

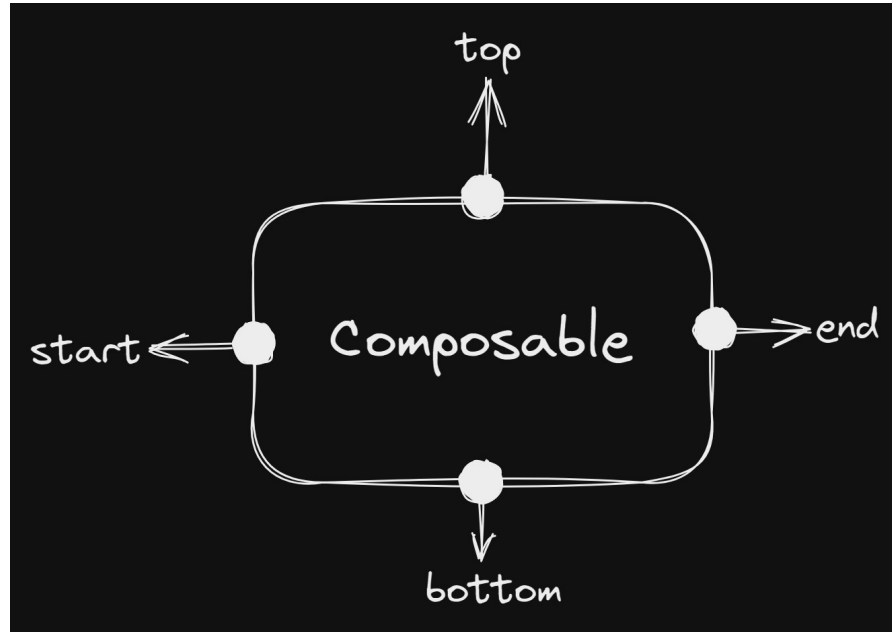
M7 - ConstraintLayout



Definició de **ConstraintLayout**

- És un layout que s'utilitza en dissenys complexos i evita que haguem d'utilitzar vistes aniuades (un row dins d'un column, per exemple).
- El posicionament dels elements es fa en relació als altres. Per això tindrem en compte que cada element té les següents zones que el delimiten: top, bottom, start, end.

Definició de layout




Dependències

- Per poder treballar amb ConstraintLayout al nostre projecte haurem d'afegir la següent dependència a l'arxiu gradle (nivell app):

```
implementation("androidx.constraintlayout:constraintlayout-compose:1.0.1")
```

- Recordem que, cada vegada que fem un canvi als arxius gradle, hem de sincronitzar.

 Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

[Sync Now](#) [Ignore these changes](#)

- També haurem d'importar:
`import androidx.constraintlayout.compose.ConstraintLayout`

ConstraintLayout

- Per crear un ConstraintLayout escrivim el següent codi:

```
@Composable
fun MyConstraintLayout(modifier: Modifier = Modifier){
    ConstraintLayout {
        Box(modifier = Modifier.size(120.dp).background(Color.Red))
        Box(modifier = Modifier.size(120.dp).background(Color.Yellow))
        Box(modifier = Modifier.size(120.dp).background(Color.Green))
        Box(modifier = Modifier.size(120.dp).background(Color.Cyan))
        Box(modifier = Modifier.size(120.dp).background(Color.Magenta))
    }
}
```

ConstraintLayout: referències

- Amb el codi anterior, obviament els quatre Box que hem creat es veuran un sobre l'altre. Per poder ubicar un Box en funció dels altres, hem de definir unes referències a cada Box. Creem les següents variables:

```
ConstraintLayout {  
    val (boxRed, boxYellow, boxGreen, boxCyan, boxMagenta) = createRefs()  
    ...  
}
```

- La funció createRefs crea una referència per a cada variable que hem definit.

ConstraintLayout: referències

- A continuació haurem d'assignar les referències creades a cada element (box en el nostre exemple):

```
Box(modifier = Modifier.size(120.dp).background(Color.Red).constrainAs(boxRed){})  
Box(modifier = Modifier.size(120.dp).background(Color.Yellow).constrainAs(boxYellow){})  
Box(modifier = Modifier.size(120.dp).background(Color.Green).constrainAs(boxGreen){})  
Box(modifier = Modifier.size(120.dp).background(Color.Cyan).constrainAs(boxCyan){})  
Box(modifier = Modifier.size(120.dp).background(Color.Magenta).constrainAs(boxMagenta){})
```

- Ara ens falta definir les restriccions (com a mínim una horitzontal i una altra vertical) dins de les claus ({}) de cada Box.

ConstraintLayout: referències

- Per alinear al centre de la pantalla el primer box escrivim:

```
Box(modifier = Modifier.size(120.dp).background(Color.Red).constrainAs(boxRed){  
    top.linkTo(parent.top)  
    start.linkTo(parent.start)  
    bottom.linkTo(parent.bottom)  
    end.linkTo(parent.end)  
})
```

- “parent” fa referència a la vista superior de l'element actual. En l'exemple, la pantalla.
- Per centrar un element dins del seu parent, hem de definir totes les restriccions (top, bottom, start, end) i igualar-les entre si.

