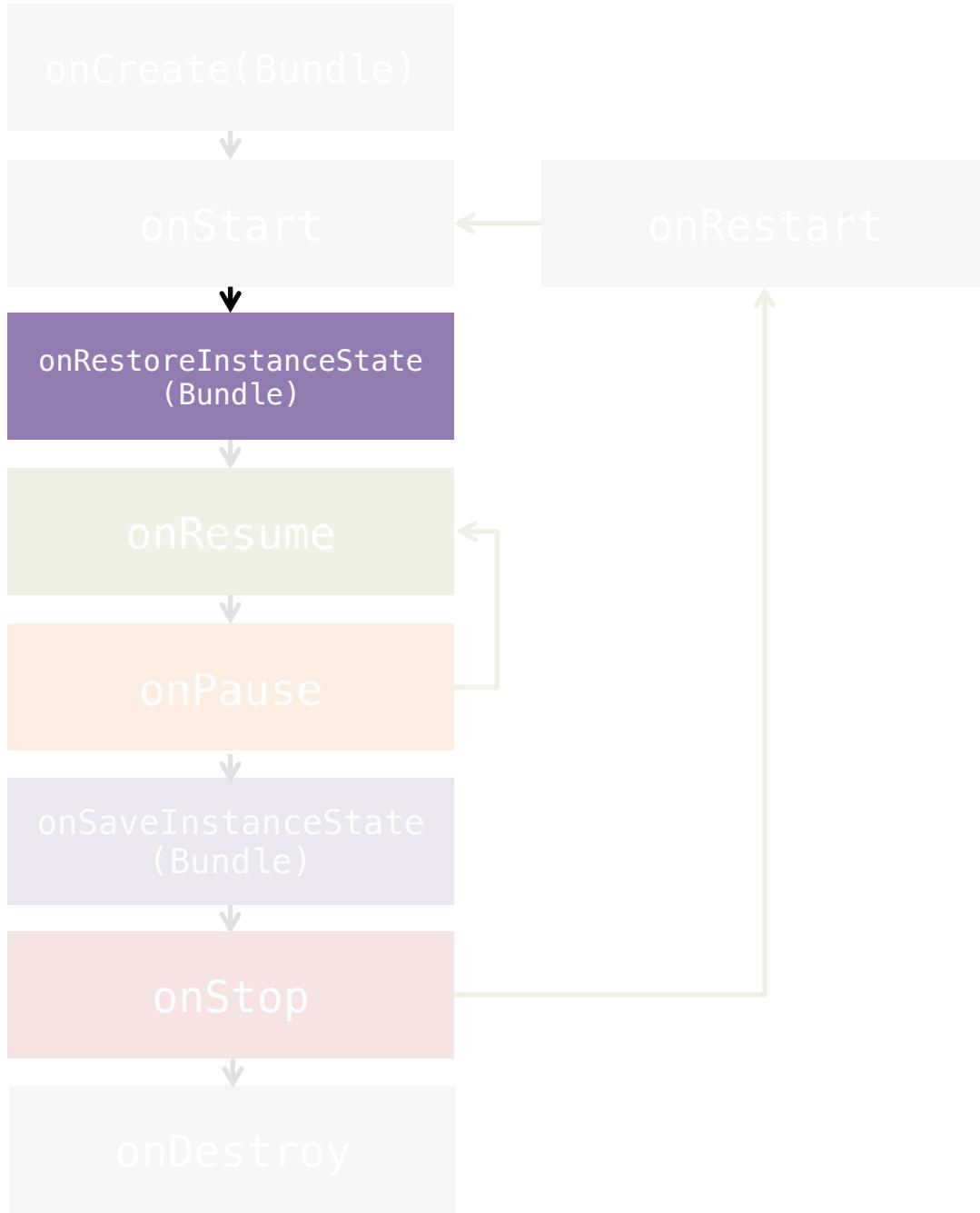


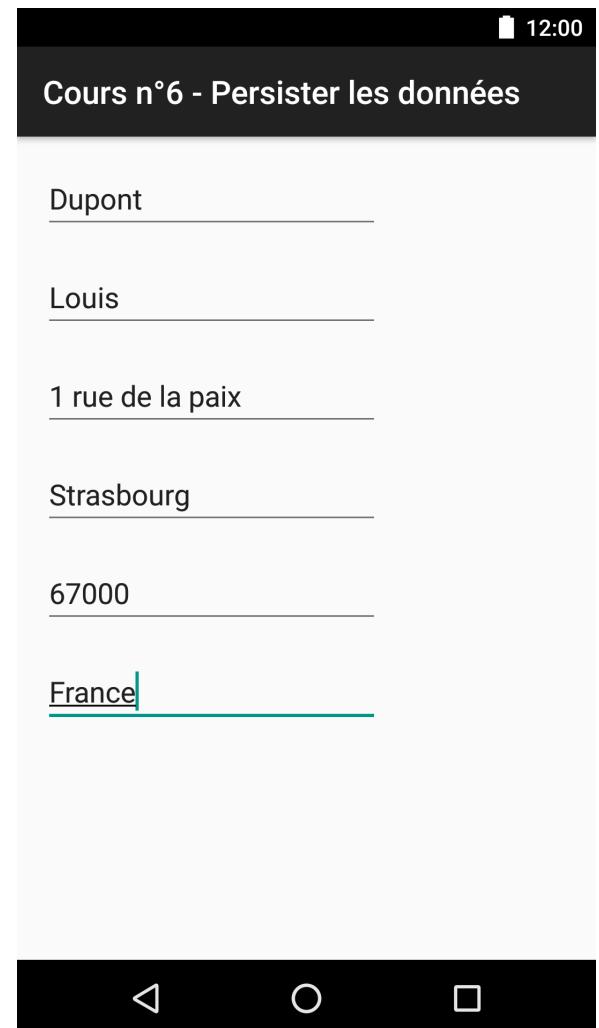
