¿List vs ArrayList? Es una de las preguntas más típicas de Java para un programador cuando no tenemos demasiada experiencia. Vamos a hablar un poco a fondo de ello . Cuando nos referimos a que usar nos estamos refiriendo a si referenciar una variable usando List o referenciar una variable usando ArrayList . Para entenderlo mejor vamos a ver un poco de código:

```
import java.util.ArrayList;

public class Principal {

	public static void main(String[] args) {
		ArrayList<String> milista= new ArrayList<String>();
		milista.add("hola");
		milista.add("que");
		milista.add("tal");
		milista.add("estas");
		milista.add("hola");
		imprimirLista(milista);

	}
	public static void imprimirLista(ArrayList<String> lista) {
		for (String cadena: lista) {
			System.out.println(cadena);
		}
	}

}
```

En este bloque de código lo único que hacemos es iniciar un ArrayList y recorrerlo a través de un método estático. En principio el código funciona correctamente y no hay mucho que decir.

Interfaces y Flexibilidad

Sin embargo tenemos que tener en cuenta que el código funciona de la misma manera si cambiamos ArrayList por List a nivel de definición de tipos .

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

public class Principal2 {

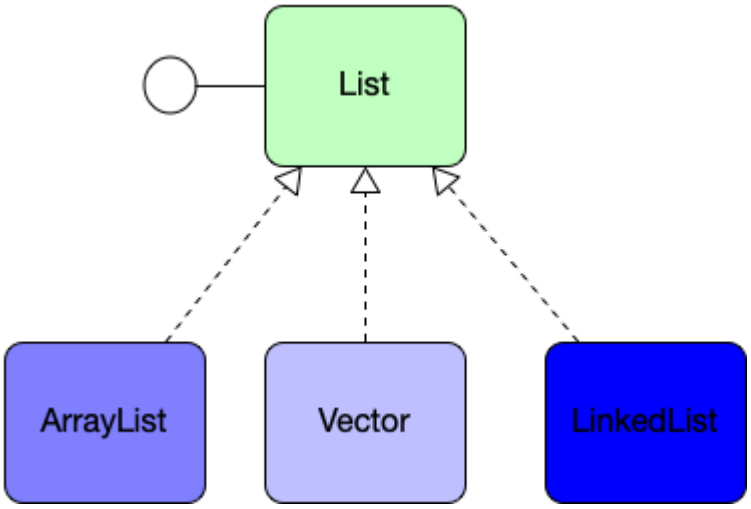
public static void main(String[] args) {
                List<String> milista= new ArrayList<String>();
                milista.add("hola");
                milista.add("que");
                milista.add("tal");
                milista.add("estas");
                milista.add("hola");
                imprimirLista(milista);

        }
        public static void imprimirLista(List<String> lista) {
                for (String cadena: lista) {
                        System.out.println(cadena);
                }
        }


}
```

Recordemos que el interface List esta implementado por varias clases concretamente por

Vector, LinkedList y ArrayList



El hecho de aplicar estos cambios aporta flexibilidad al programa ya que permite que en un futuro cambiemos de implementación si lo necesitamos por ejemplo podríamos cambiar de ArrayList a LinkedList y el método de imprimir no se vería afectado.

```
package com.arquitecturajava;

import java.util.LinkedList;
import java.util.List;

public class Principal3 {

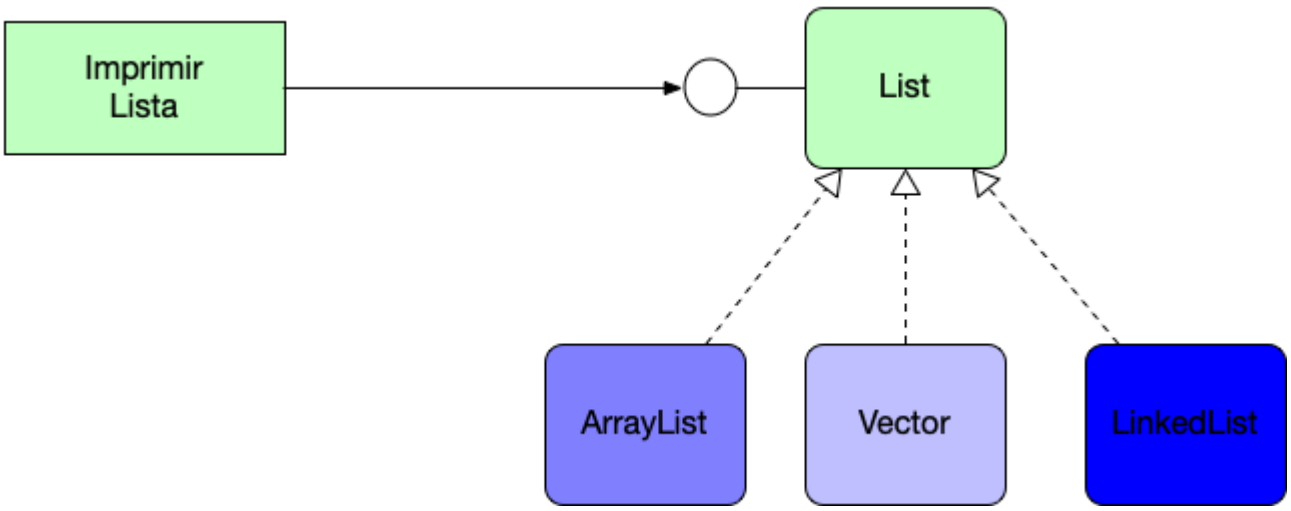
public static void main(String[] args) {
                List<String> milista= new LinkedList<String>();
                milista.add("hola");
                milista.add("que");
                milista.add("tal");
                milista.add("estas");
                milista.add("hola");
```

