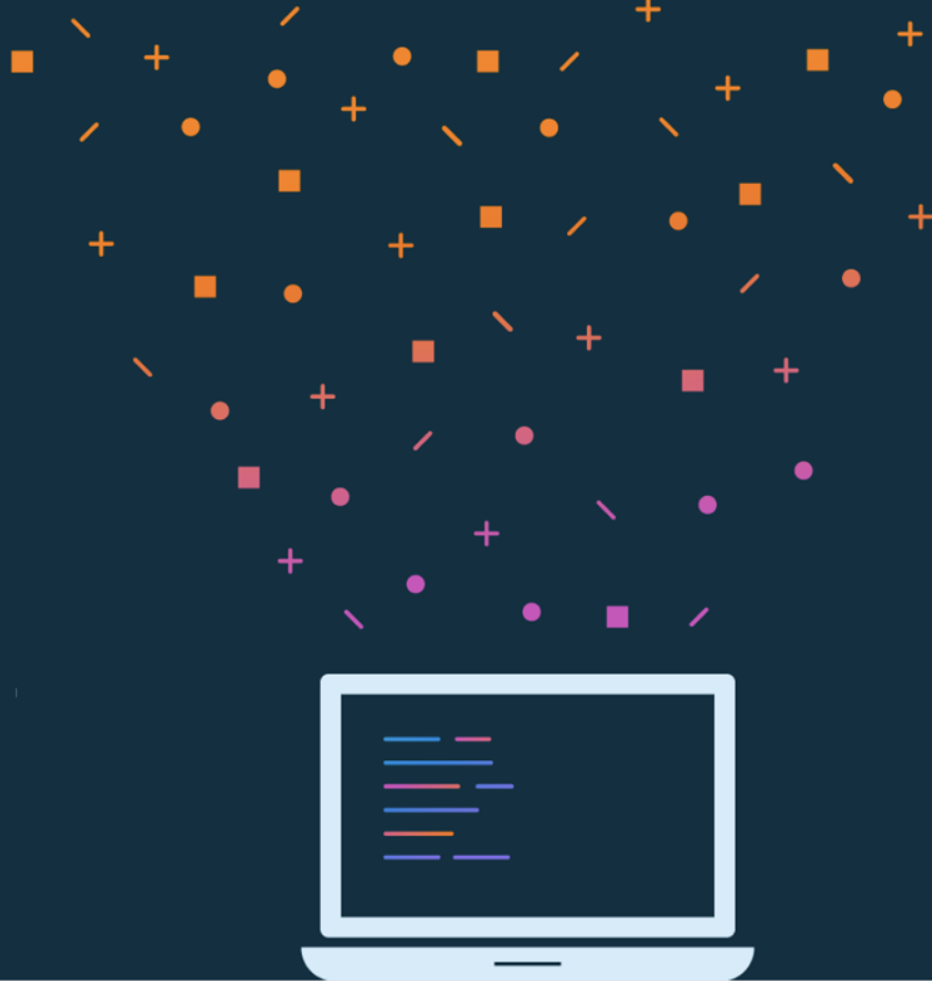




# Bài học 5: Bố cục



# Giới thiệu về bài học này

## Bài học 5: Bố cục

- [Bố cục trong Android](#)
- [ConstraintLayout](#)
- [Chủ đề bổ sung về ConstraintLayout](#)
- [Liên kết dữ liệu](#)
- [Hiển thị danh sách bằng RecyclerView](#)
- [Tóm tắt](#)

# Bố cục trong Android

# Thiết bị Android

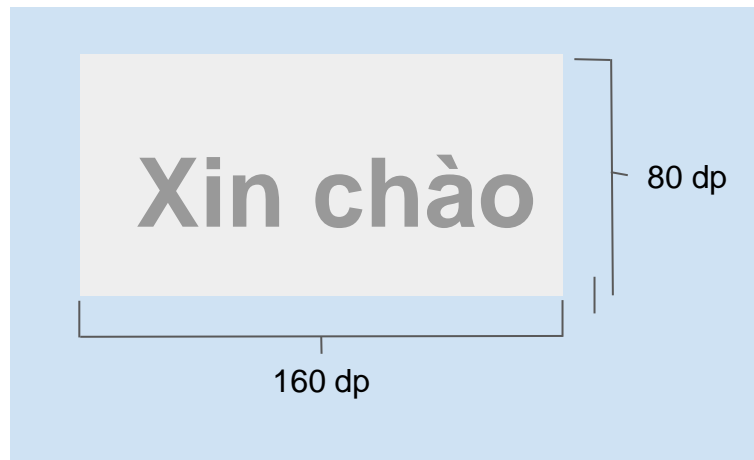
- Thiết bị Android có nhiều kiểu dáng.
- Số pixel trên mỗi inch ngày càng tăng trên các màn hình thiết bị.
- Các nhà phát triển cần có khả năng chỉ định kích thước bố cục nhất quán trên các thiết bị.



# Pixel không phụ thuộc vào mật độ (dp)

Dùng dp khi chỉ định các kích thước trong bố cục của bạn, chẳng hạn như chiều rộng hoặc chiều cao của chế độ xem.

