Programación

Act09_Interfaces

Francisco José García Cutillas | 1FPGS_DAM

Índice

Ejercicio A01 Futbolista

- **Diseñar la clase Futbolista que tendrá los siguientes atributos: dni, nombre, edad y nº de goles.**

```java
package clases;

/**
 * Clase Futbolista
 * @author Fran
 */
public class Futbolista {

    private String dni;
    private String nombre;
    private int edad;
    private int num_goles;
```

- **Debemos tener un constructor en el que no se le pase el número de goles y este será cero (recuerda hacer uso de this) así como encapsulamiento básico.**

```java
    /**
     * Constructor de la clase que recibe todos los parámetros
     * @param _dni
     * @param _nombre
     * @param _edad
     * @param _num_goles
     */
    public Futbolista (String _dni, String _nombre, int _edad, int _num_goles){

        this.dni = _dni;
        this.nombre = _nombre;
        this.edad = _edad;
        this.num_goles = _num_goles;

    }

    /**
     * Constructor de la clase que recibe todos los parámetros menos el
     * número de goles que será por defecto 0
     * @param _dni
     * @param _nombre
     * @param _edad
     */
    public Futbolista (String _dni, String _nombre, int _edad){

        this.dni = _dni;
        this.nombre = _nombre;
        this.edad = _edad;
        this.num_goles = 0;

    }
```

```java
    /**
     * Método para establecer un nuevo DNI al Futbolista
     *
     */
    public void setDni(String dni) {
        this.dni = dni;
    }

    /**
     * Método para obtener el nombre del Futbolista
     * @return Nombre del Futbolista
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Método para establecer un nuevo nombre al Futbolista
     *
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Método para obtener la edad del Futbolista
     * @return Edad del Futbolista
     */
    public int getEdad() {
        return edad;
    }

    /**
     * Método para establecer una nueva edad al Futbolista
     *
     */
    public void setEdad(int edad) {
        this.edad = edad;
    }

    /**
     * Método para obtener el número de goles del Futbolista
     * @return Número de goles del Futbolista
     */
    public int getNum_goles() {
        return num_goles;
    }

    /**
     * Método para establecer un nuevo número de goles al Futbolista
     *
     */
    public void setNum_goles(int num_goles) {
        this.num_goles = num_goles;
    }
```

- **Métodos toString() modificado por ti y equals (el dni determina si dos futbolistas son el mismo).**

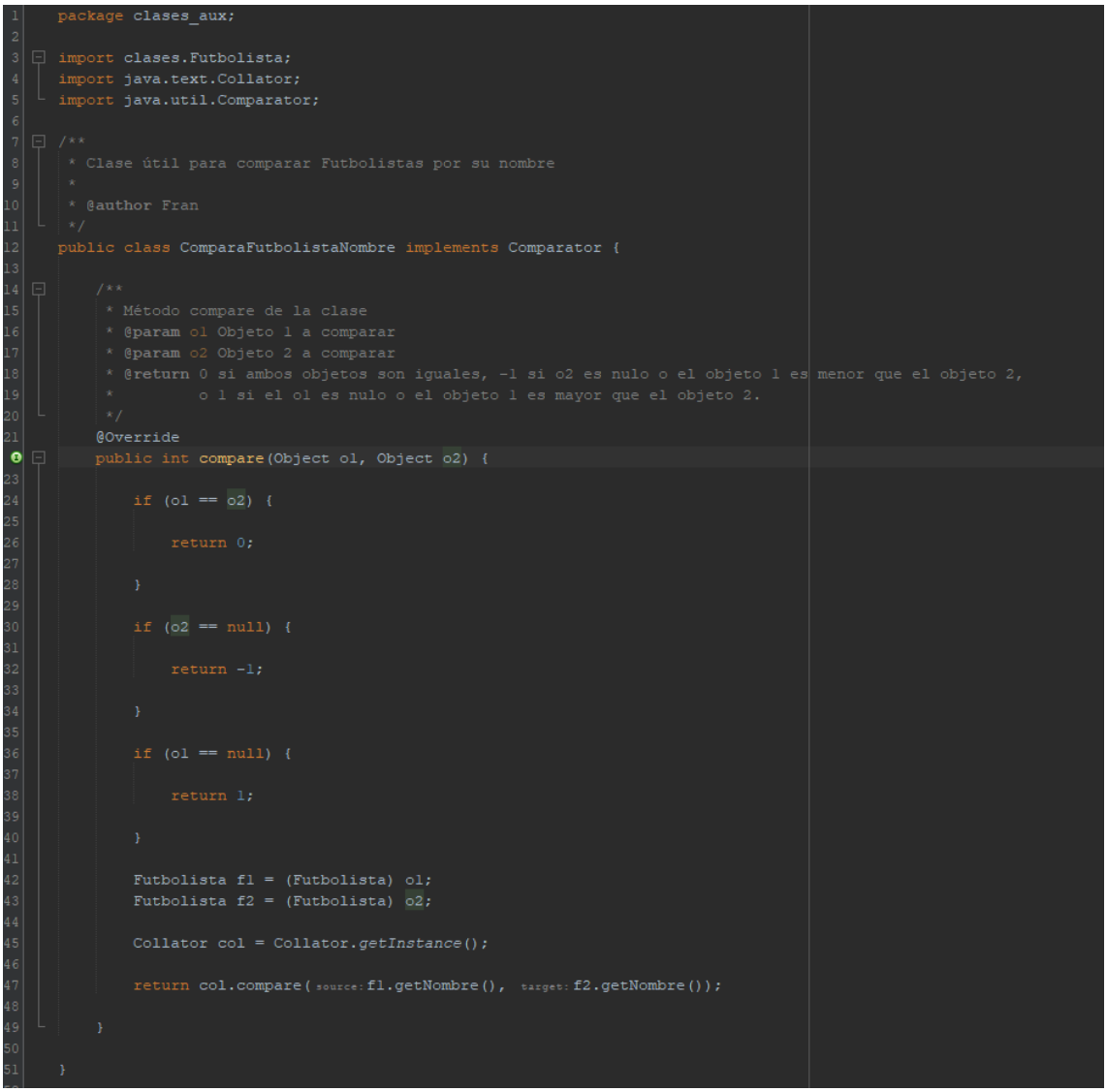
```
47
48      /**
49       * Método toString de la clase
50       * @return Devuelve los datos del futbolista
51       */
52      @Override
        public String toString(){
54
55          return "DNI: " + this.dni + ", Nombre: " + this.nombre +
56                  ", Edad: " + this.edad + ", Número de goles: " +  this.num_goles;
57
58      }
59
60      /**
61       * Método equals de la clase
62       * @param obj Objeto a comparar
63       * @return Devuelve true si el objeto es igual que el introducido por parámetro.
64       * Dos futbolistas son iguales si tienen el mismo DNI
65       */
66      @Override
        public boolean equals(Object obj) {
68          if (this == obj) {
69              return true;
70          }
71          if (obj == null) {
72              return false;
73          }
74          if (getClass() != obj.getClass()) {
75              return false;
76          }
77          Futbolista other = (Futbolista) obj;
78          return Objects.equals( a: this.dni,  b: other.dni);
79      }
```

