- Esperamos hasta las 19:05 parara que ingresen todos sus compañeros


# Patrones de Diseño de Software

PATRONES COMPORTAMIENTO - ITERATOR

# Patron de diseño Iterator

## PROPOSITO:

**Iterator** es un patrón de diseño de comportamiento que te permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).

# Patrones de diseño – Iterator

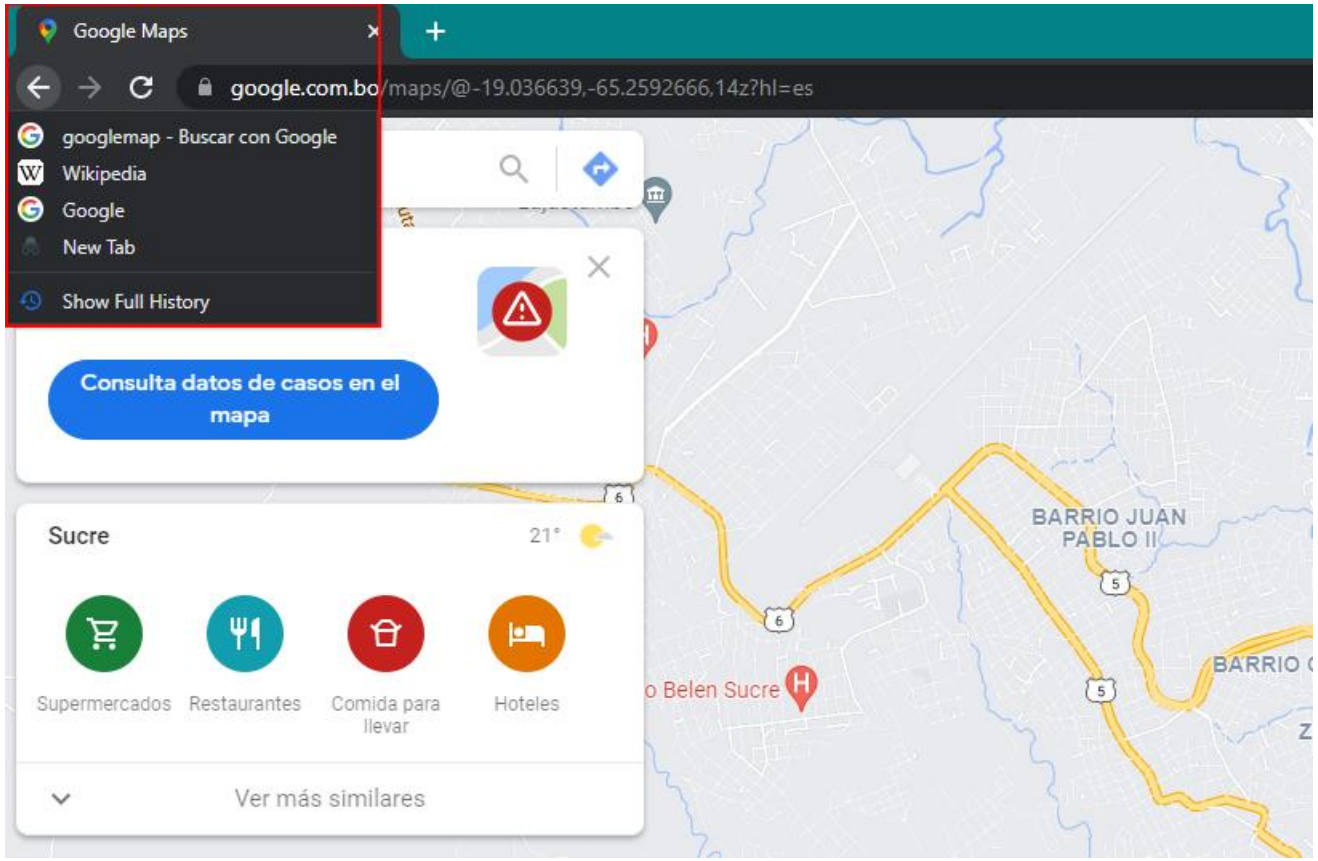
**Problema:** Diferentes maneras de recorrer el historial del browser

