

# TEMA 3

CONCEPTOS DE JAVA PARA ESTRUCTURAS DE DATOS  
Genericidad

## Objetivos

En este tema se plantea cómo la **programación genérica** contribuye a la **reutilización** del *software*.

- Se presentará la sintaxis y la terminología necesaria para utilizar genericidad en las versiones actuales de Java
- Como bibliografía básica del tema se recomienda consultar el tutorial “*Learning the Java Language. Generics*”, publicado en *java.sun.com*:

<http://java.sun.com/docs/books/tutorial/java/generics/>

2

## Índice

### 1.- Clases genéricas

- Introducción
- Definición y uso
- Clases genéricas estándar: la clase *ArrayList*

### 2.- Restricción de tipos genéricos

### 3.- Métodos genéricos

### 4.- El interfaz *Comparable*

- El problema de la ordenación

3

## 1.- CLASES GENÉRICAS

### Introducción – Utilización de la herencia

- Todas las clases en Java heredan de la clase base *Object*. Se puede emplear *Object* para implementar algoritmos genéricos
- **Ejemplo:** diseño de una clase *Caja* que pueda guardar objetos de cualquier tipo:

```
public class Caja {  
    private Object dato;  
    public Caja() { super(); }  
    public Object dame() {  
        return dato;  
    }  
    public void pon(Object x) {  
        dato = x;  
    }  
}
```

```
public class TestCaja {  
    public static void  
    main(String args[]) {  
        Caja c = new Caja();  
        c.pon(new Integer(46));  
        Integer n;  
        n = (Integer) c.dame();  
    }  
}
```

4

## 1.- CLASES GENÉRICAS

### Introducción – Genericidad en *Java*

- El siguiente programa ¿daría error de compilación? ¿Y en ejecución?

```
public static void main(String args[]) {
    Caja c = new Caja();
    c.pon("Hola");
    Integer n = (Integer) c.dame();
}
```

- *Generics* aparece a partir del *JDK 1.5*. Ventajas:
  - Detección de errores en tiempo de compilación
  - Evitan las conversiones de tipo forzosas (*castings*)
  - Disminuyen errores y mejoran la eficiencia

5

## 1.- CLASES GENÉRICAS

### Definición y uso

```
public class Caja <T> {
    private T dato;
    public Caja() { super(); }
    public T dame() { return dato; }
    public void pon(T x) { dato = x; }
}
```

Parámetro formal de tipo

- El valor de la variable *T* puede ser cualquier clase o interfaz (nunca un tipo primitivo)
- Cuando se declara un objeto de tipo *Caja* es necesario indicar el tipo de la variable *T*. Ejemplo:

```
Caja <Integer> c;
```

6

## 1.- CLASES GENÉRICAS

### Definición y uso

- Ejemplo de uso de la clase genérica *Caja*:

```
public class TestCaja {
    public static void main(String args[]) {
        Caja <Integer> c = new Caja <Integer> ();
        c.pon(new Integer(46));
        Integer n = c.dame(); ← Sin casting!
    }
}
```

- El siguiente programa ahora sí daría error de compilación:

```
public static void main(String args[]) {
    Caja <String> c = new Caja <String> ();
    c.pon("Hola");
    Integer n = c.dame(); → Error: tipos incompatibles!
}
```

7

## 1.- CLASES GENÉRICAS

### Clases genéricas estándar: la clase *ArrayList*

- Hay multitud de clases genéricas en *Java*.
- Ejemplo: *ArrayList*, que implementa un *array* redimensionable

```
package java.util;
public class ArrayList <E> {
    public ArrayList();
    public ArrayList(int initialCapacity);
    public boolean add(E element);
    public E set(int index, E element);
    public E get(int index);
    public E remove(int index);
    public int size();
    ...
}
```

8

## 1.- CLASES GENÉRICAS

### Ejercicio

- **Ejercicio 1:** Utilizando la clase *ArrayList*, escribe un programa que realice los siguientes pasos:
  - Guarde en el array 10 objetos de la clase *Caja* que contengan números enteros. La primera caja contendrá el número 1, la segunda el 2, y así sucesivamente.
  - Recorra el array y muestre por pantalla el contenido de todas las cajas.