- **Implementa la interfaz Comparable con un criterio de ordenación basado en el dni.**

```
10  public class Futbolista implements Comparable{
11
```

```
82      /**
83       * Método compareTo para comparar dos DNIs
84       * @param o Objeto con el que queremos comparar
85       * @return Número negativo si el objeto es nulo o si el DNI es menor que el introducido, 0 si
86       *         los DNis son iguales o son el mismo objeto o un número positivo si el DNI
87       *         es mayor que el introducido
88       */
89      @Override
90      public int compareTo(Object o) {
91
92          if (this == o) {
93
94              return 0;
95
96          }
97
98          if (o == null) {
99
100             return -1;
101
102         }
103
104         Futbolista f = (Futbolista) o;
105
106         Collator col = Collator.getInstance();
107
108         return col.compare( source: this.getDni(),  target: f.getDni());
109
110     }
111
```

- **Un comparador para hacer ordenaciones basadas en el nombre y otro basado en la edad (en el caso de tener la misma edad, debe ser el nombre el que determine quién va antes).**



```java
package clases_aux;

import clases.Futbolista;
import java.text.Collator;
import java.util.Comparator;

/**
 * Clase útil para comparar Futbolistas por su nombre
 *
 * @author Fran
 */
public class ComparaFutbolistaNombre implements Comparator {

    /**
     * Método compare de la clase
     * @param o1 Objeto 1 a comparar
     * @param o2 Objeto 2 a comparar
     * @return 0 si ambos objetos son iguales, -1 si o2 es nulo o el objeto 1 es menor que el objeto 2,
     *         o 1 si el o1 es nulo o el objeto 1 es mayor que el objeto 2.
     */
    @Override
    public int compare(Object o1, Object o2) {

        if (o1 == o2) {

            return 0;

        }

        if (o2 == null) {

            return -1;

        }

        if (o1 == null) {

            return 1;

        }

        Futbolista f1 = (Futbolista) o1;
        Futbolista f2 = (Futbolista) o2;

        Collator col = Collator.getInstance();

        return col.compare( source: f1.getNombre(),  target: f2.getNombre());

    }

}
```

```java
package clases_aux;

import clases.Futbolista;
import java.util.Comparator;

/**
 * Clase útil para comparar futbolistas por su edad. En caso de que la edad sea
 * igual, compara por nombre
 *
 * @author Fran
 */
public class ComparaFutbolistaEdad implements Comparator {

    /**
     * Método compare de la clase
     *
     * @param o1 Objeto 1 a comparar
     * @param o2 Objeto 2 a comparar
     * @return 0 si ambos objetos son iguales, -1 si o2 es nulo o el objeto 1 es
     * menor que el objeto 2, o 1 si el o1 es nulo o el objeto 1 es mayor que el
     * objeto 2.
     */
    @Override
    public int compare(Object o1, Object o2) {

        if (o1 == o2) {

            return 0;

        }

        if (o2 == null) {

            return -1;

        }

        if (o1 == null) {

            return 1;

        }

        Futbolista f1 = (Futbolista) o1;
        Futbolista f2 = (Futbolista) o2;

        if (f1.getEdad() == f2.getEdad()) {

            ComparaFutbolistaNombre c1 = new ComparaFutbolistaNombre();

            return c1.compare(o1, o2);

        } else if (f1.getEdad() > f2.getEdad()) {

            return 1;

        } else {

            return -1;

        }
```

- **Crea una tabla con 6 futbolistas y muéstralos ordenados por dni, nombre y edad**

```java
package com.mycompany.garciacutillasfranciscojose_act09_interfaces_ejercicio1;

import clases.Futbolista;
import clases_aux.ComparaFutbolistaEdad;
import clases_aux.ComparaFutbolistaNombre;
import java.util.Arrays;

/**
 *
 * @author Fran
 */
public class GarciaCutillasFranciscoJose_Act09_Interfaces_Ejercicio1 {

    public static void main(String[] args) {

        Futbolista f1 = new Futbolista( _dni: "11111111A", _nombre: "Pepe", _edad: 22, _num_goles: 3);
        Futbolista f2 = new Futbolista( _dni: "22222222B", _nombre: "Antonio", _edad: 19);
        Futbolista f3 = new Futbolista( _dni: "63333333C", _nombre: "Paula", _edad: 22);
        Futbolista f4 = new Futbolista( _dni: "44444444D", _nombre: "Ana", _edad: 17, _num_goles: 11);
        Futbolista f5 = new Futbolista( _dni: "55111111E", _nombre: "Zinedine", _edad: 19, _num_goles: 6);
        Futbolista f6 = new Futbolista( _dni: "11111221F", _nombre: "Xavi", _edad: 29);

        Futbolista[] arrayFutbolistas = {f6, f3, f2, f5, f1, f4};
        Arrays.sort( a: arrayFutbolistas);

        System.out.println( x: "\nFutbolistas ordenados por DNI");

        for (int i = 0; i < arrayFutbolistas.length; i++) {

            System.out.println(arrayFutbolistas[i]);

        }

        ComparaFutbolistaNombre cfn = new ComparaFutbolistaNombre();
        Arrays.sort( a: arrayFutbolistas, c: cfn);

        System.out.println( x: "\nFutbolistas ordenados por nombre");

        for (int i = 0; i < arrayFutbolistas.length; i++) {

            System.out.println(arrayFutbolistas[i]);

        }

        ComparaFutbolistaEdad cfe = new ComparaFutbolistaEdad();
        Arrays.sort( a: arrayFutbolistas, c: cfe);

        System.out.println( x: "\nFutbolistas ordenados por edad");

        for (int i = 0; i < arrayFutbolistas.length; i++) {

            System.out.println(arrayFutbolistas[i]);

        }

    }
}
```

```
Futbolistas ordenados por DNI
DNI: 11111111A, Nombre: Pepe, Edad: 22, Número de goles: 3
DNI: 11111221F, Nombre: Xavi, Edad: 29, Número de goles: 0
DNI: 22222222B, Nombre: Antonio, Edad: 19, Número de goles: 0
DNI: 44444444D, Nombre: Ana, Edad: 17, Número de goles: 11
DNI: 55111111E, Nombre: Zinedine, Edad: 19, Número de goles: 6
DNI: 63333333C, Nombre: Paula, Edad: 22, Número de goles: 0

Futbolistas ordenados por nombre
DNI: 44444444D, Nombre: Ana, Edad: 17, Número de goles: 11
DNI: 22222222B, Nombre: Antonio, Edad: 19, Número de goles: 0
DNI: 63333333C, Nombre: Paula, Edad: 22, Número de goles: 0
DNI: 11111111A, Nombre: Pepe, Edad: 22, Número de goles: 3
DNI: 11111221F, Nombre: Xavi, Edad: 29, Número de goles: 0
DNI: 55111111E, Nombre: Zinedine, Edad: 19, Número de goles: 6

Futbolistas ordenados por edad
DNI: 44444444D, Nombre: Ana, Edad: 17, Número de goles: 11
DNI: 22222222B, Nombre: Antonio, Edad: 19, Número de goles: 0
DNI: 55111111E, Nombre: Zinedine, Edad: 19, Número de goles: 6
DNI: 63333333C, Nombre: Paula, Edad: 22, Número de goles: 0
DNI: 11111111A, Nombre: Pepe, Edad: 22, Número de goles: 3
DNI: 11111221F, Nombre: Xavi, Edad: 29, Número de goles: 0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.341 s
Finished at: 2023-04-24T19:00:58+02:00
------------------------------------------------------------------------
```

