

PROGRAMACIÓN

ACTIVIDAD EVALUABLE UD2_2



SAMMY CABELO

(+34) 653085610

- 1) Desarrollar un programa (comenta y documenta el código) que modele una cuenta bancaria que tiene los siguientes atributos, que deben ser de acceso protegido:

- ✓ Saldo, de tipo float.
- ✓ Número de consignaciones con valor inicial cero, de tipo int.
- ✓ Número de retiros con valor inicial cero, de tipo int.
- ✓ Tasa anual (porcentaje), de tipo float.
- ✓ Comisión mensual con valor inicial cero, de tipo float.

La clase Cuenta tiene un constructor que inicializa los atributos saldo y tasa anual con valores pasados como parámetros. La clase Cuenta tiene los siguientes métodos:

- ✓ Consignar una cantidad de dinero en la cuenta actualizando su saldo.
- ✓ Retirar una cantidad de dinero en la cuenta actualizando su saldo. El valor a retirar no debe superar el saldo.
- ✓ Calcular el interés mensual de la cuenta y actualiza el saldo correspondiente.

```
public void calcularInterés()
{
    float tasaMensual = tasaAnual / 12;
    float interesMensual = saldo * tasaMensual;
    saldo += interesMensual;
}

✓ Extracto mensual: actualiza el saldo restándole la comisión mensual y calculando el interés mensual correspondiente (invoca el método anterior).
✓ Imprimir: muestra en pantalla los valores de los atributos.
✓ CuentaMayor: método estático que recibe como argumentos dos cuentas y compara el saldo de ambas. Devuelve la cuenta de mayor saldo.
```

Modelado de Cuentas Bancarias

Para comenzar, he creado la clase "Cuenta", utilizando la palabra clave "class" en Java. Esta clase es la base del modelo y contiene atributos protegidos para garantizar que sean accesibles únicamente desde la misma clase y sus subclases. Los atributos que he incluido son:

- - "saldo", de tipo "float", que representa el dinero disponible en la cuenta.
- "número de consignaciones", de tipo "int", con un valor inicial de cero.
- "número de retiros", de tipo "int", también inicializado en cero.
- "tasa anual", de tipo "float", para gestionar el interés anual.
- "comisión mensual", de tipo "float", que también comienza en cero.

Para inicializar estos atributos, he definido un constructor que recibe como parámetros el saldo y la tasa anual. Dentro del constructor, asigno los valores pasados a los atributos correspondientes. Esto permite crear instancias de "Cuenta" con diferentes valores iniciales.

Posteriormente, he implementado métodos que realizan las operaciones principales sobre una cuenta bancaria:

- "consignar": Este método recibe una cantidad como parámetro y la añade al saldo. Para hacerlo, simplemente sumo la cantidad al atributo "saldo" y aumento en uno el contador de consignaciones.
 - "retirar": Aquí, primero verifico que la cantidad a retirar no exceda el saldo disponible. Si es válido, reduzco el saldo en esa cantidad y aumento en uno el contador de retiros.
 - "calcularInterés": Para calcular el interés mensual, divido la tasa anual entre 12 y luego multiplico el saldo por esta tasa mensual. El resultado se suma al saldo actual.
 - "extractoMensual": Este método combina la aplicación de la comisión mensual y el cálculo del interés mensual. Primero, reduzco el saldo por el valor de la comisión mensual y luego invoco el método "calcularInterés" para actualizar el saldo con el interés correspondiente.
 - "imprimir": Finalmente, este método imprime en pantalla los valores actuales de todos los atributos, lo que facilita el seguimiento de las operaciones.
- Usando un operador ternario, mencionado en la Unidad Didáctica 1, he creado la manera de ver cuál de las dos cuentas tiene más dinero depositado con el método "cuentaMayor".

```
1 package cuentasbancarias;
2
3 /**
4 * Representa una cuenta bancaria genérica con operaciones básicas.
5 * Contiene atributos y métodos comunes para diferentes tipos de cuentas.
6 */
7 public class Cuenta {
8     // Atributos protegidos
9     protected float saldo;
10    protected int numConsignaciones;
11    protected int numRetiros;
12    protected float tasaAnual;
13    protected float comisionMensual;
14
15    /**
16     * Constructor de la cuenta bancaria. Valores inicializados según el enunciado de la actividad.
17     */
18    public Cuenta(float saldoInicial, float tasaAnual) {
19        this.saldo = saldoInicial;
20        this.tasaAnual = tasaAnual;
21        this.numConsignaciones = 0;
22        this.numRetiros = 0;
23        this.comisionMensual = 0;
24    }
25
26    /**
27     * Método para meter dinero en la cuenta. Aumenta el número de consignaciones.
28     */
29    public void consignar(float cantidad) {
30        saldo += cantidad;
31        numConsignaciones++;
32    }
33
34    /**
35     * Método para retirar dinero de la cuenta. Aumenta el número de retiros.
36     */
37    public boolean retirar(float cantidad) {
38        if (cantidad <= saldo) {
39            saldo -= cantidad;
40            numRetiros++;
41            return true;
42        }
43        return false;
44    }
45
46    /**
47     * Calcula y añade el interés mensual al saldo.
48     */
49    public void calcularInterés() {
50        float tasaMensual = tasaAnual / 12;
51        float interesMensual = saldo * tasaMensual;
52        saldo += interesMensual;
53    }
54
55    /**
56     * Genera el extracto mensual de la cuenta.
57     */
58    public void extractoMensual() {
59        saldo -= comisionMensual;
60        calcularInterés();
61    }
62
63    /**
64     * Imprime la información de la cuenta.
65     */
66    public void imprimir() {
67        System.out.println("Saldo: " + saldo);
68        System.out.println("Número de consignaciones: " + numConsignaciones);
69        System.out.println("Número de retiros: " + numRetiros);
70    }
71
72    public static Cuenta cuentaMayor(Cuenta cuenta1, Cuenta cuenta2) {
73        return cuenta1.saldo > cuenta2.saldo ? cuenta1 : cuenta2;
74    }
75}
76 }
```

