# EJERCICIOS DE INTERNET

<https://www.discoduroderoer.es/ejercicios-propuestos-y-resueltos-programacion-orientado-a-objetos-java/>

## CUENTA BANCARIA

Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular y cantidad (puede tener decimales).

El titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumpla lo anterior.

Crea sus métodos get, set y toString.

Tendrá dos métodos especiales:

* ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
* retirar(double cantidad): se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

**package** unidad5;

**import** java.util.Scanner;

/\*\*

\* CLASE CUENTA

\* **@author** vero

\*

\*/

**public** **class** Cuenta {

// atributos ////

**private** String titular;

**private** Double cantidad;

// constructores ////

**public** Cuenta(String titular, **double** cantidad) {

**this**.titular = titular;

**this**.cantidad = cantidad;

}

**public** Cuenta(String titular) {

**this**.titular = titular;

**this**.cantidad = (**double**) 0;

}

// get ////

**public** String getTitular() {

**return** titular;

}

**public** Double getCantidad() {

**return** cantidad;

}

// set ////

**public** **void** setTitular(String titular) {

**this**.titular = titular;

}

**public** **void** setCantidad(**double** cantidad) {

**this**.cantidad = cantidad;

}

// toString ////

@Override

**public** String toString() {

**return** "Titular cuenta: " + titular + ". Cantidad: " + cantidad + "€";

}

// métodos ////

**public** **void** ingresar(**double** cantidad) {

**if** (cantidad > 0 ) {

**this**.cantidad += cantidad;

}

}

**public** **void** retirar(**double** cantidad) {

**if** (**this**.cantidad - Math.*abs*(cantidad) < 0) {

**this**.cantidad = (**double**) 0;

} **else** {

**this**.cantidad -= Math.*abs*(cantidad);

}

}

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

// **TODO** Auto-generated method stub

Cuenta cuenta\_1 = **new** Cuenta("Paco");

Cuenta cuenta\_2 = **new** Cuenta("Pepe", 300);

Integer importe;

System.***out***.println (cuenta\_1 + "\n" + cuenta\_2 + "\n");

System.***out***.println ("Nombre de la cuenta:");

Scanner input = **new** Scanner (System.***in***); // Creación de objeto Scanner

String nombre = input.nextLine (); // Almacenamos el texto captado por teclado

**if** (cuenta\_1.titular.equals(nombre)) { // Uso if-else porque switch solo permite usar expresiones constantes. Uso equals en lugar de == para que funcione

System.***out***.println ("Saldo en cuenta: " + cuenta\_1.cantidad + "€");

System.***out***.println ("¿Cuánto dinero desea ingresar?");

importe = input.nextInt (); // Almacenamos el texto captado por teclado

cuenta\_1.ingresar(importe);

System.***out***.println ("Saldo en cuenta: " + cuenta\_1.cantidad);

System.***out***.println ("¿Cuánto dinero desea retirar?");

importe = input.nextInt ();

cuenta\_1.retirar(importe);

System.***out***.println ("Saldo en cuenta: " + cuenta\_1.cantidad);

System.***out***.println ("INFO CUENTA");

System.***out***.println(cuenta\_1);

} **else** **if** (cuenta\_2.titular.equals(nombre)) { // Uso equals en lugar de == para que funcione con el atributo del objeto

System.***out***.println ("Saldo en cuenta: " + cuenta\_2.cantidad + "€");

System.***out***.println ("¿Cuánto dinero desea ingresar?");

importe = input.nextInt (); // Almacenamos el texto captado por teclado

cuenta\_2.ingresar(importe);

System.***out***.println ("Saldo en cuenta: " + cuenta\_2.cantidad);

System.***out***.println ("¿Cuánto dinero desea retirar?");

importe = input.nextInt ();

cuenta\_2.retirar(importe);

System.***out***.println ("Saldo en cuenta: " + cuenta\_2.cantidad);

System.***out***.println ("INFO CUENTA");

System.***out***.println(cuenta\_2);

} **else** {

System.***out***.println ("ERROR, no existe la cuenta.");

}

}

}

# EJERCICIOS 2EVAL U5

## Ejercicio 1: Animal

Definir una clase llamada Animal según las especificaciones siguientes:

* + • **Atributos de instancia:**
  + o nombre, de tipo String
  + o fecha, de tipo LocalDate
  + • **Métodos de instancia:** o Constructor que inicialice el estado del objeto con los valores que reciba a través de sus parámetros formales.
  + o Constructor que inicialice el estado del objeto con un nombre que reciba a través de un parámetro formal y con la fecha actual.
  + o Métodos get y set.
  + o Redefinir el método toString para que retorne la cadena:

"Nombre: nombre – Edad: edad"

Escribir un programa que ponga a prueba la clase Animal utilizando todos sus constructores y métodos.

### SOLUCIÓN PROFE

|  |
| --- |
| package unidad5; |
|  |  |
|  | import java.time.LocalDate; |
|  |  |
|  | public class Animal { |
|  |  |
|  | private String nombre; |
|  | private LocalDate fecha; |
|  |  |
|  | public Animal(String nombre, LocalDate fecha) { |
|  | this.nombre = nombre; |
|  | this.fecha = fecha; |
|  | } |
|  |  |
|  | public Animal(String nombre) { |
|  | this.nombre = nombre; |
|  | fecha = LocalDate.now(); |
|  | } |
|  |  |
|  | public String getNombre() { |
|  | return nombre; |
|  | } |
|  |  |
|  | public void setNombre(String nombre) { |
|  | this.nombre = nombre; |
|  | } |
|  |  |
|  | public LocalDate getFecha() { |
|  | return fecha; |
|  | } |
|  |  |
|  | public void setFecha(LocalDate fecha) { |
|  | this.fecha = fecha; |
|  | } |
|  |  |
|  | @Override |
|  | public String toString() { |
|  | return "Animal [nombre=" + nombre + ", fecha=" + fecha + "]"; |
|  | } |
|  |  |
|  | } |

### SOLUCIÓN VERO ACTUALIZADA

**package** pruebas;

**import** java.time.LocalDate;

**import** java.time.Period;

**import** java.time.format.DateTimeFormatter;

**public** **class** Animal {

// atributos

String nombre;

LocalDate fecha;