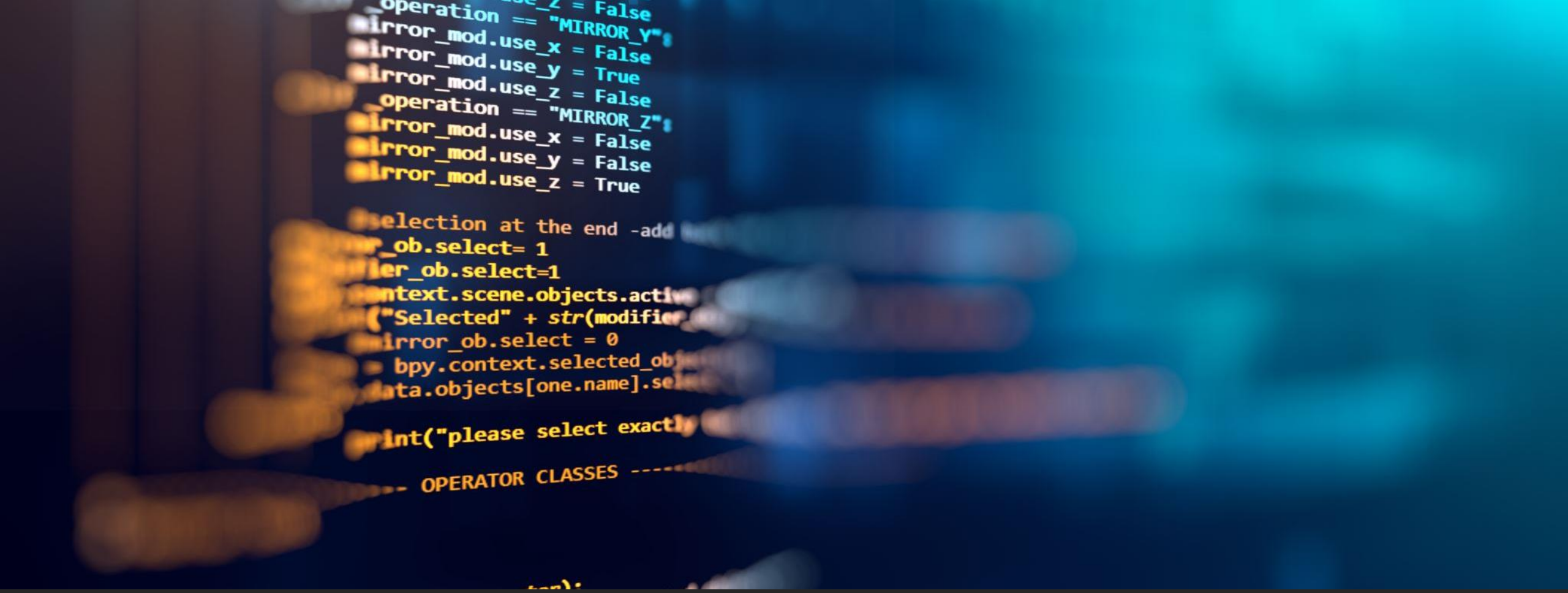
---

**Paso 1:** Implementemos nuestro *BrowseHistory* haciendo uso de una lista.

**Paso 2:** Ahora supongamos que deseamos cambiar esa implementación por otra estructura de datos como por ejemplo un arreglo.

