

**Instituto FOC**  
**Módulo de desarrollo de aplicaciones web**  
Programación

**BRUNO MARENCO CERQUEIRA**

**Tarea Individual 4: Herencia**

Enero/2018

## Índice

Índice .....	2
Crear un proyecto en NetBeans denominado, "CuentasBancarias" .....	3
Crear un paquete denominado "modeloBancario" .....	3
<b>Dentro del paquete "modeloBancario", crear una clase denominada Cliente, que modele los distintos clientes del banco que tienen una cuenta asociada para almacenar su dinero. Las características de la clase Cliente son: .....</b>	<b>4</b>
<b>Dentro del paquete "modeloBancario", crear una clase abstracta denominada "Cuenta", que modele las distintas cuentas bancarias que mantiene la entidad financiera donde los clientes depositan su dinero. ....</b>	<b>5</b>
<b>Dentro del paquete "modeloBancario", crear una clase denominada "CuentaCorriente", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés fijo del 1.5%. ....</b>	<b>7</b>
<b>Dentro del paquete "modeloBancario", crear una clase denominada "CuentaAhorro", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés variable y un saldo mínimo necesario. ....</b>	<b>8</b>
Crear un paquete denominado "Prueba", dentro de dicho paquete crear una clase "test" que importe el paquete "modeloBancario" y que posea un método main, que permita probar las distintas clases Cuentas implementadas. ....	9

Crear un proyecto en NetBeans denominado, "CuentasBancarias".

**Nuevo Aplicación Java**

**Pasos**

1. Seleccionar proyecto
2. **Nombre y ubicación**

**Nombre y ubicación**

Nombre proyecto:

Ubicación del proyecto:

Carpeta proyecto:

☐ Usar una carpeta dedicada para almacenar las bibliotecas

Carpeta de Bibliotecas:

Usuarios y proyectos diferentes pueden compartir las mismas librerías de compilación (ver la Ayuda para más detalles).

☐ Crear clase principal

Creando proyecto nuevo  
25%

< Atrás    Siguiente >    Terminar    Cancelar    **Ayuda**

Crear un paquete denominado "modeloBancario".

**Nuevo Java Package**

**Pasos**

1. Escoja el tipo de archivo
2. **Name and Location**

**Name and Location**

Package Name:

Project:

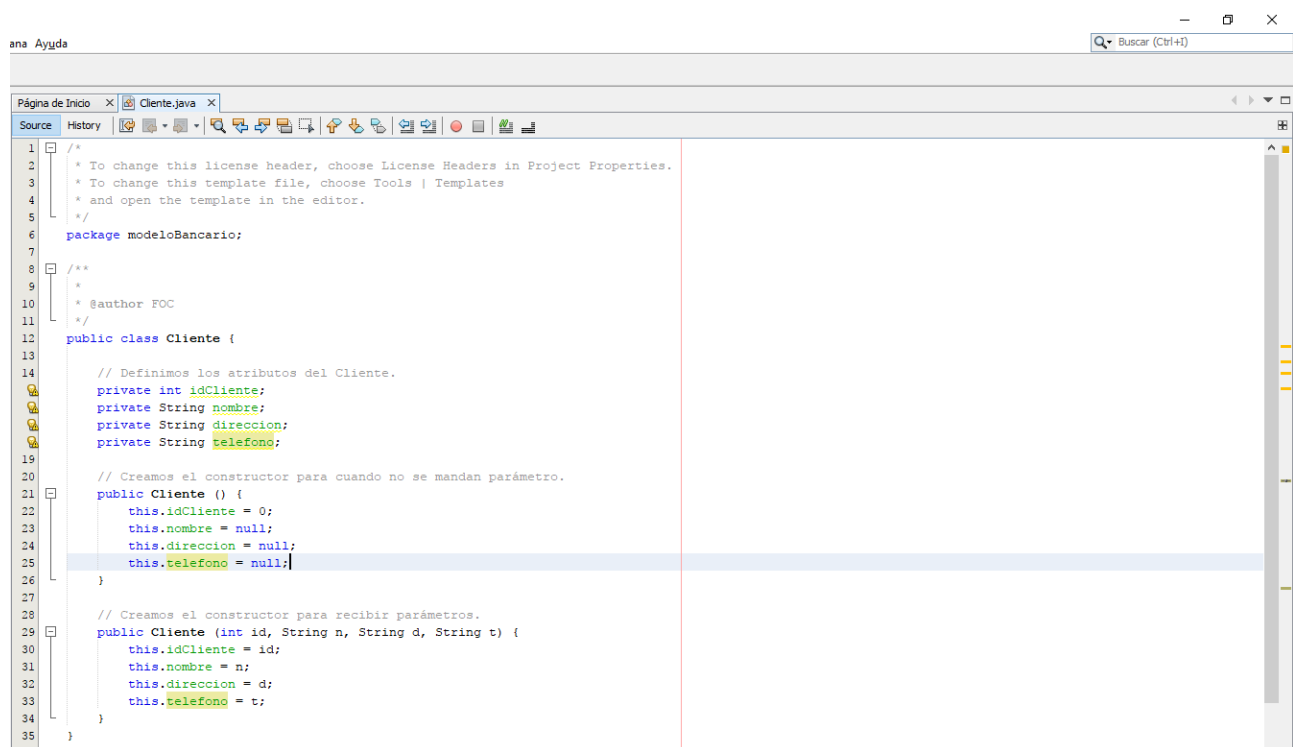
Location:

