**Tarea para PROG09.**

|  |
| --- |
| **Enunciado.** |
| **Estamos en disposición de dar persistencia a los datos que utilizan nuestras aplicaciones. Hasta el momento los datos manejados sólo se mantienen en memoria principal: cuando nuestras aplicaciones finalizan la ejecución todos los datos se pierden.**  **Ejercicio 1**  **Se trata de modificar la aplicación desarrollada en la Unidad de Trabajo 8, Ejercicio 1 para dar persistencia a los datos de las cuentas bancarias. Para ello:**  **• Cuando la aplicación finalice, es decir, el usuario seleccione la opción Salir, la aplicación guardará el objeto ArrayList de las cuentas bancarias en un fichero binario denominado datos\_apellidos.dat.**  **• Cuando la aplicación inicie la ejecución, antes de mostrar el menú, deberá cargar en la estructura de datos el contenido del fichero datos\_apellidos.dat. Si el fichero no existe, se empezará con el arrayList vacío mostrando el menú principal.**  **Recuerda que para poder realizar estas tareas es necesario que la clase CuentaBancaria sea serializable.**  **Ejercicio 2**  **Añade una nueva opción al menú de la aplicación denominado "Listado clientes" de modo que al seleccionarla, se genere un fichero de texto denominado listado\_apellidos.txt que contenga una línea de texto por cada cuenta bancaria, escribiendo el iban y el nombre del titular. La última línea del fichero contendrá el número total de cuentas existentes y la fecha actual del sistema del momento de generación del fichero.**  **IMPORTANTE**  **• En la cabecera de las clases añade documentación indicando autor y descripción de la clase.**  **• En la cabecera de cada método añade documentación indicando la funcionalidad que implementa y el valor que devuelve.**  **• El código fuente Java de esta clase debería incluir comentarios en cada atributo (o en cada conjunto de atributos) y método (o en cada conjunto de métodos del mismo tipo) indicando su utilidad.** |

**Archivo: Prog09\_1Tarea**

**1 package prog09;**

**2**

**3 import java.util.InputMismatchException;**

**4 import java.util.\*;**

**5 import java.util.regex.\*;**

**6 import java.io.\*;**

**7 import java.util.logging.Level;**

**8 import java.util.logging.Logger;**

**9**

**10 /\*\***

**11 \* Ejercicio: 9 Práctica Persistencia de Datos**

**12 \***

**13 \* @author Juan Marí Ibáñez Fecha última modificación: 20.03.2022**

**14 \*/**

**15 public class Prog09 {**

**16**

**17 public static void main(String[] args) {**

**18**

**19 int opcion;//Variable que almacena numero de opción**

**20 Pattern patronIBAN = Pattern.compile("(ES[0-9]{22})");//Expresión regular IBAN**

**21 Pattern patronDNI = Pattern.compile("(0-9)[7-8](A-Z a-z)[1]");//Expresión regular DNI**

**22 ListaCuentas listaCuentas = new ListaCuentas();//Estructura de almacenamiento**

**23**

**24 listaCuentas.cargaArray();**

**25**

**26 do {**

**27**

**28 opcion = Menu.menu();**

**29**

**30 switch (opcion) {**

**31**

**32 ///////////////Abrir una nueva cuenta///////////////////////////**

**33 case 1: {**

**34 //Apellidos**

**35 String nombre = dameNombre();**

**36**

**37 //Nombre**

**38 String apellidos = dameApellidos();**

**39**

**40 //DNI**

**41 String DNI;**

**42 Persona cliente = null;**

**43 Dni DNICliente = null;**

**44 DNI = Dni.dameDNI();**

**45 DNICliente = new Dni(DNI);**

**46**

**47 try {**

**48 cliente = new Persona(nombre, apellidos, DNICliente);**

**49 } catch (Exception e) {**

**50 System.out.println(e.getMessage());**

**51 }**

**52**

**53 //IBAN**

**54 String IBAN = listaCuentas.crearIBAN();**

**55**

**56 //Saldo Inicial**

**57 double saldoInicial = dameSaldoInicial();**

**58**

**59 //Tipo de Cuenta**

**60 int tipoCuenta = tipoCuenta();**

**61**

**62 switch (tipoCuenta) {**

**63 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CuentaCorriente\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**64 case 1: {**

**65 CuentaBancaria cuentaCorriente = creaCuentaCorriente(cliente, saldoInicial, IBAN);**

**66 listaCuentas.abrirCuenta(cuentaCorriente);**

**67 System.out.println("----------------------------------------------------\n"**

**68 + " CUENTA CREADA CON ÉXITO\n"**

**69 + "----------------------------------------------------\n");**

**70**

**71 break;**

**72 }**

**73 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CuentaAhorro\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**74 case 2: {**

**75 CuentaBancaria cuentaAhorro = creaCuentaAhorro(cliente, saldoInicial, IBAN);**

**76 listaCuentas.abrirCuenta(cuentaAhorro);**

**77**

**78 System.out.println("----------------------------------------------------\n"**

**79 + " CUENTA CREADA CON ÉXITO\n"**

**80 + "----------------------------------------------------\n");**

**81**

**82 break;**

**83 }**

**84 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Opción por Defecto\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**85 default: {**

**86 System.out.println("----------------------------------------------------\n"**

**87 + " ERROR: DEBE INTRODUCIR UNA OPCIÓN VÁLIDA\n"**

**88 + "----------------------------------------------------\n");**

**88 + \n );**

**89**

**90 break;**

**91 }**

**92**

**93 }**

**94**

**95 break;**

**96 }**

**97**

**98 ///////////////Ver listado de cuentas disponibles.//////////////**

**99 case 2: {**

**100 String[] lista = listaCuentas.listadoCuentas();**

**101 System.out.println("\n----------------------------------------------------\n"**

**102 + " LISTADO DE CUENTAS BANCARIAS ALMACENADAS\n"**

**103 + "----------------------------------------------------\n");**

**104 for (int i = 0; i < lista.length; i++) {**

**105 System.out.println(lista[i]);**

**106 }**

**107**

**108 break;**

**109**

**110 }**

**111**

**112 //////////////////Obtener datos de una cuenta///////////////////**

**113 case 3: {**

**114 boolean existeIBAN = false;**

**115 boolean IBANOK = false;**

**116 String IBAN = "";**

**117**

**118 IBAN = Utilidades.compruebaIBAN();**

**119 existeIBAN = listaCuentas.existeIBAN(IBAN);**

**120**

**121 if (existeIBAN == false) {**

**122 System.out.println("\n----------------------------------------------------\n"**

**123 + "ERROR!! NO SE ENCUENTRA LA CUENTA ASOCIADA AL IBAN:\n "**

**124 + " " + IBAN**

**125 + "\n----------------------------------------------------\n");**

**126 } else if (existeIBAN == true) {**

**127 String datosCuenta = "\n" + listaCuentas.informacionCuenta(IBAN) + "\n";**

**128 System.out.println("\n----------------------------------------------------\n"**

**129 + "INFORMACIÓN DE LA CUENTA: \n " + datosCuenta);**

**130**

**131 System.out.println("\n----------------------------------------------------\n");**

**132**

**133 }**

**134**

**135 break;**

**136**

**137 }**

**138**

**139 ////////////// Realizar ingreso en una cuenta///////////////////**

**140 case 4: {**

**141**

**142 boolean continuaCantidadIngresar = false;**

**143 double cantidadIngresar = 0;**

**144**

**145 boolean existeIBAN = false;**

**146 boolean IBANOK = false;**

**147 String IBAN = "";**

**148**

**149 do {**

**150 IBAN = Utilidades.compruebaIBAN();**

**151 existeIBAN = listaCuentas.existeIBAN(IBAN);**

**152**

**153 if (existeIBAN == false) {**

**154 System.out.println("\n----------------------------------------------------\n"**

**155 + "ERROR!! NO SE ENCUENTRA LA CUENTA ASOCIADA AL IBAN:\n "**

**156 + " " + IBAN**

**157 + "\n----------------------------------------------------\n");**

**158 } else if (existeIBAN == true) {**

**159 IBANOK = true;**

**160 }**

**161 } while (IBANOK == false);**

**162**

**163 do {**

**164 try {**

**165 cantidadIngresar = Utilidades.llegirDouble("INTRODUZCA LA CANTIDAD QUE DESEA INGRESAR: ");**

**166 } catch (InputMismatchException e) {**

**167 System.out.println("----------------------------------------------------\n"**

**168 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**169 + "----------------------------------------------------\n");**

**170 }**

**171**

**172 if (cantidadIngresar > 0) {**

**173 continuaCantidadIngresar = true;**

**174 }**

**175**

**176 if (cantidadIngresar <= 0) {**

**177 System.out.println("----------------------------------------------------\n"**

**178 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**179 + "----------------------------------------------------\n");**

**180 }**

**181**

**182 } while (continuaCantidadIngresar == false);**

**183**

**184 boolean ingreso = listaCuentas.ingresoCuenta(IBAN, cantidadIngresar);**

**185**

**186 if (ingreso == true) {**

**187 System.out.println("----------------------------------------------------\n"**

**188 + "INGRESO REALIZADO EN LA CUENTA ASOCIADA AL IBAN:\n "**

**189 + IBAN);**

**190 System.out.println("\n----------------------------------------------------\n"**

**191 + "SU SAlDO ACTUAL ES:: " + listaCuentas.obtenerSaldo(IBAN) + "\n");**

**192**

**193 } else if (ingreso == false) {**

**194 System.out.println("----------------------------------------------------\n"**

**195 + "NO SE HA PODIDO REALIZAR EL INGRESO EN LA CUENTA CON IBAN:\n "**

**196 + IBAN);**

**197**

**198 }**

**199**

**200 break;**

**201**

**202 }**

**203**

**204 //////////////Retirar efectivo de una cuenta////////////////////**

**205 case 5: {**

**206**

**207 double cantidadRetirar = 0;**

**208 boolean continuaCantidadRetirar = false;**

**209**

**210 boolean existeIBAN = false;**

**211 boolean IBANOK = false;**

**212 String IBAN = "";**

**213**

**214 do {**

**215 IBAN = Utilidades.compruebaIBAN();**

**216 existeIBAN = listaCuentas.existeIBAN(IBAN);**

**217**

**218 if (existeIBAN == false) {**

**219 System.out.println("\n----------------------------------------------------\n"**

**220 + "ERROR!! NO SE ENCUENTRA LA CUENTA ASOCIADA AL IBAN:\n "**

**221 + " " + IBAN**

**222 + "\n----------------------------------------------------\n");**

**223 } else if (existeIBAN == true) {**

**224 IBANOK = true;**

**225 }**

**226 } while (IBANOK == false);**

**227**

**228 do {**

**229**

**230 try {**

**231**

**232 cantidadRetirar = Utilidades.llegirDouble("INTRODUZCA LA CANTIDAD QUE DESEA RETIRAR: ");**

**233 } catch (InputMismatchException e) {**

**234 System.out.println("\n----------------------------------------------------\n"**

**235 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**236 + "----------------------------------------------------\n");**