9

## 2.- RESTRICCIÓN DE TIPOS GENÉRICOS

### La cláusula *extends*

- En algunas ocasiones es necesario indicar que el valor del tipo genérico no puede ser cualquiera
- **Ejemplo:** en el *Garaje* sólo pueden guardarse *Vehiculos*:

```
public class Garaje <V extends Vehiculo> {
    private int numPlazas;
    private ArrayList<V> plazas;

    public Garaje(int numPlazas) {
        this.numPlazas = numPlazas;
        this.plazas = new ArrayList<V>(numPlazas);
    }
    ...
}
```

Sólo permitimos que V sea un Vehículo o una clase derivada de Vehículo

10

## 2.- RESTRICCIÓN DE TIPOS GENÉRICOS

### La cláusula *extends*

- **Ejercicio 2:** Completa la clase *Garaje* (ver el ejercicio 5 del tema 1). Falta por implementar los métodos *guardarVehiculo* y *leerCuota*.
- **Ejercicio 3:** Reescribe la clase *Caja* para exigir que su contenido sólo pueda ser numérico (es decir, una clase derivada de *Number*). Escribe un nuevo método en esta clase que devuelva la parte entera del valor numérico que contiene la caja.

11

## 3.- MÉTODOS GENÉRICOS

### Definición y uso

- Es posible definir variables de tipo específicas para un único método
- **Ejemplo:** la siguiente clase contiene un método genérico que muestra el contenido de un *ArrayList* por pantalla:

```
public class OperacionesArray {
    public static <T> void mostrarArray(ArrayList<T> a) {
        for (int i = 0; i < a.size(); i++)
            System.out.println(a.get(i).toString());
    }
}
```

Esta variable de tipo sólo puede utilizarse en este método

12

## 4.- EL INTERFAZ COMPARABLE

### Definición

- El interfaz genérico *Comparable* permite la comparación dos objetos entre sí:

```
public interface Comparable<T> {  
    public int compareTo(T x);  
}
```

Devuelve un número

< 0	, si this < x
> 0	, si this > x
== 0	, si this es igual a x

**Nota:** se recomienda que  
a.compareTo(b)==0 sea  
equivalente a a.equals(b)

13

## 4.- EL INTERFAZ COMPARABLE

### Uso

- Ejemplo: usando genericidad, el método *compareTo* de la clase *Figura* se escribe de la siguiente forma:

```
public abstract class Figura implements Comparable<Figura> {  
    ...  
    public int compareTo(Figura f) {  
        if (this.area() < f.area()) return -1;  
        else if (this.area() > f.area()) return 1;  
        else return 0;  
    }  
    ...  
}
```

14

## 4.- EL INTERFAZ COMPARABLE

### Definición y uso

- Ejercicio 4: Usando genericidad y la clase *ArrayList*, diseñar una clase *Operaciones* que:
  - Tenga un método que devuelva el elemento **mínimo** de un *ArrayList*
  - Tenga un método que **busque** un elemento en un *ArrayList* y devuelva su posición (devolverá -1 si el elemento no está en el array)
  - Tenga un método que borre la primera aparición en el *ArrayList* de un objeto dado y que devuelva el objeto eliminado (o *null* si no se encuentra)

15

## 4.- EL INTERFAZ COMPARABLE

### El problema de la ordenación

- El siguiente método permite ordenar un *array* de enteros:

```
public static void insercionDirecta(int a[]) {  
    for (int i = 1; i < a.length; i++) {  
        int elemAInsertar = a[i];  
        int posIns = i ;  
        for (; posIns > 0 && elemAInsertar < a[posIns-1]; posIns--)  
            a[posIns] = a[posIns-1];  
        a[posIns] = elemAInsertar;  
    }  
}
```

- Ejercicio 5: Añade el método *insercionDirecta* a la clase *Operaciones* del ejercicio 4, modificándolo para que permita ordenar un *ArrayList* genérico.

16

## 5.- LIMITACIONES DE LA GENERICIDAD

### Ejercicios

- Ejercicio 6: Indica qué instrucciones darían error y por qué:

```
public class TestGenericidad <T extends Figura> {
    public void prueba() {
        T c1 = new Circulo();
        T c2 = (T) new Circulo();
        T c3 = new T();
        boolean cmp = new Circulo().equals(c2);
        T v[];
        v = new T[10];
        v = new Figura[10];
        v = (T[]) new Figura[10];
    }
    public static void main() {
        T x = null;
    }
}
```

17

## 5.- LIMITACIONES DE LA GENERICIDAD

### Ejercicios

- Ejercicio 7: Suponiendo que la clase *Figura* implementa el interfaz *Comparable<Figura>*, indicar qué instrucciones son correctas, cuáles producirían error de compilación y cuáles error de ejecución:

```
ArrayList<Circulo> v1 = new ArrayList<Figura>();
ArrayList<Figura> v2 = new ArrayList<Circulo>();
ArrayList<Figura> v3 = new ArrayList<Figura>();
v3.add(new Figura());
v3.add(new Circulo());
Figura f1 = v3.get(0);
Circulo f2 = (Circulo) v3.get(0);
Rectangulo f3 = (Rectangulo) v3.get(0);
Comparable<Figura> c1 = v3.get(0);
Comparable<Circulo> c2 = v3.get(0);
```

18