EJERCICIOS DE JAVA

Diversión para todo el mundo

Con cariño de tu querido profe
ANTONIO SANTOS RAMOS

UML	Pag 01
Java básicos	Pag 02
Java complejos	Pag 09
BBDD (E/R)	Pag 13
BBDD (Mysql)	Pag 15
Java Web	Pag 18

Normas para los ejercicios

1. Guardar las soluciones a los ejercicios con esta nomenclatura: 01-HolaMundo, 02-Coche, etc.
2. Un ejercicio está finalizado cuando se construye un ejemplo y se comprueba su funcionamiento
3. Un ejercicio se puede hacer en una hora, en 15 minutos o en 42 horas. Tú decides, pero es mejor hacer muchos ejercicios que uno divino.
4. Nivel de dificultad
    a. Chupao
    b. Tiene algo *
    c. Tengo que pensarlo **
    d. Necesito un ron ***

EJERCICIOS POO (HOJA 01)

[UML. Diagramas de clase]

01	Crear la clase **Coche**, indicando 4 características y 10 métodos que pueda realizar
02	(*) Realizar la clase **Cafetera** que permita guardar la capacidad máxima de café y la cantidad actual
03	Diseñar el diagrama de clases de un **taller**. Dicho taller solo arregla coche (de los cuales debe especificarse el motor) y motos (de las cuales debe especificarse la cilindrada). Además, de cada coche/moto, se guarda el color y la marca
04	(*) Un **banco** dispone de distintos productos bancarios: cuentas y préstamos. De cada elemento guardado, es necesario conocer el saldo disponible, el cliente y el código de producto El banco ha lanzado una serie de cuentas y el primer producto lanzado es la CuentaJoven Las hipotecas se asocian a un préstamo
05	(**) Crea la clase **COCHE** y enlázala con las siguientes clases • Motor: con métodos para arrancar el motor y apagarlo. • Rueda: con métodos para inflar la rueda y desinflarla. • Ventana: con métodos para abrirla y cerrarla. • Puerta: con una ventana y métodos para abrir la puerta y cerrar la puerta.
06	(*) Desarrollar la clase **Canción**, **CDCanciones**, **ListaMusica**, **Reproductor** y relacionarlas
07	Crear la clase **Fecha** y diseñar los métodos que se vean necesarios
08	Una empresa dispone de **empleados** temporales, fijos y trabajadores Freelance que trabajan por horas. ¿Cómo se pueden modelar estas clases?
09	(**) Modelar el uso de un **microondas**
10	(***) En un **puerto** se alquilan amarres para barcos. Para cada alquiler se guarda el nombre y DNI del cliente, la fecha inicial y final del alquiler, la posición del amarre y el barco que lo ocupará. Un barco se caracteriza por su matrícula, su eslora en metros y año de fabricación. Un alquiler se calcula multiplicando el número de días de ocupación (incluyendo los días inicial y final) por un módulo función de cada barco (obtenido simplemente multiplicando por 10 los metros de eslora) y por un valor fijo (155€ en la actualidad). Además, se pretende diferenciar la información de algunos tipos de barcos: • número de mástiles para veleros • potencia en CV para embarcaciones deportivas a motor • potencia en CV y número de camarotes para yates de lujo.
11	(**) En la **biblioteca** de un colegio hay libros, comics y revistas. Interesa guardar siempre la fecha de publicación y el nombre. Además, de los libros se debe guardar la edición, el ISBN y el autor (que es único); de las revistas el número y la periodicidad y de los comics el número y la colección. El préstamo se puede realizar a profesores y alumnos y se debe guardar en el mismo cuando se ha realizado el préstamo y cuando se ha devuelto el libro.
12	(**) Diseñar una **calculadora**
13	(*) Realizar el diagrama de clases de: Elipse, FiguraCerrada, Pentágono. Cuadrilátero, Círculo, Polígono, Rectángulo, Rombo
14	(***) Diseñar el uso de un **cajero**

EJERCICIOS JAVA (HOJA 01)

[Clases, constructores, toString, static, Herencia]

01	Realizar un programa que escriba “Hola Mundo” (o “hola ke ase”... como prefieras)
02	Crear la clase **Coche** a) Guardar el modelo, color, si la pintura es metalizada, matrícula, el tipo de coche (utilitario, familiar, deportivo), año de fabricación y la modalidad del seguro (terceros o todo riesgo). b) Redefinir el método toString() para imprimir los datos
03	Escribir un programa que represente las asignaturas de una **carrera**. Una asignatura tiene un nombre, un código numérico y el curso en el que se imparte. El programa debe imprimir los datos de la asignatura.
04	Diseñar a) (*) La clase **Punto** (definida por dos coordenadas en el espacio) b) (**) Un método que permita calcular la distancia entre dos puntos c) (*) Un método que permita calcula la distancia de un punto al centro de coordenadas
05	Diseñar a) La clase “**ElementoTablaPeriodica**” que guarda el nombre, símbolo y número. b) El método “numeroDeElementos()” que indique cuantas instancias se han creado.
06	(*) Realizar un programa que a) Permita generar **empleados**: nombre, apellido, dirección, DNI, mail b) Permita generar **Departamentos**: nombre, ubicación, teléfono c) (***) Permita relacionar el departamento al cual se asigna el empleado y el director de cada departamento (que es también un empleado). Haz una prueba para imprimir los datos.
07	(*) Diseñar a) La clase **Rectángulo** b) Un método que pueda calcular el área de un rectángulo a partir de los datos proporcionados por teclado
08	(*) Diseñar a) La clase **Triangulo** b) Un método tal que dado un triángulo indique cuánto mide el lado más largo c) Un método tal que dado por teclado los atributos, indique cual es el lado más largo
09	(**) Se quiere diseñar una clase para llevar las notas de un **curso** de Java. Crear las clases necesarias para poder guardar la información de alumnos, profesor, asignaturas y las notas del examen realizado por cada alumno en esa asignatura. NOTA: Realiza la prueba con 2 alumnos, 2 profesores y 2 asignaturas.
10	(*) Diseñar a) La clase “**Cuenta Corriente**” que tiene un identificador, un cliente y permite definir la cantidad depositada. b) Un método para realizar ingresos. Un ingreso se define por el ordenante, la cantidad a ingresar y el número de cuenta. c) (***) La Cuenta Corriente dispondrá de unos intereses del 3% anual. Teniendo en cuenta que al abrir la cuenta se marca la fecha. Indicar los beneficios a día de hoy. Ej.- La cuenta se ha abierto el día 01/01/2020 con 100€. Hoy es 01/01/2021. El beneficio sería de 3€.