**237 }**

**238**

**239 if (cantidadRetirar > 0) {**

**240 continuaCantidadRetirar = true;**

**241 }**

**242 if (cantidadRetirar <= 0) {**

**243 System.out.println("\n----------------------------------------------------\n"**

**244 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**245 + "----------------------------------------------------\n");**

**246 }**

**247**

**248 } while (continuaCantidadRetirar == false);**

**249**

**250 boolean retirada = listaCuentas.retiradaCuenta(IBAN, cantidadRetirar);**

**251**

**252 if (retirada == true) {**

**253**

**254 System.out.println("----------------------------------------------------\n"**

**255 + "RETIRADA REALIZADA EN LA CUENTA ASOCIADA AL IBAN:\n "**

**256 + IBAN + "\n");**

**257 System.out.println("\n----------------------------------------------------\n"**

**258 + "SU SAlDO ACTUAL ES:: " + listaCuentas.obtenerSaldo(IBAN) + "\n");**

**259 } else if (retirada == false) {**

**260 // System.out.println("\n----------------------------------------------------\n"**

**261 // + "LA RETIRADA EN LA CUENTA ASOCIADA AL IBAN:\n "**

**262 // + IBAN + "\n"**

**263 // + "NO SE PUEDE REALIZAR POR FALTA DE FONDOS EN LA CUENTA");**

**264**

**265 System.out.println("\n----------------------------------------------------\n"**

**266 + "SU SAlDO ACTUAL ES:: " + listaCuentas.obtenerSaldo(IBAN) + "\n");**

**267 }**

**268**

**269 break;**

**270 }**

**271**

**272 ///////////Consultar el saldo actual de una cuenta//////////////**

**273 case 6: {**

**274**

**275 String IBAN = Utilidades.compruebaIBAN();**

**276 boolean existeIBAN = listaCuentas.existeIBAN(IBAN);**

**277**

**278 if (existeIBAN == true) {**

**279 double saldo = listaCuentas.obtenerSaldo(IBAN);**

**280**

**281 System.out.println("\n----------------------------------------------------\n"**

**282 + "EL SALDO DE LA CUENTA CON IBAN " + IBAN + " ES DE: " + saldo**

**283 + "\n----------------------------------------------------\n");**

**284 }**

**285 if (existeIBAN == false) {**

**286 System.out.println("\n----------------------------------------------------\n"**

**287 + "ERROR!! NO SE ENCUENTRA LA CUENTA ASOCIADA AL IBAN:\n "**

**288 + " " + IBAN**

**289 + "\n----------------------------------------------------\n");**

**290 }**

**291**

**292 break;**

**293 }**

**294 //////////////////Eliminar Cuenta Bancaria//////////////////////**

**295 case 7: {**

**296 String IBAN = Utilidades.compruebaIBAN();**

**297 boolean existeIBAN = listaCuentas.existeIBAN(IBAN);**

**298 int indice = 0;**

**299**

**300 if (existeIBAN == true) {**

**301 indice = listaCuentas.dameIndice(IBAN);**

**302 listaCuentas.eliminaCuenta(indice);**

**303 System.out.println("\n----------------------------------------------------\n"**

**304 + "SE HA ELIMINADO LA CUENTA ASOCIADA AL IBAN:\n "**

**305 + " " + IBAN**

**306 + "\n----------------------------------------------------\n");**

**307**

**308 }**

**309 if (existeIBAN == false) {**

**310 System.out.println("\n----------------------------------------------------\n"**

**311 + "ERROR!! NO SE ENCUENTRA LA CUENTA ASOCIADA AL IBAN: "**

**312 + " " + IBAN**

**313 + "\n----------------------------------------------------\n");**

**314**

**315 }**

**316 break;**

**317 }**

**318**

**319 ///////////////Mostrar número de Cuentas Ahorro/////////////////**

**320 case 8: {**

**321 System.out.println("\n----------------------------------------------------");**

**322 System.out.println("\n El de cuentas de Ahorro es: " + listaCuentas.numeroCuentasAhorro()**

**323 + "\n----------------------------------------------------\n");;**

**324**

**325 break;**

**326 }**

**327**

**328 /////Mostrar saldo acumulado de todas las Cuentas Corrientes////**

**329 case 9: {**

**330 double comision = 0;**

**331 boolean continuaComision = false;**

**332**

**333 do {**

**334**

**335 try {**

**336**

**337 comision = Utilidades.llegirDouble("INTRODUZCA LA COMISON POR LA QUE QUIERE FILTRAR: ");**

**338 } catch (InputMismatchException e) {**

**339 System.out.println("\n----------------------------------------------------\n"**

**340 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**341 + "----------------------------------------------------\n");**

**342 }**

**343**

**344 if (comision > 0) {**

**345 continuaComision = true;**

**346 }**

**347 if (comision <= 0) {**

**348 System.out.println("\n----------------------------------------------------\n"**

**349 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**350 + "----------------------------------------------------\n");**

**\ );**

**351 }**

**352**

**353 } while (continuaComision == false);**

**354 System.out.println("\n----------------------------------------------------");**

**355 System.out.println("\n El saldo acumulado en las cuentas \ncorrientes con comisión "+comision+" es: "**

**356 + listaCuentas.calculaSaldoTotal(comision)+"\n"**

**357 + "\n----------------------------------------------------\n");**

**358**

**359 break;**

**360**

**361 }**

**362**

**363 ///////////////Mostrar las tres cuentas con mayor saldo/////////**

**364 case 10: {**

**365**

**366 String[] lista = listaCuentas.tresMayorSaldo();**

**367 System.out.println("\n----------------------------------------------------\n"**

**368 + " TRES PRIMERAS CUENTAS CON MAYOR SALDO\n"**

**369 + "----------------------------------------------------\n");**

**370 for (int i = 0; i < lista.length; i++) {**

**371 System.out.println(lista[i]);**

**372 }**

**373**

**374 break;**

**375 }**

**376 //////////////////Listado de Clientes///////////////////////////**

**377 case 11: {**

**378**

**379 listaCuentas.listadoClientes();**

**380**

**381 break;**

**382 }**

**383 //////////////////termina el programa///////////////////////////**

**384 case 12: {**

**385**

**386 listaCuentas.guardaArray();**

**387**

**388 System.out.println("\n----------------------------------------------------");**

**389 System.out.println("----------------FIN DEL PROGRAMA-------------------");**

**390 System.out.println("----------------------------------------------------\n");**

**391**

**392 break;**

**393 }**

**394**

**395 ////////////////// Opcion por defecto //////////////////////////**

**396 default: {**

**397 System.out.println("----------------------------------------------------");**

**398 System.out.println(" ERROR: INTRODUZCA UN NUMERO DEL 1 AL 12");**

**399 System.out.println("----------------------------------------------------\n");**

**400 break;**

**401 }**

**402**

**403 }//Fin Switch**

**404**

**405 } while (opcion**

**406 != 12);**

**407**

**408 }// Fin Método principal**

**409**

**410 // Función: Pide String para introducir un nombre de cliente**

**411 // Devuelve: Devuelve String nombre con el nombre introducido**

**412 public static String dameNombre() {**

**413 String nombre = "";**

**414 boolean nombreOK = false;**

**415 do {**

**416 nombre = Utilidades.llegirString("----------------------------------------------------\n"**

**417 + "Introduzca nombre: ").toUpperCase();**

**418 if (nombre.length() != 0) {**

**419 nombreOK = true;**

**420 }**

**421 if (nombre.length() == 0) {**

**422 System.out.println("\n----------------------------------------------------\n"**

**423 + "ERROR!! DEBE INTRODUCIR UN NOMBRE: ");**

**424 }**

**425 } while (nombreOK == false);**

**426**

**427 return nombre;**

**428 }**

**429**

**430 // Función: Pide String para introducir los apellidos del cliente**

**431 // Devuelve: Devuelve String apellidos con los apellidos introducidos**

**432 public static String dameApellidos() {**

**433 String apellidos = "";**

**434 boolean apellidosOK = false;**

**435 do {**

**436 apellidos = Utilidades.llegirString("Introduzca apellidos: ").toUpperCase();**

**437 if (apellidos.length() != 0) {**

**438 apellidosOK = true;**

**439 }**

**440 if (apellidos.length() == 0) {**

**441 System.out.println("\n----------------------------------------------------\n"**

**442 + "ERROR!! DEBE INTRODUCIR LOS APELLIDOS: ");**

**443 }**

**444**

**445 } while (apellidosOK == false);**

**446**

**447 return apellidos;**

**448 }**

**449**

**450 // Función: Pide por consola un double como saldo inicial**

**451 // Devuelve: double saldoInicial con valor introducido**

**452 public static double dameSaldoInicial() {**

**453 double saldoIncial = 0;**

**454 boolean continua = false;**

**455**

**456 do {**

**457 try {**

**458**

**459 saldoIncial = Utilidades.llegirDouble("Introduzca saldo inicial: ");**

**460 } catch (InputMismatchException e) {**

**461 System.out.println("----------------------------------------------------\n"**

**462 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**463 + "----------------------------------------------------");**