La clase Cuenta tiene dos clases hijas:

Cuenta de ahorros: posee un atributo para determinar si la cuenta de ahorros está activa (tipo boolean). Si el saldo es menor a \$10 000, la cuenta está inactiva, en caso contrario se considera activa. Los siguientes métodos se redefinen:

- ✓ *Consignar: se puede consignar dinero si la cuenta está activa. Debe invocar al método heredado.*
- ✓ *Retirar: es posible retirar dinero si la cuenta está activa. Debe invocar al método heredado.*
- ✓ *Extracto mensual: si el número de retiros es mayor que 4, por cada retiro adicional, se cobra \$1000 como comisión mensual. Al generar el extracto, se determina si la cuenta está activa o no con el saldo.*
- ✓ *Un nuevo método imprimir que muestra en pantalla el saldo de la cuenta, la comisión mensual y el número de transacciones realizadas (suma de cantidad de consignaciones y retiros).*

Cuenta corriente: posee un atributo de sobregiro, el cual se inicializa en cero. Se redefinen los siguientes métodos:

- ✓ *Retirar: se retira dinero de la cuenta actualizando su saldo. Se puede retirar dinero superior al saldo.*
- ✓ *Consignar: invoca al método heredado. También debe mostrar por pantalla el saldo actual.*
- ✓ *Extracto mensual: invoca al método heredado. También debe mostrar por pantalla el saldo actual.*
- ✓ *Un nuevo método imprimir que muestra en pantalla el saldo de la cuenta, la comisión mensual y el número de transacciones realizadas (suma de cantidad de consignaciones y retiros).*

Realizar un método main que implemente un objeto Cuenta de ahorros y llame a los métodos correspondientes dentro de un fichero denominado PruebaAhorro.java.

Modelado de las clases hijas

He creado dos clases hijas: "CuentaDeAhorros" y "CuentaCorriente", utilizando la palabra clave "extends" para indicar que heredan de la clase "Cuenta".

En "CuentaDeAhorros", he añadido un atributo booleano llamado "activa", que determina si la cuenta está activa o no. He redefinido varios métodos heredados para incluir lógica adicional específica de este tipo de cuenta:

- "consignar": Antes de añadir dinero, verifico si la cuenta está activa. Si no lo está, no permito la operación.
- "retirar": Similar al método anterior, verifico si la cuenta está activa antes de permitir el retiro. Además, utilizo la lógica heredada para actualizar el saldo.
- "extractoMensual": Aquí, añado una condición para aplicar una comisión adicional si el número de retiros supera los cuatro. También recalcúlo el estado de la cuenta ("activa" o "inactiva") basado en el saldo.

En "CuentaCorriente", he introducido el atributo "sobregiro", que permite realizar retiros incluso cuando el saldo es insuficiente. Esto se gestiona restando la cantidad excedente del atributo "sobregiro". Los métodos como "retirar" y "consignar" están adaptados para manejar estas situaciones.

De la misma manera, he usado los sobregiros para gestionar los nuevos métodos “imprimir” de cada tipo de cuenta.

```
1 package cuentasbancarias;
2
3 /**
4  * Representa una cuenta corriente con características específicas.
5 */
6 public class CuentaCorriente extends Cuenta {
7     private float sobregiro;
8
9     /**
10      * Constructor de CuentaCorriente.
11      *
12      */
13     public CuentaCorriente(float saldoInicial, float tasaAnual) {
14         super(saldoInicial, tasaAnual);
15         this.sobregiro = 0;
16     }
17
18     /**
19      * Sobrescribe el método retirar para permitir sobregiro.
20      */
21     @Override
22     public boolean retirar(float cantidad) {
23         if (cantidad <= saldo) {
24             return super.retirar(cantidad);
25         } else {
26             sobregiro += cantidad - saldo;
27             saldo = 0;
28             numRetiros++;
29             return true;
30         }
31     }
32
33     /**
34      * Sobrescribe el método consignar para mostrar saldo.
35      */
36     @Override
37     public void consignar(float cantidad) {
38         super.consignar(cantidad);
39         System.out.println("Saldo actual: " + saldo);
40     }
41
42     /**
43      * Sobrescribe el método extracto mensual.
44      */
45     @Override
46     public void extractoMensual() {
47         super.extractoMensual();
48         System.out.println("Saldo actual: " + saldo);
49     }
50
51     /**
52      * Sobrescribe el método imprimir para mostrar información específica.
53      */
54     @Override
55     public void imprimir() {
56         System.out.println("Saldo: " + saldo);
57         System.out.println("Comisión mensual: " + comisionMensual);
58         System.out.println("Total transacciones: " + (numConsignaciones + numRetiros));
59         System.out.println("Sobregiro: " + sobregiro);
60     }
61 }
```

```
1 package cuentasbancarias;
2
3 /**
4 * Representa una cuenta de ahorros con características específicas.
5 */
6 public class CuentaAhorros extends Cuenta {
7     private boolean cuentaActiva;
8
9     /**
10      * Constructor de CuentaAhorros.
11      *
12      */
13     public CuentaAhorros(float saldoInicial, float tasaAnual) {
14         super(saldoInicial, tasaAnual);
15         this.cuentaActiva = saldoInicial >= 10000;
16     }
17
18     /**
19      * Sobrescribe el método consignar para verificar estado de la cuenta.
20      */
21     @Override
22     public void consignar(float cantidad) {
23         if (cuentaActiva) {
24             super.consignar(cantidad);
25             cuentaActiva = saldo >= 10000;
26         }
27     }
28
29     /**
30      * Sobrescribe el método retirar para verificar estado de la cuenta.
31      */
32     @Override
33     public boolean retirar(float cantidad) {
34         if (cuentaActiva) {
35             boolean retiroExitoso = super.retirar(cantidad);
36             cuentaActiva = saldo >= 10000;
37             return retiroExitoso;
38         }
39         return false;
40     }
41
42     /**
43      * Sobrescribe el extracto mensual para aplicar comisión adicional.
44      */
45     @Override
46     public void extractoMensual() {
47         if (numRetiros > 4) {
48             comisionMensual += (numRetiros - 4) * 1000;
49         }
50         super.extractoMensual();
51         cuentaActiva = saldo >= 10000;
52     }
53
54     /**
55      * Sobrescribe el método imprimir para mostrar información específica.
56      */
57     @Override
58     public void imprimir() {
59         System.out.println("Saldo: " + saldo);
60         System.out.println("Comisión mensual: " + comisionMensual);
61         System.out.println("Total transacciones: " + (numConsignaciones + numRetiros));
62         System.out.println("Estado de cuenta: " + (cuentaActiva ? "Activa" : "Inactiva"));
63     }
64 }
```