EJERCICIOS JAVA (HOJA 02)

[Condiciones y bucles]

11	Definir un método "**mayorDeEdad**" que, dada una edad, indique si la persona es mayor de edad (definir la mayoría de edad como una constante).
12	Escribir un método que pida un número entero e indique si es par o **impar**
13	Escribir el **abecedario** utilizando un bucle a) WHILE b) FOR c) DO… WHILE
14	(*) Dado un número n, realizar un programa que sume los primeros n números **pares**. a) para n = 7 (tope) → 2 + 4 + 6 b) para n = 7 (cantidad) → 2 + 4 + 6 + 8 +10 +12 +14
15	(**) Realizar un programa que valide una **edad** e indique un mensaje dependiendo de la edad. Ej.- • (0..4] pequeñín • (4..10] niño • (10..40] adolescente • (40 .. 80] talludito
16	(**) Realiza un simulador de **lotería primitiva** que permita varias apuestas
17	Dado un número n, realizar el **factorial** (Ej 5! → 5 * 4 * 3 * 2 * 1)
18	(*) El club de **golf** "El agujerito" quiere un programa para las cuotas de los socios. - Cuota inicial de 500 euros - Mayores de 65 años: descuento 50% - Mujeres: Descuento 15% - Menores de 18 años: descuento 25% (si el padre no es socio) o 35% (si el padre es socio) Si un usuario puede acceder a más de un descuento, sólo cogerá el mayor descuento. NOTA: Definir como constantes los porcentajes que se pueden descontar.
19	(*) Escribir un programa que pida un número de 1 a 10 y dibuje un triángulo con **asteriscos** Ej.- Para n= 4 a) * * * * * * * * * * b)(*) * * * * * * * * * * c) (**) *
20	(**) Escribir un programa que pida una cantidad de **euros** y los devuelva en monedas existentes (1 céntimo, 2 céntimos, 5 céntimos, etc.) Ej.- 5.35€ → 2 monedas de 2 euros + 1 moneda de 1 euro + 1 moneda de 20 céntimos + 1 moneda de 10 céntimos + 1 moneda de 5 céntimos.
21	(*) Escribir un programa que pida los **límites** de un rango (límite superior e inferior) y que a continuación vaya pidiendo números. Si el número se encuentra dentro de ese rango, escribirá el cubo de ese número.

EJERCICIOS JAVA (HOJA 03)

[Estructuras, Arrays, String, StringBuilder, enumerados]

22	Escribir una clase para trabajar con **arrays** (de tipo int) que incluya métodos estáticos para: a) Imprimir un array (separando los elementos por comas) b) Crear Arrays de un tamaño proporcionado (pidiendo datos) c) Indicar la suma de todos los números d) Devolver el array invertido
23	(** si lo haces con array) Escribir un programa que vaya recibiendo números y que cuando se escriba un 0 muestre los números escritos, la media aritmética, el mayor y menor y el número que más veces aparece(*** si es con Array)
24	(*) Realizar 100 tiradas de un **dado** e indicar la frecuencia de cada resultado
25	Escribir un programa que reciba un array de cadenas y devuelva un array de enteros con los tamaños de cada una de las cadenas.
26	Escribir un programa que dada una matriz 2D de enteros, **rellene** cada celda (primero la 1ª fila, luego la 2ª….) con el 0, 1, 2, 3, 4 … respectivamente
27	(***) Dada una **matriz** que representa un conjunto de puntos, diseñar un método que gire el conjunto de puntos 90º a la derecha.
28	(**) Escribir un método que reciba un array de string y devuelva otro array **ordenado** alfabéticamente
29	(*) Realiza un método que reciba un String y cambie “endeluego” por “desde luego”. Realiza dos versiones: una con String y otra con StringBuilder (** para todas las apariciones)
30	(*) Realizar un método para que, dado un texto, indique si ese texto es un **palíndromo**.
31	Una **heladería** vende copas de helado de tipo MINI (con 2 bolas), SUPER (con 3 bolas) y MEGA (con 5 bolas). Diseñar una estructura para trabajar.
32	Un túnel de **lavado** dispone de tres tipos de lavados: BÁSICO, NORMAL, SUPER. Los tiempos para cada tipo son 3’, 5’ y 7’ respectivamente. Diseñar una estructura mediante enumerados.
33	(*) Diseñar una estructura para poder trabajar con los **meses** del año
34	(**) Diseñar un **traductor** de idiomas que pida una palabra y el idioma a traducir • Alemán (vocal). Quitar vocal y añadir sufijo “ujem”: silla → sillujem • Alemán (consonante). Quitar últimas letras y añadir sufijo “ujem”: camión → camiujem • Búlgaro (vocal). Añadir sufijo “kov”: silla → sillakov • Búlgaro (consonante). Quitar consonante y añadir sufijo “kov”: camión → camiokov • Catalán. Quitar la última letra
35	(**) Escribir un método para que, dado un array de termino de un **diccionario** (nombre + definición), indique cuantas letras tiene cada palabra y cuantas palabras la definición
36	(*) Diseñar la clase **Futbolista** que tenga, entre sus atributos, dos enumerados. Uno de ellos para indicar el equipo de pertenencia y otro para indicar su posición en el campo
37	(*) Realizar un programa que pida dos frases y muestre en la salida las dos **frases mezcladas** cogiendo letras de forma alternativa Ej. Si las frases son HOLA CASA y RATO MALO la salida sería: HROALTAO CMAASLAO

