

# Clases Parametrizables

Dpto. Lenguajes y Sistemas Informáticos  
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2021-2022)

# Créditos

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
  - ▶ Emojis, <https://pixabay.com/images/id-2074153/>
- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

# Objetivos

- Conocer las clases parametrizables y su utilidad
- Saber identificarlas en un diagrama de clases
- Saber definirlas y utilizarlas

# Contenidos

- 1 **Introducción**
- 2 **Clases parametrizables**
- 3 **Clases parametrizables en UML**
- 4 **Clases parametrizables en Java**
  - Clases parametrizables e interfaces
  - Algunas consideraciones de bajo nivel

# Introducción a las clases parametrizables

- Suponed que se necesita
  - ▶ Una lista de objetos de la clase Persona
  - ▶ Una lista de objetos de la clase Vehículo
  - ▶ Una lista de objetos de la clase Mascota
  - ▶ Se estima que se pueden necesitar listas de objetos de otras clases
  - ▶ Todas las listas se van a gestionar igual: insertar, borrar, etc.

★ ¿De qué modo podría diseñarse/implementarse?

## Pseudocódigo: ¿Mejorable?

```
1 class ListaPersona { void insertar (Persona p) { ... } ... }  
2 class ListaVehículo { void insertar (Vehiculo v) { ... } ... }  
3 class ListaMascota { void insertar (Mascota m) { ... } ... }
```



# Clases parametrizables

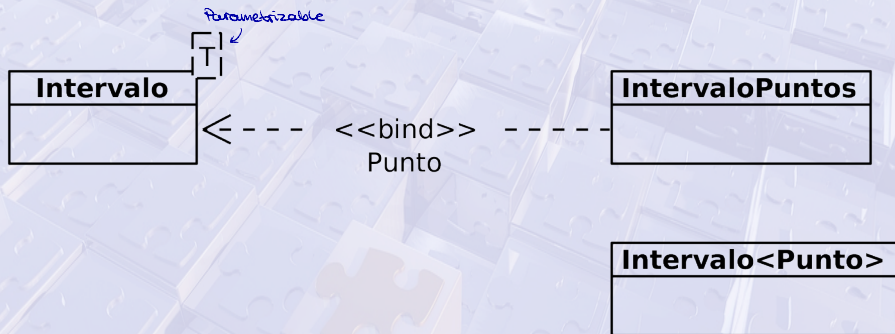
- Clases definidas en función de un tipo de dato (clase)
- Se generalizan un conjunto de operaciones válidas para diferentes tipos de datos
- El ejemplo clásico son los contenedores
  - ▶ Se puede definir una lista independientemente del tipo concreto de elementos que vaya a contener

## Pseudocódigo: Clase parametrizable

```
1 // Lista de objetos de la clase cualquiera T
2 class Lista <T> { void insertar (T e) {...} ... }
3
4 // Cuando se necesite una lista de cualquier clase,
5 // solo hay que instanciarla indicando la clase concreta para esa lista
6
7 Lista <Persona> listaPersonas = new ...
8 Lista <Vehiculo> listaVehiculos = new ...
9 Lista <Mascota> listaMascotas = new ...
```



# Clases parametrizables en UML





# Clases parametrizables en Java

- Este concepto se implementa mediante los tipos genéricos (*generics*)
- Permite pasar tipos como parámetros a clases e interfaces
  - ▶ Esos parámetros (que representan tipos) se pueden usar allí donde habitualmente se usa un tipo, por ejemplo:
    - ★ Al declarar un atributo
    - ★ Al declarar el tipo devuelto por un método
    - ★ Al declarar el tipo de un parámetro de un método
- Se puede forzar que el tipo suministrado a una clase parametrizable:

- ▶ Tenga que ser subclase de otro, o

```
class Clase <T extends ClaseBase>
```

- ▶ Tenga que realizar una interfaz

```
class Clase <T extends Interfaz>
```

**CUIDADO**

cuando una clase usa una interfaz, se pone implements Aquí pone extends!!