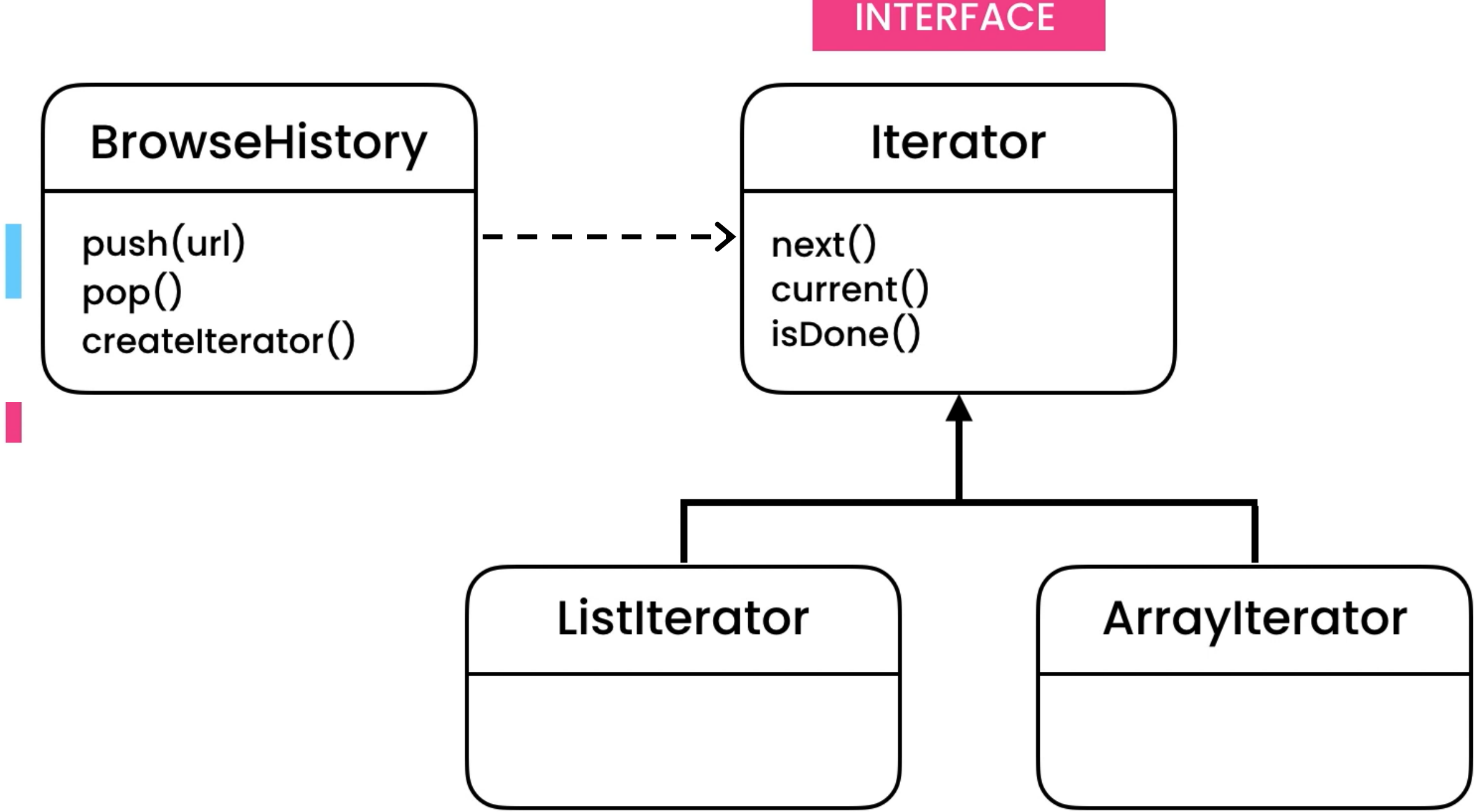
**¿Que pasaría** si estuviésemos recorriendo nuestro objeto en mas de una clase?

