



UNIDAD I2: COLECCIONES

CLASE I: ESTRUCTURAS DINÁMICAS

INTRODUCCIÓN

Implementar una clase denominada **Lista** correspondiente a una lista de cadenas de la clase String. Los String se guardarán en una tabla que cambiará de tamaño según se hagan inserciones o eliminaciones. Dentro de la clase deberán estar los siguientes métodos:

- Un **constructor** que inicialice la tabla con longitud 0.
- **Length**: Método que obtenga el tamaño de la tabla.
- **InsertLast**: Insertar una cadena al final del la tabla.
- **InsertFirst**: Insertar una cadena al principio de la tabla.
- **InsertIn**: Insertar una cadena en una posición(indice) indicada por parámetro.

INTRODUCCIÓN

- **AddLista:** Añadir los valores de una lista pasada por parámetro a esta tabla
- **DeleteIn:** Eliminar una cadena de la tabla cuyo índice se pasa por parámetro.
- **Get:** Obtener la cadena almacenada en una posición pasada por parámetro.
- **Search:** Buscar una cadena en la tabla, devolviendo el índice en el que encuentre la primera coincidencia. Si no encuentra coincidencias devolverá -1.
- **toString:** Mostrar todas las cadenas de la lista por pantalla.

ESTRUCTURAS ESTÁTICAS

Las estructuras estáticas presentan severos problemas:

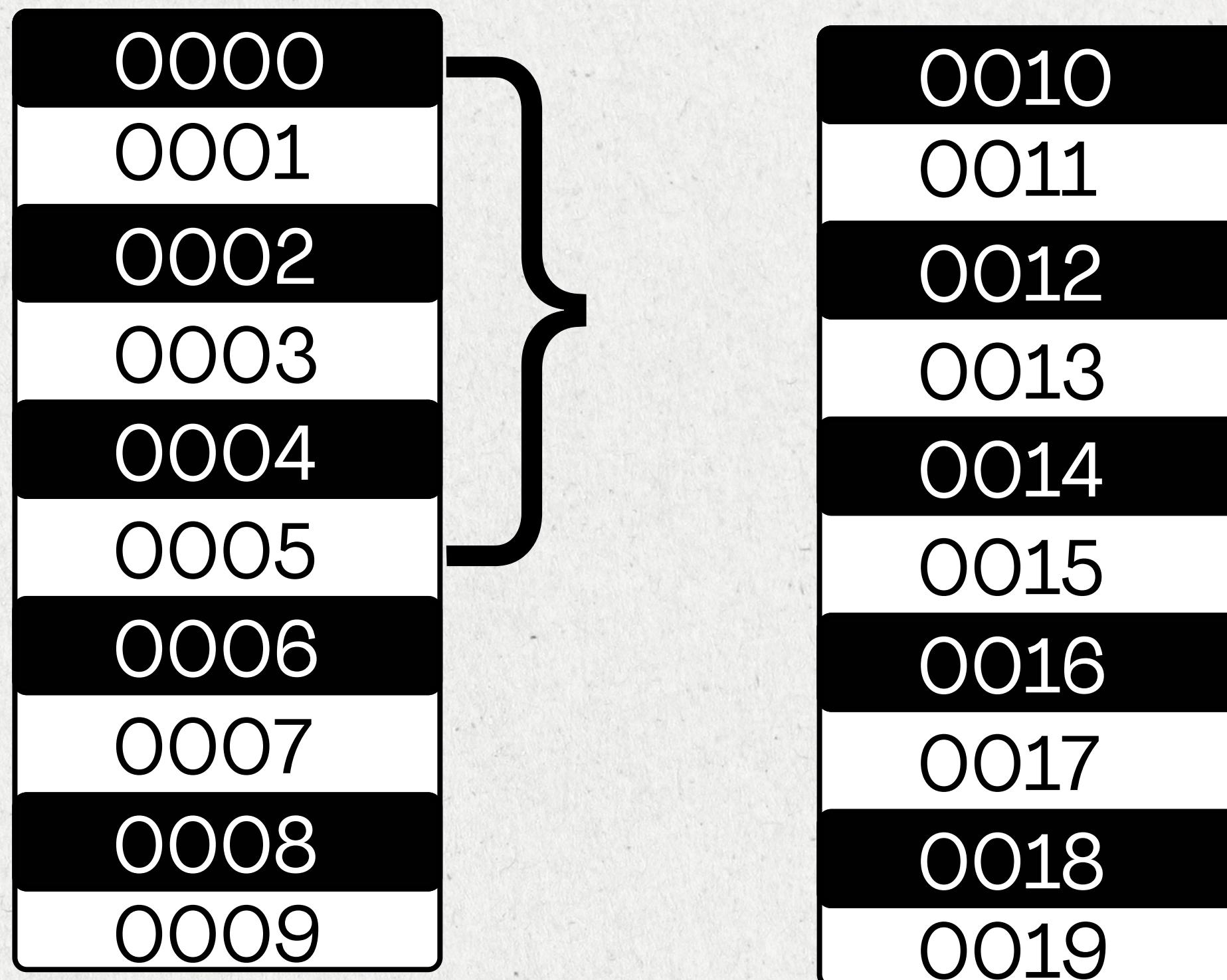
- Almacenan una cantidad concreta de elementos
- Para poder modificar su capacidad, creamos nuevas estructuras y copiamos los datos en esta nueva.
- Estas continuas creaciones de tablas sobrecargan la memoria.