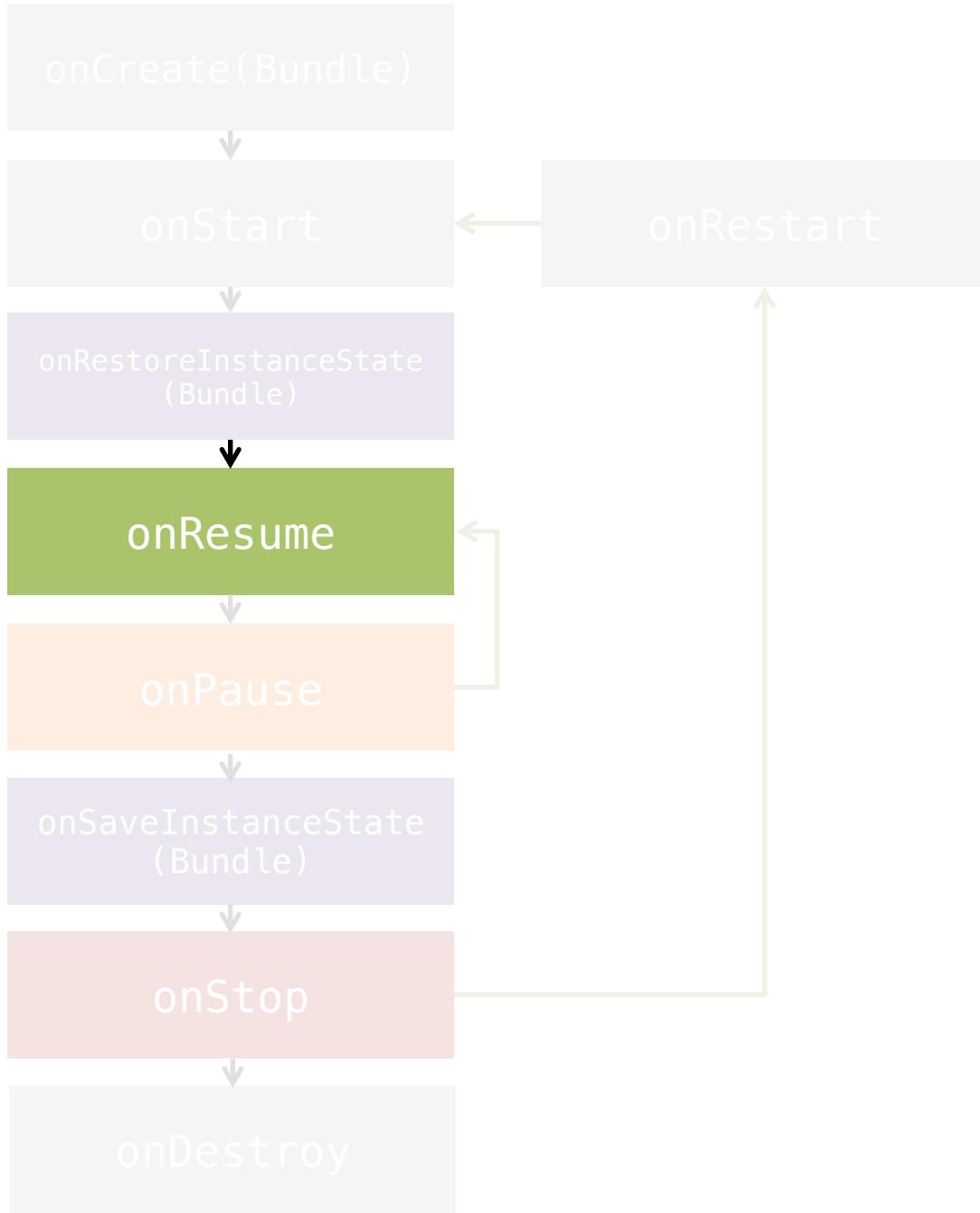