Ejercicio A02. Cartas

- **Las cartas de una baraja española están formadas por un palo y un número (Valor entre 1 y 12).**

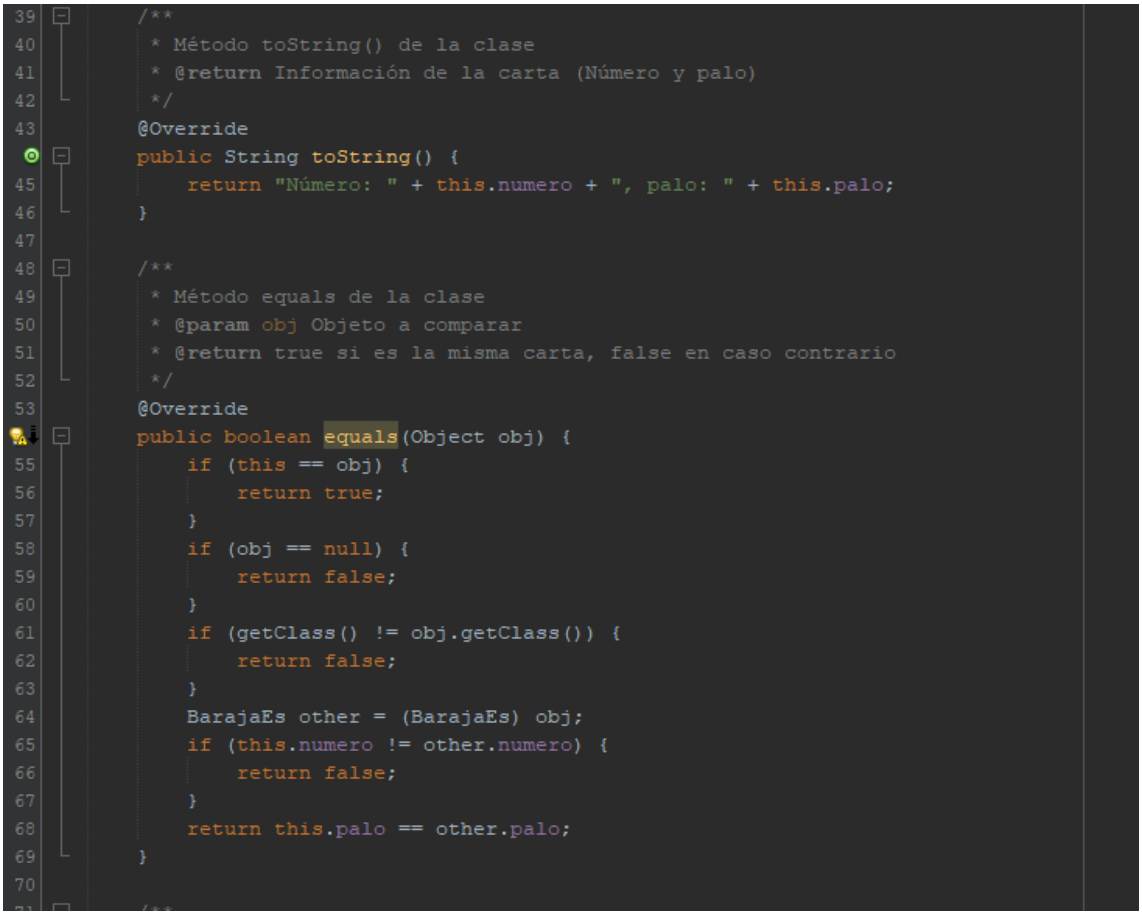
```java
package clases;

/**
 * Clase Baraja Española
 * @author Fran
 */
public class BarajaEs {

    public enum Palo {

        OROS, BASTOS, COPAS, ESPADAS;

    }

    private Palo palo;
    private int numero;
```

- **Diseñar la clase con su constructor y el encapsulamiento básico. Además debe tener el método toString() y el método equals (tú mismo debes determinar el motivo para que 2 cartas son iguales).**

```java
    /**
     * Constructor de la clase
     * @param _palo Palo de la carta a crear (Oros, Bastos, Copas, Espadas)
     * @param _numero Número de la carta a crear del 1 al 12
     */
    public BarajaEs (Palo _palo, int _numero){

        this.palo = _palo;

        if (_numero >= 1 && _numero <= 12) {

            this.numero = _numero;

        } else {

            System.out.println( x: "El número de la baraja debe estar entre 1 y 12");

        }

    }
```

```java
    /**
     * Método para obtener el palo de la carta
     * @return Palo de la carta
     */
    public Palo getPalo() {
        return palo;
    }

    /**
     * Método para establecer un nuevo palo a la carta
     * @param palo Nuevo palo
     */
    public void setPalo(Palo palo) {
        this.palo = palo;
    }

    /**
     * Método para obtener el número de la carta
     * @return Número de la carta
     */
    public int getNumero() {
        return numero;
    }

    /**
     * Método para establecer un nuevo número a la carta
     * @param numero Nuevo número del 1 al 12
     */
    public void setNumero(int numero) {

        if (numero >= 1 && numero <= 12) {

            this.numero = numero;

        } else{

            System.out.println( x: "El número debe ser entre 1 y 12");

        }

    }
```

