Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estructura de Datos

Semana 2

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Logro de la sesión

Al finalizar la sesión, el estudiante:

- **Utiliza en forma crítica estructuras de datos lineales: arreglos y listas enlazadas para la solución de problemas**

- **Explica el marco de colecciones de Java, incluidas las listas y usa estas estructuras juntas para manipular y examinar datos de muchas maneras para resolver problemas de programación**

Estructuras de datos lineales

Arreglos y Listas enlazadas

01

Listas

- **Representa una secuencia de elementos de algún tipo, sobre la que podemos insertar, borrar, ordenar, consultar o buscar elementos**
- Ejemplos.
  - Lista de canales de la Tele, de alumnos de un grupo de clase, de espera para una operación.
  - Nombre se personas (strings): María, Pepe, Juan, …
  - Números enteros: 5,6,1,-3,0,2,1,…
  - Objetos (instancias) de la clase Punto
  - Objetos (instancias) de la clase Empleado

- Todos los elementos deben pertenecer al mismo tipo de datos. No podemos tener una lista con canales, niños y paciente. No tiene sentido. Otra característica es la ordenación. Es importante saber en qué posición está cada elemento.

Hay dos formas de implementar las listas: arreglos y listas enlazadas

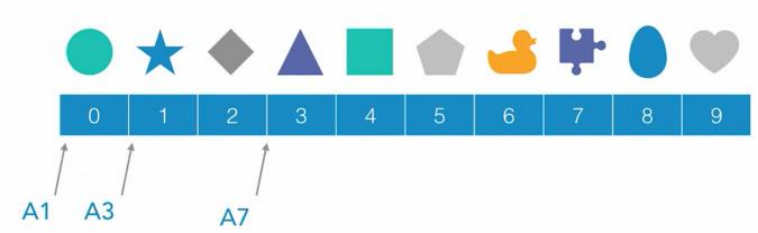
Listas

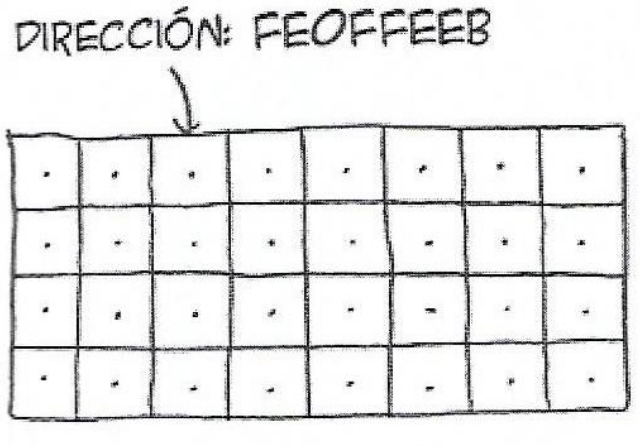
¿Cómo funciona la memoria de una computadora?

Imagina que vas a un espectáculo y necesitas guardar tus cosas, te ofrecen una cómoda

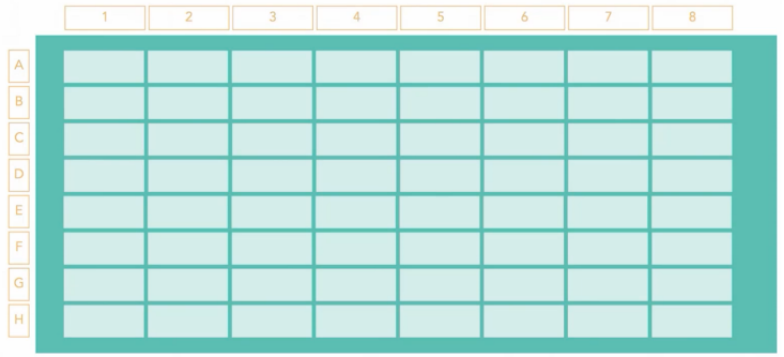
Cada cajón de esa cómoda puede contener un solo elemento. Quieres guardar dos elementos así que solicitas dos cajones y guardas tus cosas.

Básicamente es así como funciona la memoria, parece un conjunto gigante de cajones y cada cajón tiene una dirección.
Cada vez que quieras almacenar un elemento en la memoria le pides espacio a tu ordenador y este te da una dirección donde pueda almacenarlo.





fe0ffeeb es la dirección de un espacio en la memoria.



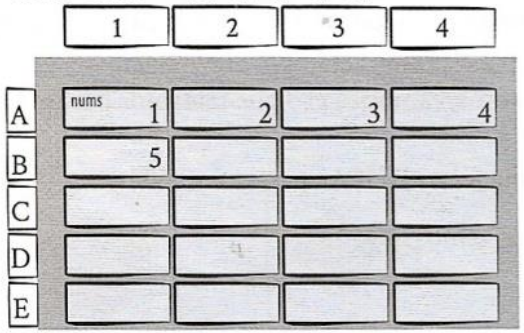
Listas – Implementación estática: Arrays (Estructura de datos lineal)

- Ejemplo: un array (tamaño 6) de números enteros.
- Un entero toma 4 bytes en espacio de memoria.
- Si se conoce la dirección inicial de un array y el índice de un elemento, puede calcular fácilmente su dirección

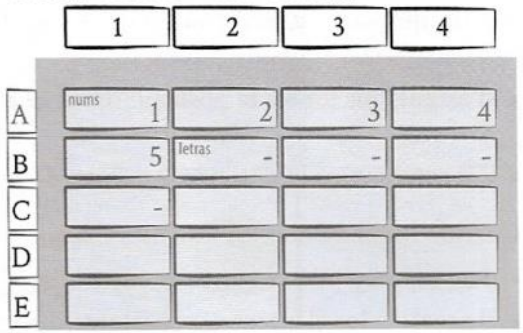
Dirección de memoria	100	104	108	112	116	120
Elementos	**21**	**33**	**-5**	**8**	**12**	**2**
Posición respecto al array	0	1	2	3	4	5

Listas – Implementación estática: Arrays (Estructuras de datos lineal)

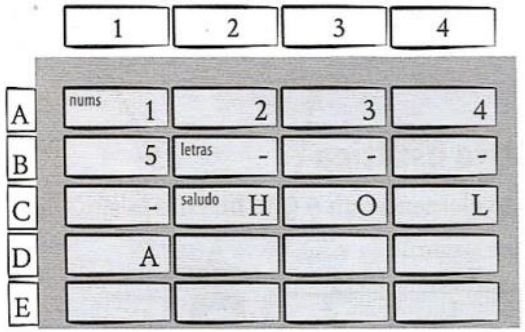
```
int[] nums = [1, 2, 3, 4, 5];
```



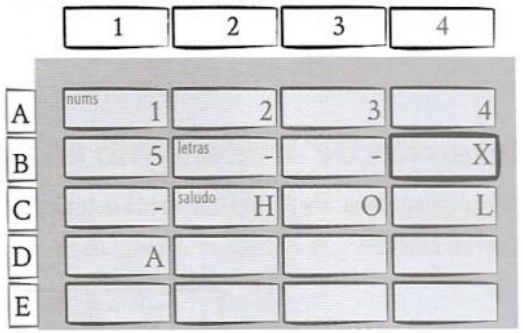

```
char[] letras = new char[4];
```




```
char[] saludo = "HOLA";
```

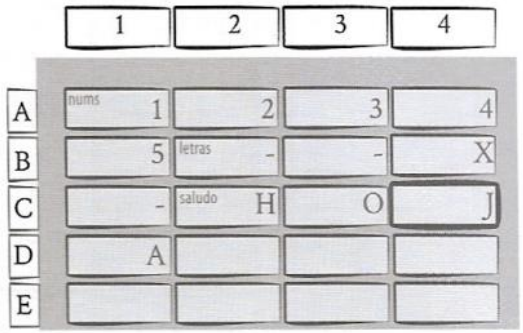



```
letras[2] = 'X';
```

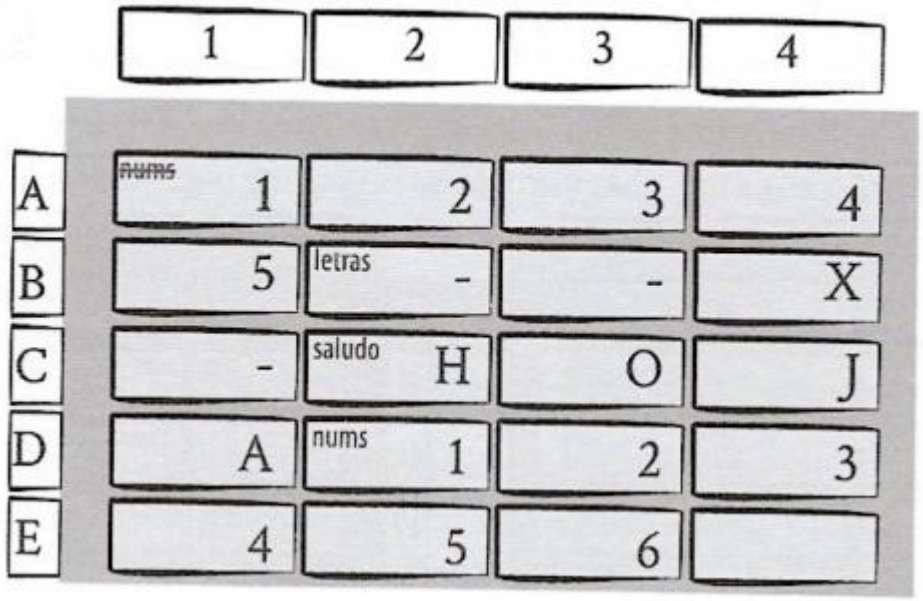



```
nums[2] = 8;
```


```
saludo[2] = 'J';
```



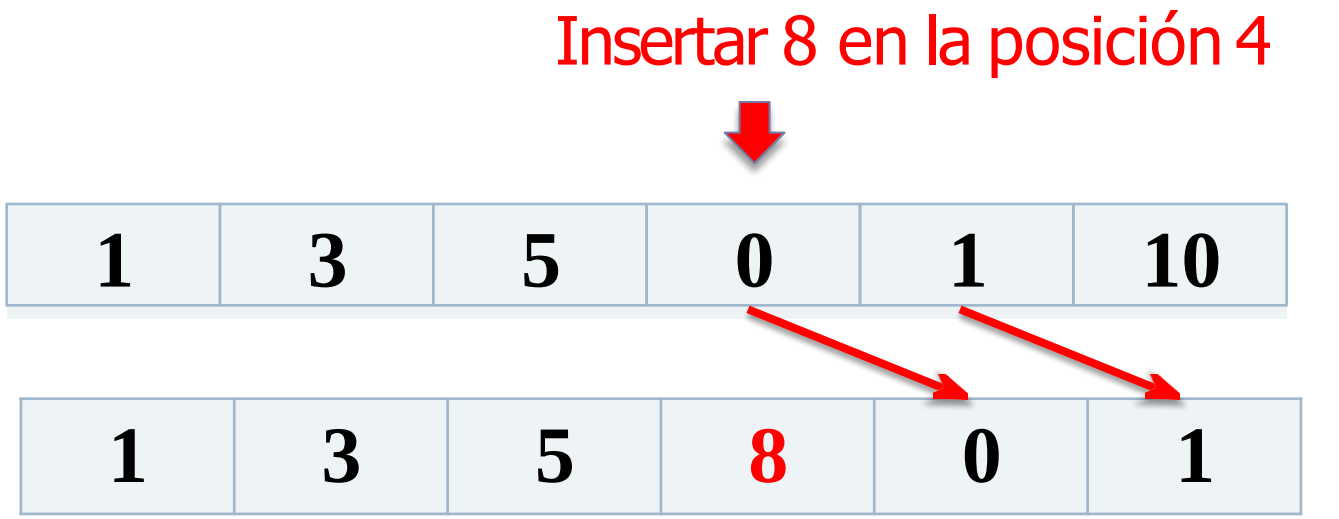

```
nums += 6;
```



Arrays

- **Ventajas:** acceso directo a cada elemento del array a través de su índice.

- **Desventajas:** Las operaciones de borrado e inserción pueden requerir que algunos elementos se muevan. Estas operaciones son **lentas**.



Perdemos el último elemento: 10

Arrays

- **Desventajas:**
  - Los arrays tienen tamaños fijos (no se pueden modificar en tiempo de ejecución)
  - ¡Algunas veces el tamaño puede ser insuficiente!
  - Un tamaño muy grande puede conllevar a un uso ineficiente de la memoria
  - A veces no se puede saber el tamaño necesario para tu problema.

