

## TEMA 3

# Conceptos de Java para Estructuras de Datos

## Genericidad

### EJERCICIOS RESUELTOS

**Ejercicio 1.-** Utilizando la clase *ArrayList*, escribe un programa que realice los siguientes pasos:

1. 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.
2. Recorra el array y muestre por pantalla el contenido de todas las cajas.

#### Solución:

```
import java.util.*;

public class TestCaja {
    public static void main(String args[]) {
        // Creamos el array
        ArrayList <Caja<Integer>> v = new ArrayList <Caja<Integer>> ();

        // Añadimos las 10 cajas al array
        for (int i = 1; i <= 10; i++) {
            Caja<Integer> c = new Caja<Integer> ();
            c.pon(new Integer(i));
            v.add(c);
        }

        // Recorremos el array y mostramos su contenido
        for (Caja<Integer> c: v)
            System.out.println(c.dame());
    }
}
```

**Nota:** el recorrido del *ArrayList* se podría hacer también de esta otra forma:

```
for (int i = 0; i < v.size(); i++)
    System.out.println(v.get(i).dame());
```

**Ejercicio 2.-** Completa la clase *Garaje* (ver el ejercicio 6 del tema 1). Falta por implementar los métodos *guardarVehiculo* y *leerCuota*.

**Solución:**

```
import java.util.*;

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);
    }
    public void guardarVehiculo(V v, int plaza) {
        if (plaza >= 0 && plaza < numPlazas)
            plazas.set(plaza, v);
    }
    public int leerCuota(int plaza) {
        int cuota = 0;
        if (plaza >= 0 && plaza < numPlazas) {
            V v = plazas.get(plaza);
            if (v != null) {
                if (v instanceof Coche)
                    cuota = v.potencia() * ((Coche) v).numAsientos();
                else if (v instanceof Moto)
                    cuota = 2 * v.potencia();
            }
        }
        return cuota;
    }
}
```

**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.

**Solución:**

```
public class Caja <T extends Number> {
    private T dato;
    public Caja() {
        super();
    }
    public T dame() {
        return dato;
    }
    public void pon(T x) {
        dato = x;
    }
    public int valorEntero() {
        return dato.intValue();
    }
}
```

**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).

**Solución:**

```
import java.util.*;

public class Operaciones {

    public static <T extends Comparable<T>> T minimo(ArrayList<T> v) {
        if (v.size() == 0) return null;
        T min = v.get(0);
        for (int i = 1; i < v.size(); i++)
            if (v.get(i).compareTo(min) < 0) min = v.get(i);
        return min;
    }

    public static <T> int buscar(T x, ArrayList<T> v) {
        int pos = -1;
        for (int i = 0; i < v.size() && pos == -1; i++)
            if (v.get(i).equals(x)) pos = i;
        return pos;
    }

    public static <T> T borrar(T x, ArrayList<T> v) {
        int pos = buscar(x, v);
        if (pos == -1) return null;
        return v.remove(pos);
    }
}
```

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

**Solución:**

```
public static <T extends Comparable<T>> void insercionDirecta(ArrayList<T> a) {
    for (int i = 1; i < a.size(); i++) {
        T elemAInsertar = a.get(i);
        int posIns = i ;
        for (; posIns > 0 && elemAInsertar.compareTo(a.get(posIns-1)) < 0; posIns--)
            a.set(posIns, a.get(posIns-1));
        a.set(posIns, elemAInsertar);
    }
}
```

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

```
1.  public class TestGenericidad <T extends Figura> {
2.      public void prueba() {
3.          T c1 = new Circulo();
4.          T c2 = (T) new Circulo();
5.          T c3 = new T();
6.          boolean cmp = new Circulo().equals(c2);
7.          T v[];
8.          v = new T[10];
9.          v = new Figura[10];
10.         v = (T[]) new Figura[10];
11.     }
12.     public static void main() {
13.         T x = null;
14.     }
15. }
```

### Solución:

Línea 3: tipos incompatibles (T y Circulo).

Línea 5: no pueden crearse objetos de tipo genérico T.

Línea 8: no se permite la creación de *arrays* genéricos.

Línea 9: tipos incompatibles (T[ ] y Figura[ ]).

Línea 13: la variable T no es accesible en métodos estáticos.

### 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:

```
1.  ArrayList<Circulo> v1 = new ArrayList<Figura>();
2.  ArrayList<Figura> v2 = new ArrayList<Circulo>();
3.  ArrayList<Figura> v3 = new ArrayList<Figura>();
4.  v3.add(new Figura());
5.  v3.add(new Circulo());
6.  Figura f1 = v3.get(0);
7.  Circulo f2 = (Circulo) v3.get(0);
8.  Rectangulo f3 = (Rectangulo) v3.get(0);
9.  Comparable<Figura> c1 = v3.get(0);
10. Comparable<Circulo> c2 = v3.get(0);
```

### Solución:

Línea 1: *Error de compilación*: tipos incompatibles (ArrayList<Circulo> y ArrayList<Figura>).

Línea 2: *Error de compilación*: aunque un Circulo es una Figura, un ArrayList<Circulo> no es un ArrayList<Figura>.

Línea 4: *Error de compilación*: la clase Figura es abstracta.

Línea 8: *Error de ejecución*: un Circulo no es un Rectangulo.

Línea 10: *Error de compilación*: la clase Figura y sus derivadas implementan la interfaz Comparable<Figura>, no Comparable<Circulo>.

**Ejercicio 8 [adicional].-** Corrige los errores del siguiente código (la clase *Caja<T>* es la que figura en los apuntes y la excepción de usuario *ExplosionFallida* se supone correctamente implementada):

```
public interface Explosivo {
    void explotar();
}

public class Bomba implements Explosivo {
    public void explotar() throws ExplosionFallida {
        if (Math.random() < 0.2)
            throw new ExplosionFallida("Bomba fallida");
        System.out.println("Explosión!!!");
    }
}

public class PaqueteBomba extends Caja<Bomba> {
    public void abrirPaquete() {
        try {
            Thread.currentThread().sleep(1000);
            activarBomba();
        } catch (Exception e) { System.out.println(e); }
    }
    public void activarBomba() {
        dame().explotar();
    }
}
```

### Solución:

```
public interface Explosivo {
    void explotar() throws ExplosionFallida;
}

public class Bomba implements Explosivo {
    public void explotar() throws ExplosionFallida {
        if (Math.random() < 0.2)
            throw new ExplosionFallida("Bomba fallida");
        System.out.println("Explosión!!!");
    }
}

public class PaqueteBomba extends Caja<Bomba> {
    public void abrirPaquete() {
        try {
            Thread.currentThread().sleep(1000);
            activarBomba();
        } catch (Exception e) { System.out.println(e); }
    }
    public void activarBomba() throws ExplosionFallida {
        dame().explotar();
    }
}
```