```
            imprimirLista(milista);

        }
        public static void imprimirLista(List<String> lista) {
            for (String cadena: lista) {
                System.out.println(cadena);
            }
        }

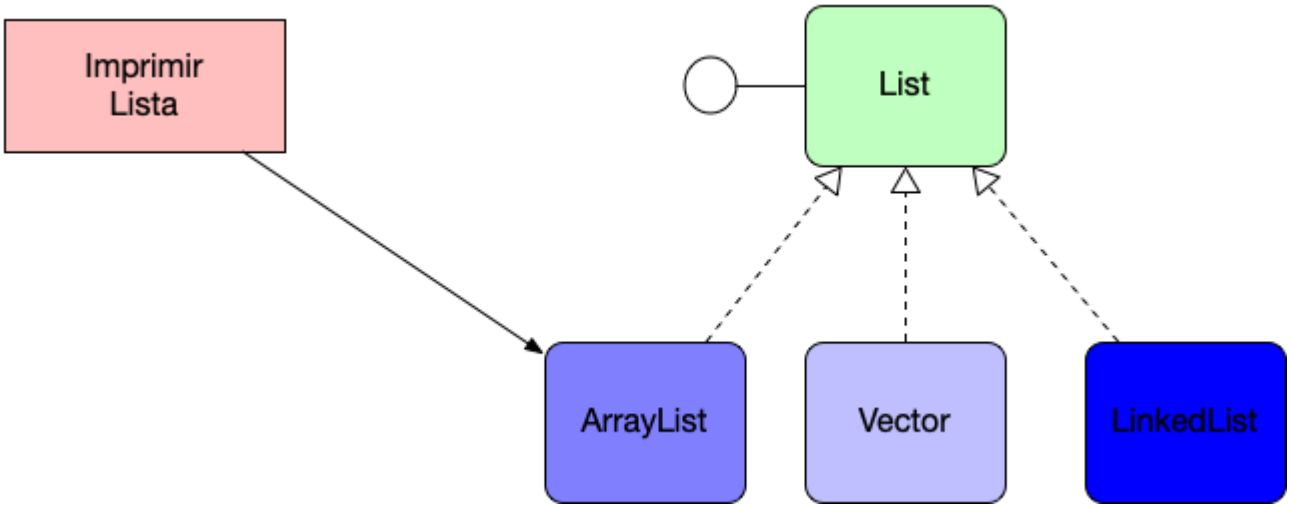

}
```

De esta manera se aporta flexibilidad ya que el método imprimirLista no depende directamente de una implementación concreta de la clase sino de un interface que muchas clases pueden implementar según nuestras necesidades.



List vs ArrayList

Esta es una buena práctica programar siempre orientado a interfaces y no a sus implementaciones concretas.



Esto es clave cuando estamos trabajando con clases como Repositorios que continuamente devuelven listas de datos. A veces uno se encuentra métodos como:

```
public interface FacturaRepository {

	ArrayList<Factura> buscarTodos();
}
```

Esto es fuertemente incorrecto y deberíamos tener métodos que se encarguen de devolver variables que apunten a interfaces . En este caso

```
package com.arquitecturajava;

import java.util.List;

public interface FacturaRepository {

	List<Factura> buscarTodos();
}
```

Tengámoslo siempre en cuenta y nos ayudará a construir APIs más flexibles y a entender mejor las que utilizamos en nuestro día a día . Por ejemplo Connection , ResultSet y

Statement son interfaces de JDBC.

Otros artículos relacionados

- Java Iterable to List
- Java ArrayList for y sus opciones
- Java 9 Factory Methods (List,Set,Map)