View Binding vs findViewById<>()

View Binding gera automaticamente uma classe de vinculação para cada arquivo de layout no projeto.
View Binding é eficiente para vinculação de views em tempo de compilação

build.gradle:

```
android {
    buildFeatures {
        viewBinding true
    }
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val binding = TesteActivityBinding.inflate(layoutInflater)
     val recyclerView = binding.TesteActivityBindingRecyclerview
        recyclerView.adapter = adapter
       configuraFab()
 }
```

FindViewById() recupera a referência de uma View e vincula em tempo de execução.
Serve para referenciar o componente visual do layout com o componente da activity.

```
class ListaNotasActivity : AppCompatActivity(R.layout.lista_notas_activity) {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val recyclerView = findViewById<RecyclerView>(R.id.lista_notas_activity_recyclerview)
        recyclerView.adapter = adapter
        configuraFab()
    }

    private fun configuraFab() {
            val fab = findViewById<ExtendedFloatingActionButton>
            (R.id.lista_notas_activity_fab)
            fab.setOnClickListener {
        }
    }
}
```

Lateinit é útil em um contexto que não quero inicializar uma variável no momento da declaração, mas utilizá-la em um momento posterior. Permite que você adie a inicialização da propriedade.

```
class LoginFragment : Fragment() {

    private lateinit var usernameEditText: EditText
    private lateinit var passwordEditText: EditText

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        usernameEditText = view.findViewById(R.id.username_edit_text)
        passwordEditText = view.findViewById(R.id.password_edit_text)

    }
}
```

Lazy é útil em um cenário em que queremos criar um objeto dentro de uma classe. A inicialização é feita automaticamente e uma vez, isso evita a sobrecarga desnecessária do sistema.

```
class LoginFragment : Fragment() {

 private val usernameEditText by lazy { requireView().findViewById<EditText>(
        R.id.username_edit_text) }
 private val passwordEditText by lazy { requireView().findViewById<EditText>
        (R.id.password_edit_text) }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

    }
}
```

OU

```
class LoginFragment : Fragment() {

 private val binding by lazy { FragmentLoginBinding.inflate(layoutInflater) }
    private val usernameEditText by lazy { binding.usernameEditText }
    private val passwordEditText by lazy { binding.passwordEditText }
```

```
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View {
                              return binding.root
    }

   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

    }
}
```

RESUMO DE 4 LINHAS:

View Binding gera automaticamente uma classe de vinculação para cada arquivo de layout no projeto. Já o FindViewById() recupera a referência de uma View e vincula em tempo de execução. O Lateinit permite que você adie a inicialização da propriedade e o Lazy é útil em um cenário em que queremos criar um objeto dentro de uma classe.