Created Folder:

< Atrás    Siguiente >    **Terminar**    Cancelar    Ayuda

Dentro del paquete "modeloBancario", crear una clase denominada Cliente, que modele los distintos clientes del banco que tienen una cuenta asociada para almacenar su dinero. Las características de la clase Cliente son:

- Atributos (Todos los atributos de la clase Cliente deben tener visibilidad privada):
  - idCliente: número entero que representa el identificador único del cliente dentro del banco.
  - nombre: cadena de caracteres que representa el nombre del cliente del banco.
  - direccion: cadena de caracteres que representa la dirección donde vive el cliente del banco.
  - teléfono: cadena de caracteres que representa el teléfono que permite contactar con el cliente.
- Métodos (Todos los métodos de la clase Cliente deben tener visibilidad pública):
  - constructor sin parámetros: constructor que inicializa todos los atributos de tipo cadenas de caracteres al valor null y los números enteros a 0.
  - constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes.



```
1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package modeloBancario;
7
8   /**
9   *
10  * @author FOC
11  */
12  public class Cliente {
13
14      // Definimos los atributos del Cliente.
15      private int idCliente;
16      private String nombre;
17      private String direccion;
18      private String telefono;
19
20      // Creamos el constructor para cuando no se mandan parámetro.
21      public Cliente () {
22          this.idCliente = 0;
23          this.nombre = null;
24          this.direccion = null;
25          this.telefono = null;
26      }
27
28      // Creamos el constructor para recibir parámetros.
29      public Cliente (int id, String n, String d, String t) {
30          this.idCliente = id;
31          this.nombre = n;
32          this.direccion = d;
33          this.telefono = t;
34      }
35  }
```

**Dentro del paquete "modeloBancario", crear una clase abstracta denominada "Cuenta", que modele las distintas cuentas bancarias que mantiene la entidad financiera donde los clientes depositan su dinero.**

- Atributos (Todos los atributos de la clase Cuenta deben tener visibilidad protegida)
  - numeroDeCuenta: número entero que representa el identificador único asociado a cada una de las cuentas del banco.
  - saldo: número real que representa la cantidad de dinero almacenado en dicha cuenta.
  - titular: atributo de tipo Cliente que representa la persona que está asociada a dicha cuenta.
- Métodos (Todos los métodos de la clase Cuenta deben tener visibilidad pública)
  - constructor sin parámetros: constructor que inicializa el cliente titular de la cuenta a null, y el saldo y el número de cuenta a cero.
  - constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes.
  - getNumeroDeCuenta: método que permite obtener el número de cuenta.
  - getSaldo: método que permite obtener el saldo de la cuenta.
  - getTitular: método que permite obtener el titular de la cuenta
  - setNumeroDeCuenta: método que recibe un parámetro entero que representa el nuevo número de cuenta que se desea asignar y asigna parámetro el valor de dicho al atributo numeroDeCuenta.
  - setSaldo: método que recibe un parámetro entero que representa el nuevo saldo que se desea asignar y asigna parámetro el valor de dicho al atributo saldo.
  - setTitular: método que recibe un parámetro Cliente que representa el nuevo titular que se desea asignar y asigna parámetro el valor de dicho al atributo titular.
  - ingresar: recibe un parámetro real que representa la cantidad que se desea ingresar en la cuenta. El método incrementará el saldo en la cantidad recibida como parámetro.
  - retirar: método abstracto que permitirá sacar una cantidad de la cuenta (si hay saldo disponible para ello), no se implementará ya que dependerá del tipo de cuenta, por tanto su implementación recaerá en las clases hijas.
  - actualizarSaldo: método abstracto que actualizará el saldo de la cuenta, dependiendo del tipo de interés de cada una de las cuenta, por tanto su implementación recaerá en las clases hijas.