Creación de Métodos Main

He creado dos programas principales denominados "PruebaAhorro.java" y "PruebaCorriente.java". En estos programas, instancio objetos de las clases hijas ("CuentaDeAhorros" y "CuentaCorriente") y pruebo los métodos definidos. En cada caso,

verifico que las operaciones funcionen correctamente y que los resultados se impriman adecuadamente en la terminal.

```
● ● ●
1 package cuentasbancarias;
2
3 /**
4  * Clase de prueba para demostrar el funcionamiento de CuentaAhorros.
5 */
6 public class PruebaAhorro {
7     public static void main(String[] args) {
8         // Crear una cuenta de ahorros con saldo inicial de 15000 y tasa anual del 5%
9         CuentaAhorros miCuenta = new CuentaAhorros(15000, 0.05f);
10
11     System.out.println("--- Estado Inicial ---");
12     miCuenta.imprimir();
13
14     // Realizar algunas consignaciones y retiros
15     miCuenta.consignar(5000);
16     miCuenta.retirar(2000);
17     miCuenta.retirar(2000);
18     miCuenta.retirar(2000);
19     miCuenta.retirar(2000);
20     miCuenta.retirar(2000); // Quinto retiro
21
22     System.out.println("\n--- Después de Transacciones ---");
23     miCuenta.imprimir();
24
25     // Generar extracto mensual
26     miCuenta.extractoMensual();
27
28     System.out.println("\n--- Después de Extracto Mensual ---");
29     miCuenta.imprimir();
30 }
31 }
```

Salida por terminal:

```
● Ahorro
--- Estado Inicial ---
Saldo: 15000.0
Comisión mensual: 0.0
Total transacciones: 0
Estado de cuenta: Activa

--- Después de Transacciones ---
Saldo: 10000.0
Comisión mensual: 0.0
Total transacciones: 6
Estado de cuenta: Activa

--- Después de Extracto Mensual ---
Saldo: 9037.5
Comisión mensual: 1000.0
Total transacciones: 6
Estado de cuenta: Inactiva
```



The screenshot shows a Java code editor with the following code:

```
1 package cuentasbancarias;
2
3 /**
4  * Clase de prueba para demostrar el funcionamiento de CuentaCorriente.
5 */
6 public class PruebaCorriente {
7     public static void main(String[] args) {
8         // Crear una cuenta corriente con saldo inicial de 10000 y tasa anual del 6%
9         CuentaCorriente miCuenta = new CuentaCorriente(10000, 0.06f);
10
11     System.out.println("--- Estado Inicial ---");
12     miCuenta.imprimir();
13
14     // Realizar algunas consignaciones y retiros
15     miCuenta.consignar(5000);
16     miCuenta.retirar(12000); // Probando sobregiro
17
18     System.out.println("\n--- Después de Transacciones ---");
19     miCuenta.imprimir();
20
21     // Generar extracto mensual
22     miCuenta.extractoMensual();
23
24     System.out.println("\n--- Después de Extracto Mensual ---");
25     miCuenta.imprimir();
26 }
27 }
```

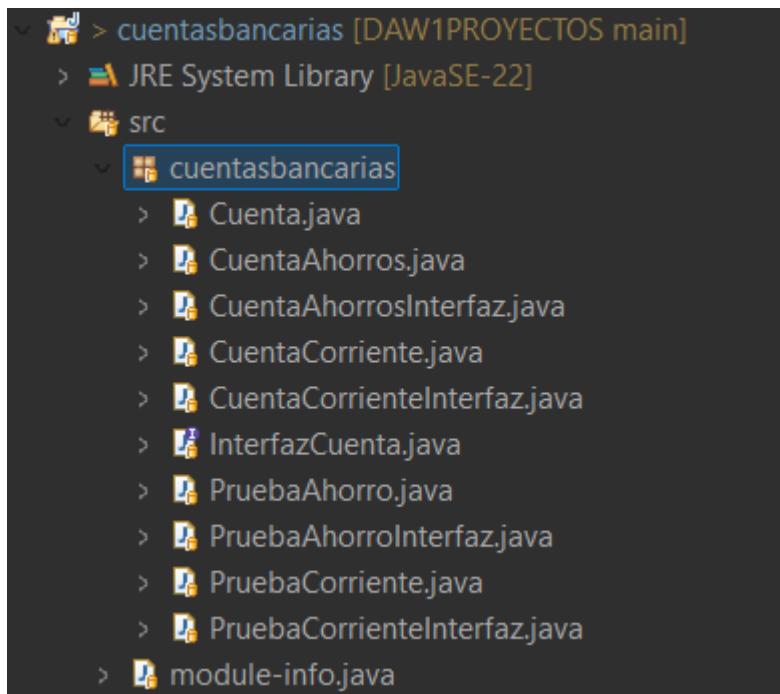
Salida por terminal:

```
--- Estado Inicial ---
Saldo: 10000.0
Comisión mensual: 0.0
Total transacciones: 0
Sobregiro: 0.0
Saldo actual: 15000.0

--- Después de Transacciones ---
Saldo: 3000.0
Comisión mensual: 0.0
Total transacciones: 2
Sobregiro: 0.0
Saldo actual: 3015.0

--- Después de Extracto Mensual ---
Saldo: 3015.0
Comisión mensual: 0.0
Total transacciones: 2
Sobregiro: 0.0
```

Se han creado 5 ficheros de cada. Un fichero para cada clase. Todas dentro del paquete cuentasbancarias. También se ve aquí las interfaces que abordaré más adelante.



