

Développement Mobile

Introduction



Principes de ce cours

2

- Un sujet de projet qui servira
 - à apprendre par vous-même
 - à vous proposer des cours en fonction de vos difficultés et/ou incompréhension
- Des cours d'introduction
 - IOS & Swift
 - Android & Java/Kotlin
- Des tutoriels/exPLICATION en TP



Projet : Travelling Tracker

3

- Garder une trace des lieux visités lors d'un périple
 - liste des lieux : nom, type, date
 - garder la position du lieu, l'afficher sur une carte
 - y associer une photo
 - retrouver la position d'un lieu d'après une photo, l'afficher sur une carte
- Sujet détaillé sur Moodle



Xcode & IOS developpement (1)

Langage Swift



Swift



Swift...

5



- Swift est un nouveau langage de programmation pour iOS, OSX, watchOS,
- Basé sur Objective-C, il intègre des mécanismes issus de C++ et de Python
- reprend les paramètres nommés de objective-C pour une « meilleure lisibilité »
- L'exemple de base : « Hello World »
 - ouvrir Xcode
 - créer un playground
 - tester l'exemple : `print("Hello, world!")`

Variables et constantes

6



- Let pour définir des constantes
- var pour définir des variables

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

- Typage dynamique

```
let implicitInteger = 70
let implicitDouble = 70.0
let explicitDouble: Double = 70
```

- Pas de conversion implicite => nécessité de conversion explicite

```
let label = "The width is "
let width = 94
let widthLabel = label + String(width)
```

Tableaux et dictionnaires

7



- Les tableaux et dictionnaires utilisent la syntaxe []

```
var shoppingList = ["catfish", "water", "tulips", "blue paint"]
shoppingList[1] = "bottle of water"
```

```
var occupations = [
    "Malcolm": "Captain",
    "Kaylee": "Mechanic",
]
occupations["Jayne"] = "Public Relations »"
```

- Les tableaux de Swift sont issus de la classe NSArray
d'objective-C => ce sont des objets
- initialisation par appel au constructeur :

```
var tab=[]
var tabint=[Int]()
var tabinit=[Int](count: 10, repeatedValue:0)
tabinit.count
```

Structure de contrôle

8



On retrouve les grands classiques : `if`, `switch`, `for-in`, `for`, `while` et `repeat-while`

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
print(teamScore)
```

```

let vegetable = "red pepper"
switch vegetable {
    case "celery":
        print("Add some raisins and make ants on a log.")
    case "cucumber", "watercress":
        print("That would make a good tea sandwich.")
    case let x where x.hasSuffix("pepper"):
        print("Is it a spicy \((x)?")
    default:
        print("Everything tastes good in soup.")
}

```

Le switch de Swift est plus « riche » que celui du C ou d'autres langage :

- s'arrête dès qu'un case est accepté
- possibilité d'avoir des patterns grâce aux affectations de contrôle
- gère les intervalles : case 1.. <5



Le `for-in` fonctionne comme celui de Python :

```
let interestingNumbers = [
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0
for (kind, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
print(largest)
```



Valeurs optionnelles

11



- Mécanisme propre à Swift (mécanisme similaire en Kotlin)
- S'apparente au `nil` de objective-c ou `NULL` de C mais valable pour toute variable
- Très utile par exemple pour indiquer qu'une fonction peut ne retourner aucune valeur plutôt que de faire retourner une valeur particulière
- Se marque avec un ?

```
var optionalName: String? = nil
var greeting = "Hello!"
if (optionalName != nil) {
    greeting = "Hello, \(optionalName!)"
}
```

- Par défaut une valeur optionnelle vaut `nil`
- On peut forcer l'évaluer d'une valeur optionnelle par `!`
- Certaines opérations (conversion notamment) peuvent renvoyer `nil` => nécessiter d'utiliser des valeurs optionnelles

```
let snumber = "123"
let n : Int? = Int(snumber)
if (n != nil) {
    print("snumber converti contient une valeur entière")
}
```

- On peut utiliser les affectations optionnelles pour contrôler :

```
let tnumber = "123"
if let nn = Int(tnumber) {
    print("tnumber converti contient \(nn)")
}
else{
    print("\tnumber) n'a pas pu être converti")
}
```



guard pour vérifier les optionnelles

13



L'opérateur *guard* sert initialement à gérer les cas d'erreur.

Il offre un moyen élégant de traiter les cas où une variable pourrait ne pas avoir de valeur et provoquer une erreur.

Comparons les 3 façons de traiter une valeur qui pourrait être indéfinie :

- « à l'ancienne » , c.-à.-d. comme en C, Java ou Objective C ;
- en utilisant les affectations conditionnelles de Swift ;
- avec l'opérateur *guard*.

Remarque : *guard* impose dans sa clause *else* une sortie du bloc ou de la fonction, par *break*, *continue*, *return* ou *throw* (*ou éventuellement par une fonction comme certaines fonctions d'erreur*)

Méthode utilisée dans les langages classiques

```
var x : Int? = some_func();
if (x == nil) || (x <= 0) {
    // pas de valeur valable, gérer ce cas d'erreur
    return
}
// faire quelque chose avec la valeur de x
x!.description
```

Inconvénients :

- la condition vérifie les valeurs qu'on ne veut pas : cela peut prêter à confusion, en particulier si il y a de multiples vérifications imbriquées
- pour le reste du code, il faudra forcer (unwrap) la valeur optionnelle x dans tout le reste du code



Méthode avec les affectations optionnelles

```
var x = some_func();
if let x = x, x > 0 {
    // faire quelque chose avec la valeur de x
    x.description
}
// pas de valeur valable, gérer ce cas d'erreur
return
```

Les inconvénients précédents ont été enlevés mais désormais le code qui doit s'exécuter normalement est dans une condition.

Cela peut nuire à la lisibilité et surtout devient complexe si il y a de multiples vérifications imbriquées.

Il est toujours préférable de vérifier toutes les pre-conditions en début de fonction.



Méthode avec *guard*

```
var x = some_func();
guard let x = x, x > 0 else {
    // pas de valeur valable, gérer ce cas d'erreur
    return
}
// faire quelque chose avec la valeur de x
x.description
```

Avantages :

- on vérifie la condition d'erreur
- les cas d'erreur pourront tous être vérifiés indépendamment en début de fonction, le reste du code étant la fonction elle-même
- après le guard, la variable x est désormais considérée comme non optionnelle et n'a pas besoin d'être forcée



Fonctions

17



- `func` permet de déclarer une fonction
- l'appel se fait classiquement par le nom de la fonction suivi de parenthèses avec la liste des paramètres
- les paramètres sont nommés,
- l'appel doit comporter le nom des paramètres,
- à la déclaration, les paramètres sont séparés du type de retour par `->`

```
func greet(name: String, day: String) -> String {  
    return "Hello \(name), today is \(day)."  
}  
greet(name: "Bob", day:"Tuesday")
```

- ❑ les tuples permettent de retourner plusieurs valeurs
- ❑ les fonctions admettent un nombre variable de paramètres qui seront rassemblés dans un tableau

```
func calculateStatistics(scores: UInt...) -> (moyenne:Float, min: UInt,  
max: UInt) {  
    var min : UInt = scores[0]  
    var max : UInt = scores[0]  
    var sum : UInt = 0  
    for score in scores {  
        if score > max {  
            max = score  
        }  
        else if score < min {  
            min = score  
        }  
        sum += score  
    }  
    return (Float(sum)/Float(scores.count),min, max)  
}  
let resultats = calculateStatistics(scores: 5, 3, 100, 3, 9)  
print(resultats.moyenne)  
print(resultats.2)  
let (moy,min,max) = calculateStatistics(scores: 5, 3, 100, 3, 9)
```



- ❑ les fonctions peuvent être imbriquées
- ❑ les fonctions peuvent prendre des fonctions en paramètre
- ❑ les fonctions peuvent renvoyer des fonctions

```
func makeIncrementer() -> ((Int) -> Int) {  
    func addOne(number: Int) -> Int {  
        return 1 + number  
    }  
    return addOne  
}  
var increment = makeIncrementer()  
increment(7)
```

- ❑ les fonctions sont en fait un cas particulier des « Closures »



Les Closures

20



- Blocs de code qui peuvent être appelés, nommés, utilisés

- Écrits entre { }, le code séparé des paramètres par `in`

```
numbers.map({  
    (number: Int) -> Int in  
    let result = 3 * number  
    return result  
})
```

- On peut éventuellement omettre le type de retour ou des paramètres

```
let mappedNumbers = numbers.map({ number in 3 * number })
```

- On peut même se référer aux paramètres par leur numéro, et en dernier paramètre on peut omettre les ()

```
let sortedNumbers = numbers.sort { $0 > $1 }
```

Les objets

21



- `class` permet de définir un objet
- les *propriétés* sont définies comme les variables et les constantes
- les *méthodes* sont définies comme les fonctions
- on accède aux membres via la *notation pointée*
- `init` est l'initialisateur des instances : pas un *constructeur* !
- `deinit` pour désinitialiser : attention pas un *destructeur* !
- `self` permet de différencier paramètres d'une méthode d'une propriétés
- les propriétés peuvent définir des *getter* et des *setter*

```

class NamedShape {
    var numberOfSides: Int = 0
    var name: String

    init(name: String) {
        self.name = name
    }

    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}

```

- On peut hériter (notation classique :)
- On peut surcharger les méthodes grâce au mot clef `override`
- `oublier override provoque une erreur`
- possibilité de définir des propriétés calculées



```

class EquilateralTriangle: NamedShape {
    var sideLength: Double = 0.0

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 3
    }

    var perimeter: Double {
        get {
            return 3.0 * sideLength
        }
        set {
            sideLength = newValue / 3.0
        }
    }

    override func simpleDescription() -> String {
        return "An equilateral triangle with sides of length \
(sideLength)."
    }
}

var triangle = EquilateralTriangle(sideLength: 3.1, name: "a triangle")
print(triangle.perimeter)
triangle.perimeter = 9.9

```



- ❑ possibilité de définir des observateurs de propriété

```
class TriangleAndSquare {  
    var triangle: EquilateralTriangle {  
        willSet {  
            square.sideLength = newValue.sideLength  
        }  
    }  
    var square: Square {  
        willSet {  
            triangle.sideLength = newValue.sideLength  
        }  
    }  
    init(size: Double, name: String) {  
        square = Square(sideLength: size, name: name)  
        triangle = EquilateralTriangle(sideLength: size, name: name)  
    }  
}
```

- ❑ attention, une classe modifiant les propriétés d'une superclasse, déclenche l'appel des observateurs



- ❑ Possibilité aussi de définir des variables et des constantes de classe :
 - Elles se définissent avec le mot clef **static**.
 - On peut aussi utiliser le mot clef **class** qui permet alors de la surcharger dans les classes dérivées.

```
class SomeClass {  
    static var storedTypeProperty = "Some value."  
    static var computedTypeProperty: Int {  
        // return an Int value here  
    }  
    class var overrideableComputedTypeProperty: Int {  
        // return an Int value here  
    }  
}
```



Les protocoles : classes abstraites

26



- Tout est virtuel en Swift, comme dans tout langage objet
- Définit un nouveau type abstrait
- N'importe quelle classe peut « hériter d'un protocole »
- Une classe peut implémenter plusieurs protocoles grâce au mot clef protocol< >

```
protocol ExampleProtocol {  
    var simpleDescription: String { get }  
    mutating func adjust()  
}  
  
class SimpleClass: ExampleProtocol {  
    var simpleDescription: String = "A very simple class."  
    var anotherProperty: Int = 69105  
    func adjust() {  
        simpleDescription += " Now 100% adjusted."  
    }  
}
```

Extensions

27



- Il est possible d'ajouter des fonctionnalités à un type via une extension

```
extension Int: ExampleProtocol {  
    var simpleDescription: String {  
        return "The number \(self)"  
    }  
    func adjust() {  
        self += 42  
    }  
}  
print(7.simpleDescription)
```

Généricité

28



- On peut définir un type ou une fonction générique :

```
func repeatItem<Item>(item: Item, number0fTimes: Int) -> [Item] {  
    var result = [Item]()  
    for _ in 0..<number0fTimes {  
        result.append(item)  
    }  
    return result  
}
```

- On peut mettre des conditions sur le type générique :

```
func anyCommonElements <T: SequenceType, U: SequenceType where  
T.Generator.Element: Equatable, T.Generator.Element == U.Generator.Element> (lhs:  
T, _ rhs: U) -> Bool {  
    for lhsItem in lhs {  
        for rhsItem in rhs {  
            if lhsItem == rhsItem {  
                return true  
            }  
        }  
    }  
    return false  
}
```

Xcode & IOS développement (2)

XCode : environnement de développement



Choose a template for your new project:

iOS				
Application	Master-Detail Application	Page-Based Application	Single View Application	Tabbed Application
Framework & Library				
Other				
OS X				
Application	Game			
Framework & Library				
System Plug-in				
Other				

Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

[Cancel](#) [Previous](#) [Next](#)

Créer un nouveau projet en sélection une

Single View Application

31

Choose options for your new project:

Product Name: TravellerTracker

Organization Name: Fiorio Christophe

Organization Identifier: Polytech.Univ-Montp2.fr

Bundle Identifier: Polytech.Univ-Montp2.fr.TravellerTracker

Language: Swift

Devices: Universal

Use Core Data

Include Unit Tests

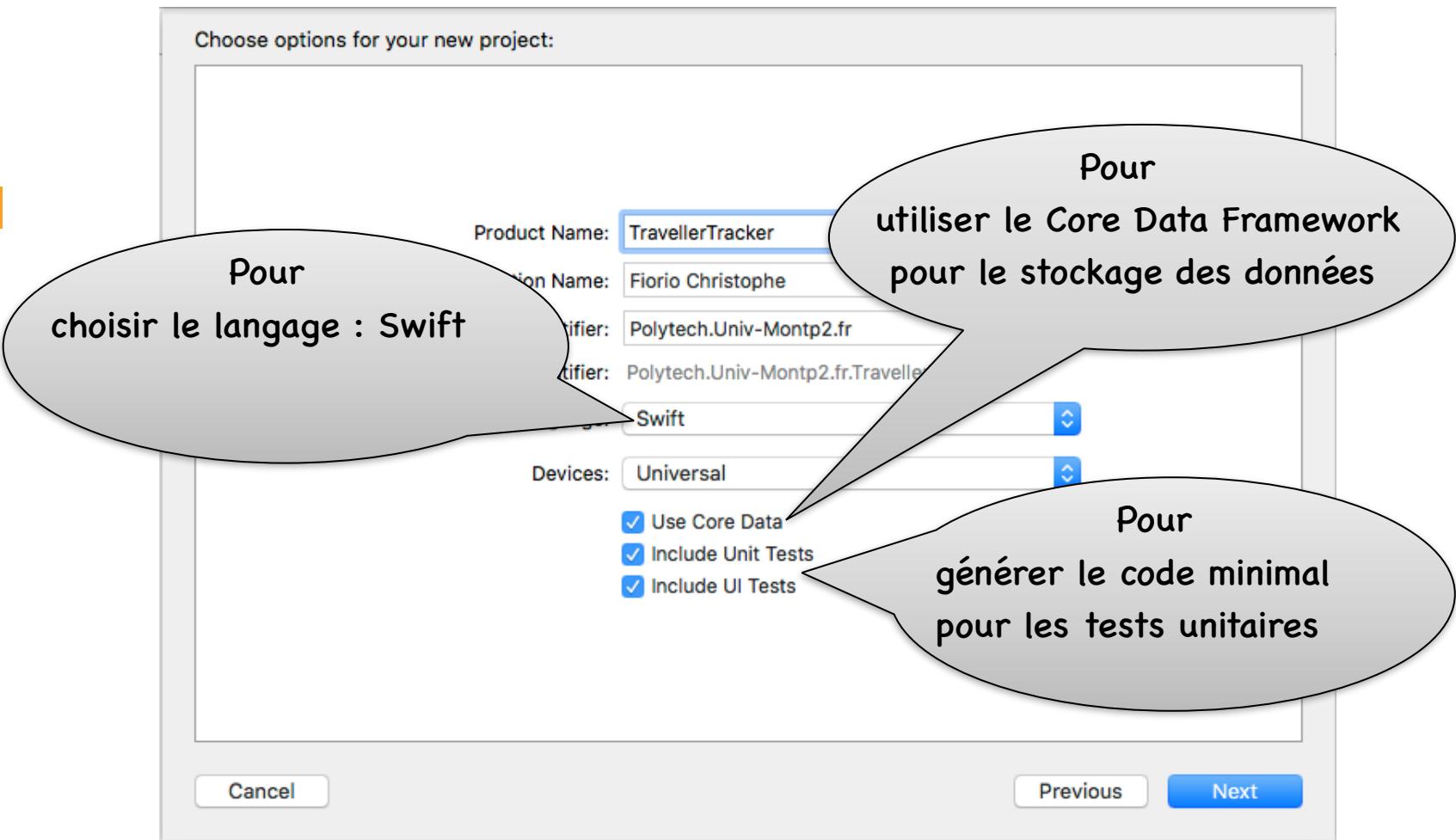
Include UI Tests

Cancel

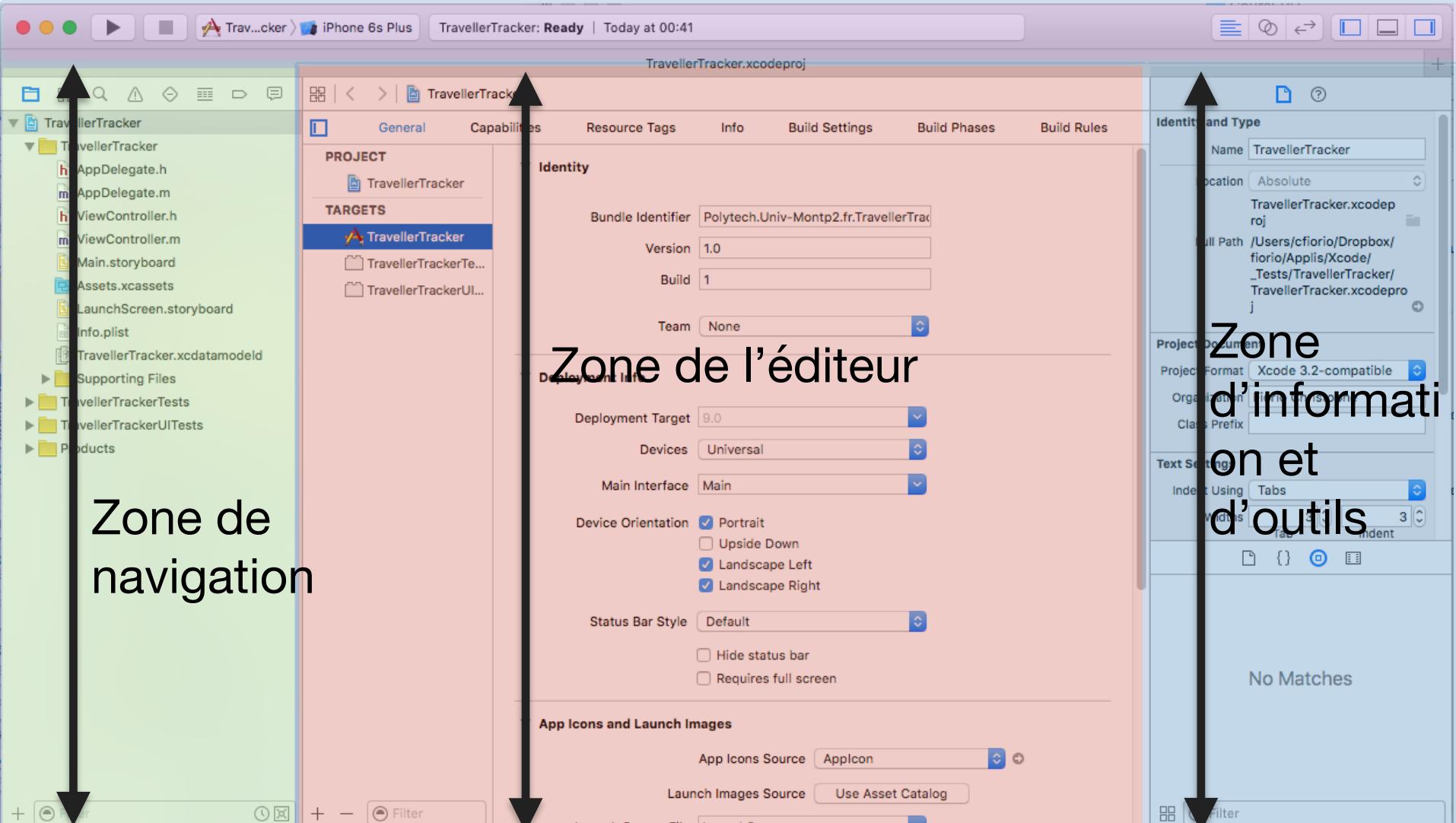
Previous

Next

Nom du projet : TravellerTracker



Nom du projet : TravellerTracker



Les différentes zones d'édition d'un projet XCode

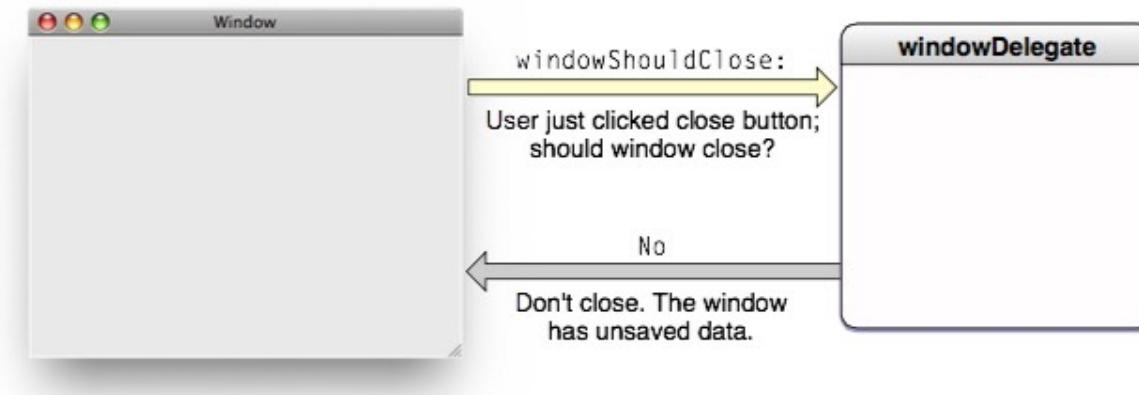
Fichiers	Description
Info.plist	<i>Fichier des propriétés de l'application : nom de l'appli, icône, qu'elle fenêtre ouvrir au lancement,</i>
AppDelegate	<i>Classe déléguée de l'application (Design pattern "Délégation de contrôle") permettant d'enrichir le comportement de la classe UIApplication</i>
ViewController	<i>Classe réceptrice des évènements de l'interface — la vue — (Design pattern Model-View-Controller)</i>

Design Pattern Délégation de contrôle

34



- Le design pattern délégation de contrôle permet à une classe de s'appuyer sur une autre pour enrichir son propre comportement.

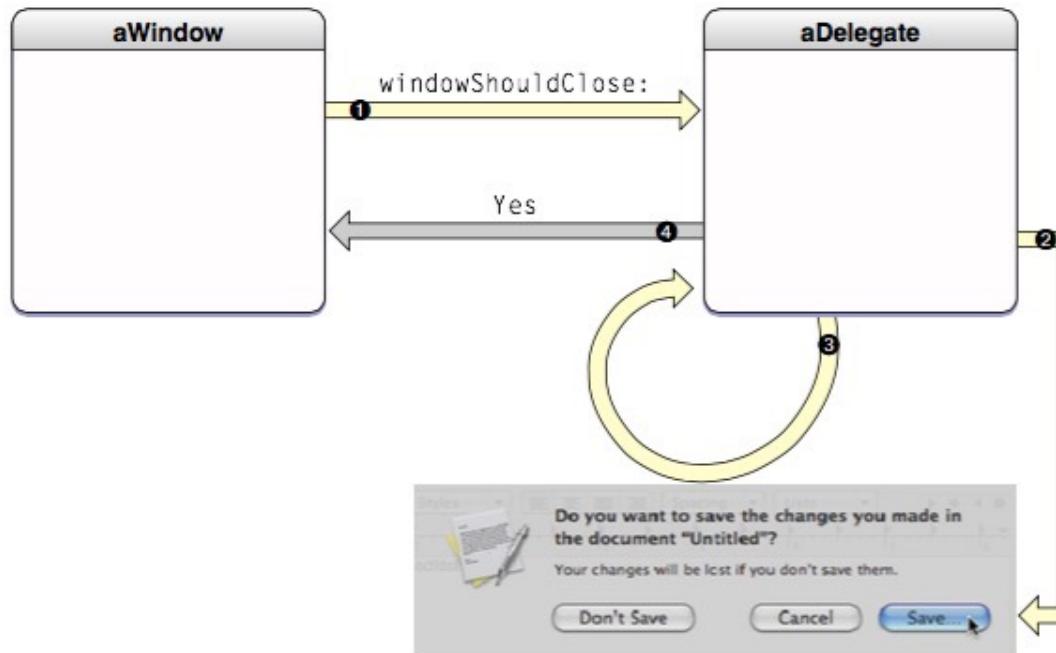


Design Pattern Délégation de contrôle

34



- Le design pattern délégation de contrôle permet à une classe de s'appuyer sur une autre pour enrichir son propre comportement.



Définis par des protocoles, les classes déléguées permettent à l'utilisateur d'intervenir à certains moments clefs du déroulement 35 des événements