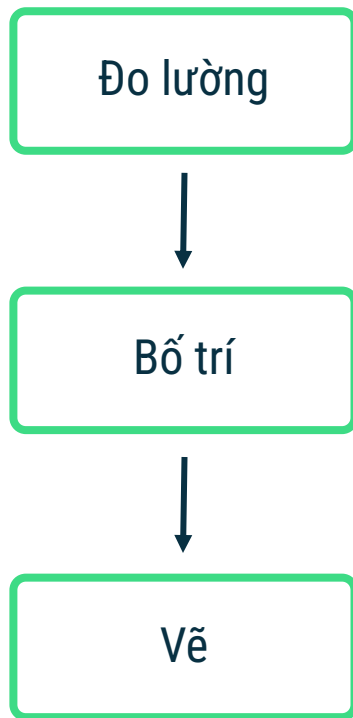
- Pixel không phụ thuộc vào mật độ (dp) xem xét đến mật độ màn hình.
- Các chế độ xem của Android được tính bằng số pixel không phụ thuộc vào mật độ.
- $dp = \frac{\text{chiều rộng tính bằng pixel} * 160}{\text{mật độ màn hình}}$



# Nhóm mật độ màn hình

Bộ định tính mật độ	Nội dung mô tả	DPI ước tính
ldpi (hầu như không dùng đến)	Mật độ thấp	~120 dpi
mdpi (mật độ cơ sở)	Mật độ trung bình	~160 dpi
hdpi	Mật độ cao	~240 dpi
xhdpi	Mật độ siêu cao	~320 dpi
xxhdpi	Mật độ siêu siêu cao	~480 dpi
xxxhdpi	Mật độ siêu siêu siêu cao	~640 dpi

# Chu kỳ hiển thị Chế độ xem của Android

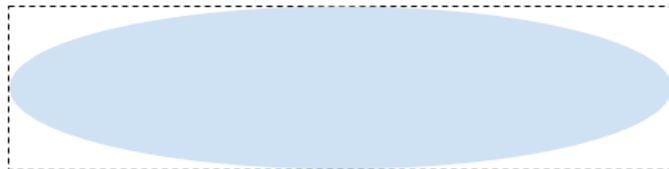


# Vùng vẽ

Những gì chúng ta thấy:



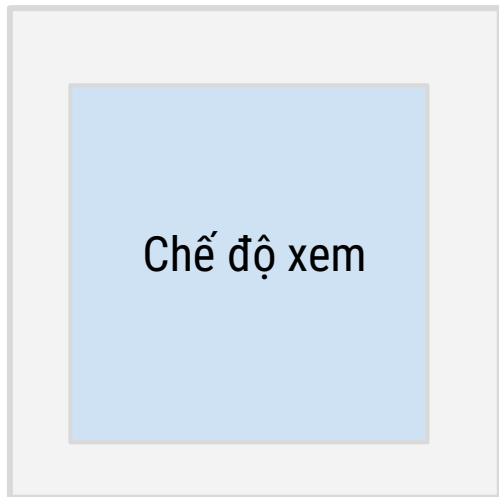
Cách hệ thống vẽ:



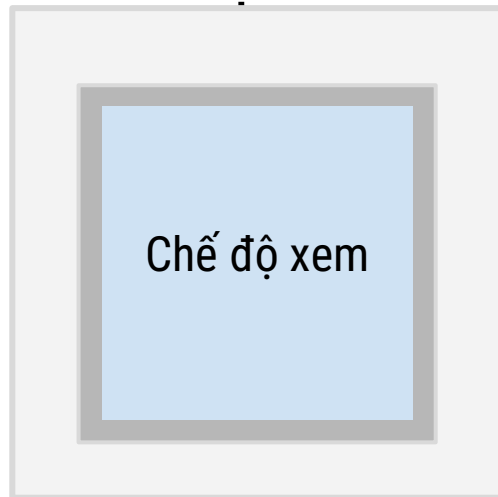


# Lề và khoảng đệm của chế độ xem

Chế độ xem có lề



Chế độ xem có lề và khoảng đệm



# ConstraintLayout

# Bố cục được lồng sâu khá là tốn kém

- ViewGroup được lồng sâu đòi hỏi nhiều hoạt động tính toán hơn
- Chế độ xem có thể được đo lường nhiều lần
- Có thể khiến giao diện người dùng bị chậm và phản hồi kém

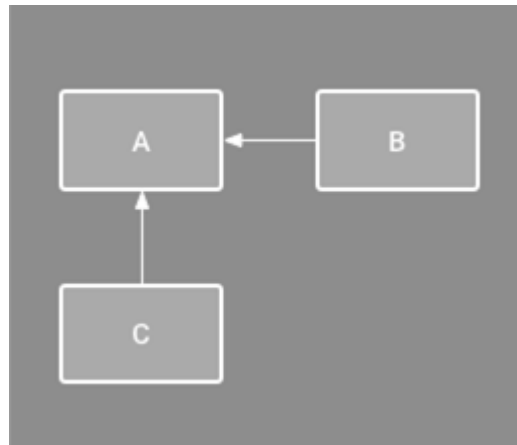
Hãy dùng ConstraintLayout để tránh một trong số các vấn đề này!