**464**

**465 }**

**466**

**467 if (saldoIncial > 0) {**

**468 continua = true;**

**469**

**470 }**

**471 if (saldoIncial <= 0) {**

**472 System.out.println("----------------------------------------------------\n"**

**473 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**474 + "----------------------------------------------------");**

**475 continua = false;**

**476 }**

**477**

**478 } while (continua == false);**

**479**

**480 return saldoIncial;**

**481 }**

**482**

**483 //Función: pide elección de tipo de cuenta por consola**

**484 //Devuelve: int valor 1 o 2 en función del tipo de cuenta que se desea crear**

**485 public static int tipoCuenta() {**

**486 int tipoCuenta = 0;**

**487 boolean opcion = false;**

**488**

**489 do {**

**490 try {**

**491 tipoCuenta = Utilidades.llegirSencer("----------------------------------------------------\n"**

**492 + "Por favor elija una opción \n"**

**493 + "1. Cuenta Corriente\n"**

**494 + "2. Cuenta de Ahorro\n"**

**495 + "----------------------------------------------------\n");**

**496 } catch (InputMismatchException e) {**

**497 System.out.println("----------------------------------------------------\n"**

**498 + " ERROR: SÓLO ADMITE OPCION 1 O 2\n"**

**499 + "----------------------------------------------------");**

**500**

**501 }**

**502 if (tipoCuenta == 1 | tipoCuenta == 2) {**

**503 opcion = true;**

**504 } else {**

**505 System.out.println("----------------------------------------------------\n"**

**506 + " ERROR: DEBE INTRODUCIR UNA OPCIÓN VÁLIDA\n"**

**507 + "----------------------------------------------------");**

**508**

**509 }**

**510 } while (opcion == false);**

**511**

**512 return tipoCuenta;**

**513 }**

**514**

**515 //Función: crea un objeto CuentaCorriente con los parametros que le pasamos al método introducidos previamente**

**516 //Devuelve: objeto CuentaCorriente con los valores introducidos por consola**

**517 public static CuentaCorriente creaCuentaCorriente(Persona cliente, double saldoIncial, String IBAN) {**

**518 CuentaCorriente cuentaCorriente = null;**

**519 Persona persona = cliente;**

**520 double sInicial = saldoIncial;**

**521 String numIBAN = IBAN;**

**522 boolean continuaComisionMantenimiento = false;**

**523 double comisionMantenimiento = 0;**

**524 boolean continuaTipoInteresDescubierto = false;**

**525 d bl ti I t D bi t 0**

**525 double tipoInteresDescubierto = 0;**

**526 boolean continuaMaximoDescubierto = false;**

**527 double maximoDescubierto = 0;**

**528**

**529 //////////Introduccion Válida de Comisión de mantenimiento//////////////**

**530 do {**

**531 try {**

**532 comisionMantenimiento = Utilidades.llegirDouble("----------------------------------------------------\n"**

**533 + "INTRODUZCA LA COMISIÓN DE MANTENIMIENTO: ");**

**534 } catch (InputMismatchException e) {**

**535 System.out.println("----------------------------------------------------\n"**

**536 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**537 + "----------------------------------------------------");**

**538**

**539 }**

**540**

**541 if (comisionMantenimiento > 0) {**

**542 continuaComisionMantenimiento = true;**

**543**

**544 }**

**545 if (comisionMantenimiento <= 0) {**

**546 System.out.println("----------------------------------------------------\n"**

**547 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**548 + "----------------------------------------------------");**

**549 continuaComisionMantenimiento = false;**

**550 }**

**551 } while (continuaComisionMantenimiento == false);**

**552**

**553 /////////Introducion Válida de Tipo de Interes por Descubierto//////////**

**554 do {**

**555 try {**

**556 tipoInteresDescubierto = Utilidades.llegirDouble(**

**557 "INTRODUZCA EL TIPO DE INTERES POR DESCUBIERTO: ");**

**558 } catch (InputMismatchException e) {**

**559 System.out.println("----------------------------------------------------\n"**

**560 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**561 + "----------------------------------------------------");**

**562**

**563 }**

**564 if (tipoInteresDescubierto > 0) {**

**565 continuaTipoInteresDescubierto = true;**

**566**

**567 }**

**568 if (tipoInteresDescubierto <= 0) {**

**569 System.out.println("----------------------------------------------------\n"**

**570 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**571 + "----------------------------------------------------");**

**572 continuaComisionMantenimiento = false;**

**573**

**574 }**

**575 } while (continuaTipoInteresDescubierto == false);**

**576**

**577 ////////////Introducción Válida de Máximo Descubierto///////////////////**

**578 do {**

**579 try {**

**580 maximoDescubierto = Utilidades.llegirDouble(**

**581 "INTRODUZCA LA CANTIDAD MÁXIMA POR DESCUBIERTO: ");**

**582 } catch (InputMismatchException e) {**

**583 System.out.println("----------------------------------------------------\n"**

**584 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**585 + "----------------------------------------------------");**

**586**

**587 }**

**588 if (maximoDescubierto > 0) {**

**589 continuaMaximoDescubierto = true;**

**590**

**591 }**

**592 if (maximoDescubierto <= 0) {**

**593 System.out.println("----------------------------------------------------\n"**

**594 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**595 + "----------------------------------------------------");**

**596**

**597 continuaMaximoDescubierto = false;**

**598 }**

**599**

**600 } while (continuaMaximoDescubierto == false);**

**601**

**602 try {**

**603 cuentaCorriente = new CuentaCorriente(persona, sInicial, numIBAN, comisionMantenimiento, tipoInteresDescubierto, maximoDescubierto);**

**604 } catch (Exception e) {**

**605 System.out.println(e.getMessage());**

**606 }**

**607 return cuentaCorriente;**

**608 }**

**609**

**610 //Función: crea un objeto CuentaAhorro con los parametros que le pasamos al método introducidos previamente**

**611 //Devuelve: objeto CuentaAhorro con los valores introducidos por consola**

**612 public static CuentaAhorro creaCuentaAhorro(Persona cliente, double saldoInicial, String IBAN) {**

**612 public static CuentaAhorro creaCuentaAhorro(Persona cliente, double saldoInicial, String IBAN) {**

**613**

**614 CuentaAhorro cuentaAhorro = null;**

**615 Persona persona = cliente;**

**616 double sInicial = saldoInicial;**

**617 double tipoInteresAnual = 0;**

**618 String numIBAN = IBAN;**

**619 boolean continuaTipoInteresAnual = false;**

**620 do {**

**621 try {**

**622 tipoInteresAnual = Utilidades.llegirDouble("----------------------------------------------------\n"**

**623 + "INTRODUZCA EL TIPO DE INTERES ANUAL: ");**

**624**

**625 } catch (InputMismatchException e) {**

**626 System.out.println("----------------------------------------------------\n"**

**627 + " ERROR: ENTRADA DE DATOS NO COMPATIBLE\n"**

**628 + "----------------------------------------------------");**

**629**

**630 }**

**631 if (tipoInteresAnual > 0) {**

**632 continuaTipoInteresAnual = true;**

**633 }**

**634 if (tipoInteresAnual <= 0) {**

**635 System.out.println("----------------------------------------------------\n"**

**636 + " ERROR: DEBE INTRODUCIR UN VALOR POSITIVO\n"**

**637 + "----------------------------------------------------");**

**638**

**639 continuaTipoInteresAnual = false;**

**640 }**

**641**

**642 } while (continuaTipoInteresAnual == false);**

**643 try {**

**644 cuentaAhorro = new CuentaAhorro(persona, sInicial, numIBAN, tipoInteresAnual);**

**645 } catch (Exception e) {**

**646 System.out.println(e.getMessage());**

**647 }**

**648 return cuentaAhorro;**

**649 }**

**650**

**651 }//Fin Clase**

**Archivo: ListaCuentas.java**

**1 package prog09;**

**2**

**3 import java.io.\*;**

**4 import java.util.\*;**

**5 import java.util.logging.Level;**

**6 import java.util.logging.Logger;**

**7 import java.util.regex.Matcher;**

**8 import java.util.regex.Pattern;**

**9**

**10 /\*\***

**11 \* Ejercicio Programa 9**

**12 \***

**13 \* @author Juan Marí Ibáñez Fecha última modificación: 20.03.2022**

**14 \*/**

**15 public class ListaCuentas {**

**16**

**17 private ArrayList<CuentaBancaria> listaCuentas;// Lista Dinámica de cuentas(La variable Genérica debe ser una clase**