ESTRUCTURAS DINÁMICAS

Las estructuras dinámicas no precisan de este redimensionamiento.

Estas estructuran reservan un espacio de memoria y, en el momento en el que se quedan sin espacio, realizan una copia.

ESTRUCTURAS DINÁMICAS



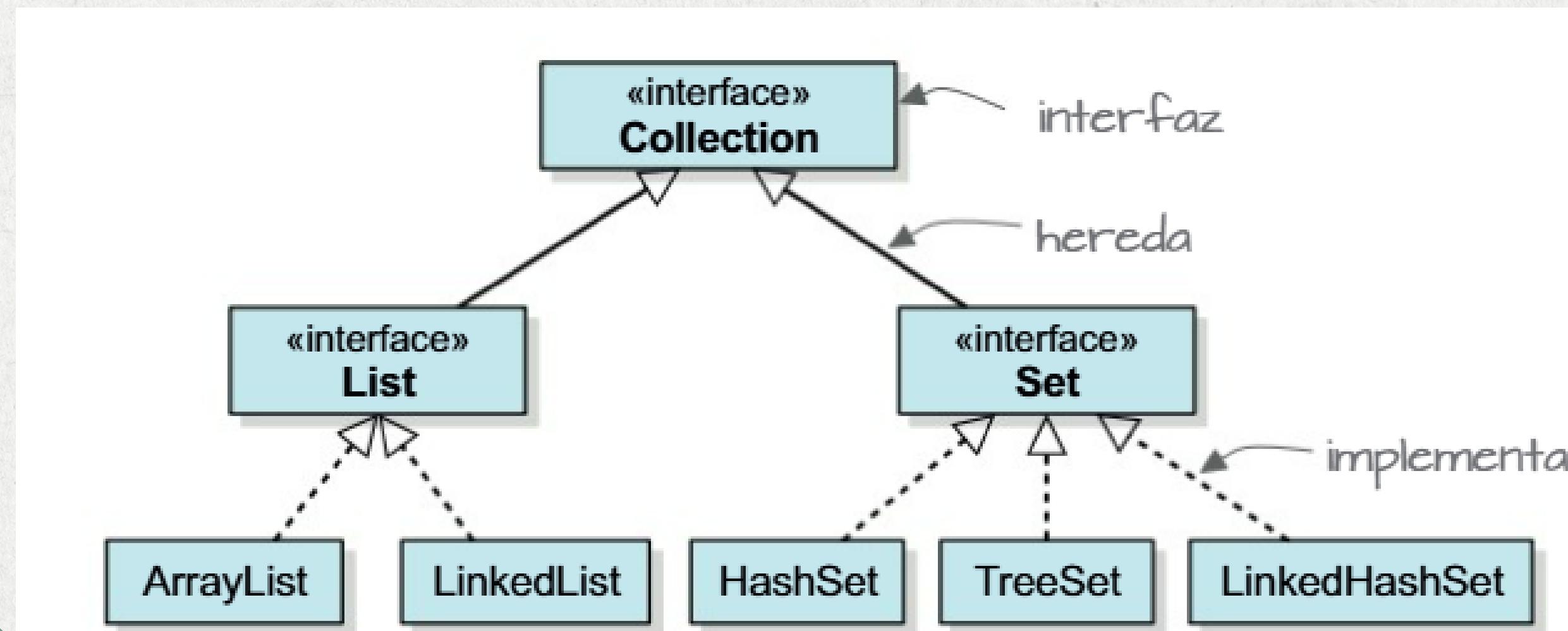
ESTRUCTURAS DINÁMICAS

0000
0001
0002
0003
0004
0005
0006
0007
0008
0009

0010
0011
0012
0013
0014
0015
0016
0017
0018
0019

ESTRUCTURAS DINÁMICAS

Todas las estructuras dinámicas que vamos a estudiar heredan de la interfaz “Collection”