**int** edad;

**static** DateTimeFormatter *fmt* = DateTimeFormatter.*ofPattern*("dd-MM-yyyy");

// metodos

**public** Animal(String nombre) {

**this**.nombre = nombre;

**this**.fecha = LocalDate.*now*();

**this**.edad = getEdad();

}

**public** Animal(String nombre, LocalDate fecha) {

**this**.nombre = nombre;

**this**.fecha = fecha;

**this**.edad = getEdad();

}

**public** Animal(String nombre, String fecha) {

**this**.nombre = nombre;

**this**.fecha = LocalDate.*parse*(fecha, *fmt*);

**this**.edad = getEdad();

}

**public** String getNombre() {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** String getFecha() {

**return** **this**.fecha.format(*fmt*);

}

**public** **int** getEdad() {

LocalDate hoy = LocalDate.*now*();

Period periodo = Period.*between*(**this**.fecha, hoy);

**this**.edad = periodo.getYears();

**return** **this**.edad;

}

**public** **void** setFecha(LocalDate fecha) {

**this**.fecha = fecha;

}

**public** **void** setFecha(String fecha) {

**this**.fecha = LocalDate.*parse*(fecha, *fmt*);

}

@Override

**public** String toString() {

**return** "Nombre: " + **this**.nombre + " - Fecha nac.: " + **this**.fecha.format(*fmt*) + " - Edad: " + **this**.edad;

}

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

// **TODO** Auto-generated method stub

Animal anim1 = **new** Animal("Bichin");

System.***out***.println(anim1);

Animal anim2 = **new** Animal("Tukon","06-02-2016");

System.***out***.println(anim2);

anim1.setFecha("01-01-2018");

System.***out***.println(anim1.getFecha());

System.***out***.println(anim1);

}

}

Nombre: Bichin - Fecha nac.: 15-03-2021 - Edad: 0

Nombre: Tukon - Fecha nac.: 06-02-2016 - Edad: 5

### SOLUCIÓN VERO

**package** unidad5;

**import** java.time.LocalDate;

**import** java.util.Scanner;

**public** **class** Animal {

// atributos //////////////////

**private** String nombre;

**private** LocalDate fecha;

// métodos ///////////////

// constructor

// opc1: se pasan los dos parámetros

**public** Animal(String nombre, LocalDate fecha) { // LocalDate se declara con LocalDate.parse("aaaa-mm-dd") o LocalDate.of(aaaa,mm,dd)

**this**.nombre = nombre;

**this**.fecha = fecha;

}

// opc2: solo se pasa nombre

**public** Animal(String nombre) {

**this**.nombre = nombre;

fecha = LocalDate.*now*();

}

// getter

**public** String getNombre() {

**return** nombre;

}

**public** LocalDate getFecha() {

**return** fecha;

}

// setter

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** **void** setFecha(LocalDate fecha) {

**this**.fecha = fecha;

}

**public** **void** setFecha(String fecha) {

**this**.fecha = LocalDate.*parse*(fecha);

}

// toString

@Override

**public** String toString() {

**return** "Animal: Nombre=" + nombre + " - Fecha=" + fecha;

}

/\*\*

\* EJECUCIÓN

\* **@param** args

\*/

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

Animal animal1 = **new** Animal("Tuko",LocalDate.*parse*("2016-02-06")); // Crea obj con constructor. LocalDate se declara con LocalDate.parse("aaaa-mm-dd") o LocalDate.of(aaaa,mm,dd)

Animal animal2 = **new** Animal("Tikaa"); // Crea obj con constructor solo con nombre

Scanner input=**new** Scanner (System.***in***);

System.***out***.println("Dar de alta animal:\n¿Nombre?");

Animal animal3 = **new** Animal(input.nextLine()); // Se crea obj con constructor solo con nombre

System.***out***.println("¿Fecha nacimiento? (aaaa-mm-dd)");

animal3.setFecha(input.nextLine()); // Setea fecha

System.***out***.println("Lista de animales:");

System.***out***.println(animal1);

System.***out***.println(animal2);

System.***out***.println(animal3);

System.***out***.println("Corregir nombre de "+ animal2.nombre); // Modifica nombre

animal2.setNombre(input.nextLine());

System.***out***.println("Nombre modificado: "+ animal2.getNombre()); // Devuelve nombre

System.***out***.println("¿Fecha nacimiento "+ animal2.nombre + "? (aaaa-mm-dd)"); // Setea fecha

animal2.setFecha(input.nextLine());

System.***out***.println("Fecha establecida: "+ animal2.getFecha()); // Devuelve fecha

System.***out***.println(animal2); // Devuelve toString

input.close();

}

}

## Ejercicio 2: Hucha

Definir una clase llamada Hucha según las especificaciones siguientes: • La hucha solo admite monedas de uno y dos euros y billetes de 5, 10, 20 y 50 euros. • La hucha podrá estar abierta o cerrada. Para abrir o cerrar la hucha se requiere una contraseña. • La contraseña se podrá cambiar en cualquier momento. • La hucha podrá proporcionar información acerca de la cantidad de monedas o billetes que almacena de cada clase, así como el valor total en euros. • Se podrán retirar de la hucha monedas o billetes indicando el tipo y la cantidad de los mismos que se desea retirar. Si la cantidad solicitada supera la cantidad almacenada se retirará únicamente la cantidad disponible. • Se podrá retirar de la hucha una cantidad de euros específica minimizando el número de monedas y billetes que se retirarán. • Se podrá construir una hucha vacía, una hucha con una cantidad específica de monedas y billetes de cada tipo o una hucha con una cantidad inicial de euros que se desglosará utilizando la mínima cantidad de monedas y billetes. Escribir un programa que ponga a prueba la clase Hucha utilizando todos sus constructores y métodos

### SOLUCION VERO ACTUALIZADA (a medias)

**package** pruebasHucha;

**public** **class** Hucha {

**private** **boolean** abierta = **true**;

**private** String clave;

**private** **int**[][] contenido = {{50,0},{20,0},{10,0},{5,0},{2,0},{1,0}};

**private** **int** total;

**public** Hucha() {

**this**.total = 0;

}

**public** Hucha(**int** importe) {

**this**.total = importe;

reparte(importe);

}

**public** **void** setImporte(**int** importe) {

**this**.total = importe;

reparte(importe);

}

@Override

**public** String toString() {

String msg = "Hucha con " + total + " euros.\n";

**for** (**int**[] a: contenido) {

**if** (a[1] != 0) {

msg += a[1];

**if** (a[0]==2 || a[0]==1) {

msg += " monedas de ";

} **else** {

msg += " billetes de ";

}

msg += a[0] + " euros\n";

}

}

**return** msg;

}

**public** String reparte(**int** importe) {

String msg = "";