En nuestro caso vamos a considerar que dos cartas son iguales si tienen el mismo número y el mismo palo. Aunque en una baraja simple sólo hay una carta de cada tipo, lo vamos a determinar de esta forma.



```java
    /**
     * Método toString() de la clase
     * @return Información de la carta (Número y palo)
     */
    @Override
    public String toString() {
        return "Número: " + this.numero + ", palo: " + this.palo;
    }

    /**
     * Método equals de la clase
     * @param obj Objeto a comparar
     * @return true si es la misma carta, false en caso contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        BarajaEs other = (BarajaEs) obj;
        if (this.numero != other.numero) {
            return false;
        }
        return this.palo == other.palo;
    }
```

- **Construir las clases necesarias para que sea posible hacer la ordenación de un array de cartas tanto por palo y número o únicamente por número (sin importar el palo).**

```java
package clases_aux;

import clases.BarajaEs;
import java.text.Collator;
import java.util.Comparator;

/**
 * Clase para comparar cartas por palo y número
 * @author Fran
 */
public class ComparaCartaPaloNum implements Comparator {

    /**
     * Método compare de la Interfaz Comparator
     *
     * @param o1 Objeto 1 a comparar
     * @param o2 Objeto 2 a comparar
     * @return 0 si las dos cartas son del mismo número o palo, 1 si o1 es mayor que o2
     * ó -1 si o1 es menor que o2
     */
    @Override
    public int compare(Object o1, Object o2) {

        if (o1 == o2) {

            return 0;

        }

        if (o2 == null) {

            return -1;

        }

        if (o1 == null) {

            return 1;

        }

         BarajaEs b1 = (BarajaEs) o1;
         BarajaEs b2 = (BarajaEs) o2;

         if (b1.getPalo() == b2.getPalo()) {

             ComparaCartaNum c1 = new ComparaCartaNum();

             return c1.compare(o1, o2);

         }

         Collator col = Collator.getInstance();

         return col.compare( source: b1.getPalo().toString(),  target: b2.getPalo().toString());


    }

}
```

```java
package clases_aux;

import clases.BarajaEs;
import java.util.Comparator;

/**
 * Clase para comparar cartas por número
 *
 * @author Fran
 */
public class ComparaCartaNum implements Comparator {

    /**
     * Método compare de la Interfaz Comparator
     *
     * @param o1 Objeto 1 a comparar
     * @param o2 Objeto 2 a comparar
     * @return 0 si las dos cartas son del mismo número, 1 si o1 es mayor que o2
     * ó -1 si o1 es menor que o2
     */
    @Override
    public int compare(Object o1, Object o2) {

        if (o1 == o2) {

            return 0;

        }

        if (o2 == null) {

            return -1;

        }

        if (o1 == null) {

            return 1;

        }

        BarajaEs b1 = (BarajaEs) o1;
        BarajaEs b2 = (BarajaEs) o2;

        if (b1.getNumero() == b2.getNumero()) {

            return 0;

        } else if (b1.getNumero() > b2.getNumero()) {

            return 1;

        } else {

            return -1;

        }

    }

}
```

- **Crea un método ESTÁTICO que genere una carta al azar (que sea válida)**

```
74
75      /**
76       * Método que genera una carta al azar
77       * @return Carta de un número y palo generado aleatoriamente
78       */
79      public static BarajaEs generaCarta(){
80
81          int palo = (int) (Math.floor(Math.random()*(4-1+1)+1));
82          int numero = (int) (Math.floor(Math.random()*(12-1+1)+1));
83
84          Palo paloSeleccionado;
85
86          if (palo == 1) {
87
88              paloSeleccionado = Palo.BASTOS;
89
90          } else if (palo == 2) {
91
92              paloSeleccionado = Palo.COPAS;
93
94          } else if (palo == 3) {
95
96              paloSeleccionado = Palo.ESPADAS;
97
98          } else {
99
00              paloSeleccionado = Palo.OROS;
01
02          }
03
04          BarajaEs cartaAleatoria = new BarajaEs( _palo: paloSeleccionado,  _numero: numero);
05
06          return cartaAleatoria;
07
08      }
```

- **Haciendo uso del método estático en el programa principal general 10 cartas y muestra estas ordenadas por palo y número y luego las mismas cartas ordenadas únicamente por el número.**

```java
package com.mycompany.garciacutillasfranciscojose_act09_interfaces_ejercicio2;

import clases.BarajaEs;
import clases_aux.ComparaCartaNum;
import clases_aux.ComparaCartaPaloNum;
import java.util.Arrays;

/**
 * Clase principal
 * @author Fran
 */
public class GarciaCutillasFranciscoJose_Act09_Interfaces_Ejercicio2 {

    public static void main(String[] args) {

        //Genera un array de 10 elementos y rellena con cartas aleatorias
        BarajaEs[] arrayCartas = new BarajaEs[10];

        for (int i = 0; i < arrayCartas.length; i++) {

            arrayCartas[i] = BarajaEs.generaCarta();

        }

        //Ordena el array aleatorio por palo y número
        ComparaCartaPaloNum ccpn = new ComparaCartaPaloNum();
        Arrays.sort( a: arrayCartas,  c: ccpn);

        System.out.println( x: "\n**Cartas ordenadas por palo y número**");
        for (int i = 0; i < arrayCartas.length; i++) {

            System.out.println(arrayCartas[i]);

        }

        //Ordena el array aleatorio por número
        ComparaCartaNum ccn = new ComparaCartaNum();
        Arrays.sort( a: arrayCartas,  c: ccn);

        System.out.println( x: "\n**Cartas ordenadas por número**");
        for (int i = 0; i < arrayCartas.length; i++) {

            System.out.println(arrayCartas[i]);

        }

    }
}
```