Listas – Implementación dinámica: Listas Enlazadas (Estructuras de datos lineal)

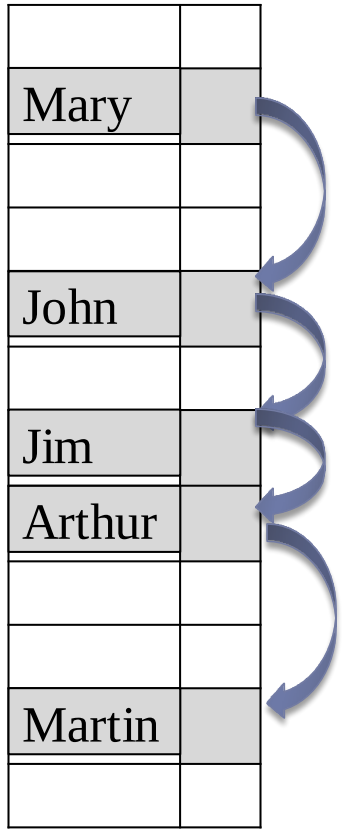
- Los elementos no se almacenan en posiciones de memoria consecutivas, sino también con espacios entre ellos.
- Puede crecer o reducirse en tiempo de ejecución
- Uso eficiente de la memoria (solo pueden ocupar la memoria necesaria)
- Alternativa a los arrays para implementar TAD lineales.

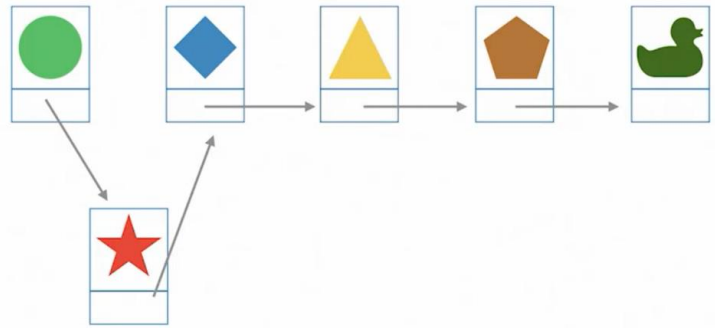
Array

memoria

Lista Enlazada

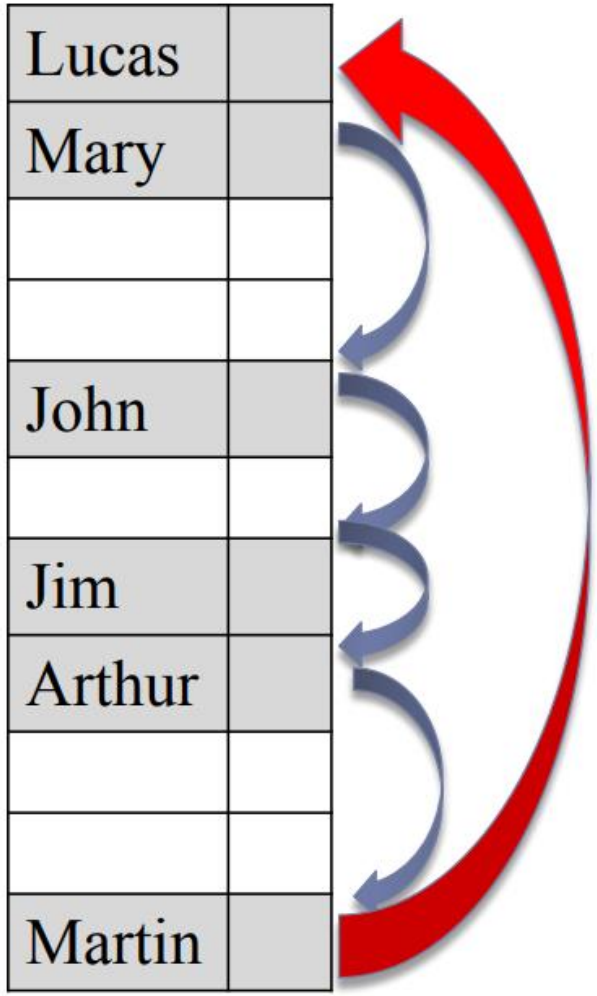
memoria



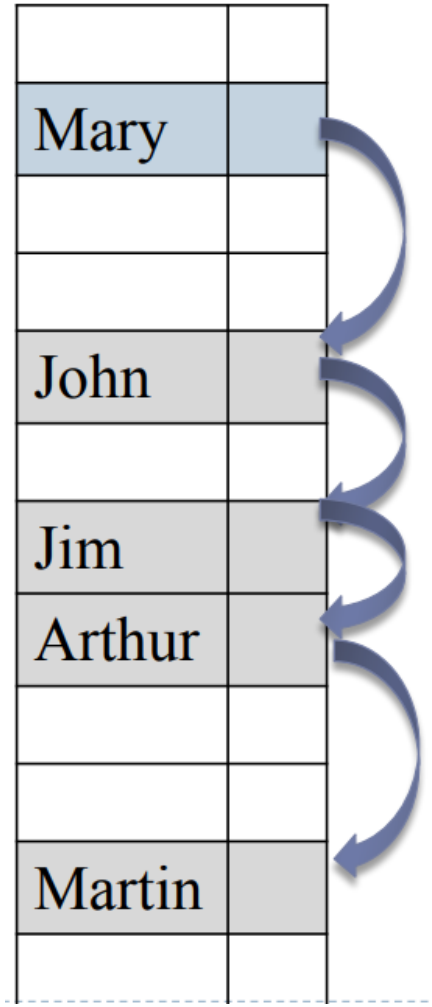


Listas – Implementación dinámica: Listas Enlazadas (Estructuras de datos lineal)

Los espacios en la memoria permiten que el orden físico y lógico en la memoria pueda ser diferente

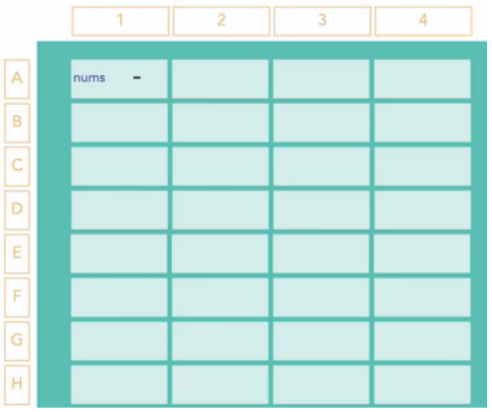


Cada ubicación no sólo almacena (una referencia) un objeto, sino también la referencia (dirección) a su sucesor en la Lista



Listas – Implementación dinámica: Listas Enlazadas (Estructuras de datos lineal)

```
List<int> nums = new List();
```




```
List<char> letras = new List(4);
```

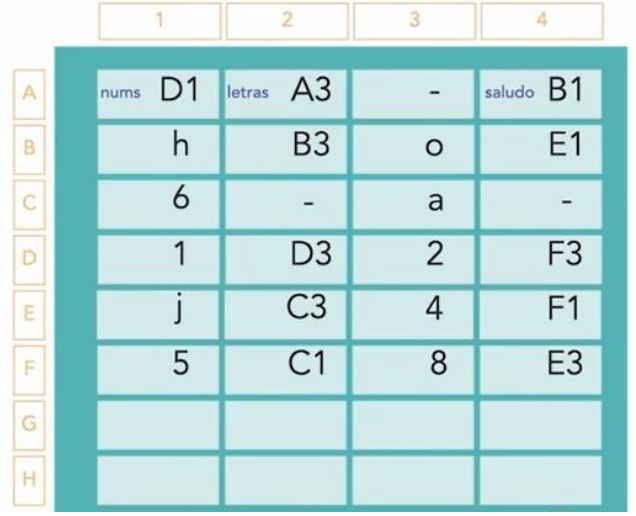

```
List<char> saludo = "hola";
```


```
nums.addAll({1,2,3,4,5})
```


```
nums.set(2,8);
```

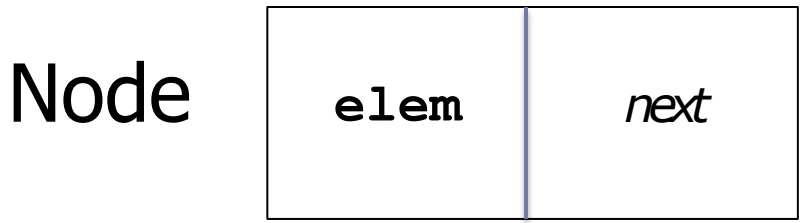

```
saludo.set(2,'j');
```


```
nums.adds(6);
```



Listas – Implementación de un TAD lineal utilizando una estructura dinámica

- Necesitamos definir una clase, **Node**, para representar cada ubicación. La clase **Node** tiene dos atributos:
  - **elem**: es la referencia a un elemento almacenado en la Lista. Es decir, almacena la dirección de memoria donde se almacena el elemento. El tipo de datos de **elem** debe ser del mismo tipo que los elementos de la Lista
  - **next**: es la referencia al nodo que almacena el siguiente elemento en la Lista. Su tipo de datos debe ser **Node**



Class Snode (simplemente node)

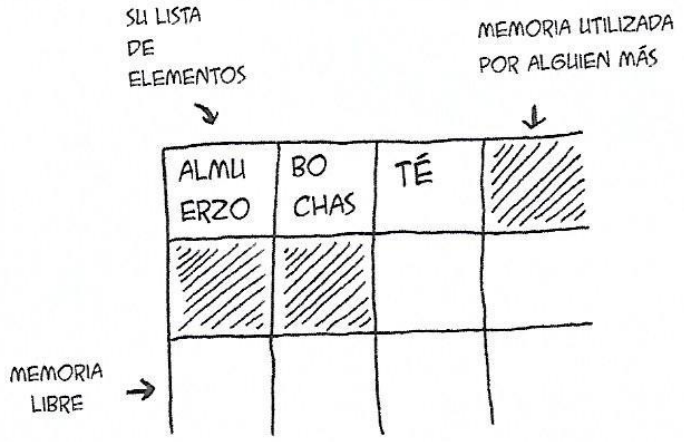
```
public class SNode {

    public String elem;
    public SNode next;

    public SNode(String e) {
        elem = e;
    }
}
```

Permite almacenar un objeto de tipo String. Sin embargo, puede definir un node que almacene cualquier tipo de objeto.

Estructuras de datos lineales

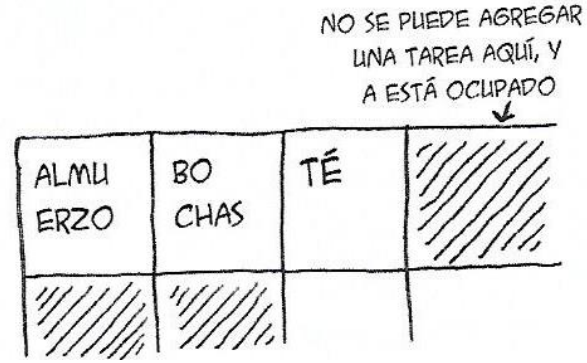
Asume que estas elaborando una aplicación para administrar tus tareas pendientes

¿Se debe usar un arreglo o listas enlazadas?