# SavedInstanceState

```
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
    // Sauvegarde des données chargées par le webservice  
    outState.putSerializable("data", neighbourhoods);  
}
```

# SavedInstanceState

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    ...  
  
    if (savedInstanceState == null) {  
        // Première création de l'Activity  
        callWebservice();  
    } else {  
        // Activity déjà créée, récupération des données sauvegardées  
        neighbourhoods = savedInstanceState.getSerializable("data");  
        fillList();  
    }  
}  
  
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
    // Sauvegarde des données chargées par le webservice  
    outState.putSerializable("neighbourhoods", neighbourhoods);  
}
```

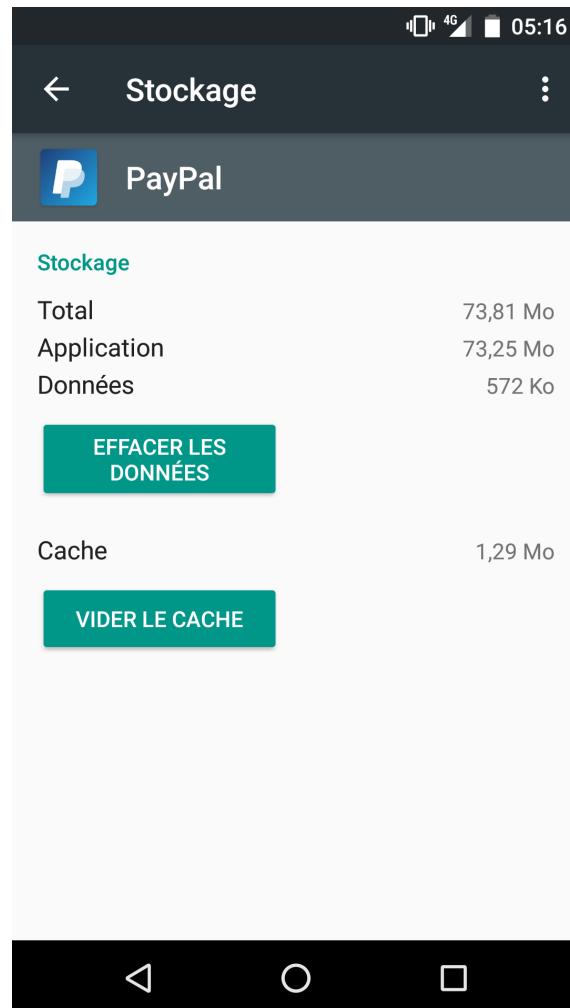
# SavedInstanceState



# SharedPreferences

- Interface permettant de stocker des données simples de type clé-valeur
