

Cuaderno de ejercicios: Refactorización de código

[Solución: Cuaderno de ejercicios: Refactorización de código](#)

Ejercicio 1: Refactor	2
Ejercicio 2: Empresa	3
Ejercicio 3: Shape	4
Ejercicio 4: Alumno	5
Ejercicio 5: GestorBD	6
Ejercicio 6: Minecraft Cubos	7
Ejercicio 7: Animales	8
Ejercicio 8: Cliente	8
Ejercicio 9: Venta	9
Ejercicio 10: Factura	10
Ejercicio 11: Proveedor	11
Ejercicio 12: AngryBirds	12
Ejercicio 13: Validación de datos	12
Ejercicio 14: Creador de usuarios	13
Ejercicio 15: Videoclub	13
Ejercicio 16: Sanitarios	16
Ejercicio 17: Laberinto	17
Ejercicio 18: Carta	19
Ejercicio 19: Encripta	21

Ejercicio 1: Refactor

Realiza la refactorización del siguiente programa

Dificultad: ★★

```
public class Ejemplo1 {
    public static float suma(float b) {
        int c = 6;
        float a = b * b;
        a++;
        b = b * b;
        b = b * b;
        b = b * b;
        if (a < 0)
            a = -a;
        a = a + b;
        return a;
    }

    public static void main(String[] args) {
        System.out.println(suma(4));
        System.out.println(suma(-4));
        System.out.println(suma(0));
    }
}
```

Ejercicio 2: Empresa

Realiza la refactorización del siguiente programa

Dificultad: ★ ★ ★

```
public class Empresa {
    public String state;
    public final static String VIGO = "vigo";
    public final static String CORUNA = "coruna";
    public final static String LUGO = "lugo";

    public final static double VIGO_IMPUESTO = 0.21;
    public final static double CORUNA_IMPUESTO = 0.18;
    public final static double LUGO_IMPUESTO = 0.16;

    public double base = 0.2;

    public double calc;
    public double points;

    public double calculo() {
        double rate, amt;

        if (state == VIGO) {
            rate = VIGO_IMPUESTO;
            amt = base * VIGO_IMPUESTO;
            calc = 2 * basis(amt) + extra(amt) * 1.05;
        } else if ((state == CORUNA) || (state == LUGO)) {
            rate = (state == CORUNA) ? CORUNA_IMPUESTO : LUGO_IMPUESTO;
            amt = base * rate;
            calc = 2 * basis(amt) + extra(amt) * 1.05;
            if (state == CORUNA)
                points = 2;
        } else {
            rate = 1;
            amt = base;
            calc = 2 * basis(amt) + extra(amt) * 1.05;
        }

        return calc;
    }

    private double basis(double e) {
        return e + e;
    }

    private double extra(double e) {
        return e * e;
    }

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.state = Empresa.LUGO;
        e1.calculo();
        System.out.println("c=" + e1.calc);
        System.out.println("c=" + e1.points);
    }
}
```

```

        Empresa e2 = new Empresa();
        e2.state = Empresa.CORUNA;
        e2.calculo();
        System.out.println("c=" + e2.calc);
        System.out.println("c=" + e2.points);

        Empresa e3 = new Empresa();
        e3.state = Empresa.VIGO;
        e3.calculo();
        System.out.println("c=" + e3.calc);
        System.out.println("c=" + e3.points);
    }
}

```

Ejercicio 3: Shape

Realiza la refactorización del siguiente programa

Dificultad: ★★

```

public class Shape {
    public static final int SQUARE = 1;
    public static final int CIRCLE = 2;
    public static final int RIGHT_TRIANGLE = 3;

    private int shapeType;
    private double size;

    public Shape(int shapeType, double size) {
        this.shapeType = shapeType;
        this.size = size;
    }

    // ... other methods ...
    public double area() {
        switch (shapeType) {
            case SQUARE:
                return size * size;
            case CIRCLE:
                return Math.PI * size * size / 4.0;
            case RIGHT_TRIANGLE:
                return size * size / 2.0;
        }
        return 0;
    }
}