```
class MyDelegate: NSObject, NSWindowDelegate {  
    func window(NSWindow, willUseFullScreenContentSize  
proposedSize: NSSize) -> NSSize {  
        return proposedSize  
    }  
}  
var myDelegate: NSWindowDelegate? = MyDelegate()  
if let fullScreenSize = myDelegate?.window?(myWindow,  
willUseFullScreenContentSize: mySize) {  
    println(NSStrongFromSize(fullScreenSize))  
}
```

Extrait de: Apple Inc. « Using Swift with Cocoa and Objective-C. » iBooks. <https://itun.es/fr/1u3-0.1>

UIApplicationDelegate Protocol

36



❑ applicationDidFinishLaunching:

L'application est prête à démarrer : storyboard et boucle événementielle prêts ; c'est le lieu où placer du code à exécuter au démarrage. Attention à ne pas surcharger !

❑ applicationWillTerminate:

L'application va se terminer : quand l'utilisateur quitte l'application (Home, appel, ...)

❑ applicationDidReceiveMemoryWarning:

Problème mémoire : il faut libérer des objets sous peine de voir le système faire quitter l'application

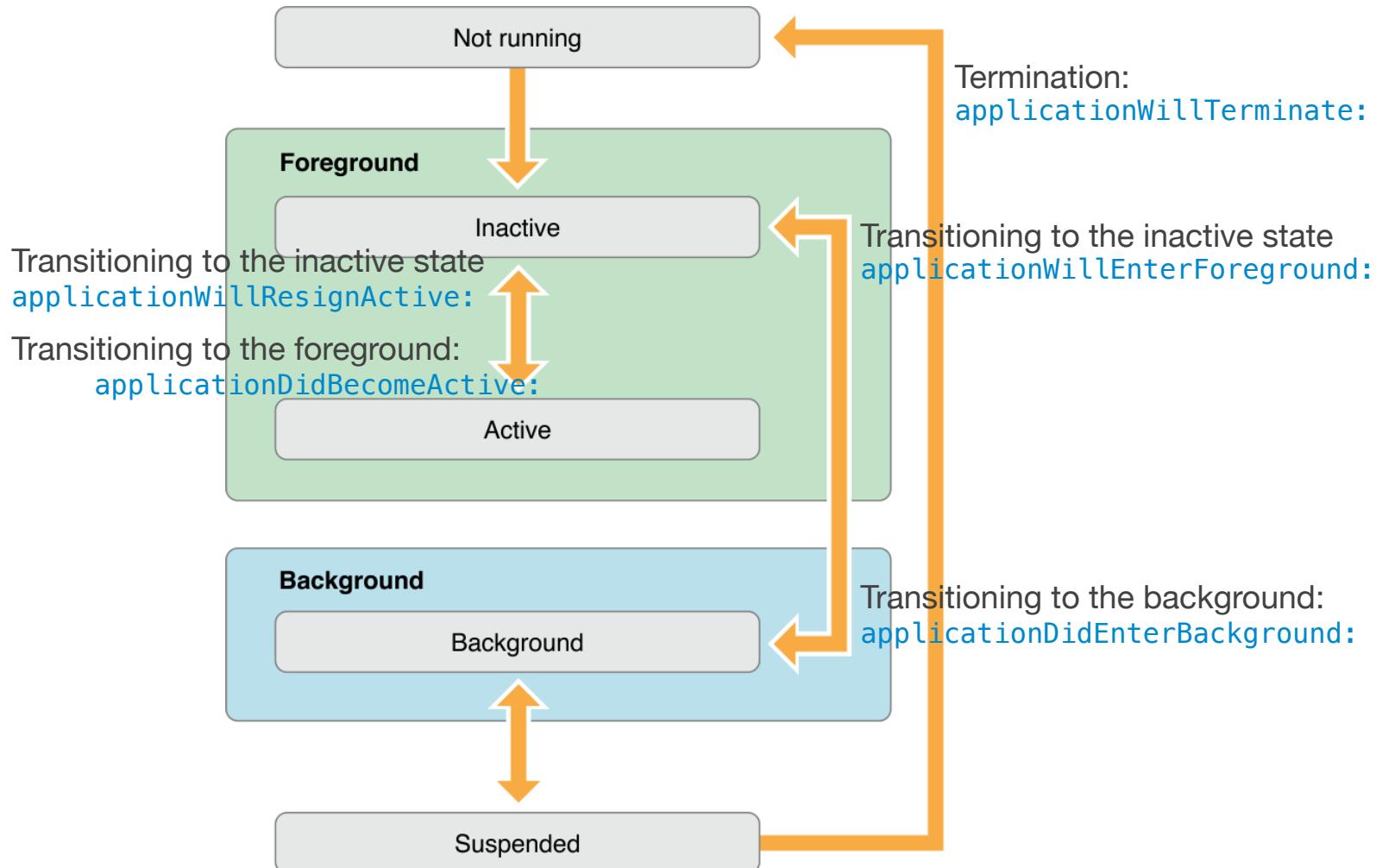
❑ applicationWillResignActive:

L'application passe à l'arrière plan : il faut sauver l'état de l'appli

Launch time:

`application:willFinishLaunchingWithOptions:`
`application:didFinishLaunchingWithOptions:`

37



Storyboard

38



- Le storyboard montre les différents écrans de l'application et les transitions entre ces écrans.
- Il faut dire à l'application de l'utiliser comme interface principale (fait normalement par défaut en ayant choisi Single View Application)
- Pour cela sélectionnez le projet dans le nagiaviteur et ...

Trav...cker > iPhone 6s Plus TravellerTracker: Ready | Today at 00:41

TravellerTracker.xcodeproj

TravellerTracker

TravellerTracker

AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m
Main.storyboard
Assets.xcassets
LaunchScreen.storyboard
Info.plist
TravellerTracker.xcdatamodeld
Supporting Files
TravellerTrackerTests
TravellerTrackerUITests
Products

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

PROJECT
TravellerTracker

TARGETS
TravellerTracker
TravellerTrackerTe...
TravellerTrackerUi...

Identity

Bundle Identifier Polytech.Univ-Montp2.fr.TravellerTrad
Version 1.0
Build 1
Team None

Deployment Info

Deployment Target 9.0
Devices Universal
Main Interface Main
Device Orientation Portrait
 Upside Down
 Landscape Left
 Landscape Right
Status Bar Style Default
 Hide status bar
 Requires full screen

App Icons and Launch Images

App Icons Source AppIcon
Launch Images Source Use Asset Catalog
Launch Screen File LaunchScreen

Identity and Type

Name TravellerTracker
Location Absolute
TravellerTracker.xcodeproj
Full Path /Users/cfiorio/Dropbox/fiorio/Applications/Xcode/_Tests/TravellerTracker/TravellerTracker.xcodeproj

Project Document

Project Format Xcode 3.2-compatible
Organization Fiorio Christophe
Class Prefix

Text Settings

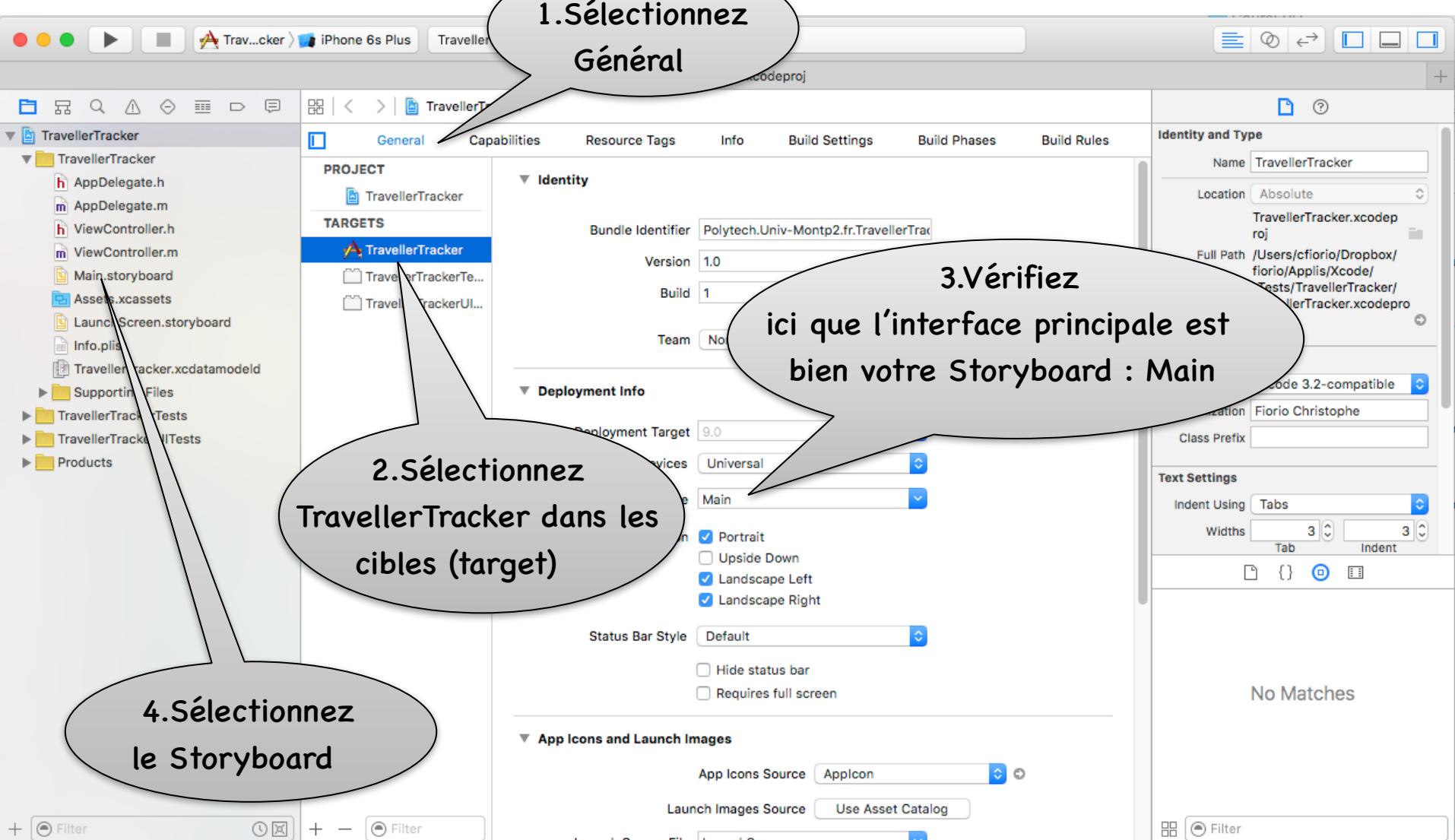
Indent Using Tabs
Widths Tab Indent
No Matches

1.Sélectionnez
Général

2.Sélectionnez
TravellerTracker dans les
cibles (target)

4.Sélectionnez
le Storyboard

3.Vérifiez
ici que l'interface principale est
bien votre Storyboard : Main



Ajouter une scène

40

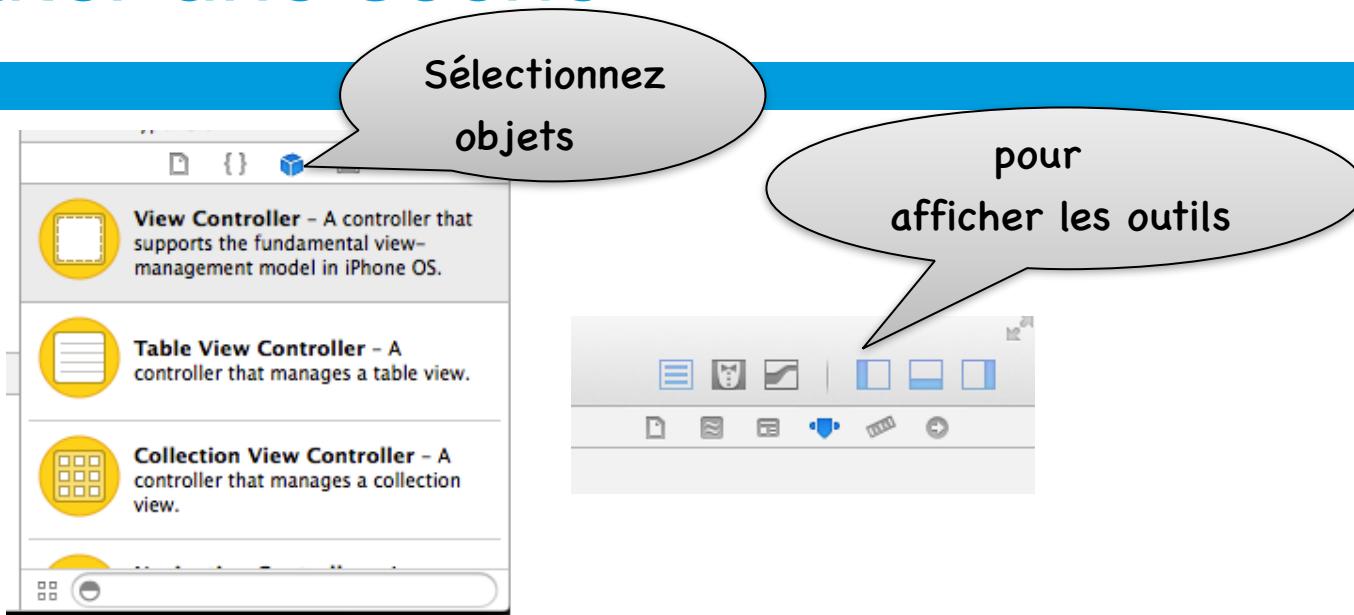


- On va ajouter un contrôleur de vue : cet objet gère la vue correspondante ainsi que les vues dépendantes — il représente à la fois la classe contrôleur et la classe graphique vue

Dans le navigateur, sélectionnez Main.storyboard

Ajouter une scène

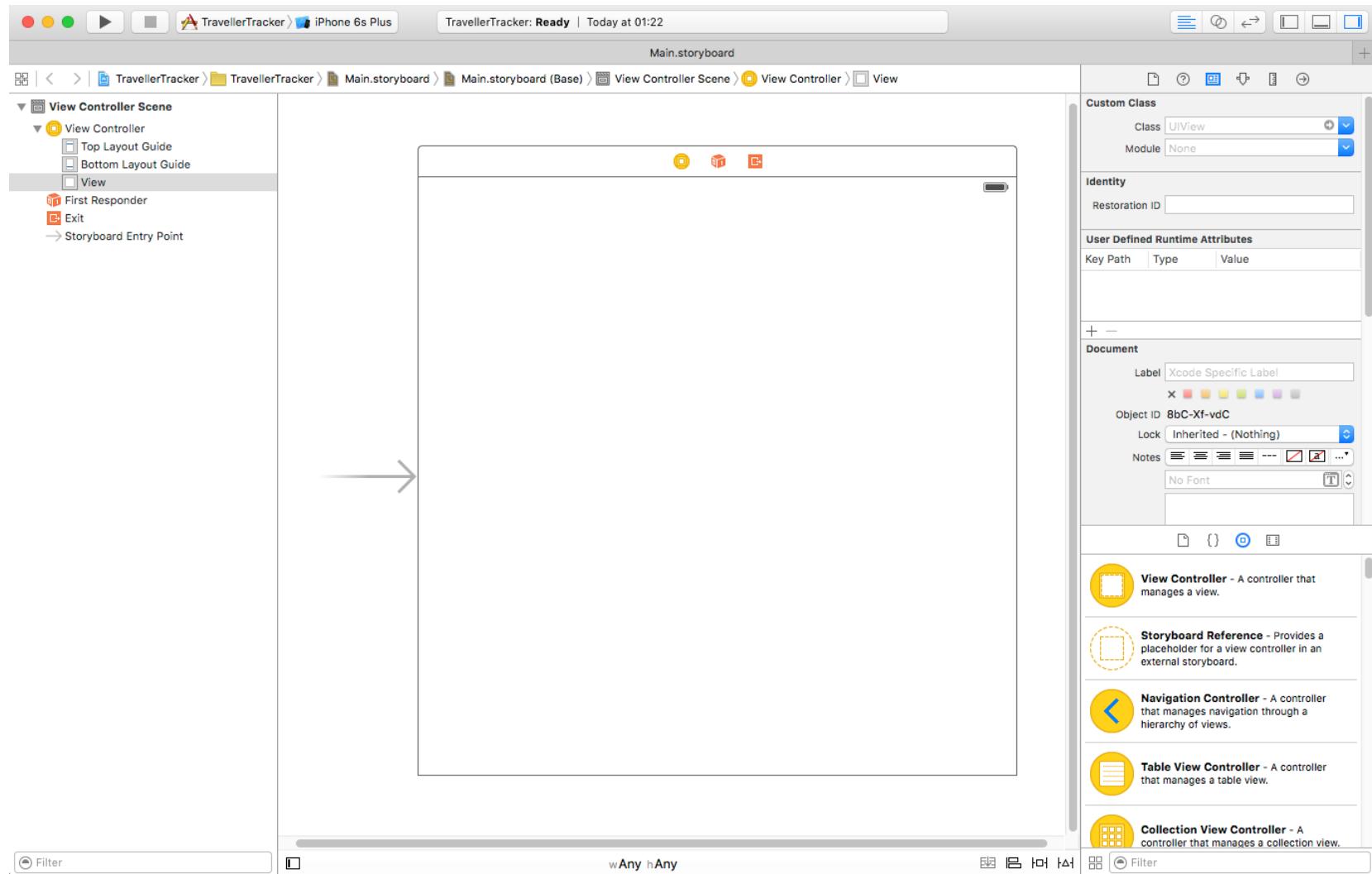
40



- Ouvrir la librairie d'objets : une liste montre les différents objets disponibles

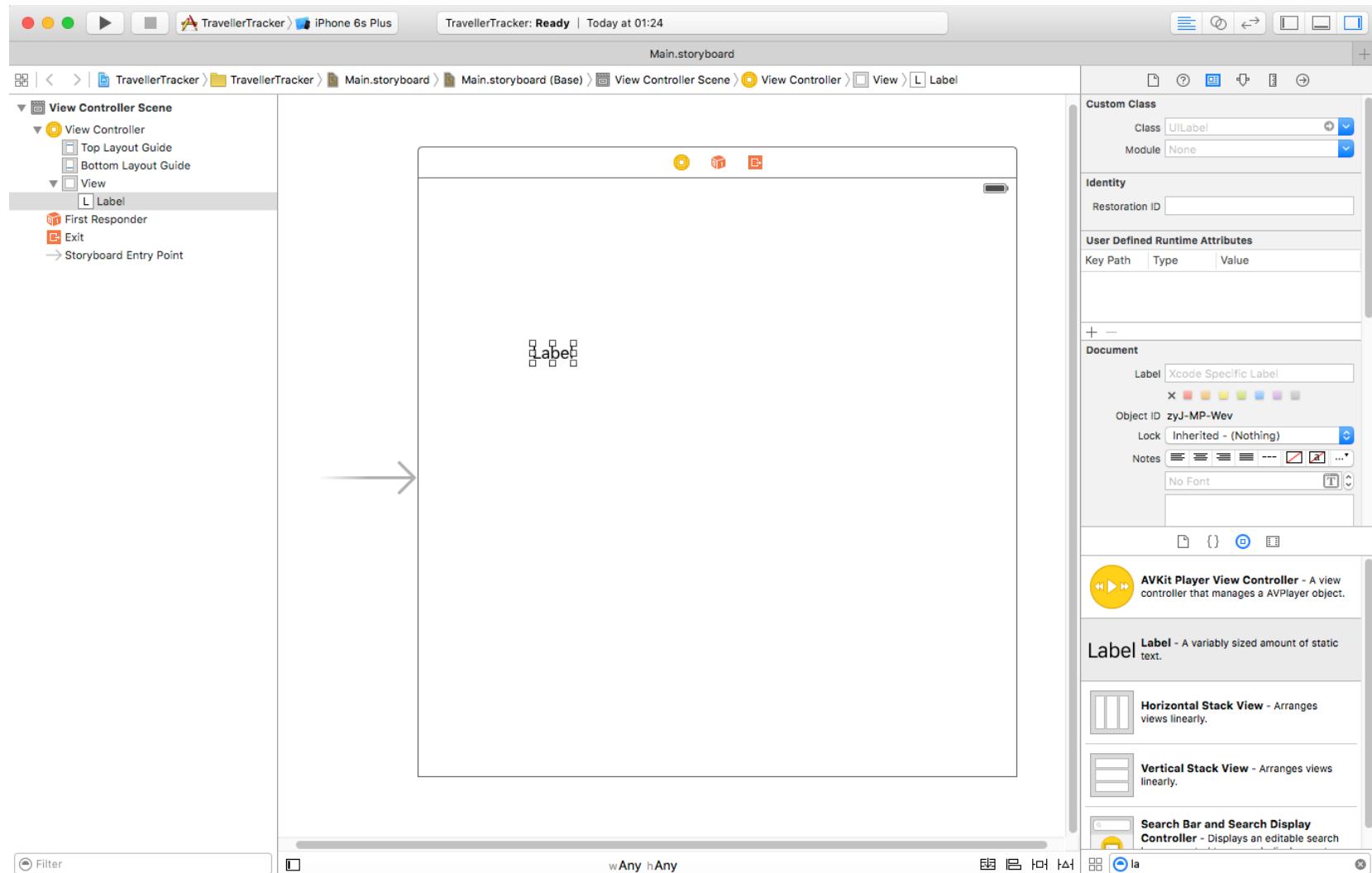
Si vous voulez rajouter une nouvelle vue, il suffirait de Glissez-déposer un View Controller sur l'écran

41

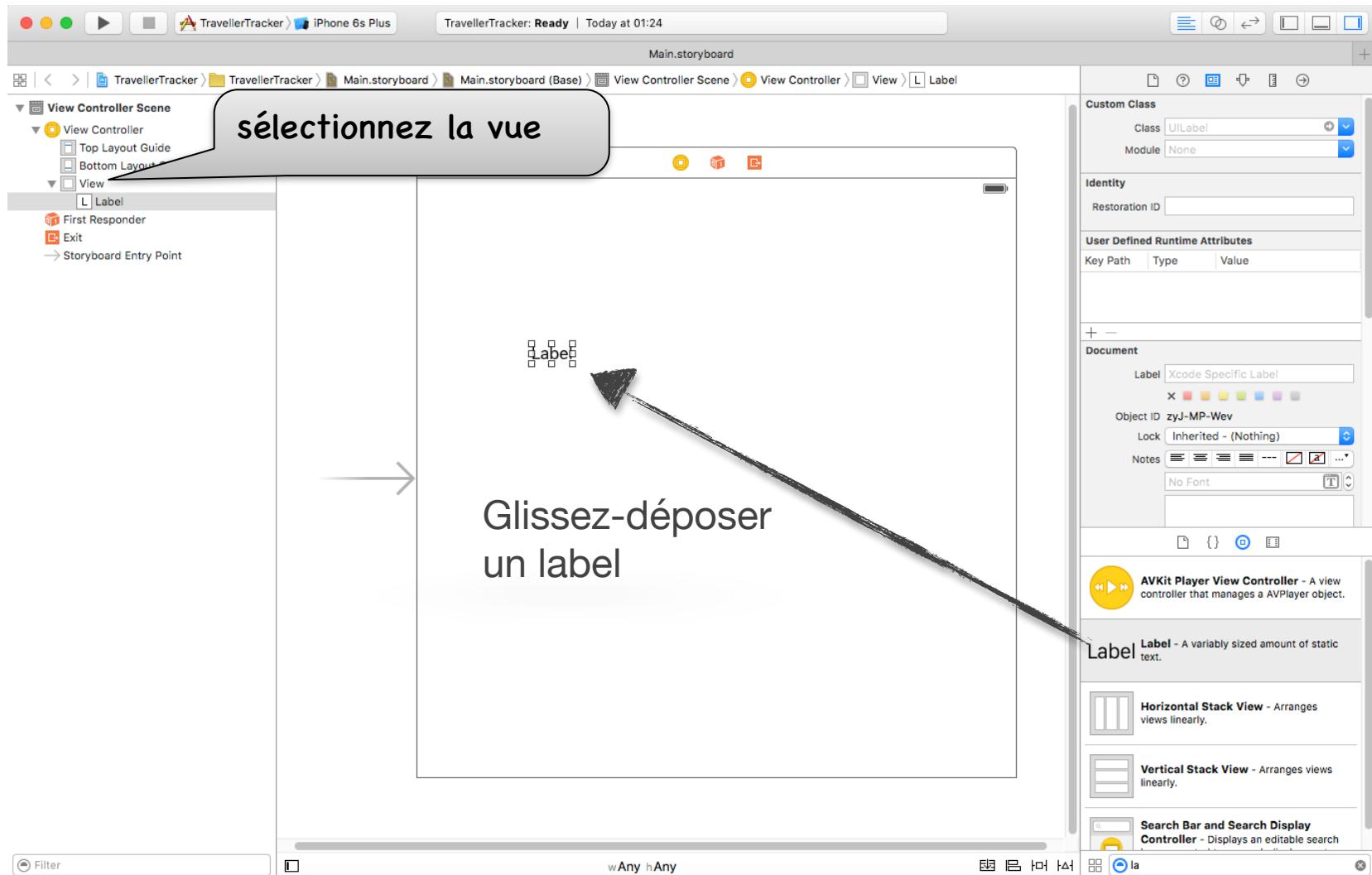


Le storyboard représente les écrans de votre application.
On y trouve des « scenes » et des « segues »

Ici pas de segues car une application Single View possède un seul écran



Au démarrage, le storyboard est chargé et le contrôleur initial est instancié.
Le contrôleur initial gère la première scène.



Au démarrage, le storyboard est chargé et le contrôleur initial est instancié.
Le contrôleur initial gère la première scène.

Schéma d'exécution

43



1. L'exécution démarre et crée un objet `UIApplication`,
2. le fichier `Info.plist` est lu pour obtenir le nom du fichier Storyboard à charger pour instancier la fenêtre ainsi que le délégué de l'application (la classe `AppDelegate` dans notre cas).
3. Une boucle de gestion des événements est démarrée par l'objet `UIApplication`.
4. La méthode `applicationDidFinishLaunching` du délégué est appelée.
5. La boucle de gestion des événements traite un par un les événements reçus par l'application.
6. Quand l'utilisateur ferme l'application (en cliquant sur le bouton Home par exemple), la boucle s'arrête, l'application rend la main et se termine.

Changer la couleur de fond

The screenshot shows the Xcode interface with the storyboard editor open. The storyboard file is named "Main.storyboard". The main view of the storyboard contains a single label with the text "Label". The background color of the view is set to a medium green. On the left side, the "Attributes Inspector" is visible, showing settings for the selected "View" object. The "Background" color is set to yellow, and the "Tint" color is blue. The "Mode" is set to "Scale To Fill". The "Drawing" section has "Opaque" checked. At the bottom of the screen, there are several interface builder tools and a search bar.

Main.storyboard

TravellerTracker: Ready | Today at 01:37

View

Controller Scene

View Controller

Top Layout Guide

Bottom Layout Guide

View

Label

First Responder

Exit

Storyboard Entry Point

Colors

Banana

Opacity: 100 %

AVKit Player View Controller - A view controller that manages a AVPlayer object.

Label - A variably sized amount of static text.

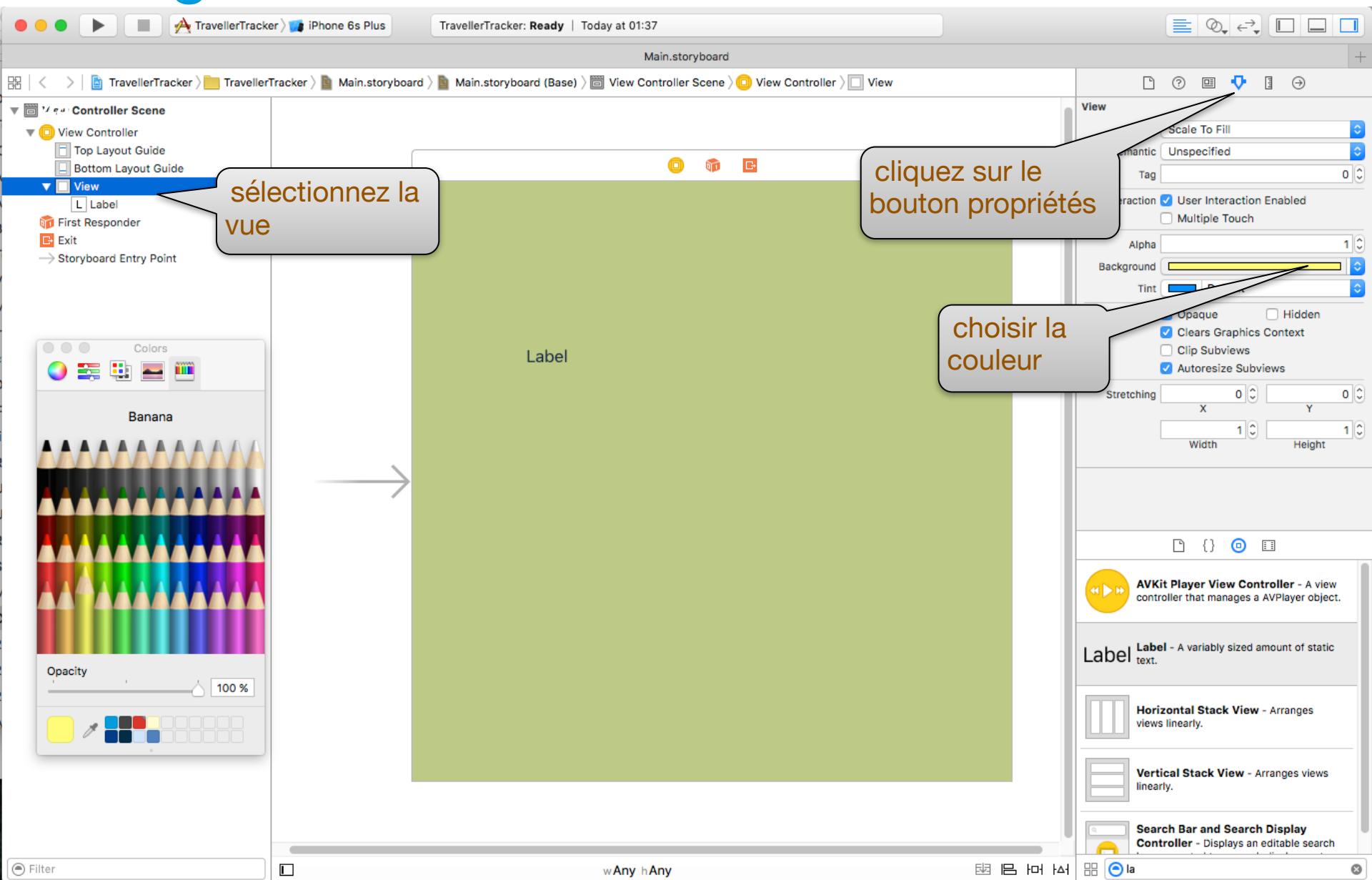
Horizontal Stack View - Arranges views linearly.

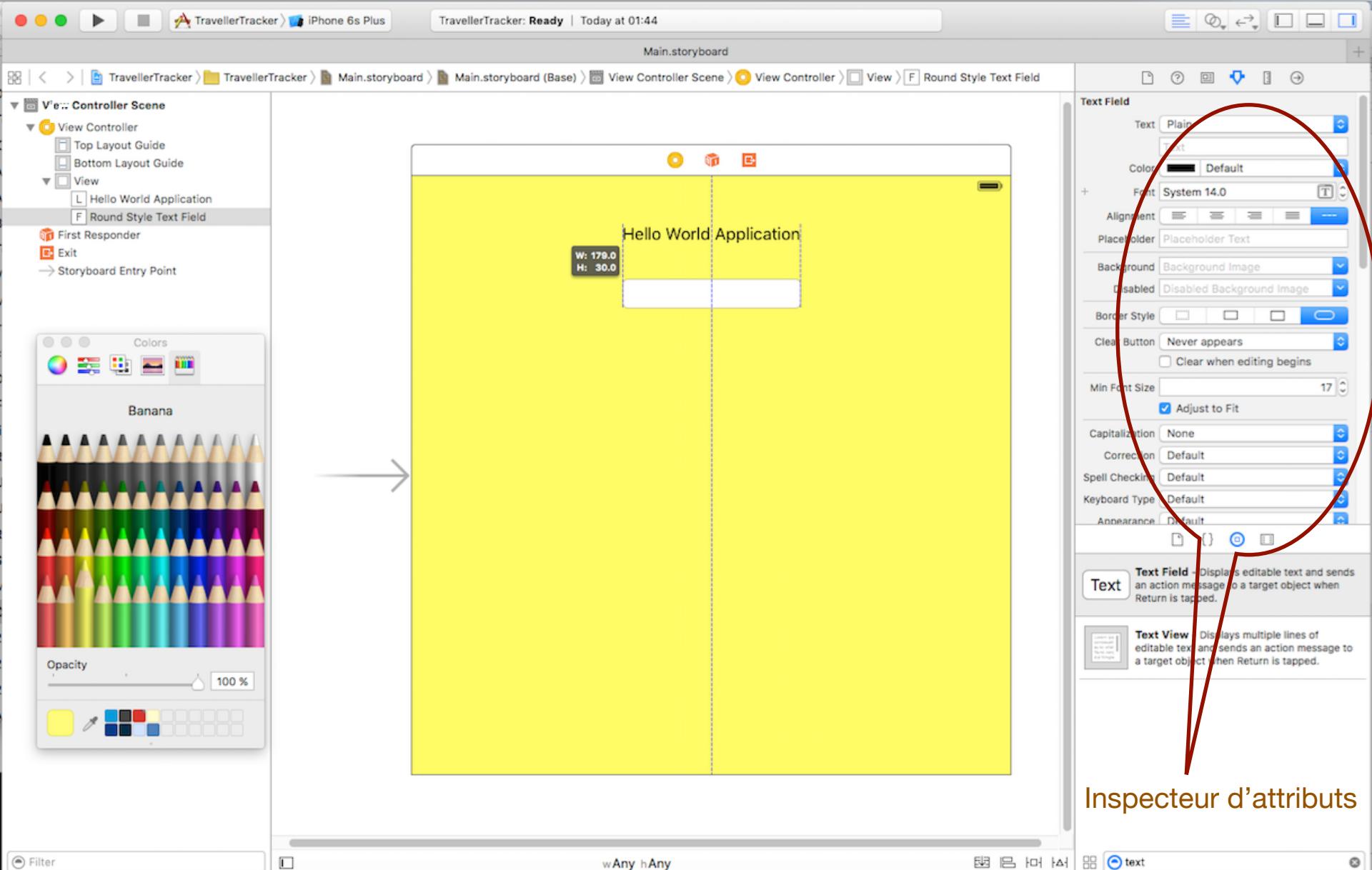
Vertical Stack View - Arranges views linearly.

Search Bar and Search Display Controller - Displays an editable search

wAny hAny

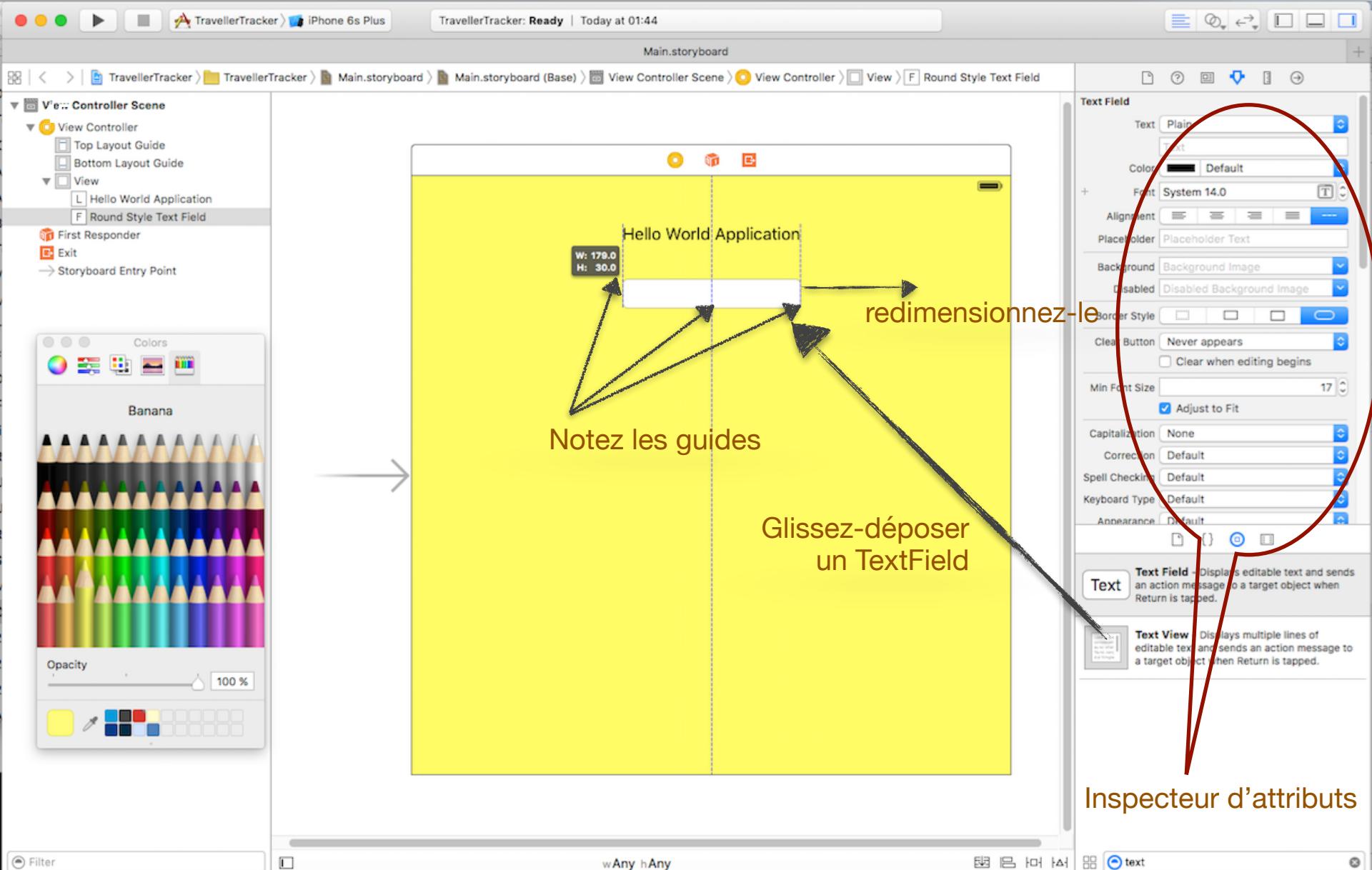
Changer la couleur de fond





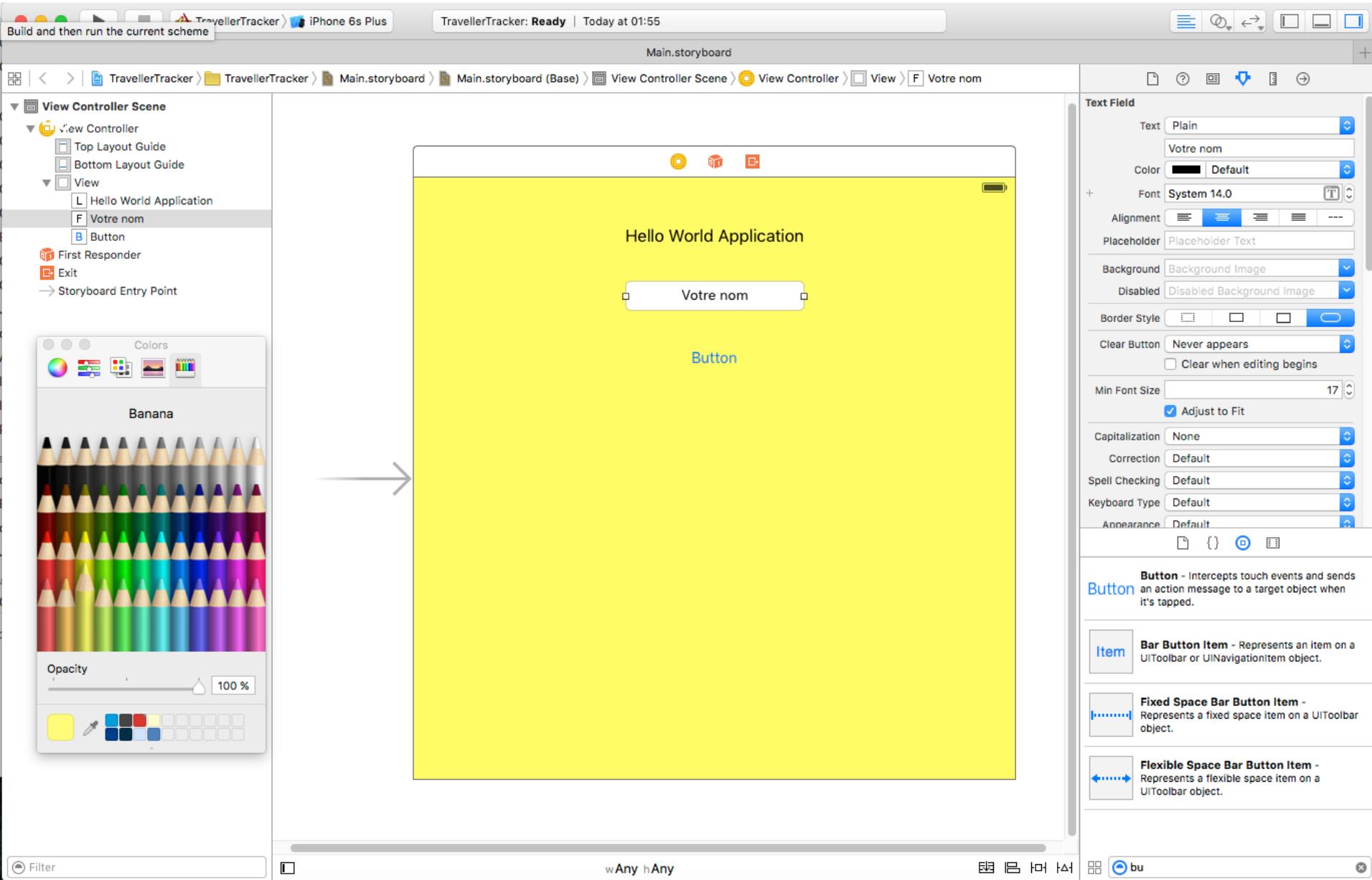
Pour ajouter des éléments :

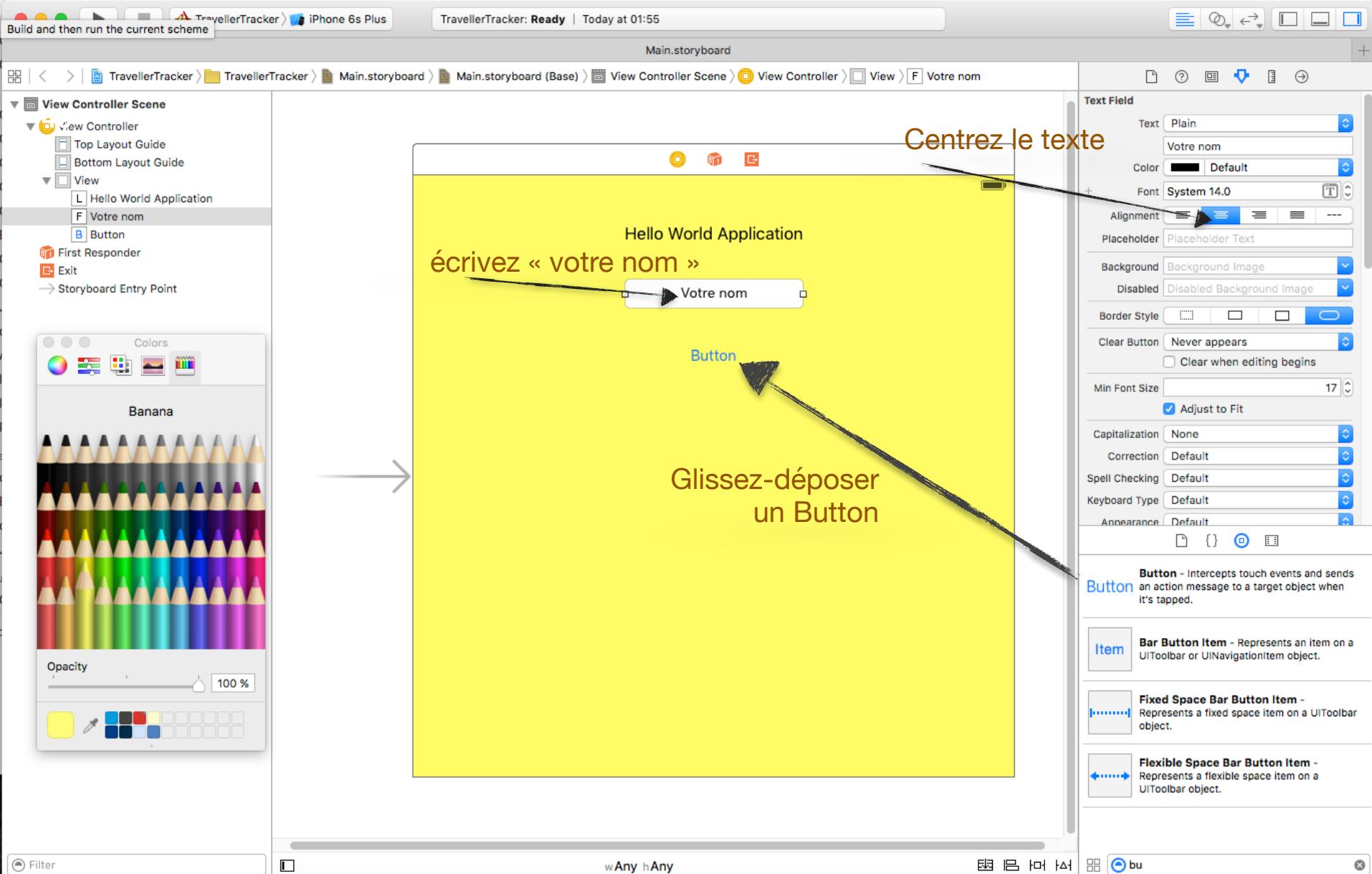
- cliquer sur la librairie d'objets, choisir « Controls »
- Glisser-déposer un « Text Field »



Pour ajouter des éléments :

- cliquer sur la librairie d'objets, choisir « Controls »
- Glisser-déposer un « Text Field »





Testez votre application

47



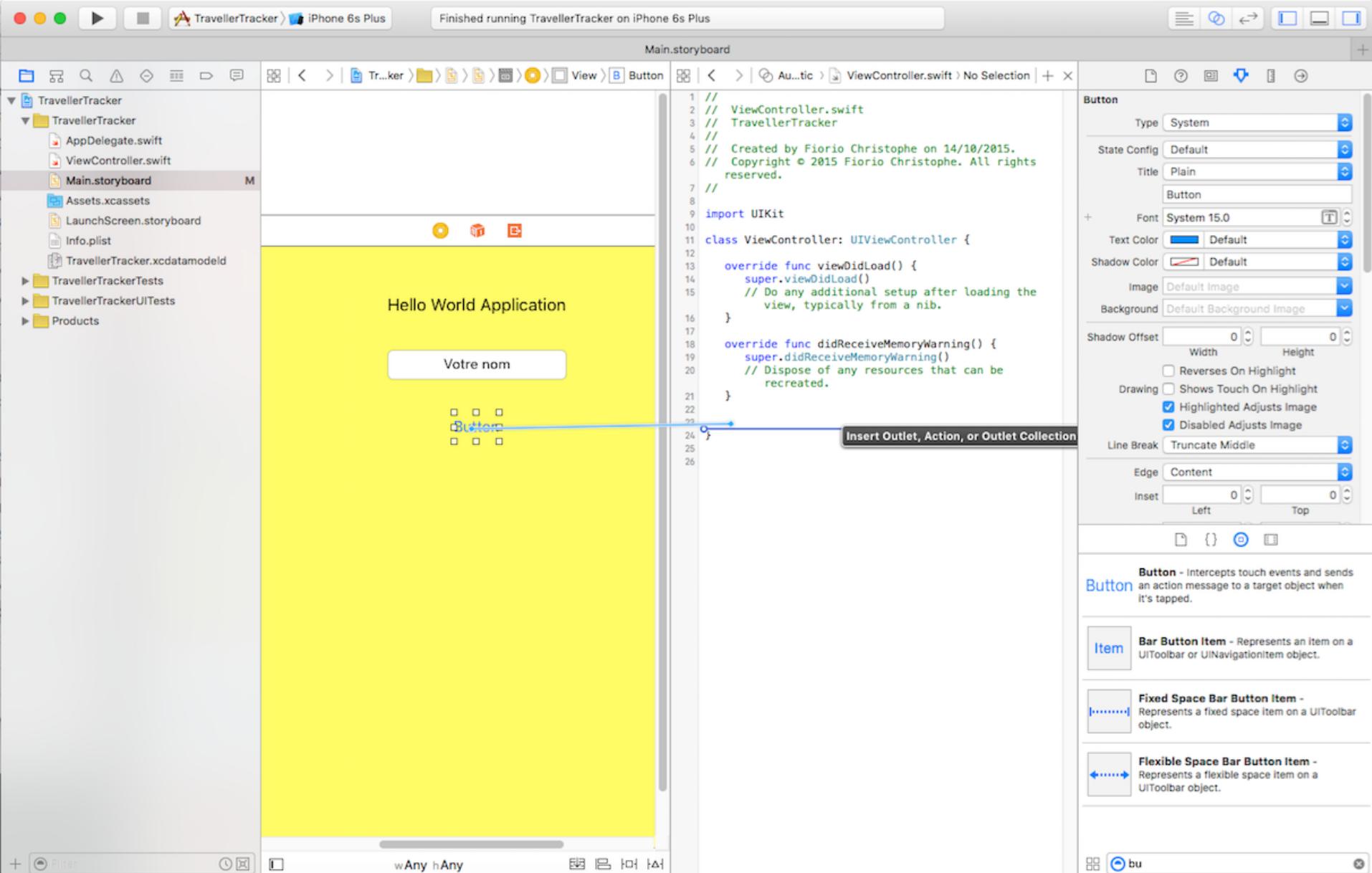
Notre interface apparaît...

Créer une action pour le bouton

48



- Quand l'utilisateur active un élément de l'interface, l'élément peut envoyer un message d'action vers un objet qui sait comment traiter ce message
- Ce processus est la base du mécanisme « target-action » de Cocoa
- Avec Xcode, on peut ajouter des actions à traiter à un élément de l'interface, simplement en control-glissant l'action voulue de l'élément vers le fichier code-source.



TravellerTracker > iPhone 6s Plus

Finished running TravellerTracker on iPhone 6s Plus

Main.storyboard

TravellerTracker

AppDelegate.swift

ViewController.swift

Main.storyboard

Assets.xcassets

LaunchScreen.storyboard

Info.plist

TravellerTracker.xcdatamodeld

TravellerTrackerTests

TravellerTrackerUITests

Products

View Controller

Button

1 // ViewController.swift
2 // TravellerTracker
3 // Created by Fiorio Christophe on 14/10/2015.
4 // Copyright © 2015 Fiorio Christophe. All rights reserved.
5 //
6 //
7 //
8 import UIKit
9
10 class ViewController: UIViewController {
11
12 override func viewDidLoad() {
13 super.viewDidLoad()
14 // Do any additional setup after loading the view, typically from a nib.
15 }
16
17 override func didReceiveMemoryWarning() {
18 super.didReceiveMemoryWarning()
19 // Dispose of any resources that can be recreated.
20 }
21
22
23
24 }
25
26

Insert Outlet, Action, or Outlet Collection

Button

Type System

State Config Default

Title Plain

Font System 15.0

Text Color Default

Shadow Color Default

Image Default Image

Background Default Background Image

Shadow Offset Width 0 Height 0

Reverses On Highlight

Drawing

Shows Touch On Highlight

Highlighted Adjusts Image

Disabled Adjusts Image

Line Break Truncate Middle

Edge Content

Inset Left 0 Top 0

Item

Bar Button Item - Represents an item on a UIBarButtonItem or UINavigationController object.

Fixed Space Bar Button Item - Represents a fixed space item on a UIBarButtonItem object.

Flexible Space Bar Button Item - Represents a flexible space item on a UIBarButtonItem object.

Votre nom

Hello World Application

Buttons

- Control-glissez le bouton vers l'interface du contrôleur

TravellerTracker > iPhone 6s Plus

Finished running TravellerTracker on iPhone 6s Plus

Main.storyboard

Triggered Segues

Outlet Collections

Sent Events

Referencing Outlets

Referencing Outlet Collections

Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Item - Represents an item on a UIBarButtonItem or UINavigationItem object.

Fixed Space Bar Button Item - Represents a fixed space item on a UIBarButtonItem object.

Flexible Space Bar Button Item - Represents a flexible space item on a UIBarButtonItem object.

bu

TravellerTracker

AppDelegate.swift

ViewController.swift

Main.storyboard

Assets.xcassets

LaunchScreen.storyboard

Info.plist

TravellerTracker.xcdatamodeld

TravellerTrackerTests

TravellerTrackerUITests

Products

View Controller

View

Button

Triggered Segues

action

Outlet Collections

gestureRecognizers

Sent Events

Did End On Exit

Editing Changed

Editing Did Begin

Editing Did End

Primary Action Triggered

Touch Cancel

Touch Down

Touch Down Repeat

Touch Drag Enter

Touch Drag Exit

Touch Drag Inside

Touch Drag Outside

Touch Up Inside

Touch Up Outside

Value Changed

Referencing Outlets

New Referencing Outlet

Referencing Outlet Collections

New Referencing Outlet Collection

Connection Action

Object View Controller

Name changeHello

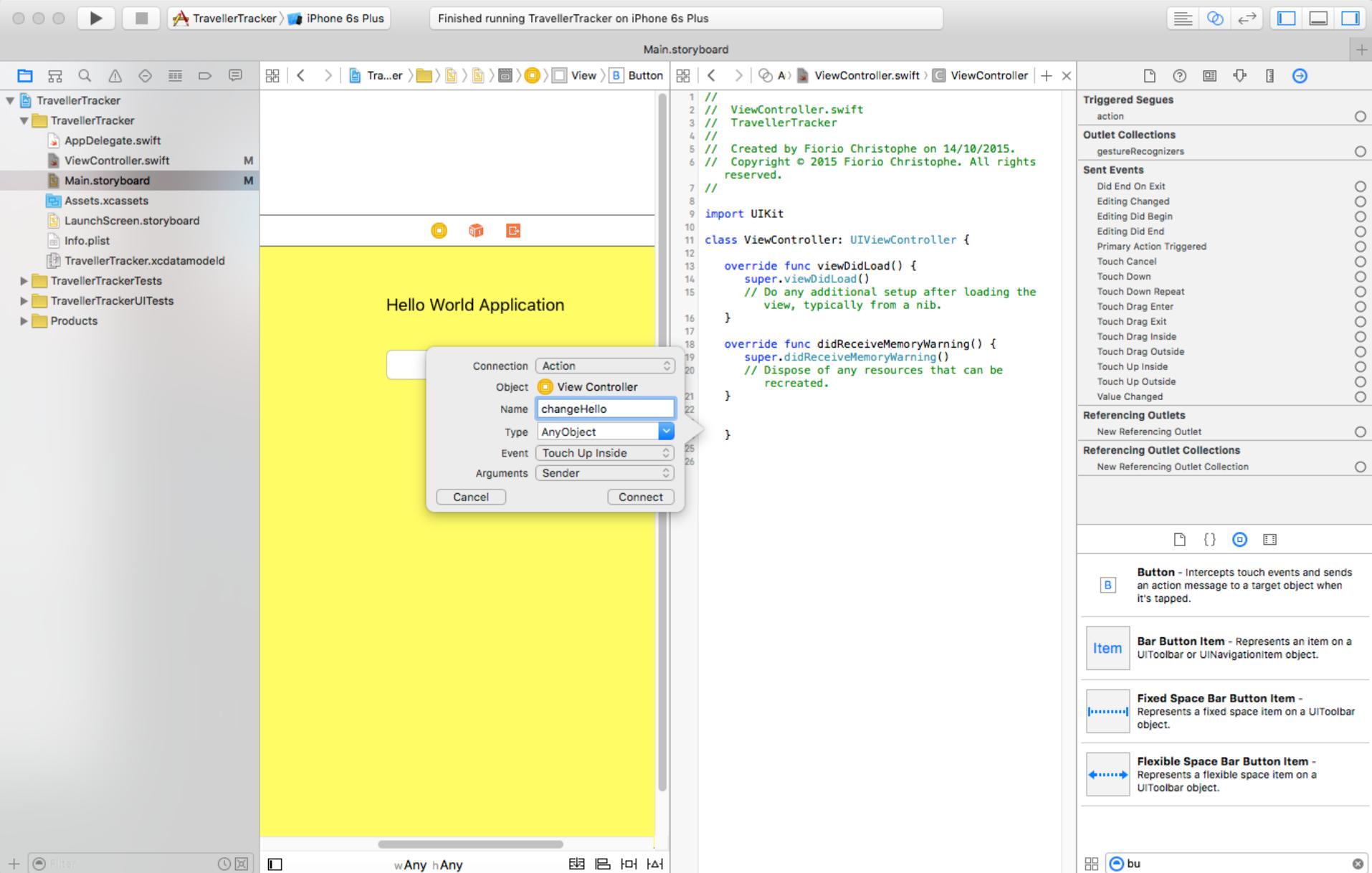
Type AnyObject

Event Touch Up Inside

Arguments Sender

Cancel Connect