Usar un arreglo significa que todas las tareas se almacenan contiguamente en la memoria

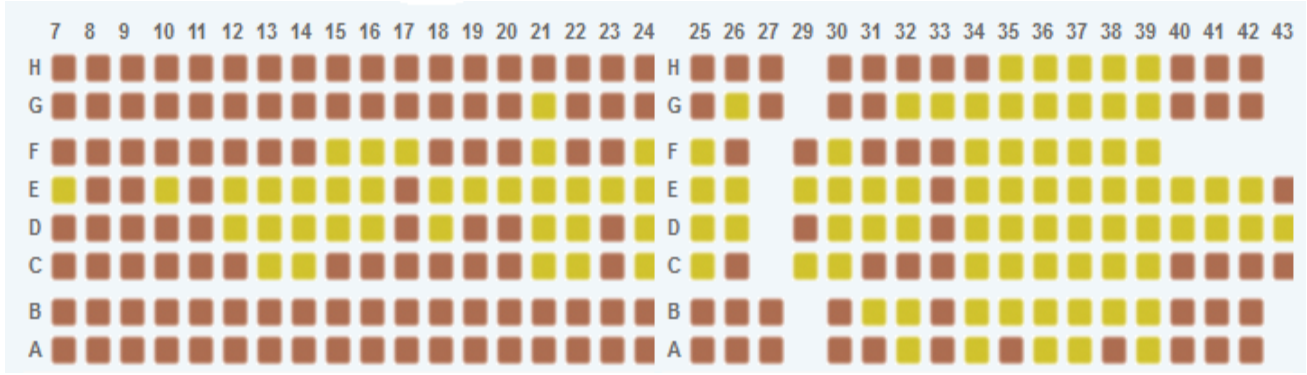
Ahora asume que quieres agregar una cuarta tarea !pero el próximo cajón está ocupado por las cosas de alguien más!. Hay que cambiarlas en un lugar donde todos quepan, qué molestia



En un arreglo el tamaño está definido de antemano, agregar un nuevo elemento es lento. Una solución es reservar asientos o espacios de memoria de antemano, los cuales traen como desventaja lo siguiente:

- *Pueden ser demasiados, y se desperdicia espacio.*
- *En algún momento puede volverse a necesitar más espacios de memoria contiguas.*

Es como ir al cine con varios amigos y no encontrar asientos juntos…

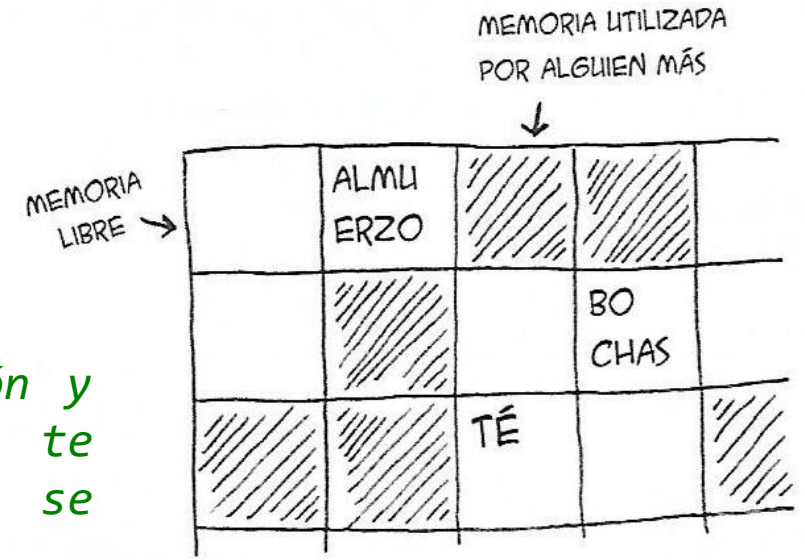


Estructuras de datos lineales

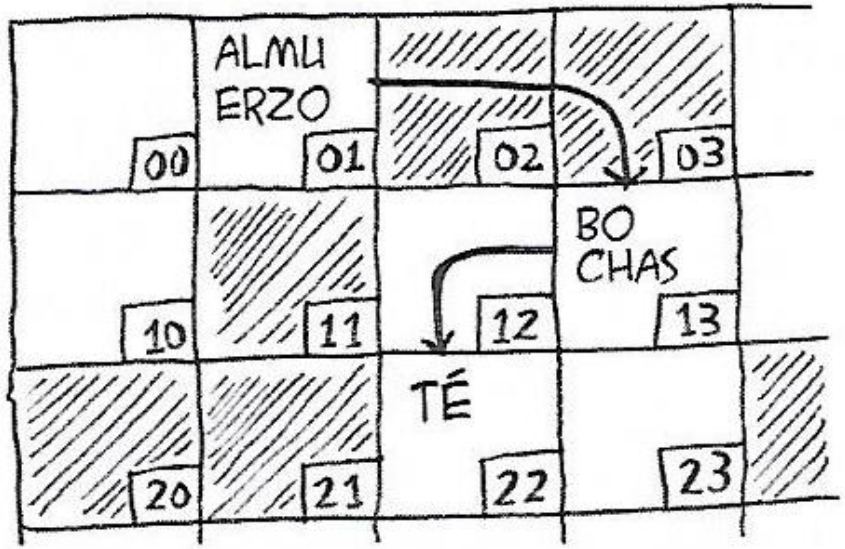
Listas enlazadas

Con las listas enlazadas tus elementos pueden estar en cualquier lugar de la memoria

Cada elemento guarda la dirección del próximo elemento de la lista. Un grupo de direcciones de memoria aleatorias están enlazadas entre sí



Es como la búsqueda del tesoro, vas a la primera dirección y dice, el próximo elemento se encuentra en la dirección 123, te diriges a la dirección 123 y dice el próximo elemento se encuentra en la dirección 847 y así sucesivamente



Estas buscando 10,000 casillas para un arreglo. La memoria tiene 10,000 casillas disponibles, pero no juntas. La única forma de utilizar las 10,000 casillas es con listas enlazadas

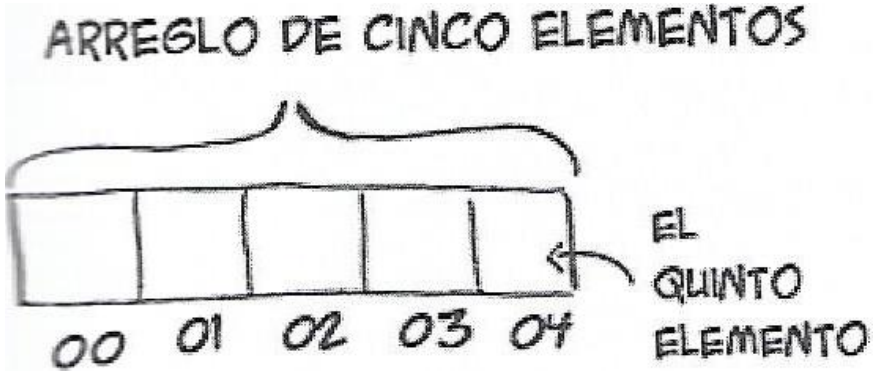
Entonces cuál es la ventaja de los arreglos?

Estructuras de datos lineales

El problema con las listas enlazadas es la lectura de sus elementos. No conoces la dirección de cada elemento. Supón que quieres leer el último elemento de la lista enlazada, tienes que ir al elemento #1 para obtener la dirección del elemento #2, al elemento #2 para obtener la dirección del elemento #3, y así sucesivamente hasta llegar al último.

Arreglos

Los arreglos son diferentes en ese sentido, conoces la dirección de cada elemento de tu arreglo. Si tu arreglo contiene 5 elementos y conoces que comienza en la dirección 00 ¿Cuál es la dirección para el 5to elemento? Rpta: 04



Los arreglos son ideales si quieres obtener elementos en posiciones aleatorias, porque puedes buscar cualquier elemento en forma instantánea

	ARREGLOS	LISTAS
LECTURA	O(1)	O(N)

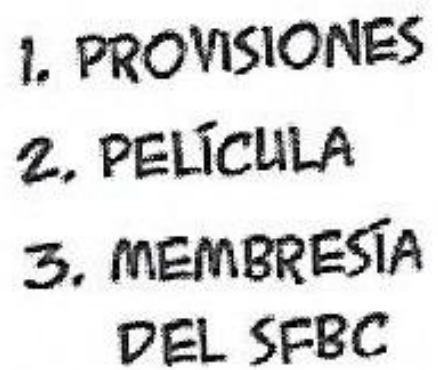
Notación Big O

Estructuras de datos lineales

Ejercicio

Supón que estás creando una aplicación para mantener el control de tus finanzas.

Cada día, anotas todo el dinero que gastaste. Al final del mes, revisas tus compras y sumas cuánto has gastado. Así que tienes muchas inserciones y unas pocas lecturas ¿Deberías utilizar un arreglo o una lista enlazada?



Inserción en el medio de una lista

Ahora en la aplicación quieres añadir tareas en el orden en que deberían ser completadas

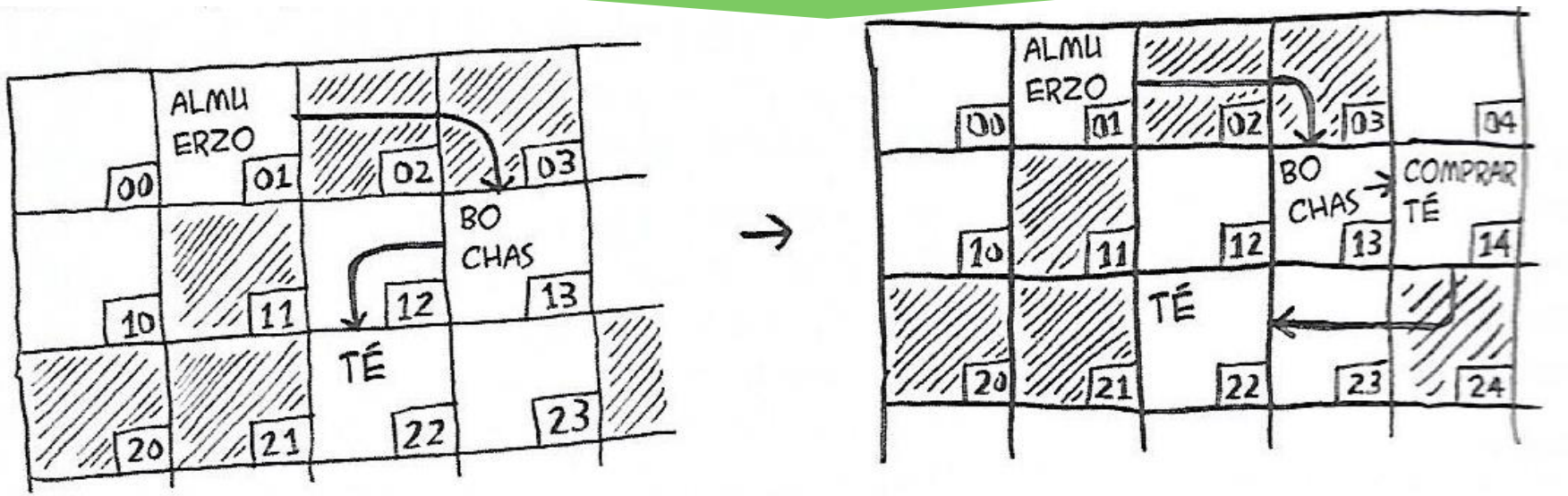
¿Qué es mejor si quieres insertar elementos en el medio: arreglos o listas?

Desordenada

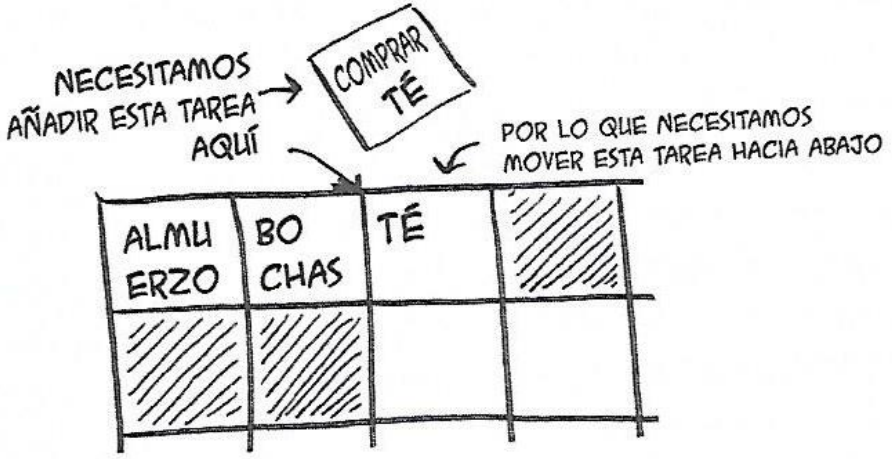
Ordenada

Estructuras de datos lineales

Con las listas es tan sencillo como cambiar cuál es el elemento anterior al que apunta



Pero para los arreglos, tienes que desplazar el resto de los elementos hacia abajo



Y si no existe espacio, !puede que tengas que copiar todos los elementos a una nueva zona de memoria!

Las listas son mas efectivas si tienes que insertar un elemento en el medio

Estructuras de datos lineales

¿Qué pasa si quieres borrar un elemento?

Nuevamente las listas son mejores porqué solo tienes que cambiar cuál es el elemento previo al que apunta.