```

Ejercicio 4: Alumno

Realiza la refactorización del siguiente programa

Dificultad: ★

```
public class Alumno {  
  
    int numeroDeRetrasos;  
  
    public Alumno(int numeroDeRetrasos) {  
        this.numeroDeRetrasos = numeroDeRetrasos;  
    }  
  
    public int obtenerCalificacion() {  
        // return (masDeCincoRetrasos()) ? 1 : 5;  
        if (masDeCincoRetrasos()) {  
            return 1;  
        } else {  
            return 5;  
        }  
    }  
  
    private boolean masDeCincoRetrasos() {  
        return numeroDeRetrasos > 5;  
    }  
  
    public static void main(String[] args) {  
        Alumno a1 = new Alumno(5);  
        System.out.println(a1.obtenerCalificacion());  
  
        Alumno a2 = new Alumno(1);  
        System.out.println(a2.obtenerCalificacion());  
  
        Alumno a3 = new Alumno(7);  
        System.out.println(a3.obtenerCalificacion());  
    }  
}
```

Ejercicio 5: GestorBD

Realiza la refactorización del siguiente programa

Dificultad: ★ ★ ★

```
import java.sql.Connection;
import java.util.ArrayList;

public class GestorBD {
    public Connection getConnection() {
        // Abrimos la conexión con la base de datos Ej MySQL (*
        // imaginatelo)
        return null;
    }

    public void cerrarConnection() {
        // Cerramos la conexión con la base de datos (* imaginatelo)
    }

    public ArrayList getListaEstudiantes() {
        Connection c = getConnection();
        // obtenemos la lista de alumnos de la base de datos (*
        // imaginatelo)
        // ... select * from estudiantes
        cerrarConnection();
        return null;
    }

    public void borrarEstudiantes() {
        Connection c = getConnection();
        // borramos los alumnos de la base de datos (* imaginatelo)
        // ... delete from estudiantes
        cerrarConnection();
    }

    public ArrayList getListaProfesores() {
        Connection c = getConnection();
        // obtenemos la lista de profesores de la base de datos (*
        // imaginatelo)
        // ... select * from profesores
        cerrarConnection();

        return null;
    }
}
```

```
}
```

Ejercicio 6: Minecraft Cubos

Realiza la refactorización del siguiente programa

Dificultad: ★★

```
class Cubo {
    enum TipoCubo {
        hierba, metal, tnt
    }

    String nombre;
    String textura;
    String daño;
    Point2D posicion;
    TipoCubo tipo;

    int rangoExplosion = 200;
    int dañoExplosion = 10;

    @Override
    void dibujar() {
        switch (tipo) {
            case tnt:
                // Dibujar en pantalla un cubo con textura tnt
                break;
            case metal:
                // Dibujar en pantalla un cubo con textura metal
                break;
            case hierba:
                // Dibujar en pantalla un cubo con textura hierba
                break;
            default:
                throw new UnsupportedOperationException();
        }
    }

    public void explotar() {
        switch (tipo) {
            case tnt:
                // Buscar usuario en un rango de 200 (rangoExplosion)
                // Aplicarles un daño de 10/200 = daño/rangoExplosion
                break;
            default:
                throw new UnsupportedOperationException();
        }
    }
}
```

Ejercicio 7: Animales

Realiza la refactorización los siguientes métodos

Dificultad: ★★

```
public char a(String m) {
    return m.toLowerCase().charAt(0);
}

public int suma(int a, int b, int c, int d, int e, int f) {
    return a + b + c + d + e + f;
}

private int obtener_numero_caracteres(String m) {
    return m.length();
}

public int calcularPeligrosidad(String animal) {
    switch (animal) {
        case "lobo":
            return 8;
        case "leon":
            return 10;
        case "gato":
            return 1;
    }
    return 0;
}

public char otenerUltimoCaracter(String texto) {
    if (texto == null)
        System.out.println("Error, el texto no puede ser nulo");
    return texto.trim().toLowerCase().charAt(texto.length() - 1);
}
```

Ejercicio 8: Cliente

Realiza la refactorización del siguiente programa

Dificultad: ★

```
public class Cliente {
    public String nombre;
    public String te; // el telefono
    public String direccion;
    public String ciudad;
    public String email;
    public int cp;
```



```

public void registrarCliente() {
    // guardamos el cliente
}

public boolean validarDireccion() {
    // validamos los datos del cliente
    return true;
}

public String getProvincia(int cp) {
    String textoCp = String.valueOf(cp);
    if (textoCp.startsWith("36"))
        return "Pontevedra";
    else if (textoCp.startsWith("37"))
        return "Salamanca";
    // else ....
    else
        System.err.println("No existe el codigo postal");
    return "error";
}
}

```

Ejercicio 9: Venta

Realiza la refactorización del siguiente programa.