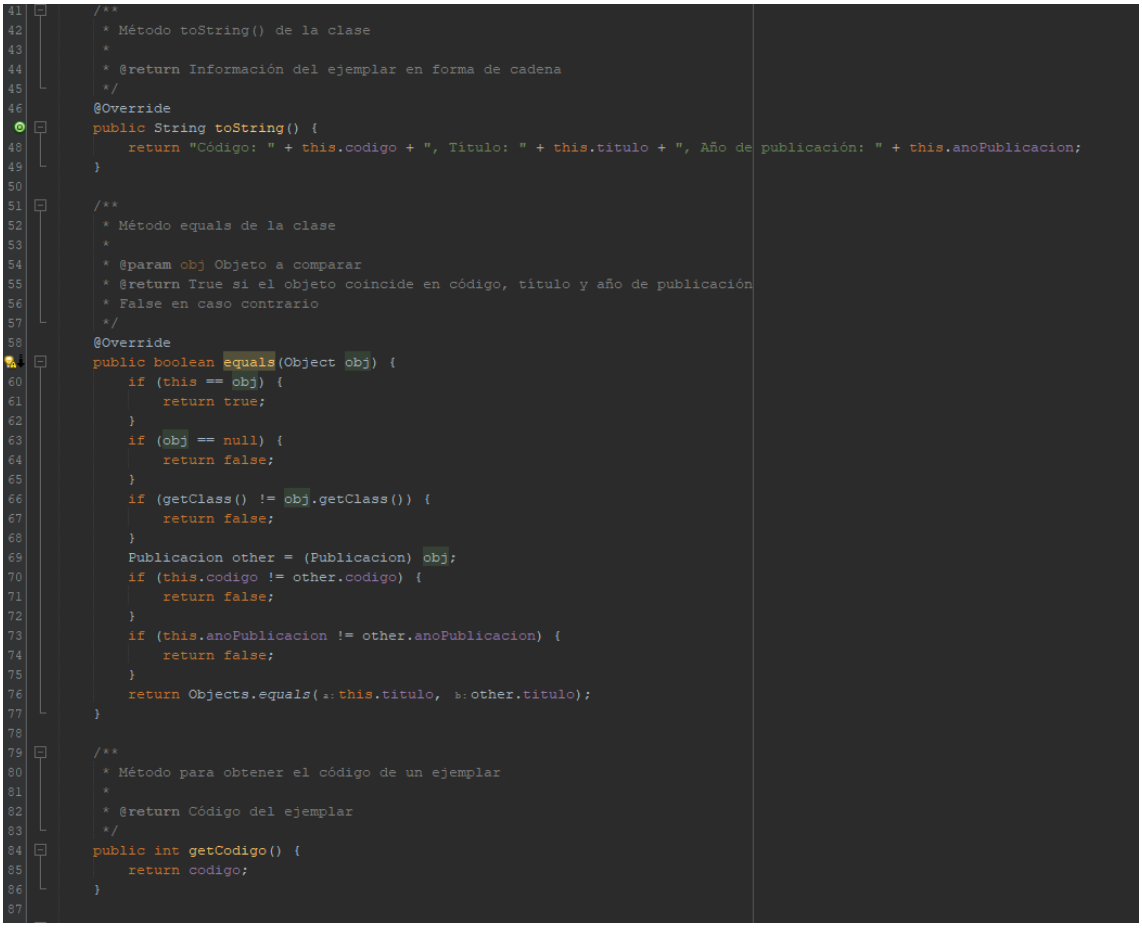
```
**Cartas ordenadas por palo y número**
Número: 3, palo: BASTOS
Número: 4, palo: BASTOS
Número: 9, palo: BASTOS
Número: 12, palo: BASTOS
Número: 6, palo: COPAS
Número: 7, palo: COPAS
Número: 6, palo: ESPADAS
Número: 1, palo: OROS
Número: 4, palo: OROS
Número: 6, palo: OROS

**Cartas ordenadas por número**
Número: 1, palo: OROS
Número: 3, palo: BASTOS
Número: 4, palo: BASTOS
Número: 4, palo: OROS
Número: 6, palo: COPAS
Número: 6, palo: ESPADAS
Número: 6, palo: OROS
Número: 7, palo: COPAS
Número: 9, palo: BASTOS
Número: 12, palo: BASTOS
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.317 s
Finished at: 2023-04-25T16:50:39+02:00
```

Ejercicio A03. Biblioteca

Escribe un programa para una biblioteca que contenga libros y revistas

- **Encapsulamiento básico en todas las clases, junto con los métodos toString y equals**
- **Las características comunes que se almacenan tanto para las revistas como para los libros son el código, el título, y el año de publicación. Estas tres características se pasan por parámetro en el momento de crear los objetos. Todas estas características se van a implementar en una superclase de la que no se deben poder crear objetos (abstracta) que va a contener todo lo común a libro y revista que va a llevar por nombre Publicación (recomiendo crear las clases sin tilde).**

```java
package clases;

import java.time.LocalDate;
import java.util.Objects;

/**
 * Clase Publicación. Superclase de Libro y Revista de tipo abstracta
 *
 * @author Fran
 */
public abstract class Publicacion {

    private int codigo;
    private String titulo;
    private int anoPublicacion;

    /**
     * Constructor de la clase
     *
     * @param _codigo Código del ejemplar
     * @param _titulo Título del ejemplar
     * @param _anoPublicacion Año de publicación del ejemplar
     */
    Publicacion(int _codigo, String _titulo, int _anoPublicacion) {

        this.codigo = _codigo;
        this.titulo = _titulo;

        if (_anoPublicacion > 0 && _anoPublicacion <= LocalDate.now().getYear()) {

            this.anoPublicacion = _anoPublicacion;

        } else {

            System.out.println( x: "No has introducido un año correcto");

        }

    }
```



```java
    /**
     * Método toString() de la clase
     *
     * @return Información del ejemplar en forma de cadena
     */
    @Override
    public String toString() {
        return "Código: " + this.codigo + ", Título: " + this.titulo + ", Año de publicación: " + this.anoPublicacion;
    }

    /**
     * Método equals de la clase
     *
     * @param obj Objeto a comparar
     * @return True si el objeto coincide en código, título y año de publicación
     * False en caso contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Publicacion other = (Publicacion) obj;
        if (this.codigo != other.codigo) {
            return false;
        }
        if (this.anoPublicacion != other.anoPublicacion) {
            return false;
        }
        return Objects.equals( a: this.titulo, b: other.titulo);
    }

    /**
     * Método para obtener el código de un ejemplar
     *
     * @return Código del ejemplar
     */
    public int getCodigo() {
        return codigo;
    }

```

```java
    /**
     * Método para establecer un nuevo código al ejemplar
     *
     * @param codigo Nuevo código
     */
    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    /**
     * Método para obtener el título del ejemplar
     *
     * @return Título del ejemplar
     */
    public String getTitulo() {
        return titulo;
    }

    /**
     * Método para establecer un nuevo título al ejemplar
     *
     * @param titulo Nuevo título
     */
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    /**
     * Método para obtener el año de publicación del ejemplar
     *
     * @return Año de publicación
     */
    public int getAnoPublicacion() {
        return anoPublicacion;
    }

    /**
     * Método para establecer un nuevo año de publicación del ejemplar
     *
     * @param anoPublicacion Nuevo año de publicación
     */
    public void setAnoPublicacion(int anoPublicacion) {
        this.anoPublicacion = anoPublicacion;
    }

}
```