```
1 // ViewController.swift
2 // TravellerTracker
3 //
4 //
5 // Created by Fiorio Christophe on 14/10/2015.
6 // Copyright © 2015 Fiorio Christophe. All rights reserved.
7 //
8
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         // Do any additional setup after loading the view, typically from a nib.
16     }
17
18     override func didReceiveMemoryWarning() {
19         super.didReceiveMemoryWarning()
20         // Dispose of any resources that can be recreated.
21     }
22 }
```



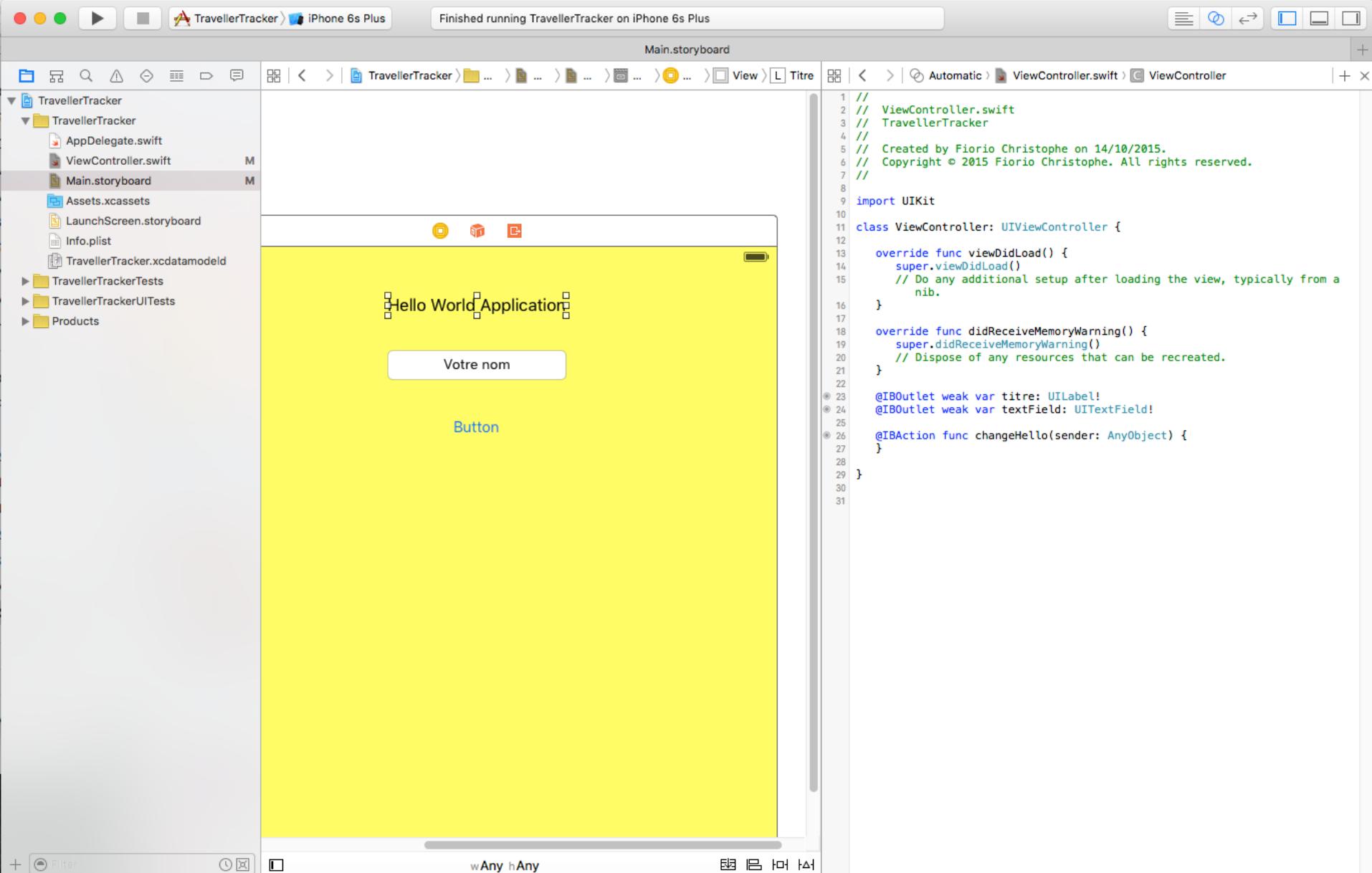
- 1) Control-glissez le bouton vers l'interface du contrôleur
- 2) Dans la boîte de dialogue, choisissez « action » et pour le nom « changeHello » et cliquez sur connect

Créer un « outlet » pour le textfield et le label

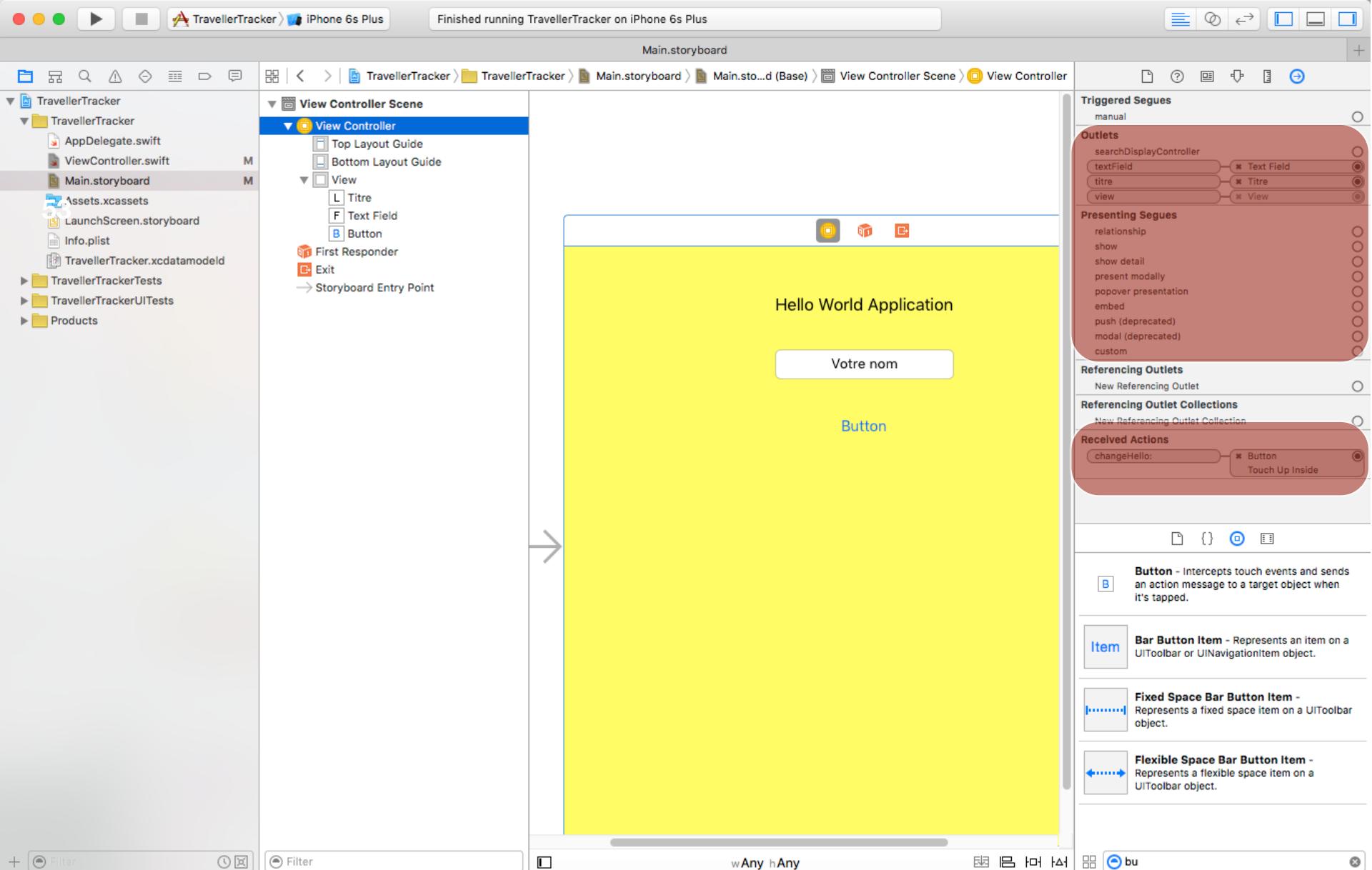
51



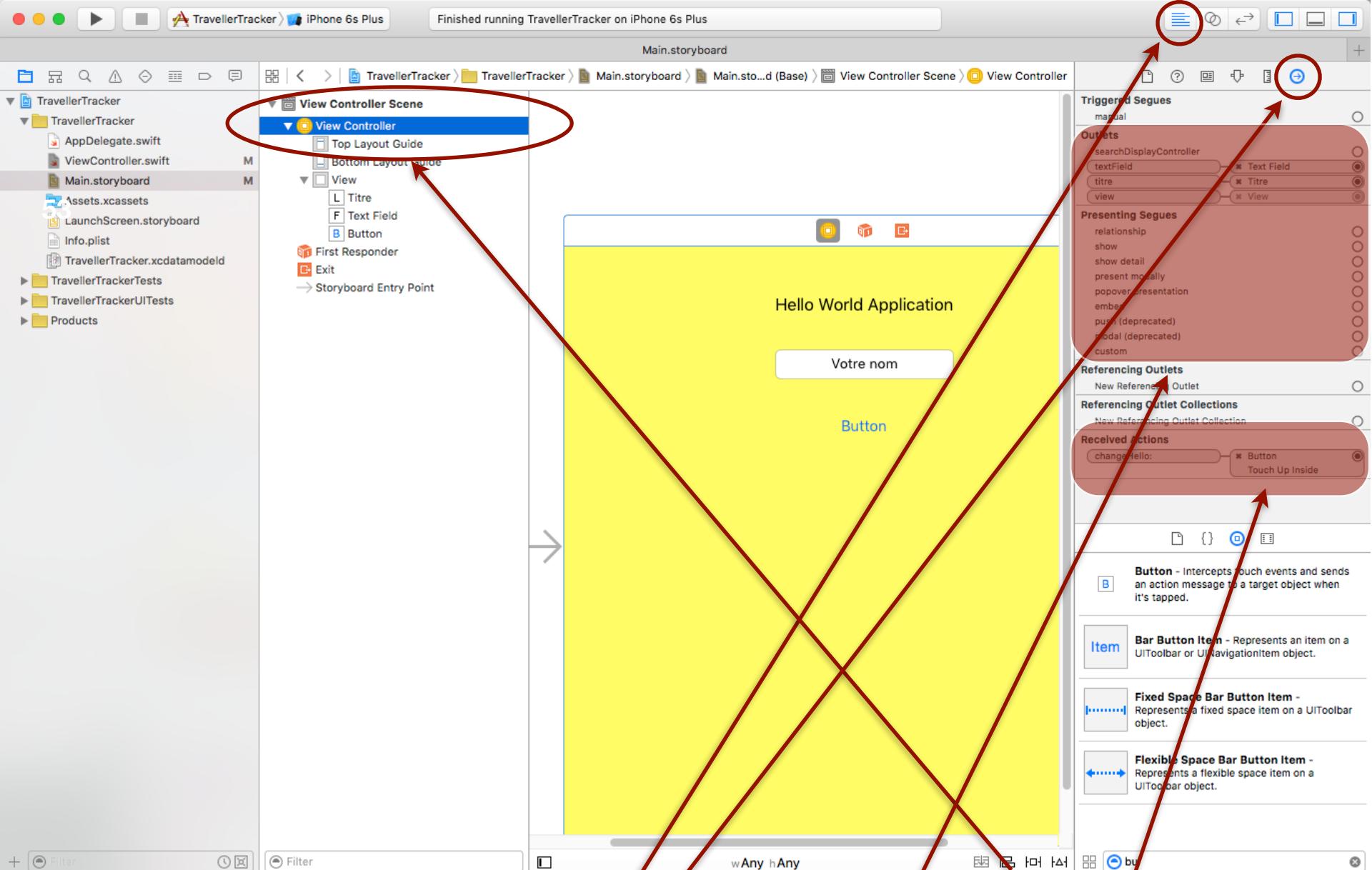
- Un « outlet » décrit une connection entre deux objets
- Lorsqu'on veut qu'un objet (généralement le contrôleur) communique avec un autre objet, il faut déclarer l'objet receveur comme un « outlet »
- Ici, on veut récupérer le texte entré par l'utilisateur dans le textfield et l'afficher dans le label.
- On va donc connecter le contrôleur avec ces deux objets.



- 1) Control-glissez le Textfield vers l'interface du Contrôleur
- 2) Control-glissez le label vers l'interface du Contrôleur
- 3) Appelez-les outlets resp. textField et titre



1) Cliquez sur l'éditeur standard, sur le Controller,
2) Cliquez sur l'utilitaire de connection pour les visualiser
3) on peut alors voir le détail des connexions et actions



1)
2)
3)

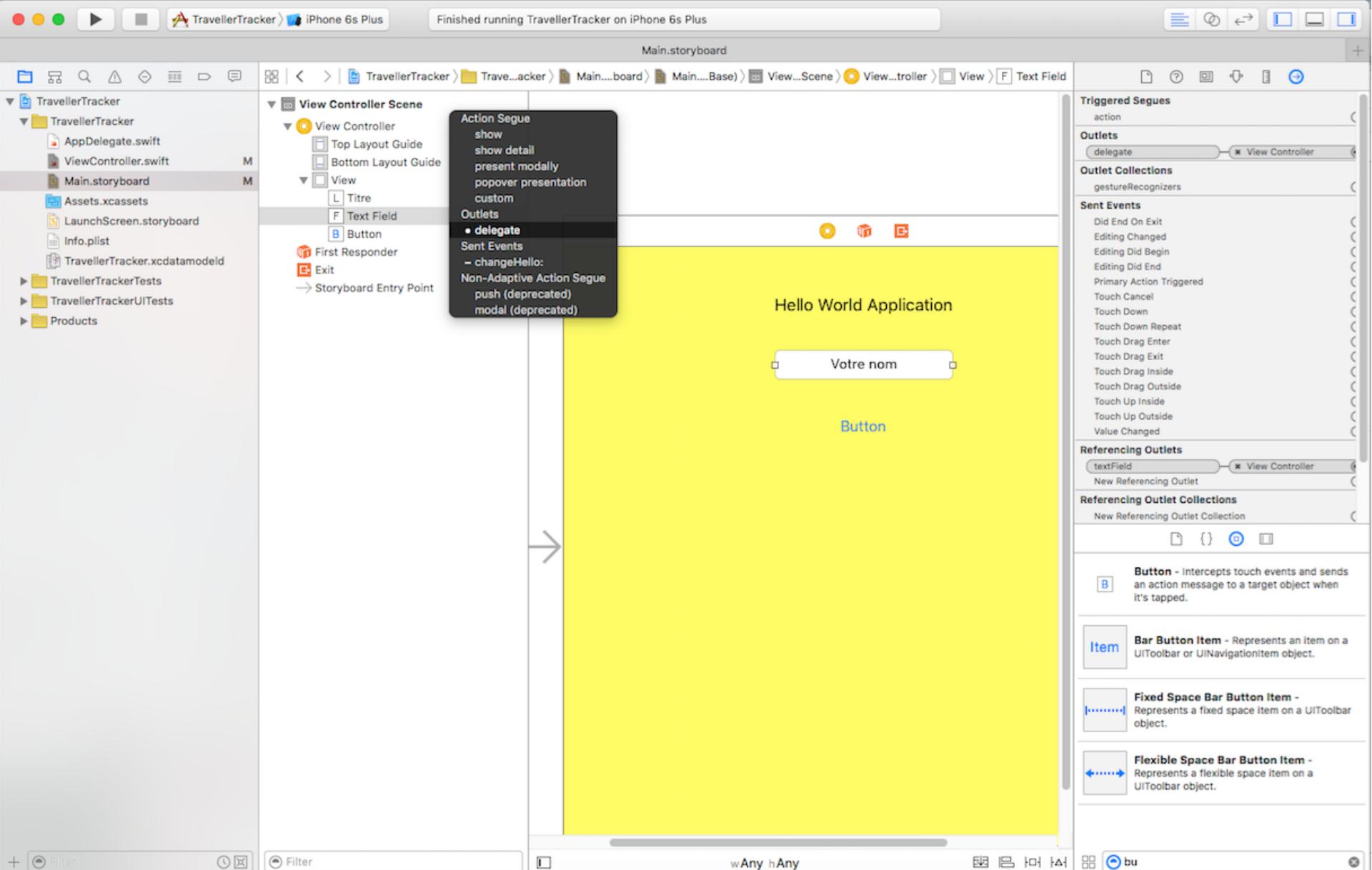
Cliquez sur l'éditeur standard, sur le Controller,
Cliquez sur l'utilitaire de connection pour les visualiser
on peut alors voir le détail des connexions et actions

Déclarer un objet délégué pour le TextField

54

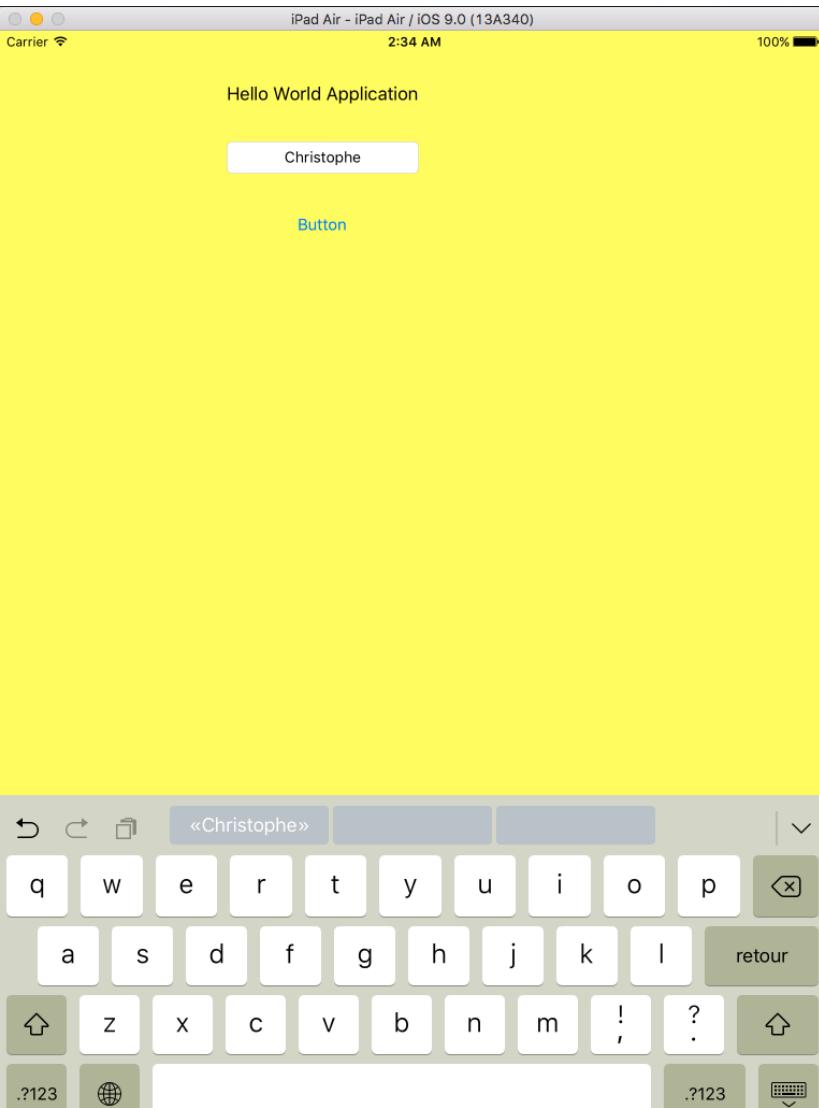


- Un délégué est l'objet qui recevra le message de l'émetteur ;
- On veut récupérer la saisie de l'utilisateur, lorsqu'il cliquera sur le bouton ;
- il faut donc qu'un objet reçoive les messages du TextField au déclenchement de l'action du bouton ;
- Il faut donc déclarer un objet comme délégué du TextField : le contrôleur.

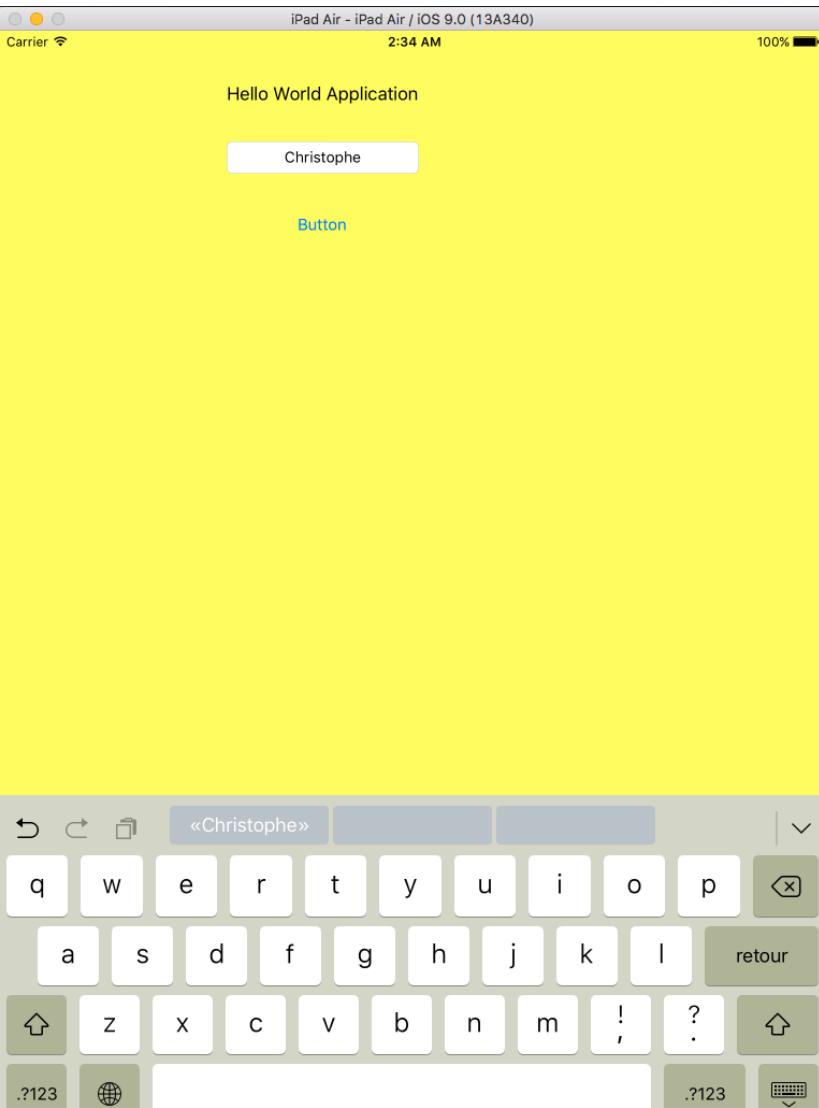


1)
2)
3)

Affichez le storyboard,
Control-glissez le TextField vers le contrôleur
Choisissez « delegate »



- Lancez le simulateur
- vous pouvez désormais saisir un nom
- mais vous ne pouvez pas valider,
- ou annuler la saisie
- ni même enlever le clavier



- Lancez le simulateur
- vous pouvez désormais saisir un nom
- mais vous ne pouvez pas valider,
- ou annuler la saisie
- ni même enlever le clavier

Il faut pour cela implémenter les méthodes de délégation

