TD2 - Interface utilisateur

### But du TD: Découverte des ressources, layouts et vues

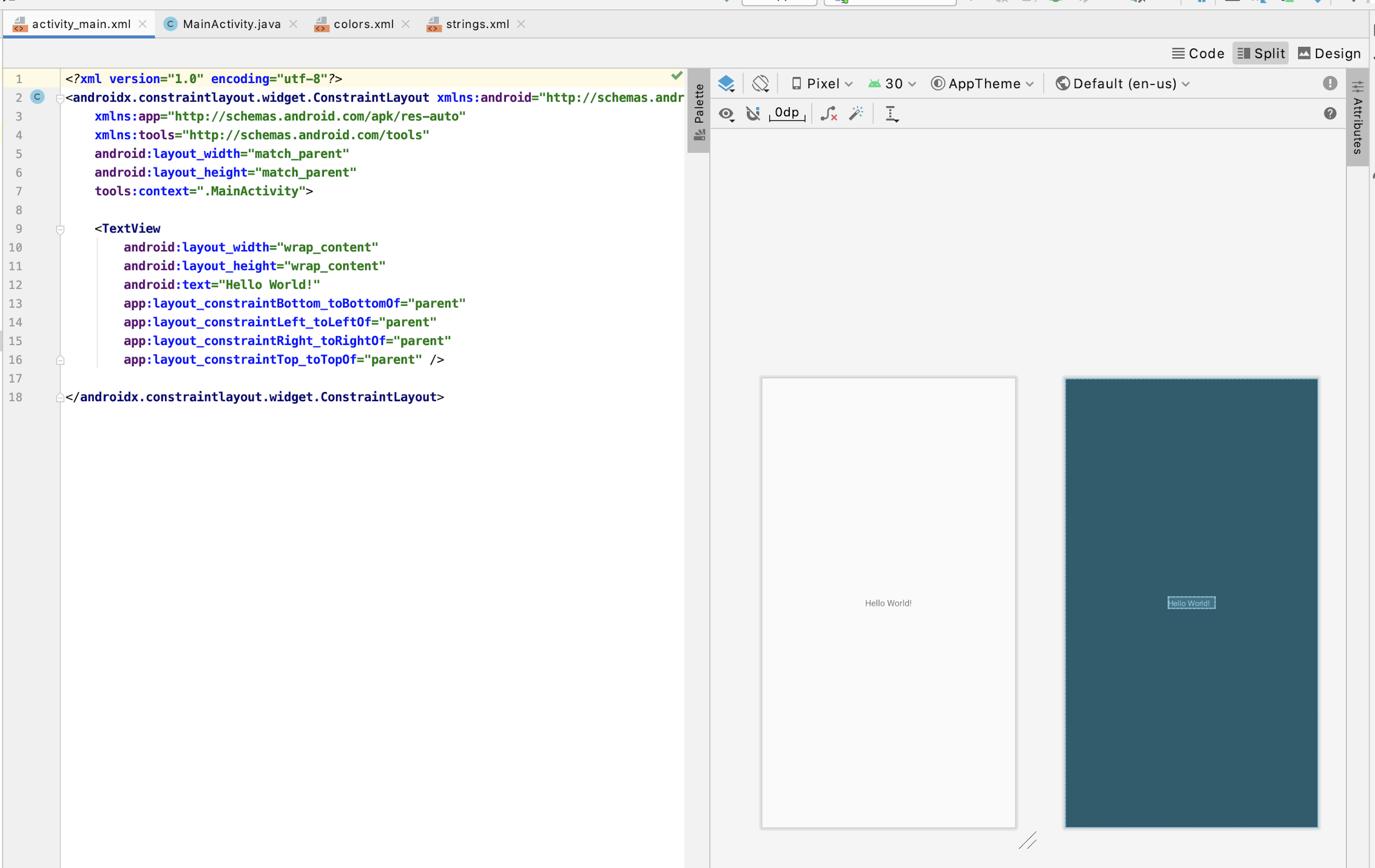
On va partir d’un projet minimal, type « Empty Activity », nommez-le TD2

1. Ressources

Ce sont les fichiers XML qui sont dans projet/src/main/res.