- 2) Modifica el programa anterior (comenta y documenta el código) para que los ficheros hagan lo mismo pero la clase Cuenta sea una interface.**

Para esta parte, he transformado la clase "Cuenta" en una interfaz utilizando la palabra clave "interface". Esto implica declarar únicamente los métodos que las clases hijas deben implementar, sin incluir su lógica.

Por ejemplo, los métodos "consignar", "retirar", "calcularInterés", entre otros, se declaran en la interfaz, y luego cada clase que implementa esta interfaz (como "CuentaDeAhorros" y "CuentaCorriente") debe definir la lógica correspondiente en sus propios métodos.

```
1 package cuentasbancarias;
2
3 /**
4  * Interfaz que define el contrato para cuentas bancarias.
5  * Especifica los métodos que deben implementar todas las cuentas.
6  */
7 public interface InterfazCuenta {
8     /**
9      * Consigna una cantidad de dinero en la cuenta.
10     *
11     */
12     void consignar(float cantidad);
13
14     /**
15      * Retira una cantidad de dinero de la cuenta.
16     *
17     */
18     boolean retirar(float cantidad);
19
20     /**
21      * Calcula el interés mensual de la cuenta.
22     *
23     */
24     void calcularInterés();
25
26     /**
27      * Genera el extracto mensual de la cuenta.
28     *
29     */
30     void extractoMensual();
31
32     /**
33      * Imprime la información de la cuenta.
34     *
35     */
36     /**
37      * Método estático para comparar dos cuentas y devolver la de mayor saldo.
38     *
39     */
40     static InterfazCuenta cuentaMayor(InterfazCuenta cuenta1, InterfazCuenta cuenta2) {
41         // Asumimos que las cuentas tienen un método para obtener saldo
42         return cuenta1.getSaldo() > cuenta2.getSaldo() ? cuenta1 : cuenta2;
43     }
44
45     /**
46      * Obtiene el saldo actual de la cuenta.
47     *
48     */
49     float getSaldo();
50 }
```

```
1 package cuentasbanarias;
2
3 /**
4  * Implementación de cuenta de ahorros que sigue el contrato de la interface InterfazCuenta.
5 */
6 public class CuentaAhorrosInterfaz implements InterfazCuenta {
7     // Atributos
8     private float saldo;
9     private int numConsignaciones;
10    private int numRetiros;
11    private float tasaAnual;
12    private float comisionMensual;
13    private boolean cuentaActiva;
14
15    /**
16     * Constructor de CuentaAhorrosInterfaz.
17     *
18     */
19    public CuentaAhorrosInterfaz(float saldoInicial, float tasaAnual) {
20        this.saldo = saldoInicial;
21        this.tasaAnual = tasaAnual;
22        this.numConsignaciones = 0;
23        this.numRetiros = 0;
24        this.comisionMensual = 0;
25        this.cuentaActiva = saldoInicial >= 10000;
26    }
27
28    @Override
29    public void consignar(float cantidad) {
30        if (cuentaActiva) {
31            saldo += cantidad;
32            numConsignaciones++;
33            cuentaActiva = saldo >= 10000;
34        }
35    }
36
37    @Override
38    public boolean retirar(float cantidad) {
39        if (cuentaActiva && cantidad <= saldo) {
40            saldo -= cantidad;
41            numRetiros++;
42            cuentaActiva = saldo >= 10000;
43            return true;
44        }
45        return false;
46    }
47
48    @Override
49    public void calcularInterés() {
50        float tasaMensual = tasaAnual / 12;
51        float interesMensual = saldo * tasaMensual;
52        saldo += interesMensual;
53    }
54
55    @Override
56    public void extractoMensual() {
57        if (numRetiros > 4) {
58            comisionMensual += (numRetiros - 4) * 1000;
59        }
60        saldo -= comisionMensual;
61        calcularInterés();
62        cuentaActiva = saldo >= 10000;
63    }
64
65    @Override
66    public void imprimir() {
67        System.out.println("Saldo: " + saldo);
68        System.out.println("Comisión mensual: " + comisionMensual);
69        System.out.println("Total transacciones: " + (numConsignaciones + numRetiros));
70        System.out.println("Estado de cuenta: " + (cuentaActiva ? "Activa" : "Inactiva"));
71    }
72
73    @Override
74    public float getSaldo() {
75        return saldo;
76    }
77 }
```

```
1 package cuentasbancarias;
2
3 /**
4  * Implementación de cuenta corriente que sigue el contrato de la interface InterfazCuenta.
5 */
6 public class CuentaCorrienteInterfaz implements InterfazCuenta {
7     // Atributos
8     private float saldo;
9     private int numConsignaciones;
10    private int numRetiros;
11    private float tasaAnual;
12    private float comisionMensual;
13    private float sobregiro;
14
15    /**
16     * Constructor de CuentaCorrienteInterfaz.
17     */
18    public CuentaCorrienteInterfaz(float saldoInicial, float tasaAnual) {
19        this.saldo = saldoInicial;
20        this.tasaAnual = tasaAnual;
21        this.numConsignaciones = 0;
22        this.numRetiros = 0;
23        this.comisionMensual = 0;
24        this.sobregiro = 0;
25    }
26
27    @Override
28    public void consignar(float cantidad) {
29        saldo += cantidad;
30        numConsignaciones++;
31        System.out.println("Saldo actual: " + saldo);
32    }
33
34    @Override
35    public boolean retirar(float cantidad) {
36        if (cantidad <= saldo) {
37            saldo -= cantidad;
38            numRetiros++;
39            return true;
40        } else {
41            sobregiro += cantidad - saldo;
42            saldo = 0;
43            numRetiros++;
44            return true;
45        }
46    }
47
48    @Override
49    public void calcularInterés() {
50        float tasaMensual = tasaAnual / 12;
51        float interesMensual = saldo * tasaMensual;
52        saldo += interesMensual;
53    }
54
55    @Override
56    public void extractoMensual() {
57        saldo -= comisionMensual;
58        calcularInterés();
59        System.out.println("Saldo actual: " + saldo);
60    }
61
62    @Override
63    public void imprimir() {
64        System.out.println("Saldo: " + saldo);
65        System.out.println("Comisión mensual: " + comisionMensual);
66        System.out.println("Total transacciones: " + (numConsignaciones + numRetiros));
67        System.out.println("Sobregiro: " + sobregiro);
68    }
69
70    @Override
71    public float getSaldo() {
72        return saldo;
73    }
74 }
```

```
1 package cuentasbancarias;
2
3 /**
4  * Clase de prueba para demostrar el funcionamiento de CuentaAhorrosInterfaz.
5 */
6 public class PruebaAhorroInterfaz {
7     public static void main(String[] args) {
8         // Crear una cuenta de ahorros con saldo inicial de 15000 y tasa anual del 5%
9         InterfazCuenta miCuenta = new CuentaAhorrosInterfaz(15000, 0.05f);
10
11     System.out.println("--- Estado Inicial ---");
12     miCuenta.imprimir();
13
14     // Realizar algunas consignaciones y retiros
15     miCuenta.consignar(5000);
16     miCuenta.retirar(2000);
17     miCuenta.retirar(2000);
18     miCuenta.retirar(2000);
19     miCuenta.retirar(2000);
20     miCuenta.retirar(2000); // Quinto retiro
21
22     System.out.println("\n--- Después de Transacciones ---");
23     miCuenta.imprimir();
24
25     // Generar extracto mensual
26     miCuenta.extractoMensual();
27
28     System.out.println("\n--- Después de Extracto Mensual ---");
29     miCuenta.imprimir();
30 }
31 }
```