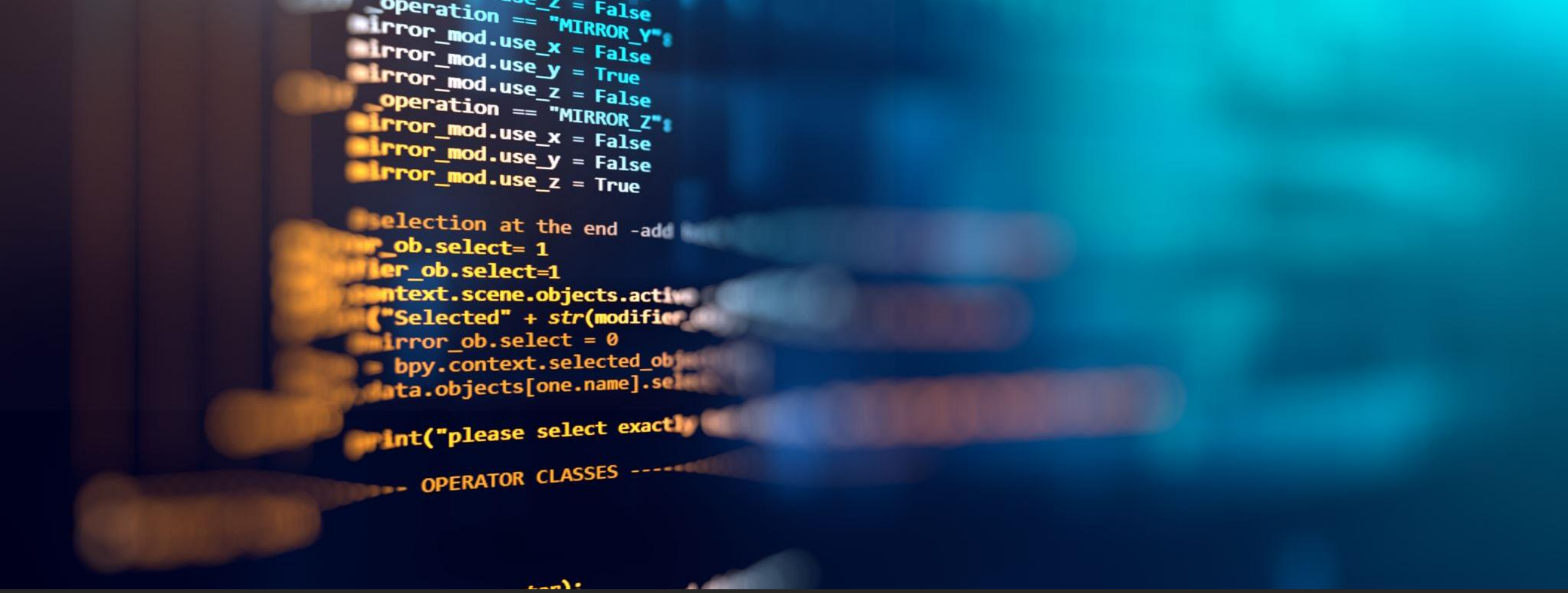





# Demo

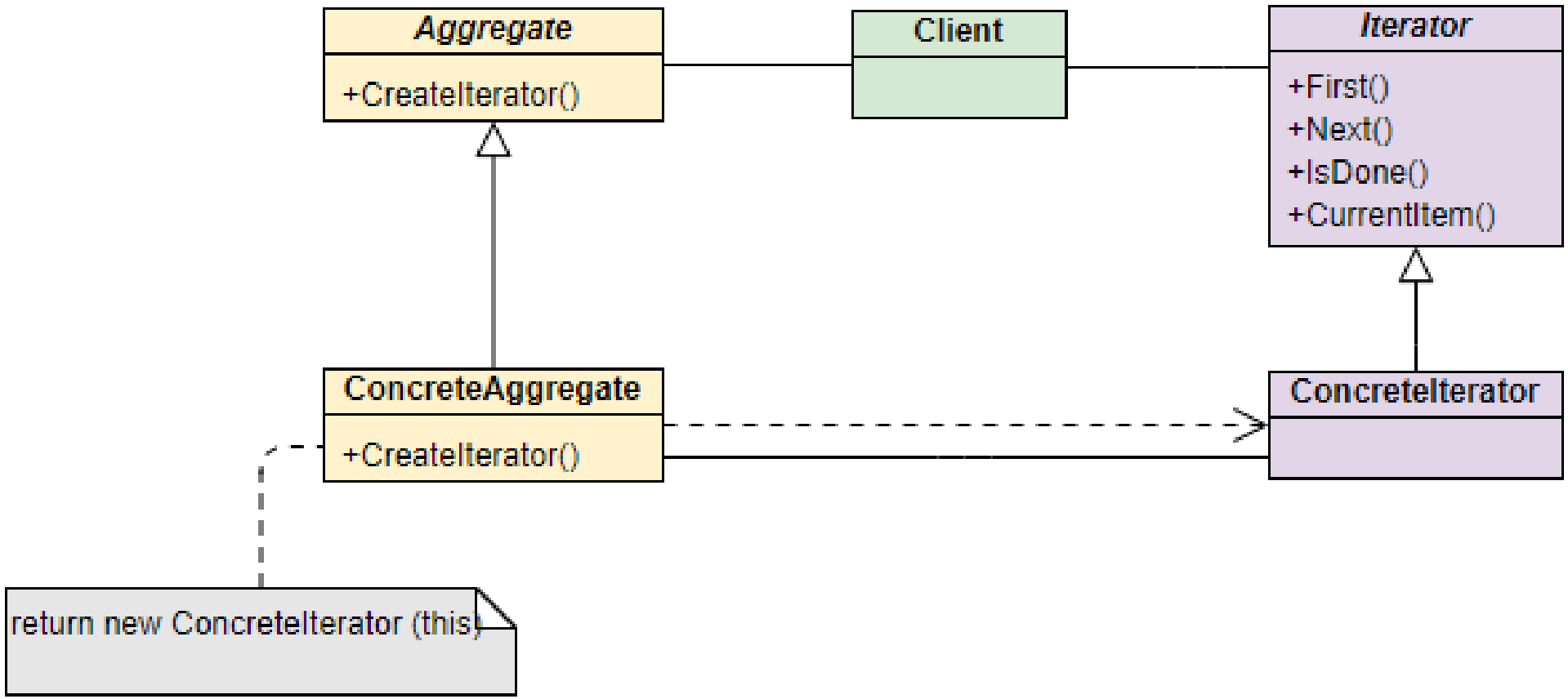
Revisemos en Código la implementación del problema