**18**

**19 //Metodo Constructor por Defecto**

**20 public ListaCuentas() {**

**21 listaCuentas = new ArrayList<>();**

**22 }**

**23**

**24 //Método que crea y devuelve una copia del ArrayList original**

**25 public ArrayList<CuentaBancaria> getCuentaBancaria() {**

**26 ArrayList<CuentaBancaria> nuevaListaCuentas = new ArrayList<>();**

**27 return nuevaListaCuentas;**

**28 }**

**29**

**30 // Método que crea un IBAN válido a partir de una expresión regular y comprueba que sea unico**

**31 //devuelve String IBAN**

**32 public String crearIBAN() {**

**33 String IBAN = "";**

**34 Pattern patronIBAN = Pattern.compile("(ES[0-9]{22})");**

**35 boolean continuaIBAN = false;**

**36**

**37 do {**

**38 IBAN = Utilidades.llegirString( //Lee por teclado el IBAN**

**39 "Introduzca IBAN: ").toUpperCase();**

**40 Matcher coincidencia = patronIBAN.matcher(IBAN);**

**41**

**42 if (IBAN.length() == 0) {//Si el valor introducido es 0 salta error**

**43 System.out.println("\n----------------------------------------------------\n"**

**44 + "ERROR!! EL IBAN NO PUEDE QUEDAR VACÍO ");**

**45 continuaIBAN = false;**

**46**

**47 }**

**48**

**49 if (coincidencia.matches()) {// Si el valor intoducido por teclado es coincide con el patron continúa**

**50 if (listaCuentas.size() == 0) {//Si la lista esta vacía continua con el proceso**

**51 continuaIBAN = true;**

**52 }**

**53 if (listaCuentas.size() != 0) {//si hay elementos en la lista recorre la lista y busca si esta repetido**

**54**

**55 for (CuentaBancaria b : listaCuentas) {**

**56**

**57 if (b.getIBAN().equals(IBAN)) {//Si el IBAN está repetido salta error**

**58 System.out.println("\n----------------------------------------------------\n"**

**59 + "ERROR!! YA EXISTE EL IBAN: " + IBAN + "\n POR FAVOR, INTRODUZCA OTRO IBAN");**

**60 continuaIBAN = false;**

**61**

**62 }**

**63 if (!b.getIBAN().equals(IBAN)) {// Si el valor no está repetido true**

**64 continuaIBAN = true;**

**65 }**

**66 }**

**67 }**

**68 } else {//Si el valor introducido no coincide con el patron salta error**

**69 System.out.println("\n----------------------------------------------------\n"**

**70 + "ERROR!! INTRODUZCA UN IBAN VÁLIDO (ES1111111111111111111111: ");**

**71 continuaIBAN = false;**

**72 }**

**73**

**74 } while (continuaIBAN == false);**

**75**

**76 return IBAN;**

**77 }**

**78**

**79 //Metodo que comprueba si el IBAN existe en el ArrayList listaCuentas**

**80 //devuelve boolean IBANOK con true o false**

**81 public boolean existeIBAN(String IBAN) {**

**82 boolean IBANOK = false;**

**83**

**84 for (CuentaBancaria b : listaCuentas) {**

**85**

**86 if (b.getIBAN().equals(IBAN)) {**

**( g () q ( )){**

**87**

**88 IBANOK = true;**

**89 }**

**90**

**91 }**

**92**

**93 return IBANOK;**

**94**

**95 }**

**96**

**97 //Método que añade cuentas al ArrayList que recibe por parametro**

**98 //Comprueba que no hayamos llegado al número máximo permitido**

**99 public boolean abrirCuenta(CuentaBancaria nuevaCuentaBancaria) {**

**100 listaCuentas.add(nuevaCuentaBancaria);**

**101**

**102 return true;**

**103**

**104 }**

**105**

**106 // Método que recorre e imprimelos datos de las cuentas almacenadas en cuentaBancaria**

**107 // devuelve matriz listadoCuentas**

**108 public String[] listadoCuentas() {**

**109**

**110 CuentaBancaria[] array = new CuentaBancaria[listaCuentas.size()];**

**111 listaCuentas.toArray(array);**

**112 String[] listadoCuentas = new String[array.length];**

**113**

**114 listadoCuentas = new String[array.length];**

**115 for (int i = 0; i < array.length; i++) {**

**116**

**117 listadoCuentas[i] = "IBAN: " + array[i].getIBAN() + "\t NOMBRE: " + array[i].getTitular().getNombreCliente() + " "**

**118 + array[i].getTitular().getApellidosCliente() + "\tDNI: " + array[i].getTitular().getDNI().getDNI() + "\t SALDO: "**

**119 + array[i].getSaldo();**

**120**

**121 }**

**122 return listadoCuentas;**

**123**

**124 }**

**125**

**126 //Método que comprueba la información almacenada en la matriz cuentaBancaria**

**127 //Devuelve un String informaciónCuenta con los datos que el método toString devuelve en la posición cuentaBancaria[i]**

**128**

**129 public String informacionCuenta(String IBAN) {**

**130**

**131 String informacionCuenta = null;**

**132**

**133 for (CuentaBancaria b : listaCuentas) {**

**134 if (b.getIBAN().equals(IBAN)) {**

**135 informacionCuenta = "IBAN: " + b.getIBAN() + "\t NOMBRE: " + b.getTitular().getNombreCliente() + " "**

**136 + b.getTitular().getApellidosCliente() + "\t DNI: " + b.getTitular().getDNI().getDNI() + "\t SALDO: " + b.getSaldo();**

**137 }**

**138**

**139 }**

**140**

**141 return informacionCuenta;**

**142 }**

**143**

**144 //Método que busca la posición mediante el IBAN de un elemento en el ArrayList**

**145 //Devuelve un numero entero con la posición del elemento solicitado**

**146 public int dameIndice(String IBAN) {**

**147 int indice = 0;**

**148 for (CuentaBancaria b : listaCuentas) {**

**149 if (b.getIBAN().equals(indice)) {**

**150 indice = listaCuentas.indexOf(b);**

**151 }**

**152 }**

**153**

**154 return indice;**

**155 }**

**156**

**157 //Método que borra un elemento de la posición solicitada a traves de un int**

**158 public void eliminaCuenta(int indice) {**

**159**

**160 listaCuentas.remove(indice);**

**161**

**162 }**

**163**

**164 //Método que calcula el saldo acumulado de todas las cuentas**

**165 public double calculaSaldoTotal(double comision) {**

**166 double suma = 0;**

**167 int i = 0;**

**168 double filtroComision;**

**169 for (CuentaBancaria b : listaCuentas) {**

**170**

**171 if (b instanceof CuentaCorriente) {**

**i i i i i i i i i i**

**172 if (((CuentaCorriente) listaCuentas.get(i)).getComisionMantenimiento() >= comision) {**

**173**

**174 suma += b.getSaldo();**

**175 }**

**176 }**

**177 i++;**

**178 }**

**179**

**180 return suma;**

**181 }**

**182**

**183 //Método que devuelve el número total de cuentas tipo Ahorro**

**184 public int numeroCuentasAhorro() {**

**185 int numeroCuentasAhorro = 0;**

**186 for (CuentaBancaria b : listaCuentas) {**

**187 if (b instanceof CuentaAhorro) {**

**188 numeroCuentasAhorro++;**

**189 }**

**190 }**

**191**

**192 return numeroCuentasAhorro;**

**193 }**

**194**

**195 // Metodo que ingresa cantidades en una cuenta concreta almacenada en el ArrayList listaCuentas**

**196 // devuelve un boolean infoIngreso para saber si se ha realizado correctamente o no**

**197 public boolean ingresoCuenta(String IBAN, double cantidadIngresar) {**

**198**

**199 boolean infoIngreso = false;**

**200**

**201 double ingreso = cantidadIngresar;**

**202**

**203 for (CuentaBancaria b : listaCuentas) {**

**204 if (b.getIBAN().equals(IBAN)) {**

**205**

**206 double saldo = b.getSaldo();**

**207 double saldoActual = saldo + ingreso;**

**208 b.setSaldo(saldoActual);**

**209 infoIngreso = true;**

**210**

**211 }**

**212**

**213 }**

**214**

**215 return infoIngreso;**

**216 }**

**217**

**218 // Metodo que retira cantidades en una cuenta concreta almacenada en el ArrayList listaCuentas**

**219 // devuelve un boolean infoRetirada para saber si se ha realizado correctamente o no**

**220 public boolean retiradaCuenta(String IBAN, double cantidadRetirar) {**

**221**

**222 boolean infoRetirada = false;**

**223**

**224 double retirada = cantidadRetirar;**

**225**

**226 for (CuentaBancaria b : listaCuentas) {**

**227**

**228 if (b.getIBAN().equals(IBAN)) {//Si el IBAN existe continua con lo siguiente**

**229 infoRetirada = b.retirada(retirada);**

**230 } else if (!b.getIBAN().equals(IBAN)) {**

**231 infoRetirada = false;**

**232**

**233 }**

**234**

**235 }**

**236**

**237 return infoRetirada;**

**238 }**

**239**

**240 // Metodo que informa del saldo acumulado en una cuenta concreta almacenada en el ArrayList listaCuentas**

**241 // devuelve un double saldoACtual con la cantidad actual de fondos en la cuenta**

**242 public double obtenerSaldo(String IBAN) {**

**243**

**244 double saldoActual = 0;**

**245 boolean existeIBAN = existeIBAN(IBAN);**

**246**

**247 for (CuentaBancaria b : listaCuentas) {**

**248**

**249 if (b.getIBAN().equals(IBAN)) {**

**250**

**251 saldoActual = b.getSaldo();**

**252**

**253 }**

**254**

**255 }**

**256 return saldoActual;**

**257 }**

**258**

**259 //Metodo que ordena listaCuentas de Mayor a Menor e imprime las tres primeras por pantalla**

**260 public String[] tresMayorSaldo() {**

**261**

**262 Collections.sort(listaCuentas);**

**263 CuentaBancaria[] array = new CuentaBancaria[listaCuentas.size()];**

**264 listaCuentas.toArray(array);**

**265 String[] listaTresCuentas = new String[array.length];**

**266**

**267 //Si el array es menor de 3 continua con lo siguiente:**

**268 if (array.length < 3) {**

**269 listaTresCuentas = new String[array.length];**

**270 for (int i = 0; i < array.length; i++) {**

**271**

**272 listaTresCuentas[i] = "IBAN: " + array[i].getIBAN() + "\t NOMBRE: " + array[i].getTitular().getNombreCliente() + " "**

**273 + array[i].getTitular().getApellidosCliente() + "\tDNI: " + array[i].getTitular().getDNI().getDNI() + "\t SALDO: "**