EJERCICIOS JAVA (HOJA 04)

[Diseño aplicaciones, Interfaces, herencia, polimorfismo, colecciones, generics]

38	Diseñar la estructura **Estudiante** y Profesor de la forma más optimizada posible
39	Diseñar la clase **Pixel** (*) Diseñar la clase imagen BMP
40	(*) Diseñar la clase **Examen** a) que guarde la asignatura, el aula, la Fecha y la Hora. Para guardar estos dos atributos debe construirse las clases Fecha y Hora (usar java.time) b) Emplear esa clase Examen para guardar los exámenes de una asignatura
41	Diseñar una clase **Taza** que permita, de la forma más optimizada posible, trabajar con distintos materiales (cristal, porcelana, etc.) ¿Cómo se puede indicar que algunos elementos de la vajilla podrían romperse?
42	(*) Realizar un método tal que dado un array (con datos primitivos) lo **convierta** a un ArrayList. Realizar lo mismo para que funcione con cualquier tipo de objetos (**)
43	(***) Realiza un sistema que permita **relacionar** circulo, rectángulo, dibujable, geometría, rectángulo grafico (con color), sol, folio, campo de futbol, tablero ajedrez
44	(*) Escribir un método "**frecuencia**" que indique cuantas veces aparece un elemento `<E> int frecuencia(Collection<E> col, E elem)`
45	(***) Una empresa está teniendo mucho éxito con la venta de **polígonos**. Los polígonos los hacen de distintos materiales, principalmente cartón, plástico y madera. Cada polígono tiene un precio, 100, 300 y 500 euros respectivamente… todo un chollo, oiga. Los polígonos se encuentran formados de puntos. Este año se llevan mucho los polígonos coloreados. Se diferencian de los polígonos normales porque llevan color. Valen un 15% más. Para la colección primavera-verano 2020 se han sacado al mercado pentágonos de color con firmas de los principales diseñadores… va a ser un pelotazo. Y sólo valen un 20% más. Cómprate uno ya!!!
46	(**) Un **aviso** se identifica por un número de orden, una fecha, un nivel de severidad (LEVE, GRAVE, CRITICO) y la descripción del mismo. Dada una colección de avisos a) Realizar un método para imprimir los avisos críticos b) Realizar un método para imprimir los mensajes del año actual c) Realizar un método para imprimir los mensajes que contengan la palabra ERROR
47	(***) Diseñar una estructura de **Pila** que incluya los métodos • estaVacia() • mostrarPrimero() • imprimirYVaciar() • estaLlena() • cantidadElementos() • extraerPrimero() • incluir()
48	(**) Pensar cual sería la mejor forma para guardar datos de • Notas de un curso • Librería • Parking • Biblioteca pública • Parking japo • Mi Biblioteca

EJERCICIOS JAVA (HOJA 05)

[Excepciones]

<table>
<tr><td>49</td><td>Realizar un programa que pida dos números entre 0 y 10 y los divida</td></tr>
<tr><td>50</td><td>Completar la clase “Punto” (Ej04) añadiendo una excepción para indicar si el punto es NULL</td></tr>
<tr><td>51</td><td>Realizar un ejercicio en el cual se escoja un número en un rango
Se pida un número entre 1 y 100
Si el número no pertenece a ese rango se captura la excepción y se vuelve a pedir
Si el número es correcto se indica el número de intentos hasta conseguirlo</td></tr>
<tr><td>52</td><td>(*) Realizar un ejercicio en el cual se adivine un número
El ordenador calcula un número entre 1 y 100
Se pide un número entre 1 y 100
Si el número no es correcto se tramita la excepción y se le indica si es mayor o menor
Si el número es correcto se indica en número de intentos hasta conseguirlo</td></tr>
<tr><td>53</td><td>(**) Dispongo de la clase atleta que tiene como atributos el nombre y la energía.
Al correr el objeto “Corredor” decrementa de 10 en 10 su energía hasta llegar al valor cero.
Crear un entrenamiento con las siguientes características
Crea un objeto de la clase “Corredor”. Ese objeto tiene energía 50.
Se hace correr al corredor hasta que agote su energía 3 veces (es decir, llega a 0).
La primera vez que se agote → reponer la energía a 30
La segunda vez que se agote → reponer la energía a 10
La tercera vez que se agote → Se acabó. Tomar un Aquarius</td></tr>
<tr><td>54</td><td>(*) Realizar una alarma que se pueda activar para un periodo concreto de segundos
Cuando llega a cero emite un aviso</td></tr>
<tr><td>55</td><td>(*) Para un Parque crear dos métodos que, dada una edad, indiquen si el usuario puede acceder

<table><tr><td></td><td>montañaRusa(edad)</td><td>castilloHinchable(edad)</td></tr><tr><td>Lanzar excepción “EdadNegativaException”</td><td>Si la edad < 0</td><td>Si la edad < 0</td></tr><tr><td>Lanzar excepción “DemasiadoMayorException”</td><td>Si la edad > 70</td><td>Si la edad > 10</td></tr><tr><td>Lanzar excepción “DemasiadoJovenException”</td><td>Si la edad < 5</td><td>Si la edad < 3</td></tr></table>
a) Realizar los métodos. En el resto de casos dejar montar en la MontañaRusa o en el CastilloHinchable respectivamente
b) Realizar un método que permita hacer una prueba proporcionando un array de valores P{-2, 19, 69, 95, 1, 4, 2, 7}</td></tr>
<tr><td>56</td><td>(**) Realizar una versión mejorada del ejercicio 46</td></tr>
<tr><td>57</td><td>(*) Realizar una versión mejorada del ejercicio 15</td></tr>
</table>

EJERCICIOS JAVA (HOJA 06)

[Ficheros. Entrada/salida]