* res/layout : ce sont les définitions d’écrans des activités, par exemple activity\_main.xml. Les balises XML et leurs attributs définissent les éléments affichés : boutons, zones de saisie, etc. Dans les classes Java, ce layout sera désigné par R.layout.activity\_main.
* res/values :
  + Le fichier strings.xml contient les textes de votre application : labels, messages, etc. Ces textes sont identifiés par l’attribut name et référencés dans les layouts par @string/nom, et par R.string.nom dans les sources Java. Si vous voulez traduire ces textes en allemand, faites une copie du dossier values en values-de et traduisez tous les textes sans changer les attributs name.
  + Le fichier color.xml contient les couleurs en hexadécimal de votre application. Identifiés par l’attribut name et référencés dans les layouts par @color/nom et par R.color.nom dans les sources Java.
  + Le fichier style.xml contient les styles de votre application. Identifiés par l’attribut name et référencés dans les layouts par @style/nom et par R.style.nom dans les sources Java.
* res/mipmap-\*dpi et res/drawable-\*dpi : vous y trouverez des icônes de différentes tailles. Chaque image PNG devient une ressource identifiée par R.drawable.son\_nom en Java et @drawable/son\_nom dans un layout. C’est le système qui choisit automatiquement, selon la densité de votre écran, celle qu’il faut afficher.

Les fichiers XML peuvent être affichés soit sous forme d’un formulaire ou de manière graphique, soit sous forme XML brute. Voyez l’onglet juste en haut de l’éditeur, ex: Design, Code ou Split → hybride entre Code et design.



Actuellement, dans layout/activity\_main.xml, il y a un message en dur dans android:text. C’est signalé par un avertissement dans Android Studio, car normalement, il faut placer tous les messages dans res/values/strings.xml, afin qu’ils puissent être facilement traduits dans d’autres langues. Alors, dans ce dernier fichier, créez un nouvel élément “Salut tout le monde” puis référencez cette chaîne dans le layout avec android:text="@string/message". Constatez que le message a été pris en compte. Pour certains layouts, il faut identifier les vues. Ça consiste à rajouter un attribut android:id="@+id/nom

| <TextView  android:id="@+id/texte" ... /> |
| --- |

Ensuite, on peut référencer cet identifiant avec une syntaxe subtilement différente, sans le +:

| <Button  android:layout\_below="@id/texte" ... /> |
| --- |

Android contient plusieurs dizaines de composants d’interface :

* des vues : TextView, Button, EditText…
* des groupes : LinearLayout, RelativeLayout, TableLayout, ConstraintLayout…([Layouts](https://developer.android.com/guide/topics/ui/declaring-layout.html))

Vous pouvez retrouver tous ces derniers sur la doc officiel → <https://developer.android.com/guide/topics/ui>

Dites vous bien que nous ne pouvons pas tous tester, ni tout connaître, ni tout aborder dans ce cours. Aidez vous des documentations officielles pour aller plus loin ou pour utiliser des nouveaux composants.

1. Les groupements de vues (ViewGroup)

On va s’intéresser aux vues qui permettent d’arranger d’autres vues. Elles héritent de la classe ViewGroup. Pour simplifier, elles ont des vues enfants et décident de leur placement, taille et position, en fonction de propriétés présentes sur les enfants. Voici les quatre à connaître :

* LinearLayout pour former des lignes ou des colonnes,
* TableLayout pour disposer en tableau,
* RelativeLayout pour mises en page statiques plus complexes,
* ConstraintLayout pour des dispositions dynamiques complexes.