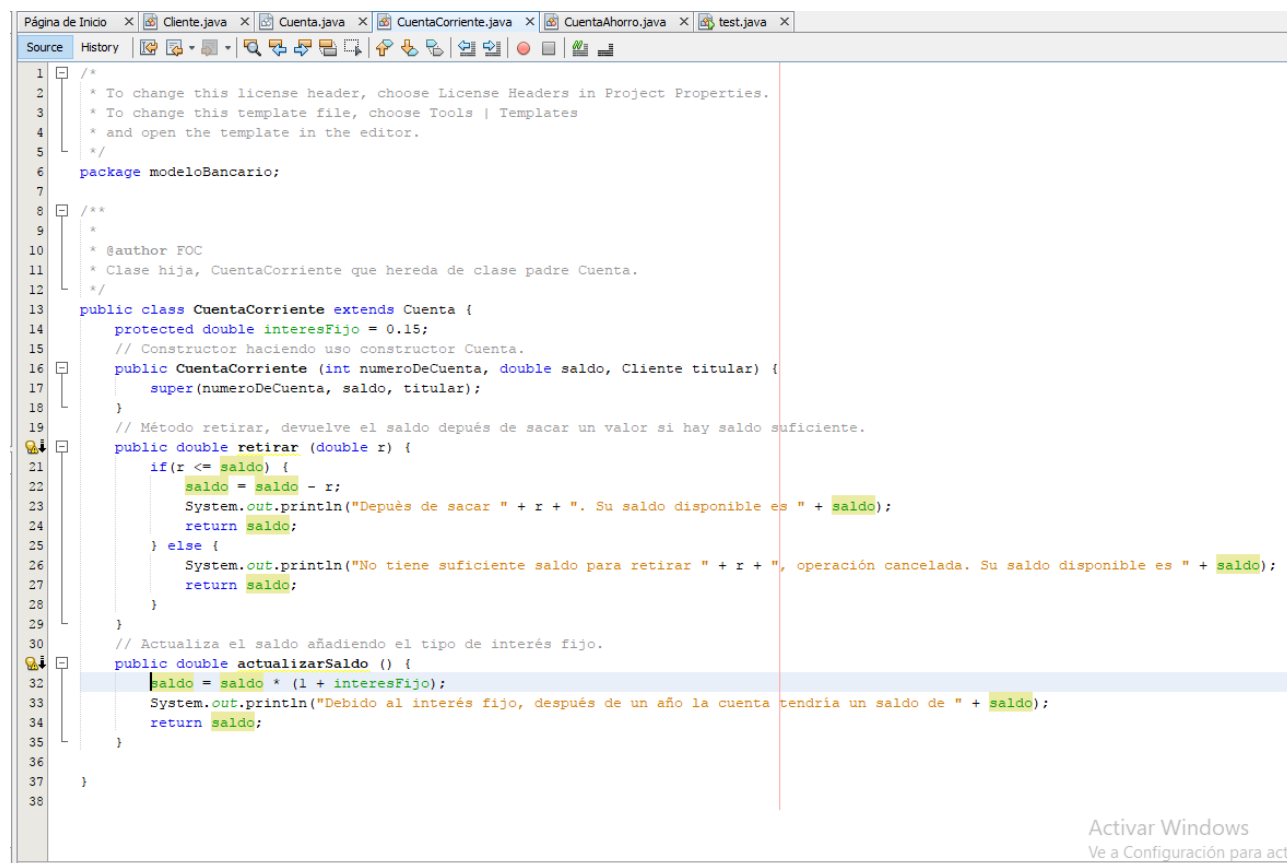
```
Página de Inicio x Cliente.java x Cuenta.java x
Source History
1 /**
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package modeloBancario;
7
8  /**
9   *
10   * @author FOC
11   */
12  public abstract class Cuenta {
13      protected int numeroDeCuenta;
14      protected double saldo;
15      protected Cliente titular;
16      // Constructor sem parámetros.
17      public Cuenta() {
18          this.titular = null;
19          this.saldo = 0;
20          this.numeroDeCuenta = 0;
21      }
22      // Constructor que recibe parámetros.
23      public Cuenta(int ncuenta, double s, Cliente t) {
24          this.titular = t;
25          this.saldo = s;
26          this.numeroDeCuenta = ncuenta;
27      }
28      // Método que permite obtener el número de cuenta.
29      public int getNumeroDeCuenta () {
30          return numeroDeCuenta;
31      }
32      // Método que permite obtener el saldo de la cuenta.
33      public double getSaldo () {
34          return saldo;
35      }
36      // Método getTitular, que permite obtener el titular de la cuenta
37      public Cliente getTitular () {
38          return titular;
39      }
40      // Método setNumeroDeCuenta que recibe un parámetro entero que representa el nuevo número de cuenta
```

```
Página de Inicio x Cliente.java x Cuenta.java x CuentaCorriente.java x CuentaAhorro.java x test.java x
Source History
41  * que se desea asignar y asigna parámetro el valor de dicho al atributo numeroDeCuenta.
42  */