58	Realizar un programa que lea del teclado líneas y haga un "eco". (**) Si fuera un número, debe incrementarlo en una unidad y mostrarlo en pantalla incrementado
59	Diseñar un programa que escriba en un fichero de texto el alfabeto
60	Realizar un programa que lea datos de un fichero de texto e indique cuantos de los elementos leídos son números o palabras
61	Dado un archivo de texto, indicar en pantalla el número de letras, el número de palabras, el número de mayúsculas y el número de líneas del texto incluido en el fichero
62	(*) Realiza un ejercicio que lea de un fichero y, mediante excepciones, cuente cuantos números, blancos, consonantes y vocales existen.
63	(*) Dado un fichero de texto con una lista de palabras desordenadas, escribirlas de forma ordenada en un segundo fichero. Controla que ambos ficheros existen
64	Realizar un método al cual se le pasen como argumento dos ficheros binarios y realice una copia del primero en el segundo
65	(*) Realizar un método que sea capaz de escribir en pantalla el listado del directorio actual. Para cada archivo mostrar sus propiedades y atributos principales.
66	(**) Realizar un programa que lea una serie de números de un fichero y los escriba en otro fichero como números romanos.
67	(**) Realizar una variación sobre alguno de los ejercicios del curso para que lea datos de prueba de un fichero y los guarde modificados

EJERCICIOS JAVA (HOJA 07)

[Concurrencia]

68	Realizar un programa que una vez que haya pasado un segundo escriba en pantalla un mensaje
69	Simular tres threads lanzados a la vez y que se duerman un tiempo aleatorio. Imprimir en pantalla un mensaje cada vez que se despierten. Modificar para que el programa principal termine una vez que hayan acabado los hijos
70	(*) Realiza un productor-consumidor en el cual se incluya un almacén sincronizado
71	(**) Adaptar el productor consumidor para que puedan almacenar 5 elementos. Se vacía una vez que se encuentre completo
72	(***) Realiza el ejercicio de la cena de los filósofos http://es.wikipedia.org/wiki/Problema_de_la_cena_de_los_filósofos
73	(***) Realiza el ejercicio del peluquero Dormilón. http://es.wikipedia.org/wiki/Problema_del_barbero_durmiente

EJERCICIOS JAVA (ESPECIALES)

E01 - EMPLEADOS

[Uso de Herencia, encapsulación]

En una aplicación creada para una empresa, se quieren guardar los datos de los empleados: nombre, salario, fecha de Nacimiento.

Los gerentes son empleados que además se hacen cargo de un departamento de la empresa. Su método "incentivo" multiplica el salario por 1.05

Cuando un gerente llega a director se le asigna un coche de empresa. La aplicación guarda la matrícula de ese coche. Su método "incentivo" multiplica el salario por 1.10 y le incrementa 100

Comprobar que las clases creadas funcionan y revisar si el método de incentivación funciona. Para verlo, crea un array de 5 empleados: 3 empleados normales, 1 gerente, 1 director.

E02 - KENIA

[Clases abstractas, interfaces]

Se pide desarrollar un programa que permita gestionar todos los parques nacionales que existen en Kenia.

Cada parque posee una extensión (km^2), un número de especies de animales y un nombre.

Un parque puede ser de uno de estos dos tipos: las Reservas de Caza o Áreas Protegidas.

- Los primeros, tienen un coste de licencia y un sólo tipo de arma a utilizar en él
- Los segundos, poseen una subvención del gobierno (en Chelines) y colabora en ellos una ONG.

Las Áreas Protegidas pueden ser acuáticas (incluyen número de lagos), terrestres (tipo de terreno) y el resto, que no se clasifican bajo ningún tipo específico.

Se pide:

a) Crear 2 Reservas de Caza y 3 Áreas Protegidas (una de cada) e insertarlas en un array.
b) Mostrar toda la información de los parques por consola recorriendo dicho array.
c) Mostrar sólo el nombre de las áreas protegidas almacenadas.
d) Algunos parques son visitables.
    a. Crear un array de "zonas visitables".
    b. Hacer que las Áreas Protegidas sean visitables.
    c. Definir el método visitar(), que devolverá "Visitando área protegida XXXX".

E03 - BIBLIOTECA

[Colecciones, herencia, interfaces, encapsulación]

En la biblioteca hay libros, comics y revistas. Interesa guardar siempre la fecha de publicación y el nombre. Además, de los libros se debe guardar la edición, el ISBN y el autor (que es único); de las revistas el número y la periodicidad y de los comics el número y la colección.

Sólo presto mis libros y debo saber si un libro está prestado o no. Aunque de momento no lo necesito puede que el futuro quiera prestar otros elementos de mi biblioteca. ¿Utilizo una interfaz o herencia? ¿Qué métodos necesito?

Escribir la clase Disco -hereda de Multimedia (titulo (String), formato(enumerado), duración) teniendo en cuenta que los discos se pueden prestar.

Diseñar el sistema de biblioteca y crear un menú para las opciones

a) Número de libros prestados ¿Se podría reutilizar el método para discos?
b) Publicaciones anteriores a una fecha
c) Imprimir el listado de publicaciones y discos
d) Imprimir el listado de publicaciones y discos diferenciando por prestado o no

E04 - SUPERMERCADO

[Colecciones, herencia, interfaces]

El Supermercado online "La compra Alegre" acaba de abrir sus puertas y quiere disponer de un sistema informático para su utilización.

Para que funcione se quiere diseñar un backoffice para dar de alta los productos. El sistema debe disponer de un menú con las siguientes opciones

1) Dar de alta un producto
2) Eliminar un producto
3) Listar los productos existentes
4) Dado un producto, indicar las cantidades existentes
5) Listado de los productos con existencias menores a 5 unidades

De un producto siempre se deben guardar cuatro valores: Código (debe ser único), Nombre del producto, Categoría (frutas, lácteos, etc.), Cantidad existente del producto

Crear el programa correspondiente

E05 - JUKEBOX

[Herencia, enumerado, interfaces]

Desarrollar un programa que permita trabajar con dos tipos de ficheros de música.

- Tipo 1: se modelará en base a los siguientes requerimientos:
  - Se pretende realizar un programa que gestione la música en un formato determinado que poseemos en nuestra casa. Toda la música que tenemos, está en el mismo formato.
  - Como datos de interés estarán el propio formato de audio, el nombre del fichero, la calidad expresada en kbits/seg (int), una valoración (0 mala y 5 muy buena) y una categoría (pop, rock, melódica, etc.).
- Tipo 2: posee la misma información que el anterior, con la diferencia que además incluye el video-clip de la canción.