Salida por terminal:

```
--- Estado Inicial ---
Saldo: 15000.0
Comisión mensual: 0.0
Total transacciones: 0
Estado de cuenta: Activa

--- Después de Transacciones ---
Saldo: 10000.0
Comisión mensual: 0.0
Total transacciones: 6
Estado de cuenta: Activa

--- Después de Extracto Mensual ---
Saldo: 9037.5
Comisión mensual: 1000.0
Total transacciones: 6
Estado de cuenta: Inactiva
```

```
● ● ●

1 package cuentasbancarias;
2
3 /**
4  * Clase de prueba para demostrar el funcionamiento de CuentaCorrienteInterfaz.
5 */
6 public class PruebaCorrienteInterfaz {
7     public static void main(String[] args) {
8         // Crear una cuenta corriente con saldo inicial de 10000 y tasa anual del 6%
9         InterfazCuenta miCuenta = new CuentaCorrienteInterfaz(10000, 0.06f);
10
11     System.out.println("--- Estado Inicial ---");
12     miCuenta.imprimir();
13
14     // Realizar algunas consignaciones y retiros
15     miCuenta.consignar(5000);
16     miCuenta.retirar(12000); // Probando sobregiro
17
18     System.out.println("\n--- Después de Transacciones ---");
19     miCuenta.imprimir();
20
21     // Generar extracto mensual
22     miCuenta.extractoMensual();
23
24     System.out.println("\n--- Después de Extracto Mensual ---");
25     miCuenta.imprimir();
26 }
27 }
```

Salida por terminal:

```
● CorrienteInterfaz
--- Estado Inicial ---
Saldo: 10000.0
Comisión mensual: 0.0
Total transacciones: 0
Sobregiro: 0.0
Saldo actual: 15000.0

--- Después de Transacciones ---
Saldo: 3000.0
Comisión mensual: 0.0
Total transacciones: 2
Sobregiro: 0.0
Saldo actual: 3015.0

--- Después de Extracto Mensual ---
Saldo: 3015.0
Comisión mensual: 0.0
Total transacciones: 2
Sobregiro: 0.0
```

3) Se necesita desarrollar una aplicación sobre la jerarquía de animales siguientes:

Animal es la clase raíz con los siguientes atributos: hábitat o entorno, comida, sonidos y nombre científico (son todos de tipo String). La clase Animal tiene los siguientes métodos abstractos:

Public abstract String getSonido()

Public abstract String getNombreCientífico()

Public abstract String getEntorno()

Public abstract String getComida()

Los felinos y cánidos son subclases de Animal.

- Los perros son cánidos, su sonido es el ladrido, su comida es carnívora, su entorno es doméstico y su nombre científico es *Canis lupus familiaris*.
- Los lobos son cánidos, su sonido es el aullido, su comida es carnívora, su entorno es el bosque y su nombre científico es *Canis lupus*.
- Los leones son felinos, su sonido es el rugido, su comida es carnívora, su entorno es la pradera y su nombre científico es *Panthera leo*.
- Los gatos son felinos, su sonido es el maullido, su comida son los ratones, su entorno es doméstico y su nombre científico es *Felis silvestris catus*.