# ConstraintLayout là gì?

- Bố cục mặc định đề xuất cho Android
- Giải quyết vấn đề tốn kém khi lồng quá nhiều bố cục, đồng thời cho phép thực hiện hành vi phức tạp
- Xác định vị trí và kích thước của các chế độ xem trong bố cục bằng cách dùng một tập hợp các hạn chế

# Hạn chế là gì?

Là một hạn chế hoặc giới hạn đối với các thuộc tính của Chế độ xem mà bố cục cố tuân theo



# Các hạn chế tương đối khi xác định vị trí

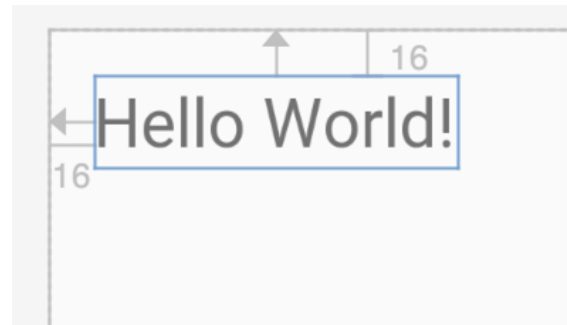
Có thể thiết lập một hạn chế tương quan với vùng chứa mẹ

**Định dạng:** `layout_constraint<SourceConstraint>_to<TargetConstraint>Of`

Example attributes on a TextView:

`app:layout_constraintTop_toTopOf="parent"`

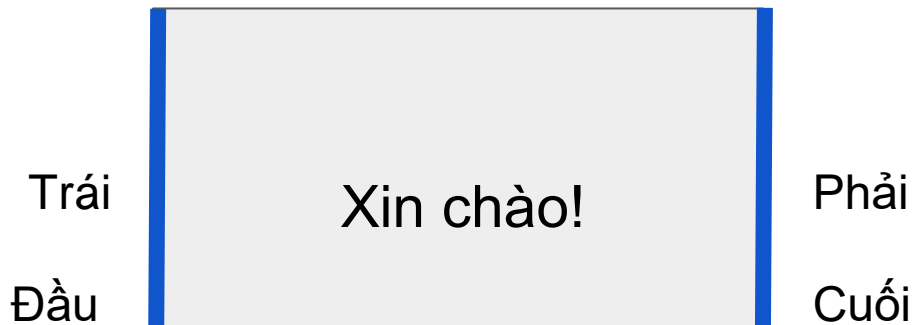
`app:layout_constraintLeft_toLeftOf="parent"`



# Các hạn chế tương đối khi xác định vị trí



# Các hạn chế tương đối khi xác định vị trí





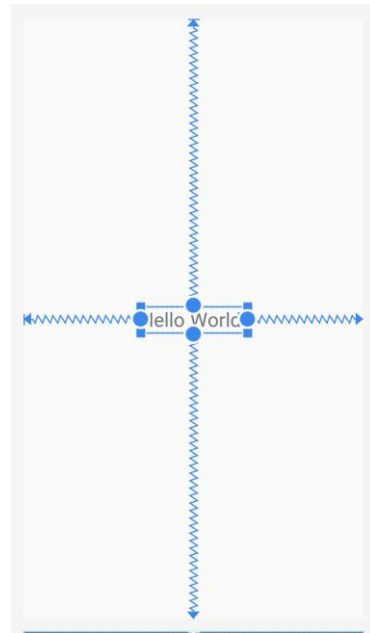
# Ví dụ đơn giản về ConstraintLayout

```
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">
```

```
<TextView  
    ...
```