- Données enregistrées dans un fichier XML accessible uniquement par l'application
- Données manipulées à travers des méthodes de lecture et d'écriture fournies par l'interface
- Les données persistent jusqu'à désinstallation de l'application ou suppression à partir des paramètres du téléphone

# SharedPreferences : Suppression

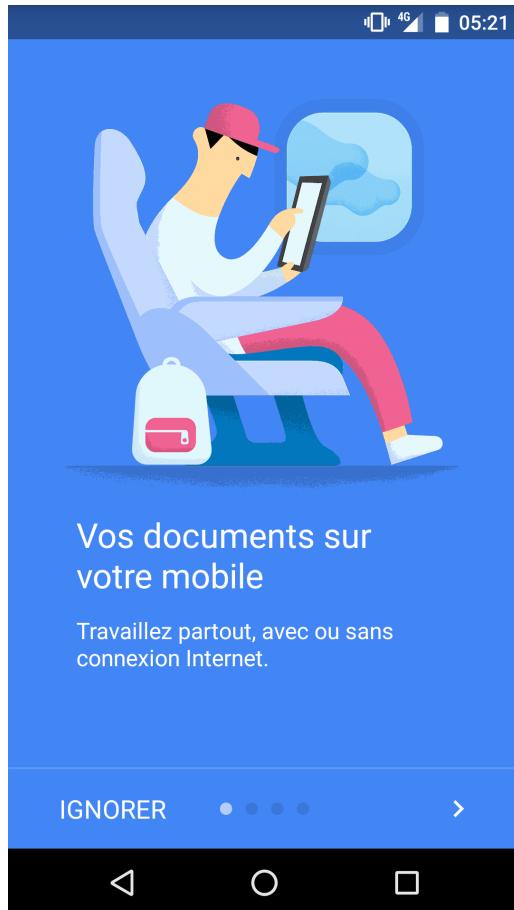


Infos appli → Stockage / Paramètres → Application

# SharedPreferences : Cas d'utilisation

- Sauvegarder l'identifiant de l'utilisateur pour reconnexion automatique
- Sauvegarder des flags pour affichage cinématique de première connexion ou tutoriel
- Sauvegarder des choix de l'utilisateur qu'on ne souhaite pas redemander pour chaque session
- Utiliser en arrière plan par Android pour gérer les écrans paramètre d'une application

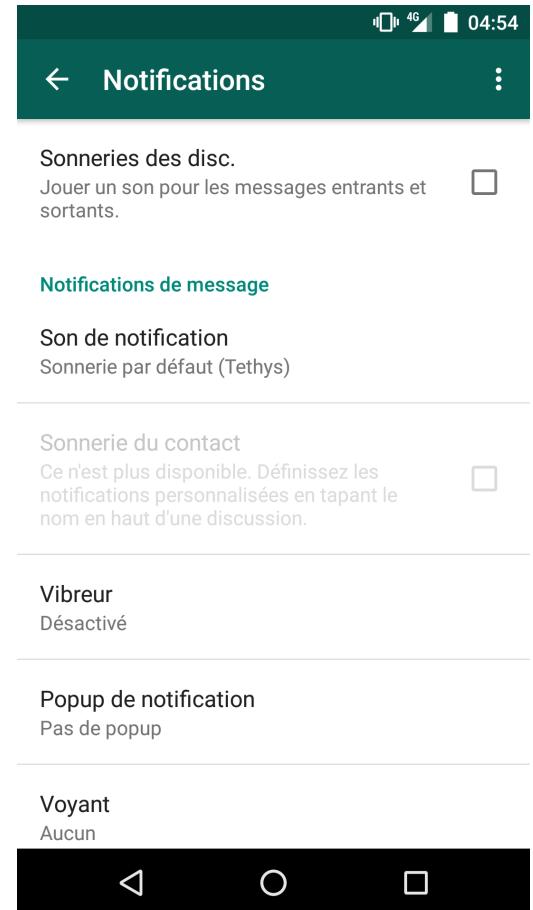
# SharedPreferences : Cas d'utilisation



Première connexion  
Google Docs



Option de lecture Spotify



Paramètre notifications  
WhatsApp

# SharedPreferences