```
1 //  
2 // ViewController.swift  
3 // TravellerTracker  
4 //  
5 // Created by Fiorio Christophe on 14/10/2015.  
6 // Copyright © 2015 Fiorio Christophe. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 class ViewController: UIViewController {  
12  
13     var nom : NSString = "";  
14  
15     override func viewDidLoad() {  
16         super.viewDidLoad()  
17         // Do any additional setup after loading the view, typically from a nib.  
18     }  
19  
20     override func didReceiveMemoryWarning() {  
21         super.didReceiveMemoryWarning()  
22         // Dispose of any resources that can be recreated.  
23     }  
24  
25     @IBOutlet weak var titre: UILabel!  
26     @IBOutlet weak var textField: UITextField!  
27  
28     @IBAction func changeHello(sender: AnyObject) {  
29         if let nomsaisi = self.textField.text{  
30             self.nom = nomsaisi;  
31         }  
32         else{  
33             self.nom = "@World";  
34         }  
35         self.titre.text = "Hello \(self.nom)";  
36     }  
37  
38  
39 }
```

Rajoutez une propriété `NSString nom` pour stocker le nom saisi
Notez que cette propriété devrait être dans le modèle
compléter le code de `changeHello` pour changer le titre

Hello Christophe

Christophe

Button

- Lancez le simulateur
- Cliquez sur le bouton
- le titre change

Mais on ne peut toujours pas gérer la saisie
dans le TextField

Configurer le contrôleur pour agir en tant que délégué du TextField

59



- Dans une application IOS, le clavier est montré automatiquement quand un utilisateur sélectionne (désigne comme « premier répondeur ») un élément d'interface nécessitant une saisie clavier ;
- Le clavier est automatiquement retiré lorsque l'objet n'est plus considéré comme « premier répondeur » ;
- Mais comme on ne peut envoyer de message du clavier vers le contrôleur, pas moyen de faire perdre ce statut au TextField lorsque l'utilisateur appuie sur return

Configurer le contrôleur pour agir en tant que délégué du TextField

59



Configurer le contrôleur pour agir en tant que délégué du TextField

60



- Le protocole `UITextFieldDelegate` comprend la méthode `textFieldShouldReturn` que le Textfield appelle lorsque l'utilisateur appuie sur return
- Déclarez le contrôleur comme implémentant le protocole `UITextFieldDelegate`
`class ViewController : UIViewController, UITextFieldDelegate`
- Implémentez la méthode
`func textFieldShouldReturn(textField: UITextField) -> Bool`
La méthode doit dire au TextField de perdre le statut de « premier répondeur »
- Pour cela, elle fait appel à la méthode `textField.resignFirstResponder()`

```
//  
//  ViewController.swift  
//  TravellerTracker  
//  
import UIKit  
  
class ViewController: UIViewController {  
  
    61   var nom : NSString = "";  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        // Do any additional setup after loading the view, typically from a nib.  
    }  
    override func didReceiveMemoryWarning() {  
        super.didReceiveMemoryWarning()  
        // Dispose of any resources that can be recreated.  
    }  
    @IBOutlet weak var titre: UILabel!  
    @IBOutlet weak var textField: UITextField!  
    @IBAction func changeHello(sender: AnyObject) {  
        if let nomsaisi = self.textField.text{  
            self.nom = nomsaisi;  
        }  
        else{  
            self.nom = "@World";  
        }  
        self.titre.text = "Hello \(self.nom)";  
    }  
    func textFieldShouldReturn(textField: UITextField) -> Bool {  
        // Hide the keyboard.  
        textField.resignFirstResponder()  
        return true  
    }  
}
```

- Initialisez le titre avec « Hello, World! »
- Modifiez le code pour que le titre change au return
- Notez que le bouton est désormais inutile

Android studio (1)

Langage Java



Java...

- Développé initialement par James Gosling en 1995
- Actuellement version 8, avec différentes versions disponibles :
 - Java SE (version standard),
 - J2EE (framework pour application d'entreprise 3-tiers),
 - J2ME (mobile application).
- Orienté objet,
- Simple (à relativiser) avec une syntaxe « à la » C
- Portable
- Performances ?

Variables et constantes

64



- **final** pour définir des constantes (attention final est plus que le simple **let** de swift ou **const** de C)

```
final int c = 3;
```

- définition des variables « à la C »

```
int myVariable = 42;
```

- Conversion implicite mais possibilité de Cast

```
int x = 5/2;
float y = (float) 5 / (float) 2;
```

Tableaux

65



- Les tableaux utilisent la syntaxe []

```
double[] myArray;
```

- Les tableaux Java sont des objets

- initialisation par appel au constructeur :

```
double[] myArray = new double[4];
double[] otherAr = {0.0, 0.1, 0.2, 0.3}
```

- longueur du tableau obtenu par accès à un attribut

```
myarray.length == 4
```

- possibilité de parcourir les éléments à la « foreach »

```
for(double elt: other){
    System.out.println(elt);
}
```

Structure de contrôle

66



On retrouve les grands classiques : `if`, `switch`, `for-in`, `for`, `while` et `do-while`

```
int[] individualScores = {75, 43, 103, 87, 12};  
int teamScore = 0  
for(int score: individualScores {  
    if(score > 50){  
        teamScore += 3;  
    }  
    else {  
        teamScore += 1  
    }  
}  
System.out.println(teamScore)
```

Le `switch` de Java fonctionne comme celui du C

Fonctions

67



- Les fonctions se déclarent comme en C
- l'appel se fait classiquement par le nom de la fonction suivi de parenthèses avec la liste des valeurs des paramètres

```
String greet(String name, String day){  
    return String("Hello" + name + ", today is " + day);  
}  
System.out.println(greet("Bob", "Tuesday"))
```

Les objets

68



- `class` permet de définir un objet
- les *propriétés* sont définies comme des variables
- les *méthodes* sont définies comme les fonctions
- on accède aux membres via la *notation pointée*
- Le constructeur est une fonction particulière qui n'a pas de valeur de retour et porte le même nom que la classe
- On peut associer un main à chaque objet permettant de le tester indépendamment. De fait le programme principal sera codé par un objet particulier.

```
class NamedShape {  
    int numOfSides;  
    String name;  
  
    public NamedShape(String s){  
        this.numOfSides = 0;  
        this.name = s;  
    }  
  
    public String toString(){  
        return "A shape with "+ String(numOfSides)+" sides.";  
    }  
}
```



- On peut hériter grâce au mot clef **extends**
- On peut surcharger les méthodes grâce simplement en les redéfinissant
- une méthode **final** ne peut être surchargée
- possibilité de protéger les accès par les classiques mots clefs **public**, **protected**, **private**
- le mot clef **super** permet d'accéder à un élément de la superclass qui aurait été surchargé ; il permet aussi d'invoquer le constructeur de la superclass
- **instanceof** permet de vérifier le type (au sens objet) de l'instance

```

class EquilateralTriangle extends NamedShape {
    double sideLength;

    public EquilateralTriangle(double sideLength, String name) {
        super(name);
        this.sideLength = sideLength;
        this.numberOfSides = 3;
    }

    double getPerimeter(){
        return 3.0 * sideLength;
    }
    void setPerimeter(double val){
        this.sideLength = val / 3.0;
    }

    String toString(){
        return "An equilateral triangle with sides of length" + sideLength;
    }
}

```

```

EquilateralTriangle triangle = EquilateralTriangle(3.1,"a triangle");
System.out.println(triangle.getPerimeter())
triangle.setPerimeter(9.9);

```



- ❑ Possibilité aussi de définir des variables et des constantes de classe :
 - Elles se définissent avec le mot clef **static**.
 - Le mot clef **final** quant à lui permet d'indiquer une définition finale, donc une constante dans le cas d'une variable ou une méthode ne pouvant être surchargée dans le cas d'une méthode
 - **static** peut être combiné avec **final** pour définir une constante de classe

```
class SomeClass {  
    public final int number = 3;  
}
```



Les classes abstraites

73



- Les classes abstraites peuvent (pas une obligation) contenir des méthodes abstraites, c'est à dire des méthodes déclarées comme **abstract** et étant juste déclarée
- Une classe comportant une méthode **abstract** *doit* être **abstract** elle-même
- Une classe déclarée **abstract** ne peut être instanciée : il faut définir une autre classe qui hérite de celle-ci et qui définit les méthodes **abstract**.

Les interfaces : types abstraits

74



- ❑ Similaire à une classe abstraite mais ne peut pas contenir de variables ou de méthodes définies : il ne peut y avoir que de simple déclarations de méthodes et des constantes
- ❑ Définit un nouveau type abstrait
- ❑ N'importe quelle classe peut « implémenter » des interfaces
- ❑ Implémenter une interface impose de définir toutes les méthodes de l'interface

```
interface ExampleInterface {  
    void adjust();  
}  
class SimpleClass implements ExampleInterface {  
    String simpleDescription;  
    SimpleClass(){  
        this.simpleDescription = String("description is ");  
    }  
    void adjust(){  
        simpleDescription += " Now 100% adjusted.";  
    }  
}
```

Android studio (2)

Langage Kotlin



Variables et constantes

76



- `val` pour définir des constantes
- `var` pour définir des variables

```
var myVariable = 42
myVariable = 50
val myConstant = 42
```

- Typage dynamique

```
val implicitInteger = 70
val implicitDouble = 70.0
val explicitDouble: Double = 70
```

- Conversion implicite : appelé smart cast

```
val label = "The width is "
val width = 94
val widthLabel = label + width
```

Tableaux et dictionnaires

77



- Les tableaux et dictionnaires utilisent la syntaxe []

```
var shoppingList = arrayOf("catfish", "water", "tulips", "blue  
paint")
```

```
shoppingList[1] = "bottle of water"
```

```
val asc = Array(5, { i -> (i * i).toString() })
```

```
var occupations = hashMapOf(
```

```
    "Malcolm" to "Captain",
```

```
    "Kaylee" to "Mechanic"
```

```
)
```

```
occupations["Jayne"] = "Public Relations"
```

Structure de contrôle

78



On retrouve les grands classiques : `if`, `switch`, `for-in`, `for`,
`while` et `repeat-while`

```
val individualScores = ArrayOf(75, 43, 103, 87, 12)
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
print(teamScore)
```

```

val vegetable = "red pepper"
when vegetable {
    "celery" -> print("Add some raisins and make ants on a log.")
    "cucumber", "watercress" -> print("That would make a good tea
sandwich.")
    vegetable.hasSuffix(" pepper") -> print("Is it a spicy $x ?")
    else -> print("Everything tastes good in soup.")
}

```

Le when de Kotlin est aussi riche que le switch de Swift et donc bien plus complet que celui de Java :

- s'arrête dès qu'un case est accepté
- possibilité d'avoir des patterns
- gère les intervalles : case 1..5



Le for-in fonctionne comme celui de Python ou de Swift:

```
val interestingNumbers = hashMapOf(  
    "Prime" to ArrayOf(2, 3, 5, 7, 11, 13),  
    "Fibonacci" to ArrayOf(1, 1, 2, 3, 5, 8),  
    "Square" to ArrayOf(1, 4, 9, 16, 25),  
)  
var largest = 0  
for (kind, numbers) in interestingNumbers {  
    for number in numbers {  
        if number > largest {  
            largest = number  
        }  
    }  
}  
print(largest)
```

à noter la possibilité de faire des boucles descendantes :

```
for (i in 6 downTo 0 step 2) {  
    ...  
}
```



Null Safety

- Mécanisme similaire aux optionnels de Swift
- Permet d'éviter la fameuse NullPointerException
- Très utile par exemple pour indiquer qu'une fonction peut ne retourner aucune valeur plutôt que de faire retourner une valeur particulière
- Se marque avec un ?