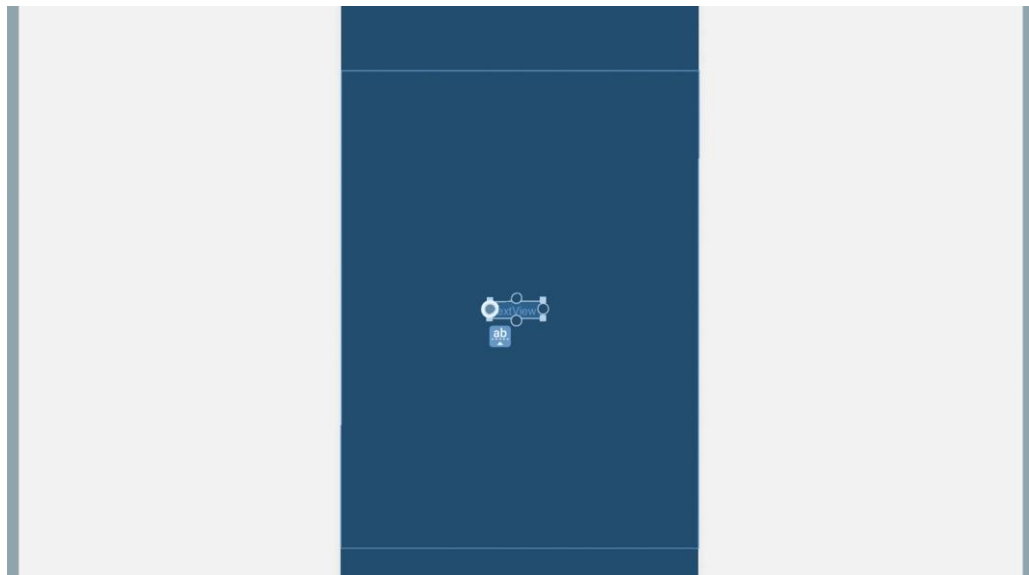
```
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```



# Layout Editor trong Android Studio

Bạn có thể nhấp và kéo để thêm các hạn chế cho một Chế độ xem.



# Tiện ích hạn chế trong Layout Editor



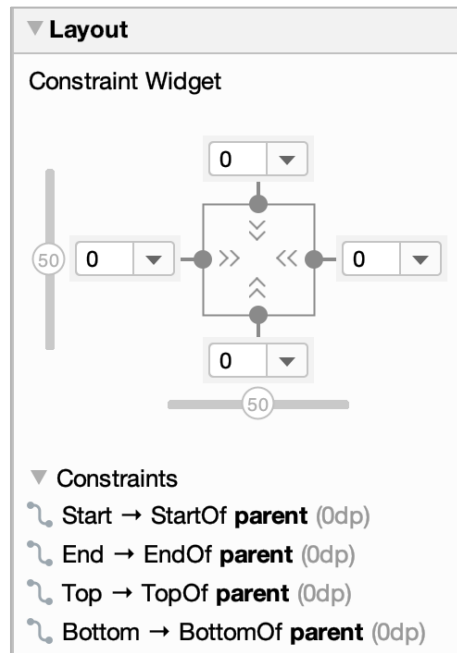
Cố định



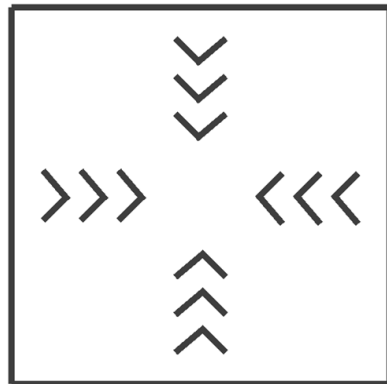
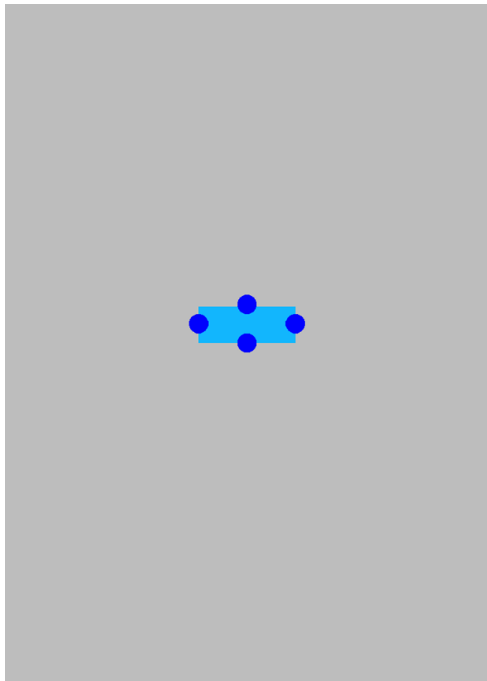
Co giãn theo nội dung



Khớp với các hạn chế



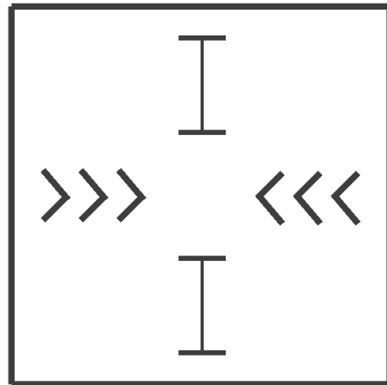
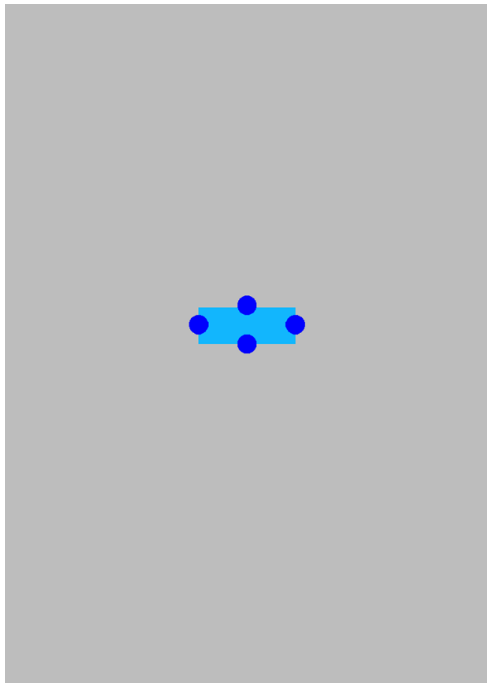
# Co giãn theo nội dung cho chiều rộng và chiều cao



`layout_width`    `wrap_content`

`layout_height`    `wrap_content`

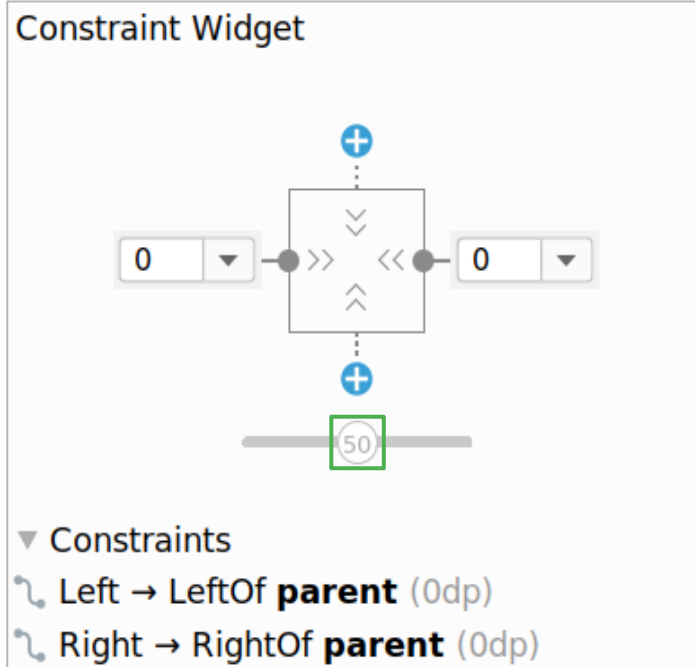
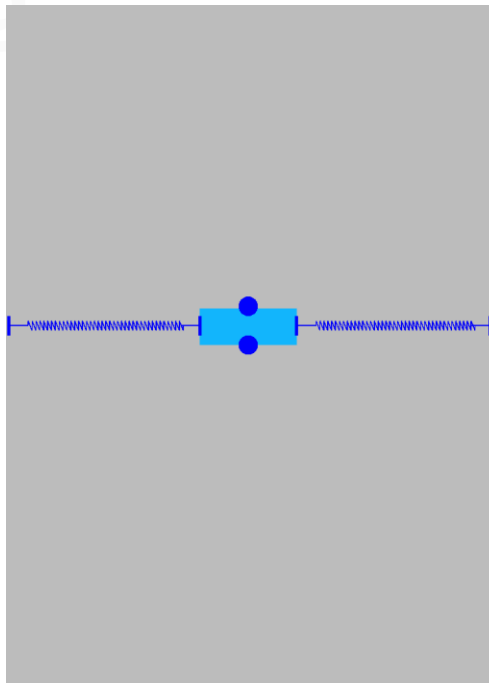