**274 + array[i].getSaldo();**

**275 }**

**276**

**277 //Si el array list es mayor o igual a 3 continua con lo siguyiente:**

**278 } else if (array.length >= 3) {**

**279 listaTresCuentas = new String[3];**

**280 for (int i = 0; i < 3; i++) {**

**281 listaTresCuentas[i] = "IBAN: " + array[i].getIBAN() + "\t NOMBRE: " + array[i].getTitular().getNombreCliente()**

**282 + " " + array[i].getTitular().getApellidosCliente() + "\t DNI: " + array[i].getTitular().getDNI().getDNI()**

**283 + "\t SALDO: " + array[i].getSaldo();**

**284 }**

**285 }**

**286 return listaTresCuentas;**

**287**

**288 }**

**289**

**290 //Metodo que crea archivo binario a partir del ArrayList listaCuentas**

**291 public void guardaArray() {**

**292**

**293 try {**

**294**

**295 FileOutputStream fichero = new FileOutputStream("datos\_mari\_ibanez.dat");//busca.crea fichero**

**296 ObjectOutputStream guardaFichero = new ObjectOutputStream(fichero);//crea flujo**

**297 guardaFichero.writeObject(listaCuentas);//escribe en el fichero el objeto que te paso**

**298 guardaFichero.close();//cierra flujo**

**299 fichero.close();**

**300 System.out.println("Fichero Guardado");**

**301**

**302 } catch (FileNotFoundException e) {**

**303 System.out.println(e.getMessage());**

**304 } catch (IOException e) {**

**305 System.out.println(e.getMessage());**

**306 }**

**307**

**308 }**

**309**

**310 //Metodo que carga el archivo binario creado a partir del ArrayList listaCuentas**

**311 public void cargaArray() {**

**312**

**313 FileInputStream fichero = null;**

**314 try {**

**315 fichero = new FileInputStream("datos\_mari\_ibanez.dat"); //busca fichero**

**316 ObjectInputStream leeFichero;**

**317**

**318 leeFichero = new ObjectInputStream(fichero); //crea flujo**

**319**

**320 listaCuentas = (ArrayList<CuentaBancaria>) leeFichero.readObject();**

**321**

**322 leeFichero.close();//cierra flujo**

**323 fichero.close();**

**324**

**325 if (fichero != null) {**

**326 System.out.println("Fichero Cargado");**

**327 }**

**328**

**329 } catch (FileNotFoundException e) {**

**330 //System.out.println(e.getMessage());**

**331 } catch (IOException e) {**

**332 System.out.println(e.getMessage());**

**333 } catch (ClassNotFoundException e) {**

**334 System.out.println(e.getMessage());**

**335 }**

**336**

**337 }**

**338**

**339 //Metodo que crea archivo texto a partir del ArrayList listaCuentas**

**340 // y muestra por pantalla el listado de clientes (IBAN y Titular)**

**341 public void listadoClientes() {**

**342**

**343 //Escribe en el fichero**

**343 //Escribe en el fichero**

**344 FileWriter fichero = null;**

**345 PrintWriter pwFichero;**

**346**

**347 try {**

**348 fichero = new FileWriter("datos\_mari\_ibanez.txt");**

**349 } catch (IOException ex) {**

**350 Logger.getLogger(ListaCuentas.class.getName()).log(Level.SEVERE, null, ex);**

**351 }**

**352 pwFichero = new PrintWriter(fichero);**

**353**

**354 pwFichero.println("----------------------------------------------------\n"**

**355 + "IBAN\t\t\tCliente\n"**

**356 + "----------------------------------------------------");**

**357**

**358 for (CuentaBancaria b : listaCuentas) {**

**359**

**360 pwFichero.println(b.getIBAN() + "\t" + b.getTitular().getNombreCliente() + " " + b.getTitular().getApellidosCliente());**

**361**

**362 }**

**363**

**364 // Numero de Cuentas Bancarias**

**365 int numCuentas = 0;**

**366 for (CuentaBancaria b : listaCuentas) {**

**367 numCuentas++;**

**368 }**

**369 pwFichero.println("----------------------------------------------------\n" + "Numero de Cuentas: " + numCuentas);**