```
// Définition du fichier de stockage à utiliser
SharedPreferences pref = getSharedPreferences("ensiie", MODE_PRIVATE);
// Interface permettant de modifier le fichier
SharedPreferences.Editor editor = pref.edit();
// Insertion / remplacement de données
editor.putString("my_key", "my_value");
editor.putBoolean("my_boolean", true);
editor.putInt("my_integer", 1);
// Sauvegarde
editor.apply();
```

# SharedPreferences

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="my_boolean" value="true" />
    <int name="my_integer" value="1" />
    <string name="my_key">my_value</string>
</map>
```

Fichier xml crée en interne par le système

# SharedPreferences

```
// Définition du fichier de stockage à utiliser
SharedPreferences pref = getSharedPreferences("ensiie", MODE_PRIVATE);
// Récupération de données sauvegardées
// et définition de valeurs par défaut si elles n'existent pas
String myValue = pref.getString("my_key", null);
boolean myBoolean = pref.getBoolean("my_boolean", false);
int myInteger = pref.getInt("my_integer", 0);
```

# Base de données SQLite

- Base de données open source embarquée dans chaque téléphone Android
- L'accès aux données se fait à travers l'implémentation de la classe SQLiteOpenHelper et nécessite l'utilisation d'un background Thread
- Données pouvant être accessibles pour d'autres applications en utilisant un ContentProvider
- Les données persistent jusqu'à désinstallation de l'application ou suppression à partir des paramètres du téléphone