- **Los libros tienen además un atributo prestado. Los libros cuando se crean no están prestados.**

```java
package clases;

/**
 * Clase Libro que hereda de Publicación
 *
 * @author Fran
 */
public class Libro extends Publicacion {

    private boolean prestado;

    /**
     * Constructor de la clase
     *
     * @param _codigo Codigo del libro
     * @param _titulo Título del libro
     * @param _anoPublicacion Año de publicación del libro
     */
    public Libro(int _codigo, String _titulo, int _anoPublicacion) {

        super(_codigo, _titulo, _anoPublicacion);
        this.prestado = false;

    }

    /**
     * Método toString() de la clase
     *
     * @return Información del libro en una cadena
     */
    @Override
    public String toString() {
        return super.toString() + "Prestado: " + prestadoCadena( estadoLibro: this.prestado);
    }

    /**
     * Método para convertir el boolean del estado del libro en cadena
     *
     * @param libro Estado del libro
     * @return Si el estado es true, devuelve "Sí", en caso contrario devuelve
     * "No"
     */
    private String prestadoCadena(boolean estadoLibro) {

        if (estadoLibro) {

            return "Si";

        }

        return "No";

    }
    /**
     * Método que devuelve si el libro está prestado o no
     *
     * @return True si está prestado, False en caso contrario
     */
    public boolean GetPrestado() {
        return prestado;
    }

    /**
     * Método para establecer si un libro está prestado o no
     *
     * @param prestado Estado del libro (Prestado: True, no prestado: False)
     */
    public void setPrestado(boolean prestado) {
        this.prestado = prestado;
    }

}
```

En el caso de los libros, vamos a considerar que dos libros son iguales si tienen el mismo código, título y año de publicación. Por lo tanto, sin tener en cuenta si dicho libro está prestado o no. Por ello, en la clase Libro no vamos a implementar el método equals, puesto que va a utilizar el mismo que el de la superclase Publicación.

- **Las revistas tienen un número. En el momento de crear las revistas se pasa el número por parámetro.**

```java
package clases;

/**
 * Clase Revista que hereda de Publicación
 *
 * @author Fran
 */
public class Revista extends Publicacion {

    private int numero;

    /**
     * Constructor de la clase
     *
     * @param _codigo Código de la revista
     * @param _titulo Título de la revista
     * @param _anoPublicacion Año de publicación de la revista
     * @param _numero Número de la revista
     */
    public Revista(int _codigo, String _titulo, int _anoPublicacion, int _numero) {

        super(_codigo, _titulo, _anoPublicacion);
        this.numero = _numero;

    }

    /**
     * Método toString() de la clase
     *
     * @return Información de la revista en una cadena
     */
    @Override
    public String toString() {
        return super.toString() + ", Número: " + this.numero;
    }

```

```java
    /**
     * Método equals de la clase
     *
     * @param obj Objeto a comparar
     * @return True si dos revistas coinciden en código, título, año de
     * publicación y número, False en caso contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Revista other = (Revista) obj;
        if (super.getCodigo() != other.getCodigo()) {
            return false;
        }
        if (super.getAnoPublicacion() != other.getAnoPublicacion()) {
            return false;
        }
        if (!super.getTitulo().equals( anObject: other.getTitulo())) {
            return false;
        }

        return this.numero == other.getNumero();

    }

    /**
     * Método que devuelve el número de la revista
     *
     * @return Número de la revista
     */
    public int getNumero() {
        return numero;
    }

    /**
     * Método para establecer un nuevo número a la revista
     *
     * @param numero Nuevo número a establecer
     */
    public void setNumero(int numero) {
        this.numero = numero;
    }

}
```

- **Para prevenir posibles cambios en el programa se tiene que implementar una interfaz Prestable con los métodos prestar, devolver y prestado. Los dos primeros métodos permiten prestar un libro, devolverlo o consultar el estado en el que se encuentra.**

```java
package interfaces;

/**
 * Interfaz Prestable
 * @author Fran
 */
public interface Prestable {

    /**
     * Método para prestar libros
     */
    public void prestar();

    /**
     * Método para devolver libros
     */
    public void devolver ();

    /**
     * Método para consultar estado de los libros
     * @return Si el libro está disponible o no
     */
    public String prestado();

}
```

- **La clase libro debe implementar esta interfaz que hemos definido.**

```java
/**
 * Clase Libro que hereda de Publicación
 *
 * @author Fran
 */
public class Libro extends Publicacion implements Prestable {
```

```
    /**
     * Método de la interfaz Prestable que nos permite poner un libro como prestado
     */
    @Override
    public void prestar() {

        this.prestado = true;

    }
    /**
     * Método de la interfaz Prestable que nos permite poner un libro como disponible
     */
    @Override
    public void devolver() {

        this.prestado = false;

    }

    /**
     * Método de la interfaz Prestable que nos permite consultar el estado de un libro
     * @return Prestado si el libro no está disponible, o disponible en caso contrario
     */
    @Override
    public String prestado() {

        if (this.prestado) {

            return "Prestado";

        }

        return "Disponible";

    }

}
```

- **Escribe una clase Biblioteca que va a estar formada por un array que va a ser de tipo Publicación (podrá almacenar tanto Libros como Revistas). Una biblioteca se crea en el momento de recibir su primera Publicación. Aunque un array tendrá un tamaño fijo, es posible hacerlo crecer como vimos en los materiales de clase).**

```java
package clases;

import java.util.Arrays;

/**
 * Clase Biblioteca
 * @author Fran
 */
public class Biblioteca {

    private Publicacion[] publicaciones;

    /**
     * Constructor de la clase
     * @param _publicaciones Array de publicaciones. Pueden ser tanto libros como revistas
     */
    public Biblioteca (Publicacion[] _publicaciones){

        this.publicaciones = _publicaciones;

    }


    /**
     * Método equals de la clase
     * @param obj Objeto a comparar
     * @return True si dos bibliotecas tienen las mismas publicaciones,
     *         False en caso contrario
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Biblioteca other = (Biblioteca) obj;
        return Arrays.deepEquals( a1: this.publicaciones, a2: other.publicaciones);
    }

    /**
     * Método toString() de la clase
     * @return Información sobre las publicaciones de la biblioteca en forma de cadena
     */
    @Override
    public String toString() {

        String cadena = "";

        for (int i = 0; i < this.publicaciones.length; i++) {

            if (this.publicaciones[i] != null && i == 0) {

                cadena += this.publicaciones[i].toString();

            } else {

                cadena += "\n" + this.publicaciones[i].toString();

            }

        }

        return cadena;
    }
```

```java

    /**
     * Método que devuelve las publicaciones de la biblioteca
     * @return Publicaciones de la biblioteca
     */
    public Publicacion[] getPublicaciones() {
        return publicaciones;
    }

    /**
     * Método para establecer nuevas publicaciones a la biblioteca
     * @param publicaciones Nueva publicación
     */
    public void setPublicaciones(Publicacion[] publicaciones) {
        this.publicaciones = publicaciones;
    }