**370**

**371 //Fecha del sistema**

**372 pwFichero.println("----------------------------------------------------");**

**373 Date fecha = new Date();**

**374 pwFichero.println("Fecha: " + fecha);**

**375 pwFichero.println("----------------------------------------------------\n");**

**376**

**377 try {**

**378 fichero.close();**

**379 } catch (IOException ex) {**

**380 Logger.getLogger(ListaCuentas.class.getName()).log(Level.SEVERE, null, ex);**

**381 }**

**382**

**383 //Lee del Fichero**

**384 try {**

**385 FileReader frFichero = new FileReader("datos\_mari\_ibanez.txt");**

**386 BufferedReader brFichero = new BufferedReader(frFichero);**

**387 String linea = "";**

**388 while (linea != null) {**

**389 linea = brFichero.readLine();**

**390 if (linea != null) {**

**391 System.out.println(linea);**

**392 }**

**393 }**

**394 } catch (FileNotFoundException ex) {**

**395 Logger.getLogger(ListaCuentas.class.getName()).log(Level.SEVERE, null, ex);**

**396 } catch (IOException ex) {**

**397 Logger.getLogger(ListaCuentas.class.getName()).log(Level.SEVERE, null, ex);**

**398 }**

**399**

**400 }**

**401**

**402 }//Fin Clase**

**Archivo: Utilidades.java**

**1 package prog09;**

**2**

**3 import java.util.\*;**

**4 import java.util.regex.Matcher;**

**5 import java.util.regex.Pattern;**

**6**

**7 /\*\***

**8 \* Ejercicio 8.1 Clase Utilidades del programa de facturación Funciones: Leer**

**9 \* int, double, String por teclado y validar DNI**

**10 \***

**11 \* @author Juan Marí Ibáñez Fecha última modificación:20.03,2022**

**12 \*/**

**13 public class Utilidades {**

**14**

**15 // Cadena con las letras posibles del DNI ordenados para el cálculo de DNI**

**16 private static final String LETRAS\_DNI = "TRWAGMYFPDXBNJZSQVHLCKE";**

**17**

**18 // Atributos de objeto para calculos del NIF**

**19 private static int numDNI;**

**20**

**21 // Función: Lee entero por teclado**

**22 // Devuelve: variable entero con entero introducido por usuario**

**23 public static int llegirSencer(String prompt) throws InputMismatchException {**

**24**

**25 Scanner teclado = new Scanner(System.in);**

**26 int entero = 0;**

**27**

**28 System.out.print(prompt);**

**29 entero = teclado.nextInt();**

**30 return entero;**

**31 }**

**32**

**33 // Función: Lee doble por teclado**

**34 // Devuelve: variable real con double introducido por usuario**

**35 public static double llegirDouble(String prompt) throws InputMismatchException {**

**36**

**37 Scanner teclado = new Scanner(System.in);**

**38 double real = 0;**

**39**

**40 System.out.print(prompt);**

**41 real = teclado.nextDouble();**

**42 return real;**

**43 }**

**44**

**45 // Función: Lee cadena de texto por teclado**

**46 // Devuelve: variable String con cadema introducida por usuario**

**47 public static String llegirString(String prompt) {**

**48**

**49 Scanner teclado = new Scanner(System.in);**

**50 String cadena;**

**51**

**52 System.out.print(prompt);**

**53 cadena = teclado.nextLine();**

**54 return cadena;**

**55 }**

**56**

**57**

**58 // Método que crea un IBAN válido a partir de una expresión regular y comprueba que sea unico**

**59 //devuelve String IBAN**

**60 public static String compruebaIBAN() {**

**61 String IBAN = "";**

**62**

**63 Pattern patronIBAN = Pattern.compile("(ES[0-9]{22})");**

**64 boolean continuaIBAN = false;**

**65**

**66 do {**

**67 IBAN = Utilidades.llegirString( //Lee por teclado el IBAN**

**68 "Introduzca IBAN: ").toUpperCase();**

**69 Matcher coincidencia = patronIBAN.matcher(IBAN);**

**70 if (IBAN.length() == 0) {//Si el valor introducido es 0 salta error**

**71 System.out.println("\n----------------------------------------------------\n"**

**72 + "ERROR!! EL IBAN NO PUEDE QUEDAR VACÍO ");**

**73 continuaIBAN = false;**

**74**

**75 }**

**76**

**77 if (!coincidencia.matches()) {//Si el valor introducido no coincide con el patron salta error**

**78 System.out.println("\n----------------------------------------------------\n"**

**79 + "ERROR!! INTRODUZCA UN IBAN VÁLIDO (ES1111111111111111111111: ");**

**80 continuaIBAN = false;**

**81 }**

**82**

**83 if (IBAN.length() != 0 & coincidencia.matches()) {// si el valor introducido es diferente de 0 continua**

**84 continuaIBAN = true;**

**85 }**

**86**

**87 } while (continuaIBAN == false);**

**88**

**89**

**90**

**91 return IBAN;**

**92 }**

**93**

**94 }//Fin Clase**

**Archivo: Menu.java**

**1 package prog09;**

**2**

**3 import java.util.InputMismatchException;**

**4 import java.util.Scanner;**

**5**

**6 /\*\***

**7 \* Ejercicio 9 Programa Menú del programa para la interaccion con el usuario**

**8 \***

**9 \* @author Juan Marí Ibáñez Fecha última modificación: 20.03.2022**

**10 \*/**

**11 public class Menu {**

**12**

**13 public static int menu() {**

**14 int numero = 0;**

**15**

**16 System.out.println("\n----------------------------------------------------");**

**17 System.out.println(" Menú de Opciones ");**

**18 System.out.println("----------------------------------------------------");**

**19 System.out.println("1. Abrir una nueva cuenta.");**

**20 System.out.println("2. Ver listado de cuentas disponibles.");**

**21 System.out.println("3. Obtener datos de una cuenta.");**

**22 System.out.println("4. Realizar ingreso en una cuenta.");**

**23 System.out.println("5. Retirar efectivo de una cuenta.");**

**24 System.out.println("6. Consultar el saldo actual de una cuenta.");**

**25 System.out.println("7. Eliminar Cuenta Bancaria.");**

**26 System.out.println("8. Mostrar número de Cuentas Ahorro.");**

**27 System.out.println("9. Mostrar saldo acumulado de Cuentas Corrientes.");**

**28 System.out.println("10. Mostrar 3 primeras Cuentas con mayor saldo.");**

**29 System.out.println("11. Listado de Clientes.");**

**30 System.out.println("12. Salir de la aplicación.");**

**31 System.out.println("----------------------------------------------------");**

**32 System.out.print("Introduzca una opción: ");**

**33**

**34 try {**

**35 Scanner sc = new Scanner(System.in);**

**36 numero = sc.nextInt();**

**37 } catch (InputMismatchException e) {**

**38**

**39 }**

**40**

**41 return numero;**

**42 }**

**43**

**44 }//Fin Clase**

**Archivo Dni.java**

**1 package prog09;**

**2**

**3 import java.io.Serializable;**

**4 import java.util.regex.Matcher;**

**5 import java.util.regex.Pattern;**

**6**

**7 /\*\***

**8 \* Ejercicio Programa 9**

**9 \***

**10 \* @author Juan Marí Ibáñez Fecha última modificación:20.03,2022**

**11 \*/**

**12 public class Dni implements Serializable{**

**13**

**14 // Cadena con las letras posibles del DNI ordenados para el cálculo de DNI**

**15 private static final String LETRAS\_DNI = "TRWAGMYFPDXBNJZSQVHLCKE";**

**16**

**17 // Atributos de objeto para calculos del NIF**

**18 private static int numDNI;**

**19**

**20 private String Dni;**

**21**

**22 //Contructor Dni**

**23 public Dni(String DNI) {**

**24**

**25**

**26 this.Dni = DNI;**

**27**

**28 }**

**29**

**30 // Método setter**

**31 public void setDNI(String DNI) {**

**32**

**33**

**34 this.Dni = DNI;**

**35**

**36 }**

**37**

**38 //Método getter**

**39 public String getDNI() {**

**40 return this.Dni;**

**41 }**

**42**

**43 // Función: Envia numero de DNI almacenado en numDNI**

**44 // Devuelve: variable int numDNI con el numero del DNI almacenado**

**45 public static int obtenerDNI() {**

**46 return numDNI;**

**47 }**

**48**

**49 // Función: establece el numero de DNI en la variable numDNI**

**50 // realiza comprobaciones para numero de DNI valido**

**51 public static void establecer(String nif) throws Exception {**

**52 if (validarNIF(nif)) { // Valor válido: lo almacenamos**

**53 numDNI = extraerNumeroNIF(nif);**

**54 } else { // Valor inválido: lanzamos una excepción**

**55 throw new Exception("----------------------------------------------------\n"**

**56 + " ERROR: EL DNI NO ES VALIDO\n"**

**57 + "----------------------------------------------------");**

**58 }**

**59**

**60 }**

**61**

**62 public static void establecer(int dni) throws Exception {**

**63**

**64 // Comprobación