# Co giãn theo nội dung cho chiều rộng, chiều cao cố định



`layout_width`    `wrap_content`

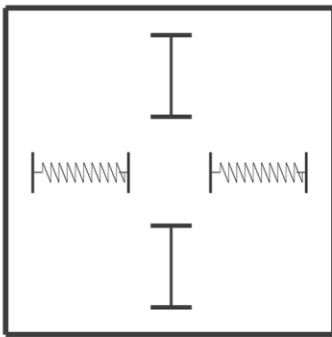
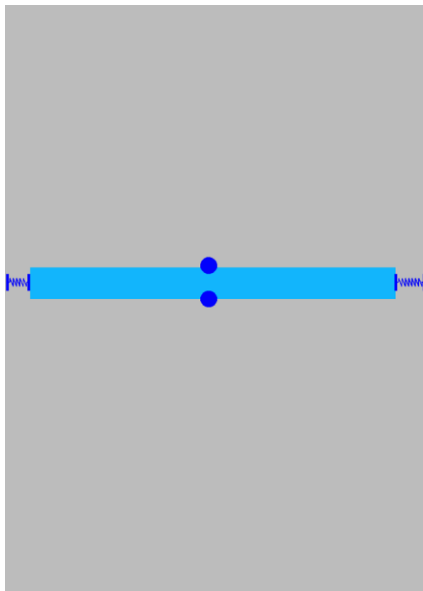
`layout_height`    `48dp`

# Căn giữa chế độ xem theo hướng ngang



# Dùng `match_constraint`

Không thể dùng `match_parent` trên chế độ xem con, hãy dùng `match_constraint`



`layout_width`     `0dp(match_constraint)`

`layout_height`   `48dp`

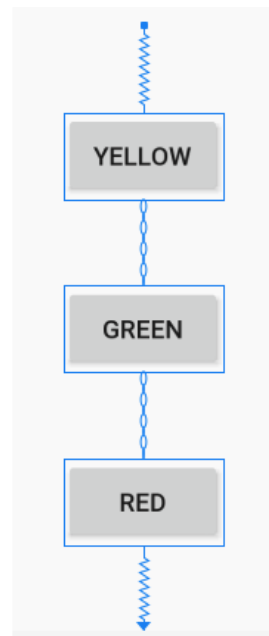
# Chuỗi

- Cho phép bạn xác định vị trí tương quan của các chế độ xem với nhau
- Có thể được liên kết theo hướng ngang hoặc hướng dọc
- Cung cấp nhiều chức năng LinearLayout



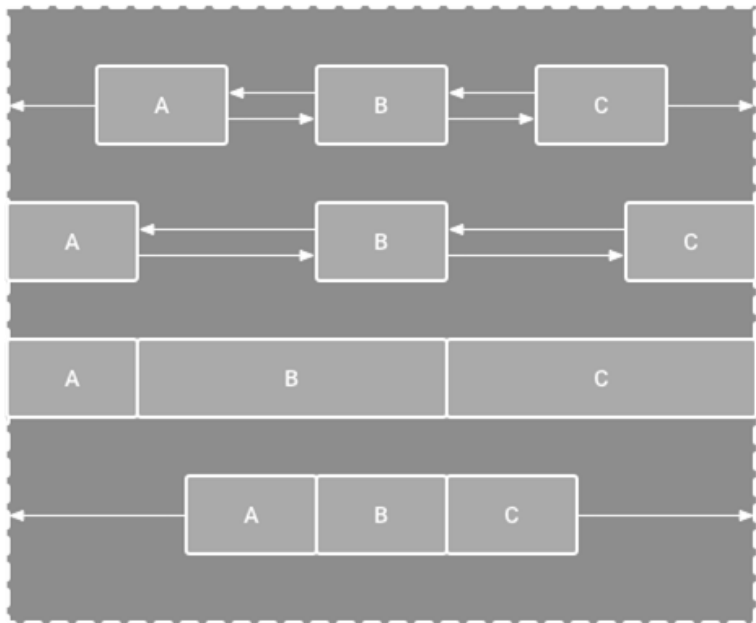