```

- **En la clase Biblioteca debemos implementar los siguientes métodos**
  - **addPublicacion. Añade la Publicación que se pasa como parámetro al array de publicaciones de la biblioteca.**
  - **numPublicacionesAnterioresA. Recibe un año y devuelve el número de publicaciones que tienen un año anterior al que se le pasa como parámetro.**
  - **mostrarPublicaciones. Muestra por terminal todas las publicaciones que tiene la Biblioteca.**

```java
    /**
     * Método para añadir nuevas publicaciones a la biblioteca
     *
     * @param nuevaPublicacion Publicacion a añadir
     */
    public void addPublicacion(Publicacion nuevaPublicacion) {

        this.publicaciones = Arrays.copyOf( original: publicaciones, this.publicaciones.length + 1);
        this.publicaciones[this.publicaciones.length - 1] = nuevaPublicacion;

    }

    /**
     * Método que cuenta el número de publicaciones anteriores a un año pasado
     * por parámetro
     *
     * @param ano Año hasta el que se desea contar
     * @return Número de publicaciones anteriores al año pasado por parámetro
     */
    public int numPublicacionesAnterioresA(int ano) {

        int numPublicaciones = 0;

        for (int i = 0; i < this.publicaciones.length; i++) {

            if (this.publicaciones != null && this.publicaciones[i].getAnoPublicacion() < ano) {

                numPublicaciones++;

            }

        }

        return numPublicaciones;

    }

    /**
     * Método que muestra por terminal las publicaciones de la biblioteca
     */
    public void mostrarPublicaciones() {

        System.out.println( x: toString());

    }

```

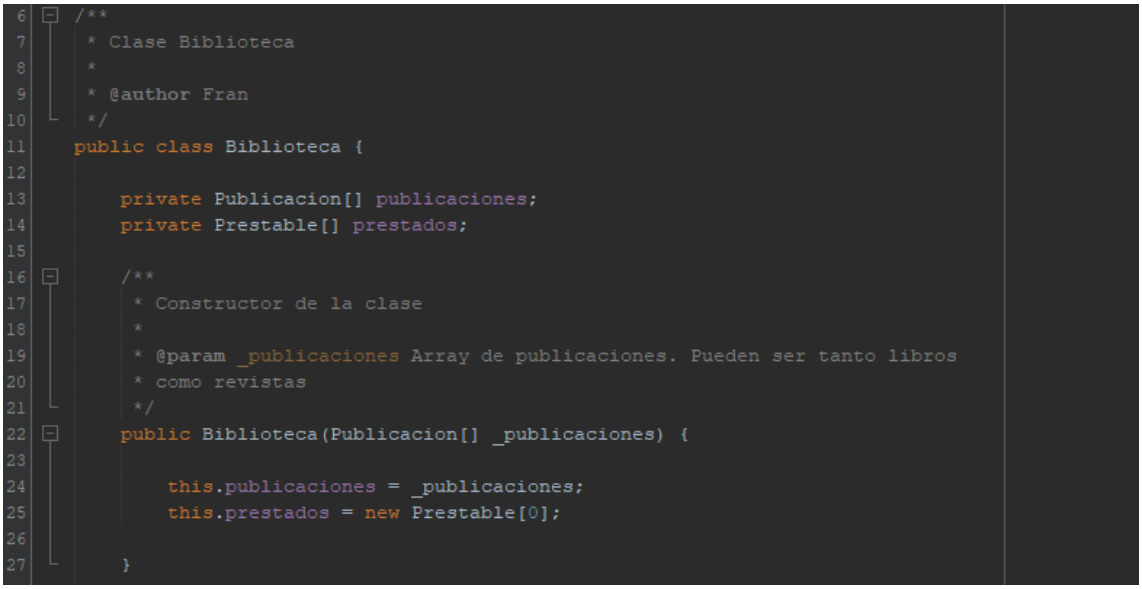
- **Crea una biblioteca con varios objetos de cada tipo y utiliza los métodos creados.**

```java
package com.mycompany.garciacutillasfranciscojose_act09_interfaces_ejercicio3;

import clases.Biblioteca;
import clases.Libro;
import clases.Publicacion;
import clases.Revista;

/**
 * Clase principal
 * @author Fran
 */
public class GarciaCutillasFranciscoJose_Act09_Interfaces_Ejercicio3 {

    public static void main(String[] args) {

        Libro l1 = new Libro (_codigo: 1, _titulo: "El Quijote. Nueva edición", _anoPublicacion: 2016);
        Libro l2 = new Libro (_codigo: 2, _titulo: "Los pilares de la tierra", _anoPublicacion: 2018);

        Revista r1 = new Revista (_codigo: 3, _titulo: "Super Pop", _anoPublicacion: 2020, _numero: 22);
        Revista r2 = new Revista (_codigo: 4, _titulo: "Caza y pesca", _anoPublicacion: 2021, _numero: 247);

        Publicacion[] arrayPublicaciones = {l1};

        Biblioteca bl = new Biblioteca(_publicaciones: arrayPublicaciones);

        bl.addPublicacion(nuevaPublicacion: r1);
        bl.addPublicacion(nuevaPublicacion: r2);
        bl.addPublicacion(nuevaPublicacion: l2);
        bl.mostrarPublicaciones();

        System.out.println(x: "");

        System.out.println("Publicaciones anteriores a 2016: " + bl.numPublicacionesAnterioresA(ano: 2016));
        System.out.println("Publicaciones anteriores a 2020: " + bl.numPublicacionesAnterioresA(ano: 2020));
        System.out.println("Publicaciones anteriores a 2022: " + bl.numPublicacionesAnterioresA(ano: 2022));

    }
}
```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ GarciaCutillasFranciscoJose_Act09_Interfaces_Ejercicio3 ---
Código: 1, Título: El Quijote. Nueva edición, Año de publicación: 2016, Prestado: No
Código: 3, Título: Super Pop, Año de publicación: 2020, Número: 22
Código: 4, Título: Caza y pesca, Año de publicación: 2021, Número: 247
Código: 2, Título: Los pilares de la tierra, Año de publicación: 2018, Prestado: No

Publicaciones anteriores a 2016: 0
Publicaciones anteriores a 2020: 2
Publicaciones anteriores a 2022: 4
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.341 s
Finished at: 2023-04-26T19:03:07+02:00
------------------------------------------------------------------------
```

Ejercicio A04. Biblioteca v2

- **Haz una copia del proyecto anterior (ponle un nombre diferente).**
- **Añade a la biblioteca un nuevo array del tipo de la interfaz (Prestable) que va a contener los elementos que han sido prestados (en la actualidad únicamente podrán ser libros ya que son los únicos elementos que han sido prestados). Cuando se crea la biblioteca, este array estará vacío (esto puede ser un problema si no se gestiona adecuadamente).**



