

# Clases Parametrizables

Prof. Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos  
Universidad de Granada

Programación y Diseño Orientado a Objetos

Doble Grado en Ingeniería Informática  
y Administración y Dirección de Empresas  
(Curso 2024-2025)

# 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 definir las y utilizarlas

# Contenidos

- 1 **Introducción**
- 2 **Clases parametrizables**
- 3 **Clases parametrizables en UML**
- 4 **Clases parametrizables en Java**
  - Clases parametrizables e interfaces

# 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>
```



# Ejemplo

## Java: Clase parametrizable

```

1 public class Equipo<T extends Jugador> {
2
3     private String nombre;
4     private T capitan;
5     private ArrayList<T> jugadores;
6
7     public Equipo (String nom, T cap) {
8         nombre = nom;
9         capitan = cap;
10        jugadores = new ArrayList<>();
11        jugadores.add (cap);
12    }
13
14    public T getCapitan () {
15        return capitan;
16    }
17
18    public ArrayList<T> getJugadores () {
19        return jugadores;
20    }
21
22    public void addJugador (T jug) {
23        if (!jugadores.contains (jug)) {
24            jugadores.add (jug);
25        }
26    }
27 }

```



## : Uso de la clase

```

1 public static void main(String[] args)
2 {
3     Futbolista pele;
4     pele = new Futbolista ("Pelé");
5
6     Equipo<Futbolista> brasil;
7     brasil = new Equipo<>("Brasil",pele);
8
9     Futbolista tostao;
10    tostao = new Futbolista ("Tostao");
11    brasil.addJugador (tostao);
12
13    Baloncestista gasol;
14    gasol = new Baloncestista ("Gasol");
15
16    // Error, gasol no es Futbolista
17    brasil.addJugador(gasol);
18 }

```

# 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
  
```

## 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
  
```

# 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

## 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<>();
```

# 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

# Clases Parametrizables

Prof. Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos  
Universidad de Granada

Programación y Diseño Orientado a Objetos

Doble Grado en Ingeniería Informática  
y Administración y Dirección de Empresas  
(Curso 2024-2025)