# Tạo một chuỗi trong Layout Editor

1. Chọn các đối tượng mà bạn muốn có trong chuỗi.
2. Nhấp chuột phải rồi chọn **Chuỗi**.
3. Tạo một chuỗi ngang hoặc chuỗi dọc.



# Kiểu chuỗi

Điều chỉnh không gian giữa các chế độ xem với những kiểu chuỗi sau.



Chuỗi trải rộng

Chuỗi trải rộng bên trong

Chuỗi trọng số

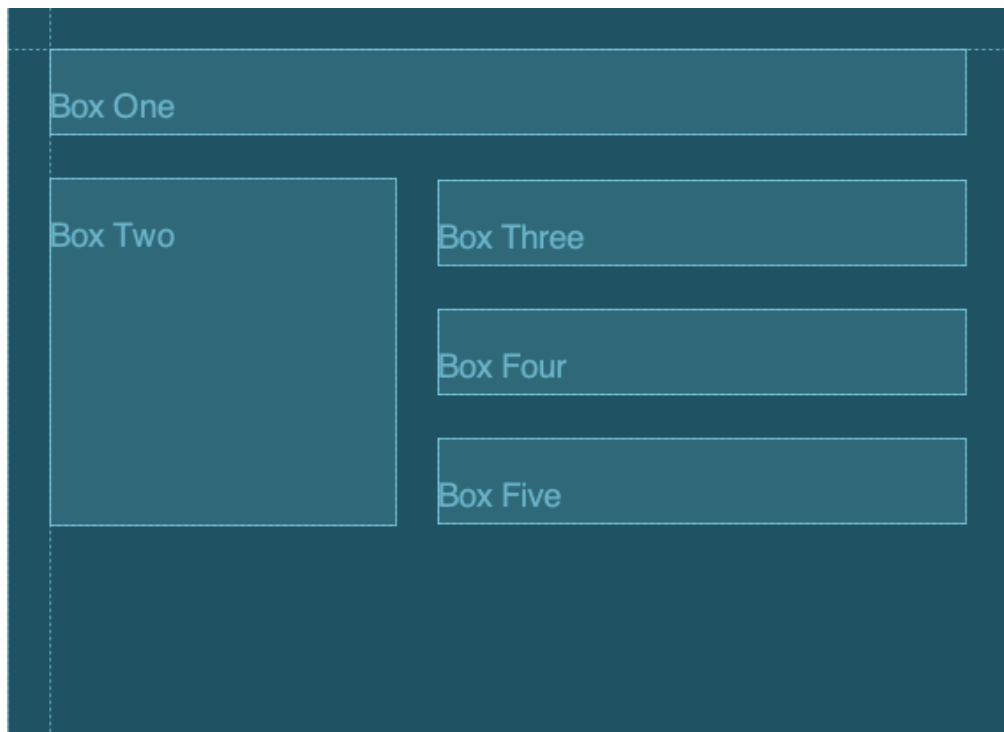
Chuỗi đóng gói

# Chủ đề bổ sung về ConstraintLayout

# Đường căn

- Cho phép bạn xác định vị trí của nhiều chế độ xem có liên quan đến một đường căn
- Có thể theo hướng dọc hoặc hướng ngang
- Cho phép cộng tác hiệu quả hơn với các nhóm thiết kế/trải nghiệm người dùng
- Không được vẽ trên thiết bị

# Đường căn trong Android Studio



# Ví dụ về đường căn