**for** (**int**[] a: contenido) {

**int** uds = (**int**) importe / a[0];

**if** (uds >= 1 ) {

a[1] = uds;

importe = importe % a[0];

msg += uds + " unidades de " + a[0] + " euros\n";

}

}

msg += "Saldo: " + **this**.total;

**return** msg;

}

**public** **void** retira(**int** importe) {

**if** (importe > **this**.total) {

System.***out***.println("No tiene suficiente saldo");

} **else** {

System.***out***.println("Saldo: " + **this**.total + ". A retirar: " + importe);

**this**.total -= importe;

System.***out***.println(reparte(**this**.total));

}

}

}

### SOLUCION PROFE

|  |
| --- |
| package unidad5; |
|  |  |
|  | import java.util.Arrays; |
|  |  |
|  | public class Hucha { |
|  | public static final int CINCUENTA = 0; |
|  | public static final int VEINTE = 1; |
|  | public static final int DIEZ = 2; |
|  | public static final int CINCO = 3; |
|  | public static final int DOS = 4; |
|  | public static final int UNO = 5; |
|  |  |
|  | private static int [] tipos = {50, 20, 10, 5, 2, 1}; |
|  | private int [] desglose = new int[tipos.length]; |
|  | private boolean abierta; |
|  | private String contraseña = ""; |
|  |  |
|  | public Hucha() { |
|  | } |
|  |  |
|  | public Hucha(boolean abierta, String contraseña) { |
|  | this.abierta = abierta; |
|  | this.contraseña = contraseña; |
|  | } |
|  |  |
|  | public Hucha(int cincuenta, int veinte, int diez, int cinco, int dos, int uno) { |
|  | desglose[0] = cincuenta; |
|  | desglose[1] = veinte; |
|  | desglose[2] = diez; |
|  | desglose[3] = cinco; |
|  | desglose[4] = dos; |
|  | desglose[5] = uno; |
|  | } |
|  |  |
|  | public Hucha(int cincuenta, int veinte, int diez, int cinco, int dos, int uno, boolean abierta) { |
|  | this(cincuenta, veinte, diez, cinco, dos, uno); |
|  | this.abierta = abierta; |
|  | } |
|  |  |
|  | public Hucha(int cincuenta, int veinte, int diez, int cinco, int dos, int uno, boolean abierta, String contraseña) { |
|  | this(cincuenta, veinte, diez, cinco, dos, uno, abierta); |
|  | this.contraseña = contraseña; |
|  | } |
|  |  |
|  | public Hucha(int cantidad) { |
|  | for (int i=0; i<tipos.length && cantidad > 0; i++) { |
|  | if (cantidad > tipos[i]) { |
|  | desglose[i] = cantidad / tipos[i]; |
|  | cantidad %= tipos[i]; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | public Hucha(int cantidad, boolean abierta) { |
|  | this(cantidad); |
|  | this.abierta = abierta; |
|  | } |
|  |  |
|  | public Hucha(int cantidad, boolean abierta, String contraseña) { |
|  | this(cantidad, abierta); |
|  | this.contraseña = contraseña; |
|  | } |
|  |  |
|  | public int getDesglose(int tipo) throws HuchaVaciaException { |
|  | if (abierta) |
|  | return desglose[tipo]; |
|  | else |
|  | throw new HuchaVaciaException(); |
|  | } |
|  |  |
|  | public int retirar(int cantidad, int tipo) throws HuchaVaciaException { |
|  | if (abierta) { |
|  | if (desglose[tipo] < cantidad) |
|  | cantidad = desglose[tipo]; |
|  | desglose[tipo] -= cantidad; |
|  | return cantidad; |
|  | } |
|  | else |
|  | throw new HuchaVaciaException(); |
|  | } |
|  |  |
|  | public int retirar(int cantidad) throws HuchaVaciaException { |
|  | // modificar esto para hacer lo que se pide |
|  | if (abierta) { |
|  | for (int i=0; i<tipos.length && cantidad > 0; i++) { |
|  | if (cantidad > tipos[i]) { |
|  | desglose[i] = cantidad / tipos[i]; |
|  | cantidad %= tipos[i]; |
|  | } |
|  | } |
|  | return 0; // esto no esta bien |
|  | } |
|  | else |
|  | throw new HuchaVaciaException(); |
|  | } |
|  |  |
|  | public void abrir(String contraseña) throws ContraseñaIncorrectaException { |
|  | if (this.contraseña.equals(contraseña)) |
|  | abierta = true; |
|  | else |
|  | throw new ContraseñaIncorrectaException(); |
|  | } |
|  |  |
|  | public void cerrar(String contraseña) throws ContraseñaIncorrectaException { |
|  | if (this.contraseña.equals(contraseña)) |
|  | abierta = false; |
|  | else |
|  | throw new ContraseñaIncorrectaException(); |
|  | } |
|  |  |
|  | @Override |
|  | public String toString() { |
|  | return Arrays.toString(desglose); |
|  | } |
|  |  |
|  | } |

package unidad5;

public class PruebaHucha {

public static void main(String[] args) {

Hucha hucha = new Hucha(325);

try {

hucha.abrir("a");

System.out.println(hucha.getDesglose(Hucha.DIEZ));

} catch (HuchaVaciaException e) {

System.out.println("La hucha está cerrada");

} catch (ContraseñaIncorrectaException e1) {

System.out.println("Contraseña incorrecta");

}

}

}

### SOLUCIÓN VERO

package unidad5;

import java.util.Arrays;

import java.util.Scanner;

public class Hucha {

// atributos //

private String clave;

private int importe; // parece que no hace falta

private boolean abierta;

private int[] tipos = {50,20,10,5,2,1}; // tipos de monedas/bill admitidos

private int[] desglose = new int[tipos.length]; // array para almacenar uds de cada tipo

public int cincuenta, veinte, diez, cinco, dos, uno;

// constructores //

public Hucha(String clave, boolean abierta) { // crea hucha vacía

this.clave = clave;

this.abierta = abierta;

this.importe = 0;

}

public Hucha(String clave, boolean abierta, int cincuenta, int veinte, int diez, int cinco, int dos, int uno) { // crea hucha con billetes y/o monedas

this.clave = clave;

this.abierta = abierta;

this.desglose[0] = cincuenta;

this.desglose[1] = veinte;

this.desglose[2] = diez;

this.desglose[3] = cinco;

this.desglose[4] = dos;

this.desglose[5] = uno;

//this(cincuenta,veinte,diez,cinco,dos,uno); sería otro modo más rápido de asignar

this.importe = (cincuenta \* 50 + veinte \* 20 + diez \* 10 + cinco \* 5 + dos \* 2 + uno);