Con arreglos todo tiene que ser desplazado cuando eliminas un elemento

A diferencia de las inserciones, las eliminaciones siempre funcionan (una inserción puede fallar si no hay espacio suficiente en la memoria, pero siempre es posible borrar un elemento)

Aquí tienes los tiempos de ejecución más comunes para arreglos y listas enlazadas

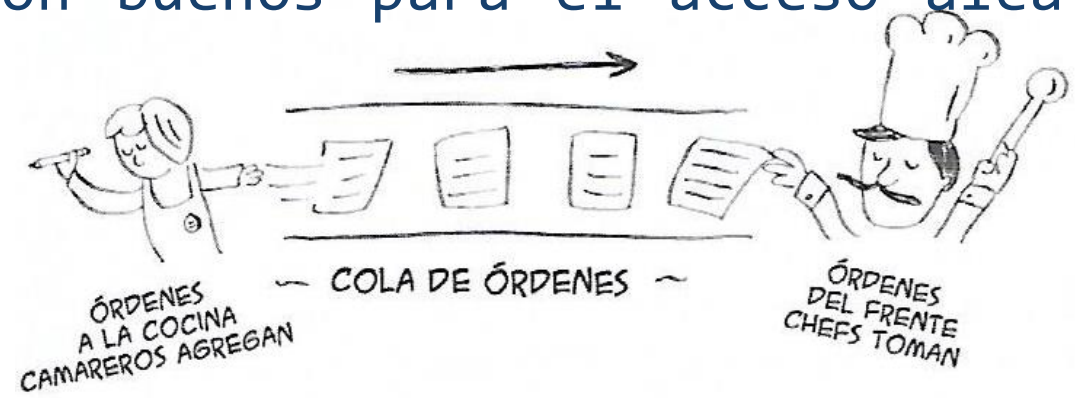
	ARREGLOS	LISTAS
LECTURA	O(1)	O(N)
INSERCIÓN	O(N)	O(1)
ELIMINACIÓN	O(N)	O(1)

Vale la pena mencionar que la inserción y eliminación son O(1) solamente si puedes acceder inmediatamente al elemento en cuestión. Es una práctica común mantener una referencia al primer y último elemento de una lista enlazada, de tal forma que tome un tiempo O(1) eliminar uno de ellos.

Estructuras de datos lineales

Ejercicio 1

Supongamos que estás creando una aplicación para recibir órdenes de los clientes en un restaurante. Su aplicación necesita almacenar una lista de órdenes. Los meseros se mantienen agregando órdenes a esta lista, y los chefs toman las órdenes de la lista para preparar los platos. Es una cola ordenada: Los meseros agregan órdenes al final de la cola, y el chef toma la primera orden de la cola y la cocina.
¿Usarías un arreglo o una lista enlazada para implementar esta cola? (Sugerencia: las listas enlazadas son buenas para insertar / eliminar, y los arreglos son buenos para el acceso aleatorio. ¿Cuál utilizarías aquí?)



Estructuras de datos lineales

Ejercicio 2

Hagamos un experimento mental. Supongamos que Facebook mantiene una lista de nombres de usuario. Cuando alguien intenta iniciar sesión en Facebook, se realiza una búsqueda de su nombre de usuario. Si su nombre está en la lista de nombres de usuario, pueden iniciar sesión. Las personas inician sesión en Facebook bastante a menudo, por lo que hay muchas búsquedas en esta lista de nombres de usuario. Supongamos que Facebook usa la búsqueda binaria para buscar en la lista. La búsqueda binaria necesita acceso aleatorio: debe poder llegar al centro de la lista de nombres de usuario al instante. Sabiendo esto, ¿implementaría la lista como un arreglo o una lista enlazada?

Estructuras de datos lineales

Ejercicio 3

Las personas se crean perfiles en Facebook frecuentemente. Supón que decides usar un arreglo para almacenar la lista de usuarios. ¿Cuáles son las desventajas de usar un arreglo cuando se realiza una inserción? En particular, asume que estás utilizando la búsqueda binaria para buscar los nombres de usuario cuando intentan acceder al sitio. ¿Qué sucede cuando añades nuevos usuarios al arreglo?

Estructuras de datos lineales

Ejercicio 4

En realidad, Facebook no utiliza ni un arreglo ni una lista enlazada para almacenar información del usuario. Consideremos una estructura de datos híbrida: un arreglo de listas enlazadas. Tienes un arreglo con 26 posiciones. Cada posición apunta a una lista enlazada. Por ejemplo, la primera posición en el arreglo apunta a una lista enlazada que contiene todos los nombres de usuario que comienzan con “a”. La segunda posición apunta a una lista enlazada que contiene todos los nombres de usuario que comienzan con “b”, y así sucesivamente.

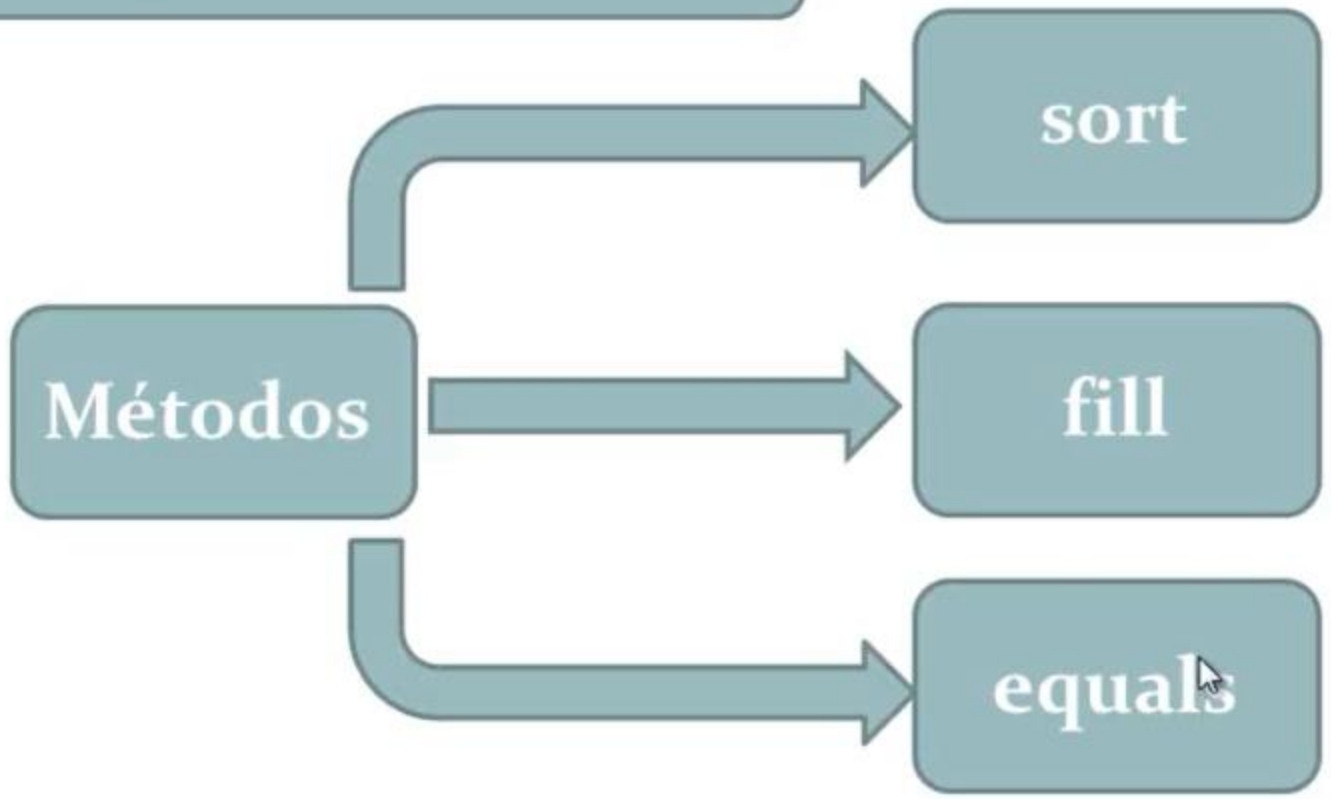
Suponga que Alfonso C se registra en Facebook y quieres agregarlo a la lista. Ud. va a la primera posición del arreglo, y agregas Alfonso C al final de lista enlazada que se encuentra en dicha posición. Ahora, suponga que desea buscar Zoila H. En este caso vas a la posición 26, que apunta a una lista enlazada con todos los nombres que empiezan con Z. Compara esta estructura de datos híbrida con los arreglos y listas enlazadas. ¿Es más lento o más rápido que cada uno para buscar e insertar? No tiene que dar tiempos de ejecución de Big O, solo si la nueva estructura de datos sería más rápida o más lenta.

Memoria Estática y la Clase Arrays de Java

Memoria Estática

Qué es Memoria Estática: Espacio de memoria que no se modifica al menos en tiempo de ejecución.

import java.util.Arrays;



La clase Arrays

La Clase Arrays en el paquete java.util tiene métodos estáticos útiles para manipular arreglos

Nombre del método	Descripción
binarySearch (arreglo, valor)	Devuelve el índice del valor dado en una matriz ordenada (ó < 0 si no se encuentra)
copyOf (arreglo, longitud)	Devuelve una nueva copia de un arreglo
equals (arreglo1, arreglo2)	Devuelve verdadero si los dos arreglos contienen los mismos elementos en el mismo orden
fill (arreglo, valor)	Establece cada elemento del arreglo al valor dado
sort(arreglo)	Organiza los elementos en orden
toString(arreglo)	Devuelve un String que representa el arreglo, como "[10, 30, -25, 17]"

Sintaxis: Arrays.nombredelmetodo(parametros)

Arrays.toString

- Arrays.toString acepta un arreglo como parámetro y devuelve una representación String de sus elementos..

```
int[] e = {0, 2, 4, 6, 8};
e[1] = e[3] + e[4];
System.out.println("e es " + Arrays.toString(e));
```

Salida:

e es [0, 14, 4, 6, 8]

–Debe importar java.util.*;

Memoria Estática y la Clase Arrays de Java

Escribir el siguiente programa en Java

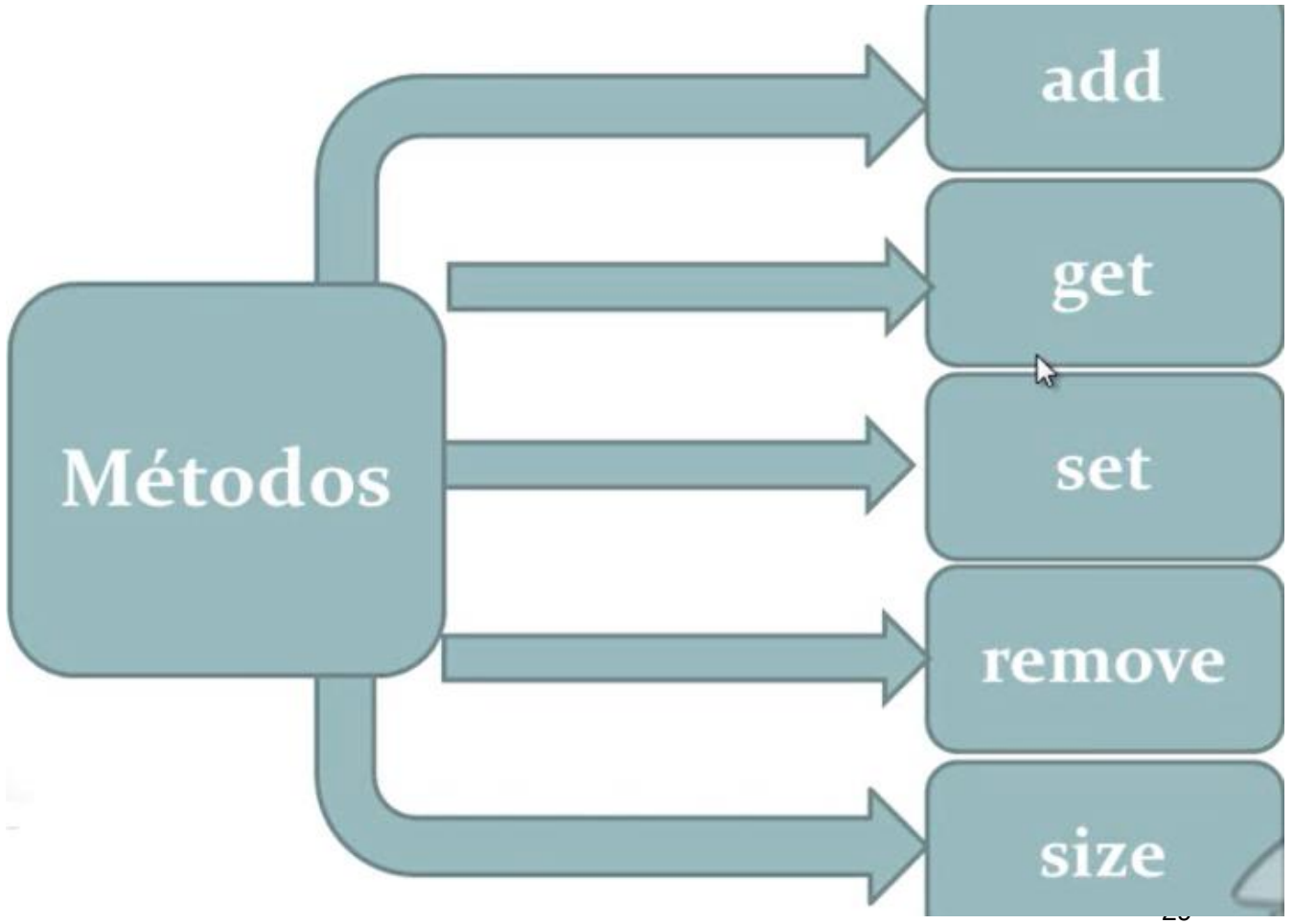
```
import java.util.Arrays;
import javax.swing.JOptionPane;
public class AED_Programa5 {
    public static void main(String[] args) {
        int[] numeros = new int[5];
        String[] palabras = new String[5];
        int[] aleatorio = new int[numeros.length];
        for(int i=0;i<5;i++){
            numeros[i]=Integer.parseInt(JOptionPane.showInputDialog("Ingrese el elemento del indice "+i));
        }
        System.out.println("Los datos sin ordenar son:");
        for(int elemento:numeros){
            System.out.print("["+elemento+"]");
        }
        System.out.println("\nLos datos ordenados son: ");
        Arrays.sort(numeros);//Aqui ordenamos al arreglo números
        for(int elemento:numeros){
            System.out.print("["+elemento+"]");
        }
        Arrays.fill(palabras,"Bienvenido a la clase");//Poblando el arreglo palabras
        System.out.println("\nElementos del arreglo palabras");
        for(String elemento:palabras){
            System.out.println("["+elemento+"]");
        }
        //Copiando los elementos del arreglo numeros a aleatorio
        System.arraycopy(numeros, 0, aleatorio, 0, numeros.length);
        System.out.println("Elementos del arreglo aleatorio");
        for(int elemento:aleatorio){
            System.out.print("["+elemento+"]");
        }
    }
}
```