```
<ConstraintLayout>
```

```
    <androidx.constraintlayout.widget.Guideline
```

```
        android:id="@+id/start_guideline"
```

```
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
```

```
        android:orientation="vertical"
```

```
        app:layout_constraintGuide_begin="16dp" />
```

```
    <TextView ...
```

```
        app:layout_constraintStart_toEndOf="@id/start_guideline" />
```

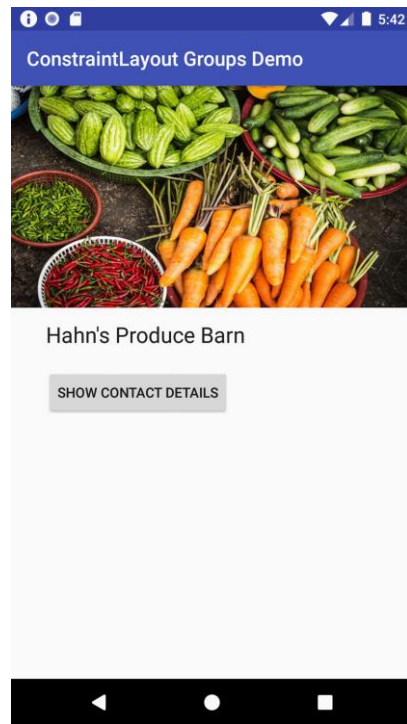
```
</ConstraintLayout>
```

# Tạo đường căn

- `layout_constraintGuide_begin`
- `layout_constraintGuide_end`
- `layout_constraintGuide_percent`

# Nhóm

- Kiểm soát chế độ hiển thị của một tập hợp tiện ích
- Chế độ hiển thị của nhóm có thể được bật/tắt bằng mã





# Ví dụ về nhóm

```
<androidx.constraintlayout.widget.Group  
    android:id="@+id/group"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:constraint_referenced_ids="locationLabel,locationDetails"/>
```

# Mã ứng dụng của nhóm

```
override fun onClick(v: View?) {  
    if (group.visibility == View.GONE) {  
        group.visibility = View.VISIBLE  
        button.setText(R.string.hide_details)  
    } else {  
        group.visibility = View.GONE  
        button.setText(R.string.show_details)  
    }  
}
```

# Liên kết dữ liệu

# Phương pháp tiếp cận hiện tại: findViewById()

Truyền tải hệ phân cấp Chế độ xem mỗi lần

MainActivity.kt

```
val name = findViewById(...)
val age = findViewById(...)
val loc = findViewById(...)

name.text = ...
age.text = ...
loc.text = ...
```

findViewById

findViewById

findViewById

activity\_main.xml

```
<ConstraintLayout ... >
  <TextView
    android:id="@+id/name"/>
  <TextView
    android:id="@+id/age"/>
  <TextView
    android:id="@+id/loc"/>
</ConstraintLayout>
```

# Dùng liên kết dữ liệu

Liên kết các thành phần giao diện người dùng trong bố cục của bạn với các nguồn dữ liệu trong ứng dụng.

MainActivity.kt

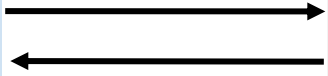
```
Val binding:ActivityMainBinding
```

```
binding.name.text = ...
```

```
binding.age.text = ...
```

```
binding.loc.text = ...
```

initialize binding



activity\_main.xml

```
<layout>
  <ConstraintLayout ... >
    <TextView
      android:id="@+id/name"/>
    <TextView
      android:id="@+id/age"/>
    <TextView
      android:id="@+id/loc"/>
  </ConstraintLayout>
</layout>
```

# Sửa đổi tệp build.gradle

```
android {  
    ...  
    buildFeatures {  
        dataBinding true  
    }  
}
```

# Thêm thẻ layout

**<layout>**

```
<androidx.constraintlayout.widget.ConstraintLayout>  
    <TextView ... android:id="@+id/username" />  
    <EditText ... android:id="@+id/password" />  
</androidx.constraintlayout.widget.ConstraintLayout>
```

**</layout>**

# Tăng cường bố cục bằng liên kết dữ liệu

Thay thế phương thức này

```
setContentView(R.layout.activity_main)
```

bằng phương thức này

```
val binding: ActivityMainBinding = DataBindingUtil.setContentView(  
    this, R.layout.activity_main)
```

```
binding.username = "Melissa"
```



# Biến bố cục liên kết dữ liệu

```
<layout>
    <data>
        <variable name="name" type="String"/>
    </data>
    <androidx.constraintlayout.widget.ConstraintLayout>
        <TextView
            android:id="@+id/textView"
            android:text="@{name}" />
        </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