```
var optionalName: String? = null
var greeting = "Hello!"
if (optionalName != null) {
    greeting = "Hello, ${optionalName!!}"
}
```

- On peut forcer l'évaluer d'une valeur optionnelle par !!
- Certaines opérations (conversion notamment) peuvent renvoyer null => nécessiter d'utiliser des valeurs optionnelles

```
val snumber = "123"
val n : Int? = Int(snumber)
if (n != null) {
    print("snumber converti contient une valeur entière")
}
```

- On peut utiliser les affectations optionnelles pour contrôler :

```
val tnumber = "123"
val nn: Int = if (tnumber != null) tnumber else -1
val nn: Int = tnumber ?: -1
```

- On peut utiliser le chainage optionnel (safe call) :

```
val nn : Int = t?.length
```



Fonctions

83



- `fun` permet de déclarer une fonction
- l'appel se fait classiquement par le nom de la fonction suivi de parenthèses avec la liste des paramètres
- les paramètres sont nommés,
- les paramètres *peuvent* être nommés à l'appel
- à la déclaration, les paramètres sont séparés du type de retour par :

```
fun greet(name: String, day: String) : String {  
    return "Hello $name, today is $day."  
}  
greet(name= "Bob", day="Tuesday")
```

- les fonctions admettent un nombre variable de paramètres qui seront rassemblés dans un tableau

```
fun calculateStatistics(varargs scores: UInt) : Float {  
    var min : UInt = scores[0]  
    var max : UInt = scores[0]  
    var sum : UInt = 0  
  
    for score in scores {  
        if score > max {  
            max = score  
        } else if score < min {  
            min = score  
        }  
        sum += score  
    }  
    return (Float(sum)/Float(scores.size))  
}  
val resultats = calculateStatistics(5, 3, 100, 3, 9)  
print(resultats)
```



- ❑ les fonctions peuvent être imbriquées
- ❑ les fonctions peuvent prendre des fonctions en paramètre
- ❑ les fonctions peuvent renvoyer des fonctions

```
fun makeIncrementer() : (Int) -> Int {  
    fun addOne(number: Int) -> Int {  
        return 1 + number  
    }  
    return addOne  
}  
var increment = makeIncrementer()  
increment(7)
```

- ❑ les fonctions sont en fait un cas particulier des « lambdas »



Lambdas (cf Closures en Swift)

86



- Blocs de code qui peuvent être appelés, nommés, utilisés

- Écrits entre { }, le code séparé des paramètres par `in`

```
numbers.map({  
    number: Int -> 3 * number  
})
```

- Si il n'y a qu'un seul paramètre, on peut utiliser le paramètre implicite `it`

```
let sortedNumbers = numbers.filter { it > 0 }
```

Les objets

87



- class permet de définir un objet
- les *propriétés* sont définies comme les variables et les constantes
- les *méthodes* sont définies comme les fonctions
- on accède aux membres via la *notation pointée*
- il doit y avoir un *constructeur primaire* et il peut y avoir des *constructeurs secondaires*
- this permet de différencier paramètres d'une méthode d'une propriétés
- les propriétés peuvent définir des *getter* et des *setter*

```

class NamedShape(name: String, n : Int) {
    var name: String = name // déclaration de propriété
    var number0fSides = n // déclaration de propriété

    init {
        code exécuté à l'initialisation
    }

    constructor(name: String) : this(name,2){
        // number0fSides=2
    }

    fun simpleDescription() : String {
        return "A shape with $number0fSides sides."
    }
}

```

- On peut hériter (notation classique :), la classe dont on hérite doit être **open**
- On peut surcharger les méthodes grâce au mot clef **override**



```
class EquilateralTriangle(sidelen: Double, name: String) : NamedShape(name,
3) {
    var sideLength: Double = sidelen

    var perimeter: Double
        get() = 3.0 * sideLength
        set(newvalue) {
            sideLength = newValue / 3.0
        }
}

override fun simpleDescription(): String {
    return "An equilateral triangle with sides of length
$sideLength."
}
}
var triangle = EquilateralTriangle(sideLength: 3.1, name: "a triangle")
print(triangle.perimeter)
triangle.perimeter = 9.9
```



Les interfaces : classes abstraites

90



- Définit un nouveau type abstrait
- N'importe quelle classe peut « implémenter une interface »
- Une classe peut implémenter plusieurs interfaces

```
interface ExampleProtocol {  
    var simpleDescription: String  
    fun adjust()  
}  
  
class SimpleClass: ExampleProtocol {  
    var simpleDescription: String = "A very simple class."  
    var anotherProperty: Int = 69105  
    fun adjust() {  
        simpleDescription += " Now 100% adjusted."  
    }  
}
```

Android Studio (3)

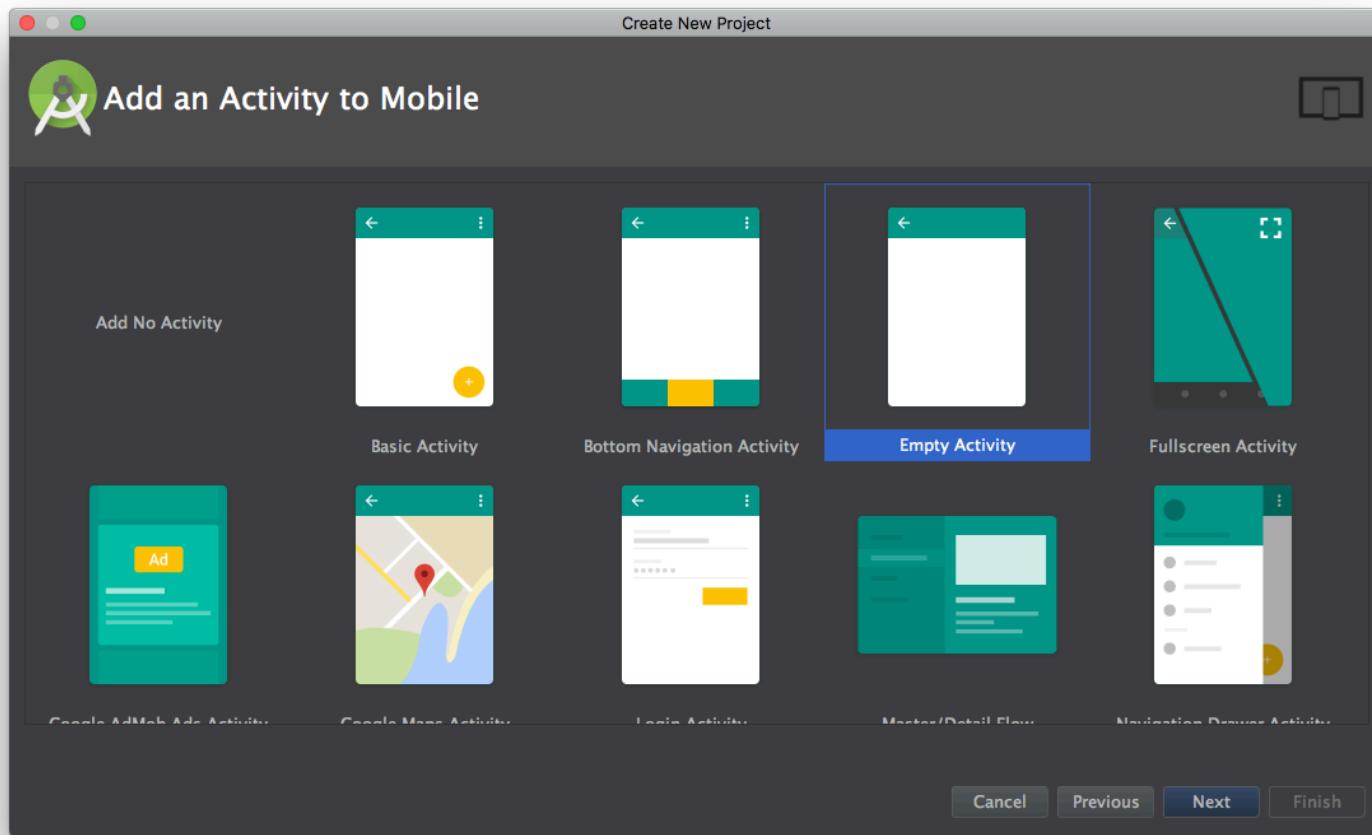
Environnement de développement



Android Studio

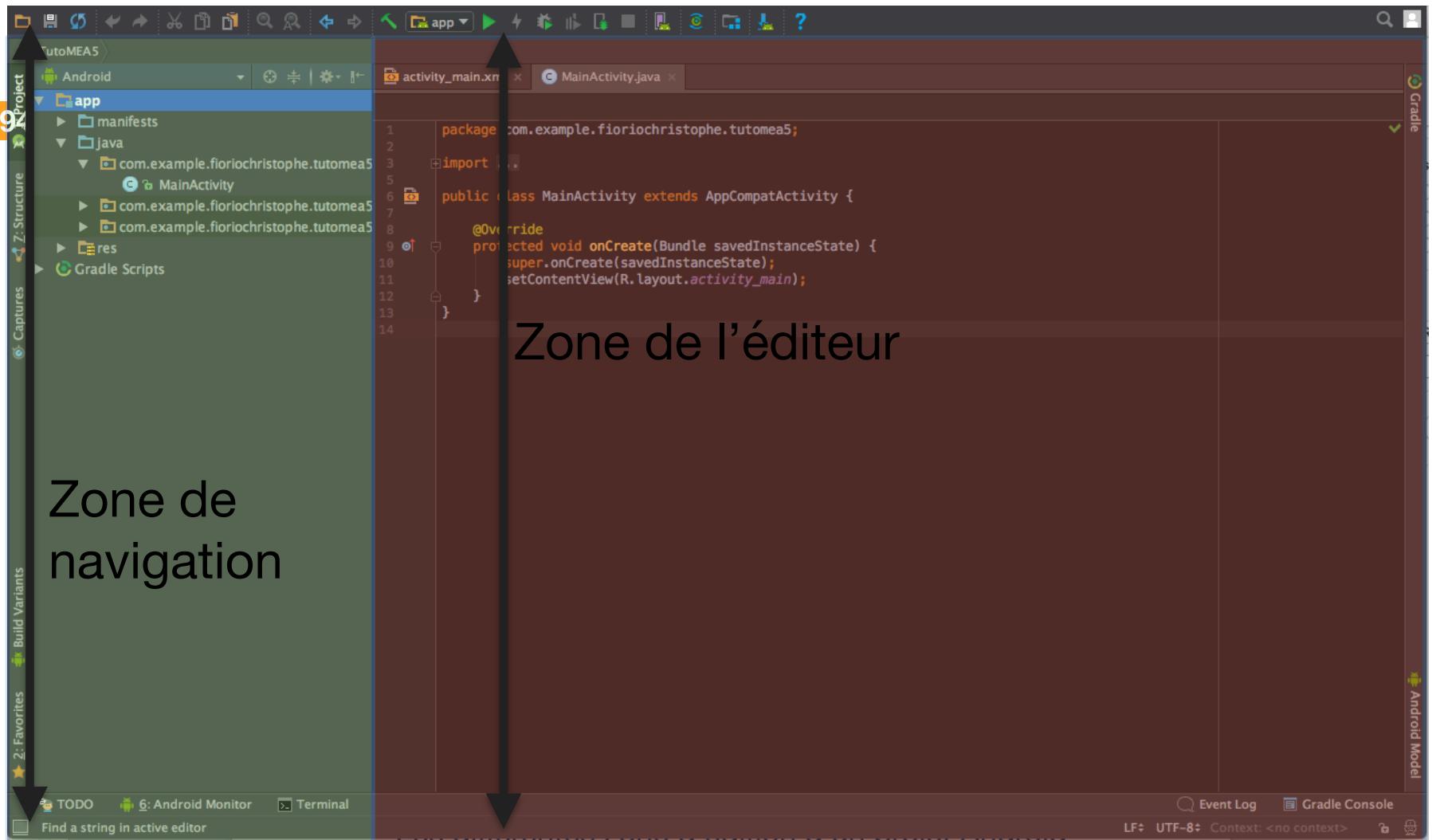


- ❑ activité = écran - 1 seule activité à la fois
- ❑ mais une activité est un peu plus que cela car elle contient le contexte de l'application (active = celle avec laquelle interagit l'utilisateur, suspendue ou *paused* = en partie visible mais pas active, arrêtée) en plus de la gestion de l'interface graphique -> app delegate + view controller
- ❑ vue = éléments graphiques (widgets) généralement rangés dans des containers comme en swing-java
- ❑ vue = ressource, décrite par un fichier xml



Créer un nouveau projet en sélection une

Empty Activity



Les différentes zones d'édition d'un projet Android

94

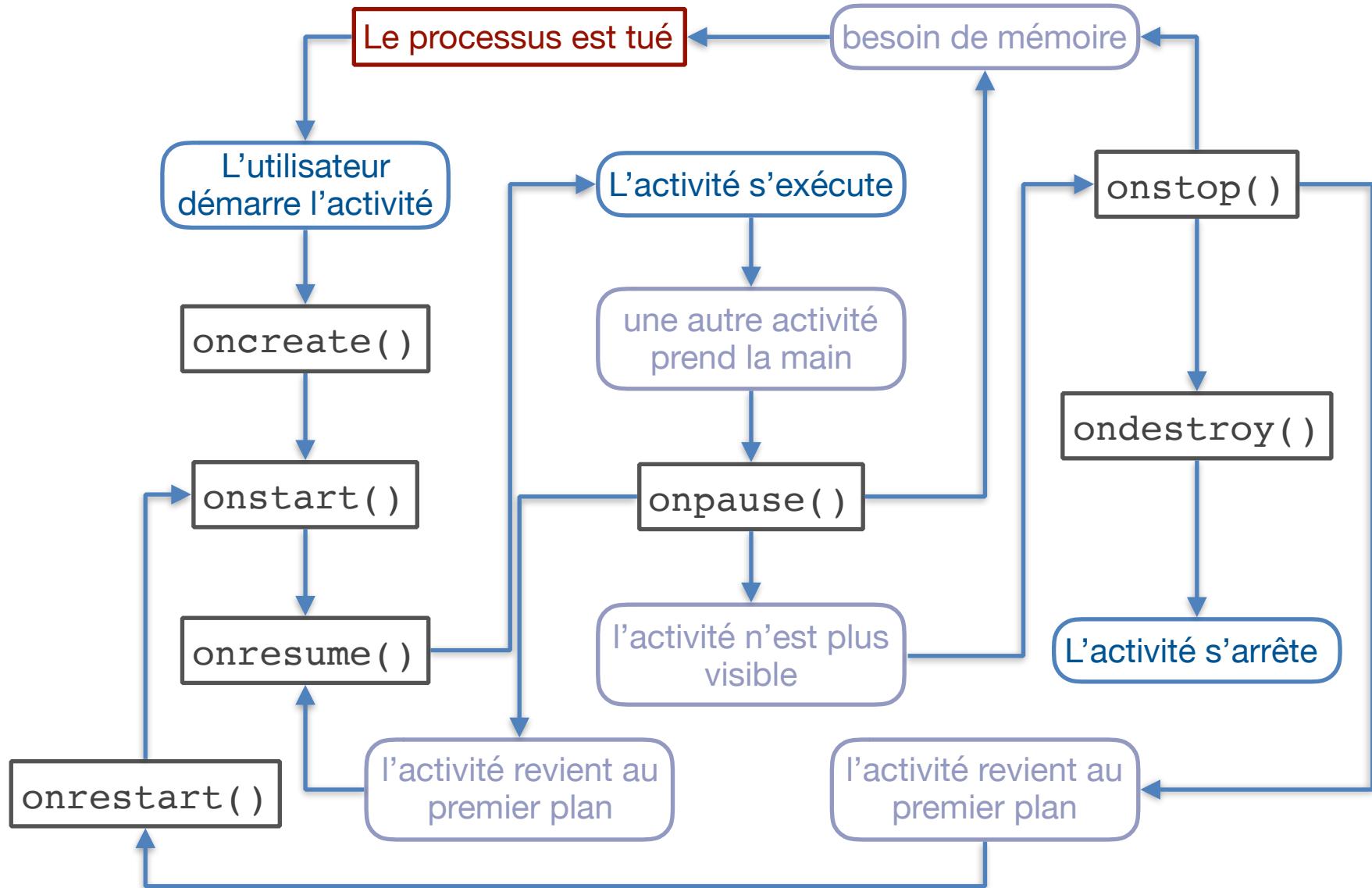
```
1 package com.example.fioriochristophe.tutomea5;
2
3 import ...
4
5 public class MainActivity extends AppCompatActivity {
6     @Override
7     protected void onCreate(Bundle savedInstanceState) {
8         super.onCreate(savedInstanceState);
9         setContentView(R.layout.activity_main);
10    }
11 }
12
13
14
```

activité

Un clic sur MainActivity -> ouvre le fichier source correspondant

Cycle de vie d'une activité

95



Mise en œuvre : « hello world »

Mise en œuvre : « hello world »

96



On modifie le code par défaut de la fonction `onCreate()` pour initialiser la vue avec « Hello World »

Mise en œuvre : « hello world »

On modifie le code par défaut de la fonction `onCreate()` pour initialiser la vue avec « Hello World »

on ajoute du texte au contenu de la vue

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    TextView text = new TextView(this);  
    text.setText("Bonjour, vous me devez 1 000 000€");  
    setContentView(text);  
    //setContentView(R.layout.activity_main);  
}
```

On verra plus tard la signification de ce code
Pour l'instant on le commente

Mise en œuvre : « hello world »

96



On modifie le code par défaut de la fonction `onCreate()` pour initialiser la vue avec « Hello World »

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    TextView text = new TextView(this);  
    text.setText("Bonjour, vous me devez 1 000 000€");  
    setContentView(text);  
    //setContentView(R.layout.activity_main);  
}
```

on ajoute du texte au contenu de la vue

On verra plus tard la signification de ce code
Pour l'instant on le commente

Pour que la vue `Text` soit connue, il faut importer le package la définissant :

```
import android.widget.TextView;
```

Première simulation

97



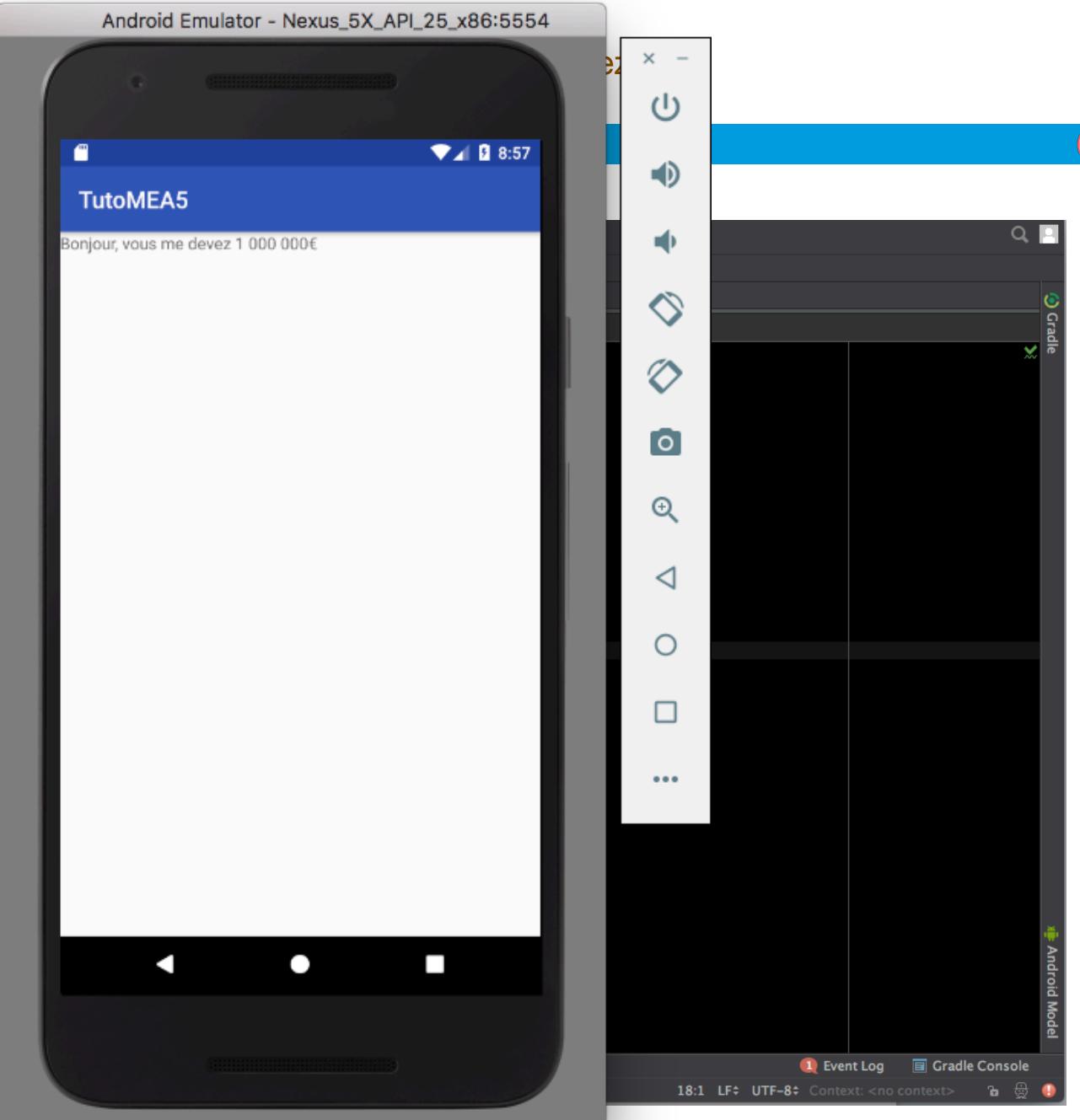
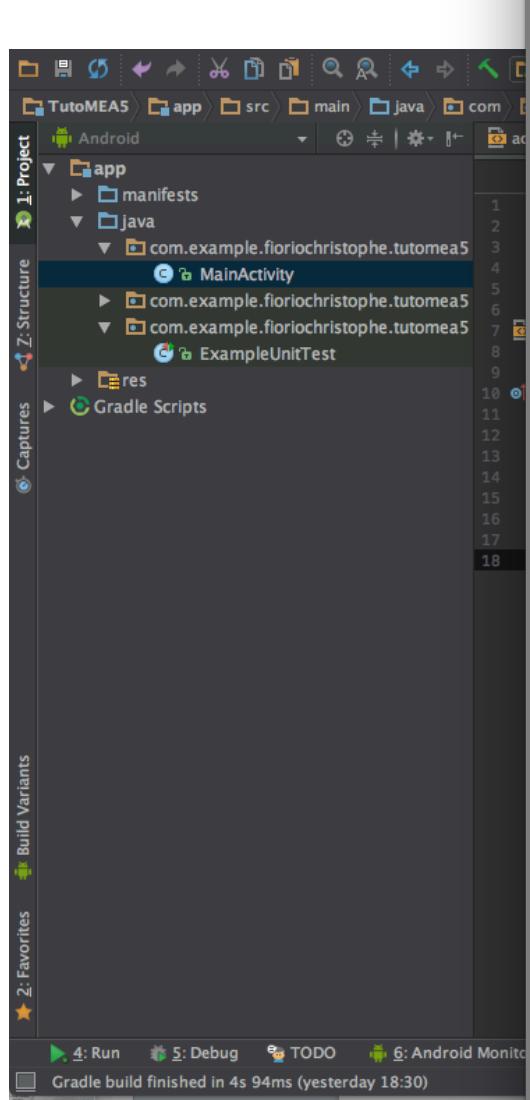
The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TutoMEAS" and contains an "app" module. The "app" module has "manifests", "java", and "res" directories. The "java" directory contains "MainActivity" and "ExampleUnitTest".
- MainActivity.java Content:**

```
1 package com.example.fioriochristophe.tutomea5;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.TextView;
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        TextView text = new TextView(this);
13        text.setText("Bonjour, vous me devez 1 000 000€");
14        setContentView(text);
15    }
16
17
18 }
```
- Toolbars and Menus:** Standard Android Studio toolbars and menus are visible at the top.
- Side Panels:** The "Project", "Android", "Structure", "Captures", "Build Variants", "Favorites", and "Android Model" panels are open on the left and right sides.
- Bottom Bar:** Includes icons for Run, Debug, TODO, Android Monitor, Terminal, Messages, Event Log, and Gradle Console.
- Status Bar:** Shows "Gradle build finished in 4s 94ms (yesterday 18:30)" and "18:1 LF+ UTF-8+ Context: <no context>".

Première s

97



Les ressources

98



Tout ce qui n'est pas code est géré par des ressources. On y trouve par exemple

- ⌚ des images
- ⌚ des textes localisées
- ⌚ ...

Ces ressources sont décrites en langage xml

Ce sont des valeurs constantes.

The screenshot shows the Android Studio interface with the project 'TutoMEAS' open. The left sidebar displays the project structure, including the app module with Java and resources folders, and a Java test file 'ExampleUnitTest'. The main editor window shows the 'strings.xml' file under the 'values' folder in the 'res' directory. The code in the editor is:

```
<resources>
    <string name="app_name">Mon premier tuto</string>
</resources>
```

The 'strings.xml' file is selected in the project tree. The bottom status bar indicates a full build is in progress and an Instant Run detection.



99

Les ressources



Cliquez sur « res »
pour ressource

The screenshot shows the Android Studio interface. On the left, the Project tool window displays the project structure under the 'app' module. A red oval highlights the 'res' folder. A red arrow points from the text 'Cliquez sur < res >' to the 'res' folder in the Project window. The 'values' folder within 'res' contains three files: 'colors.xml', 'strings.xml' (which is selected), and 'styles.xml'. In the center, the Editor pane shows the 'strings.xml' file with the following XML code:

```
<resources>
    <string name="app_name">Mon premier tuto</string>
</resources>
```

At the bottom, the status bar indicates: 'Performing full build and install: // Instant Run detected that a resource referenced from the merged AndroidManifest has changed. Learn more. // (Don't ... (a minute ago) 4:1 LF+ UTF-8 Context: <no context>'. The bottom right corner shows the Android Model icon.

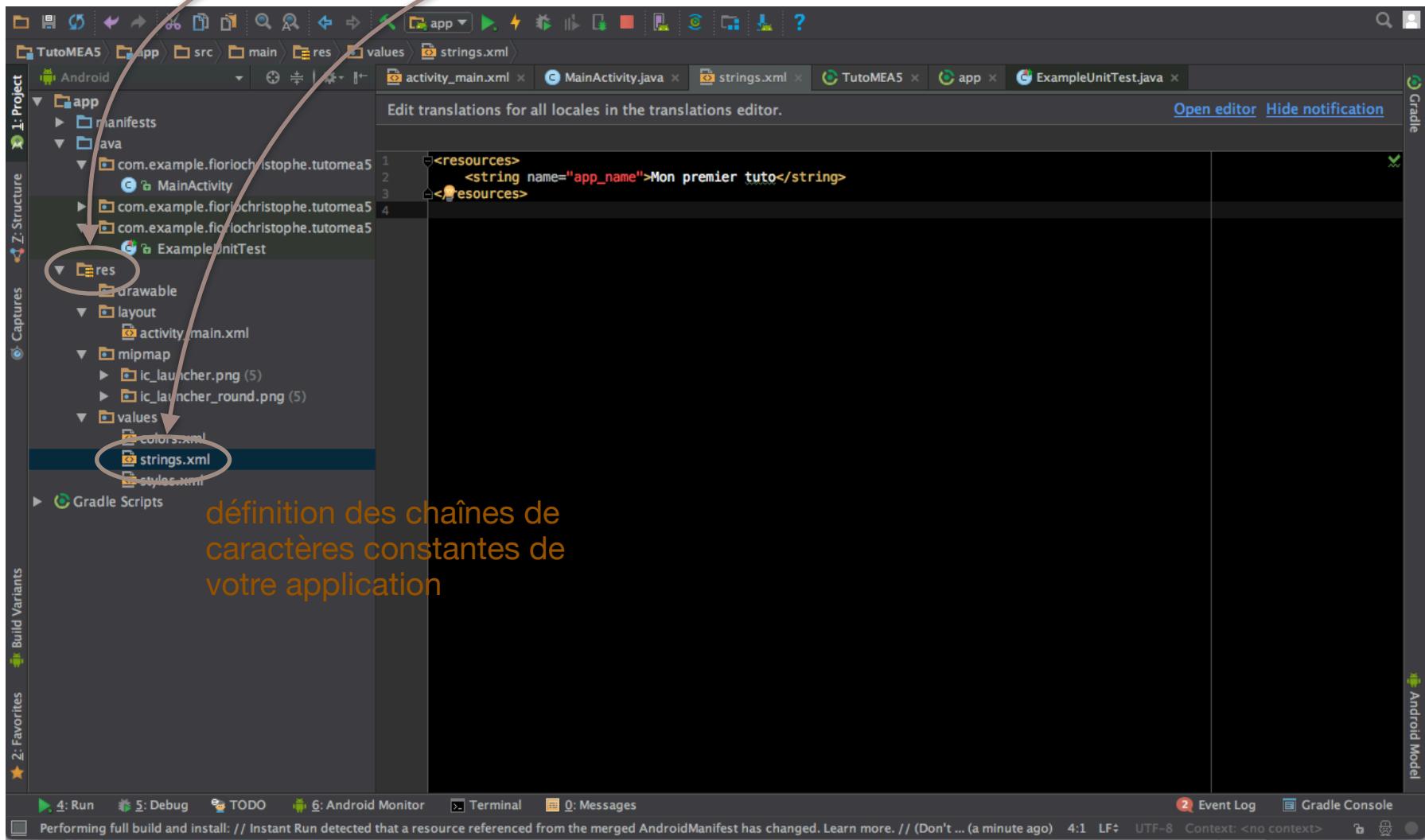


99

Les ressources

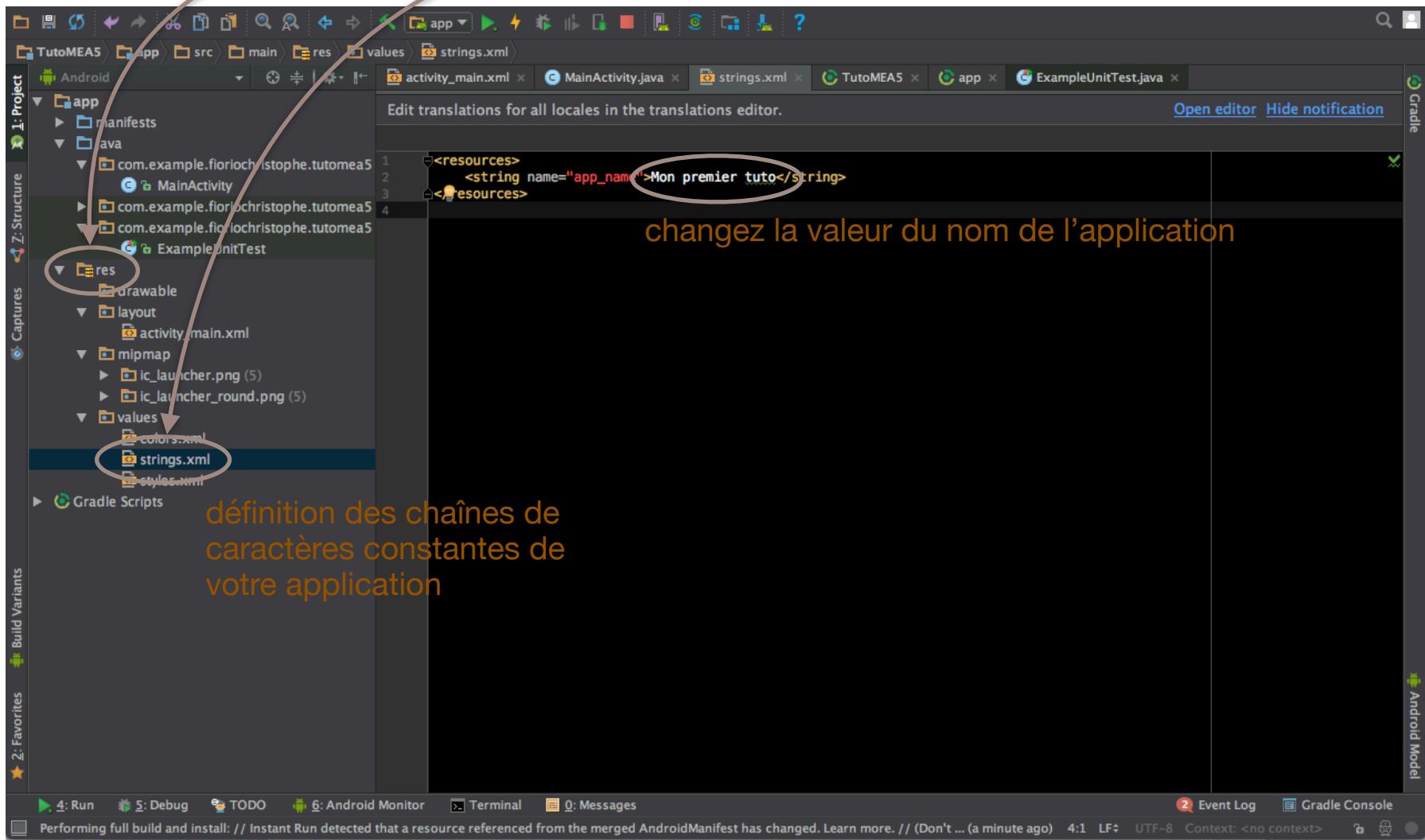
Cliquez sur « res »
pour ressource

double-cliquez sur strings.xml

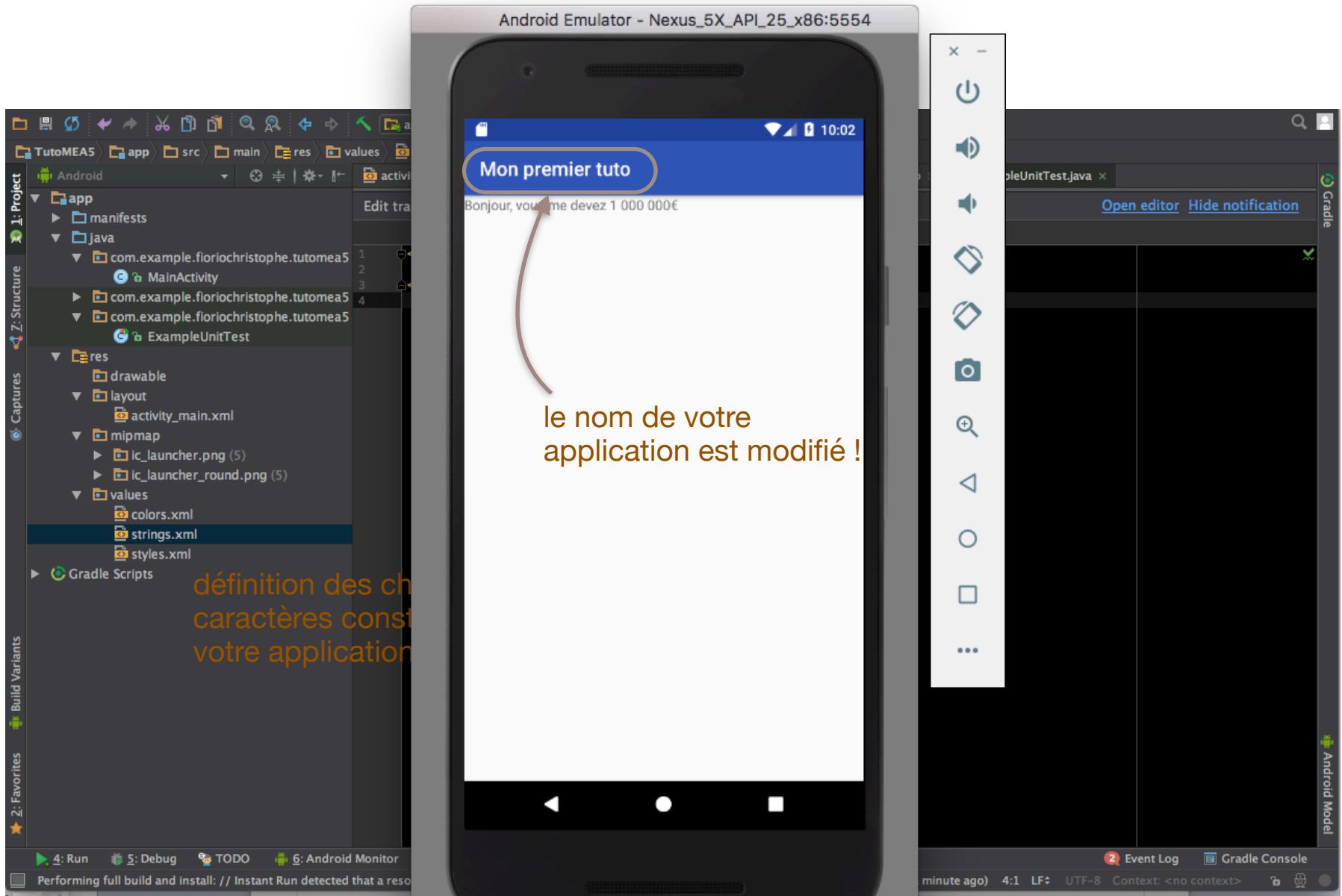


Cliquez sur « res »
pour ressource

double-cliquez sur strings.xml



définition des chaînes de
caractères constantes de
votre application



The screenshot shows the Android Studio interface with the following details:

- Project View:** Shows the project structure under "app".
 - Java:** Contains packages for com.example.fioriochristophe.tutomea5 (MainActivity) and com.example.fioriochristophe.tutomea5 (ExampleUnitTest).
 - res:** Contains drawable, layout, mipmap, and values folders.
 - layout:** Contains activity_main.xml.
 - mipmap:** Contains ic_launcher.png (5) and ic_launcher_round.png (5).
 - values:** Contains colors.xml, strings.xml, and styles.xml.
- Editor View:** Displays the strings.xml file with the following content:

```
<resources>
    <string name="app_name">Mon premier tuto</string>
</resources>
```
- Annotations:** Handwritten annotations explain the resources:
 - points to the drawable folder: "les images : jpeg, png, ou même xml"
 - points to the layout folder: "la disposition des vues"
 - points to the mipmap folder: "les icônes"
 - points to the values folder: "différentes valeurs"
 - points to the strings.xml file in the editor: "mais aussi : menus, raw, etc..."
- Bottom Bar:** Includes icons for Run, Debug, TODO, Android Monitor, Terminal, Messages, Event Log, and Gradle Console.
- Status Bar:** Shows a message: "Performing full build and install: // Instant Run detected that a resource referenced from the merged AndroidManifest has changed. Learn more. // (Don't ... (a minute ago) 4:1 LF+ UTF-8 Context: <no context>".



100

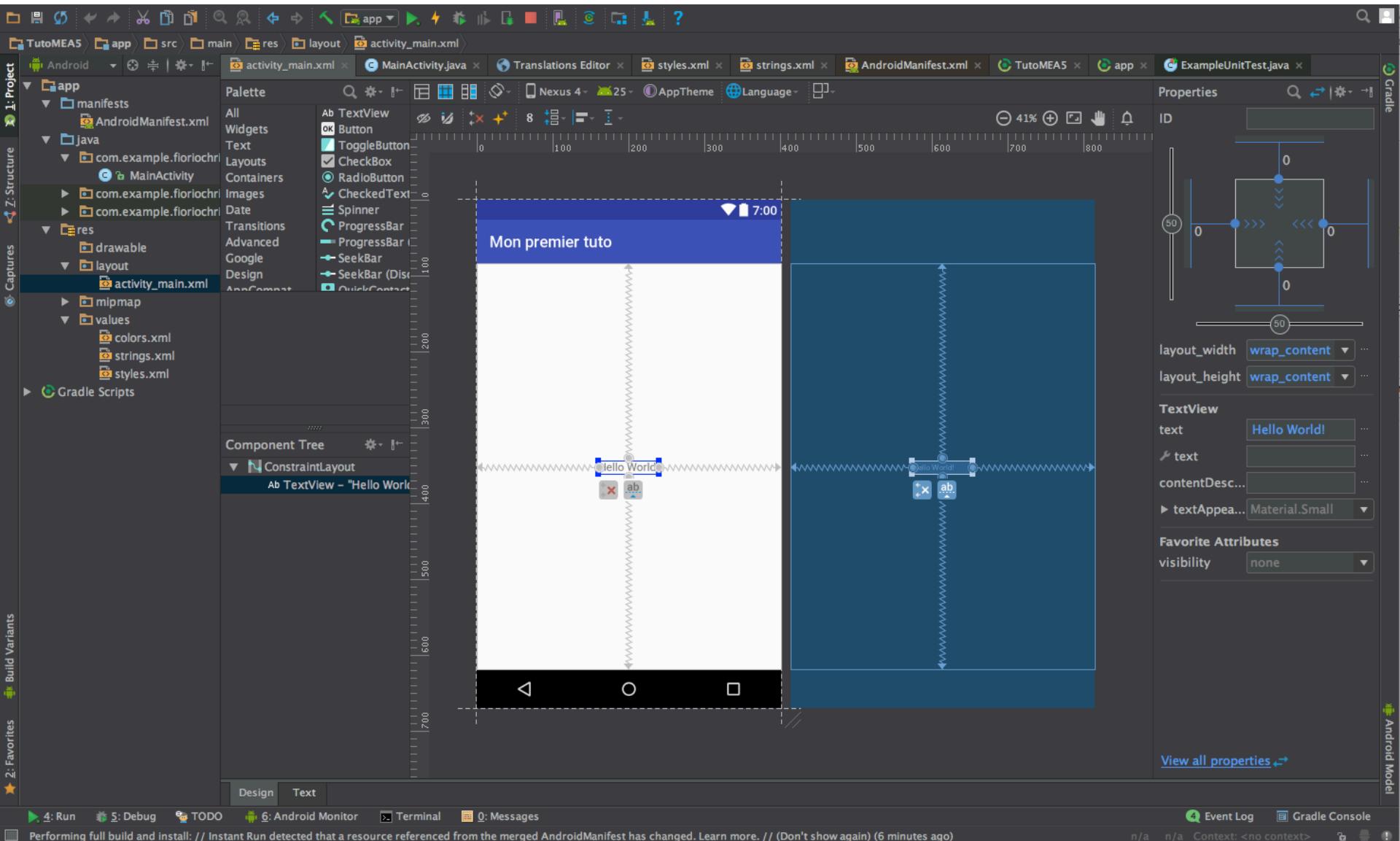
Les ressources

Layout Editor

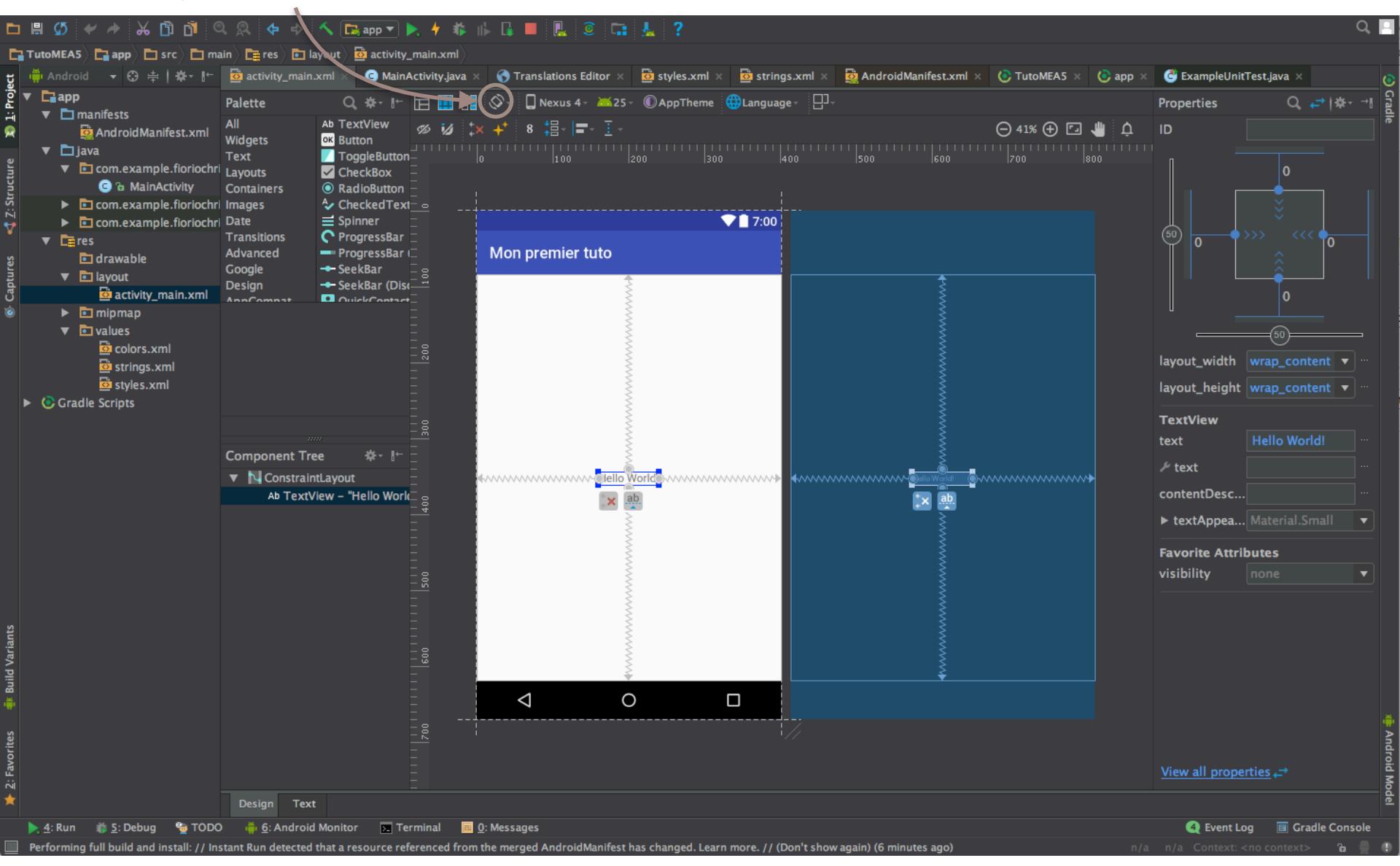
101



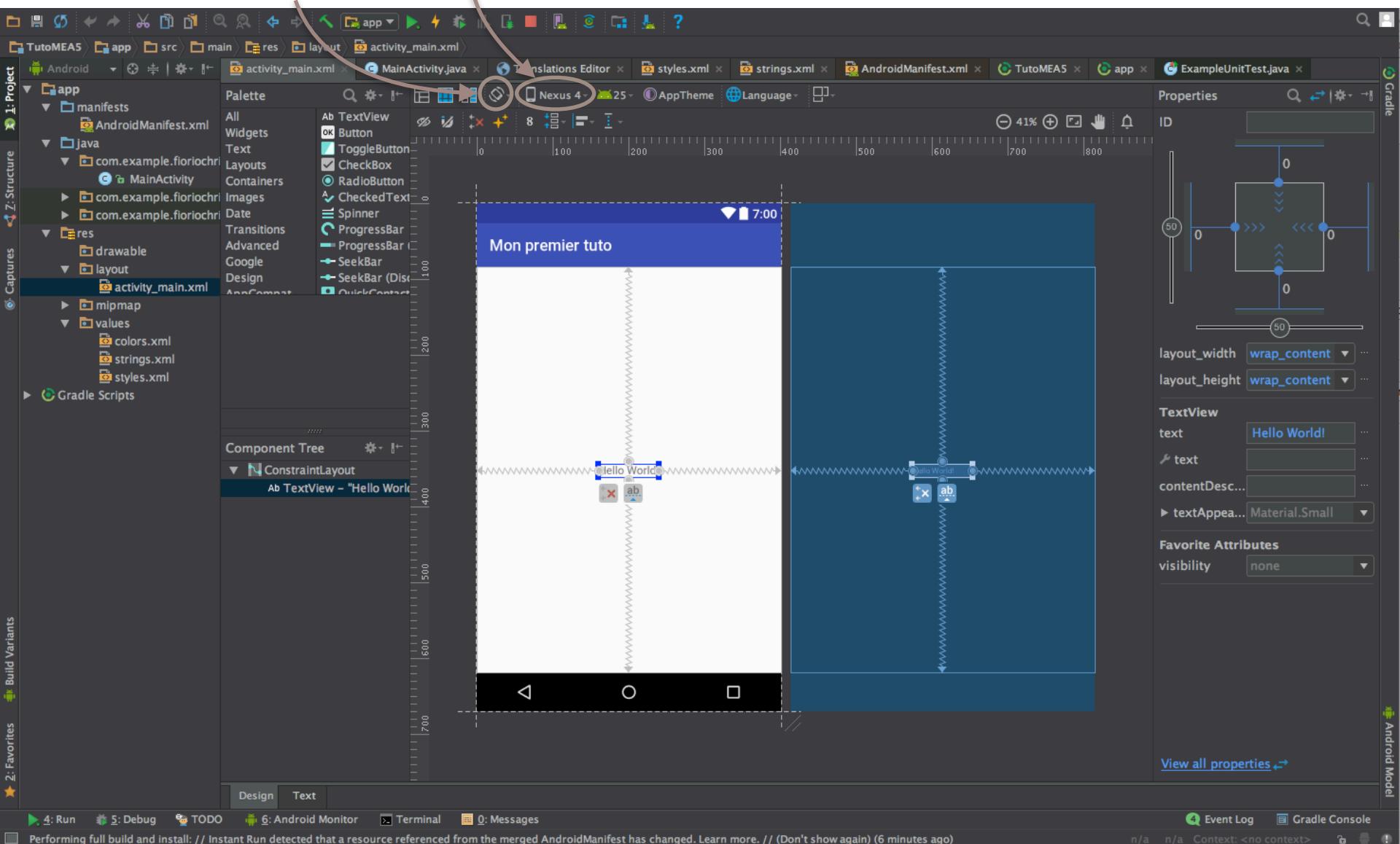
- L'éditeur de mise en page (Layout editor) permet de visualiser un écran et de définir sa mise en page
- La mise en page est définie dans les ressources, et donc en xml
- Android Studio permet d'éditer la mise en page graphiquement
- Il faudra définir un layout pour chaque type d'écran !
- *Attention* : l'outil n'est pas aussi précis, rigoureux ou même sûr que l'édition directe du fichier ressource xml correspondant !



gérer l'orientation

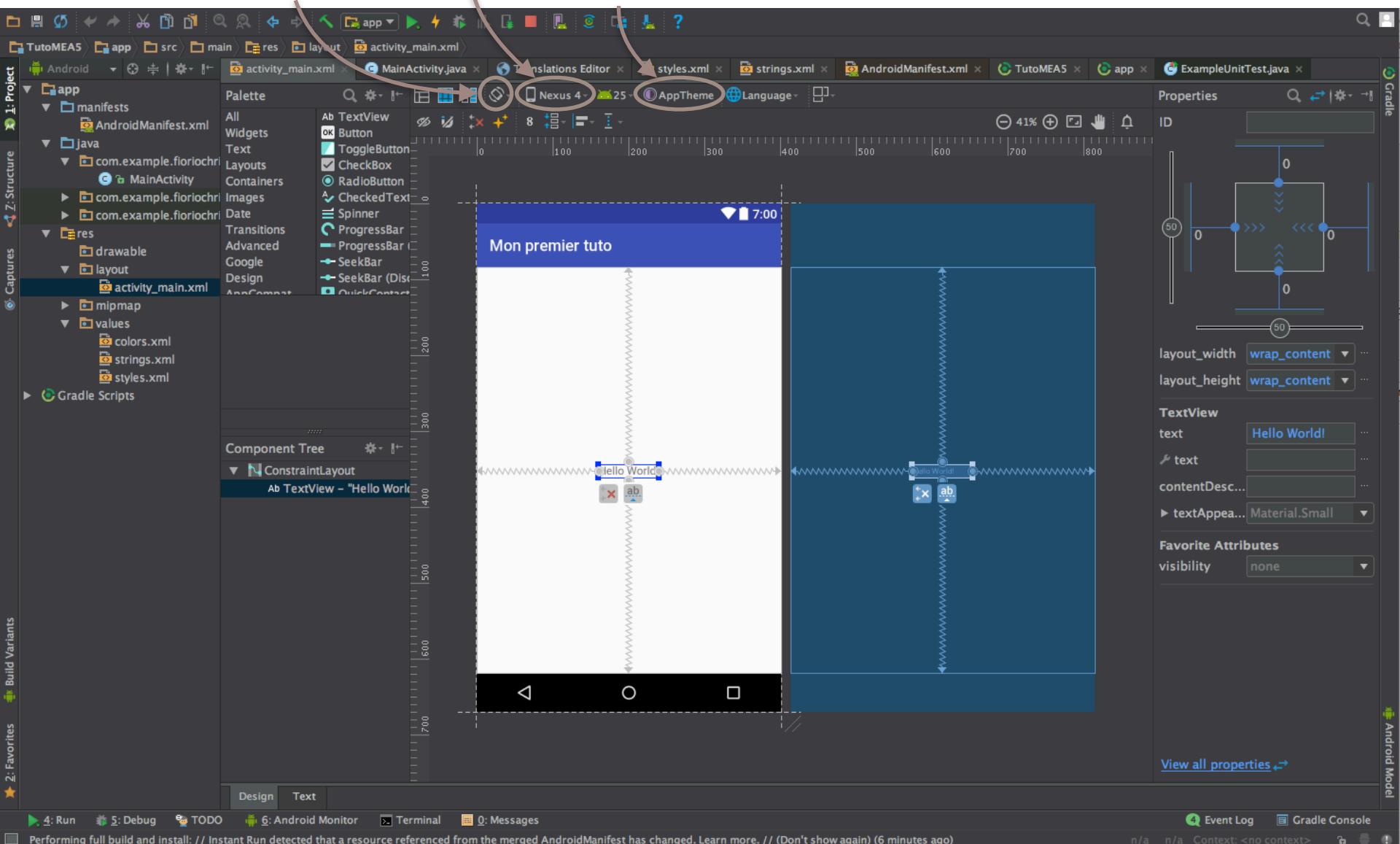


choisir le modèle gérer l'orientation



choisir le modèle
gérer l'orientation

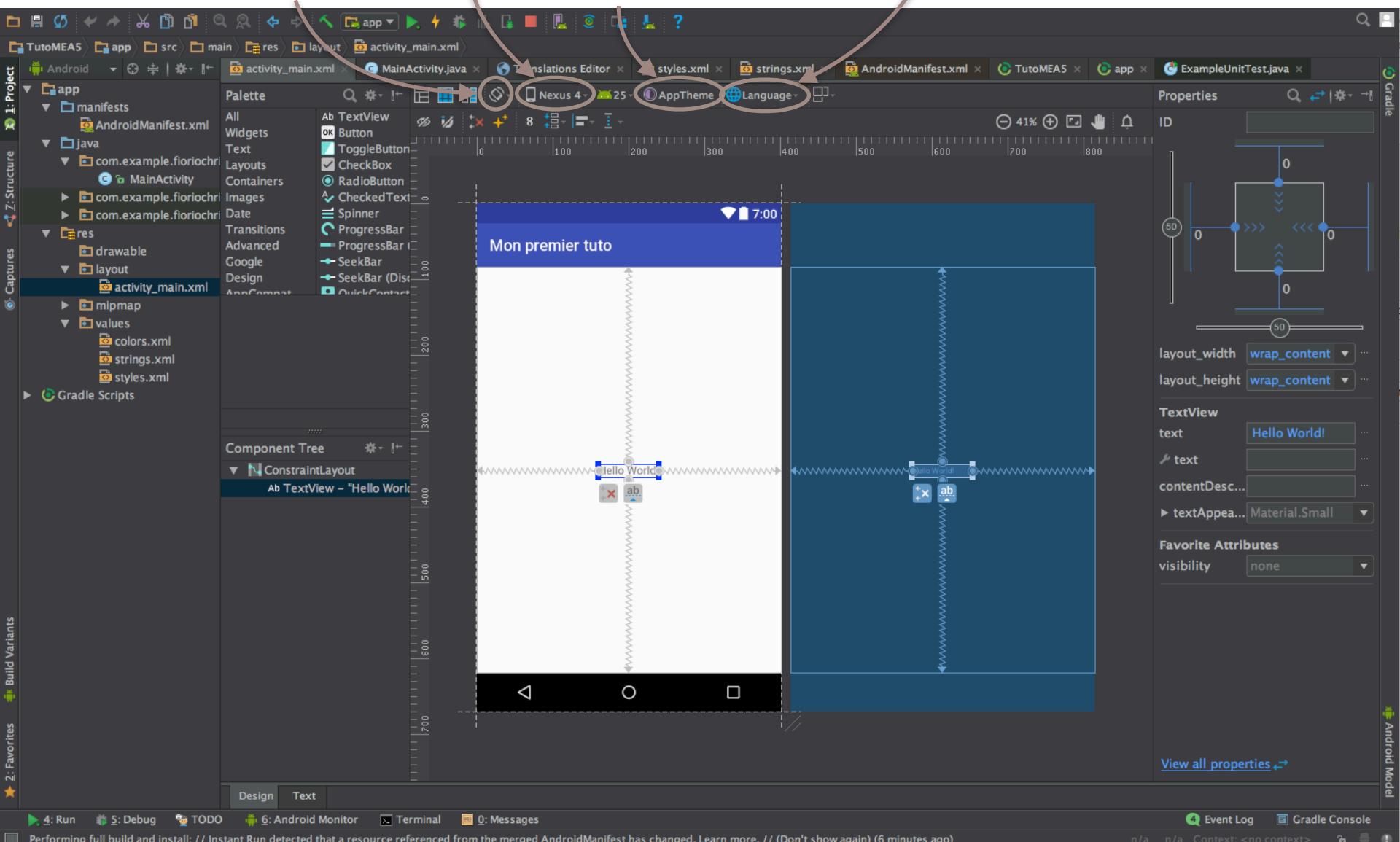
choisir le thème

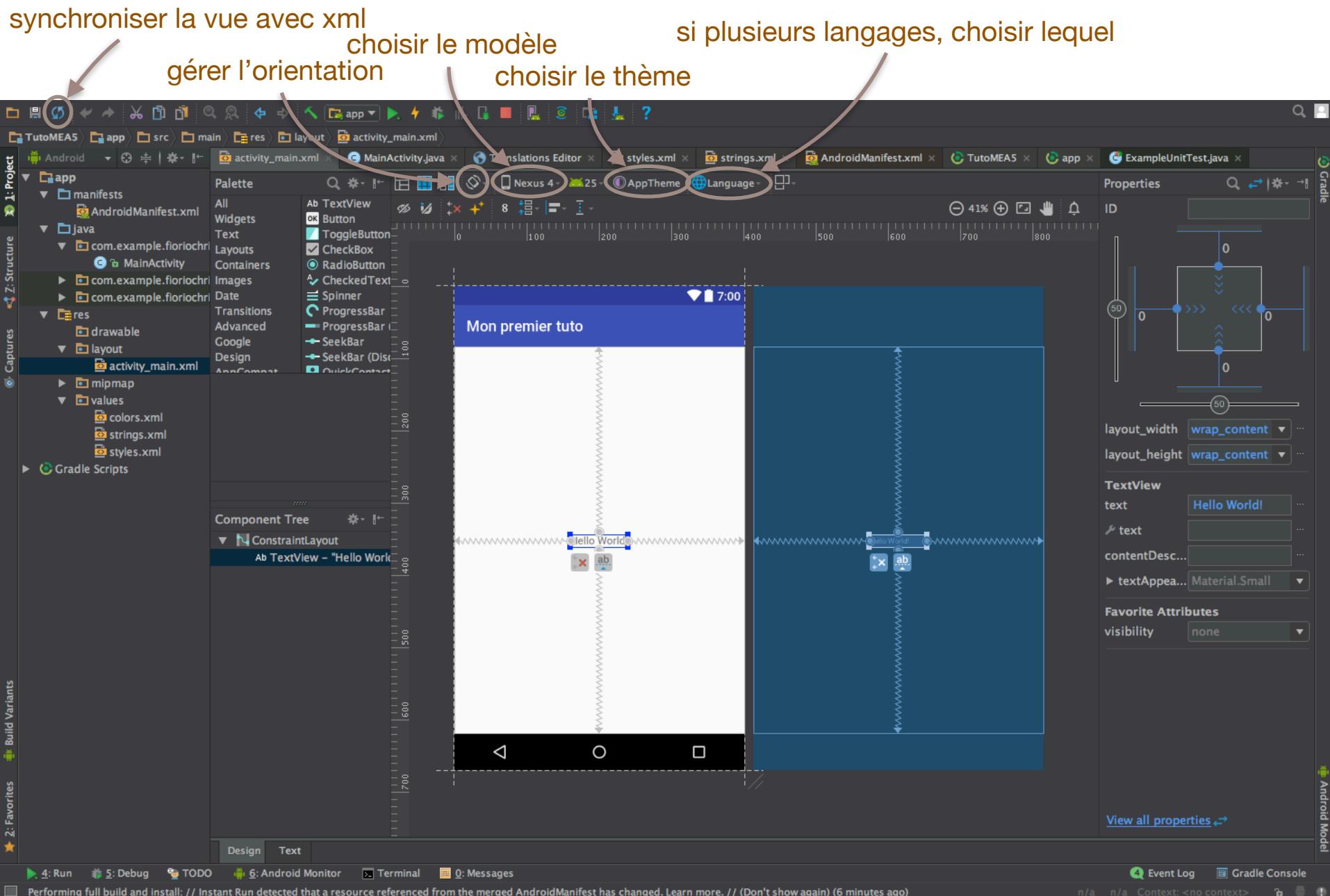


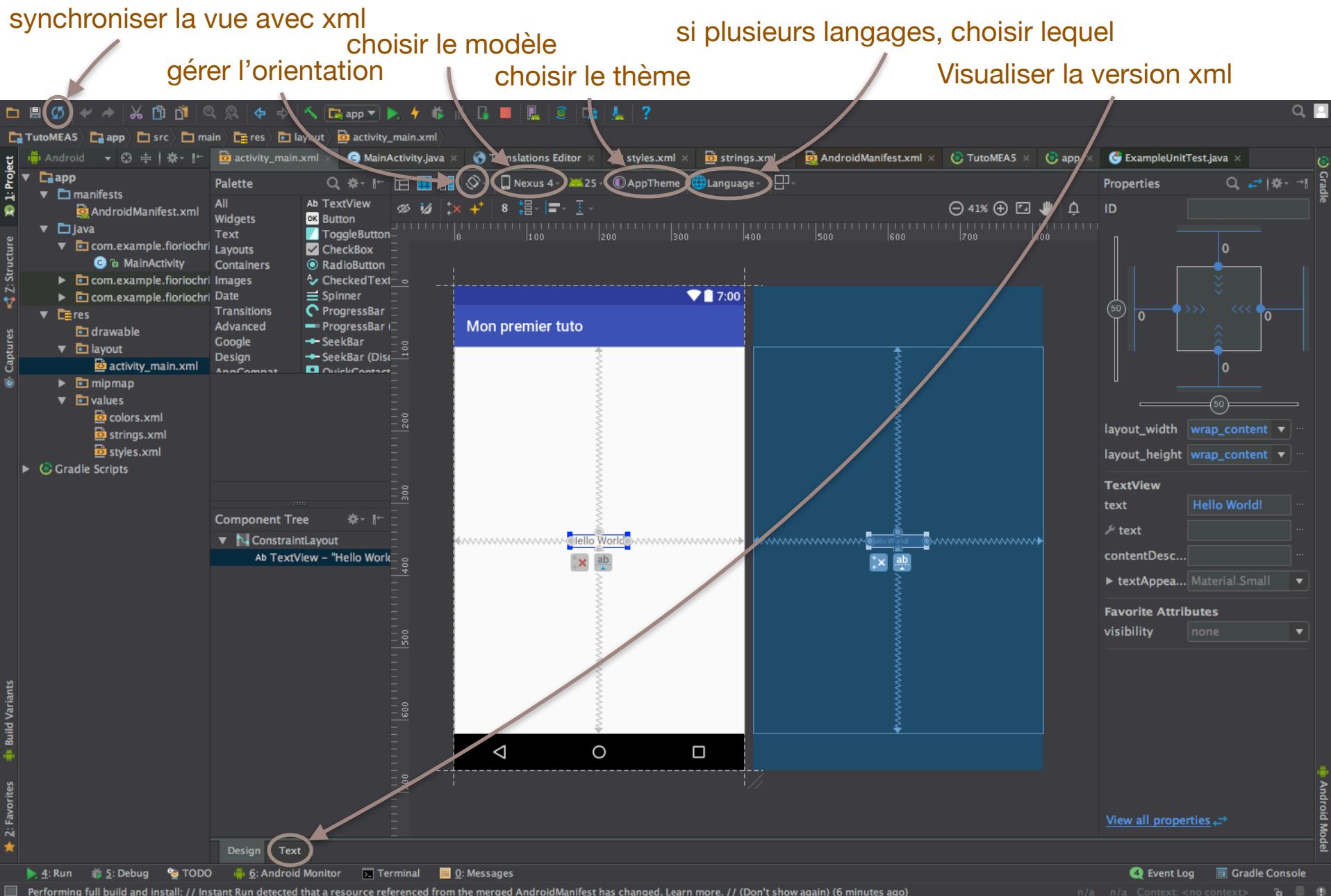
[View all properties ↗](#)

choisir le modèle
gérer l'orientation

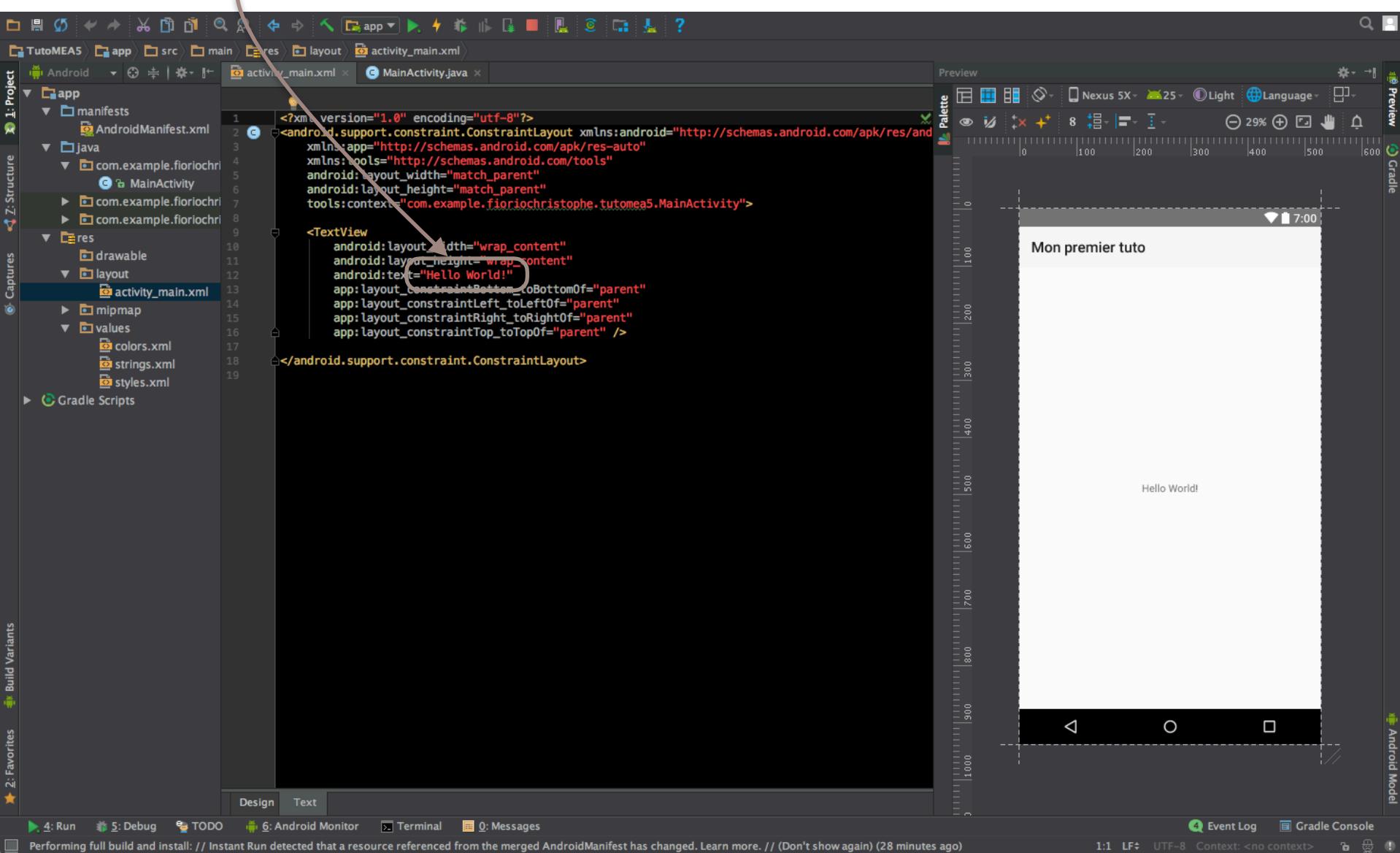
si plusieurs langages, choisir lequel
choisir le thème





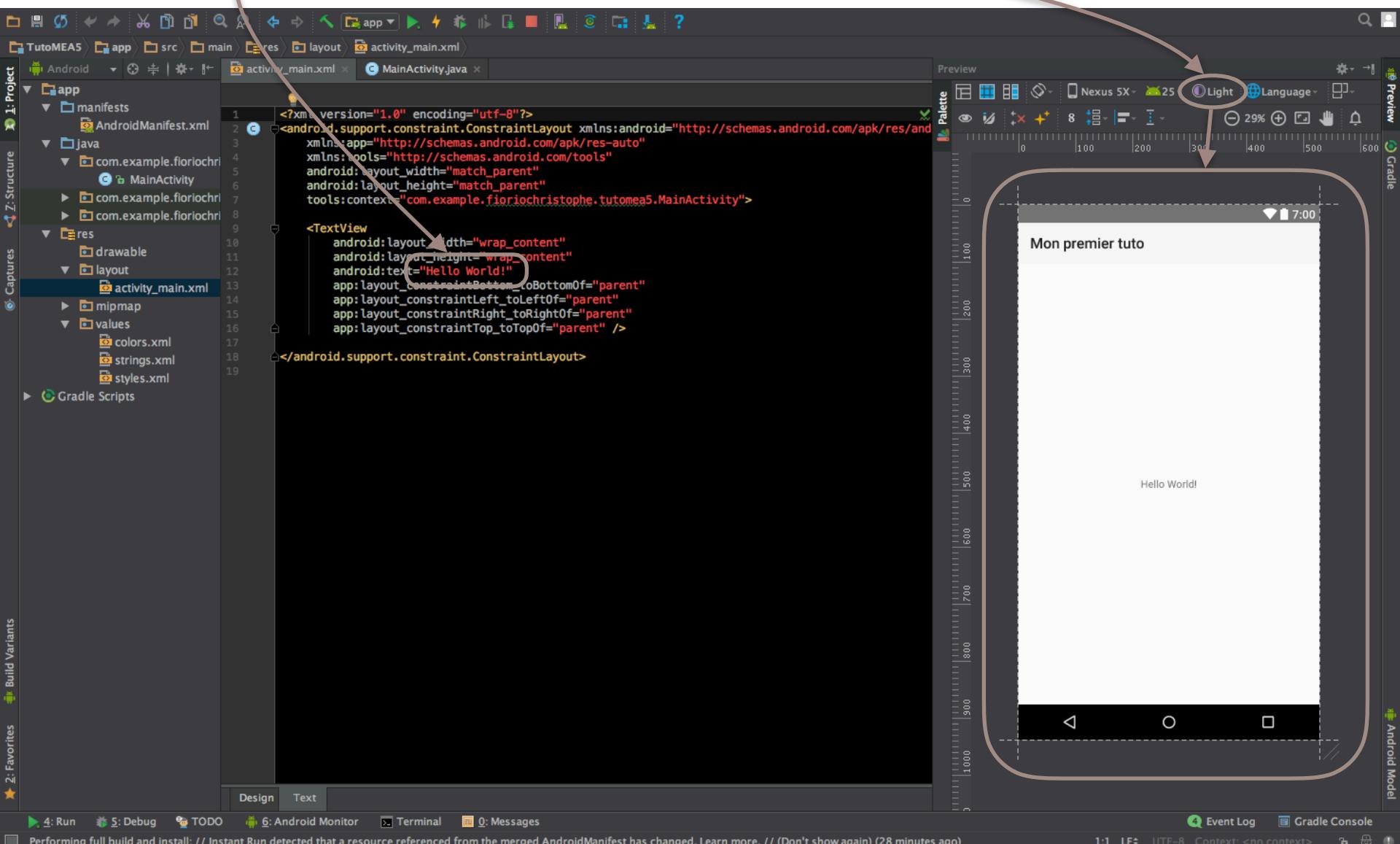


changez le texte en « waiting ! »



changez le texte en « waiting ! »

Le thème a été changé en light material



Utilisation de la ressource Layout

104



Modifier à nouveau le code en revenant à la valeur par défaut

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    //super.onCreate(savedInstanceState);  
    //TextView text = new TextView(this);  
    //text.setText(" Bonjour, vous me devez 1 000 000€");  
    //setContentView(text);  
    setContentView(R.layout.activity_main);  
}
```