Haz los test unitarios

Dificultad: ★ ★

```

public class Venta {
    float coste;
    float dto; // Descuento

    enum TipoCliente {
        GOLD, PLATINUM, NORMAL
    };

    boolean esEmpleado;
    TipoCliente tipoCliente;
    boolean tieneCuponGratis;

    float calculaCosteTotal() {
        // determinamos si el coste de la venta es gratis.
        if (esEmpleado)
            return 0;
    }
}

```

```

        if (tipoCliente == TipoCliente.GOLD)
            return 0;
        if (tipoCliente == TipoCliente.PLATINUM)
            return 0;
        if (tieneCuponGratis)
            return 0;

        return coste * dto + coste;
    }
}

```

Ejercicio 10: Factura

Realiza la refactorización del siguiente programa

Dificultad: ★★

```

import java.util.Date;

public class Factura {
    String Numero;
    String nombreCliente;
    String nifCliente;
    String dirCliente;
    String cifEmpresa;
    String dirEmpresa;
    String[] articulosNombre;
    float[] articulosPrecio;
    Date vencimiento;

    void imprime_factura() {
        // Pintamos La cabecera
        System.out.println("FACTURA nº " + Numero);
        System.out.println("Fecha: " + new Date());

        // Pintamos Los datos de La empresa que genera La factura
        System.out.println("CEBEM SL");
        System.out.println("CIF: " + cifEmpresa);
        System.out.println("Dirección" + dirEmpresa);
        System.out.println("36204 Vigo (Pontevedra)");

        // Pintamos Los datos del cliente
        System.out.println("CLIENTE:");
        System.out.println("-----");
        System.out.println("Nombre:" + nombreCliente);
    }
}

```

```

System.out.println("NIF: " + nifCliente);
System.out.println("Dirección" + dirCliente);
System.out.println("36204 Vigo (Pontevedra)");

// Pintamos Los datos de La factura
System.out.println("ARTICULOS:");
System.out.println("-----");

float total = 0;
for (int i = 0; i < articulosNombre.length; i++) {
    System.out.println(articulosNombre[i] + " " + articulosPrecio[i]);
    total = total + articulosPrecio[i];
}
System.out.println(" TOTAL: " + total);

// Pintamos el pie de la factura
System.out.println("Vencimiento: " + vencimiento.toString());
}
}

```

Ejercicio 11: Proveedor

Realiza la refactorización del siguiente programa

Dificultad: ★★ ★

```

class Proveedor {
    boolean activo = true;
}

public class OrdenVenta {
    float total;
    Proveedor proveedor;
    float peso;
    String tipo;

    public float calculaDescuento() {
        if (proveedor.activo == true && peso < 1000 &&
            tipo.toLowerCase().equals("normal")) {
            // aplicamos un descuento normal
            float coste = total * 0.5f;
            if (coste > 100)
                coste = 100;
            return coste;
        } else {
            // aplicamos un descuento especial
            float coste = total * 0.8f;
        }
    }
}

```

```

    if (coste > 200)
        coste = 200;
    return coste;
}
}

```

Ejercicio 12: AngryBirds

Crea las clases (métodos y atributos) del juego AngryBirds

Dificultad: ★ ★ ★

Ejercicio 13: Validación de datos

Realiza la refactorización del siguiente programa

Dificultad: ★ ★

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class DataValidation {

    public boolean validarEmail(String email) {
        if (email == null) {
            System.out.println("Email no puede ser nulo");
        } else {
            Pattern pattern =
                Pattern.compile("^([\\w-_.]+)*([\\w-_.]\\.\\.\\@|([\\w]+\\.\\.)+[\\w]+[\\w]$)");
            Matcher matcher = pattern.matcher(email);
            return matcher.matches();
        }
        return false;
    }