Memoria Dinámica y la Clase ArraysList de Java

Memoria Dinámica

Qué es Memoria Dinámica: Espacio de memoria la cual su tamaño puede variar durante la ejecución del programa.

```
import java.util.ArrayList;
```



Memoria Dinámica y la Clase ArraysList de Java

Escribir el siguiente programa en Java

```
import java.util.ArrayList;
import javax.swing.JOptionPane;
public class AED_Programa6 {
    public static void main(String[] args) {
        // TODO code application logic here
        ArrayList<String> cadenas=new ArrayList();//Un objeto o instancia de tipo ArrayList
        String frase, respuesta;
        do{
            frase = JOptionPane.showInputDialog("Ingresa la frase");
            cadenas.add(frase);//Agregando una frase a la lista
            respuesta = JOptionPane.showInputDialog("¿Deseas ingresar otra frase (SI/NO)?");
        }while(!respuesta.equalsIgnoreCase("NO"));
        //mostrando la lista con for clásico
        System.out.println("Frases originales");
        for(int i=0;i<cadenas.size();i++){
            System.out.println(cadenas.get(i));
        }
        cadenas.set(1, "Se modificó");//cambiando una frase
        System.out.println("Frases modificadas");
        for(String elemento:cadenas){
            System.out.println(elemento);
        }
        cadenas.remove(0);//borrando la primera frase
        System.out.println("Frases que quedan");
        for(String elemento:cadenas){
            System.out.println(elemento);
        }

    }
}
```

Clase ArrayList de Java

Escribir el siguiente programa en Java

```java
package Semana1;
import java.util.ArrayList;
public class AED_Programa4 {
    public static void main(String[] args) {
        // crea un nuevo objeto ArrayList de objetos String con una capacidad inicial de 10
        ArrayList<String> items = new ArrayList();
        items.add("red"); // adjunta un elemento a la lista
        items.add(0, "yellow"); // inserta "yellow" en el índice 0
        // encabezado
        System.out.print("Muestra contenido de lista con ciclo for con contador:");
        // muestra los colores en la lista
        for (int i = 0; i < items.size(); i++)
            System.out.printf(" %s", items.get(i));
        // muestra los colores con el método display
        display(items,"%nMostrar contenido de la lista con for mejorado:");
        items.add("green"); // agrega "green" al final de la lista
        items.add("yellow"); // agrega "yellow" al final de la lista
        display(items, "Lista con dos nuevos elementos:");
        items.remove("yellow");// elimina el primer "yellow"
        items.remove("yellow");// elimina el primer "yellow"
        display(items, "Eliminar primera instancia de yellow:");
        items.remove(1); // elimina elemento en índice 1
        display(items, "Eliminar segundo elemento de la lista (green):");
        // Verifica si hay un elemento en la lista
        System.out.printf("\"red\" está %sen la lista%n",items.contains("red") ? "": "no ");
        // muestra el número de elementos en la lista
        System.out.printf("Tamaño: %s%n", items.size());
    }
    // muestra los elementos de ArrayList en la consola
    public static void display(ArrayList<String> items, String header)
    {
        System.out.printf(header); // mostrar encabezado
        // muestra cada elemento en items
        for (String item : items)
            System.out.printf(" %s", item);
        System.out.println();
    }
}
```

Clase LinkedList de Java

Escribir el siguiente programa en Java

```
import java.util.LinkedList;
public class AED_Programa7 {
    public static void main(String[] args) {
        LinkedList<Integer> lista=new LinkedList();
        lista.add(10);
        lista.add(20);
        lista.add(-34);
        lista.add(1);
        lista.add(16);
        int i=0;
        while(i<lista.size()){
            System.out.print("["+lista.get(i)+"]->");
            i++;
        }
        lista.remove(2);
        lista.add(3,100);
        System.out.println();
        i=0;
        while(i<lista.size()){
            System.out.print("["+lista.get(i)+"]->");
            i++;
        }
        lista.sort(null);
        System.out.println();
        i=0;
        while(i<lista.size()){
            System.out.print("["+lista.get(i)+"]->");
            i++;
        }
    }
}
```

Clase LinkedList de Java

Escribir los siguientes programas en Java

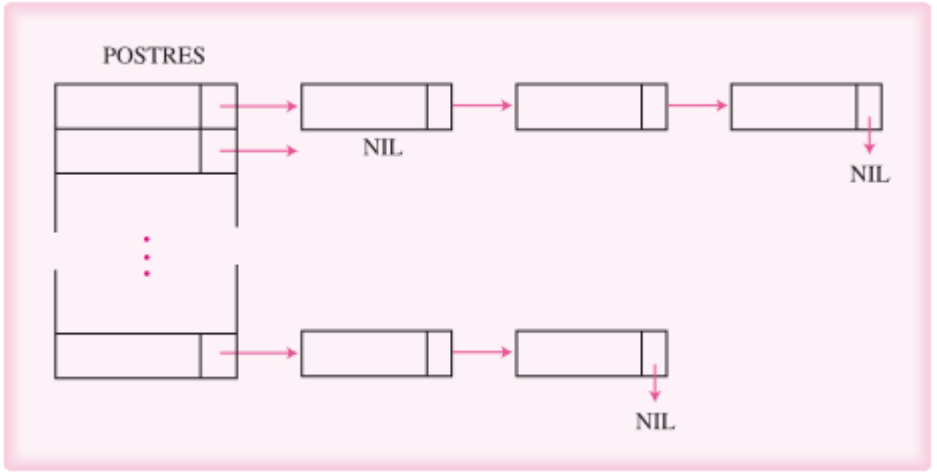
```java

import java.util.*;

public class PruebaLinkedList {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        LinkedList<String> personas=new LinkedList<String>();

        personas.add("Pepe");

        personas.add("Sandra");

        personas.add("Ana");

        personas.add("Laura");

        System.out.println(personas.size());

        for (String persona : personas){

            System.out.println(persona);
        }


    }

}

```

```java
package Semana1;
import java.util.LinkedList;
public class AED_Programa3 {
    public static void main(String[] args) {
        LinkedList<String> lista= new LinkedList();
        lista.add("Palabra1");
        lista.add("Palabra2");
        lista.add("Palabra3");
        lista.add("Palabra4");
        lista.removeLast();
        System.out.println("El tamaño de la lista es : "+lista.size());
        for(String elemento:lista){
            System.out.println(elemento);
        }
    }


}
```

Arreglos y listas enlazadas

Ejercicio Propuesto

Se ha definido la siguiente estructura de datos:



En el arreglo postres se almacenan nombres de postres ordenados alfabéticamente a su vez cada elemento del arreglo tiene una lista de todos los ingredientes que requieren dichos postres. Escriba un programa que:

- *Dado el nombre de un postre imprima la lista de todos sus ingredientes*
- *Dado el nombre de un postre inserte nuevos ingredientes a su correspondiente lista*
- *Dado el nombre de un postre elimine alguno de sus ingredientes*
- *Dé de alta/baja un postre con todos sus ingredientes*

Agenda

- ArrayLists
- La interface Comparable
- Lists, Sets Maps, Iterators
- Estudio de caso: Comparación de Vocabulario

Ejercicio

- Escriba un programa que lea un archivo y muestre las palabras de ese archivo como una lista.
  - Primero muestre todas las palabras.
  - Luego muestre todos los plurales (que terminan en "s") en mayúscula.
  - Luego muéstrelos en orden inverso.
  - Luego muéstrelos con todas las palabras plurales eliminadas.

- ¿Deberíamos resolver este problema usando un arreglo?
  - ¿Por qué sí o por qué no?

Solución ingenua

```
String[] allWords = new String[1000];
int wordCount = 0;

Scanner input = new Scanner(new File("data.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords[wordCount] = word;
    wordCount++;
}
```

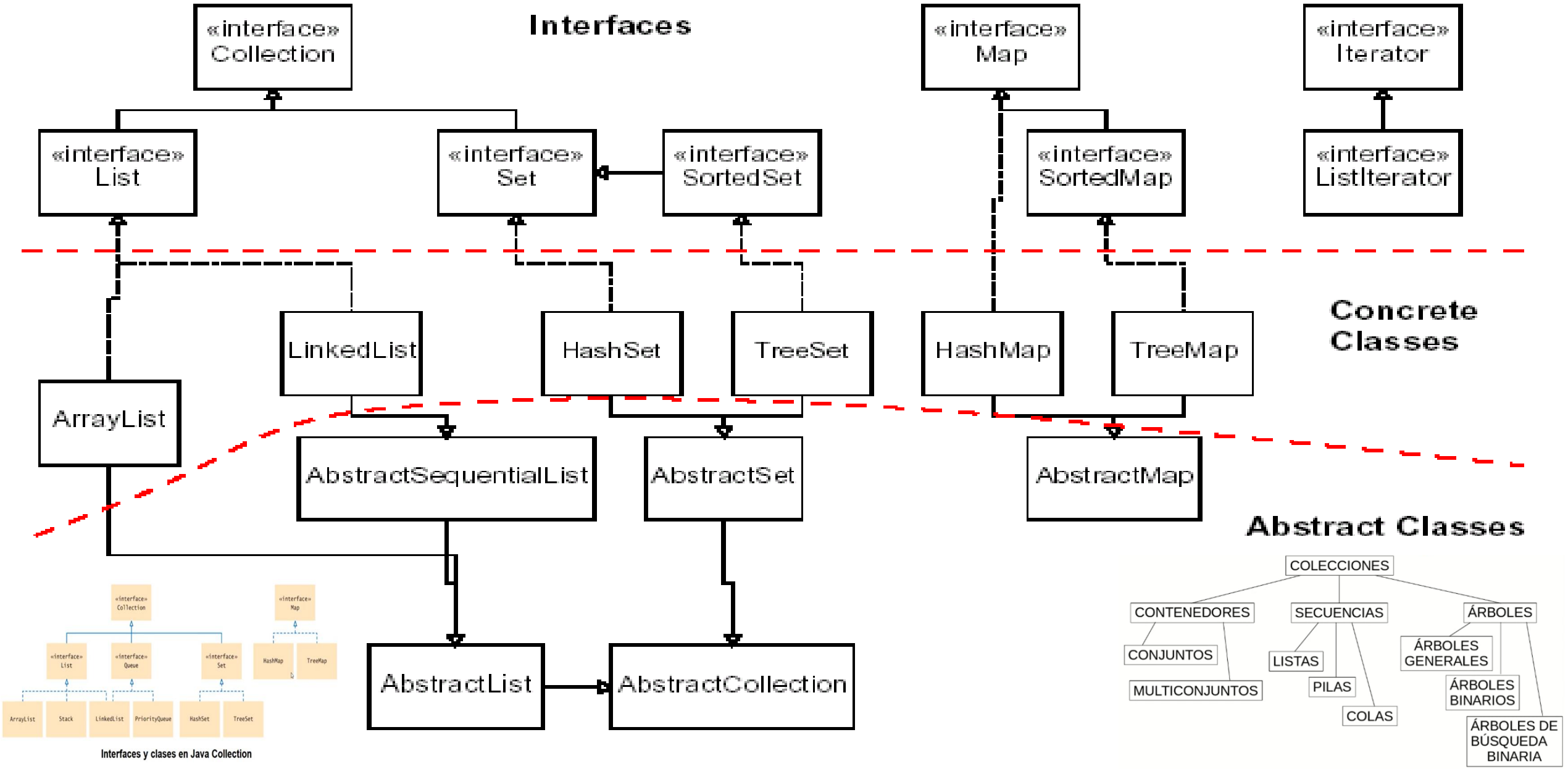
- Problema: No sabe cuántas palabras tendrá el archivo.
  - Difícil de crear un arreglo del tamaño apropiado.
  - Las partes posteriores del problema son más difíciles de resolver.
- Afortunadamente, hay otras formas de almacenar datos además de hacerlo en un arreglo.