}

public Hucha (String clave, boolean abierta, int importe) { //crea hucha indicando solo el importe

this.clave = clave;

this.abierta = abierta;

this.importe = importe;

for (int i=0; i<tipos.length && importe > 0; i++) { // recorre el array tipos

if (importe >= tipos[i]) { // si es mayor o igual que el bill/moneda

desglose[i] = importe / tipos[i]; // se graba en array desglose

importe %= tipos[i]; // se asigna el resto del importe para la siguiente iteración

}

}

}

// métodos get

public int[] getDesglose() { // devuelve desglose

return desglose;

}

public String getAbierta() {

if (abierta) {

return "Abierta";

} else {

return "Cerrada";

}

}

public int ingresar(int cantidad, int tipo) {

//Arrays.asList(this.tipos);

Arrays.asList(this.tipos).indexOf(tipo); // crea lista de array para poder usar búsqueda que devuelva posición, p.e.: busca "20" y devuelve pos 1 del array

this.importe += (cantidad \* tipo);

return this.importe;

}

// métodos set

public void setClave(String clave) {

this.clave = clave;

}

public void setImporte(int importe) {

this.importe = importe;

for (int i=0; i<tipos.length && importe > 0; i++) { // recorre el array tipos

if (importe >= tipos[i]) { // si es mayor o igual que el bill/moneda

desglose[i] = importe / tipos[i]; // se graba en array desglose

importe %= tipos[i]; // se asigna el resto del importe para la siguiente iteración

}

}

}

/\* NO HACE FALTA

public void setDesglose(int a, int b, int c, int d, int e, int f) {

int[] tmpDesglose = {a,b,c,d,e,f};

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

this.desglose[i] = tmpDesglose[i];

}

}

\*/

public void setAbierta(String clave) {

if (clave.equals(this.clave)) {

if (abierta) {

abierta=false;

System.out.println("Hucha cerrada");

} else {

abierta=true;

System.out.println("Hucha abierta");

}

} else {

System.out.println("Clave incorrecta");

}

}

public void setRetirar(int cantidad) {

if (this.importe>=cantidad) {

this.importe -= cantidad;

System.out.println(this.importe);

} else {

System.out.println("No tiene suficiente saldo");

}

}

// toString //

@Override

public String toString() {

return "Hucha con "+ importe +"€\nBilletes: 50x" + desglose[0] + " 20x" + desglose[1] + " 10x" + desglose[2] + " 5x" + desglose[3] + "\nMonedas: 2x" + desglose[4] + " 1x" + desglose[5] + "\nLa hucha está abierta: " + abierta;

}

// EJECUCION //

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner input = new Scanner (System.in); // Lo uso para String

Scanner in = new Scanner (System.in); // Lo uso para Int

System.out.println("Alta de nueva hucha. Introduzca clave.");

String tmpClave=input.nextLine();

System.out.println("¿Abierta o cerrada?");

boolean tmpAbierta=true;

String tmpEstado=input.nextLine();

if (tmpEstado.equalsIgnoreCase("abierta") ) {

tmpAbierta=true;

} else if (tmpEstado.equalsIgnoreCase("cerrada")) {

tmpAbierta=false;

} else {

System.out.println("ERROR");

System.exit(1);

}

System.out.println("Indique importe inicial");

int tmpImporte=in.nextInt();

Hucha hucha1 = new Hucha(tmpClave,tmpAbierta,tmpImporte);

System.out.println(hucha1);

String accion="inicio";

do {

System.out.println("Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir");

accion = input.nextLine();

switch (accion) {

case "ingresar":

System.out.println("Valor del bill/moneda");

int a = in.nextInt();

System.out.println("Unidades");

int b = in.nextInt();

System.out.println(hucha1.ingresar(b,a));

break;

case "retirar":

System.out.println("Cantidad");

hucha1.setRetirar(in.nextInt());

break;

case "abrir":

case "cerrar":

System.out.println("Introduzca clave");

hucha1.setAbierta(input.nextLine());

break;

case "estado":

System.out.println(hucha1);

break;

case "clave":

System.out.println("Introduzca nueva clave");

hucha1.setClave(input.nextLine());

break;

default:

System.out.println("FIN");

System.exit(1); //Termina programa de forma inesperada

}

} while (accion!="salir");

input.close();

in.close();

}

}

### RESULTADO VERO

Alta de nueva hucha. Introduzca clave.

1234

¿Abierta o cerrada?

abierta

Indique importe inicial

500

Hucha con 500€

Billetes: 50x10 20x0 10x0 5x0

Monedas: 2x0 1x0

La hucha está abierta: true

Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir

ingresar

Valor del bill/moneda

20

Unidades

2

540

Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir

retirar

Cantidad

10

530

Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir

estado

Hucha con 530€

Billetes: 50x10 20x0 10x0 5x0

Monedas: 2x0 1x0

La hucha está abierta: true

Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir

cerrar

Introduzca clave

1234

Hucha cerrada

Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir

clave

Introduzca nueva clave

321

Acciones: ingresar, retirar, abrir o cerrar, estado, clave, salir

salir

FIN

## Ejercicio 3: Autor

Definir una clase llamada Autor según las especificaciones siguientes: • Atributos de instancia: o nombre, de tipo String o email, de tipo String o género (para almacenar el género: masculino o femenino), de tipo String • Métodos de instancia: o constructor que inicialice el estado del objeto con los valores que reciba a través de sus parámetros formales. o Getters para obtener el nombre, el email y el género de una instancia de la clase. o Setter para cambiar el correo electrónico de las instancias de la clase. o Redefinir el método toString para que retorne una cadena que contenga una breve descripción del estado del objeto que lo invoca con el formato: "nombre\_autor (género) email". Ejemplo: "Alberto Fernández (masculino) afernandez@mail.com" Escribir un programa que ponga a prueba la clase Autor utilizando todos sus constructores y métodos

**package** unidad5;

**public** **class** Autor {

// ATRIBUTOS

**private** String nombre, email, genero;

// METODOS

// Constructor

**public** Autor(String nombre, String email, String genero) {

**this**.nombre = nombre;

**this**.email = email;

**this**.genero = genero;

}

// Getters

**public** String getNombre() {

**return** **this**.nombre;

}

**public** String getEmail() {

**return** **this**.email;

}

**public** String getGenero() {

**return** **this**.genero;

}

// Setters

**public** **void** setEmail(String email) {

**this**.email = email;

}

@Override

**public** String toString() {

**return** (**this**.nombre + " (" + **this**.genero + ") " + **this**.email);

}

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

// **TODO** Auto-generated method stub

Autor autor1 = **new** Autor("Paco","paco@gmail.com","masculino");