Cuando se reproduzca un fichero de música simple, se escuchará la canción, mientras que si se reproduce un archivo de tipo 2 se escuchará y, además, se podrá visualizar el video.

El menú del programa incluirá las siguientes opciones

1. **Listado De canciones**
2. **Canciones (solo música)**
3. **Canciones (sólo vídeos)**
4. **play (al azar)**
5. **Valoración media de mis canciones**
6. **play (sólo canciones de valoración 5)**
7. **play (solo rock)**
8. **play (de un listado)**

E06 - AGENDA

[Uso de concurrencia, E/S, encapsulación, colecciones]

Diseñar una agenda de teléfonos (y su menú) que permita incluir contactos, eliminarlos, buscarlos, listarlos (lista completa y lista a partir de una letra), y modificarlos.

Además, debe permitir agrupar los teléfonos (por ejemplo “familia”, amigos”) y poder listar los teléfonos de esos grupos.

E07 - PARKING

[Uso de Generics, ArrayList, Threads y E/S]

Diseñar una aplicación que modele la gestión de un parking.

En nuestro parking se permiten sólo tres tipos de vehículos (coches, motos y camiones). El sistema quiere almacenar de cada vehículo la matrícula, la marca, el modelo y color.

Adicionalmente, de cada uno de ellos, se quiere recoger estos datos:

- **COCHES**: número de puertas
- **MOTOS**: si tiene sidecar
- **CAMIONES**: tara

Nada más arrancar la aplicación se solicita el **número de plazas** (puede aparecer como constante). A continuación, se muestra al usuario un menú con las siguientes opciones (y siempre debe aparecer el número de plazas disponibles).

1. Añadir un nuevo vehículo
2. Eliminar un vehículo
3. Búsqueda de un vehículo (por matrícula)
4. Listado de vehículos aparcados
5. Listado de vehículos (por categorías de motos/coches/camiones)
6. Pagar parking (por matrícula)
7. Serializar parking
8. Salida de la aplicación
9. Leer datos de prueba (desde método)
0. Leer datos de prueba (desde fichero)

Simular el funcionamiento del parking (opciones 1, 2, 3, 4, 5, 8). Ten en cuenta que pueden entrar a la vez tanto coches, camiones como motos.

Modif 1) Modificar las opciones 4 y 5 para preguntar si se quiere obtener el listado en un fichero. De ser así, generar un fichero de texto y guardar la información

Modif 2) Añadir una opción adicional en el menú (Opción 6 del "menú") para obtener lo que debe pagar cada vehículo. Para calcular tiempo puedes emplear `System.currentTimeMillis();`

COCHE: 30 €/hora	MOTO: 20 €/hora	CAMIÓN: 50 €/hora

Modif 3) Añadir una opción adicional en el menú (Opción 0 del "menú") para cargar datos desde un fichero de texto externo

Modif 4) Añadir una opción adicional en el menú (Opción 7 del "menú") para serializar en un fichero binario el estado actual del parking.

Modif 5) Realiza un duplicado y crea una nueva versión que no tenga menú y en la cual se trabaje con thread (es decir, no haya interacción). Realiza estos casos

a) Imagina que tienes varios carriles de acceso al parking, pero sólo una barrera de entrada (y una de salida). Modifica el código empleando threads.
b) Añade varios carriles de salida pero una única barrera de salida.
c) Añade 3 barreras (una para coches, para motos y otra para camiones)

NOTA Para usar dinero emplea dos decimales:

```
import java.text.DecimalFormat;
DecimalFormat formateador = new DecimalFormat("#.##");
System.out.println (formateador.format(3.43242383)); //3,43
```

EJERCICIOS BB.DD. RELACIONALES

(para crear E/R y modelo relacional)

01

ALMACEN

Un almacén de venta de productos navideños quiere realizar una base de datos de cara a la nueva temporada 2030-2031.

a) Elaborar la BBDD para recoger los datos de productos y proveedores
b) (en otro diagrama) Ampliarla para incluir información de los clientes a los que vende sus productos y conocer que les ha vendido

02

CENSO DE LA VIVIENDA

En un municipio se quiere llevar un censo de las viviendas y personas que las habitan

Cada persona solo puede habitar una vivienda y estar empadronada en un municipio, pero puede ser propietaria de varias viviendas.

Interesa conocer también las personas que dependen del Cabeza de Familia (C.F).

03

FORMACIÓN A EMPRESAS

El departamento de formación de una empresa desea construir una base de datos para planificar y gestionar la formación de sus empleados.

La empresa organiza cursos internos de formación de los que se desea conocer el código del curso, el nombre, una descripción, el número de horas de duración y el coste del curso.

Un curso puede tener como prerrequisito haber realizado otro(s) previamente, y, a su vez la realización de un curso puede ser prerrequisito de otros. Un curso que es un prerrequisito de otro puede serlo de forma obligatoria o sólo recomendable.

Un mismo curso tiene diferentes ediciones, es decir, se imparte en diferentes lugares, fechas y con diferentes horarios (intensivo, de mañana o de tarde). En una misma fecha de inicio sólo puede impartirse una edición de un curso.

Los cursos se imparten por personal de la propia empresa.

De los empleados se desea almacenar su código de empleado, nombre, apellidos, dirección, teléfono, NIF, fecha de nacimiento, nacionalidad, sexo, firma y salario, así como si está o no capacitado para impartir cursos.

Un mismo empleado puede ser docente de una edición de un curso y alumno en otra edición, pero nunca puede ser ambas cosas a la vez (en una mima edición de curso o lo imparte o lo recibe).