Colecciones

- **Colecciones**: un objeto que almacena datos; también conocido como "estructura de datos"
  - Los objetos almacenados son llamados **elementos**
  - Algunas colecciones mantienen un orden; algunas permiten duplicados
  - Típicas operaciones: *add*, *remove*, *clear*, *contains* (search), *size*
  - Ejemplos encontrados en las bibliotecas de clases Java:
    - `ArrayList`, `LinkedList`, `HashMap`, `TreeSet`, `PriorityQueue`
  - Todas las colecciones están en el paquete java.util

```
import java.util.*;
```

Marco de colecciones Java

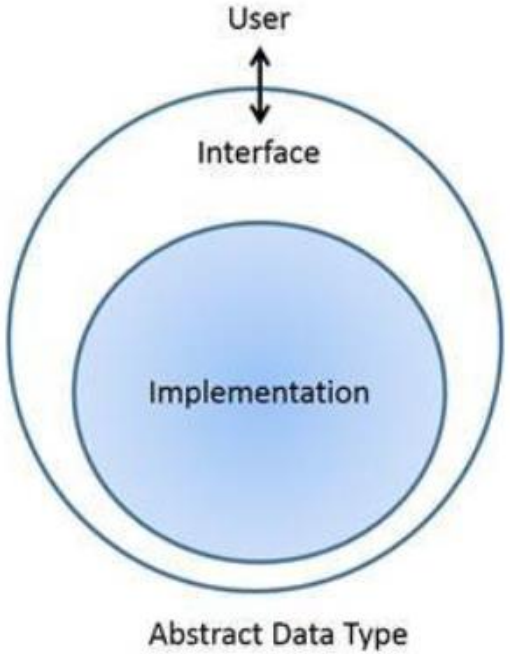


Interfaces y clases en Java Collection

Introducción a las Estructuras de Datos

Tipo abstracto de datos (TAD ó ADT en Inglés) : una especificación de una colección de datos y las operaciones que se pueden realizar en ellos.

- Describe *lo que* hace una colección, no *cómo* lo hace.

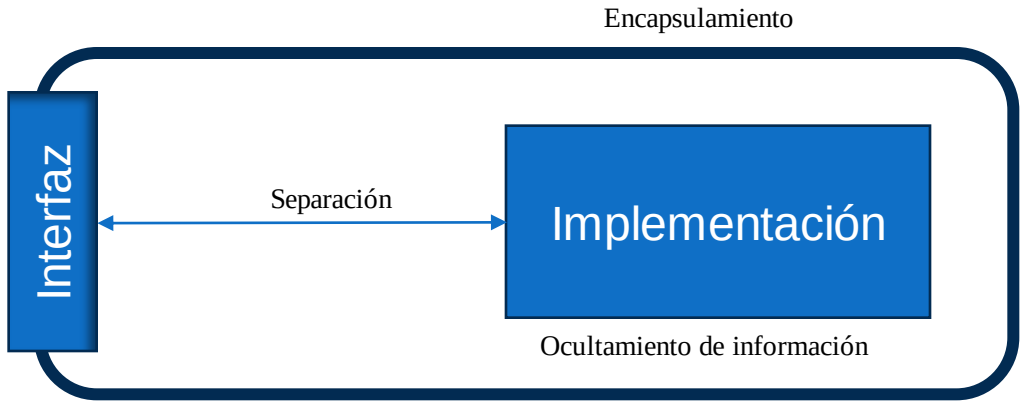


El marco de colección de Java especifica ADT con interfaces:

- `Collection`, `Deque`, `List`, `Map`, `Queue`, `Set`, `SortedMap`

Un ADT se puede implementar de múltiples maneras por clases:

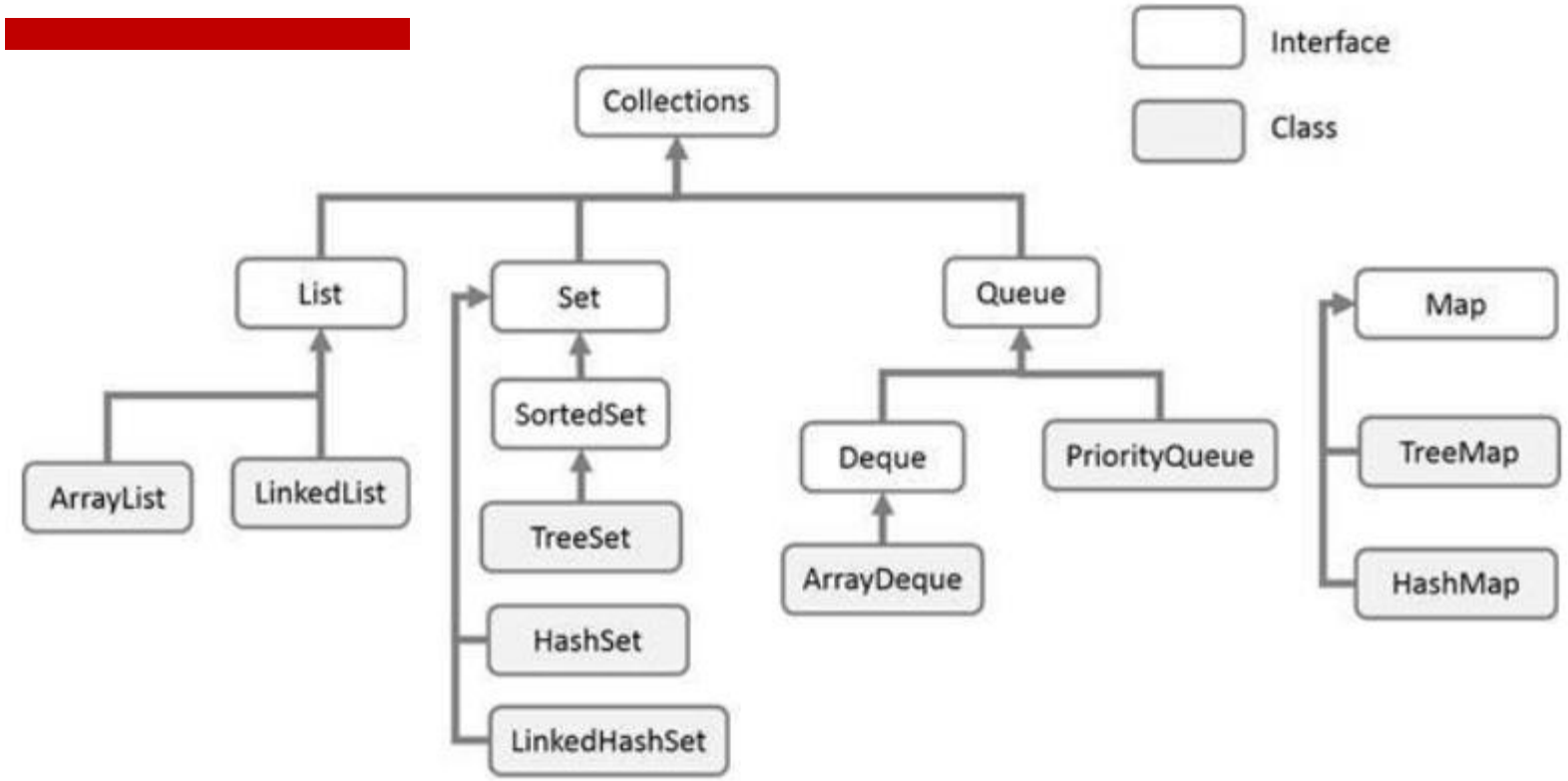
- `ArrayList` y `LinkedList` implementan `List`
- `HashSet` y `TreeSet` implementan `Set`
- `LinkedList` , `ArrayDeque` , etc. implementan `Queue`



ADT (Tipo abstracto de datos)

Un tipo abstracto de datos (ADT) es una descripción lógica de los datos y las operaciones que se permiten sobre ellos. ADT se define desde el punto de vista del usuario de los datos. ADT se preocupa por los posibles valores de los datos y la interfaz expuesta por ellos. ADT no se preocupa por la implementación real de la estructura de datos. Por ejemplo, un usuario quiere almacenar algunos números enteros y encontrar su valor medio. El ADT en este caso admitirá dos funciones, una para sumar números enteros y otra para obtener el valor medio. El ADT no habla sobre cómo se implementará exactamente. Pero la estructura de datos que representa el ADT si lo hará

¿Porqué usar el Marco de colecciones de Java?



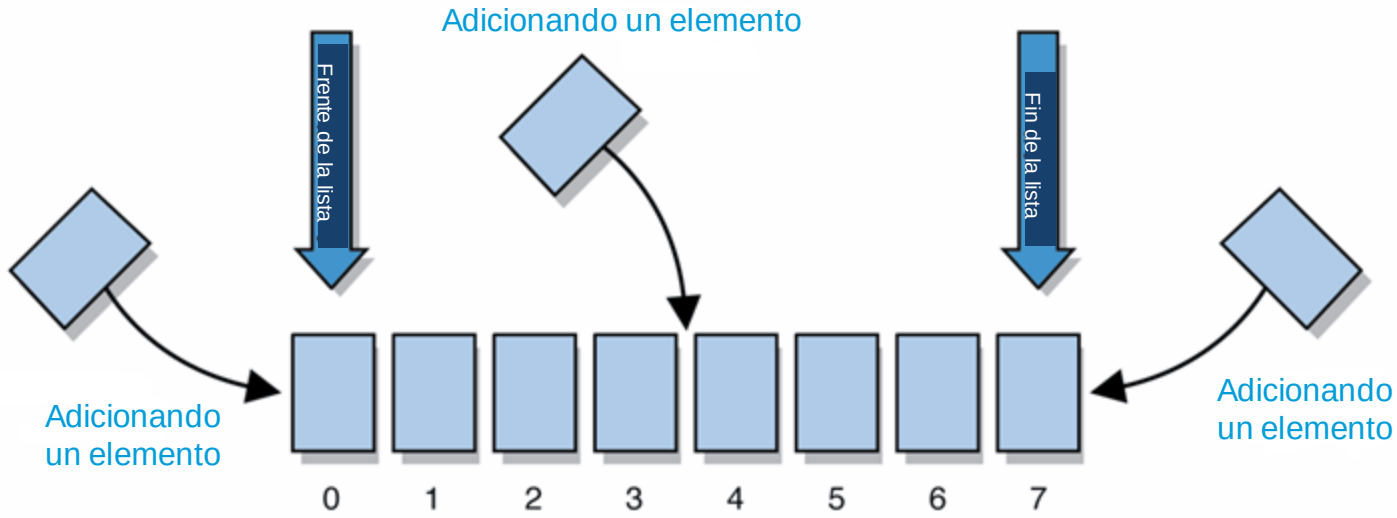
Estructura de datos

Las estructuras de datos son representaciones concretas de datos y se definen desde el punto de vista del programador de datos. La estructura de datos representa cómo se almacenarán los datos en la memoria. Todas las estructuras de datos tienen sus pros y sus contras. Dependiendo del tipo de problema, elegimos la estructura de datos que mejor se adapte a él. Por ejemplo, podemos almacenar datos en una matriz, una lista enlazada, una pila, una cola, un árbol, etc. Nota: - En estas primeras sesiones del curso, estudiaremos varias estructuras de datos y su API, para que el usuario pueda utilizarlos sin conocer su implementación interna.

1. Los programadores no tienen que implementar estructuras de datos y algoritmos básicos repetidamente. De ese modo evita la reinvención de la rueda. Así, el programador puede dedicar más esfuerzo a la lógica empresarial.
2. El marco de colecciones Java es un código bien probado, de alta calidad y alto rendimiento. Su uso aumenta la calidad de los programas.
3. El costo de desarrollo se reduce a medida que se reutilizan las estructuras de datos básicas y los algoritmos que se implementan en el marco de colecciones.
4. Fácil de revisar y comprender los programas escritos por otros desarrolladores, ya que la mayoría de los desarrolladores de Java utilizan el marco de colecciones. Además, está bien documentado.