System.***out***.println(autor1);

autor1.setEmail("paco2@gmail.com");

System.***out***.println(autor1.getNombre());

System.***out***.println(autor1.getGenero());

System.***out***.println(autor1.getEmail());

}

}

### RESULTADO

Paco (masculino) paco@gmail.com

Paco

masculino

[paco2@gmail.com](mailto:paco2@gmail.com)

## Ejercicio 4: Libro

### SOLUCIÓN VERO

**package** unidad5;

**import** java.util.Arrays;

**public** **class** Libro {

/\*

Ejercicio 4: Libro

Definir una clase llamada Libro según las especificaciones siguientes:

• Atributos de instancia:

o título, de tipo String

o autores, de tipo Collection<Autor>

o precio, de tipo float

o stock, de tipo int

• Métodos de instancia:

o Constructor que inicialice el estado del objeto con los datos recibidos en los parámetros

formales siguientes: título de tipo String, autores de tipo Collection<Autor> y

precio de tipo float. El atributo stock se inicializará con el valor por defecto.

o Constructor que inicialice todos los atributos con datos recibidos a través de parámetros

formales.

o Getters

o Setters para los atributos precio y stock.

o Redefinir el método toString para que retorne información con el formato:

"título\_libro (autor1, autor2, …) precio € - stock unidades en stock"

Escribir un programa que ponga a prueba la clase Libro utilizando todos sus constructores y métodos.

\*/

// ATRIBUTOS

**private** String titulo;

**private** **float** precio;

**private** **int** stock;

**private** String[] autoresArray;

// METODOS

// Constructor

**public** Libro(String titulo, String[] autoresArray, **float** precio, **int** stock) {

**this**.titulo = titulo;

**this**.autoresArray = autoresArray;

**this**.precio = precio;

**this**.stock = stock;

}

// Getters

**public** String getTitulo() {

**return** **this**.titulo;

}

**public** String getAutores() {

**return** Arrays.*toString*(**this**.autoresArray);

}

**public** **float** getPrecio() {

**return** **this**.precio;

}

**public** **int** getStock() {

**return** **this**.stock;

}

// Setters

**public** **void** setPrecio(**float** precio) {

**this**.precio = precio;

}

**public** **void** setStock(**int** stock) {

**this**.stock = stock;

}

// toString

@Override

**public** String toString() {

**return** (**this**.titulo + " - " + Arrays.*toString*(**this**.autoresArray) + " - " + **this**.precio + "€ - " + stock + " unidades en stock");

}

// EJECUCIÓN

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

// **TODO** Auto-generated method stub

String miArray[] = {"Varios autores"};

Libro libro1 = **new** Libro("Cuentos de cuna",miArray,(**float**)10.50,5);

System.***out***.println(libro1);

String miArray2[] = {"Karlos Arguiñano", "Eva Arguiñano"};

Libro libro2 = **new** Libro("Recetas de cocina",miArray2,(**float**)14,8);

System.***out***.println(libro2);

libro1.setPrecio((**float**) 8.50);

System.***out***.println(libro1.getPrecio());

libro2.setStock(7);

System.***out***.println(libro2.getStock());

System.***out***.println(libro2.getAutores());

System.***out***.println(libro1.getTitulo());

}

}

### RESULTADO VERO

Cuentos de cuna - [Varios autores] - 10.5€ - 5 unidades en stock

Recetas de cocina - [Karlos Arguiñano, Eva Arguiñano] - 14.0€ - 8 unidades en stock

8.5

7

[Karlos Arguiñano, Eva Arguiñano]

Cuentos de cuna

## Ejercicio 5: Naipe

### SOLUCIÓN VERO

## Ejercicio 6: Mazo

Definir la clase Mazo para poder representar un mazo de naipes según las especificaciones siguientes: • Un objeto Mazo almacenará una colección de naipes (usar un ArrayList). • Constructores: o Constructor sin parámetros que crea un mazo en el que se encuentran los 13 naipes de cada uno de los cuatro palos (52 naipes en total). o Costructor que recibe un parámetro de n tipo int para construir un mazo formado por n veces los 13 naipes de los cuatro palos (n \* 52 naipes en total) • Métodos de instancia: o get: retorna un naipe elegido de forma aleatoria sin retirarlo del mazo. o remove: retorna un naipe elegido de forma aleatoria retirándolo del mazo. o add: añade un naipe al mazo. o addAll: añade al mazo los naipes almacenados en una colección que se recibe a través de un parámetro forma. o Añadir cualesquiera métodos adicionales que se consideren útiles.

### SOLUCIÓN VERO

**package** unidad5;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Random;

**public** **class** Mazo {

// ATRIBUTOS

**private** **int**[] cartasPosibles = {

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};

**private** ArrayList<Integer> naipes = **new** ArrayList<>();

// METODOS

// Constructores

**public** Mazo() {

//this.naipes.add(cartasPosibles);

//naipes = new ArrayList<>(Arrays.asList(cartasPosibles));

**for** (**int** carta: cartasPosibles) { // Recorre array

**this**.naipes.add(carta); // Añade valor de array a arrayList

}

}

**public** Mazo(**int** n) {

**for** (**int** i=1; i<=n;i++) {

**for** (**int** carta: cartasPosibles) { // Recorre array

**this**.naipes.add(carta); // Añade valor de array a arrayList

}

}

}

// get

**public** **int** getNaipe() {

Random r = **new** Random();

**int** naipePosicion = r.nextInt(**this**.naipes.size()); // número random entre 0 y tamaño de ArrayList

**return** **this**.naipes.get(naipePosicion); // devuelve valor de la posición dentro del ArrayList

}

// remove

**public** **int** removeNaipe() {

Random r = **new** Random();

**int** naipePosicion = r.nextInt(**this**.naipes.size()); // número random entre 0 y tamaño de ArrayList

**int** carta = **this**.naipes.get(naipePosicion);

**this**.naipes.remove(naipePosicion);

**return** carta;

}

// add

**public** **void** addNaipe(**int** carta) {

**this**.naipes.add(carta);

}

// addAll

**public** **void** addAll(Collection listaCartas) {

**this**.naipes.addAll(listaCartas);

}

//toString

@Override

**public** String toString() {

**return** ("Número de cartas de la baraja: " + **this**.naipes.size() + "\n" + **this**.naipes);

}

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

// **TODO** Auto-generated method stub

Mazo nuevoMazo = **new** Mazo(2); // Creo un mazo con dos barajas

System.***out***.println(nuevoMazo); // Imprimo contenido mazo

System.***out***.println("Se selecciona un naipe: " + nuevoMazo.getNaipe()); // Devuelvo naipe al azar

