

## Introducción a los Genéricos en Java

Puri Arenas Sánchez (Grupos A y D)  
 Yolanda García Ruiz (Grupo E)  
 Facultad de Informática (UCM)  
 Curso 2019/2020

## Introducción

Si queremos implementar una clase que maneje un array de enteros, podemos escribir:

```
public class IntList {
    private final static int _INIT_SIZE = 2;
    private Integer[] elem;
    private int last;

    public IntList() {
        last = -1;
        elem = new Integer[_INIT_SIZE];
    }

    public void addElem(Integer e) {
        if (last == elem.length - 1) elem = Arrays.copyOf(elem, elem.length * 2);
        elem[++last] = e;
    }

    public Integer getElem(int index) {
        if (index > last) return null;
        else return elem[index];
    }
}

/** Main
 * IntList x = new IntList();
 * x.addElem(12);
 * x.addElem(244);
 * x.addElem(31);
 * System.out.println(x.getElem(0));
 * System.out.println(x.getElem(1));
 * System.out.println(x.getElem(2));
 */
```

## Introducción

Si queremos implementar una clase que maneje un array de String, podemos escribir:

```
public class StringList {
    private final static int _INIT_SIZE = 2;
    private String[] elem;
    private int last;

    public StringList() {
        last = -1;
        elem = new String[_INIT_SIZE];
    }

    public void addElem(String e) {
        if (last == elem.length - 1) elem = Arrays.copyOf(elem, elem.length * 2);
        elem[++last] = e;
    }

    public String getElem(int index) {
        if (index > last) return null;
        else return elem[index];
    }
}

/** Main
StringList x = new StringList();
x.addElem("Hola");
x.addElem("Hey");
x.addElem("bla");
System.out.println(x.getElem(0));
System.out.println(x.getElem(1));
System.out.println(x.getElem(2));
```

3

## Introducción

- Ambas clases **IntList** y **StringList** son iguales, salvo que cambia el tipo **Int** por **String**.
- Desventaja: **difícil mantenimiento del código**  
Tenemos dos implementaciones, una para cada tipo.  
Por ejemplo si encontramos un error, tendremos que corregirlo en ambas versiones.
- Si quisiéramos una lista de elementos de tipo "**Double**", otra vez tendríamos que generar una nueva clase.

¿Cómo se puede solucionar en **Java**?

Usando  
genéricos

4

## Introducción

Podemos utilizar elementos de tipo **Object**!

```
public class ObjList {
    private final static int _INIT_SIZE = 2;
    private Object[] elem;
    private int last;

    public ObjList() {
        last = -1;
        elem = new Object[_INIT_SIZE];
    }

    public void addElem(Object e) {
        if (last == elem.length - 1) elem = Arrays.copyOf(elem, elem.length * 2);
        elem[++last] = e;
    }

    public Object getElem(int index) {
        if (index > last) return null;
        else return elem[index];
    }
}
```

¿Cuál es el Problema?

```
/** Main
ObjList x = new ObjList();
x.addElem(1);
x.addElem(13);

ObjList y = new ObjList();
y.addElem("Hola!");
y.addElem("Adios!");
```

5

## Introducción

- ❑ **Problema 1:** Obliga a hacer "casting".

```
ObjList x = new ObjList();
x.addElem(1);
x.addElem(13);

int cont = 0;
for (int i=0; i< x.size(); i++)
    cont = cont + x.getElem(i);

System.out.println(cont);
```

**x.getElem(i)**  
devuelve un elemento de  
tipo **Object**

- ✓ Los elementos de tipo **Object** no soportan "+"
- ✓ No se puede sumar un **Object** y un **Integer** !!!!

**Necesitamos usar casting:**

```
cont = cont + (Integer) x.getElem(i);
```

6

## Introducción

- ❑ **Problema 2:** Por otro lado podríamos generar tablas con elementos incompatibles, que dieran problemas en ejecución (por ejemplo al ser comparados).

```
ObjList x = new ObjList();
x.addElem(1);
x.addElem(13);
x.addElem("hola");
```

7

## Solución: Genéricos en Java

- Los genéricos son una utilidad de Java, introducida en J2SE 5.0 en 2005, que permite definir clases e interfaces sobre un **tipo general**, evitando los problemas anteriores.
- Con los genéricos sólo tendríamos una clase **GenericList<T>**, donde **T** es un tipo genérico que representa el nombre de una clase. Si queremos usar una lista de enteros pondríamos **GenericList<Integer>**. Para una lista de **String** pondríamos **GenericList<String>**.
- La cabecera de una *clase genérica* tiene, en general, la forma:
 

```
public class ClaseGenerica< T extends OtraClase >
```
- Si sólo ponemos **ClaseGenerica<T>**, se entiende que **T** extiende a **Object**.
- **OtraClase** actúa como la cota superior y hace que sólo se pueda instanciar con clases que sean subclases de **OtraClase**.