04 **EMPRESA**

Queremos almacenar información sobre los trabajadores, departamentos y secciones departamentales de una empresa, para lo cual disponemos de los siguientes datos:

- De los trabajadores se necesita almacenar su Identificación, nombre, dirección, teléfono, y cónyuge (en caso de que este trabaje también en la empresa).
- De los departamentos se conoce su nombre (único), su único director (que es un trabajador), tarea asignada al mismo y localidad donde está ubicada la dirección.
- Los departamentos están subdivididos en secciones cada una con una subtarea específica. Cada sección tiene un director, que también es un trabajador de la empresa. De las secciones también se necesita conocer su nombre (único), localidades donde está ubicada y número mínimo de empleados.
- Una determinada sección sólo pertenece a un departamento.
- Los trabajadores pueden pertenecer a más de una sección, siempre y cuando no pertenezcan a más de una sección por departamento.
- Los trabajadores sólo pueden dirigir un departamento.
- Los trabajadores sólo pueden dirigir una sección.

05 **CENTRO ESCOLAR**

En un centro escolar se imparten numerosos cursos.

En cada uno de estos cursos se encuentran matriculados un grupo de alumnos de los cuales uno de ellos es el delegado del grupo. Los alumnos cursan asignaturas.

06 **CLÍNICA DENTAL**

Una clínica dental realiza una BBDD para guardar un historial electrónico de los pacientes, el tratamiento aplicado, el costo, el médico implicado en cada uno y las enfermeras que participaron.

Para cualquier operación hay un presupuesto previo

07 **CONCESIONARIO**

Un concesionario dispone de una serie de coches para su venta de los que se la matrícula, marca y modelo, el color y el precio de venta de cada coche.

Los datos a conocer de cada cliente son NIF, nombre, dirección, ciudad y teléfono. Tiene un código interno automático cuando un cliente se da de alta en ella.

Un cliente puede comprar tantos coches como desee a la empresa.

Un coche determinado solo puede ser comprado por un único cliente.

El concesionario también se encarga de llevar a cabo las revisiones que se realizan a cada coche. Cada revisión tiene asociado un código que se incrementa automáticamente por cada revisión que se haga. De cada revisión se desea saber si se ha hecho cambio de filtro, si se ha hecho cambio de aceite, si se ha hecho cambio de frenos u otros.

Los coches pueden pasar varias revisiones en el concesionario.

EJERCICIOS BB.DD. RELACIONALES (RESTAURANTE)

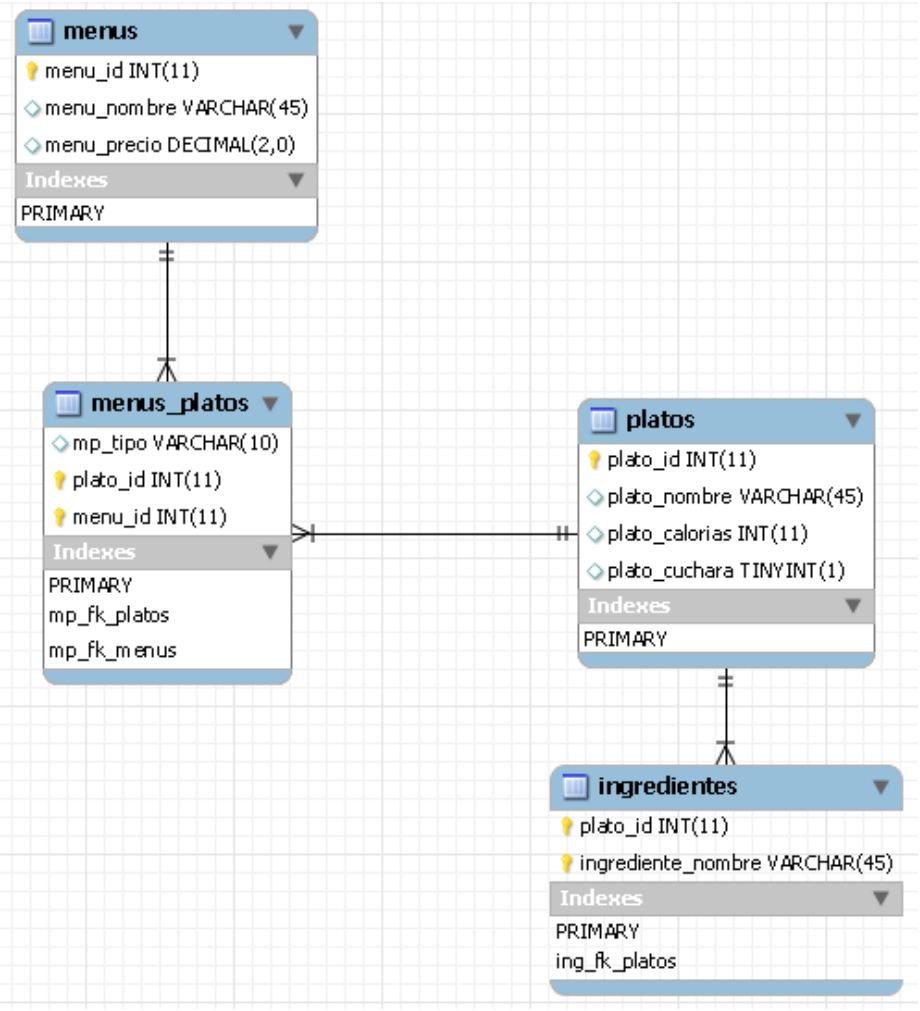
(Para Mysql)