Listas

- **Lista :** Una colección que almacena una secuencia ordenada de elementos
  - Cada elemento es accesible por un índice basado en 0
  - Una lista tiene un tamaño (número de elementos que se han adicionado)
  - Los elementos se pueden agregar al frente, por la parte posterior o en cualquier otro lado
  - En Java, una lista se puede representar como un objeto ArrayList



Idea de una lista

- En lugar de crear un arreglo, cree un objeto que represente una "lista" de elementos. (inicialmente una lista vacía)

  []

- Puede agregar elementos a la lista.
  - El comportamiento predeterminado es adicionar al final de la lista.

    [hello, ABC, goodbye, okay]

- El objeto lista mantiene seguimiento de los valores de los elementos que se adicionan, su orden, índices y su tamaño total.
  - Piense en un “ArrayList" como un arreglo de objetos de cambio de tamaño automático.
  - Internamente, la lista se implementa utilizando un arreglo y un atributo de tamaño.

Métodos ArrayList

`add(`**valor**`)`	Adiciona valor al final de la lista
`add(`**indice**`, `**valor**`)`	Inserta el valor dado justo antes del índice dado, desplazando los valores posteriores a la derecha
`clear()`	Elimina todos los elementos de la lista
`indexOf(`**valor**`)`	Devuelve el primer índice donde se encuentra el valor dado en la lista (-1 si no se encuentra)
`get(`**indice**`)`	Devuelve el valor en el índice dado
`remove(`**indice**`)`	Elimina/devuelve el valor en el índice dado, desplazando los valores posteriores a la izquierda
`set(`**indice**`, `**valor**`)`	Reemplaza el valor en el índice dado con el valor dado
`size()`	Devuelve el número de elementos en la lista
`toString()`	Devuelve una representación de cadena de la lista como "[3, 42, -7, 15]"

Métodos ArrayList 2

`addAll(`**list**`)` `addAll(`**indice**`, `**list**`)`	Adiciona todos los elementos de la lista dada a esta lista (al final de la lista, o los inserta en el índice dado)
`contains(`**valor**`)`	Devuelve true si el valor dado se encuentra en algún lugar de esta lista
`containsAll(`**list**`)`	Devuelve true si esta lista contiene todos los elementos de la lista dada
`equals(`**list**`)`	Devuelve true si otra lista contiene los mismos elementos
`iterator()` `listIterator()`	Devuelve un objeto utilizado para examinar el contenido de la lista (visto más adelante)
`lastIndexOf(`**valor**`)`	Devuelve el último índice donde se encuentra el valor en la lista (-1 si no se encuentra)
`remove(`**valor**`)`	Encuentra y elimina el valor dado de esta lista
`removeAll(`**list**`)`	Elimina todos los elementos encontrados en la lista dada de esta lista
`retainAll(`**list**`)`	Elimina cualquier elemento que no se encuentre en la lista dada de esta lista
`subList(`**desde**`, `**hasta**`)`	Devuelve la subparte de la lista entre índices desde (inclusivo) hasta (exclusivo)
`toArray()`	Devuelve los elementos en esta lista como un arreglo

Parámetros de tipo (genéricos)

```
ArrayList<Tipo> nombre = new ArrayList<Tipo>();
```

- Al construir un ArrayList, debe especificar el tipo de elementos que contendrá entre < >.
  - Esto se llama un parámetro de tipo o una clase genérica.
  - Permite que la misma clase ArrayList almacene listas de diferentes tipos.

```
ArrayList<String> nombres = new ArrayList<String>();
nombres.add("Javier Antonio");
nombres.add("Vanessa Karina");
```

ArrayList vs. array

- Construcción

```
String[] nombres = new String[5];
ArrayList<String> list = new ArrayList<String>();
```

- Almacenar un valor

```
nombres[0] = "Jessica";
list.add("Jessica");
```

- Recuperar un valor

```
String s = nombres[0];
String s = list.get(0);
```

ArrayList vs. Array 2

- Haciendo algo a cada valor que comienza con "B"

```
for (int i = 0; i < nombres.length; i++) {
    if (nombres[i].startsWith("B")) { ... }
}
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) { ... }
}
```

- Ver si se encuentra el valor "Benson"

```
for (int i = 0; i < nombres.length; i++) {
    if (nombres[i].equals("Benson")) { ... }
}
for (int i = 0; i < list.size(); i++) {
if (list.contains("Benson")) { ... }
}
```

Revisemos nuevamente el ejercicio inicial

- Escriba un programa que lea un archivo y muestre las palabras de ese archivo como una lista.
  - Primero muestre todas las palabras.
  - Luego muestre todos los plurales (que terminan en "s") en mayúscula.
  - Luego muéstrelos en orden inverso.
  - Luego muéstrelos con todas las palabras plurales eliminadas.

Solución de ejercicio (parcial)

```
ArrayList<String> allPalabras = new ArrayList<String>();
Scanner input = new Scanner(new File("palabras.txt"));
while (input.hasNext()) {
    String palabra = input.next();
    allPalabras.add(palabra);
}
System.out.println(allPalabras);
// eliminar todas las palabras plurales
for (int i = 0; i < allPalabras.size(); i++) {
    String palabra = allPalabras.get(i);
    if (palabra.endsWith("s")) {
        allPalabras.remove(i);
        i--;
    }
}
System.out.println(allPalabras);
```

ArrayList como parámetro

```
public static void nombre(ArrayList<Tipo> nombre) {
```

- Ejemplo:

```
// Elimina todas las palabras plurales de la lista dada.
public static void removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            list.remove(i);
            i--;
        }
    }
}
```

- También puede devolver una lista:

```
public static ArrayList<Tipo> nombreMetodo(params)
```

ArrayList de primitivas?

- El tipo que especifique al crear una ArrayList debe ser un tipo de objeto; No puede ser un tipo primitivo.

```
// ilegal: int no puede ser un parámetro de tipo
ArrayList<int> list = new ArrayList<int>();
```

- Pero aún podemos usar ArrayList con tipos primitivos mediante el uso de clases especiales llamadas clases “wrapper” (envoltura) en su lugar.

```
// crea una lista de ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

Clases envoltura de tipos (“wrapper”)

Tipo Primitiva	Tipo Wrapper
int	Integer
double	Double
char	Character
boolean	Boolean

- Una envoltura es un objeto cuyo único propósito es mantener un valor primitivo.
- Una vez que construya la lista, úsela con primitivas como de costumbre:

```
ArrayList<Double> notas = new ArrayList<Double>();
notas.add(13.2);
notas.add(12.7);
...
double miNota = notas.get(0);
```

Ejercicio

- Escriba un programa que lea un archivo lleno de números y muestre todos los números como una lista, luego:
  - Imprime el promedio de los números.
  - Imprime el número más alto y más bajo.
  - Filtra todos los números pares.

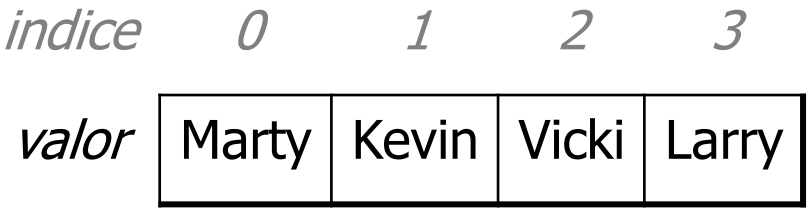
Solución de ejercicio (parcial)

```
ArrayList<Integer> numeros = new ArrayList<Integer>();
Scanner input = new Scanner(new File("numeros.txt"));
while (input.hasNextInt()) {
    int n = input.nextInt();
    numeros.add(n);
}
System.out.println(numeros);
filtrarPares(numeros);
System.out.println(numeros);
...
// Elimina todos los elementos con valores pares de la lista dada.
public static void filtrarPares(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        int n = list.get(i);
        if (n % 2 == 0) {
            list.remove(i);
        }
    }
}
```

Out-of-bounds

- Los índices legales están entre 0 y el tamaño de la lista()- 1
  - Leer o escribir cualquier índice fuera de este rango provocará una excepción IndexOutOfBoundsException.

```
ArrayList<String> nombres = new ArrayList<String>();
nombres.add("Marty");
nombres.add("Kevin");
nombres.add("Vicki");
nombres.add("Larry");
System.out.println(nombres.get(0));        // okay
System.out.println(nombres.get(3));        // okay
System.out.println(nombres.get(-1));       // exception
nombres.add(9, "Aimee");                   // exception
```



ArrayList "misterio"

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 10; i++) {
    list.add(10 * i);   // [10, 20, 30, 40, ..., 100]
}
```

- ¿Cuál es la salida del siguiente código?

```
for (int i = 0; i < list.size(); i++) {
    list.remove(i);
}
System.out.println(list);
```

- Respuesta:

  [20, 40, 60, 80, 100]

ArrayList "misterio“ 2

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 1; i <= 5; i++) {
    list.add(2 * i);   // [2, 4, 6, 8, 10]
}
```

- ¿Cuál es la salida del siguiente código?

```
int size = list.size();
for (int i = 0; i < size; i++) {
    list.add(i, 42);   // adicione 42 en el índice i
}
System.out.println(list);
```

- Respuesta:

```
[42, 42, 42, 42, 42, 2, 4, 6, 8, 10]
```

ArrayList como parámetro

```
public static void nombre(ArrayList<Tipo> nombre) {
```

- Ejemplo:

```
// Elimina todas las palabras plurales de la lista dada.
public static void removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            list.remove(i);
            i--;
        }
    }
}
```

- También puede devolver una lista:

```
public static ArrayList<Tipo> nombreMetodo (params)
```

Ejercicios

- Escriba un método addStars que acepte un arraylist de cadenas como parámetro y coloque un * después de cada elemento.
  - Ejemplo: si un arraylist llamado lista almacena inicialmente:

    [the, quick, brown, fox]
  - Luego de addStars (lista); lo hace almacenar:

    [the, *, quick, *, brown, *, fox, *]

- Escriba un método removeStars que acepte un arraylist de cadenas, y elimine las estrellas (deshaciendo lo que hizo addStars arriba).

Solución de los ejercicios

```
public static void addStars(ArrayList<String> list)
   {
      for (int i = 0; i < list.size(); i += 2) {
          list.add(i, "*");
      }
}
```

```
public static void removeStars(ArrayList<String>
   list) {
      for (int i = 0; i < list.size(); i++) {
          list.remove(i);
      }
}
```

Ejercicio

- Escriba un método “intersect” que acepte dos listas ordenadas de enteros como parámetros y devuelva una nueva lista que contenga solo los elementos que se encuentran en ambas listas.
  - Ejemplo: si las listas llamadas list1 y list2 inicialmente se almacenan:

    [1, **4**, 8, 9, **11**, 15, **17**, **28**, 41, **59**]

    [**4**, 7, **11**, **17**, 19, 20, 23, **28**, 37, **59**, 81]

  - Luego, la llamada intersect(list1, list2) devuelve la lista:

    [4, 11, 17, 28, 59]

Otros ejercicios

- Escriba un método que invierta el orden de los elementos en un ArrayList de cadenas.

- Escriba un método que acepte un ArrayList de cadenas y reemplace cada palabra que termina con una "s" con su versión en mayúsculas.

- Escriba un método que acepte un ArrayList de cadenas y elimine cada palabra de la lista que termine con una "s", sin distinción entre mayúsculas y minúsculas.

Objetos que almacenan colecciones

- Un objeto puede tener un arreglo, una lista u otra colección como atributo.

