🔥 1. Memory leak là gì?

Memory leak xảy ra khi object không còn được dùng nhưng vẫn được giữ reference → không được GC thu hồi → làm app nặng, chậm, crash (OutOfMemoryError).

🔥 2. Các nguyên nhân memory leak phổ biến

A. Context Leak

❗ Sai lầm phổ biến:

Giữ context (đặc biệt Activity/Fragment context) trong một object có thời gian sống lâu hơn → memory leak.

❌ Ví dụ leak:

```
object NetworkHelper {
    lateinit var context: Context
}
```

`object` sống suốt vòng đời ứng dụng → giữ luôn Activity → leak.

Cách đúng:

- Dùng **Application context** nếu cần giữ lâu
- Không giữ Activity context trong singleton hoặc static object

```
object NetworkHelper {
    fun initialize(appContext: Context) {
        this.context = appContext.applicationContext
    }
}
```

B. Static Reference Leak

❗ Lỗi phổ biến:

```
class MainActivity : AppCompatActivity() {
    companion object {
        var instance: MainActivity? = null
```

```
    }

    override fun onCreate(...) {
        instance = this  // ❌ giữ Activity static → leak
    }
}
```

Cách tránh:

Không bao giờ giữ Activity/Fragment trong biến static.

C. Lifecycle Misuse (quản lý vòng đời sai)

❗ Ví dụ leak trong Fragment:

```
class MyFragment : Fragment() {
    private var binding: FragmentBinding? = null

    override fun onDestroyView() {
        super.onDestroyView()
        // Không set binding = null → leak view hierarchy
    }
}
```

Cách đúng:

```
override fun onDestroyView() {
    binding = null
    super.onDestroyView()
}
```

D. Handler / Runnable Leak

❗ Lỗi:

```
val handler = Handler()
handler.postDelayed({
    // giữ Activity reference trong lambda
}, 50000)
```

Giải pháp:

- Dùng **LifecycleScope**
- Hoặc huỷ delay trong onDestroy()

E. Listener / Callback không gỡ bỏ

Ví dụ: đăng ký listener nhưng không unregister → leak Activity.

Gỡ trong onPause()/onStop()/onDestroy().

---

1. Android Profiler

Công cụ trong Android Studio dùng để phân tích:

- Memory usage
- CPU usage
- Network calls
- Garbage collection events

Memory Profiler giúp:

- tìm leak
- xem đối tượng bị giữ reference
- dump heap

---

2. LeakCanary (Square)

📌 Là thư viện giúp:

- tự động detect leak
- hiển thị leak trace
- cực hiệu quả khi debug

Cài đặt:

```
debugImplementation 'com.squareup.leakcanary:leakcanary-android:2.12'
```

Khi leak xảy ra:

LeakCanary tự hiện thông báo và cho bạn stacktrace chỉ ra đối tượng nào bị giữ.

Một ứng dụng Android mượt yêu cầu:

- RecyclerView chạy mượt
- render nhẹ
- tránh notifyDataSetChanged()
- phân trang để giảm tải

---

1. DiffUtil → Tối ưu RecyclerView

❗ Vấn đề:

Dùng `notifyDataSetChanged()` làm:

- refresh toàn bộ list
- tốn CPU
- UI giật lag

DiffUtil giải quyết bằng cách:

Chỉ update những item **thật sự thay đổi**.

Ví dụ DiffUtil:

```
class UserDiffUtil : DiffUtil.ItemCallback<User>() {
    override fun areItemsTheSame(old: User, new: User) = old.id == new.id
    override fun areContentsTheSame(old: User, new: User) = old == new
}
```

---

2. ListAdapter → RecyclerView tối ưu

`ListAdapter` tích hợp sẵn DiffUtil → tự tính toán update list trên background thread.

Ví dụ:

```
class UserAdapter :
    ListAdapter<User, UserAdapter.UserVH>(UserDiffUtil()) {

    override fun onBindViewHolder(holder: UserVH, position: Int) {
        holder.bind(getItem(position))
    }
}
```

→ Mượt hơn rất nhiều so với RecyclerView.Adapter thông thường.

---

3. Pagination (Paging 3)

Tại sao cần pagination?

- List lớn (1000+ item) → load toàn bộ = lag
- Tốn network & RAM
- Tăng thời gian render RecyclerView

Paging 3 giúp:

- Tải dữ liệu theo trang
- Tối ưu memory
- Support caching, retry, loading state
- Kết hợp tốt với Flow + Room

Ví dụ Paging 3:

Step 1: DataSource (PagingSource)

```
class UserPagingSource(
    private val api: ApiService
) : PagingSource<Int, User>() {

    override suspend fun load(params: LoadParams<Int>): LoadResult<Int, User> {
        val page = params.key ?: 1
        val result = api.getUsers(page)

        return LoadResult.Page(
            data = result.users,
            prevKey = if (page == 1) null else page - 1,
            nextKey = if (result.users.isEmpty()) null else page + 1
        )
    }
}
```

Step 2: Repository

```
fun getUsersPager() = Pager(
    PagingConfig(pageSize = 20)
) {
    UserPagingSource(api)
}.flow
```

Step 3: Use in ViewModel

```
val flow = repository.getUsersPager().cachedIn(viewModelScope)
```

Step 4: Submit to ListAdapter

```
lifecycleScope.launch {
    viewModel.flow.collect { data ->
        adapter.submitData(data)
    }
}
```