1. Tipo de arreglo ideal

El mejor candidato es un **buffer circular (ring buffer)** de tamaño fijo (100).

- **Trade-off:**
  - Circular → memoria constante, operaciones O(1), pero requiere lógica de índices.
  - Dinámico → más flexible, pero desperdicia memoria y tiempo en redimensionar, lo cual no es necesario aquí. Dado que siempre son 100 mensajes, el buffer circular es la opción óptima.

2. Interfaz (métodos y firmas)

pseudocode

```
class MessageBuffer(capacidad = 100):

  method add(message): void
  method getLatest(): Message
  method getLastN(n: int): List<Message>
  method size(): int
```

3. Optimización para el patrón de uso

- **O(1):**
  - add(message) → insertar nuevo mensaje y descartar el más viejo si está lleno.
  - getLatest() → acceso directo al índice actual.
  - size() → contador simple.
- **O(n):**
  - getLastN(n) → recorrer los últimos N mensajes en orden cronológico.
  - Esto es aceptable porque se usa ocasionalmente.

4. Algoritmo de evicción

Cuando el buffer está lleno y llega un nuevo mensaje:

- Sobrescribir el mensaje más viejo.
- Avanzar el puntero circular (head).
- Mantener siempre los últimos 100 mensajes.

5. Pseudocódigo

pseudocode

```
class MessageBuffer:
   var buffer[capacidad]
   var capacidad = 100
   var head = 0       // índice del mensaje más reciente
   var count = 0      // número de mensajes almacenados

   method add(message):
      head = (head + 1) mod capacidad
      buffer[head] = message
      if count < capacidad:
         count = count + 1

   method getLatest():
      return buffer[head]

   method getLastN(n):
      if n > count:
         n = count
      result = []
      start = (head - n + 1 + capacidad) mod capacidad
      for i in 0..(n-1):
         index = (start + i) mod capacidad
         result.append(buffer[index])
      return result

   method size():
      return count
```

6. Complejidad

- add(message) → **O(1)**
- getLatest() → **O(1)**
- getLastN(n) → **O(n)**
- size() → **O(1)**

7. Uso de memoria

- Exactamente **100 slots**.
- No hay desperdicio por sobreasignación ni redimensionamiento.
- Constante y predecible.

8. Diagrama ASCII de circularidad

Código

```
Buffer (capacidad = 100)

[ M1 ][ M2 ][ M3 ] ... [ M100 ]
  ^                 ^
 oldest              newest (head)

Cuando llega M101:
[ M101 ][ M2 ][ M3 ] ... [ M100 ]
  ^                   ^
 oldest overwritten          newest
```

9. Comparación con ArrayList

- **ArrayList:**
  - Crece dinámicamente → innecesario aquí.
  - Evicción manual → costosa (O(n) para eliminar el primero).
  - Más memoria desperdiciada por sobreasignación.
- **MessageBuffer (circular):**
  - Memoria fija → exacta, sin desperdicio.
  - Evicción automática en O(1).
  - Acceso al último mensaje inmediato.
  - Perfecto para escenarios de alta frecuencia de inserción.