    /*
     * Este método valida un código postal.
     * Es decir un número entre 01000 y 52999:
     */
    public boolean validarCP(String input) {
        if (input.length() == 5 && Integer.valueOf(input) >= 1000 &&
            Integer.valueOf(input) <= 52999) {
            return true;
        } else {
            return false;
        }
    }
}
```

```
}  
}
```

Ejercicio 14: Creador de usuarios

Realiza la refactorización del siguiente programa

Dificultad: ★★

```
class User {  
    User(String nombre, String pass) {  
    }  
  
    String nombre;  
    String email;  
    String password;  
}  
  
public class UserCreator {  
    boolean valid(String valor) {  
        if (valor != null && valor.length() > 0)  
            return true;  
        else  
            return false;  
    }  
  
    User create(String email, String pass) {  
        if (!valid(email)) {  
            System.out.println("Email incorrecto");  
        } else {  
            if (valid(pass)) {  
                return new User(email, pass);  
            } else {  
                System.out.println("Password incorrecto");  
            }  
        }  
        return null;  
    }  
}
```

Ejercicio 15: Videoclub

Realiza la refactorización del siguiente programa

Dificultad: ★★★

```
import java.time.LocalDate;
import static java.time.temporal.ChronoUnit.DAYS;

class Pelicula {
    float precio;
    String tipo = "Normal"; // normal, estreno, clasica
    String titulo;
    String sinopsis;
}

class Alquiler {
    Pelicula pelicula;
    LocalDate fechaAlquiler = LocalDate.now();

    long getDiasAlquiler() {
        LocalDate fechaActual = LocalDate.now(); // fecha actual
        return DAYS.between(fechaAlquiler, fechaActual);
    }
}

class Cliente {
    String nombre;
    String calle;
    String piso;
    String letra;
    String cp;
    String telefono;

    Alquiler[] alquileres;

    float calculaTotalAlquiler(Alquiler alquiler) {
        float recargo = 0;
        float precioPelicula = 0;

        // Calcular recargo por días alquilado
        if (alquiler.getDiasAlquiler() <= 7) {
            recargo = 0;
        }
        if (alquiler.getDiasAlquiler() >= 8 && alquiler.getDiasAlquiler() < 20) {
            recargo = alquiler.getDiasAlquiler() * 10;
        }
        if (alquiler.getDiasAlquiler() > 20) {
            recargo = 250;
        }

        // Calculo precio pelicula
        if (alquiler.pelicula.tipo == "Clasica") {
            precioPelicula = alquiler.pelicula.precio * 1;
        }
        if (alquiler.pelicula.tipo == "Normal") {
            precioPelicula = alquiler.pelicula.precio * 2;
        }
        if (alquiler.pelicula.tipo == "Estreno") {
            precioPelicula = alquiler.pelicula.precio * 4;
        }
    }
}
```

```

        return recargo + precioPelicula;
    }

    float calcularTotalAlquileres() {
        float total = 0;
        for (Alquiler al : alquileres) {
            total += calculaTotalAlquiler(al);
        }
        return total;
    }

    // Un cliente es moroso si tiene algún alquiler de más de 20 días
    boolean esMoroso() {
        boolean moroso = false;
        for (Alquiler al : alquileres) {
            if (al.getDiasAlquiler() > 20) {
                moroso = true;
            }
        }
        return moroso;
    }
}

```

Ejercicio 16: Sanitarios

Realiza la refactorización del siguiente programa

Dificultad: ★★

```
// Tenemos 3 tipos de sanitarios ,
// enfermeros, medicos de familia y cirujanos
public class Sanitario {

    private int numeroGuardias; // solo aplicable a los medicos de familia y cirujanos
    private int numeroOperaciones; // Solo aplicable a los cirujanos
    private Sanitario[] sanitariosACargo; // solo aplicable a los medicos de familia y
    cirujanos
    private Sanitario jefe; // Aplicable a cualquier tipo de sanitario
    private int inyeccionesSuministradas; // solo aplicable a los enfermeros
    private int sueldo_bruto; // Aplicable a cualquier tipo de sanitario