et lancez l'application...

Utilisation de

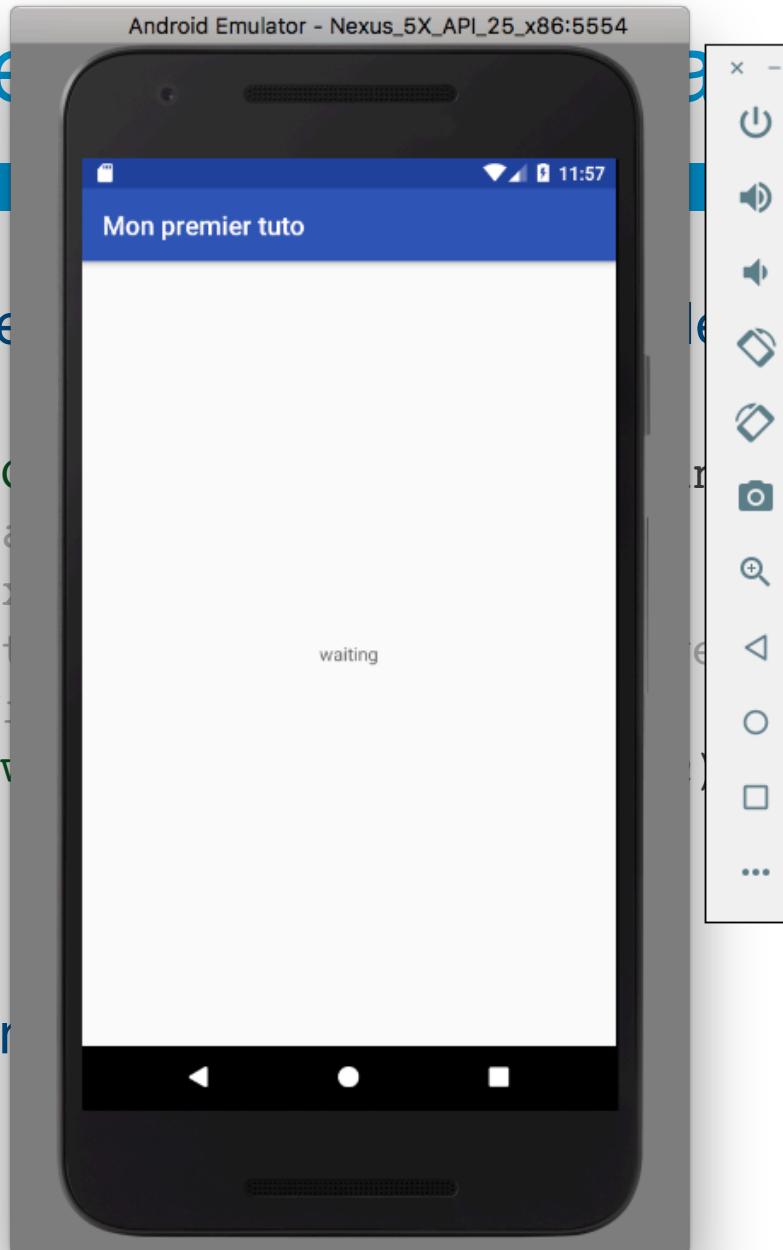
104



Modifier à nouveau le

```
@Override  
protected void onCreate(  
    //super.onCreate(  
    //TextView text  
    //text.setText(  
    //setContentVi  
setContentView(  
}  
}
```

et lancez l'application



out

bar défaut

state) {

000 000€");

Changer la couleur de fond

105



The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TutoMEAS". The "app" module contains "AndroidManifest.xml", "src" (with Java packages "com.example.fioriochristophe" and "MainActivity"), "res" (with drawable, layout, mipmap, and values folders), and "Gradle Scripts".
- Code View:** The "activity_main.xml" file is open, showing XML code for a ConstraintLayout. It includes a TextView with the text "waiting" and constraints relative to the parent.
- Design View:** The "Design" tab is selected in the bottom navigation bar. The preview shows a green background with a white text view containing the word "waiting".
- Preview Panel:** The right side of the screen displays the "Preview" panel, which shows a Nexus 5X device with a light gray background and a green content area. The text "Mon premier tuto" is visible at the top.
- Bottom Navigation:** The navigation bar includes icons for Run (4), Debug (5), TODO, Android Monitor, Terminal, and Messages.
- Bottom Status Bar:** The status bar shows "Gradle build finished in 3s 218ms (20 minutes ago)", "Event Log", "Gradle Console", and system status like battery level and signal strength.

Changer la couleur de fond

Insérez cette ligne de code

105



Insérez cette ligne de code

The screenshot shows the Android Studio interface. On the left, the Project structure is visible, including files like `AndroidManifest.xml`, `MainActivity.java`, and `activity_main.xml`. The `activity_main.xml` file is open in the code editor, showing XML code for a ConstraintLayout. A red circle highlights the line `android:background="#8bc046"`. The `Design` tab is selected at the bottom of the editor. To the right, the `Preview` window shows a green-themed mobile application interface with a central text view containing the text "Mon premier tuto". The bottom of the screen shows the Android navigation bar.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#8bc046"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="waiting"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

105

1: Project

2: Structure

Captures

Gradle Scripts

4: Run 5: Debug TODO 6: Android Monitor Terminal 7: Messages

Gradle build finished in 3s 218ms (20 minutes ago)

Event Log Gradle Console

20:1 LF+ UTF-8 Context: <no context>

Changer la couleur de fond

Insérez cette ligne de code

105



The screenshot shows the Android Studio interface. On the left, the Project structure is visible, with the colors.xml file selected. In the center, the XML code for activity_main.xml is displayed:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#8cb046"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="waiting"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

A green square icon is positioned next to the line `android:background="#8cb046"`. A callout arrow points from the text "Une fois fait, vous pouvez éditer la couleur en cliquant ici" to this icon. Another callout arrow points from the text "Insérez cette ligne de code" to the same line of code in the XML.

On the right, the Preview window shows a green screen with the text "Mon premier tuto" at the top and a small "waiting" text view in the bottom-left corner.

At the bottom, the status bar shows "Gradle build finished in 3s 218ms (20 minutes ago)" and the time "20:1".

Positionnement des widgets

106



The screenshot shows the Android Studio interface with the following details:

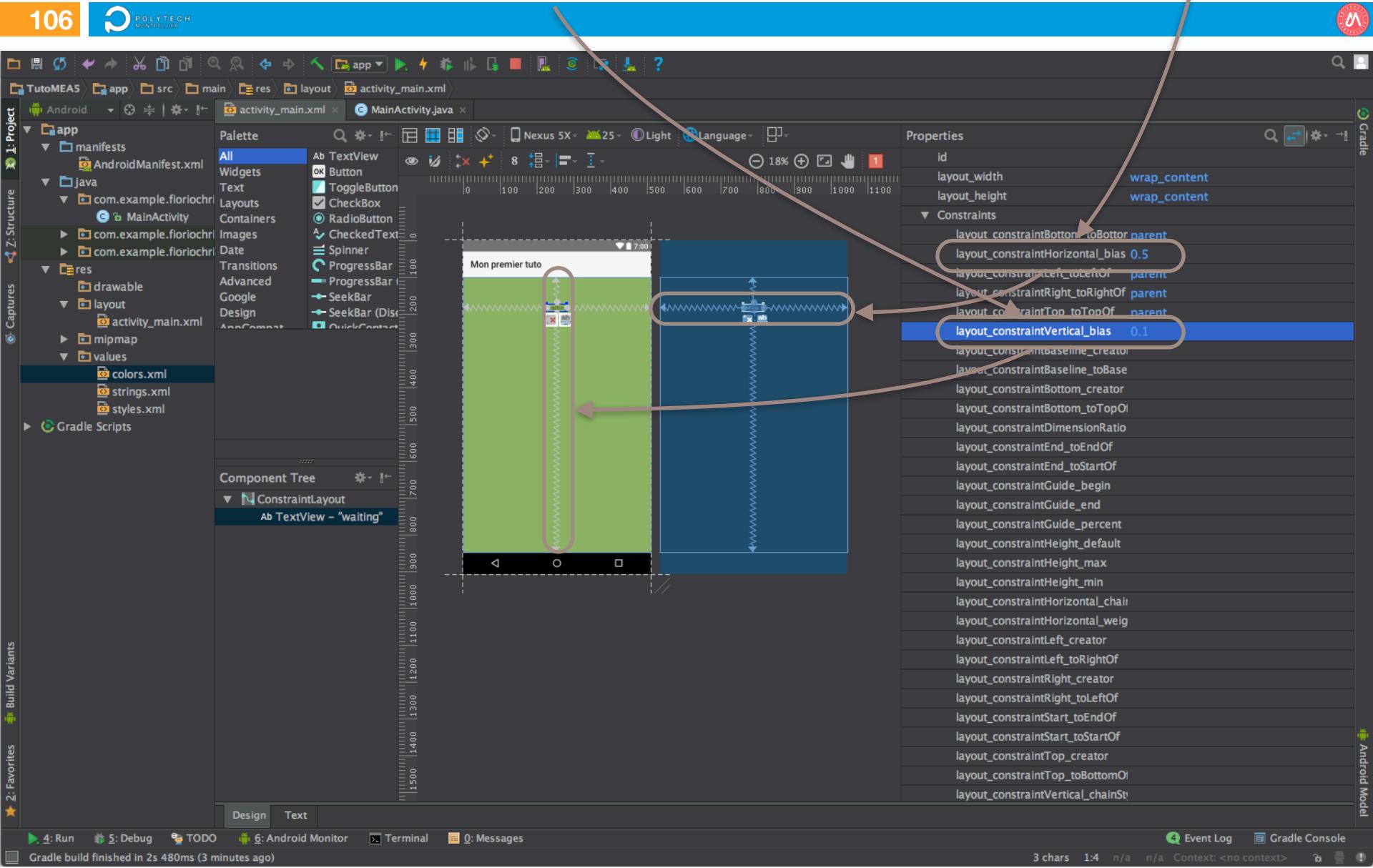
- Project Structure:** The left sidebar shows the project structure under "app".
- Main Activity:** The "activity_main.xml" file is open in the center.
- Design View:** The main workspace displays a ConstraintLayout containing a single TextView with the text "Mon premier tuto".
- Properties Panel:** On the right, the "Properties" panel is open, showing various layout parameters. The "layout_constraintVertical_bias" parameter is currently selected and highlighted in blue.
- Component Tree:** Below the design view, the "Component Tree" shows the hierarchy: ConstraintLayout > TextView.
- Bottom Navigation:** The bottom bar includes tabs for "Design" and "Text", along with icons for Run, Debug, TODO, Android Monitor, Terminal, and Messages.
- Bottom Status:** The status bar at the bottom indicates "Gradle build finished in 2s 480ms (3 minutes ago)".

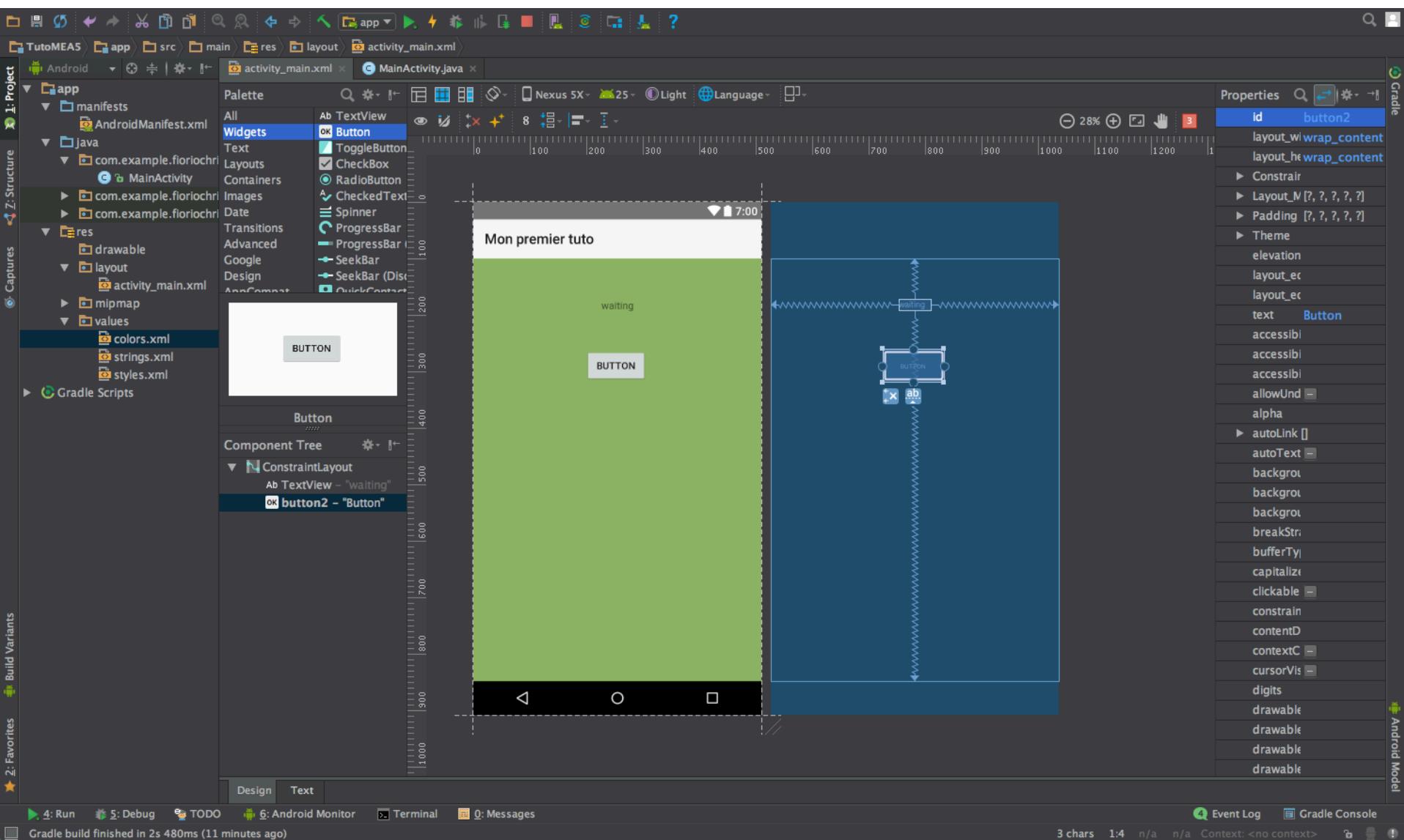
Positionnement des widgets

Placer le texte 10% vers le haut

centerer le texte

106

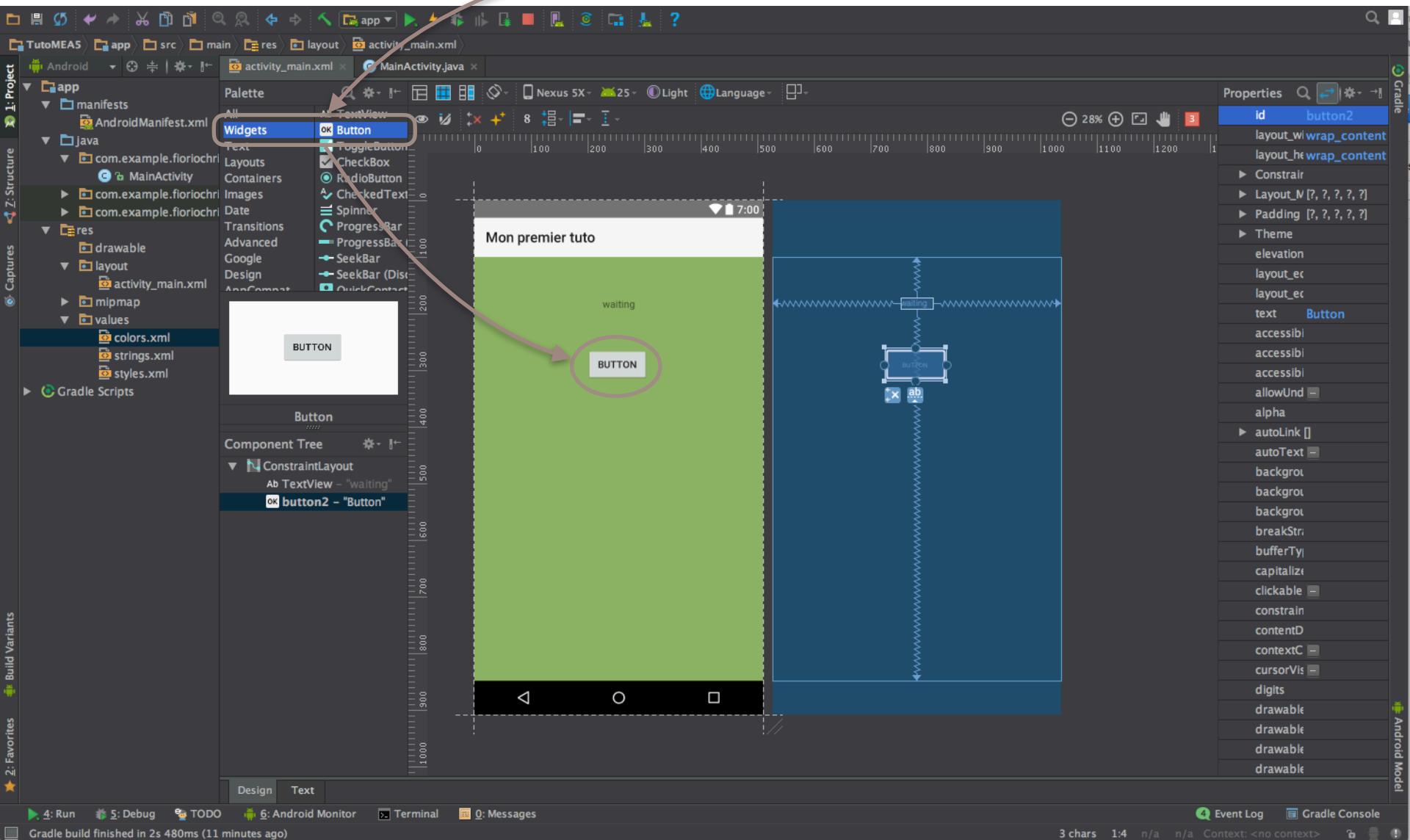




107

Positionnement des widgets

Ajoutez un bouton



The screenshot shows the Android Studio interface with the following details:

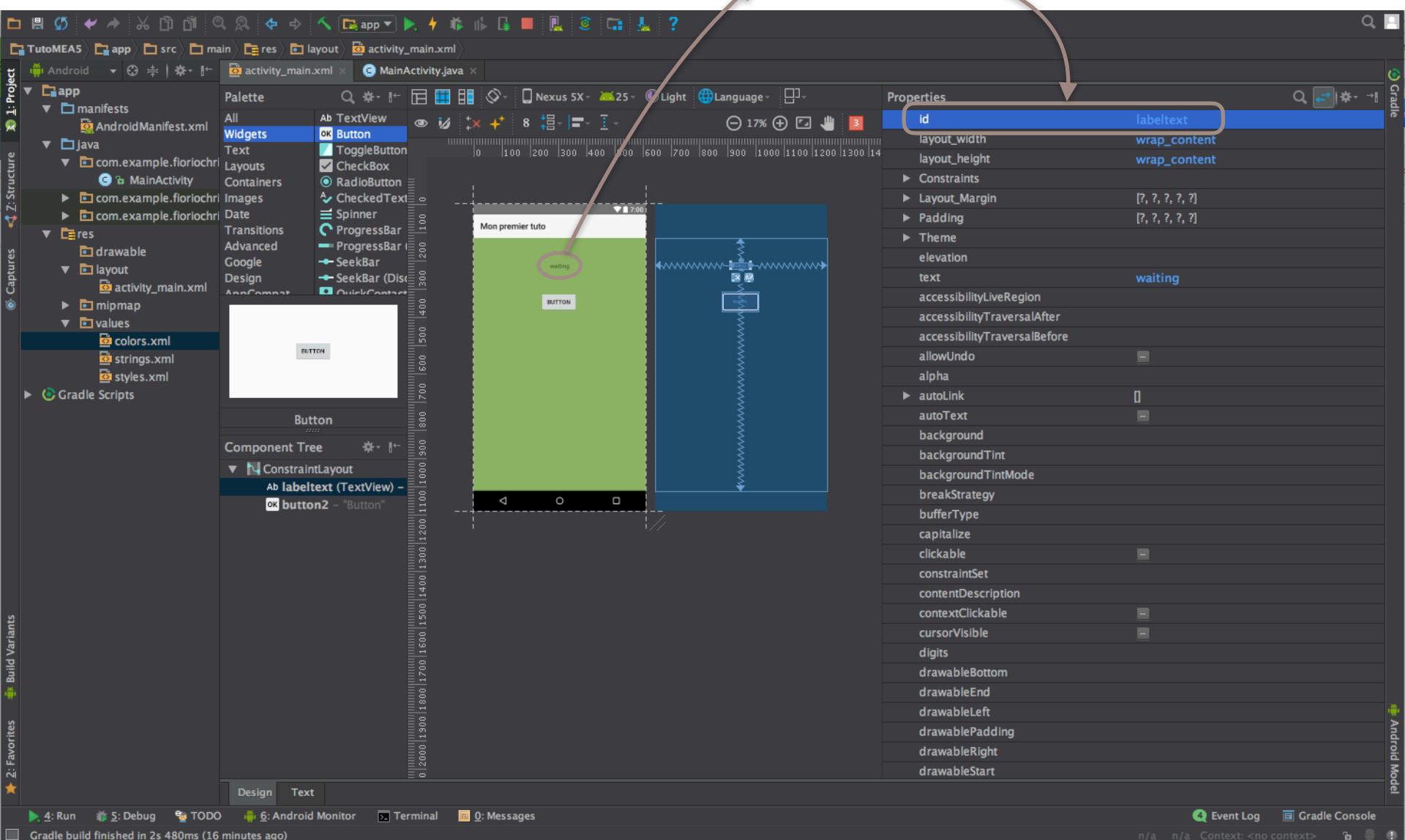
- Project:** TutoMEAS
- Activity:** activity_main.xml
- Editor:** Layout Editor (Design tab)
- Widget:** Button
- Properties:** id: button2, labeltext: waiting
- Component Tree:** Ab labeltext (TextView) - button2 - "Button"



108

Positionnement des widgets

Ajoutez un id au texte



The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TutoMEAS". The "app" module contains "AndroidManifest.xml", "MainActivity.java", and several resource files like "activity_main.xml", "colors.xml", "strings.xml", and "styles.xml".
- Layout Editor:** The "activity_main.xml" layout is being edited. It features a ConstraintLayout with a green `TextView` containing the text "Mon premier tuto" and a white `Button` containing the text "BUTTON". The button is positioned at the bottom right of the screen.
- Properties Panel:** The "Properties" panel on the right lists numerous constraint-related properties, such as `layout_constraintTop_toBottomOf`, `layout_constraintBottom_toTopOf`, and `layout_constraintWidth_default`.
- Bottom Status Bar:** The status bar displays the message "Gradle build finished in 2s 480ms (16 minutes ago)".

Positionnez le bouton par rapport au texte

The screenshot shows the Android Studio interface with the following components:

- Project Bar:** Shows the project structure with files like `AndroidManifest.xml`, `MainActivity.java`, and various resource files.
- Layout Editor:** Displays the XML layout file `activity_main.xml`. It contains a green `ConstraintLayout` containing a `TextView` with the text "Mon premier tuto" and a `Button` with the text "BOUTON".
- Properties Panel:** Shows the properties for the selected button. A red arrow points from the text "Positionnez le bouton par rapport au texte" to the `layout_constraintTop_toBottomOf` constraint in the properties list.
- Component Tree:** Shows the hierarchy of the layout, starting with `ConstraintLayout`, then `labeltext (TextView)`, and finally `button2 - "Button"`.
- Bottom Bar:** Includes tabs for `Design` and `Text`, and icons for `Run`, `Debug`, `TODO`, `Android Monitor`, `Terminal`, and `Messages`.

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TutoMEAS". The `activity_main.xml` file is selected.
- Palette:** The "Widgets" category is selected, showing various UI components like `TextView`, `Button`, `ToggleButton`, etc.
- Design View:** The main workspace displays the layout XML. It features a green `ConstraintLayout` with a white `TextView` at the top containing the text "Mon premier tuto". Below it are two white `Button` components. The layout uses constraint-based positioning.
- Properties Panel:** The "Properties" panel on the right lists numerous layout parameters. The `layout_marginTop` parameter is highlighted and set to `20dp`.
- Bottom Bar:** The toolbar includes icons for Run, Debug, TODO, Android Monitor, Terminal, and Messages, along with a message indicating a Gradle build finished in 2s 480ms.

Rajoutez une marge pour les séparer

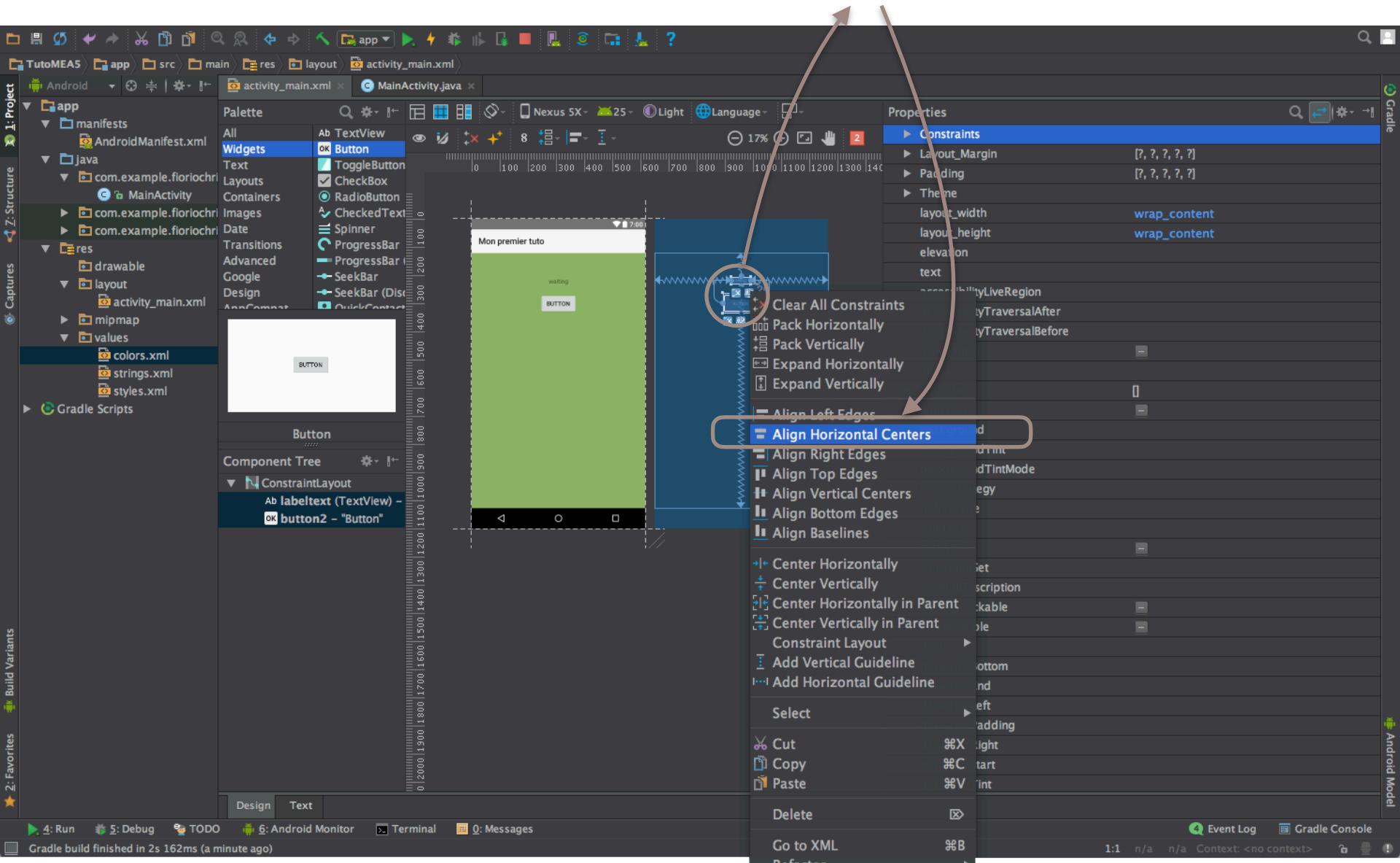
The screenshot shows the Android Studio interface with the following components:

- Project Bar:** Shows the project structure with files like `AndroidManifest.xml`, `MainActivity.java`, and `activity_main.xml`.
- Palette:** Displays various UI components under the **Widgets** category.
- Layout Editor:** Shows the `activity_main.xml` layout. It contains a green `ConstraintLayout` containing a `labeltext` (Text View) and a `button2` (Button). A blue arrow points from the text "Rajoutez une marge pour les séparer" to the `button2` component.
- Component Tree:** Shows the hierarchy of the layout.
- Properties Panel:** Shows the properties for the `button2` component, specifically focusing on the `layout_marginTop` property set to `20dp`.
- Bottom Bar:** Includes tabs for `Design` and `Text`, and icons for Run, Debug, and Terminal.

The screenshot shows the Android Studio interface with the following details:

- Project:** TutoMEAS
- Activity:** activity_main.xml
- Palette:** Widgets (selected), showing Button, TextView, ToggleButton, CheckBox, RadioButton, CheckedTextView, Spinner, ProgressBar, ProgressWheel, SeekBar, and SeekBar (Design).
- Properties:** Constraints section is open, showing options like Layout_Margin, Padding, Theme, layout_width (wrap_content), layout_height (wrap_content), elevation, and text.
- Component Tree:** Shows a ConstraintLayout containing a TextView labeled "labeltext" and a Button labeled "button2 - "Button"".
- Design View:** Displays a green rectangular view with a white button inside labeled "BUTTON". The button has a constraint labeled "waiting" attached to its top edge.
- Bottom Bar:** Includes icons for Run (4), Debug (5), TODO, Android Monitor, Terminal, Messages, and a status message: "Gradle build finished in 2s 162ms (a minute ago)".

Sélectionnez les 2 widget, puis bouton droit et « align horizontal center »

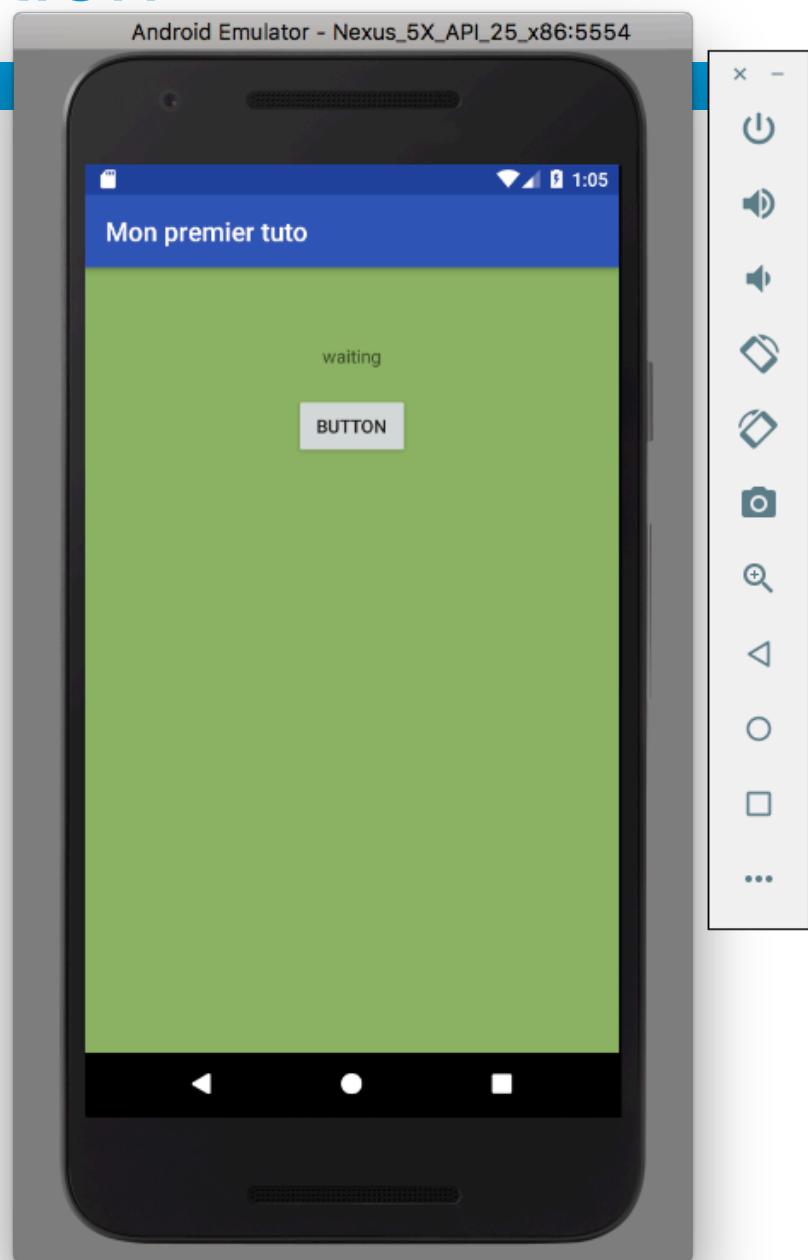


Testez votre application

112



Notre interface apparaît...



Créer une action pour le bouton

113



- Pour accéder aux différentes widget, il faut en fait accéder aux ressources correspondantes :

- par la valeur statique en passant par la classe R,

```
R.string.app_main
```

- en récupérant un objet grâce à `findViewById`

```
findViewById(R.id.button1)
```

- Pour créer une action au bouton, il faut, après l'avoir récupérer, ajouter un listener au bouton

```
import android.view.View;
import android.widget.TextView;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView textLabel = (TextView) findViewById(R.id.labeltext);
        final Button button = (Button) findViewById(R.id.button1);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                textLabel.setText(R.string.app_name);
            }
        });
    }
}
```



Gestion d'un textfield

115



Il faut

- rajouter un textfield ;
- s'assurer qu'il gère l'action *done* ;
- comme pour le bouton, il faut lui rajouter un listener ;

The screenshot shows the Android Studio interface with the following details:

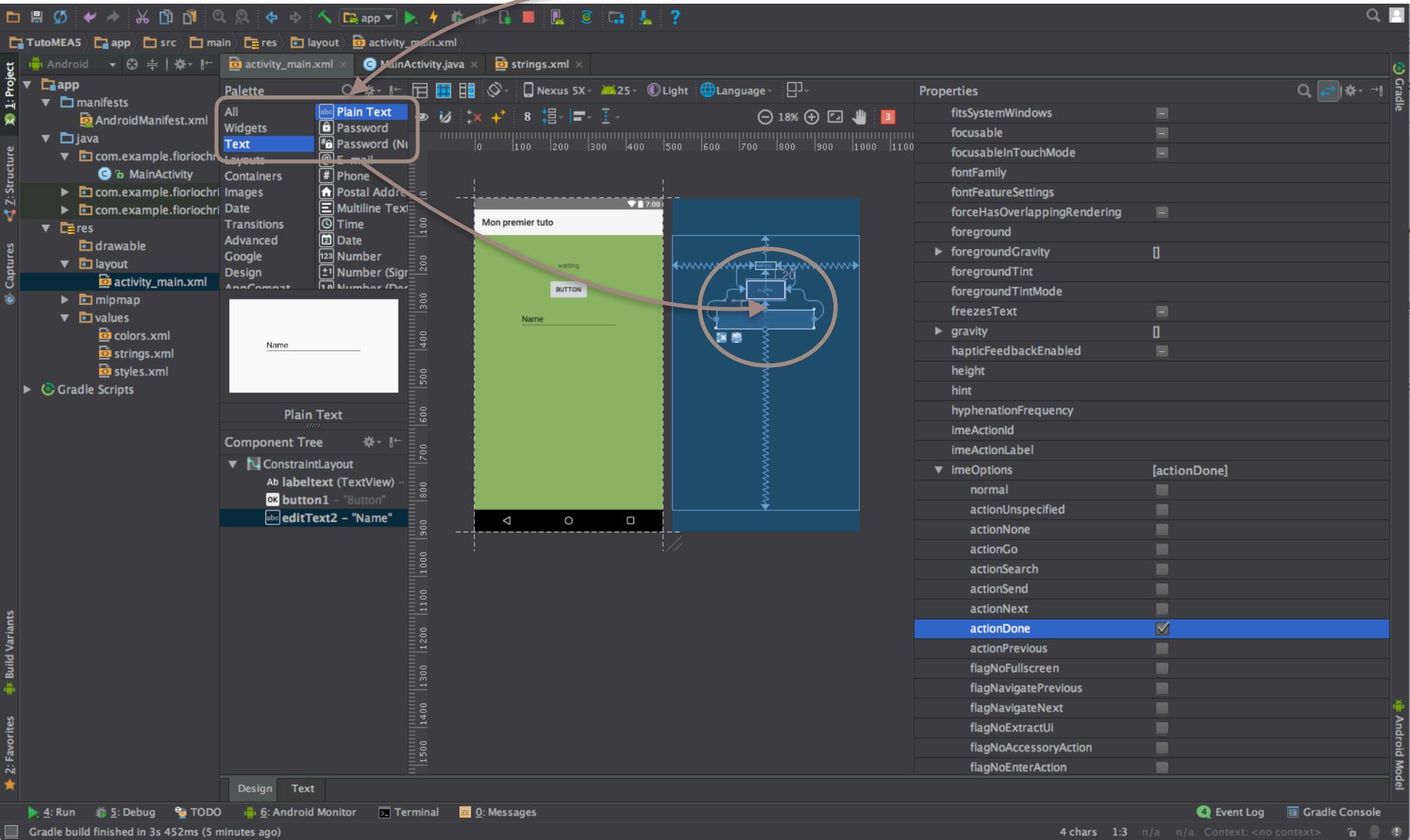
- Project Structure:** The project is named "TutoMEAS" and the active file is "activity_main.xml".
- Toolbars:** Standard Android Studio toolbars for Run, Debug, TODO, Android Monitor, Terminal, and Messages.
- Side Panels:** "Captures" and "Build Variants" panels are visible on the left.
- Main Editor Area:**
 - Design Tab:** Shows a green-themed layout preview with a title bar "Mon premier tuto", a "waiting" button, and an "EditText" with placeholder "Name".
 - Text Tab:** Shows the XML code for the ConstraintLayout, specifically targeting the "editText2" element.
- Properties Panel:** Lists various properties for the selected "editText2" component, including "imeOptions" which is currently set to "[actionDone]" with the "actionDone" checkbox checked.
- Bottom Bar:** Includes buttons for Run, Debug, and a terminal, along with status information like "4 chars 1:3 n/a n/a Context: <no context>" and links to Event Log and Gradle Console.



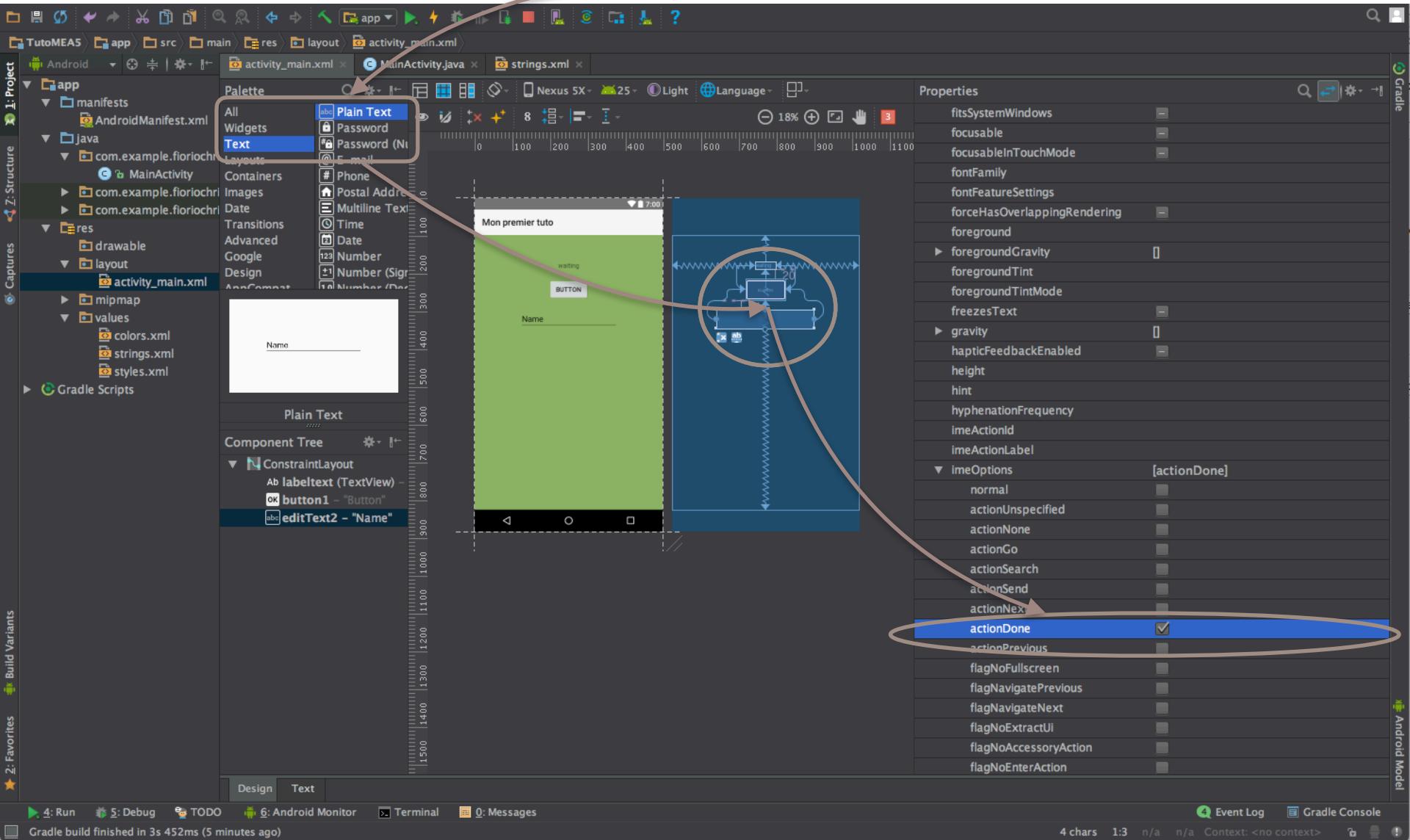
116

Gestion d'un textfield

Ajoutez un textfield



Ajoutez un textfield



```
final EditText textfield = (EditText)
findViewById(R.id.editText1);
textfield.setOnEditorActionListener(new
TextView.OnEditorActionListener(){
    @Override
    public boolean onEditorAction(TextView textfield, int
editInfo, KeyEvent keyevent){
        if (editInfo== EditorInfo.IME_ACTION_DONE){
            String text = textfield.getText().toString();
            textLabel.setText(text);
            textfield.setText(" ");
        }
        return false;
    }
});
```

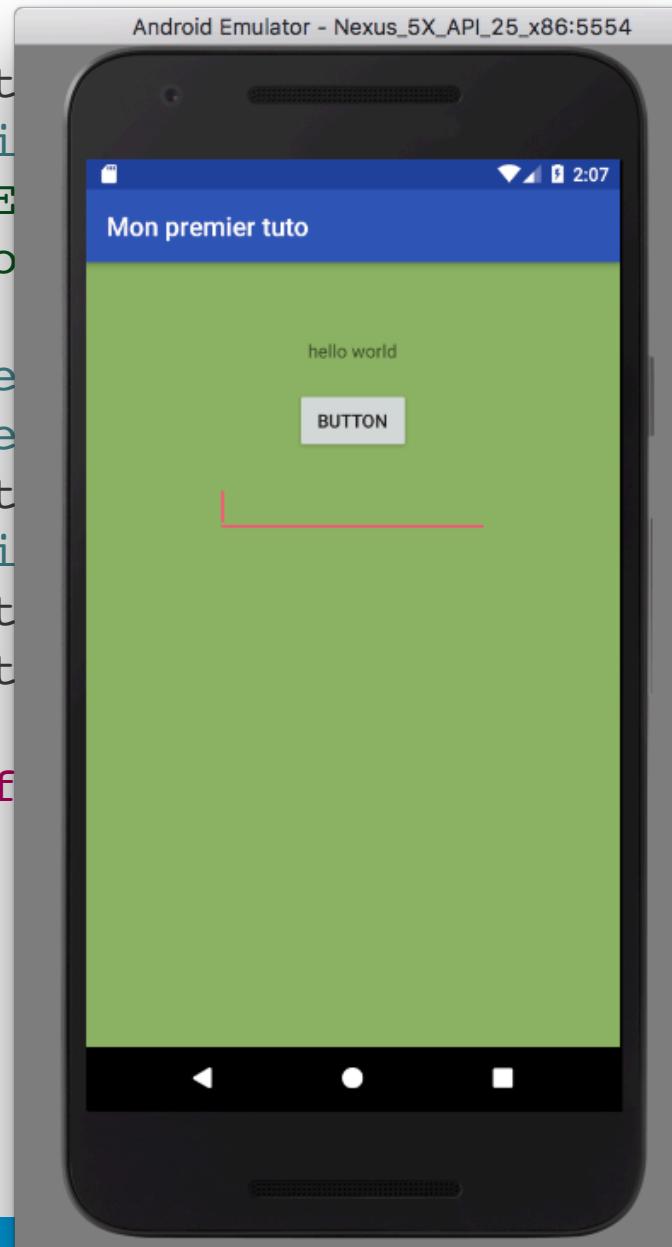


```
final EditText textfield = (EditText)
findViewById(R.id.editText1);
textfield.setOnEditorActionListener(new
TextView.OnEditorActionListener(){
    @Override
    public boolean onEditorAction(TextView textfield, int
editInfo, KeyEvent keyevent){
        if (editInfo== EditorInfo.IME_ACTION_DONE){
            String text = textfield.getText().toString();
            textLabel.setText(text);
            textfield.setText(" ");
        }
        return false;
    }
});
```

Pour rendre la main après la validation
équivalent au resignFirstResponder()
sous IOS



```
final EditText t  
findViewById(R.i  
textfield.setOnE  
TextView.OnEdito  
    @Override  
    public boolean  
editInfo, KeyEvent  
        if (edit  
            Stri  
            text  
            text  
        }  
        return f  
    }  
});
```



```
x -  
power  
volume  
rotation  
camera  
zoom  
navigation  
...  
a validation  
responder()
```

```
textfield, int  
V_DONE) {  
    () .toString();
```