ConstraintLayout: referències

- Definim les següents restriccions pel boxYellow:

```
Box(modifier = Modifier.size(120.dp).background(Color.Yellow).constrainAs(boxYellow){  
    start.linkTo(boxRed.start)  
    bottom.linkTo(boxRed.top)  
})
```

- Definim les següents restriccions pel boxGreen:

```
Box(modifier = Modifier.size(120.dp).background(Color.Green).constrainAs(boxGreen){  
    start.linkTo(boxRed.start)  
    top.linkTo(boxRed.bottom)  
})
```

ConstraintLayout: referències

- Definim les següents restriccions pel boxCyan:

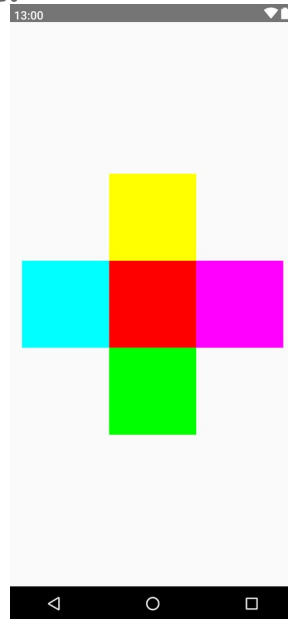
```
Box(modifier = Modifier.size(120.dp).background(Color.Cyan).constrainAs(boxCyan){  
    top.linkTo(boxRed.top)  
    end.linkTo(boxRed.start)  
})
```

- Definim les següents restriccions pel boxMagenta:

```
Box(modifier = Modifier.size(120.dp).background(Color.Magenta).constrainAs(boxMagenta){  
    top.linkTo(boxRed.top)  
    start.linkTo(boxRed.end)  
})
```

ConstraintLayout: referències

- El resultat és el següent:



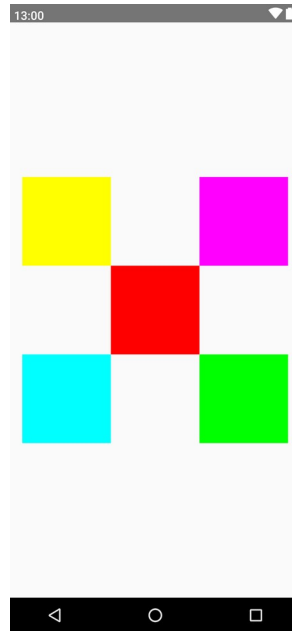
ConstraintLayout: referències

- Quan definim les restriccions amb la funció `linkTo`, podem especificar també un marge com a segon paràmetre:

```
Box(modifier = Modifier.size(120.dp).background(Color.Cyan).constrainAs(boxCyan){  
    top.linkTo(boxRed.top, margin = 20.dp)  
    end.linkTo(boxRed.start)  
})
```

Exercici

- Crea el següent disseny:



Guidelines

- Les Guidelines serveixen per crear línies invisibles que serviran per definir àrees de referència dins la pantalla del dispositiu.
- Gràcies a les Guidelines podrem ubicar els components no només en funció dels límits de la pantalla, sino també respecte a les pròpies Guidelines.

Guidelines

- Per crear una Guideline podem fer el següent:

```
ConstraintLayout {  
    val topGuide = createGuidelineFromTop(0.1f)  
    val boxRed = createRef()  
    Box(modifier = Modifier.size(120.dp).background(Color.Red).constrainAs(boxRed){  
        top.linkTo(topGuide)  
    })  
}
```

- Hem creat una Guideline horitzontal des de la part superior de la pantalla (createGuidelineFromTop) a una distància del 10% de la llargada de la pantalla (0.1f)

GuideLines

- Fixa't que hem utilitzat percentatges per ubicar la Guideline. També podem especificar una mida concreta (amb dp), però és millor utilitzar percentatges ja que la nostra app es veurà igual independentment de les diverses mides de pantalla dels diferents dispositius en que s'utilitza la nostra app.
- Podem crear Guidelines també amb les funcions:
`createGuidelineFromBottom,`
`createGuidelineFromStart, createGuidelineFromEnd.`

Chains

- Observa el següent exemple:

```
val (boxRed, boxYellow, boxGreen) = createRefs()
Box(modifier.size(50.dp).background(Color.Red).constrainAs(boxRed){
    start.linkTo(parent.start)
    end.linkTo(boxYellow.start)
})
Box(modifier.size(50.dp).background(Color.Yellow).constrainAs(boxYellow){
    start.linkTo(boxRed.end)
    end.linkTo(boxGreen.start)
})
Box(modifier.size(50.dp).background(Color.Green).constrainAs(boxGreen){
    start.linkTo(boxYellow.end)
    end.linkTo(parent.end)
})
```

Chains

- Els tres Boxes creen una cadena entre ells i es distribueixen en l'ample de la pantalla mantenint la mateixa distància entre ells.
- Aquesta distància la podem modificar creant una cadena (Chain). El següent codi va a continuació dels tres Boxes:

```
createHorizontalChain(boxRed, boxYellow, boxGreen, chainStyle = ChainStyle.Packed)
```

- Tenim 3 tipus de ChainStyle: Packed, Spread i SpreadInside. Observa les diferències entre ells.
- També tenim la funció createVerticalChain.