# Base de données SQLite

## Cas d'utilisation

- Sauvegarder et/ou partager des données complexes
- Effectuer des opérations de filtre ou de recherche sur un ensemble de données

```
public class DatabaseHelper extends SQLiteOpenHelper {  
  
    // Version de la base de données  
    private static final int DATABASE_VERSION = 1;  
  
    // Nom de la base de données  
    private static final String DATABASE_NAME = "ensiie";  
  
    // Initialisation du helper  
    public DatabaseHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_NAME = "ensiie";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Appelé lors de la création de la base de données
    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    // Appelé lorsque la base de données à besoin de se mettre à
    // jour
    @Override
    public void onUpgrade(SQLiteDatabase db, int old, int new) {
    }
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 1;

    private static final String DATABASE_NAME = "ensiie";

    // Définition des tables
    interface Tables {
        String CONTACT = "contact";
    }

    // Définition des colonnes pour chaque table
    interface ContactColumns extends BaseColumns {
        String CONTACT_NAME = "contact_name";
        String CONTACT_PHONE = "contact_phone";
    }

    public DatabaseHelper(Context context) { ... }

    ...
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL("CREATE TABLE " + Tables.CONTACT + " ("  
            + BaseColumns._ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"  
            + ContactColumns.CONTACT_NAME + " TEXT,"  
            + ContactColumns.CONTACT_PHONE + " TEXT)");  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int old, int new) {  
        db.execSQL("DROP TABLE IF EXISTS " + Tables.CONTACT);  
        onCreate(db);  
    }  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
    //region Définition des opérations CRUD (Create, Read, Update, Delete)  
  
    // Ajout d'un contact  
    public void addContact(Contact contact) { }  
  
    // Récupération de tous les contacts  
    public ArrayList<Contact> getAllContact() { }  
  
    // Récupération d'un contact  
    public Contact getContact(String name, String phone) { }  
  
    // Mise à jour d'un contact  
    public int updateContact(Contact contact) { }  
  
    // Suppression d'un contact  
    public int deleteContact(Contact contact) { }  
  
    //endregion  
}
```

```
public class Contact {  
    private int id;  
    private String name;  
    private String phone;  
  
    public Contact(String name, String phone) { ... }  
  
    public Contact(int id, String name, String phone) { ... }  
  
    public int getId() { ... }  
  
    public void setId(int id) { ... }  
  
    public String getName() { ... }  
  
    public void setName(String name) { ... }  
  
    public String getPhone() { ... }  
  
    public void setPhone(String phone) { ... }  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Ajout d'un contact  
    public void addContact(Contact contact) {  
        // Ouverture de la base de données en écriture  
        SQLiteDatabase db = getWritableDatabase();  
  
        // Construction de la donnée à insérer  
        ContentValues values = new ContentValues();  
        values.put(ContactColumns.CONTRACT_NAME, contact.getName());  
        values.put(ContactColumns.CONTRACT_PHONE, contact.getPhone());  
  
        // Méthode helper pour insérer une donnée en base  
        db.insert(Tables.CONTRACT, null, values);  
        db.close();  
    }  
  
    ...  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Récupération de tous les contacts  
    public ArrayList<Contact> getAllContact() {  
        // Ouverture de la base de données en lecture  
        SQLiteDatabase db = getReadableDatabase();  
  
        // Construction de la requête à la main  
        String selectQuery =  
            "SELECT "  
            + BaseColumns._ID + ", "  
            + ContactColumns.CONTRACT_NAME + ", "  
            + ContactColumns.CONTRACT_PHONE + ", "  
            + "FROM " + Tables.CONTRACT;  
        db.rawQuery(selectQuery, null);  
        ...  
    }  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Récupération de tous les contacts  
    public ArrayList<Contact> getAllContact() {  
        // Ouverture de la base de données en lecture  
        SQLiteDatabase db = getReadableDatabase();  
  
        // Helper pour construire la requête  
        db.query(  
            Tables.CONTRACT, // Table sur laquelle faire la requête  
            // Colonne que l'on souhaite récupérer  
            new String[]{  
                BaseColumns._ID,  
                ContactColumns.CONTRACT_NAME,  
                ContactColumns.CONTRACT_PHONE  
            },  
            null, // where  
            null, // groupBy  
            null, // having  
            null, // orderBy  
            null // limit  
        );  
        ...  
    }  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Récupération de tous les contacts  
    public ArrayList<Contact> getAllContact() {  
        // Ouverture de la base de données en lecture  
        SQLiteDatabase db = getReadableDatabase();  
  
        // Helper pour construire la requête  
        Cursor cursor = db.query(  
            Tables.CONTRACT, // Table sur laquelle faire la requête  
            // Colonne que l'on souhaite récupérer  
            new String[] {  
                BaseColumns._ID,  
                ContactColumns.CONTRACT_NAME,  
                ContactColumns.CONTRACT_PHONE  
            },  
            null, // where  
            null, // groupBy  
            null, // having  
            null, // orderBy  
            null // limit  
        );  
        ...  
    }  
}
```

# Cursor

- Représente le résultat d'une requête
- Contient un pointeur permettant de lire une ligne à la fois du résultat

# Cursor

Curseur →

_ID	CONTACT_NAME	CONTACT_PHONE
1	Jean	0623540394
2	Pierre	0627840308
3	John	0625442197

# Cursor

_ID	CONTACT_NAME	CONTACT_PHONE
1	Jean	0623540394
2	Pierre	0627840308
3	John	0625442197

cursor.moveToFirst()

# Cursor

_ID	CONTACT_NAME	CONTACT_PHONE
1	Jean	0623540394
2	Pierre	0627840308
3	John	0625442197

```
cursor.moveToFirst()
Contact contact = new Contact(
    cursor.getInt(cursor.getColumnIndex(BaseColumns._ID)),
    cursor.getString(cursor.getColumnIndex(ContactColumns.CONTACT_NAME)),
    cursor.getString(cursor.getColumnIndex(ContactColumns.CONTACT_PHONE)));
```