NOTA
No están en orden de dificultad

1. Muestre todos los valores de todos los campos de la tabla platos
2. Muestre los platos y sus calorías de la tabla platos
3. Muestre los platos y sus ingredientes
    a. Solo de aquellos platos que tengamos ingredientes
    b. De todos los platos
4. Muestre:
    a. Sin usar la clausula distinct muestre los valores distintos hay en el campo menus_platos.mp_tipo
    b. Usando la clausula distinct muestre los valores distintos hay en el campo menus_platos.mp_tipo
    c. Repetir las dos anteriores dando los resultados en orden ascendente
5. Muestre:
    a. El nombre de los platos indicando si es un primero o un segundo o un postre
    b. Elimine repetidos
    c. Muestre solo los resultados de los primeros platos
    d. Ordénelos por nombre de plato
6. Muestre:
    a. Los menús, los platos entre los que se puede elegir en cada menú, y si es primero o segundo o postre
    b. Ordénelos por calorías
    c. Ordénelos primero por tipo (primero, segundo o postre) y luego por calorías
    d. Ordénelos primero por menú (descendente), luego por tipo (primero, segundo o postre) y luego por calorías
    e. Muestre solo los menús del lunes
7. Muestre los diez primeros valores de la tabla platos
8. Muestre el nombre de todos los menús que tengan paella
9. Muestre
    a. El nombre de todos los menús que tengan algún plato de más de 100 calorías
    b. El nombre de todos los menús que tengan algún plato de más de 100 calorías como primero usando “where”
    c. El nombre de todos los menús que tengan algún plato de más de 100 calorías como primero usando “inner join”
10. Muestre:
    a. Los nombres de los menús que tengan de primero algún plato de cuchara
    b. Quite los repetidos
    c. Ordénelos en orden ascendente

11. Muestre:
    a. El nombre de cada menú ordenado de forma ascendente
    b. El nombre de cada menú ordenado de forma descendente
12. Muestre el nombre de cada plato ordenado por número de calorías
13. Muestre los platos de los menús del lunes con menos de 50 calorías
14. Muestre:
    a. Los platos con más calorías de los platos de nuestro restaurante
    b. Los platos con menos calorías de los platos de nuestro restaurante
    c. La media de las calorías de los platos de nuestro restaurante
    d. La suma de las calorías de los platos de nuestro restaurante
15. Muestre:
    a. Para cada menú el número máximo de calorías que proporciona
    b. Para cada menú y cada tipo el número máximo de calorías que sirve
    c. Para cada menú de los lunes y cada tipo el número máximo de calorías que sirve
16. Muestre:
    a. Para cada menú y cada tipo el número máximo de calorías que sirve y la media de calorías que sirve
    b. Ordénelo descendentemente según la media de las calorías (sin usar alias)
    c. Ordénelo descendentemente según la media de las calorías (use un alias)
17. Indique:
    a. La media de las calorías de cada tipo de plato (primeros, segundos, postres)
    b. Ordénelo ascendentemente
18. Cuente cuantos platos de cada menú tienen más de 75 calorías
19. Indique el número de ingredientes de cada plato
    a. Solo de los platos que tienen ingredientes
    b. De todos los platos
20. Realice la suma de las calorías de cada menú.
21. Muestre el nombre de cada menú y:
    a. El precio de cada menú
    b. A la consulta anterior añada la columna IVA (10%)
22. Realizar las siguientes modificaciones en la BBDD
    a. Aumentar el precio de cada menú en 10%:
    b. Añadir un 10% adicional a los menús especiales
    c. Aumentar el precio en 1 euro de todos los menús que cuestan 15€.
23. Modificar el precio de todos los menús de 9 Euros a 12 Euros (grabar en la tabla)
24. Borre el caviar de los ingredientes. Está muy caro. ¿se puede?
25. Borre todos los platos con cuchara ¿Se puede? ¿Ha pasado algo más?
26. Añadir un menú “Jueves extra” con coste 20€ y dejarlo sin platos

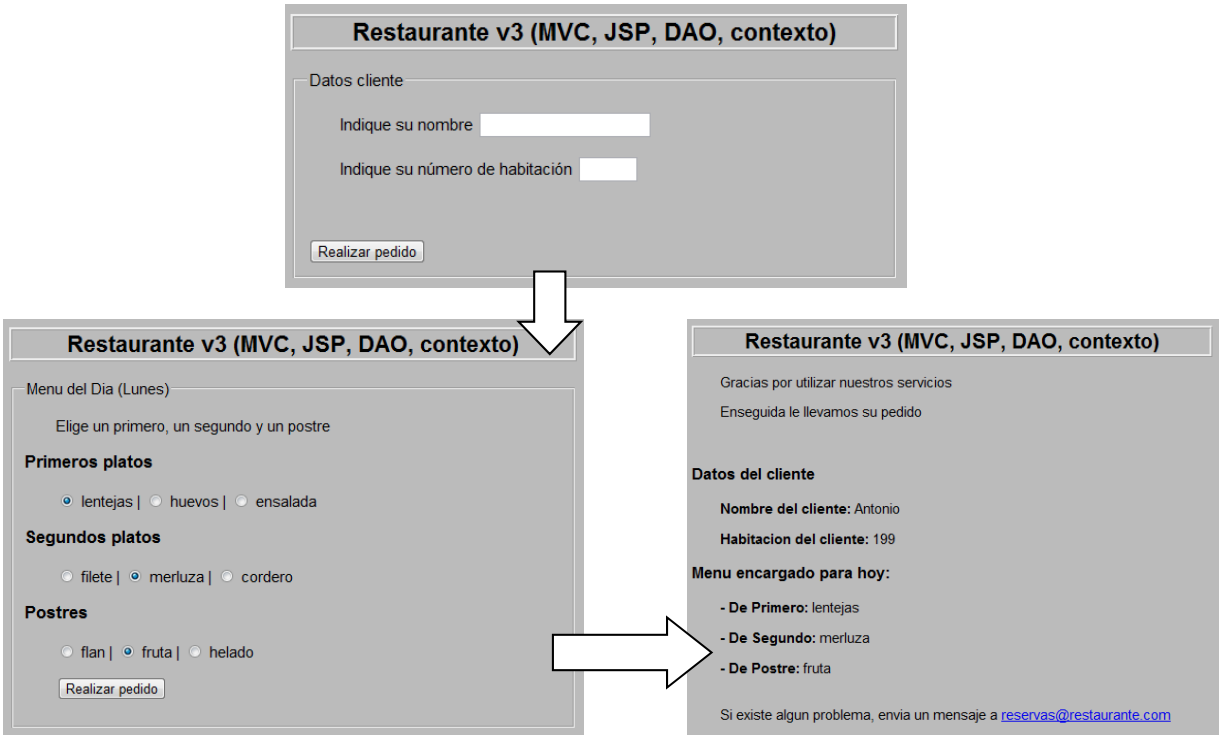
27. Ejecutar:
    a. Añadir al plato “Dorada” el ingrediente “Dorada”
    b. Añadir al plato “Dorada” el ingrediente “pimienta”
    c. Añadir al plato “Lomo” el ingrediente “pimienta”
    d. Modifique la “pimienta” de la dorada y cámbielo por “sal”
28. Muestre todos los ingredientes necesarios para confeccionar el menú del lunes
    a. Con “where”
    b. Con ‘inner join’ anidados
29. Muestre los platos con ingredientes
    a. Con “where”
    b. Con “inner join”
    c. Con “subconsultas”
30. El nombre del plato más calórico
31. El nombre del plato menos calórico
32. Muestre los platos sin ingredientes
33. Muestre los primeros platos de todos los menús que tengan algún plato de cuchara