System.***out***.println("Se descarta el naipe " + nuevoMazo.removeNaipe()); // Borro naipe al azar

System.***out***.println("Se añade naipe 8");

nuevoMazo.addNaipe(8); // Añado naipe

Collection listaCartas = **new** ArrayList<Integer>(); // Creo coleccion de naipes

listaCartas.add(12);

listaCartas.add(6);

nuevoMazo.addAll(listaCartas); // Añado coleccion al mazo

System.***out***.println(nuevoMazo); // Devuelvo contenido del mazo

}

}

### RESULTADO VERO

Número de cartas de la baraja: 112

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

Se selecciona un naipe: 4

Se descarta el naipe: 1

Se añade naipe 8

Número de cartas de la baraja: 114

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 8, 12, 6]

## Ejercicio 6- B: Naipe y Mazo

Escribir un programa que utilizando las clases Naipe y Mazo permita jugar a la carta más alta con el ordenador como oponente, teniendo en cuenta que los naipes se ordenan de menor a mayor valor de la forma siguiente: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

### SOLUCIÓN VERO

## Ejercicio 7: Automovil

### SOLUCIÓN VERO

**package** unidad5;

**public** **class** Automovil {

/\*\*

\* Definir una clase Automovil según las especificaciones siguientes:

• Cada objeto Automovil permitirá llevar el control del combustible que consume en los trayectos

que realiza.

• Los atributos con los que se describe el estado de cada objeto Automovil son:

o Modelo.

o Capacidad del depósito de combustible en litros.

o Cantidad de combustible en el depósito en litros.

o Consumo por kilómetro en litros.

o Cantidad total de kilómetros recorridos.

o Cantidad total de combustible consumido.

\*/

// ATRIBUTOS

**public** String modelo;

**private** **int** capacidad;

**private** **float** combustible;

**private** **float** consumo;

**private** **float** kmTotales;

**private** **float** consumoTotal;

// METODOS

/\*\* Constructores

o Uno que inicialice mediante sus parámetros los atributos modelo, capacidad del depósito,

cantidad inicial de litros en el depósito y consumo. El resto de atributos tendrán el valor 0.

o Uno que inicialice mediante sus parámetros los atributos modelo, capacidad del depósito y

consumo. La cantidad de combustible en el depósito se inicializará con el valor máximo y el

resto de atributos con el valor 0

\*/

**public** Automovil(String modelo, **int** capacidad, **float** combustible, **float** consumo) {

**this**.modelo = modelo;

**this**.capacidad = capacidad;

**this**.consumo = consumo;

**if** (combustible>capacidad) {

**this**.combustible = capacidad;

} **else** {

**this**.combustible = combustible;

}

**this**.kmTotales = 0;

**this**.consumoTotal = 0;

}

**public** Automovil(String modelo, **int** capacidad, **float** consumo) {

**this**.modelo = modelo;

**this**.capacidad = capacidad;

**this**.combustible = capacidad;

**this**.consumo = consumo;

**this**.kmTotales = 0;

**this**.consumoTotal = 0;

}

/\*\*

\* Método llenarDeposito. Este método estará sobrecargado:

o Una versión que llene el depósito hasta el límite.

o Otra que llene el depósito con una cantidad de litros específica. Si se excede la capacidad del

depósito, se llenará el depósito retornando la cantidad de combustible sobrante.

\*/

**public** String llenarDeposito() {

**this**.combustible = **this**.capacidad;

**return** "Combustible llenado al máximo: " + **this**.combustible;

}

**public** String llenarDeposito(**int** litros) {

String msg = "";

**if** ((**this**.combustible + litros)>**this**.capacidad) {

msg = "Combustible llenado al máximo. Sobran " + (**this**.combustible + litros - **this**.capacidad) + " litros.";

**this**.combustible = **this**.capacidad;

} **else** {

msg = "Combustible llenado hasta " + (**this**.combustible + litros) + " litros.";

}

**this**.combustible =+ litros;

**return** msg;

}

/\*\*

Método desplazar que reciba la cantidad de kilómetros que ha de recorrer el automóvil. Este

método comprobará si es posible el desplazamiento con el combustible que hay en el depósito. Si

es posible, descontará del depósito el combustible que corresponda y retornará dicha cantidad. Si

no es posible tendrá que ponerlo de manifiesto de alguna forma.

\*/

**public** String desplazar(**int** km) {

String msg = "";

**float** litrosDesp = km \* **this**.consumo / 100; // Calculo el gasto en litros del desplazamiento. P.e.: 20 km \* 6 L/100 km = 1,20 L

**if** (**this**.combustible >= litrosDesp) { // Si el consumo es igual o inferior al combustible en deposito

**this**.kmTotales += km; // Actualizo Km recorridos

**this**.consumoTotal += litrosDesp; // Actualizo consumo

**this**.combustible -= litrosDesp; // Actualizo deposito

msg = "Desplazamiento de " + km + " Kms. Total Km: " + **this**.kmTotales +". Combustible: " + **this**.combustible;

} **else** {

**float** maxKm = **this**.combustible / **this**.consumo \* 100;

msg = "Vehículo sin combustible suficiente. Máximo: " + maxKm;

}

**return** msg;

}

// Get

**public** **float** getKm() {

**return** **this**.kmTotales;

}

// Set

**public** **void** setConsumo(**int** litros) {

**this**.consumo = litros;

}

// toString

@Override

**public** String toString() {

**return** "Vehículo modelo " + **this**.modelo + ". Características:\n- Capacidad deposito: " + **this**.capacidad + "\n- Consumo: "

+ **this**.consumo + "L a los 100Km\n- Combustible actual: " + **this**.combustible + "L\n- Total Km: " + **this**.kmTotales +

"Km\n- Total litros consumidos: " + **this**.consumoTotal + "\n";

}

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

// **TODO** Auto-generated method stub

Automovil coche= **new** Automovil("Ford Ka",40,7);

System.***out***.println(coche);

System.***out***.println(coche.desplazar(25));

System.***out***.println(coche.desplazar(33));

System.***out***.println(coche.desplazar(100));

System.***out***.println(coche.llenarDeposito(10));

System.***out***.println("Distancia recorrida: " + coche.getKm() + "Km");

}

}

### RESULTADO VERO

Vehículo modelo Ford Ka. Características:

- Capacidad deposito: 40

- Consumo: 7.0L a los 100Km

- Combustible actual: 40.0L

- Total Km: 0.0Km

- Total litros consumidos: 0.0

Desplazamiento de 25 Kms. Total Km: 25.0. Combustible: 38.25

Desplazamiento de 33 Kms. Total Km: 58.0. Combustible: 35.94

Desplazamiento de 100 Kms. Total Km: 158.0. Combustible: 28.939999

Combustible llenado hasta 38.94 litros.

