Solución: Cuaderno de ejercicios: javascript

Ejercicio 1:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
public class Book {
	String ISBN;
	String title;
	String[] authors;
	String editorial;
	//Edition[] editions;
	ArrayList<Edition> editions;

	public Book(String ISBN) {
		this.ISBN=ISBN;
	}

	int countAuthors() {
		return authors.length;
	}
	void addNewEdition(Edition edition) {
		//Edition[] newArray = Arrays.copyOf(editions,
```

```
editions.length+1);
            //newArray[editions.length] = edition;
            //editions = newArray;
            editions.add(edition);
      }
}

class Edition{
      int id;
      String name;
      Date data;

      public Edition(int id, String name, Date data) {
            this.name = name;
            this.data = data;
            this.id = id;
      }
}
```

Ejercicio 2:

```
abstract public class Car {
      String model;
      String brand;
      String registration;
      abstract boolean validateRegistration();
}

class NewCar extends Car{
      @Override
      boolean validateRegistration() {
            return registration.length() == 7;
      }
}

class OldCar extends Car{
      @Override
```

```
	boolean validateRegistration() {
		return registration.length() == 8;
	}
}

class SuperNewCar extends Car{
	@Override
	boolean validateRegistration() {
		return registration.length() == 10;
	}
}
```

Ejercicio 3:

```
public class User {
	public String name;
	private final static float LIBRARY_COST = 10;
	public User(String name) {
		this.name=name;
	}
	public float costOfLibrary() {
		return LIBRARY_COST;
	}
}

class Teacher extends User{
	private final static float LIBRARY_COST = 2;
	public Teacher(String name) {
		super(name);
	}
	public float costOfLibrary() {
		return LIBRARY_COST;
	}
}

class HeadMaster extends User{
	private final static float LIBRARY_COST = 0;
	public HeadMaster(String name) {
		super(name);
	}
	public float costOfLibrary() {
		return LIBRARY_COST;
	}
}
```

Ejercicio 4:

```
import java.security.InvalidParameterException;
import java.util.Date;

class Subject{
	String name;
	String address;
	String cif;
}
class Article{
	String name; float price;
}
public class Bill {
	String cod;
	Date fecha;
	// float total;
	Subject subject;
	Article[] articles; // TOFIX: Mejor si usamos un ArrayList

	public float calculateTotal() {
		float total = 0;
		for(Article article: articles) { total += article.price; }
		return total;
	}

	// KISS
	public static String getNameDayOfWeek(int numberDay) throws
InvalidParameterException{
		if(numberDay < 1 ||  numberDay > 7 ) {
			throw new InvalidParameterException("Numero de día
invalido");
		}
		String[] days = {"Lunes", "Martes", "Miercoles", "Jueves",
"Viernes", "Sabado", "Domingo"};
		return days[numberDay-1];
	}
}
```

Ejercicio 5:

```java
public class Person {
	String name;
	String surname;
	String direction;

	private String getInitial(String text) {
		return text.toLowerCase().trim().charAt(0)+"";
	}

	public String getInitials(){
		String firstLetterOfName = getInitial(name);
		String firstLetterOfSurname = getInitial(surname);
		// System.out.println( firstLetterOfName + " " +
firstLetterOfSurname );
		return firstLetterOfName + " " + firstLetterOfSurname;
	}
	public void printInitialsOnScreen() {
		System.out.println(getInitials());
	}
}
```

Testing:

```java
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class PersonTest {

	@Test
	void test() {
		Person person = new Person();
		person.name="Pepe";
		person.surname="Gomez";
		String initals = person.getInitials();
		assertEquals("p g", initals);
	}