43  public void setNumeroDeCuenta(int ncuenta) {
44      this.numeroDeCuenta = ncuenta;
45  }
46  /* Método setSaldo, que recibe un parámetro entero que representa el nuevo saldo que
47  * se desea asignar y asigna parámetro el valor de dicho al atributo saldo.
48  */
49  public void setSaldo(double s) {
50      this.saldo = s;
51  }
52  /* Método setTitular, que recibe un parámetro Cliente que representa el nuevo titular que
53  * se desea asignar y asigna parámetro el valor de dicho al atributo titular.
54  */
55  public void setTitular(Cliente t) {
56      this.titular = t;
57  }
58  /* ingresar: recibe un parámetro real que representa la cantidad que se desea ingresar
59  * en la cuenta. El método incrementará el saldo en la cantidad recibida como parámetro.
60  */
61  public void ingresar(double ing) {
62      this.saldo = this.saldo + ing;
63      System.out.println("Después de ingresar " + ing + ". El saldo ahora es de " + this.saldo);
64  }
65  /* retirar: método abstracto que permitirá sacar una cantidad de la cuenta
66  * (si hay saldo disponible para ello), no se implementará ya que dependerá del
67  * tipo de cuenta, por tanto su implementación recaerá en las clases hijas.
68  */
69  public abstract double retirar (double r);
70  /* actualizarSaldo: método abstracto que actualizará el saldo de la cuenta,
71  * dependiendo del tipo de interés de cada una de las cuenta, por tanto
72  * su implementación recaerá en las clases hijas.
73  */
74  public abstract double actualizarSaldo ();
75 }
76
```