    Sanitario(int numeroGuardias, int numeroOperaciones, Sanitario[] sanitariosACargo,
    Sanitario jefe,
        int inyeccionesSuministradas, int sueldo_bruto) {
        this.numeroGuardias = numeroGuardias;
        this.numeroOperaciones = numeroOperaciones;
        this.sanitariosACargo = sanitariosACargo;
        this.jefe = jefe;
        this.inyeccionesSuministradas = inyeccionesSuministradas;
        this.sueldo_bruto = sueldo_bruto;
    }
}
```


Ejercicio 17: Laberinto

Realiza la refactorización del siguiente programa

Dificultad: ★★★★★

El siguiente programa, resuelve un laberinto, indicando en que dirección tienes que moverte para llegar a la salida:
Refactorízalo.

```
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.LinkedList;

class Maze {
    public int[][] maze;
    public LinkedList<Position> path = new LinkedList<Position>();
    public Position start;
}

class Position {
    public int x;
    public int y;

    public Position(int y, int x) {
        this.x = x;
        this.y = y;
    }
}

public class MazeSolver {
    // 0 = wall
    // 1 = path
    // 2 = destination

    static int[][] maze = {
        { 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
        { 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0 },
        { 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1 },
        { 1, 1, 1, 2, 0, 1, 0, 1, 0, 1, 0 },
        { 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0 },
        { 0, 0, 0, 1, 1, 1, 1, 1, 0, 1 }
    };

    static LinkedList<Position> path = new LinkedList<Position>();

    public static void main(String[] args) throws FileNotFoundException {
        solveMaze();
        for (Position p : path) {
            System.out.println(p.x + " " + p.y);
        }
    }
}
```

```

}

public static boolean solveMaze() throws FileNotFoundException {

    Position p = new Position(4, 8);
    path.push(p);

    while (true) {
        int y = path.peek().y;
        int x = path.peek().x;

        maze[y][x] = 0;

        // down
        if (isValid(y + 1, x)) {
            if (maze[y + 1][x] == 2) {
                System.out.println("Moved down. You won");
                return true;
            } else if (maze[y + 1][x] == 1) {
                System.out.println("Moved down");
                path.push(new Position(y + 1, x));
                continue;
            }
        }

        // Left
        if (isValid(y, x - 1)) {
            if (maze[y][x - 1] == 2) {
                System.out.println("Moved left. You won");
                return true;
            } else if (maze[y][x - 1] == 1) {
                System.out.println("Moved left");
                path.push(new Position(y, x - 1));
                continue;
            }
        }

        // up
        if (isValid(y - 1, x)) {
            if (maze[y - 1][x] == 2) {
                System.out.println("Moved up. You won");
                return true;
            } else if (maze[y - 1][x] == 1) {
                System.out.println("Moved up");
                path.push(new Position(y - 1, x));
                continue;
            }
        }

        // right
        if (isValid(y, x + 1)) {
            if (maze[y][x + 1] == 2) {
                System.out.println("Moved right. You won");
                return true;
            } else if (maze[y][x + 1] == 1) {

```

```

        System.out.println("Moved right");
        path.push(new Position(y, x + 1));
        continue;
    }
}

path.pop();
System.out.println("Moved back");
if (path.size() <= 0) {
    return false;
}
}

}

public static boolean isValid(int y, int x) {
    if (y < 0 ||
        y >= maze.length ||
        x < 0 ||
        x >= maze[y].length) {
        return false;
    }
    return true;
}
}

```

Ejercicio 18: Carta

Dada la siguiente clase que modela el comportamiento de un famoso juego de cartas. Se pide refactorizar la misma para obtener un código más limpio.

Dificultad: ★★ ★

```

public class Carta {
    String nombre = "default"; // nombre de la carta
    String descripcion; // Breve descripción de la carta
    boolean bloqueada = false; // Determina si la carta está bloqueada
    int danno = 100; // Daño que produce la carta al atacar
    int ps = 10; // Puntos de vida
    TipoCarta tipo = TipoCarta.Planta; // Tipo de carta (planta, fuego, agua,
    electrico)

    // Datos de los poderes (pueden ser varios) que contiene esta carta. Su
    nombre y

```