Distancia recorrida: 158.0Km

## Ejercicio 7-B: Automovil

NO SE PERMITEN EN NINGÚN MÉTODO DE LA CLASE SENTENCIAS QUE ENVÍE DATOS A LA SALIDA ESTÁNDAR. Desarrollar un programa que ponga a prueba la clase Automovil con datos que se recibirán a través de la entrada estándar según las especificaciones siguientes: • Número N de automóviles en la primera línea. • En las N líneas siguientes se recibirán los datos de los automóviles a razón de uno por línea con el formato siguiente: . No se repetirá ningún modelo, es decir, solo se creará un automóvil de cada modelo. Se almacenarán en un array todos los vehículos creados. • En el resto de las líneas excepto la última, se recibirán datos de los desplazamientos que han de realizar los automóviles con el formato siguiente: desplazar . Se pueden especificar varios desplazamientos para el mismo modelo en diferentes líneas. Por cada línea leída se ejecutará el desplazamiento inmediatamente y se mostrará en la salida estándar el resultado de la forma siguiente: o Si el desplazamiento ha sido posible, se mostrará el modelo, el combustible que queda en el depósito y el combustible consumido. o Si el desplazamiento no es posible, se mostrará el mensaje “Combustible insuficiente para este desplazamiento”. • En la última línea, la palabra “fin”. Después de procesar todas las líneas de entrada, Se mostrará por cada vehículo el modelo, el combustible que le queda, la cantidad total de kilómetros recorridos y el total de litros de combustible consumidos.

### SOLUCIÓN VERO

package unidad5;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.InputMismatchException;

import java.util.Scanner;

public class PruebaAutomovil {

public static void main(String[] args) {

// TODO Auto-generated method stub

int numAutos = 0;

Scanner texto = new Scanner (System.in);

Scanner num = new Scanner (System.in);

String altaAuto, accion = "";

String [] arAltaAuto = new String[4];

//int indice1, indice2;

System.out.println("Introduzca nº de automoviles");

numAutos = num.nextInt();

ArrayList<Automovil> arrayAutos = new ArrayList<>();

for (int i=0 ; i<numAutos ; i++) {

boolean repe = false;

System.out.println("Introduzca <modelo> <capacidad del depósito> <litros en el depósito> <consumo>");

altaAuto = texto.nextLine();

// Analizo el texto introducido

arAltaAuto = altaAuto.split(" ");

/\* ESTA SOLUCION ES VÁLIDA PERO MÁS COMPLEJA

indice1 = altaAuto.indexOf(" ",0); // Busco el primer espacio

arAltaAuto[0] = altaAuto.substring(0,indice1); // Guardo el modelo

indice2 = altaAuto.indexOf(" ", indice1+1); // Busco el siguiente espacio

arAltaAuto[1] = Integer.parseInt(altaAuto.substring(indice1+1,indice2)); // Guardo capacidad pasando a int

indice1 = altaAuto.indexOf(" ", indice2+1); // Busco el siguiente espacio

arAltaAuto[2] = Float.parseFloat(altaAuto.substring(indice2+1,indice1));

arAltaAuto[3] = Float.parseFloat(altaAuto.substring(indice1+1,altaAuto.length()));

\*/

for (Automovil auto: arrayAutos) {

if (auto.modelo.equals(arAltaAuto[0])) {

repe = true;

System.out.println("¡Ese modelo ya existe!\n");

continue;

}

}

if (!repe) {

// Creo el auto

Automovil tmpAuto = new Automovil(arAltaAuto[0],Integer.parseInt(arAltaAuto[1]),Float.parseFloat(arAltaAuto[2]),Float.parseFloat(arAltaAuto[3]));

arrayAutos.add(tmpAuto);

}

}

// Desplazar

accion = texto.nextLine();

while (!(accion.equals("fin"))) {

/\*int coincide = accion.indexOf("desplazar");

if (coincide!=-1) {\*/

try {

String [] arAccion = accion.split(" ");

for (Automovil auto: arrayAutos) {

if (auto.modelo.equals(arAccion[1]) && arAccion[0].equalsIgnoreCase("desplazar")) {

System.out.println(auto.desplazar(Integer.parseInt(arAccion[2])));

continue;

}

}

} catch(Exception e) {

System.out.println("ERROR");

}

accion = texto.nextLine();

}

for (Automovil auto: arrayAutos) {

System.out.println("RESUMEN:\n" + auto);

}

num.close();

texto.close();

}

}

### RESULTADO VERO

Introduzca nº de automoviles

3

Introduzca <modelo> <capacidad del depósito> <litros en el depósito> <consumo>

A 40 40 5

Introduzca <modelo> <capacidad del depósito> <litros en el depósito> <consumo>

B 50 20 7

Introduzca <modelo> <capacidad del depósito> <litros en el depósito> <consumo>

B 60 60 7

¡Ese modelo ya existe!

Desplazar B 50

Desplazamiento de 50 Kms. Total Km: 50.0. Combustible: 16.5

Desplazar B 200

Desplazamiento de 200 Kms. Total Km: 250.0. Combustible: 2.5

desplazar B 100

Vehículo sin combustible suficiente. Máximo: 35.714287

desplazar A 25

Desplazamiento de 25 Kms. Total Km: 25.0. Combustible: 38.75

fin

RESUMEN:

Vehículo modelo A. Características:

- Capacidad deposito: 40

- Consumo: 5.0L a los 100Km

- Combustible actual: 38.75L

- Total Km: 25.0Km

- Total litros consumidos: 1.25

RESUMEN:

Vehículo modelo B. Características:

- Capacidad deposito: 50

- Consumo: 7.0L a los 100Km

- Combustible actual: 2.5L

- Total Km: 250.0Km

- Total litros consumidos: 17.5

## Ejercicio 8: Pokemon

Se desea desarrollar un programa que simule torneos de pokemons. Para comenzar, habra que desarrollar dos clases, además de la clase principal: • Entrenador: cada entrenador tiene un nombre, un número de insignias y una colección de pokemons (usar un ArrayList). • Pokemon: cada pokemon tiene un nombre, está ligado a un elemento fundamental y tiene un determinado número de puntos de salud. Para simular los torneos, el programa recibe a través de la entrada estándar una serie de datos que tendrá que procesar para obtener el resultado de cada torneo. Las especificaciones de entrada son las siguientes: • Cada entrenador tiene un nombre único y comienza con cero insignias. • Se recibirán un número indeterminado de líneas en la entrada estándar seguidas de una linea con la palabra “torneo”. Cada una de estas líneas contiene información acerca de un pokemos y del entrenador que lo ha capturado con el formato: . • A la línea con la plabra “torneo” le sigue un número indeterminado de líneas, cada una conteniendo el nombre de uno de los elementos fundamentales (fuego, agua, electricidad, …) seguidas de una línea con la palabra “fin”. Por cada una de esta líneas se ha de comprobar, por cada entrenador, si éste tiene al menos un pokemon ligado al elemento leido. Si es así, recibirá una insignia. En caso contrario, todos sus pokemons perderán 10 puntos de salud y aquellos que llegen a cero puntos de salud morirán, por lo que tendrán que ser eliminados de la colección del entrenador. • Cuando se lee la línea con la palabra “fin”, el torneo finaliza y se mostrará una lista de entrenadores en la que en cada línea se visualizara el nombre de un entrenador, su cantidad de insignias y el número de pokemos que le quedan. • OPCIONAL (esto se ve en la 3ª evaluación): mostrar la lista de entrenadores ordenada por número de insignias, y en caso de empate, por número de pokemos restantes.

### PRUEBAS VERO - TORNEO

**package** unidad5;

**import** java.util.ArrayList;

**import** java.util.Arrays;

//import java.util.List;

**import** java.util.Scanner;

**public** **class** Torneo {

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

// **TODO** Auto-generated method stub

ArrayList<Entrenador> arTrainer = **new** ArrayList<>();

//CODIGO PRUEBAS - FUNCIONA OK

/\*

Entrenador trainer1 = new Entrenador("Ash");

Pokemon pokem11 = new Pokemon("Charizard","fuego",100);

trainer1.setPokemon(pokem11);

Pokemon pokem12 = new Pokemon("Pikachu","electricidad",10);

trainer1.setPokemon(pokem12);

arTrainer.add(trainer1);

Entrenador trainer2 = new Entrenador("Brock");

Pokemon pokem21 = new Pokemon("Squirtle","agua",38);

trainer2.setPokemon(pokem21);

arTrainer.add(trainer2);

for (Entrenador a: arTrainer) {

System.out.println(a.getInsignias() + " " + a.cuentaPokemons());

}

String teclado = "agua";

while (!(teclado.equals("fin"))){

//List <String> alElementos = Arrays.asList(Pokemon.listaElementos);

//if (alElementos(Pokemon.listaElementos).contains(teclado)) {

if (Arrays.asList(Pokemon.listaElementos).contains(teclado)) {

for (Entrenador a: arTrainer) {

System.out.println("Busca elemento en: " + a);

if (a.searchElemento(teclado)) {

System.out.println("EXITO. Se añade insignia");

} else {

System.out.println("ELIMINAR 10 PUNTOS");

a.pierdeRonda();

}

}

} else {

System.out.println("Error al introducir elemento");

}

teclado = "fin";

}

System.out.println("RESULTADOS");

for (Entrenador a: arTrainer) {

System.out.println(a.getInsignias() + " " + a.cuentaPokemons());

}

// FIN CODIGO PRUEBAS

\*/

### RESULTADO PRUEBAS VERO - TORNEO

Ash 0 2

Brock 0 1

Busca elemento en: Nombre: Ash, nº insignias: 0, pokemons: [Nombre: Charizard, elemento: fuego, puntos de salud: 100, Nombre: Pikachu, elemento: electricidad, puntos de salud: 10]

ELIMINAR 10 PUNTOS

Nombre: Charizard, elemento: fuego, puntos de salud: 90

Nombre: Pikachu, elemento: electricidad, puntos de salud: 0

Muere pokemon: Nombre: Pikachu, elemento: electricidad, puntos de salud: 0

Busca elemento en: Nombre: Brock, nº insignias: 0, pokemons: [Nombre: Squirtle, elemento: agua, puntos de salud: 38]

EXITO. Se añade insignia

RESULTADOS

Ash 0 1

Brock 1 1

### SOLUCIÓN VERO

**package** unidad5;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** Torneo {

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

// **TODO** Auto-generated method stub

ArrayList<Entrenador> arTrainer = **new** ArrayList<>();

Scanner input = **new** Scanner (System.***in***);

String teclado = input.nextLine();

**while** (!(teclado.equals("torneo"))){

ArrayList<Entrenador> listaCopiada = **new** ArrayList<Entrenador>(arTrainer);

String [] tmpArray = teclado.split(" ");

Entrenador trainer = **new** Entrenador(tmpArray[0]);

Pokemon pokem = **new** Pokemon(tmpArray[1],tmpArray[2],Integer.*parseInt*(tmpArray[3]));

//System.out.println(listaCopiada);

**boolean** existe = **false**;

**for** (**int** i=0; i<listaCopiada.size();i++) {

**if** (listaCopiada.get(i).nombre.equals(trainer.nombre)) { // si existe en ArrayList se añade solo el pokemon

//System.out.println("SÍ existe entrenador");

arTrainer.get(i).setPokemon(pokem);

existe = **true**;

**break**;

} **else** {

//System.out.println("NO existe entrenador");

}

}

**if** (!existe) {

trainer.setPokemon(pokem);

arTrainer.add(trainer);

}

//System.out.println(arTrainer);

listaCopiada = arTrainer;

teclado = input.nextLine();

}

teclado = input.nextLine();

**while** (!(teclado.equals("fin"))){

**if** (Arrays.*asList*(Pokemon.*listaElementos*).contains(teclado)) {

**for** (Entrenador a: arTrainer) {

//System.out.println("Busca elemento en: " + a);

**if** (a.searchElemento(teclado)) {

//System.out.println("EXITO. Se añade insignia");

} **else** {

//System.out.println("ELIMINAR 10 PUNTOS");

a.pierdeRonda();

}

}

} **else** {

System.***out***.println("Error al introducir elemento");

}

teclado = input.nextLine();

}

System.***out***.println("RESULTADOS");

**for** (Entrenador a: arTrainer) {

System.***out***.println(a.getInsignias() + " " + a.cuentaPokemons());

}

input.close();

}

}