de rangos**

**65 if (dni > 999999 && dni < 99999999) {**

**66 numDNI = dni; // Valor válido: lo almacenamos**

**67 } else { // Valor inválido: lanzamos una excepción**

**68 throw new Exception("DNI inválido: " + String.valueOf(dni));**

**69 }**

**70 }**

**71**

**72 // Función: Calcula letra DNI a partir de matriz de letras guardadas en**

**73 // constante String LETRAS\_DNI**

**74 // Devuelve: variable char con letra DNI calculada**

**75 private static char calcularLetraNIF(int dni) {**

**76 char letra;**

**77**

**78 // Cálculo de la letra NIF**

**79 letra = LETRAS\_DNI.charAt(dni % 23);**

**80**

**81 // Devolución de la letra NIF**

**82 return letra;**

**83 }**

**84**

**85 // Función: extrae la letra del NIF del DNI introducido por usuario**

**86 // Devuelve: de vuelve variable char letra con letra extraida de DNI**

**87 private static char extraerLetraNIF(String nif) {**

**88 char letra = nif.charAt(nif.length() - 1);**

**89**

**90 return letra;**

**91 }**

**92**

**93 // Función: Extrae numero de NIF del DNI introducido por el usuario**

**94 // Devuelve: variable int numero con el numero extraido de DNI**

**95 private static int extraerNumeroNIF(String nif) {**

**96 int numero = Integer.parseInt(nif.substring(0, nif.length() - 1));**

**97 return numero;**

**98 }**

**99**

**100 // Función: comprueba que el DNI introducido por el usuario es valido**

**101 // Devuelve: variable boolean para saber si es valdio o no el DNI introducido**

**102 public static boolean validarNIF(String nif) {**

**103 boolean valido = true; // Suponemos el NIF válido mientras no se encuentre algún fallo**

**104 char letra\_calculada;**

**105 char letra\_leida;**

**106 int dni\_leido;**

**107**

**108 if (nif == null) { // El parámetro debe ser un objeto no vacío**

**109 valido = false;**

**110**

**111 // La cadena debe estar entre 8(7+1) y 9(8+1) caracteres**

**112 } else if (nif.length() < 8 || nif.length() > 9) {**

**113 valido = false;**

**114 } else {**

**115 // Extraemos la letra de NIF (letra)**

**116 letra\_leida = extraerLetraNIF(nif);**

**117 //transforma letra a mayusculas para que el usuario pueda poner la**

**118 //letra tanto en minusculas como en mayusculas**

**119 h l t l id M l Ch t t U C (l t l id )**

**119 char letra\_leida\_Mayusculas = Character.toUpperCase(letra\_leida);**

**120 // Extraemos el número de DNI (int)**

**121 dni\_leido = extraerNumeroNIF(nif);**

**122 // Calculamos la letra de NIF a partir del número extraído**

**123 letra\_calculada = calcularLetraNIF(dni\_leido);**

**124 // Comparamos la letra extraída con la calculada**

**125 if (letra\_leida\_Mayusculas == letra\_calculada) {**

**126 // Todas las comprobaciones han resultado válidas. El NIF es válido.**

**127 valido = true;**

**128 } else {**

**129 valido = false;**

**130 }**

**131 }**

**132**

**133 return valido;**

**134 }**

**135**

**136 // Función: Pide String para introducir un DNI válido de cliente**

**137 // Devuelve: Devuelve String DNI con el DNI válido introducido**

**138 public static String dameDNI() {**

**139 String DNI = "";**

**140 boolean DNIValido = false;**

**141 boolean continuaDNI = false;**

**142**

**143 do {**

**144 DNI = Utilidades.llegirString("Introduzca DNI: ").toUpperCase();**

**145**

**146 DNIValido = validarNIF(DNI);**

**147**

**148 if (DNIValido == true) {**

**149**

**150 continuaDNI = true;**

**151 }**

**152**

**153 if (DNIValido == false) {**

**154 System.out.println("\n----------------------------------------------------\n"**

**155 + "ERROR!! INTRODUZCA UN DNI VÁLIDO (11111111A): ");**

**156 continuaDNI = false;**

**157 }**

**158**

**159 } while (continuaDNI == false);**

**160**

**161 return DNI;**

**162 }**

**163**

**164 }**

**Archivo: Persona.java**

**1 package prog09;**

**2**

**3 import java.io.Serializable;**

**4**

**5 /\*\***

**6 \*Ejercicio 9**

**7 \*Programa Administración de cuentas bancarias**

**8 \*Estudio: Composición, Herencia, Clases Abstractas, Interfaces.**

**9 \*@author Juan Marí Ibáñez**

**10 \*Fecha última modificación: 20.03,2022**

**11 \*/**

**12**

**13 public class Persona implements Serializable{**

**14**

**15**

**16 private String NombreCliente;**

**17 private String ApellidosCliente;**

**18 private Dni DNI;**

**19**

**20**

**21**

**22 //Métodos constructores**

**23**

**24 //Contructor por defecto**

**25 public Persona(){**

**26**

**27 }**

**28**

**29 //Contructor con parámetros**

**30 public Persona(String NombreCliente, String ApellidosCliente, Dni DNI) throws Exception {**

**31 if (Dni.validarNIF(DNI.getDNI())==false){**

**32 throw new Exception("\n----------------------------------------------------\n"**

**33 + "DNI INTRODUCIDO NO VÁLIDO. NO SE HA GUARDADO EL VALOR ");**

**34 }**

**35 this.NombreCliente = NombreCliente;**

**36 this.ApellidosCliente = ApellidosCliente;**

**37 this.DNI = DNI;**

**38 }**

**39**

**40 //Métodos Setters**

**41**

**42 public void setNombreCliente(String NombreCliente) {**

**43 this.NombreCliente = NombreCliente;**

**44 }**

**45**

**46 public void setApellidosCliente(String ApellidosCliente) {**

**47 this.ApellidosCliente = ApellidosCliente;**

**48 }**

**49**

**50 public void setDNI(Dni DNI) throws Exception{**

**51 if (Dni.validarNIF(DNI.getDNI())==false){**

**52 throw new Exception("\n----------------------------------------------------\n"**

**53 + "DNI INTRODUCIDO NO VÁLIDO. NO SE HA GUARDADO EL VALOR ");**

**54 }**

**55 this.DNI = DNI;**

**56 }**

**57**

**58 //Métodos Getters**

**59**

**60 public String getNombreCliente() {**

**61 return NombreCliente;**

**62 }**

**63**

**64 public String getApellidosCliente() {**

**65 return ApellidosCliente;**

**66 }**

**67**

**68 public Dni getDNI() {**

**69 return DNI;**

**70 }**

**71**

**72 //Método toString**

**73 @Override**

**74 public String toString() {**

**75 return "Titular: " + NombreCliente + " "+ApellidosCliente + "\t DNI: " + DNI ;**

**76 }**

**77**

**78**

**79**

**80 }//Fin Clase**

**Archivo: CuentaBancaria.java**

**1 package prog09;**

**2**

**3 import java.util.\*;**

**4 import java.io.\*;**

**5 import java.util.regex.Matcher;**

**6 import java.util.regex.Pattern;**

**7**

**8 /\*\***

**9 \* Ejercicio 9 Programa Administración de cuentas bancarias Estudio:**

**10 \* Composición, Herencia, Clases Abstractas, Interfaces.**

**11 \***

**12 \* @author Juan Marí Ibáñez Fecha última modificación: 20.03,2022**

**13 \*/**

**14 public abstract class CuentaBancaria implements Comparable<CuentaBancaria>, Serializable {**

**15**

**16 protected Persona titular;**

**17 protected double saldo;**

**18 protected String IBAN;**

**19**

**20 //Métodos constructores**

**21 //Método constructor por defecto**

**22 public CuentaBancaria() {**

**23**

**24 }**

**25**

**26 //Método constructor con parámetros**

**27 public CuentaBancaria(Persona titular, double saldo, String IBAN) throws Exception {**

**28 if (validaIBAN(IBAN) == false) {**

**29 throw new Exception("\n----------------------------------------------------\n"**

**30 + "IBAN INTRODUCIDO NO VÁLIDO. NO SE HA GUARDADO EL VALOR ");**

**31 }**

**32**

**33 this.titular = new Persona(titular.getNombreCliente(),titular.getApellidosCliente(), titular.getDNI());**

**34 this.saldo = saldo;**

**35 this.IBAN = IBAN;**

**36**

**37 }**

**38**

**39**

**40 //Métodos Setter**

**41 public void setTitular(Persona titular)throws Exception {**

**42 if (validaIBAN(IBAN) == false) {**

**43 throw new Exception("\n----------------------------------------------------\n"**

**44 + "IBAN INTRODUCIDO NO VÁLIDO. NO SE HA GUARDADO EL VALOR ");**

**45 }**

**46**

**47 this.titular = new Persona(titular.getNombreCliente(),titular.getApellidosCliente(), titular.getDNI());**

**48**

**49 }**

**50**

**51 public void setSaldo(double saldo) {**

**52 this.saldo = saldo;**

**53 }**

**54**

**55 public void setIBAN(String IBAN) throws Exception {**

**56 if (validaIBAN(IBAN) == false) {**

**57 throw new Exception("\n----------------------------------------------------\n"**

**58 + "IBAN INTRODUCIDO NO VÁLIDO. NO SE HA GUARDADO EL