**NOTA:** A final de curso voy que entender los mapas de Carton.

A 13 de mayo podemos entender todo menos el primero



# Ejemplo

## Java: Clase parametrizable

```

1 class NodoOctal <T> {
2     private static final int NHIJOS = 8;
3     private NodoOctal <T> padre = null;
4     private final ArrayList <NodoOctal <T>> hijos = new ArrayList <>();
5     private ArrayList <T> contenido = new ArrayList <>();
6
7     NodoOctal () { padre = null; }
8     NodoOctal (NodoOctal <T> p) { padre = p; }
9
10    void divide () {
11        if (hijos.size() == 0) {
12            for (int i = 0; i < NHIJOS; i++) { hijos.add (new NodoOctal <> (this)); }
13        }
14    }
15
16    T obtenerElemento (int i) {
17        if ((i < contenido.size()) && (i >= 0)) { return contenido.get (i); }
18        else { return null; }
19    }
20
21    void insertarElemento (T e) { contenido.add (e); }
22
23    NodoOctal <T> hijo (int i) {
24        if ((i < hijos.size()) && (i >= 0)) { return hijos.get(i); }
25        else { return null; }
26    }
27 }

```

Como un árbol binario, pero 8 en vez de 2

Cuando divididos en 8 subestructuras que pueden ser de cualquier tipo

No se pone el tipo para evitar redundancia igual que con el ArrayList

# Comprobación de tipos en tiempo de compilación

- Suponer el siguiente caso práctico
  - ▶ Un centro de estudios organiza cursos de apoyo para estudiantes de primaria y secundaria
  - ▶ Se necesita una clase `Curso` con (entre otros) un método `matricularEstudiante`
  - ▶ En un curso no puede haber estudiantes de diferentes ciclos

## Java: Solución sin clases parametrizables

```

1 abstract class Estudiante { . . . }
2 class EstudiantePrimaria extends Estudiante { . . . }
3 class EstudianteSecundaria extends Estudiante { . . . }
4 class Curso {
5   void matricularEstudiante (Estudiante e) { . . . }
6 } // Es responsabilidad del programador evitar cursos con estudiantes de diferentes ciclos
  
```

**030** En ejercicio de estos cas much gente, a veces por no darse cuenta de que puede parametrizarse algo.  
NO repetir código

## Java: Solución con clases parametrizables

```

1 . . .
2 class Curso < T extends Estudiante > {
3   void matricularEstudiante (T e) { . . . }
4 } // La comprobación de tipos evita matricular estudiantes de diferentes ciclos
  
```

Además por lo  
 Curso < Estudiante > c2;  
 pero solo llamaremos a métodos de Estudiante y  
 podremos recibir estudiantes de distintos tipos

# Clases parametrizables e interfaces

- La implementación de un método de una clase parametrizable puede requerir que `T` disponga de un determinado método

**Java:** Se asume que `T` tiene un determinado método

```
1 class Mazo <T> {  
2     T getCopiaPrimeraCarta () {  
3         T primeraCarta = cartas.remove (0);  
4         cartas.add (primeraCarta);  
5         return primeraCarta.copia ();  
6         // Se requiere que las clases que sustituyan a T tengan un método T copia()  
7     }  
8 }
```

- En ese caso:
  - El método requerido formará parte de una interfaz
  - Al declarar la clase parametrizable se indicará que el tipo que sustituya al parámetro debe realizar dicha interfaz

# Ejemplo de clases parametrizables e interfaces

Una de las soluciones para el mazo de cartas

## Java: Ejemplo de clases parametrizables e interfaces

```

1 // Las interfaces también pueden hacerse paramétricas, como las clases
2 interface Copiable <T> {
3     public T copia();
4 }
5
6 class Sorpresa implements Copiable<Sorpresa> {
7     // Unas cartas Sorpresa para algún juego
8     // Entre otras operaciones, implementa copia
9     public Sorpresa copia () {
10         return Sorpresa(this); // Hace uso de un constructor de copia
11     }
12 }
13
14 class Mazo < T extends Copiable<T> > { // Se requiere que T realice Copiable<T>
15     T getCopiaPrimeraCarta () {
16         T primeraCarta = cartas.remove (0);
17         cartas.add (primeraCarta);
18         return primeraCarta.copia ();
19         // primeraCarta, de tipo T, que realiza Copiable, sí dispone del método copia.
20     }
21 }
22
23 // Ya se puede instanciar un mazo de sorpresas
24 Mazo<Sorpresa> mazoSorpresas = new Mazo<>();