In MainActivity.kt:

```
binding.name = "John"
```

# Biểu thức bố cục liên kết dữ liệu

```
<layout>
  <data>
    <variable name="name" type="String"/>
  </data>

  <androidx.constraintlayout.widget.ConstraintLayout>
    <TextView
      android:id="@+id/textView"
      android:text="@{name.toUpperCase()}" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

# Hiển thị danh sách bằng RecyclerView

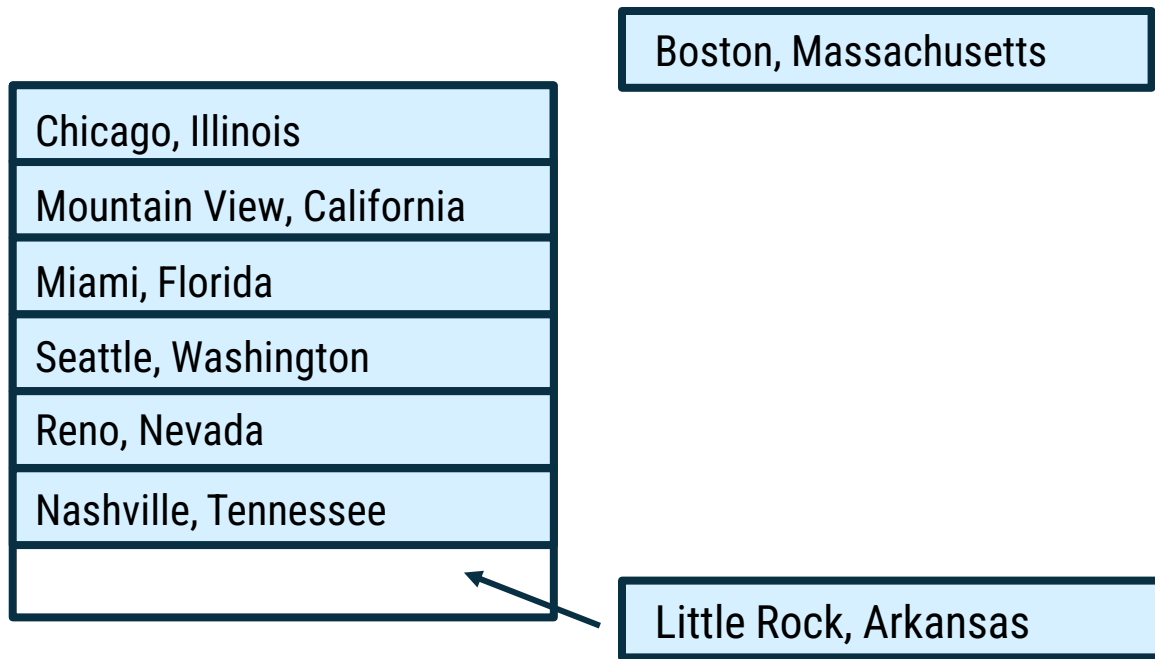
# RecyclerView

- Tiện ích để hiển thị danh sách dữ liệu
- "Tái chế" (tái sử dụng) các chế độ xem mục để giúp cuộn hiệu quả hơn
- Có thể chỉ định bố cục mục danh sách cho từng mục trong tập dữ liệu
- Hỗ trợ hoạt ảnh và quá trình chuyển đổi

# RecyclerView.Adapter

- Cung cấp dữ liệu và bố cục mà RecyclerView hiển thị
- Một Bộ chuyển đổi tùy chỉnh mở rộng từ `RecyclerView.Adapter` và ghi đè 3 hàm sau:
  - `getItemCount`
  - `onCreateViewHolder`
  - `onBindViewHolder`

# Xem tính năng tái chế trong RecyclerView



Nếu mục được cuộn ngoài màn hình, mục đó sẽ không bị hủy bỏ. Mục được đưa vào một nhóm để tái chế.

`onBindViewHolder` liên kết chế độ xem với các giá trị mới, sau đó, chế độ xem được chèn lại vào danh sách.

# Thêm RecyclerView vào bố cục của bạn

```
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/rv"  
    android:scrollbars="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

# Tạo bố cục mục danh sách

```
res/layout/item_view.xml
```

```
<FrameLayout
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content">
```

```
    <TextView
```

```
        android:id="@+id/number"
```

```
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content" />
```

```
</FrameLayout>
```



# Tạo bộ chuyển đổi danh sách

```
class MyAdapter(val data: List<Int>) : RecyclerView.Adapter<MyAdapter.MyViewHolder>()
{
    class MyViewHolder(val row: View) : RecyclerView.ViewHolder(row) {
        val textView = row.findViewById<TextView>(R.id.number)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        val layout = LayoutInflater.from(parent.context).inflate(R.layout.item_view,
            parent, false)
        return MyViewHolder(layout)
    }
    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
        holder.textView.text = data.get(position).toString()
    }
    override fun getItemCount(): Int = data.size
}
```

# Đặt bộ chuyển đổi trên RecyclerView

Trong tệp MainActivity.kt:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    val rv: RecyclerView = findViewById(R.id.rv)  
    rv.layoutManager = LinearLayoutManager(this)  
  
    rv.adapter = MyAdapter(IntRange(0, 100).toList())  
}
```



# Tóm tắt

# Tóm tắt

Trong Bài học 5, bạn đã tìm hiểu cách:

- Chỉ định độ dài tính bằng dp cho bố cục của bạn
- Làm việc với mật độ màn hình cho nhiều kiểu thiết bị Android
- Hiển thị Chế độ xem ra màn hình ứng dụng của bạn
- Bố trí các chế độ xem trong ConstraintLayout bằng cách dùng các hạn chế
- Đơn giản hóa việc nhận thông tin tham chiếu đến Chế độ xem từ bố cục bằng liên kết dữ liệu
- Hiển thị danh sách các mục văn bản bằng RecyclerView và bộ chuyển đổi tùy chỉnh

# Tìm hiểu thêm

- [Mật độ pixel trên Android](#)
- [Giãn cách](#)
- [Chỉ số về thiết bị](#)
- [Thang loại](#)
- [Xây dựng một giao diện người dùng phản hồi nhanh bằng ConstraintLayout](#)
- [Thư viện liên kết dữ liệu](#)
- [Tạo danh sách linh động bằng RecyclerView](#)

# Lộ trình

Thực hành những gì bạn đã học được bằng cách hoàn thành lộ trình này:

[Bài học 5: Bố cục](#)