VALOR ");**

**59 }**

**60**

**61 this.IBAN = IBAN;**

**62 }**

**63**

**64 //Métodos Getter**

**65 public Persona getTitular() {**

**66 Persona titularCopia=null;**

**67 try{**

**68 titularCopia=new Persona(titular.getNombreCliente(),titular.getApellidosCliente(), titular.getDNI());**

**69 }catch (Exception e){**

**69 }catch (Exception e){**

**70 System.out.println(e.getMessage());**

**71 }**

**72 return titularCopia;**

**73 }**

**74**

**75 public double getSaldo() {**

**76 return saldo;**

**77 }**

**78**

**79 public String getIBAN() {**

**80 return IBAN;**

**81 }**

**82**

**83 //Metodo recibe IBAN y valida patrón**

**84 //Devuelve String IBANOk con IBAN validado**

**85 public boolean validaIBAN(String IBAN) {**

**86 Pattern patronIBAN = Pattern.compile("(ES[0-9]{22})");**

**87 Matcher coincidencia = patronIBAN.matcher(IBAN);**

**88 boolean IBANOk = false;**

**89 if (coincidencia.matches()) {**

**90 IBANOk = true;**

**91 } else if (!coincidencia.matches()) {**

**92 IBANOk = false;**

**93 }**

**94 return IBANOk;**

**95 }**

**96**

**97 //Método Abstracto Retirada**

**98 public abstract boolean retirada(double importe);**

**99**

**100 //Método toString**

**101 @Override**

**102 public String toString() {**

**103 return "\t titular: " + titular + "\t saldo: " + saldo;**

**104 }**

**105**

**106 //Metodo Comparador**

**107 @Override**

**108 public int compareTo(CuentaBancaria b) {**

**109**

**110 if (this.getSaldo() < b.getSaldo()) {**

**111 return 1;**

**112 } else if (this.getSaldo() > b.getSaldo()) {**

**113 return -1;**

**114 } else {**

**115 return 0;**

**116**

**117 }**

**118 }**

**119 }//Fin Clase**

**Archivo: CuentaCorriente.java**

**1 package prog09;**

**2**

**3 /\*\***

**4 \* Ejercicio 9 Programa Administración de cuentas bancarias Estudio:**

**5 \* Composición, Herencia, Clases Abstractas, Interfaces. Clase que Cuenta**

**6 \* Corriente que hereda de CuentaBancaria**

**7 \***

**8 \* @author Juan Marí Ibáñez Fecha última modificación: 20.03,2022**

**9 \*/**

**10 public class CuentaCorriente extends CuentaBancaria {**

**11**

**12 private double comisionMantenimiento;**

**13 private double tipoInteresDescubierto;**

**14 private double maximoDescubierto;**

**15**

**16 //Métodos constructores**

**17 //Método constructor por defecto**

**18 public CuentaCorriente() {**

**19**

**20 }**

**21**

**22 //Método constructor subclase**

**23 public CuentaCorriente(double comisionMantenimiento,**

**24 double tipoInteresDescubierto, double maximoDescubierto) {**

**25 this.comisionMantenimiento = comisionMantenimiento;**

**26 this.tipoInteresDescubierto = tipoInteresDescubierto;**

**27 this.maximoDescubierto = maximoDescubierto;**

**28 }**

**29**

**30 //Método constructor superclase**

**31 public CuentaCorriente(Persona titular, double saldo, String IBAN,**

**32 double comisionMantenimiento, double tipoInteresDescubierto,**

**33 double maximoDescubierto) throws Exception {**

**34**

**35 super(titular, saldo, IBAN);**

**36**

**37 this.comisionMantenimiento = comisionMantenimiento;**

**38 this.tipoInteresDescubierto = tipoInteresDescubierto;**

**39 this.maximoDescubierto = maximoDescubierto;**

**40 }**

**41**

**42 //Métodos Setter**

**43 public void setComisionMantenimiento(double comisionMantenimiento) {**

**44 this.comisionMantenimiento = comisionMantenimiento;**

**45 }**

**46**

**47 public void setTipoInteresDescubierto(double tipoInteresDescubierto) {**

**48 this.tipoInteresDescubierto = tipoInteresDescubierto;**

**49 }**

**50**

**51 public void setMaximoDescubierto(double maximoDescubierto) {**

**52 this.maximoDescubierto = maximoDescubierto;**

**53 }**

**54**

**55 //Métodos Getter**

**56 public double getComisionMantenimiento() {**

**57 return comisionMantenimiento;**

**58 }**

**59**

**60 public double getTipoInteresDescubierto() {**

**61 return tipoInteresDescubierto;**

**62 }**

**63**

**64 public double getMaximoDescubierto() {**

**65 return maximoDescubierto;**

**66 }**

**67**

**68 //Método Retirada**

**69 @Override**

**70 public boolean retirada(double importe) {**

**71 boolean retiradaOK = false;**

**72 double retirada = importe;**

**73**

**74 if (this.getSaldo() - importe >= -maximoDescubierto) {**

**75 this.setSaldo(this.getSaldo() - importe);**

**76 retiradaOK = true;**

**77 }**

**78 else if (this.getSaldo() - importe < - maximoDescubierto) {**

**79 System.out.println("\n----------------------------------------------------\n"**

**80 + "LA RETIRADA EN LA CUENTA ASOCIADA AL IBAN:\n "**

**81 + this.getIBAN() + "\n"**

**82 + "NO SE PUEDE REALIZAR. SUPERA EL MÁXIMNO DESCUBIERTO");**

**83**

**84 retiradaOK = false;**

**85 }**

**86**

**87 return retiradaOK;**

**88**

**89 }**

**90**

**91 //Método toString**

**92 @Override**

**93 public String toString() {**

**94 return super.toString()**

**95 + "\t comisionMantenimiento: " + comisionMantenimiento**

**96 + "\t tipoInteresDescubierto: " + tipoInteresDescubierto**

**97 + "\t maximoDescubierto: " + maximoDescubierto;**

**98 //añadir atributos cuenta bancaria**

**99 }**

**100**

**101 }//Fin Clase**

**Archivo:CuentaAhorro.java**

**1 package prog09;**

**2**

**3 /\*\***

**4 \* Ejercicio 9 Programa Administración de cuentas bancarias Estudio:**

**5 \* Composición, Herencia, Clases Abstractas, Interfaces.**

**6 \* Clase que hereda de CuentaBancaria**

**7 \* @author Juan Marí Ibáñez Fecha última modificación: 20.03,2022**

**8 \*/**

**9 public class CuentaAhorro extends CuentaBancaria {**

**10**

**11 private double tipoInteresAnual;**

**12**

**13 //Métodos Constructores**

**14 //Método Constructor por defecto**

**15 public CuentaAhorro() {**

**16**

**17 }**

**18**

**19 //Método construcor con parámetros subclase**

**20 public CuentaAhorro(double tipoInteresAnual) {**

**21 this.tipoInteresAnual = tipoInteresAnual;**

**22 }**

**23**

**24 //Método construcor con parámetros superclase**

**25 public CuentaAhorro(Persona titular, double saldo, String IBAN, double tipoInteresAnual) throws Exception {**

**26 super(titular, saldo, IBAN);**

**27 this.tipoInteresAnual = tipoInteresAnual;**

**28 }**

**29**

**30 //Método Setter**

**31 public void setTipoInteresAnual(double tipoInteresAnual) {**

**32 this.tipoInteresAnual = tipoInteresAnual;**

**33 }**

**34**

**35 //Método Getter**

**36 public double getTipoInteresAnual() {**

**37 return tipoInteresAnual;**

**38 }**

**39**

**40 //Método Retirada**

**41 @Override**

**42 public boolean retirada(double importe) {**

**43 boolean retiradaOK=false;**

**44 double retirada = importe;**

**45**

**46 if(this.getSaldo()-importe>=0){**

**47 this.setSaldo(this.getSaldo()-importe);**

**48 retiradaOK=true;**

**49 }else if (this.getSaldo()-importe<0){**

**50 System.out.println("\n----------------------------------------------------\n"**

**51 + "LA RETIRADA EN LA CUENTA ASOCIADA AL IBAN:\n "**

**52 + this.getIBAN() + "\n"**

**53 + "NO SE PUEDE REALIZAR POR FALTA DE FONDOS EN LA CUENTA");**

**54 retiradaOK=false;**

**55 }**

**56**

**57 return retiradaOK;**

**58**

**59 }**

**60**

**61 //Método to String**

**62 @Override**

**63 public String toString() {**

**64 return super.toString()+ "\t Tipo de interes anual: " + tipoInteresAnual;**

**65 //añadir atributos cuenta bancaria**

**66 }**

**67**

**68**

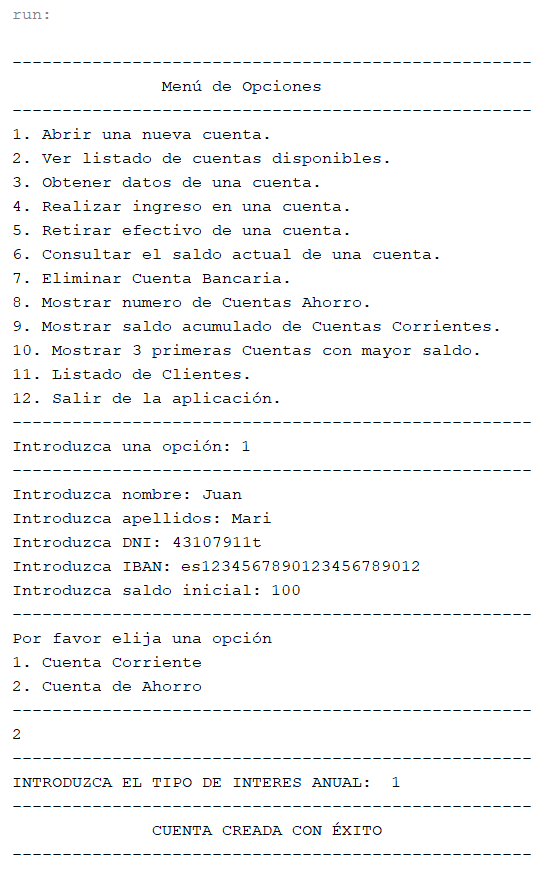
**68**

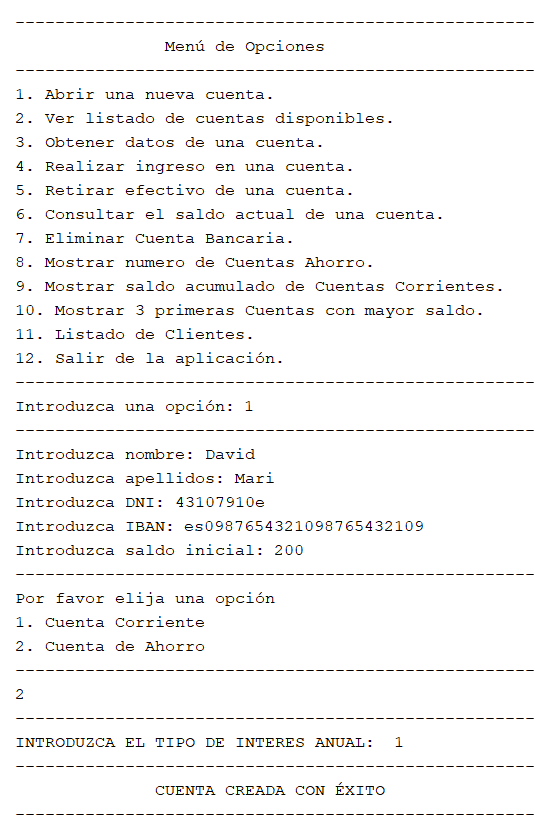
**69**

**70 }//Fin Clase**

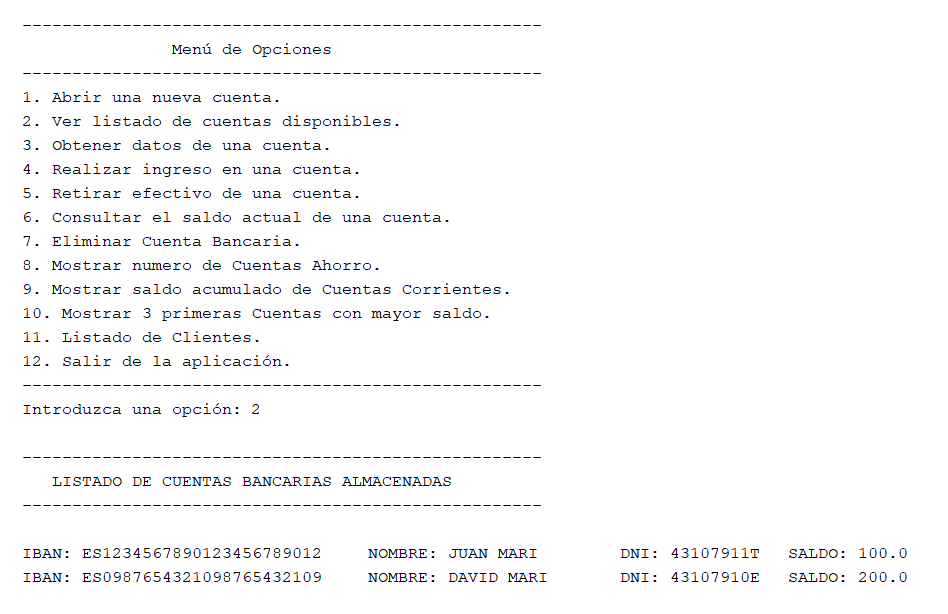
**Capturas de pantalla del test de pruebas de la ejecución de la nueva implementación, funcionamiento general y comentarios.**

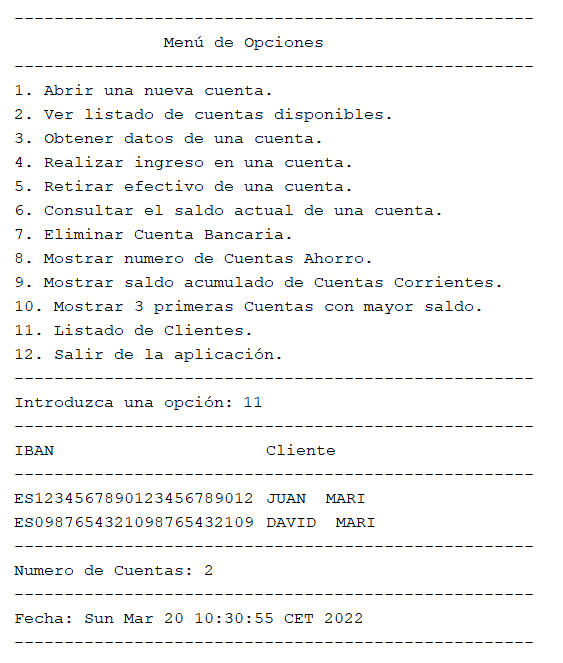
**1. Introducción de Datos**

****

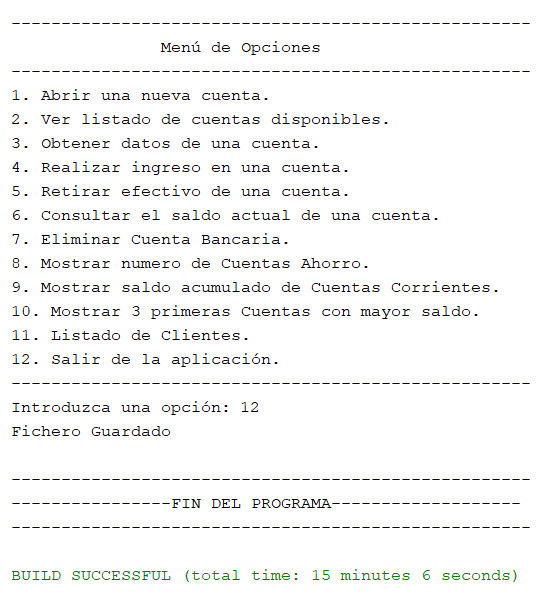
****

**2. Visualización de Datos introducidos**

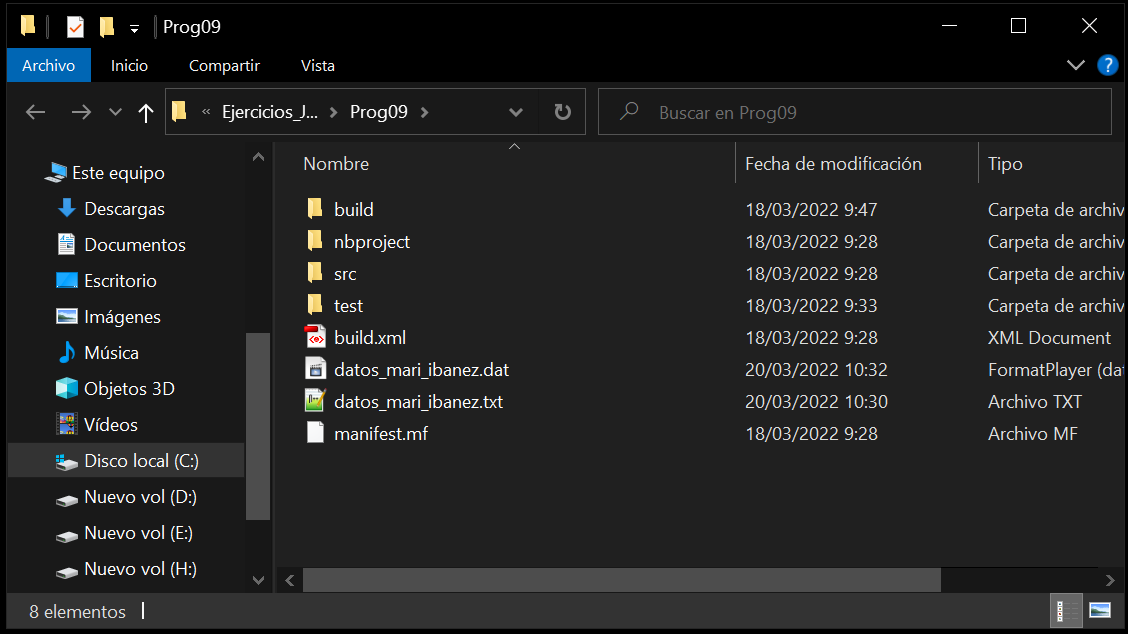
****

**3. Opción 11 Crea Informe en formato texto plano**

**4. Opción 12. Guarda objeto listaCuentas en archivo**

****

**5. Comprobación de que se han generado los archivos dónde se guardan los datos.**

****

**6. Comprobación de que se cargan los datos del fichero con el objeto listaCuentas**

****