```

*Repara.*

*Es la misma clase pero no da problema*

# Clases parametrizables en Java

## Algunas consideraciones de bajo nivel

- No se crea una instancia de cada clase paramétrica cada vez que se utiliza con un tipo concreto
- En tiempo de ejecución el parámetro pasa a ser `Object` o del tipo que se haya puesto como restricción
- Es un mecanismo para intentar evitar ciertos errores haciendo que sean detectables en tiempo de compilación

# Ejemplo

## Java: Clase parametrizable

```
1 class Clase1 {}
2 class Clase2 {}
3
4 public class Parametrizable {
5     public static void main(String[] args) {
6         Clase1 c1;
7         Clase2 c2;
8         NodoOctal<Clase1> raiz1 = new NodoOctal<>(); // Se infiere el tipo
9         raiz1.divide();
10
11         raiz1.insertarElemento (new Clase1());
12         // raiz1.insertarElemento (new Object()); // Error compilación
13
14         c1 = raiz1.obtenerElemento (0);
15         // c2 = raiz1.obtenerElemento (0); // Error compilación
16         NodoOctal<Clase1> hijo1 = raiz1.hijo (3);
17         hijo1.insertarElemento (new Clase1());
18
19         // Simulamos que no disponemos de clases parametrizables
20         NodoOctal<Object> raiz = new NodoOctal<>();
21         raiz.insertarElemento (new Clase1());
22         c1 = (Clase1) raiz.obtenerElemento (0); // Cast necesario
23         c2 = (Clase2) raiz.obtenerElemento (0); // Error ejecución
24
25         NodoOctal<Clase1> otro1 = new NodoOctal<> (raiz1);
26         // NodoOctal<Clase2> otro2 = new NodoOctal<> (raiz1); // Error compilación
27     }
28 }
```

# Ejemplo Ilamativo

## Java: Ilustra la implementación de las clases parametrizables en Java

```
1 public static void main(String[] args) {
2
3     ArrayList<String> s = new ArrayList<>();
4     s.add ("Sorpresa");
5     // s.add (1); // Error de compilación
6
7     Object o = s;
8     ArrayList<Integer> i = (ArrayList<Integer>) o; // Warning al compilar
9     // Equivalente: ArrayList i=(ArrayList<Integer>) (Object) s;
10    i.add (5);
11    // i.add("Probando"); // Error de compilación
12
13    System.out.println (s); // [Sorpresa, 5]
14
15    for (Object c : s) {
16        System.out.println (c.getClass().getSimpleName());
17    }
18    // String, Integer !!!
19 }
```



mirar ejemplo 115.zip → generico.java  
en PRADO



# Clases e interfaces parametrizables → *Diseño* ←

- Tenerlas en cuenta en aquellos casos en los que la responsabilidad de una clase implique trabajar con objetos de clases desconocidas a priori
- Si se requiere que las clases que sustituyan el parámetro implementen unos métodos concretos, recurrir a interfaces para *obligar* a que dichas clases los implementen
- Se tiene el añadido de la comprobación de tipos en tiempo de compilación

## EXAMEN

- Ejercicio de visibilidad: encuentra los fallos en este código  
falsos positivos, falsos negativos  $\Rightarrow$  Restan  
3  $\Rightarrow$  lo mandan fuera
- Polimorfismo, encuentra los fallos  $\left\{ \begin{array}{l} \text{tpo compilación} \\ \text{tpo ejecución} \end{array} \right.$  (mucho casting)

## Clases Parametrizables

Dpto. Lenguajes y Sistemas Informáticos  
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2021-2022)