}
```

Ejercicio 6

```
interface Movable{
	public void move();
}

class BipedRobot extends Robot implements Movable{
	int[] legs;
	public BipedRobot(String name) {
		super(name);
		numLegs = 2;
		legs = new int[numLegs];
	}
	public void move() {
		legs[0]+=1;		legs[1]+=1;
	}
}
class CuadrupedRobot extends Robot implements Movable{
	int[] legs;
	int leg1;int leg2;int leg3;int leg4;
	public CuadrupedRobot(String name) {
		super(name);
		numLegs=4;
		legs = new int[numLegs];
	}
	public void move() {
		legs[0]+=1; legs[1]+=1; legs[2]+=1; legs[3]+=1;
	}
}
class HexapodRobot extends Robot implements Movable{
	int[] legs;
	//int leg1;int leg2;int leg3;int leg4;int leg5;int leg6;
	public HexapodRobot(String name) {
		super(name);
		numLegs=6;
		legs = new int[numLegs];
	}
	public void move() {
		legs[0]+=1; legs[1]+=1; legs[2]+=1; legs[3]+=1; legs[4]+=1;
legs[5]+=1;
	}
}
class TurretRobot extends Robot{
	public TurretRobot(String name) {
		super(name);
	}
```

```
}


abstract public class Robot {
	public String name = "";
	protected int numLegs;
	public Robot(String name) {
		this.name=name;
	}

	static void destroyRobot(Robot r) {
		System.out.println("El robot se destruido:"+r.name+"
"+r.numLegs);
	}
	static void jumpRobot(Movable r) {
		r.move();
	}
	public static void main(String[] args) {
		BipedRobot terminator = new BipedRobot("t800");
		CuadrupedRobot bostom = new CuadrupedRobot("atlas");
		TurretRobot aniquilator = new TurretRobot("ani");
		destroyRobot(bostom);
		jumpRobot(terminator);
		//jumpRobot(aniquilator);
	}
}
```

Ejercicio 7

Corrige el código para que sea código limpio.

```
public class CoffeeMaker {
	private boolean validateCoffeeType(String coffeeType) {
		return ! coffeeType.equals("");
	}
	private boolean validateTemp(float temp) {
		return temp>0 && temp<100;
	}
	private boolean validateWaterAmount(float waterAmount) {
		return waterAmount > 10;
	}
	public boolean makeCoffee(String coffeeType, float temp, float
waterAmount) throws Exception {
```

```
            if( ! validateCoffeeType(coffeeType)) throw new
Exception("tipo de café incorrecto");

            if( ! validateTemp(temp)) throw new Exception("temperatura
incorrecta");

            if( ! validateWaterAmount(waterAmount)) throw new
Exception("nivel agua incorrecto");

            // hacer café
            return true;
      }

}
```

Ejercicio 8:

```
public class EjemploLimpio {

      public static int sumar(int x, int y) {
            return x+y;
      }
      public static int elevarCuadrado(int x) {
            return x*x;
      }
      public static void main(String[] args) {
            int resultado = sumar(3,4);
            System.out.println(resultado);
            resultado = elevarCuadrado(4);
            System.out.println(resultado);
      }

}
```

Ejercicio 9:

```
abstract class Vehiculo{
	public Vehiculo(int vmax) {
		this.velocidadMaxima = vmax;
	}
	//Velocidad actual
	int velocidadActual;
	//Velocidad maxima
	int velocidadMaxima = 100;
	abstract void arrancar();
}
abstract class Barco extends Vehiculo{
		public Barco(int vmax) { super(vmax);	}
		//abstract void arrancar();
}

abstract class Coche extends Vehiculo{
	public Coche(int vmax) { super(vmax); }
	//abstract void arrancar();
}
class Peugeot extends Coche{
		public Peugeot() {	super(110);	}
		void arrancar() {
			velocidadActual=10;
		}
}
class Ferrari extends Coche{
		public Ferrari() {	super(320);	}
		void arrancar() {
			velocidadActual=10;
		}
}

public class EjemploLimpio {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

