# Cours 13

Appel C# à partir du Java

- Tant que possible, il est préférable de faire l'appel à partir du C# vers le Java. Même si celui-ci sera par réflexion, il sera synchrone et la valeur de retour sera automatiquement convertie en primitive C# ou encapsuler avec **`AndroidJavaObject`** permettant d'obtenir plus d'informations avec d'autres appels du C# vers la Java.

- Ceci n’est pas toujours disponible, surtout si le code implique un appel asynchrone (comme **`runOnUiThread`**).
- Unity offre une méthode permettant d’envoyer un message à un *game object* C# à partir du Java et n’importe quel fil d’exécution:

```
UnityPlayer.UnitySendMessage(
    gameObjectName,
    methodName,
    message)
```

- Cette méthode vient avec plusieurs limitations. Vous devez connaître le nom du *game object* et le nom de la méthode, ce qui comporte autant de risque que la réflexion à partir du C#. Aussi, la méthode DOIT prendre un seul argument et il doit être de type **`string`**. Vous ne pouvez pas n'envoyer aucun message (**`null`**), vous devez absolument donner une chaîne de caractères.

- **`UnitySendMessage`** mettra alors votre appel en queue, et celui-ci sera fait sur le fil principale Unity dès que possible. Par contre, **`UnitySendMessage`** retourne immédiatement, n'attend pas le résultat, et ne permet pas de retourner de valeurs.
- Malgré ces limitations, **`UnitySendMessage`** est généralement suffisant pour tous les besoins. Par exemple, si vous voulez envoyer un objet plutôt qu'un message, il suffit de le garder dans une variable Java puis de le récupérer à partir du C# à l'aide de **`AndroidJavaObject`** et/ou **`AndroidJavaClass`**.

- Vous pouvez également extraire les informations à utiliser avec **`UnitySendMessage`** à partir d'un **`Action<string>`**, ce qui permet de protéger contre les noms d'objets invalides, les changements de nom de méthode, et enforcer que celle-ci prend bel et bien un paramètre **`string`**.

- Dans le C#, obtenez le nom et la méthode à partir de l'**`Action<string>`** et envoyez-les au Java.

```
public void ExampleAsynchrone(Action<string> callback) {

    using (AndroidJavaClass cls = new AndroidJavaClass("ca.bart.demo.JavaClass")) {
        cls.CallStatic("MethodeJava",
            ((Component)callback.Target).gameObject.name,
            callback.Method.Name);
    }
}
```

- Dans la classe Java, vous pouvez maintenant effectuer du code asynchrone et mettre le résultat dans un objet complexe:

```
public static ObjetComplexe resultat;


public void MethodeJava(final String gameObject, final String methodName) {

    UnityPlayer.currentActivity.runOnUiThread(new Runnable() {

        @Override
        public void run() {

            resultat = new ObjectComplexe();

            UnityPlayer.UnitySendMessage(gameObject, methodName, /* nom de la variable */ "resultat");
        }
    });
}
```

- Finalement, la méthode qui a été passée comme **`Action<string>`** prend ce nom de variable et récupère l'objet.

```
public void Callback(string variableName) {

    using (AndroidJavaClass cls = new AndroidJavaClass("ca.bart.demo.JavaClass")) {
        using (AndroidJavaObject result = cls.GetStatic<AndroidJavaObject>(variableName)) {

            // faites votre travail avec le résultat!
        }
    }
}
```

- Cette façon de procéder fonctionne très bien pour bien des problèmes, par contre, elle devient plus complexe si on doit supporter plusieurs appels à la méthode (car le résultat d'un appel écrase le résultat du précédent).
- Aussi, contrairement à **`AndroidJavaObject`** qui réserve une référence et empêche l'objet Java de se faire effacer, cette méthode ne garantit pas que l'objet C# sera encore existant au moment de l'appel.