```java
/**
 * Clase Biblioteca
 *
 * @author Fran
 */
public class Biblioteca {

    private Publicacion[] publicaciones;
    private Prestable[] prestados;

    /**
     * Constructor de la clase
     *
     * @param _publicaciones Array de publicaciones. Pueden ser tanto libros
     * como revistas
     */
    public Biblioteca(Publicacion[] _publicaciones) {

        this.publicaciones = _publicaciones;
        this.prestados = new Prestable[0];

    }
```

- **Implementa los siguientes métodos**
  - **prestarElemento. Recibe el código de un Libro. Si este no estaba prestado en la Biblioteca, se cambia su estado al estado de prestado y se añade al array de Prestados.**

```java
    /**
     * Método para prestar un libro
     *
     * @param codigo Código del libro a prestar
     */
    public void prestarElemento(int codigo) {

        //Comprueba que el libro está en la biblioteca
        for (int i = 0; i < this.publicaciones.length; i++) {

            //Comprueba que la publicación coincide con el código pasado por parámetro
            if (this.publicaciones[i].getCodigo() == codigo) {

                //Comprueba que la publicación es un libro
                if (this.publicaciones[i].getClass().getSimpleName().equals( anObject: "Libro")) {

                    Libro libro = (Libro) this.publicaciones[i];//Cast a tipo Libro para poder usar sus métodos

                    //Comprueba que el libro no está prestado
                    if (!libro.GetPrestado()) {

                        libro.prestar(); //Presta el libro

                        //Añádelo al array de libros prestados, redimensionando el mismo
                        this.prestados = Arrays.copyOf( original: this.prestados, this.prestados.length + 1);

                        /*Con el uso del método que cuenta los elementos prestados podemos situarnos en
                          la posición idónea para insertar el siguiente libro prestado*/
                        this.prestados[cuentaPrestados()] = libro;

                        //Si el libro ya está prestado, lo muestra por pantalla
                    } else {

                        System.out.println("El libro con código " + codigo + ", ya está prestado");

                    }

                }

            }

        }

    }
```

  - **cuentaPrestados. Indica el número de elementos que están prestados (debemos tener cuidado si el array tiene nulos).**

```java
    /**
     * Método que cuenta los libros que hay prestados en la biblioteca
     *
     * @return Número de libros prestados
     */
    public int cuentaPrestados() {

        int totalPrestados = 0;

        for (int i = 0; i < this.prestados.length; i++) {

            if (this.prestados[i] != null) {

                totalPrestados++;

            }

        }

        return totalPrestados;

    }
```

- **devolverElemento. Si el código del Libro que se pasa por parámetro está prestado para la Biblioteca, se cambia su estado a no prestado y se elimina del array de Elementos Prestados.**

```
193     /**
194      * Método para devolver un libro
195      *
196      * @param codigo Código del libro
197      */
198     public void devolverElemento(int codigo) {
199 
200         for (int i = 0; i < this.prestados.length; i++) {
201 
202             Libro libro = (Libro) this.prestados[i];
203 
204             if (libro != null && libro.getCodigo() == codigo && libro.GetPrestado()) {
205 
206                 libro.devolver();
207                 this.prestados[i] = null;
208 
209             }
210 
211         }
212 
213     }
```

- **NOTA: Para poder prestar un elemento (y devolverlo), el elemento (que será un Libro) que se pasa como parámetro debe pertenecer a la Biblioteca**

- **Crea una biblioteca con varios objetos de cada tipo y utiliza los métodos creados.**

```java
/**
 * Clase principal
 *
 * @author Fran
 */
public class GarciaCutillasFranciscoJose_Act09_Interfaces_Ejercicio3 {

    public static void main(String[] args) {

        Libro l1 = new Libro( _codigo: 1,  _titulo: "El Quijote. Nueva edición",  _anoPublicacion: 2016);
        Libro l2 = new Libro( _codigo: 2,  _titulo: "Los pilares de la tierra",  _anoPublicacion: 2018);
        Libro l3 = new Libro( _codigo: 5,  _titulo: "Cómo aprender a programar y no morir en el intento",  _anoPublicacion: 2015);
        Libro l4 = new Libro( _codigo: 6,  _titulo: "100 años de soledad",  _anoPublicacion: 2010);

        Revista r1 = new Revista( _codigo: 3,  _titulo: "Super Pop",  _anoPublicacion: 2020,  _numero: 22);
        Revista r2 = new Revista( _codigo: 4,  _titulo: "Caza y pesca",  _anoPublicacion: 2021,  _numero: 247);

        Publicacion[] arrayPublicaciones = {l1, l4};

        Biblioteca b1 = new Biblioteca( _publicaciones: arrayPublicaciones);

        b1.addPublicacion( nuevaPublicacion: r1);
        b1.addPublicacion( nuevaPublicacion: r2);
        b1.addPublicacion( nuevaPublicacion: l2);
        b1.addPublicacion( nuevaPublicacion: l3);


        System.out.println("Prestados: " + b1.cuentaPrestados());

        System.out.println( x: "\nPresto los libros con código 2 y 1");
        b1.prestarElemento( codigo: 2);
        b1.prestarElemento( codigo: 1);
        System.out.println("Prestados: " + b1.cuentaPrestados());

        System.out.println( x: "\nIntento volver a prestar el libro con código 2");
        b1.prestarElemento( codigo: 2);
        System.out.println("Prestados: " + b1.cuentaPrestados());

        System.out.println( x: "\nDevuelvo los libros 2 y 1");
        b1.devolverElemento( codigo: 2);
        b1.devolverElemento( codigo: 1);
        System.out.println("Prestados: " + b1.cuentaPrestados());

        System.out.println( x: "\nVuelvo a prestar 1, 5 y 6");
        b1.prestarElemento( codigo: 1);
        b1.prestarElemento( codigo: 5);
        b1.prestarElemento( codigo: 6);
        System.out.println("Prestados: " + b1.cuentaPrestados());

        System.out.println( x: "\nIntento prestar una revista con código 4");
        b1.prestarElemento( codigo: 4);
        System.out.println("Prestados: " + b1.cuentaPrestados());

    }
}

```

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ GarciaCutillasFranciscoJose_Act09_Interfaces_Ejercicio4 ---
Prestados: 0

Presto los libros con código 2 y 1
Prestados: 2

Intento volver a prestar el libro con código 2
El libro con código 2, ya está prestado
Prestados: 2

Devuelvo los libros 2 y 1
Prestados: 0

Vuelvo a prestar 1, 5 y 6
Prestados: 3

Intento prestar una revista con código 4
Prestados: 3
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  0.311 s
Finished at: 2023-04-27T18:00:47+02:00
------------------------------------------------------------------------
```