Modelado de Animales

En este ejercicio, he diseñado una jerarquía de clases que representa diferentes tipos de animales. Comencé creando la clase base "Animal", declarando los atributos comunes a todos los animales:

- - "hábitat", que representa el entorno donde vive el animal.
- "comida", que indica qué consume el animal.
- "sonidos", para almacenar el sonido característico del animal.
- "nombre científico", que identifica formalmente al animal.

La clase "Animal" también incluye métodos abstractos como "getSonido", "getNombreCientífico", "getEntorno" y "getComida". Estos métodos se declaran como abstractos utilizando la palabra clave "abstract", obligando a las subclases a proporcionar su propia implementación.

A continuación, he creado las clases "Cánido" y "Felino", que heredan de "Animal". Estas clases actúan como categorías intermedias en la jerarquía, pero no implementan directamente los métodos abstractos. En cambio, las clases concretas como "Perro", "Lobo", "León" y "Gato" son las que definen los métodos abstractos para proporcionar detalles específicos de cada animal.

Por ejemplo, en la clase "Perro", he implementado el método "getSonido" para devolver 'ladrido', el método "getNombreCientífico" para devolver 'Canis lupus familiaris', y así con los demás atributos. Esta lógica se repite con las características únicas de los lobos, leones y gatos.

```
● ● ●  
1 package animales;  
2  
3 public abstract class Animal {  
4     // Atributos  
5     private String habitat;  
6     private String comida;  
7     private String sonido;  
8     private String nombreCientifico;  
9  
10    // Constructor  
11    public Animal(String habitat, String comida, String sonido, String nombreCientifico) {  
12        this.habitat = habitat;  
13        this.comida = comida;  
14        this.sonido = sonido;  
15        this.nombreCientifico = nombreCientifico;  
16    }  
17  
18    // Métodos abstractos  
19    public abstract String getSonido();  
20    public abstract String getNombreCientifico();  
21    public abstract String getEntorno();  
22    public abstract String getComida();  
23 }
```

```
● ● ●  
1 package animales;  
2  
3 public abstract class Canido extends Animal {  
4     public Canido(String habitat, String comida, String sonido, String nombreCientifico) {  
5         super(habitat, comida, sonido, nombreCientifico);  
6     }  
7 }
```

```
● ● ●  
1 package animales;  
2  
3 public abstract class Felino extends Animal {  
4     public Felino(String habitat, String comida, String sonido, String nombreCientifico) {  
5         super(habitat, comida, sonido, nombreCientifico);  
6     }  
7 }
```

```
1 package animales;
2
3 public class Gato extends Felino {
4     public Gato() {
5         super("doméstico", "ratones", "maullido", "Felis silvestris catus");
6     }
7
8     @Override
9     public String getSonido() {
10        return "maullido";
11    }
12
13    @Override
14    public String getNombreCientifico() {
15        return "Felis silvestris catus";
16    }
17
18    @Override
19    public String getEntorno() {
20        return "doméstico";
21    }
22
23    @Override
24    public String getComida() {
25        return "ratones";
26    }
27 }
```

```
● ● ●  
1 package animales;  
2  
3 public class Leon extends Felino {  
4     public Leon() {  
5         super("pradera", "carnívora", "rugido", "Panthera leo");  
6     }  
7  
8     @Override  
9     public String getSonido() {  
10        return "rugido";  
11    }  
12  
13    @Override  
14    public String getNombreCientifico() {  
15        return "Panthera leo";  
16    }  
17  
18    @Override  
19    public String getEntorno() {  
20        return "pradera";  
21    }  
22  
23    @Override  
24    public String getComida() {  
25        return "carnívora";  
26    }  
27}
```

```
1 package animales;
2
3 public class Lobo extends Canido {
4     public Lobo() {
5         super("bosque", "carnívora", "aullido", "Canis lupus");
6     }
7
8     @Override
9     public String getSonido() {
10        return "aullido";
11    }
12
13    @Override
14    public String getNombreCientifico() {
15        return "Canis lupus";
16    }
17
18    @Override
19    public String getEntorno() {
20        return "bosque";
21    }
22
23    @Override
24    public String getComida() {
25        return "carnívora";
26    }
27 }
```

```
● ● ●
```

```
1 package animales;
2
3 public class Perro extends Canido {
4     public Perro() {
5         super("doméstico", "carnívora", "ladrido", "Canis lupus familiaris");
6     }
7
8     @Override
9     public String getSonido() {
10        return "ladrido";
11    }
12
13     @Override
14     public String getNombreCientifico() {
15        return "Canis lupus familiaris";
16    }
17
18     @Override
19     public String getEntorno() {
20        return "doméstico";
21    }
22
23     @Override
24     public String getComida() {
25        return "carnívora";
26    }
27 }
```

```
● ● ●

1 package animales;
2
3 public class Main {
4     public static void main(String[] args) {
5         // Crear instancias de cada animal
6         Perro perro = new Perro();
7         Lobo lobo = new Lobo();
8         Leon leon = new Leon();
9         Gato gato = new Gato();
10
11        // Probar métodos de Perro
12        System.out.println("Perro:");
13        System.out.println("Sonido: " + perro.getSonido());
14        System.out.println("Nombre Científico: " + perro.getNombreCientifico());
15        System.out.println("Entorno: " + perro.getEntorno());
16        System.out.println("Comida: " + perro.getComida());
17
18        // Probar métodos de Lobo
19        System.out.println("\nLobo:");
20        System.out.println("Sonido: " + lobo.getSonido());
21        System.out.println("Nombre Científico: " + lobo.getNombreCientifico());
22        System.out.println("Entorno: " + lobo.getEntorno());
23        System.out.println("Comida: " + lobo.getComida());
24
25        // Probar métodos de León
26        System.out.println("\nLeón:");
27        System.out.println("Sonido: " + leon.getSonido());
28        System.out.println("Nombre Científico: " + leon.getNombreCientifico());
29        System.out.println("Entorno: " + leon.getEntorno());
30        System.out.println("Comida: " + leon.getComida());
31
32        // Probar métodos de Gato
33        System.out.println("\nGato:");
34        System.out.println("Sonido: " + gato.getSonido());
35        System.out.println("Nombre Científico: " + gato.getNombreCientifico());
36        System.out.println("Entorno: " + gato.getEntorno());
37        System.out.println("Comida: " + gato.getComida());
38    }
39 }
```