1. Mise en page avec des LinearLayout

C’est le ViewGroup le plus simple et le plus facile à employer. Copier coller ce code dans un le fichier XML de l’activité principale :

| <?xml version="1.0" encoding="utf-8"?> <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:orientation="vertical"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <Button  android:text="OK"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"/>  <Button  android:text="Annuler"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"/>  <Button  android:text="Annuler tout"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"/> </LinearLayout> |
| --- |

Examinez la mise en page en comparant avec les propriétés des vues, puis faites les changements suivants :

* Mettez match\_parent pour android:layout\_width de l’un des boutons, puis annulez le changement.
* Mettez match\_parent pour android:layout\_height du deuxième bouton, regardez son effet, puis annulez.
* Mettez 200dp puis 300dp pour android:layout\_height du deuxième bouton, regardez son effet, puis annulez.
* Changez android:orientation en horizontal pour le LinearLayout, laissez ce changement. Passez en mode paysage si les vues sont trop encombrantes.

On voit que les vues ont une largeur dépendant de leur contenu. On voudrait qu’elles aient la même largeur.

* Ajoutez android:layout\_weight="1" pour chacune. C’est pas très visible, mais la largeur n’est pas exactement la même.
* Changez android:layout\_width="0dp" pour les trois vues. En principe, c’est le meilleur résultat. La largeur nulle combinée avec un poids non nul fait faire un calcul correct.
* Changez android:layout\_width="wrap\_content" pour le LinearLayout puis annulez ce changement.

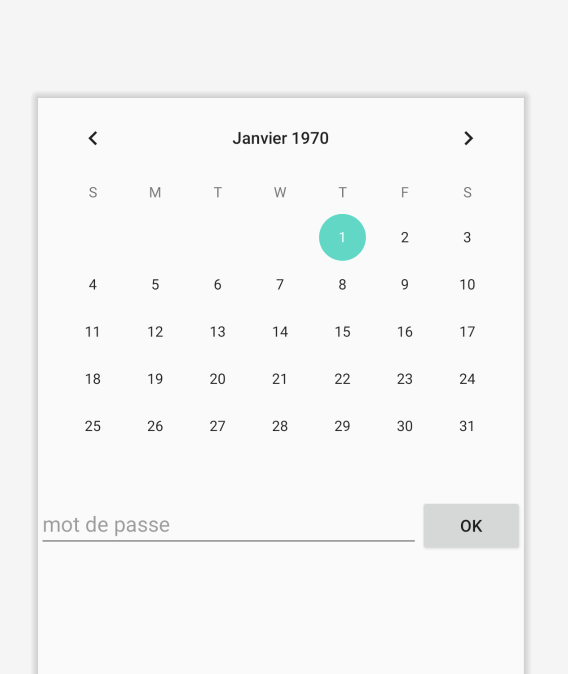
Maintenant, à vous :



Copiez le layout et appelez le password

* Supprimez les 2e et 3e boutons, ne laissez que le OK. Ajoutez un EditText devant le bouton OK. Remplacez la propriété android:text par android:hint et mettez la chaîne "mot de passe".
* Modifiez les largeurs pour obtenir le résultat demandé. C’est à dire que l’EditText prend presque toute la place en largeur, tandis que le bouton OK est le plus petit possible. Pour cela, il faut mettre un poids nul(layout\_weight) au bouton OK mais aussi lui donner une taille minimale(wrap\_content).

Dans un cas général, on est amenés à combiner des LinearLayout horizontaux et verticaux. Voici un exemple.

* Copiez password.xml et appelez-le calendar.xml
* Faites en sorte d’ajouter un CalendarView au dessus des deux vues EditText et Button pour obtenir l’image ci-dessous. Pour y arriver, vous devrez englober le LinearLayout horizontal du bas dans un LinearLayout vertical, et en jouant sur les poids et hauteurs pour garantir une visibilité et une taille à tout le monde :

1. Mise en page avec un RelativeLayout

Ce ViewGroup permet une mise en page précise et complexe, mais par contre, il est difficile d’emploi. Le principe général est de positionner une vue enfant par rapport soit aux bords du RelativeLayout, soit à des vues précédemment posées. Par exemple : « coller le bord gauche de telle vue au bord droit de telle autre ». Voici un exemple, il est très artificiel, destiné à montrer les propriétés de placement.