ESTRUCTURAS DINÁMICAS

Una colección es un objeto que agrupa otros objetos que guardan una relación entre ellos. Los métodos de estas interfaces nos permiten manejar la inserción, eliminación, búsqueda u ordenación de sus elementos.

ESTRUCTURAS DINÁMICAS

Existen una gran variedad de colecciones entre los que vamos a destacar:

- **Listas:** Manejan sucesiones de datos que pueden repetirse y pueden estar ordenados (o no)

Sustituyen a las tablas estáticas

Estas listas implementan la interfaz *List*

ESTRUCTURAS DINÁMICAS

Existen una gran variedad de colecciones entre los que vamos a destacar:

- **Conjuntos:** Manejan datos que no pueden repetirse (ni tiene sentido que suceda), el orden de los datos no es relevante, sólo si existen en el conjunto

Los conjuntos implementan la interfaz Set

ESTRUCTURAS DINÁMICAS

Existen una gran variedad de colecciones entre los que vamos a destacar:

- **Mapas o diccionarios:** Almacenan datos identificados por claves que no pueden repetirse.

Los mapas implementan la interfaz *Map* que no se encuentra bajo el árbol de *Collection*

TIPOS GENÉRICOS

Las colecciones pueden almacenar datos de cualquier tipo, por lo que su especificación se hace utilizando tipos genéricos.

Una lista puede crearse sobre enteros, cadenas u objetos de clases.

TIPOS GENÉRICOS

Un tipo genérico es un elemento que se adapta para realizar la misma funcionalidad para varios tipos de datos.

Algunos ejemplos que utilizan tipos genéricos son las interfaces Comparable y Comparator. Estas comparan elementos sin importar el tipo.

TIPOS GENÉRICOS

En últimas versiones de Java se ha introducido la sobrecarga de la interfaz Comparable tal que:

```
public class Persona implements Comparable<Persona> {  
    @Override  
    public int compareTo(Persona o) {  
        return 0;  
    }  
}
```

TIPOS GENÉRICOS

Generalmente, se utilizará T para identificar a los tipos genéricos

`Comparable<T>`

`Comparable<String>`

`Comparable<Integer>`

`Comparable<Persona>`

CLASES SOBRE TIPOS GENÉRICOS

Queremos crear una clase Contenedor que almacene un objeto de cualquier tipo

```
public class Contenedor {  
    private Object o;  
    Contenedor() {}  
    void guardar(Object otro) {  
        o = otro;  
    }  
    Object extraer() {  
        Object or = o;  
        o = null;  
        return or;  
    }  
}
```

CLASES SOBRE TIPOS GENÉRICOS

Mediante el operador <> indicamos que la implementación de Contenedor variará según el tipo especificado

```
public class Contenedor<T> {  
    private T o;  
    Contenedor() {}  
    void guardar(T otro) {  
        o = otro;  
    }  
    T extraer() {  
        T or = o;  
        o = null;  
        return or;  
    }  
}
```

CLASES SOBRE TIPOS GENÉRICOS

Podremos crear Contenedores de diversos tipos de datos

```
public static void main(String[] args) {  
    Contenedor<Integer> entero = new Contenedor<Integer>();  
    Contenedor<String> cadena = new Contenedor<>();  
    Contenedor<Persona> persona = new Contenedor<>();  
}
```

CLASES SOBRE TIPOS GENÉRICOS

¿Qué ventajas plantea utilizar tipos genéricos sobre Object?

```
public static void main(String[] args) {  
    Contenedor<Integer> cont_tg = new Contenedor<Integer>();  
    ContenedorObject cont_obj = new ContenedorObject();  
    cont_obj.guardar(5);  
    cont_tg.guardar(5);  
    Integer in_obj = (Integer)cont_obj.extraer();  
    Integer in_tg = cont_tg.extraer();  
}
```

CLASES SOBRE TIPOS GENÉRICOS

Por lo general, se identifica la T para los tipos genéricos, sin embargo, se puede utilizar cualquier letra del abecedario.