8

## Genéricos. Ejemplo

```
public class GenericList<T extends Object> {
    private final static int INIT_SIZE = 2;
    private T[] elem;
    private int last;

    public GenericList() {
        this.last = -1;
        this.elem = (T[]) new Object[INIT_SIZE];
    }

    public void addElem(T e) {
        if (this.last == this.elem.length-1)
            elem = Arrays.copyOf(elem, elem.length * 2);
        this.elem[++last] = e;
    }

    public T getElem(int index) {
        if (index > this.last) return null;
        else return this.elem[index];
    }
}
```

✓ T es un parámetro de tipo **Object**

✓ **Object** es la cota superior de T.

✓ Si es **Object**, no hace falta escribir "extends Object"

➤ Se puede usar el tipo **GenericList<T>** con cualquier T, siempre que sea una subclase de la cota superior de T.

**GenericList<String>**  
**GenericList<Integer>**  
**GenericList<Double>**

9

## Genéricos. Ejemplo

```
public class GenericList<T extends Number> {
    private final static int INIT_SIZE = 2;
    private T[] elem;
    private int last;

    public GenericList() {
        this.last = -1;
        this.elem = (T[]) new Number[INIT_SIZE];
    }

    public void addElem(T e) {
        if (this.last == this.elem.length-1)
            elem = Arrays.copyOf(elem, elem.length * 2);
        this.elem[++last] = e;
    }

    public T getElem(int index) {
        if (index > this.last) return null;
        else return this.elem[index];
    }
}
```

➤ "T extends Number" permite usar los tipos:

**GenericList<Integer>**  
**GenericList<Double>**

Pero no:

**GenericList<String>**

10

## Genéricos. Ejemplo

```
public class GenericList<T extends Object> {
```

```
    private final static int INIT_SIZE = 2;
    private T[] elem;
    private int last;
```

```
    public GenericList() {
        this.last = -1;
        this.elem = (T[]) new Object [INIT_SIZE];
    }
```

```
    public void addElem(T e) {
        if (this.last == this.elem.length-1)
            elem = Arrays.copyOf(elem, elem.length * 2);
        this.elem[++last] = e;
    }
```

```
    public T getElem(int index) {
        if (index > this.last) return null;
        else return this.elem[index];
    }
}
```

- No se pueden crear instancias de T o arrays de T.
- Normalmente hay que usar la cota superior

11

## Cómo usar la lista genérica?

```
public static void main(String[] args) {
```

```
    GenericList<String> x = new GenericList<String>();
    x.addElem(100);
    x.addElem("aaa");
    String a1 = x.getElem(0);
    Integer a2 = x.getElem(0);
```

```
    GenericList<Integer> y = new GenericList<Integer>();
    y.addElem(100);
}
```

12

## Cómo usar la lista genérica?

```
public static void main(String[] args) {  
    GenericList<String> x = new GenericList<String>();  
    x.addElem(100); // MAL  
    x.addElem("aaa");  
    String a1 = x.getElem(0);  
    Integer a2 = x.getElem(0); // MAL  
  
    GenericList<Integer> y = new GenericList<Integer>();  
    y.addElem(100);  
}
```

13

## Cómo compila Java los genéricos?

- Los programas Java se compilan a Java Bytecode.
- En Java Bytecode no hay genéricos.
- Para compilar un genérico, **Java reemplaza los parámetros por sus cotas superiores.**
- El compilador usa inferencia de tipos para detectar posibles errores y añade casting cuando es necesario.

14

## Cómo compila Java los genéricos?

```
public class GenericList<T extends Object> {
    private final static int INIT_SIZE = 2;
    private T[] elem;
    private int last;

    public GenericList() {
        this.last = -1;
        this.elem = ((T[]) new Object[INIT_SIZE]);
    }
    public void addElem(T e) {
        if (this.last == this.elem.length-1)
            this.elem = Arrays.copyOf(this.elem,
                this.elem.length * 2);
        this.elem[++last] = e;
    }
    public T getElem(int index) {
        if (index > this.last) return null;
        else return this.elem[index];
    }
}
```

```
public class GenericList {
    private final static int INIT_SIZE = 2;
    private Object[] elem;
    private int last;

    public GenericList() {
        this.last = -1;
        this.elem = new Object[INIT_SIZE];
    }
    public void addElem(Object e) {
        if (this.last == this.elem.length-1)
            this.elem = Arrays.copyOf(this.elem,
                this.elem.length * 2);
        this.elem[++this.last] = e;
    }
    public Object getElem(int index) {
        if (index > last) return null;
        else return this.elem[index];
    }
}
```