INTERFACE
BrowseHistory
push(url)
pop()
createIterator()
Iterator
next()
current()
isDone()
ListIterator
ArrayIterator




# Demo

Revisemos en Código la implementación de la solución




# Patron de diseño - Iterator

Estructura

# Patrones de diseño – Iterator

Implementación

- Declara la interfaz iteradora
- Declara la interfaz de colección y describe un método para buscar iteradores
- Implementa clases iteradoras concretas para las colecciones que quieras que sean recorridas por iteradores
- Implementa la interfaz de colección en tus clases de colección
- Repasa el código cliente para sustituir todo el código de recorrido de la colección por el uso de iteradores

# Patrones de diseño – Iterator

Ventajas:

- ✓ Principio de responsabilidad única. Puedes limpiar el código cliente y las colecciones extrayendo algoritmos de recorrido voluminosos y colocándolos en clases independientes.
- ✓ Principio de abierto/cerrado. Puedes implementar nuevos tipos de colecciones e iteradores y pasarlos al código existente sin descomponer nada.
- ✓ Puedes recorrer la misma colección en paralelo porque cada objeto iterador contiene su propio estado de iteración.
- ✓ Por la misma razón, puedes retrasar una iteración y continuar cuando sea necesario..

Desventajas:

- × Aplicar el patrón puede resultar excesivo si tu aplicación funciona únicamente con colecciones sencillas.
- × Utilizar un iterador puede ser menos eficiente que recorrer directamente los elementos de algunas colecciones especializadas.

# Patrón de diseño Iterator

Cuándo utilizar este patrón?

Utiliza el patrón Iterator cuando tu colección tenga una estructura de datos compleja a nivel interno, pero quieras ocultar su complejidad a los clientes (ya sea por conveniencia o por razones de seguridad).

Utiliza el patrón para reducir la duplicación en el código de recorrido a lo largo de tu aplicación.

Utiliza el patrón Iterator cuando quieras que tu código pueda recorrer distintas estructuras de datos, o cuando los tipos de estas estructuras no se conozcan de antemano.