Ejercicio 10:

```
import java.util.Date;


class AsignaturaSuc{
	String nombre;
	String profesor;
	String horas;
}
class AlumnoSuc{
	String nombre;
	String apellido1;
	String apellido2;
	Date fechaAlta;
	AsignaturaSuc asignatura;

	public AlumnoSuc(String nombre, String apellido1, String apellido2,
Date fechaAlta, AsignaturaSuc asignatura) {
		super();
		this.nombre = nombre;
		this.apellido1 = apellido1;
		this.apellido2 = apellido2;
		this.fechaAlta = fechaAlta;
		this.asignatura = asignatura;
	}
}
public class EjemploLimpio {

	public static void matricular(AlumnoSuc alumno) {
		// acceso a la base datos
		// inserta
	}

	public static void main(String[] args) {
		AlumnoSuc alumno = new AlumnoSuc("Jose", "Gonzalez", "Ruiz",
new Date(), new AsignaturaSuc());
		matricular(alumno);
	}
}
```

Ejercicio 11:

```
import java.util.Arrays;
public class EjemploLimpio {

	public static int[] ordenarArrayAscendente(int [] datos) {
		Arrays.sort(datos);
		return datos;
	}
	public static void pintarArrayEnPantalla(int [] datos) {
		for(int dato: datos) {
			System.out.println(dato);
		}
	}
	public static void main(String[] args) {
		int datos[] = {2,9,3,4,1};

		datos = ordenarArrayAscendente(datos);

		pintarArrayEnPantalla(datos);
	}

}
```

Ejercicio 12:

```
abstract class premio {
	abstract float getCosteMaterial();
	abstract float getCosteTipo();
	float getImporte(){
		return getCosteMaterial() * getCosteTipo() * 0.5f;
	}
}
class PinOro extends premio{
	@Override	float getCosteMaterial() { return 1000; }
	@Override float getCosteTipo() { return 2; }
}
class PinPlata extends premio{
	@Override	float getCosteMaterial() { return 100; }
	@Override float getCosteTipo() { return 2; }
}
class CopaOro extends premio{
```

```
	@Override	float getCosteMaterial() { return 1000; }
	@Override float getCosteTipo() { return 2000; }
}
class PinPlatino extends premio{
	@Override	float getCosteMaterial() { return 2000; }
	@Override float getCosteTipo() { return 2; }
}
//....  9 en total

public class EjemploSucio {

	public static void main(String[] args) {
			PinPlata p = new PinPlata();
			System.out.println(p.getImporte());

	}

}
```

Ejercicio 13

1. Documentarlo
2. Crear los tests
3. Refactorizarlo (código limpio)
4. Optimizarlo

TEST

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class DocumentoIdentificativoTest {

	@Test
	void testGetLetter() {
		DocumentoIdentificativo doc = new Nif();
		doc.text="36765546N";
		assertEquals("N", doc.getLetter());
	}
```

```
	@Test
	void testIsValid() {
		DocumentoIdentificativo doc = new Cif();
		doc.text="B34455667";
		assertEquals(true, doc.isValid());
		assertTrue(doc.isValid());
	}

}
```

CÓDIGO

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

/**
 * Clase que define un documento de tipo CIF o NIF
 * @author Angel
 * @version 1.0
 *
 */
abstract public class DocumentoIdentificativo {
	/**
	 * Texto del cif o nif
	 */
	String text;

	/**
	 * Obtiene la letra alfabetica del documento
	 * @return La letra
	 */
	abstract String getLetter();

	/**
	 * Valida si el documento es correcto
	 * @return un booleano indicando si es correcto
	 */
	abstract boolean isValid();

}

class Nif extends DocumentoIdentificativo{
```

```
	String getLetter() {
		char temp = text.charAt(text.length()-1);
		return String.valueOf(temp);
	}
	boolean isValid() {
		//TODO falta por hacer
		return false;
	}
}

class Cif extends DocumentoIdentificativo{
	String getLetter() {
		char temp =  text.charAt(0);
		return String.valueOf(temp);
	}
	boolean isValid() {
		// empezar por letra(A-Z) y luego tener 8 numeros
		Pattern pattern = Pattern.compile( "^[A-Z]\\d{8}$" );
		Matcher matcher = pattern.matcher(text);
		return matcher.matches();
	}

}
```