Salida por terminal:

● Perro:

Sonido: ladrido
Nombre Científico: Canis lupus familiaris
Entorno: doméstico
Comida: carnívora

Lobo:

Sonido: aullido
Nombre Científico: Canis lupus
Entorno: bosque
Comida: carnívora

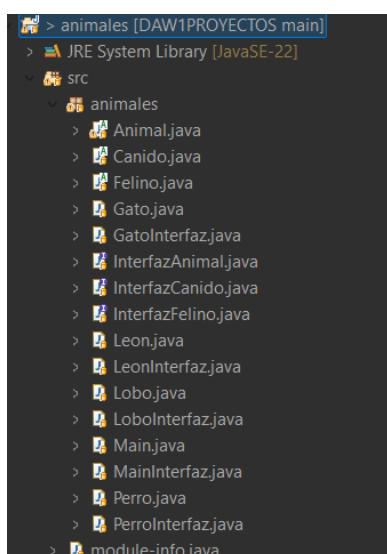
León:

Sonido: rugido
Nombre Científico: Panthera leo
Entorno: pradera
Comida: carnívora

Gato:

Sonido: maullido
Nombre Científico: Felis silvestris catus
Entorno: doméstico
Comida: ratones

Crea la jerarquía de clases en Java. Utiliza un fichero para cada clase y todas las clases deben estar dentro del paquete animales. Usa una clase para el main y prueba todos los métodos. De la misma manera, también vemos las interfaces que se abordarán en el próximo ejercicio.



4) Haz el ejercicio anterior pero con interfaces (comenta y documenta el código).

Implementación de Animales con Interfaces

En esta parte del ejercicio, he transformado la jerarquía basada en clases en un modelo basado en interfaces. Para ello, he convertido la clase `Animal` en una interfaz utilizando la palabra clave `interface`. Dentro de esta interfaz, he declarado métodos como `getSonido`, `getNombreCientífico`, `getEntorno` y `getComida`, sin proporcionar ninguna implementación.

Cada clase concreta como `Perro`, `Lobo`, `León` y `Gato` ahora implementa la interfaz `Animal`. Esto significa que he definido los métodos abstractos directamente en estas clases. Por ejemplo, en la clase `Perro`, he escrito la implementación del método `getSonido` para devolver 'ladrido', y he hecho lo mismo con los demás métodos de la interfaz.