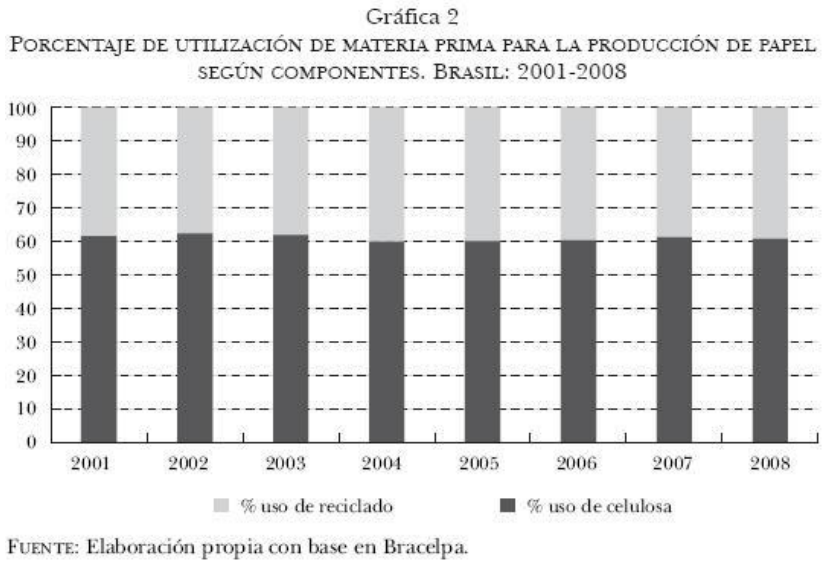
# Patrones de diseño – Iterator

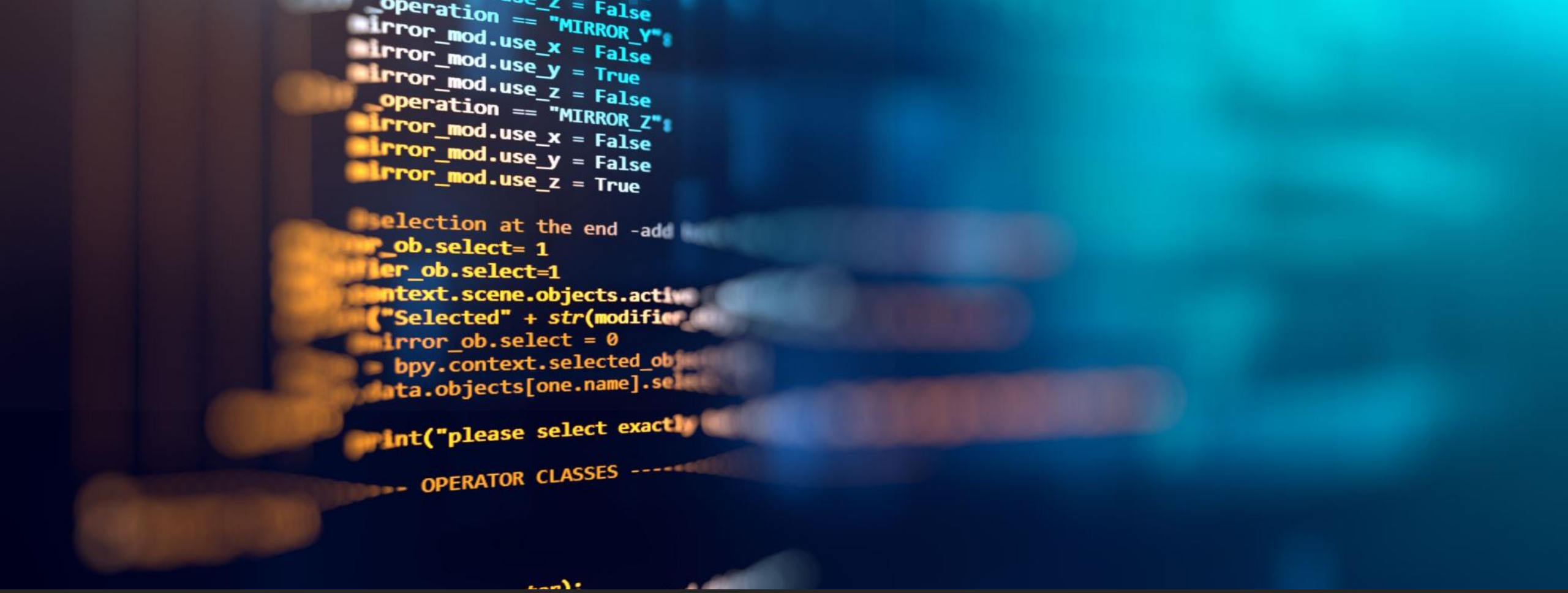
## Práctica

- Supongamos que témenos una **tienda de productos**.
- Deseamos implementar un **"Reporte de productos"** para lo cual debemos recorrer todos los productos e imprimirlos por pantalla
- El producto al menos tiene los siguientes datos: *id, nombre, cantidad y precio*
- El método **ToString()** permite mostrar todos los datos del producto