- E suele identificar a los elementos de una colección
- K para claves (key) y V para valores (value)
- N para números

CLASES SOBRE TIPOS GENÉRICOS

Se pueden crear clases dependientes de más de un tipo genérico:

```
public class Contenedor<T,U,V> {  
    private T t;  
    private U u;  
    private V v;
```

INTERFACES CON TIPOS GENÉRICOS

Podemos crear interfaces con tipos genéricos

```
public interface Comparable<T> {  
    int compareTo(T o);  
}  
  
public class Persona implements Comparable<Persona> {  
    String dni;  
    Persona(String d){dni=d;}  
    public int compareTo(Persona o) {  
        return dni.compareTo(o.dni);  
    }  
}
```

INTERFACES CON TIPOS GENÉRICOS

Podemos crear interfaces con tipos genéricos

```
public int compareTo(Object o) {  
    Persona ob = (Persona) o;  
    return dni.compareTo(ob.dni);  
}
```

```
public int compareTo(Persona o) {  
    return dni.compareTo(o.dni);  
}
```

PARÁMETROS GENÉRICOS LIMITADOS

En algunos casos, puede ser que queramos crear métodos que funcionan específicamente para un tipo de datos

Métodos aritméticos (suma, resta...) - No es aplicable a String o Persona

PARÁMETROS GENÉRICOS LIMITADOS

Podemos limitar que tipos de datos son aceptados estableciendo una “clase límite”

Se limita todos los tipos posibles a la clase límite y todas las subclases

PARÁMETROS GENÉRICOS LIMITADOS

```
public class Calculadora<T extends Number> {  
    private T a,b;
```

```
public static void main(String[] args) {  
    Calculadora<Integer> cont_int = new Calculadora<Integer>(1,2);  
    Calculadora<Double> cont_db = new Calculadora<Double>(1.0,2.0);  
    Calculadora<String> cadena = new Calculadora<String>("1","3");
```

COMODINES

Se utilizan en la declaración de atributos, variables locales o parámetros pasados a una función. Es representado por “?” y significa cualquier tipo.

Contenedor<?> c; → Crea un contenedor
de un tipo cualquiera

COMODINES

Este contenedor puede almacenar cualquier tipo al que se iguale.

Podríamos decir que es una “superclase” que puede identificar a cualquier otro tipo.

```
Calculadora<?> entero = new Calculadora<Integer>(1,2);
```

COMODINES

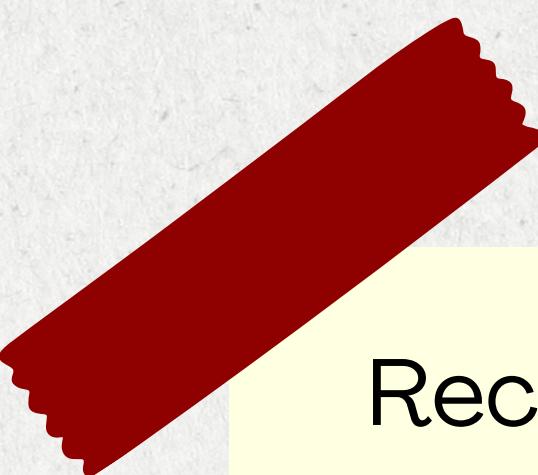
NO existe herencia entre clases con tipos genéricos



LIMITACIONES DE LOS PARÁMETROS GENÉRICOS

- Los tipos genéricos **no pueden ser primitivas** (int, char...) para estos tendremos que utilizar sus clases envoltorio (Integer, Character)
- **No se pueden crear instancias** de tipo genérico (new T())
- **No se pueden crear tablas** de tipos genéricos **fueras de clases** (new T[10])
- **No se pueden crear tablas de clases parametrizadas** (new Contenedor<Integer>[5])

EJERCICIOS



Recupera la clase Lista creada al comienzo de la clase y adáptala para que trabaje con tipos genéricos. Después crea tres listas en main que trabajen con double, String y una clase creada por ti en otro ejercicio.

EJERCICIOS

Implementa un método estático que reciba dos tablas de tipos genéricos y las fusionen en una nueva tabla que devuelva por parámetro (la primera tabla primero y la segunda después).

Después crea tablas de double y cadenas y prueba el funcionamiento.