| <?xml version="1.0" encoding="utf-8"?> <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <Button  android:id="@+id/button1" android:text="BTN1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_centerHorizontal="true"  android:layout\_alignParentTop="true"/>  <Button  android:id="@+id/button2" android:text="BTN2"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_centerVertical="true"  android:layout\_toRightOf="@id/button1"/>  <Button  android:id="@+id/button3" android:text="BTN3"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@id/button2"  android:layout\_alignLeft="@id/button1"  android:layout\_alignRight="@id/button1"/> </RelativeLayout> |
| --- |

Dans un RelativeLayout les vues enfant sont positionnées et dimensionnées par quatre contraintes, une par côté, haut, bas, gauche, droite. Les côtés droit et gauche sont aussi désignés par start et end pour pouvoir tenir compte des régionalisations (écriture de droite à gauche, etc.). Il y a des avertissements dans le layout précédent, il faudrait par exemple rajouter des propriétés comme toEndOf quand il y a toRightOf. Il y a deux catégories de contraintes :

* Positionnement par rapport au layout. La valeur de ces contraintes sont de simples booléens et on ne les mentionne que quand ils sont vrais.
* « collé au bord du parent » : layout\_alignParentXXX avec XXX valant Top, Bottom, Left et Right, ainsi que les variantes Start et End.
* « centré dans le parent » : layout\_centerInParent, layout\_centerHorizontal et layout\_centerVertical.
* Positionnement par rapport à une autre vue.

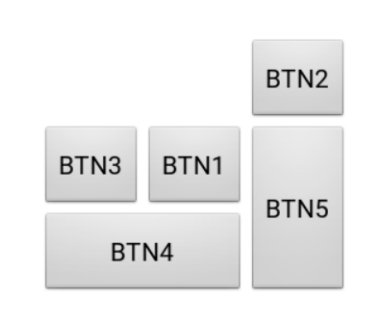
La valeur à mettre dans ces contraintes est la référence à l’identifiant d’une autre vue, @id/autre\_vue. Chaque vue concernée doit donc définir un identifiant.

* « collé au bord opposé » : layout\_toRightOf, layout\_toLeftOf, layout\_above et layout\_below. Par exemple, il faut coller le bord droit de cette vue au bord gauche de l’autre vue.
* « bords alignés » : layout\_alignXXX comme pour alignParent. Cette vue aligne son bord avec celui de l’autre vue.

Pour voir l’effet de ces contraintes, essayez ces changements et comprenez-les :

* dans le bouton1, changez la contrainte alignParentTop en alignParentBottom.
* dans le bouton1, allongez le titre android:text et constatez les répercutions sur le bouton3.
* dans le bouton3, allongez le titre, et constatez ce qui se passe.
* dans le bouton3, changez la valeur alignRight avec bouton2 au lieu de bouton1.

Maintenant, à vous, faites en sorte d’obtenir cette mise en page. Le bouton BTN1 est mis au centre de l’écran. Les boutons sont définis dans l’ordre de leur nom.



1. Mise en page avec un ConstraintLayout

Ce layout fait partie de la bibliothèque support, celle qui permet de créer des applications fonctionnant sur de vieux smartphones. Cela impose une déclaration dans le fichier build.gradle de l’application, ainsi que l’utilisation de plusieurs namespace particuliers.

Il faut écrire :

| **implementation 'androidx.constraintlayout:constraintlayout:2.0.1'** |
| --- |

Ce type de ViewGroup est une extension de RelativeLayout, permettant de mettre à jour la disposition de manière plus dynamique et plus polyvalente. Les attributs pour mettre en place les contraintes ont des noms plus homogènes, mais nettement plus compliqués, obligeant à réfléchir davantage. La forme générale des contraintes est layout\_constraintXXX\_toYYYOf avec XXX et YYY valant Top, Bottom, Left, Right ou Baseline. Cela donne des attributs comme layout\_constraintTop\_toTopOf, layout\_constraintRight\_toLeftOf, etc. Le code Baseline fait aligner sur la base du texte des vues.

Tous ces attributs prennent pour valeur une référence d’identifiant (et non pas un booléen comme certains avec les RelativeLayout). Par exemple, on écrit layout\_alignParentTop="true" dans un RelativeLayout, mais layout\_constraintTop\_toTopOf="@id/id\_du\_layout" dans un ConstraintLayout – dans ce dernier cas, on peut aussi mettre une valeur spéciale pour indiquer le layout : "parent".