**Implementemos esta característica utilizando el patrón**

**Iterator**






# Demo

Solución de la tarea

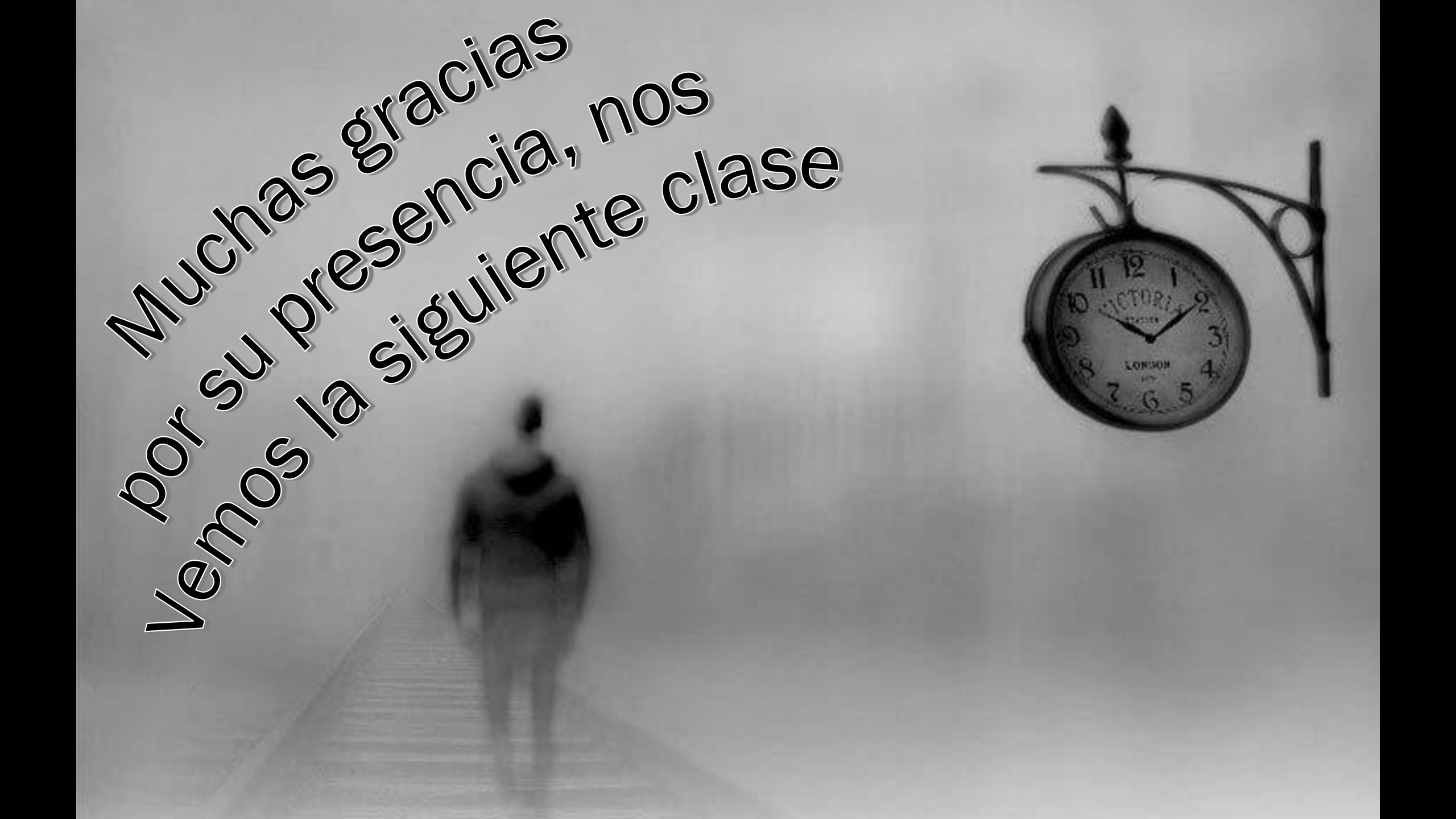
# Patrones de diseño – Iterator

## Tarea

Supongamos que témenos 3 DJs que tocaran en nuestra fiesta de esta noche, les hemos pedido a todos su lista de canciones, las cuales debemos juntar para reproducir esta noche.

Sin embargo cada DJ nos ha mandado la lista en diferentes formatos.

Simulemos este ejemplo utilizando el patrón **Iterator**.


Muchas gracias
por su presencia, nos
Vemos la siguiente clase