15

## Cómo compila Java los genéricos?

```
public class GenericList<T extends Object> {
    private final static int INIT_SIZE = 2;
    private T[] elem;
    private int last;

    public GenericList() {
        this.last = -1;
        this.elem = ((T[]) new Object[INIT_SIZE]);
    }
    public void addElem(T e) {
        if (this.last == this.elem.length-1)
            this.elem = Arrays.copyOf(this.elem,
                this.elem.length * 2);
        this.elem[++last] = e;
    }
    public T getElem(int index) {
        if (index > this.last) return null;
        else return this.elem[index];
    }
}
```

```
GenericList<Integer> y = new MyList<Integer>();
y.addElem(100);
Integer z = y.getElem(0);
```

```
public class GenericList {
    private final static int INIT_SIZE = 2;
    private Object[] elem;
    private int last;

    public GenericList() {
        this.last = -1;
        this.elem = new Object[INIT_SIZE];
    }
    public void addElem(Object e) {
        if (this.last == this.elem.length-1)
            this.elem = Arrays.copyOf(this.elem,
                this.elem.length * 2);
        this.elem[++this.last] = e;
    }
    public Object getElem(int index) {
        if (index > last) return null;
        else return this.elem[index];
    }
}
```

```
GenericList y = new GenericList();
y.addElem(100);
Integer z = (Integer) y.getElem(0);
```

16



## Múltiples parámetros de tipo

- Un genérico puede tener varios parámetros de tipo

```
public class Pair<A,B> {
    private A first;
    private B second;

    public Pair(A first, B second) {
        this.first = first;
        this.second = second;
    }
    public A getFirst() {
        return first;
    }
    public B getSecond() {
        return second;
    }
    public String toString() {
        return "(" + first + ", " + second + ")";
    }
}
```

```
Pair<Integer,String> x=new Pair<Integer,String>(1,"hola");
Pair<Integer,String> y=new Pair<Integer,String>(3,"adios");
```

```
Pair<Integer,Integer> z = new Pair<Integer,Integer>(5,6);
```

```
System.out.println(x);
System.out.println(y);
System.out.println(z);
```

(1,hola)  
(3,adios)  
(5,6)

17

## Múltiples parámetros de tipo

```
public class Pair<A extends Number,B extends GenericList<A>> {
    private A first;
    private B second;

    public Pair(A first, B second) {...}
    public A getFirst() {...}
    public B getSecond() {...}
    public String toString() {...}
}
```

(1, [1])  
(3.0, [2, 0])

```
static public void main(String[] args) {
    GenericList<Integer> l1 = new GenericList<Integer>();
    GenericList<Double> l2 = new GenericList<Double>();
    l1.addElem(1);
    l2.addElem(2.0);
    Pair<Integer,GenericList<Integer>> x = new Pair<Integer,GenericList<Integer>>(1,l1);
    Pair<Double,GenericList<Double>> y = new Pair<Double,GenericList<Double>>(3.0,l2);
    System.out.println(x);
    System.out.println(y);
}
```

18

## Interfaces genéricas

```
public interface Listl<T> {
    public abstract void addElem(T e);
    public abstract T getElem(int index);
    public abstract int size();
}
```

```
public class MyList<T> implements Listl<T> {

    private final static int INIT_SIZE = 2;
    private T[] elem;
    private int last;

    public MyList() {
        last = -1;
        elem = (T[]) new Object[INIT_SIZE];
    }
    @Override
    public void addElem(T e) {...}
    @Override
    public T getElem(int index) {...}
    @Override
    public int size() {...}
    public String toString() {...}
}
```

- La clase MyList<T> no es abstracta, pero si genérica. Necesita implementar los métodos de la interfaz.
- MyList<T> es una clase genérica

19

## Interfaces genéricas

```
public interface Listl<T> {
    public abstract void addElem(T e);
    public abstract T getElem(int index);
    public abstract int size();
}
```

```
public class MyList implements Listl<Integer> {

    private final static int INIT_SIZE = 2;
    private Integer[] elem;
    private int last;

    public MyList() {
        last = -1;
        elem = new Integer[INIT_SIZE];
    }
    @Override
    public void addElem(Integer e) {...}
    @Override
    public Integer getElem(int index) {...}
    @Override
    public int size() {...}
    public String toString() {...}
}
```

- La clase MyList no es genérica, pero implementa a una interfaz genérica instanciada para un caso concreto.