```

// su valor
String nombresPoderes[];
int valoresPoderes[];

int voltaje = 5; // Cantidad de voltaje, usado para las cartas de tipo
Electrico

enum TipoCarta {
    Planta, Fuego, Agua, Electrico
};

// Calcula si la carta se puede usar
boolean estaCao() {
    if (ps < 0 || ps == 0) {
        if (bloquedada) {
            return true;
        } else {
            return true;
        }
    } else {
        if (bloquedada) {
            return true;
        } else {
            return false;
        }
    }
}

// Calcula el daño de una carta
float calcularDannoDeLaCartaQueSeVaAAplicarCuandoAtaca() {
    switch (tipo) {
        case Planta:
            if (estaCao())
                return 0;
            return danno / 2;
        case Fuego:
            if (estaCao())
                return 0;
            return danno * 2;
        case Agua:
            if (estaCao())
                return 0;
    }
}

```

```

        return danno;
    case Electrico:
        if (estaCao())
            return 0;
        if (voltaje <= 0)
            return 0;
        return danno * voltaje * 0.1f;
    default:
        return 0;
    }
}

public static void main(String[] args) {

    Carta c = new Carta();
    c.tipo = TipoCarta.Electrico;

    System.out.println(c.calcularDannoDeLaCartaQueSeVaAAplicarCuandoAtaca());

}
}

```

Ejercicio 19: Encripta

Refactoriza el siguiente código fuente.
Si puedes optimizar algo, hazlo también.

Dificultad: ★★ ★

```

class Encripta {
    public String encripta(String textoClaro) {
        String result = "";
        for (int i = 0; i < textoClaro.toUpperCase().length(); i++) {
            if (textoClaro.toUpperCase().charAt(i) == 'A') {
                result += '4';
            } else if (textoClaro.toUpperCase().charAt(i) == 'B') {
                result += '8';
            } else if (textoClaro.toUpperCase().charAt(i) == 'C') {
                result += '(';
            }
        }
    }
}

```

```
} else if (textoClaro.toUpperCase().charAt(i) == 'D') {
    result += ')';
} else if (textoClaro.toUpperCase().charAt(i) == 'E') {
    result += '3';
} else if (textoClaro.toUpperCase().charAt(i) == 'F') {
    result += '=';
} else if (textoClaro.toUpperCase().charAt(i) == 'H') {
    result += '-';
} else if (textoClaro.toUpperCase().charAt(i) == 'I') {
    result += '3';
} else if (textoClaro.toUpperCase().charAt(i) == 'J') {
    result += '1';
} else if (textoClaro.toUpperCase().charAt(i) == 'K') {
    result += 'C';
} else if (textoClaro.toUpperCase().charAt(i) == 'L') {
    result += '7';
} else if (textoClaro.toUpperCase().charAt(i) == 'M') {
    result += 'W';
} else if (textoClaro.toUpperCase().charAt(i) == 'N') {
    result += 'n';
} else if (textoClaro.toUpperCase().charAt(i) == 'O') {
    result += '0';
} else if (textoClaro.toUpperCase().charAt(i) == 'P') {
    result += '6';
} else if (textoClaro.toUpperCase().charAt(i) == 'Q') {
    result += '@';
} else if (textoClaro.toUpperCase().charAt(i) == 'R') {
    result += '-';
} else if (textoClaro.toUpperCase().charAt(i) == 'S') {
    result += '$';
} else if (textoClaro.toUpperCase().charAt(i) == 'T') {
    result += '+';
} else if (textoClaro.toUpperCase().charAt(i) == 'U') {
    result += 'V';
} else if (textoClaro.toUpperCase().charAt(i) == 'V') {
    result += 'U';
} else if (textoClaro.toUpperCase().charAt(i) == 'W') {
    result += 'M';
} else if (textoClaro.toUpperCase().charAt(i) == 'x') {
    result += '%';
} else if (textoClaro.toUpperCase().charAt(i) == 'Y') {
    result += 'I';
}
```

```
        } else if (textoClaro.toUpperCase().charAt(i) == 'Z') {
            result += '>';
        } else {
            result += '#';
        }
    }
    return result;
}

public static void main(String[] args) {
    Encripta e = new Encripta();
    System.out.println(e.encripta("sala"));
}
}
```