Activar Windows

Dentro del paquete "modeloBancario", crear una clase denominada "CuentaCorriente", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés fijo del 1.5%.

- Atributos (Todos los atributos de la clase CuentaCorriente deben tener visibilidad protegida):
  - interesFijo: constante real cuyo valor es 0.15.
- Métodos (Todos los métodos de la clase CuentaCorriente deben tener visibilidad pública)
  - constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes. Dicho constructor debe hacer uso del constructor de la clase padre "Cuenta".
  - Implementación de los métodos abstractos retirar y actualizarSaldo.

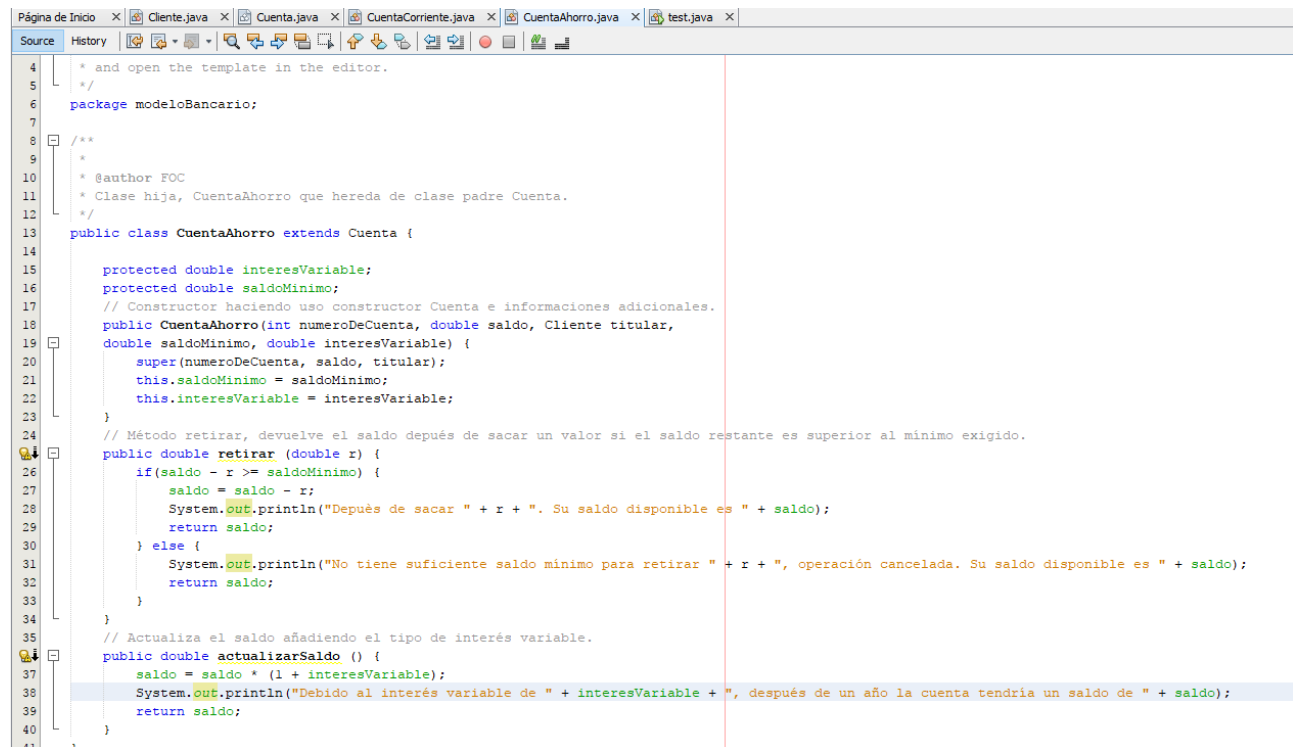


```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package modeloBancario;
7
8  /**
9   *
10   * @author FOC
11   * Clase hija, CuentaCorriente que hereda de clase padre Cuenta.
12   */
13  public class CuentaCorriente extends Cuenta {
14      protected double interesFijo = 0.15;
15      // Constructor haciendo uso constructor Cuenta.
16      public CuentaCorriente (int numeroDeCuenta, double saldo, Cliente titular) {
17          super(numeroDeCuenta, saldo, titular);
18      }
19      // Método retirar, devuelve el saldo después de sacar un valor si hay saldo suficiente.
20      public double retirar (double r) {
21          if(r <= saldo) {
22              saldo = saldo - r;
23              System.out.println("Después de sacar " + r + ". Su saldo disponible es " + saldo);
24              return saldo;
25          } else {
26              System.out.println("No tiene suficiente saldo para retirar " + r + ", operación cancelada. Su saldo disponible es " + saldo);
27              return saldo;
28          }
29      }
30      // Actualiza el saldo añadiendo el tipo de interés fijo.
31      public double actualizarSaldo () {
32          saldo = saldo * (1 + interesFijo);
33          System.out.println("Debido al interés fijo, después de un año la cuenta tendría un saldo de " + saldo);
34          return saldo;
35      }
36  }
37
38  }
```

Activar Windows  
Ve a Configuración para activar

Dentro del paquete "modeloBancario", crear una clase denominada "CuentaAhorro", que herede de la clase "Cuenta", que modele un tipo de cuenta con un interés variable y un saldo mínimo necesario.