20

## Extender clases genéricas

```
abstract public class Listl<T> {
    protected T[] elem;
    protected int last;
    protected final static int INIT_SIZE = 2;

    public Listl() {
        last = -1;
        elem = (T[]) new Object[INIT_SIZE];
    }

    public abstract void addElem(T e);

    public T getElem(int index) {
        if (index > last) return null;
        else return elem[index];
    }

    public int size(){ return last + 1; }
}
```

// redimensiona el tamaño

```
public class MyList1<T> extends Listl<T> {
    @Override
    public void addElem(T e) {
        if (last == elem.length - 1)
            elem = Arrays.copyOf(elem, elem.length * 2);
        elem[++last] = e;
    }
}
```

// sino cabe el elemento no se mete

```
public class MyList2<T> extends Listl<T> {
    @Override
    public void addElem(T e) {
        if (last < elem.length - 1)
            elem[++last] = e;
    }
}
```

21

## Métodos genéricos

- Son métodos con parámetros genéricos. No tienen necesariamente que pertenecer a una clase genérica.

```
public class Utils {

    public static <A,B> boolean compare(Pair<A,B> p1, Pair<A,B> p2) {
        return p1.getFirst().equals(p2.getFirst()) &&
            p2.getSecond().equals(p2.getSecond());
    }
}
```

- Se indica que es genérico, añadiendo la especificación de parámetros de tipo (uno o más), antes del valor de retorno del método.
- En este caso, <A, B>.
- El método acepta una clase genérica, *Pair<A,B>*.
- De hecho, se pueden usar los tipos A y B en el cuerpo del método.

22

## Métodos genéricos

- Son métodos con parámetros genéricos. No tienen necesariamente que pertenecer a una clase genérica.

```
public class Utils {

    public static <A,B> boolean compare(Pair<A,B> p1, Pair<A,B> p2) {
        return p1.getFirst().equals(p2.getFirst()) &&
            p2.getSecond().equals(p2.getSecond());
    }
}
```

```
Pair<Integer,String> x = new Pair<Integer,String>(1,"hola");
Pair<Integer,String> y = new Pair<Integer,String>(1,"hola");
if (Utils.compare(x, y)) System.out.println("iguales");
else System.out.println("distintos");
```

- ✓ No hace falta escribir los tipos en la llamada a `Utils.compare(x, y)`.
- ✓ El compilador infiere que son `<Integer,String>` usando los tipos de `x` e `y`.

23

## Métodos genéricos

```
public class Utils {

    public static <T> void print(MyList<T> t){
        for (int i=0; i<= t.size(); i++)
            System.out.println(t.getElem(i));
    }
}
```

```
public static void main(String[] args) {
    MyList<Integer> x = new MyList<Integer>();
    x.addElem(1);
    x.addElem(1);
    x.addElem(7);

    Utils.print(x);
    MyList<String> y = new MyList<String>();
    y.addElem("hola");
    y.addElem("adios");
    Utils.print(y);
}
```

24

## Genéricos y subclases

¿Es correcto el siguiente código, suponiendo creada la clase genérica `MyList<T>`?

```
MyList<Integer> x = new MyList<Integer>();
...
MyList<Number> y = x;
```



**Error de tipos:** el tipo de `x` no es subclase del tipo de `y` !!!

Es decir `MyList<Integer>` no es subclase de `MyList<Number>` aunque `Integer` sea subclase de `Number`.

25

## Genéricos y subclases



```
public static void print(MyList<Number> t){
    for (int i=0; i< t.size(); i++)
        System.out.println(i);
}
```

¿ Es correcto el siguiente código?

```
MyList<Integer> x = new MyList<Integer>();
...
print(x);
```

**Error de tipos:** el tipo `MyList<Integer>` no es subclase de `MyList<Number>`.

26

## Genéricos y subclases



```
public static <T extends Number> void print(MyList<T> t){  
    for (int i=0; i< t.size(); i++)  
        System.out.println(i);  
}
```

¿ Es correcto el siguiente código?

```
MyList<Integer> x = new MyList<Integer>();  
...  
print(x);
```

**Es correcto** porque **Integer extends Number**.

Sin embargo no podemos ejecutar **t.add(new Integer())** dentro de **print**, ya que dentro del método **print**, **MyList** no ha sido instanciada a ningún tipo.