Pour positionner une vue à une certaine distance d’une autre, on utilise les attributs layout\_marginXXX. Deux autres contraintes existent : layout\_constraintZZZ\_bias avec ZZZ étant Vertical ou Horizontal. Elles permettent de positionner une vue de manière proportionnellement à l’espace disponible – elles déséquilibrent une contrainte en faveur de l’un ou l’autre côté. La valeur à mettre est un réel entre 0 et 1, par défaut, c’est 0.5. La taille d’une vue peut être minimale wrap\_content, maximale match\_parent ou donnée par les contraintes de placement 0dp (aussi nommée match\_constraint)

Voici maintenant un exemple :

| <?xml version="1.0" encoding="utf-8"?> <androidx.constraintlayout.widget.ConstraintLayout  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <Button  android:id="@+id/button1"  android:text="BTN1"  android:layout\_height="wrap\_content"  app:layout\_constraintTop\_toTopOf="parent"  android:layout\_marginTop="32dp"  android:layout\_width="wrap\_content"  app:layout\_constraintLeft\_toLeftOf="parent"  app:layout\_constraintRight\_toRightOf="parent"  app:layout\_constraintHorizontal\_bias="0.25"/>  <Button  android:id="@+id/button2"  android:text="BTN2"  android:layout\_height="wrap\_content"  app:layout\_constraintTop\_toBottomOf="@id/button1"  android:layout\_width="0dp"  app:layout\_constraintLeft\_toLeftOf="@id/button1"  app:layout\_constraintRight\_toRightOf="parent"/> </androidx.constraintlayout.widget.ConstraintLayout> |
| --- |

Le premier bouton est positionné à 32 pixels du haut et à 25% à gauche dans la largeur de son parent. Le second bouton est mis sous le premier, et comme sa largeur est nulle, elle est forcée à ce que donnent ses contraintes : du bord gauche du bouton 1 au bord droit du parent. À vous maintenant.

Essayez de reproduire cette disposition avec un ConstraintLayout :



Le bouton 1 est centré en haut. Le bouton 2 remplit tout l’espace entre le coin bas gauche du n°1 et le coin bas gauche du parent. Le bouton 3 est au coin bas droit du n°1.

1. Exercices

Chacun de ces exercices consiste à créer un fichier res/layout/exercice.xml en mettant le titre de l’exercice à la place. Il vous suffit de changer l’appel à setContentView(R.layout.exercice) dans MainActivity.java pour passer d’un exercice à l’autre.

Par exemple, vous créez un nouveau fichier res/layout/seance.xml et vous mettez setContentView(R.layout.seance) dans la méthode onCreate de MainActivity. C’est ce layout qui sera affiché.

Dans les exercices, on demande de créer au minimum un écran qui permette de faire l’opération indiquée et au mieux, si vous avez le temps, cet écran sera ergonomique et élégant quelle que soit la taille et l’orientation de la tablette : les vues bien alignées, avec un libellé en face, etc.

Également : les textes seront à ajouter sous forme de ressources dans strings.xml. AndroidStudio vous affiche une petite ampoule pour vous suggérer de transformer la chaîne « hard-coded » en ressource.

1. Saisie de repas : repas.xml

Créez une interface pour saisir un repas : une zone de saisie texte pour l’entrée, une pour le plat principal, deux cases à cocher pour indiquer si on a mangé du fromage et un fruit (on doit pouvoir cocher les deux). Il faut rajouter deux RadioButton midi et soir tels que si on coche soir, ça décoche midi et inversement. Pour cela, il faut les placer dans un RadioGroup. N’oubliez pas un bouton pour valider et éventuellement un autre pour annuler.

1. Saisie de relevés météo : meteo.xml

Créez une interface pour saisir un relevé météo : date (DatePicker), heure(TimePicker), température (°C), vitesse du vent (km/h) et état du ciel (Spinner.. parmi dégagé, couvert et pluvieux).