```
public class Curso {
    private double[] notas;
    private ArrayList<String> nombreAlumno;

    public Curso() {
        notas = new double[4];
        nombreAlumno = new ArrayList<String>();
        ...
    }
```

- Ahora cada objeto almacena una colección de datos dentro de él.

La Interfaz Comparable

- El método Collections.sort se puede usar para ordenar una ArrayList. Es parte del paquete java.util. El siguiente programa muestra cómo usar Collections.sort:

```
    // Construye una ArrayList de Strings y la ordena.
import java.util.*;
public class Main {
  public static void main(String[] args) {
    ArrayList<String> palabras = new ArrayList<String>();
    palabras.add("four");
    palabras.add("score");
    palabras.add("and");
    palabras.add("seven");
    palabras.add("years");
    palabras.add("ago");
    // muestra la lista antes y después de ordenar
    System.out.println("antes de ordenar, las palabras son = " + palabras);
    Collections.sort(palabras);
    System.out.println("despues de ordenar, las palabras son = " + palabras);
    }
  }
```

- Este programa produce el siguiente resultado:

```
antes de ordenar las palabras son = [four, score, and, seven, years, ago]
despues de ordenas las palabras son = [ago, and, four, score, seven, years]
```

La Interfaz Comparable

- Si intenta hacer una llamada similar con un objeto “Punto”, a un ArrayList <Punto>, encontrará que el programa no se compila.
- ¿Por qué es posible ordenar una lista de objetos String pero no una lista de objetos Punto?
- La respuesta es que la clase String implementa la interfaz Comparable, mientras que la clase Punto no.

Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

La Interfaz Comparable

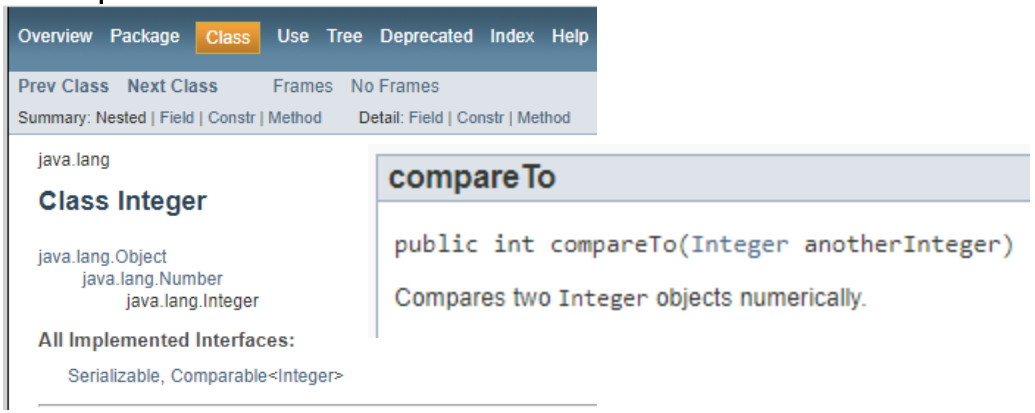
```
public interface Comparable<E> {
    public int compareTo(E otro);
}
```

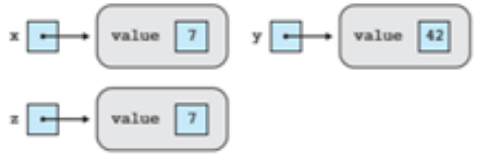
- Una clase puede implementar la interfaz Comparable para definir una función de orden natural para sus objetos (comparación).

- Una llamada a su método compareTo debería devolver:
  un valor <0 si el otro objeto viene “después" de este,
  un valor> 0 si el otro objeto viene “antes" de este,
  o 0 si el otro objeto se considera "igual" a este.

Plantilla Comparable

```
public class nombre implements Comparable<nombre>
{

    ...

    public int compareTo(nombre otro) {
        ...
    }
}
```






```
Integer x = 7;
Integer y = 42;
Integer z = 7;
System.out.println(x.compareTo(y));
System.out.println(x.compareTo(z));
System.out.println(y.compareTo(x));
```

```
-1
0
1
```

El método compareTo

- La forma estándar para que una clase Java defina una función de comparación para sus objetos es definir un método compareTo.
  - Ejemplo: en la clase String, hay un método:

    ```
    public int compareTo(String other)
    ```

- Una llamada de **A**.compareTo(**B**) devolverá:

  un valor <0 si **A** viene "antes" de **B** en el orden,

  un valor> 0 si **A** viene "después" de **B** en el orden,

  o 0 si **A** y **B** se consideran "iguales" en el orden.

Usando compareTo

- compareTo puede usarse como prueba en una declaración if.

```
String a = "alice";
String b = "bob";
if (a.compareTo(b) < 0) {  // true
    ...
}
```

Primitivas	Objetos
`if (a < b) { ...`	`if (a.compareTo(b) < 0) { ...`
`if (a <= b) { ...`	`if (a.compareTo(b) <= 0) { ...`
`if (a == b) { ...`	`if (a.compareTo(b) == 0) { ...`
`if (a != b) { ...`	`if (a.compareTo(b) != 0) { ...`
`if (a >= b) { ...`	`if (a.compareTo(b) >= 0) { ...`
`if (a > b) { ...`	`if (a.compareTo(b) > 0) { ...`

compareTo y Colecciones

- Puede usar un arreglo o una lista de cadenas con el método de búsqueda binaria incluido en Java porque llama a compareTo internamente.

```
String[] a = {"al", "bob", "cari", "dan", "mike"};
int index = Arrays.binarySearch(a, "dan");  // 3
```

- TreeSet/Map de Java utiliza compareTo internamente para realizar ordenamientos.

```
Set<String> set = new TreeSet<String>();
for (String s : a) {
    set.add(s);
}
System.out.println(s);
// [al, bob, cari, dan, mike]
```

Ordenando nuestros propios tipos

- No podemos realizar búsquedas binarias o hacer un `TreeSet/Map` de tipos arbitrarios de objetos, porque Java no sabe cómo ordenar los elementos.

  - El programa compila pero falla cuando lo ejecutamos.

```
Set<HtmlTag> tags = new TreeSet<HtmlTag>();
tags.add(new HtmlTag("body", true));
tags.add(new HtmlTag("b", false));
…
```

```
Exception in thread "main"
  java.lang.ClassCastException
        at java.util.TreeSet.add(TreeSet.java:238)
```

Ejemplo Comparable

- Puede hacer que sus propias clases implementen la interfaz. Esto abrirá una gran cantidad de soluciones que se incluyen en las bibliotecas de clases Java.

- Por ejemplo, hay métodos integrados para ordenar listas y para acelerar las búsquedas.

```
public class Punto implements Comparable<Punto> {
    private int x;
    private int y;
    …

    // ordenar por x y romper lazos por y
    public int compareTo(Punto otro) {
        if (x < otro.x) {
            return -1;
        } else if (x > otro.x) {
            return 1;
        } else if (y < otro.y) {
            return -1;    // mismo x, y más pequeño
        } else if (y > otro.y) {
            return 1;     // mismo x, mayor y
        } else {
            return 0;     // mismo x, mismo y
        }
    }
}
```

Trucos CompareTo

- ***Truco de resta*** : restar valores numéricos relacionados produce el resultado correcto para lo que desea que compareTo devuelva:

```
// ordenar por x y romper lazos por y
public int compareTo(Punto otro) {
    if (x != otro.x) {
        return x - otro.x;   // diferente x
    } else {
        return y - otro.y;   // mismo x; compara y
    }
}
```

  - La idea:
    - Si `x > otro.x`, entonces `x - otro.x > 0`
    - Si `x < otro.x`, entonces `x - otro.x < 0`
    - Si `x == otro.x`, entonces `x - otro.x == 0`
      - NOTA: Este truco no funciona para double (pero vea Math.signum)

Trucos CompareTo 2

- ***Truco de delegación:*** si los atributos de su objeto son comparables (como cadenas), use sus resultados compareTo para ayudarlo a:

```
// ordenar por nombre de empleado, por ejemplo, "Jim" <"Susan"
public int compareTo(Empleado otro) {
    return nombre.compareTo(otro.getNombre());
}
```

- *Truco toString:* Si la representación toString de tu objeto está relacionada con el orden, úsala para ayudarte a:

```
// ordenar por fecha, por ejemplo, "19/09"> "01/04"
public int compareTo(Fecha otro) {
    return toString().compareTo(otro.toString());
}
```

Ejemplo Comparable

- Exploremos una clase que se puede usar para realizar un seguimiento de una fecha del calendario. La idea es realizar un seguimiento de un mes y día en particular, pero no del año.
- Por ejemplo, una organización podría querer una lista de los cumpleaños de sus empleados que no indique cuántos años tienen..

```
// La clase FechaCalendario almacena información sobre una sola fecha
// FechaCalendario (mes y día pero no año)

public class FechaCalendario implements Comparable<FechaCalendario> {
private int mes;
private int dia;
public FechaCalendario(int dia, int mes){
this.mes = mes;
this.dia = dia;
}
// Compara esta fecha del calendario con otra fecha.
// Las fechas se comparan por mes y luego por día.
public int compareTo(FechaCalendario otro) {
if (mes != otro.mes) {
return mes - otro.mes;
} else {
return dia - otro.dia;
}
}
public int getMes() {
return mes;
}
public int getDia() {
return dia;
}
public String toString() {
return dia + "/" + mes;
}

}
```

Ejemplo Comparable

- Una de las principales ventajas de implementar la interfaz Comparable es que le brinda acceso a utilidades integradas como Collections.sort.
- La clase FechaCalendario implementa la interfaz Comparable, por lo que, podemos usar Collections.sort para ordenar un ArrayList <FechaCalendario>:

```
// Programa que crea una lista de los cumpleaños y los ordena
import java.util.*;
public class Main {
public static void main(String[] args) {
ArrayList<FechaCalendario> fechas = new ArrayList<FechaCalendario>();
fechas.add(new FechaCalendario(22, 2));
fechas.add(new FechaCalendario(30, 10));
fechas.add(new FechaCalendario(13, 4));
fechas.add(new FechaCalendario(16, 3));
fechas.add(new FechaCalendario(28, 4));
System.out.println("Cumpleaños = " + fechas);
Collections.sort(fechas);
System.out.println("Cumpleaños = " + fechas);
}
}
```

Big O Colecciones Java

- Implementaciones de Listas

	get	add	contains	next	remove(0)	iterator.remove
ArrayList	O(1)	O(1)	O(n)	O(1)	O(n)	O(n)
LinkedList	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)
CopyOnWrite-ArrayList	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)

- Implementaciones de Conjuntos (Set)

	add	contains	next	notes
HashSet	O(1)	O(1)	O(h/n)	h is the table capacity
LinkedHashSet	O(1)	O(1)	O(1)	
CopyOnWriteArraySet	O(n)	O(n)	O(1)	
EnumSet	O(1)	O(1)	O(1)	
TreeSet	O(log n)	O(log n)	O(log n)	
ConcurrentSkipListSet	O(log n)	O(log n)	O(1)	

- Implementaciones Mapas o Diccionarios (Map)

	get	containsKey	next	Notes
HashMap	O(1)	O(1)	O(h/n)	h is the table capacity
LinkedHashMap	O(1)	O(1)	O(1)	
IdentityHashMap	O(1)	O(1)	O(h/n)	h is the table capacity
EnumMap	O(1)	O(1)	O(1)	
TreeMap	O(log n)	O(log n)	O(log n)	
ConcurrentHashMap	O(1)	O(1)	O(h/n)	h is the table capacity
ConcurrentSkipListMap	O(log n)	O(log n)	O(1)	

- Implementaciones de pilas y colas (queue)

	offer	peek	poll	size
PriorityQueue	O(log n)	O(1)	O(log n)	O(1)
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(1)
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(1)
PriorityBlockingQueue	O(log n)	O(1)	O(log n)	O(1)
DelayQueue	O(log n)	O(1)	O(log n)	O(1)
LinkedList	O(1)	O(1)	O(1)	O(1)
ArrayDeque	O(1)	O(1)	O(1)	O(1)
LinkedBlockingDeque	O(1)	O(1)	O(1)	O(1)

https://www.bigocheatsheet.com/

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estudio de caso: Vocabulario

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

RESUMEN

Estructuras de Datos Lineales

- Cuando quieres guardar múltiples elementos, usa un arreglo o una lista.
- Con un arreglo, todos tus elementos se almacenan uno al lado del otro.
- Con una lista, los elementos se encuentran dispersos y un elemento guarda la dirección del próximo elemento.
- Los arreglos permiten lecturas rápidas.
- Las listas enlazadas permiten inserciones y eliminaciones rápidas.
- Todos los elementos en un arreglo deben ser del mismo tipo.

Resumiendo y Repasando…

- La clase ArrayList en el paquete java.util de Java representa una lista ampliable de objetos implementados usando un arreglo. Puede usar una ArrayList para almacenar objetos en orden secuencial. Cada elemento tiene un índice basado en cero.
- ArrayList es una clase genérica. Una clase genérica acepta un tipo de datos como parámetro cuando se crea, como ArrayList <String>.
- Una ArrayList mantiene su propio tamaño para usted; Se pueden agregar o quitar elementos de cualquier posición hasta el tamaño de la lista. Otras operaciones de ArrayList incluyen get, set, clear y toString.
- Se puede buscar en ArrayLists usando los métodos contains, indexOf y lastIndexOf.

Resumiendo y Repasando…

- El bucle Java for-each se puede usar para examinar cada elemento de un ArrayList. La lista no se puede modificar durante la ejecución del ciclo for-each.
- Cuando almacene valores primitivos como int o double en un ArrayList, debe declarar la lista con tipos especiales de contenedor como Integer y Double.
- La interfaz comparable define un orden natural para los objetos de una clase. Los objetos que implementan Comparable se pueden colocar en una ArrayList y ordenarlos. Muchos tipos comunes (como String e Integer) implementan Comparable. Puede implementar la interfaz Comparable en sus propias clases escribiendo un método compareTo.

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América