# Cursor

_ID	CONTACT_NAME	CONTACT_PHONE
1	Jean	0623540394
2	Pierre	0627840308
3	John	0625442197

```
cursor.moveToFirst()
Contact contact = new Contact(
    cursor.getInt(cursor.getColumnIndex(BaseColumns._ID)),
    cursor.getString(cursor.getColumnIndex(ContactColumns.CONTACT_NAME)),
    cursor.getString(cursor.getColumnIndex(ContactColumns.CONTACT_PHONE)));
cursor.moveToNext()
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Récupération de tous les contacts  
    public ArrayList<Contact> getAllContact() {  
        ...  
        ArrayList<Contact> contacts = new ArrayList<>();  
        if (cursor.moveToFirst()) {  
            do {  
                Contact contact = new Contact(...);  
                contacts.add(contact);  
            } while (cursor.moveToNext());  
        }  
        cursor.close();  
        db.close();  
        return contacts;  
    }  
    ...  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Récupération d'un contact  
    public Contact getContact(String name, String phone) {  
        // Ouverture de la base de données en lecture  
        SQLiteDatabase db = getReadableDatabase();  
  
        Cursor cursor = db.query(  
            // Table sur laquelle faire la requête  
            Tables.CONTACT,  
            // Colonne que l'on souhaite récupérer  
            new String[] {  
                BaseColumns._ID,  
                ContactColumns.CONTACT_NAME,  
                ContactColumns.CONTACT_PHONE  
            },  
            // where  
            ContactColumns.CONTACT_NAME + " = ? AND " +  
            ContactColumns.CONTACT_PHONE + " = ? ",  
            new String[] {name, phone}, // remplace les ? du where  
            null, // groupBy  
            null, // having  
            null, // orderBy  
            null // limit  
        );  
  
        ...  
    }  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Récupération d'un contact  
    public Contact getContact(String name, String phone) {  
        ...  
        Contact contact = null;  
  
        if (cursor.moveToFirst()) {  
            contact = new Contact(...)  
        }  
  
        cursor.close();  
        db.close();  
        return contact;  
    }  
    ...  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Mise à jour d'un contact  
    public int updateContact(Contact contact) {  
        // Ouverture de la base de données en écriture  
        SQLiteDatabase db = getWritableDatabase();  
  
        // Construction de la donnée à mettre à jour  
        ContentValues values = new ContentValues();  
        values.put(ContactColumns.CONTACT_NAME, contact.getName());  
        values.put(ContactColumns.CONTACT_PHONE, contact.getPhone());  
  
        // Méthode helper pour mettre à jour une donnée en base  
        int res = db.update(  
            Tables.CONTACT,  
            values,  
            BaseColumns.ID + " = ?"  
            ,  
            new String[]{String.valueOf(contact.getId())}  
        );  
  
        db.close();  
        return res;  
    }  
    ...  
}
```

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    ...  
  
    // Suppression d'un contact  
    public int deleteContact(Contact contact) {  
        // Ouverture de la base de données en écriture  
        SQLiteDatabase db = getWritableDatabase();  
  
        // Méthode helper pour mettre à jour une donnée en base  
        int res = db.delete(  
            Tables.CONTRACT,  
            BaseColumns._ID + " = ?" ,  
            new String[]{String.valueOf(contact.getId())}  
        );  
        db.close();  
        return res;  
    }  
}
```

```
private DatabaseHelper databaseHelper = new DatabaseHelper(this);

class MyAsyncTask extends AsyncTask<Void, Void, ArrayList<Contact>>{

    @Override
    protected ArrayList<Contact> doInBackground(Void[] params) {
        databaseHelper.addContact(new Contact("Jean", "0623540394"));
        databaseHelper.addContact(new Contact("Pierre", "0627840308"));
        databaseHelper.addContact(new Contact("John", "0625442197"));
        return databaseHelper.getAllContact();
    }
}
```