Este enfoque con interfaces promueve una mayor flexibilidad y permite que las clases concreten la lógica sin estar limitadas por una jerarquía rígida. Además, facilita la extensión futura, ya que puedo añadir nuevas clases que implementen la interfaz sin modificar las existentes.



```
1 package animales;
2
3 /**
4  * Interfaz que define los métodos básicos para todos los animales.
5  * Cada animal debe implementar estos métodos para proporcionar
6  * información específica sobre su naturaleza.
7 */
8 public interface InterfazAnimal {
9     /**
10      * Obtiene el sonido característico del animal.
11      */
12     String getSonido();
13
14     /**
15      * Obtiene el nombre científico del animal.
16      */
17     String getNombreCientifico();
18
19     /**
20      * Obtiene el entorno o hábitat natural del animal.
21      */
22     String getEntorno();
23
24     /**
25      * Obtiene el tipo de alimentación del animal.
26      */
27     String getComida();
28 }
```

```
1 package animales;
2
3 /**
4  * Interfaz que representa a los animales de la familia de los felinos.
5  * Extiende la interfaz base de Animal para añadir comportamientos específicos.
6  */
7 public interface InterfazFelino extends InterfazAnimal {
8     /**
9      * Método específico para felinos que podría añadir
10     * comportamientos o características únicas.
11     */
12     default String getCaracteristicaFelino() {
13         return "Mamífero de la familia de los felinos";
14     }
15 }
```

```
1 package animales;
2
3 /**
4  * Interfaz que representa a los animales de la familia de los cánidos.
5  * Extiende la interfaz base de Animal para añadir comportamientos específicos.
6  */
7 public interface InterfazCanido extends InterfazAnimal {
8     /**
9      * Método específico para cánidos que podría añadir
10     * comportamientos o características únicas.
11     */
12     default String getCaracteristicaCanido() {
13         return "Mamífero de la familia de los cánidos";
14     }
15 }
```

```
1 package animales;
2
3 /**
4  * Implementación de un Gato utilizando la interfaz de Felino.
5  * Proporciona detalles específicos sobre el gato doméstico.
6 */
7 public class GatoInterfaz implements InterfazFelino {
8     // Constantes para evitar repetición de valores
9     private static final String HABITAT = "doméstico";
10    private static final String COMIDA = "ratones";
11    private static final String SONIDO = "maullido";
12    private static final String NOMBRE_CIENTIFICO = "Felis silvestris catus";
13
14    @Override
15    public String getSonido() {
16        return SONIDO;
17    }
18
19    @Override
20    public String getNombreCientifico() {
21        return NOMBRE_CIENTIFICO;
22    }
23
24    @Override
25    public String getEntorno() {
26        return HABITAT;
27    }
28
29    @Override
30    public String getComida() {
31        return COMIDA;
32    }
33
34    /**
35     * Método adicional específico de los felinos.
36     */
37    @Override
38    public String getCaracteristicaFelino() {
39        return "Gato doméstico, compañero de hogar";
40    }
41 }
```

```
1 package animales;
2
3 /**
4  * Implementación de un León utilizando la interfaz de Felino.
5  * Proporciona detalles específicos sobre el león en su hábitat natural.
6 */
7 public class LeonInterfaz implements InterfazFelino {
8     // Constantes para evitar repetición de valores
9     private static final String HABITAT = "pradera";
10    private static final String COMIDA = "carnívora";
11    private static final String SONIDO = "rugido";
12    private static final String NOMBRE_CIENTIFICO = "Panthera leo";
13
14    @Override
15    public String getSonido() {
16        return SONIDO;
17    }
18
19    @Override
20    public String getNombreCientifico() {
21        return NOMBRE_CIENTIFICO;
22    }
23
24    @Override
25    public String getEntorno() {
26        return HABITAT;
27    }
28
29    @Override
30    public String getComida() {
31        return COMIDA;
32    }
33
34    /**
35     * Método adicional específico de los felinos.
36     */
37    @Override
38    public String getCaracteristicaFelino() {
39        return "León, rey de la pradera africana";
40    }
41 }
```

```
1 package animales;
2
3 /**
4  * Implementación de un Lobo utilizando la interfaz de Cánido.
5  * Proporciona detalles específicos sobre el Lobo en su hábitat natural.
6 */
7 public class LoboInterfaz implements InterfazCanido {
8     // Constantes para evitar repetición de valores
9     private static final String HABITAT = "bosque";
10    private static final String COMIDA = "carnívora";
11    private static final String SONIDO = "aulido";
12    private static final String NOMBRE_CIENTIFICO = "Canis lupus";
13
14    @Override
15    public String getSonido() {
16        return SONIDO;
17    }
18
19    @Override
20    public String getNombreCientifico() {
21        return NOMBRE_CIENTIFICO;
22    }
23
24    @Override
25    public String getEntorno() {
26        return HABITAT;
27    }
28
29    @Override
30    public String getComida() {
31        return COMIDA;
32    }
33
34    /**
35     * Método adicional específico de Los cánidos.
36     */
37    @Override
38    public String getCaracteristicaCanido() {
39        return "Lobo salvaje, depredador del bosque";
40    }
41 }
```

```
1 package animales;
2
3 /**
4  * Implementación de un Perro utilizando la interfaz de Cánido.
5  * Proporciona detalles específicos sobre el perro doméstico.
6 */
7 public class PerroInterfaz implements InterfazCanido {
8     // Constantes para evitar repetición de valores
9     private static final String HABITAT = "doméstico";
10    private static final String COMIDA = "carnívora";
11    private static final String SONIDO = "ladrido";
12    private static final String NOMBRE_CIENTIFICO = "Canis lupus familiaris";
13
14    @Override
15    public String getSonido() {
16        return SONIDO;
17    }
18
19    @Override
20    public String getNombreCientifico() {
21        return NOMBRE_CIENTIFICO;
22    }
23
24    @Override
25    public String getEntorno() {
26        return HABITAT;
27    }
28
29    @Override
30    public String getComida() {
31        return COMIDA;
32    }
33
34    /**
35     * Método adicional específico de los cánidos.
36     */
37    @Override
38    public String getCaracteristicaCanido() {
39        return "Perro doméstico, compañero del ser humano";
40    }
41 }
```

```
1 package animales;
2
3 /**
4  * Clase principal para probar las implementaciones de las interfaces de animales.
5  * Demuestra el uso de las interfaces y las implementaciones específicas.
6 */
7 public class MainInterfaz {
8     /**
9      * Método principal para ejecutar las pruebas de las clases de animales.
10     */
11    public static void main(String[] args) {
12        // Crear instancias de cada animal
13        InterfazCanido perro = new PerroInterfaz();
14        InterfazCanido lobo = new LoboInterfaz();
15        InterfazFelino leon = new LeonInterfaz();
16        InterfazFelino gato = new GatoInterfaz();
17
18        // Método para imprimir detalles de un animal
19        imprimirDetallesAnimal("Perro", perro);
20        imprimirDetallesAnimal("Lobo", lobo);
21        imprimirDetallesAnimal("León", leon);
22        imprimirDetallesAnimal("Gato", gato);
23    }
24
25 /**
26  * Método auxiliar para imprimir los detalles de un animal.
27  */
28 private static void imprimirDetallesAnimal(String nombreAnimal, InterfazAnimal animal) {
29     System.out.println("\n" + nombreAnimal + ":");
30     System.out.println("Sonido: " + animal.getSonido());
31     System.out.println("Nombre Científico: " + animal.getNombreCientifico());
32     System.out.println("Entorno: " + animal.getEntorno());
33     System.out.println("Comida: " + animal.getComida());
34
35     // Imprimir características específicas si es un Cánido o Felino
36     if (animal instanceof InterfazCanido) {
37         System.out.println("Característica Cánido: " +
38             ((InterfazCanido) animal).getCaracteristicaCanido());
39     }
40     if (animal instanceof InterfazFelino) {
41         System.out.println("Característica Felino: " +
42             ((InterfazFelino) animal).getCaracteristicaFelino());
43     }
44 }
45 }
```

Salida por terminal:

Perro:

Sonido: ladrido

Nombre Científico: *Canis lupus familiaris*

Entorno: doméstico

Comida: carnívora

Característica Cánido: Perro doméstico, compañero del ser humano

Lobo:

Sonido: aullido

Nombre Científico: *Canis lupus*

Entorno: bosque

Comida: carnívora

Característica Cánido: Lobo salvaje, depredador del bosque

León:

Sonido: rugido

Nombre Científico: *Panthera leo*

Entorno: pradera

Comida: carnívora

Característica Felino: León, rey de la pradera africana

Gato:

Sonido: maullido

Nombre Científico: *Felis silvestris catus*

Entorno: doméstico

Comida: ratones

Característica Felino: Gato doméstico, compañero de hogar