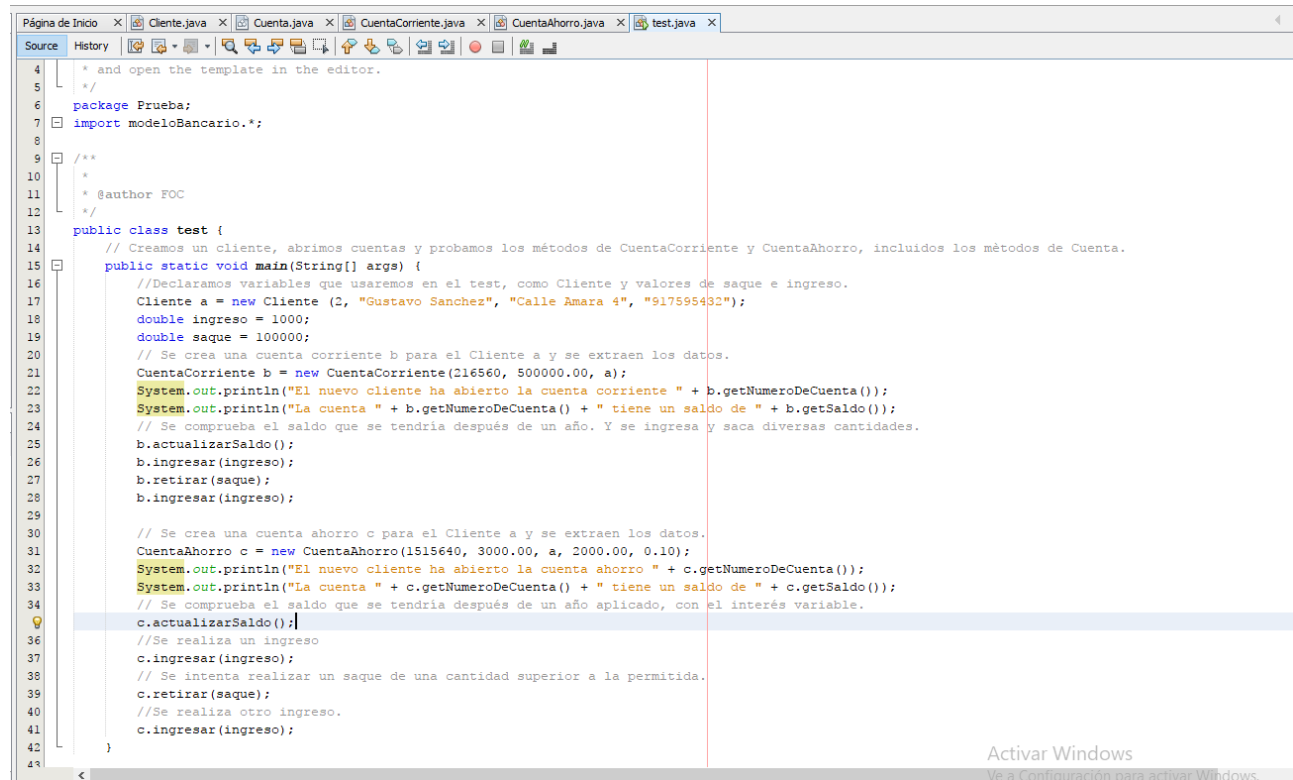
- Atributos (Todos los atributos de la clase CuentaAhorro deben tener visibilidad protegida).
  - interesVariable: número real que representa el tipo de interés que se aplica a la cuenta.
  - saldoMinimo: número real que representa el dinero mínimo que debe haber en la cuenta. Por tanto el valor del atributo saldo siempre debe ser mayor o igual que saldoMinimo
- Métodos (Todos los métodos de la clase CuentaAhorro deben tener visibilidad pública)
  - constructor con parámetros: constructor que tienen tantos parámetros como atributos tiene la clase, y que inicializa cada uno de los atributos con el valor de los parámetros correspondientes. Dicho constructor debe hacer uso del constructor de la clase padre "Cuenta".
  - Implementación de los métodos abstractos retirar y actualizarSaldo. Esta cuenta tiene como atributos el interés variable a lo largo del año y un saldo mínimo necesario. Al retirar dinero el saldo debe ser mayor o igual que el saldoMinimo.



```
4  * and open the template in the editor.
5  */
6  package modeloBancario;
7
8  /**
9   *
10  * @author FOC
11  * Clase hija, CuentaAhorro que hereda de clase padre Cuenta.
12  */
13  public class CuentaAhorro extends Cuenta {
14
15      protected double interesVariable;
16      protected double saldoMinimo;
17      // Constructor haciendo uso constructor Cuenta e informaciones adicionales.
18      public CuentaAhorro(int numeroDeCuenta, double saldo, Cliente titular,
19                          double saldoMinimo, double interesVariable) {
20          super(numeroDeCuenta, saldo, titular);
21          this.saldoMinimo = saldoMinimo;
22          this.interesVariable = interesVariable;
23      }
24      // Método retirar, devuelve el saldo después de sacar un valor si el saldo restante es superior al mínimo exigido.
25      public double retirar (double r) {
26          if(saldo - r >= saldoMinimo) {
27              saldo = saldo - r;
28              System.out.println("Después de sacar " + r + ". Su saldo disponible es " + saldo);
29              return saldo;
30          } else {
31              System.out.println("No tiene suficiente saldo mínimo para retirar " + r + ", operación cancelada. Su saldo disponible es " + saldo);
32              return saldo;
33          }
34      }
35      // Actualiza el saldo añadiendo el tipo de interés variable.
36      public double actualizarSaldo () {
37          saldo = saldo * (1 + interesVariable);
38          System.out.println("Debido al interés variable de " + interesVariable + ", después de un año la cuenta tendría un saldo de " + saldo);
39          return saldo;
40      }
41  }
```



Crear un paquete denominado "Prueba", dentro de dicho paquete crear una clase "test" que importe el paquete "modeloBancario" y que posea un método main, que permita probar las distintas clases Cuentas implementadas.



```
4  * and open the template in the editor.
5  */
6  package Prueba;
7  import modeloBancario.*;
8
9  /**
10   *
11   * @author FOC
12   */
13  public class test {
14      // Creamos un cliente, abrimos cuentas y probamos los métodos de CuentaCorriente y CuentaAhorro, incluidos los métodos de Cuenta.
15      public static void main(String[] args) {
16          //Declaramos variables que usaremos en el test, como Cliente y valores de saque e ingreso.
17          Cliente a = new Cliente (2, "Gustavo Sanchez", "Calle Amara 4", "917595432");
18          double ingreso = 1000;
19          double saque = 100000;
20          // Se crea una cuenta corriente b para el Cliente a y se extraen los datos.
21          CuentaCorriente b = new CuentaCorriente(216560, 500000.00, a);
22          System.out.println("El nuevo cliente ha abierto la cuenta corriente " + b.getNumeroDeCuenta());
23          System.out.println("La cuenta " + b.getNumeroDeCuenta() + " tiene un saldo de " + b.getSaldo());
24          // Se comprueba el saldo que se tendría después de un año. Y se ingresa y saca diversas cantidades.
25          b.actualizarSaldo();
26          b.ingresar(ingreso);
27          b.retirar(saque);
28          b.ingresar(ingreso);
29
30          // Se crea una cuenta ahorro c para el Cliente a y se extraen los datos.
31          CuentaAhorro c = new CuentaAhorro(1515640, 3000.00, a, 2000.00, 0.10);
32          System.out.println("El nuevo cliente ha abierto la cuenta ahorro " + c.getNumeroDeCuenta());
33          System.out.println("La cuenta " + c.getNumeroDeCuenta() + " tiene un saldo de " + c.getSaldo());
34          // Se comprueba el saldo que se tendría después de un año aplicado, con el interés variable.
35          c.actualizarSaldo();
36          //Se realiza un ingreso
37          c.ingresar(ingreso);
38          // Se intenta realizar un saque de una cantidad superior a la permitida.
39          c.retirar(saque);
40          //Se realiza otro ingreso.
41          c.ingresar(ingreso);
42      }
43  }
```

Salida - CuentasBancarias (run)

```
run:
El nuevo cliente ha abierto la cuenta corriente 216560
La cuenta 216560 tiene un saldo de 500000.0
Debido al interés fijo, después de un año la cuenta tendría un saldo de 575000.0
Después de ingresar 1000.0. El saldo ahora es de 576000.0
Después de sacar 100000.0. Su saldo disponible es 476000.0
Después de ingresar 1000.0. El saldo ahora es de 477000.0
El nuevo cliente ha abierto la cuenta ahorro 1515640
La cuenta 1515640 tiene un saldo de 3000.0
Debido al interés variable de 0.1, después de un año la cuenta tendría un saldo de 3300.0000000000000000
Después de ingresar 1000.0. El saldo ahora es de 4300.0
No tiene suficiente saldo mínimo para retirar 100000.0, operación cancelada. Su saldo disponible es 4300.0
Después de ingresar 1000.0. El saldo ahora es de 5300.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Salida Finished building CuentasBancarias (run).