EL RESTAURANTE

Un conocido hotel quieres diseñar una aplicación para que los clientes de su hotel acceden a una dirección web y soliciten el menú que quieren para comer. De esta forma se pasa la comanda a cocina para que esté todo preparado y se le pueda llevar a su habitación.

Se ha realizado un diseño con tres páginas



Se pide realizar las siguientes versiones:

Versión	Materia	Descripción	
V1	HTML+CSS	Sólo 3 páginas Web enlazadas (páginas "trucadas")	TODO está trucado, gañán. Aquí no se recoge ningún parámetro, mangurrino… que no te enteras, zarpas
V2	MVC básico	MVC (páginas "trucadas" que no recogen parámetros… sólo navegan) o Versión v2a: con un único servlet + web.xml o Versión v2b: con dos servlets + web.xml o Versión v2c: v2a ("servlet único") + anotaciones o Versión v2d: v2b (Multiservlet) + anotaciones o Versión v2e: v2c con varios métodos (Gestor) ✓	
V3	Servlets y JSP	MVC (sin BBDD real) (con anotaciones) o Versión v3a: Sesiones, Model, Parámetros, EL, JSTL, context (mail) ✓ o Versión v3b: v3a + eventos ocurridos	
V4	BBDD y JDBC	MVC (con BBDD real) (con anotaciones) o Versión v4a: v3e (MVC) + BBDD real + excepciones + fichero .properties	
V5	Filtros	o Versión v5a: (para saber cuánto tarda en contestar el usuario) o Versión v5b: v5a(Crono) + anotaciones o Versión v5c: (para saber cuántas veces accede a cada página el usuario)	
V6	Spring	o Adaptación con Spring MVC	

Versión 01 : Control
Versión 02 : Control + Modelo
Versión 03a: MVC + anotaciones
Versión 03b: MVC + web.xml
Versión 04 : (v03a) + Objetos
Versión 05 : (v04) + EL + JSTL
Versión 06 : (v05) + Capas

Cervezas.html (v01)(v02)(v03)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cervezas v1</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <h1>Beer Selection Page</h1>
        <form method="POST" action="SelectBeer.do">
            <p>Selector de cervezas v1</p>
            <p>Color:</p>
            <p>
                <select name="color" size="1">
                    <option value="light"> light </option>
                    <option value="amber"> amber </option>
                    <option value="brown"> brown </option>
                    <option value="dark"> dark </option>
                </select>
                <br/><br/></p>
            <p><input type="submit" value="Enviar" /></p>
        </form>
    </body>
</html>
```

Cambia el v1, v2…

Cambia el v1, v2…

BeerSelect.java (v01)

```
package cervezas.control;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author Antonio Santos
 */
public class BeerSelect extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Selector de cervezas v1</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Selector de cervezas v1</h1>");
        out.println("<p>Consejos para seleccionar una cerveza</p>");
        String c = request.getParameter("color");
        out.println("<p>Ok. Te gusta la cerveza de color " + c + "</p>");
        out.println("</body>");
        out.println("</html>");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

BeerSelect.java (v02)

```
package cervezas.control;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cervezas.modelo.*;
import java.io.*;
import java.util.*;

/**
 *
 * @author Antonio Santos
 */
public class BeerSelect extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Selector de cervezas v2</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Selector de cervezas v2</h1>");
        out.println("<p>Consejos para seleccionar una cerveza</p>");
        String c = request.getParameter("color");
        //out.println("<p>Ok. Te gusta la cerveza de color " + c + "</p>");
        BeerExpert be = new BeerExpert();
        List result = be.getBrands(c);
        Iterator it = result.iterator();
        while (it.hasNext()) {
            out.print("<p>prueba: " + it.next() + "</p>");
        }
        out.println("</body>");
        out.println("</html>");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

BeerExpert.java (v02)(v03)

```
package cervezas.modelo;

import java.util.*;

public class BeerExpert {

    public List getBrands(String color) {
        List brands = new ArrayList();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        } else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
        return (brands);
    }
}
```

BeerSelect.java (v03)

```
package cervezas.control;

import javax.servlet.*;
import javax.servlet.http.*;
import cervezas.modelo.*;
import java.io.*;
import java.util.*;

/**
 * @author Antonio Santos
 *
 */
public class BeerSelect extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String c = request.getParameter("color");
        BeerExpert be = new BeerExpert();
        List result = be.getBrands(c);
        request.setAttribute("styles", result);
        RequestDispatcher view = request.getRequestDispatcher("result.jsp");
        view.forward(request, response);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

Result.jsp (v03)

```
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html>
    <head>
        <title>Cervezas v3</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <h1>Cervezas v3. JSP con recomendaciones sobre cervezas</h1>
        <p>
            <%
                List styles = (List) request.getAttribute("styles");
                Iterator it = styles.iterator();
                while (it.hasNext()) {
                    out.print("<br>try: " + it.next());
                }
            %>
        </p>
    </body>
</html>
```

NOTAS

- Falta toda la configuración… eso es parte del ejercicio
- Hay que saber funcionar con anotaciones y con web.xml, pero las anotaciones son más modernas. Por defecto hacerlos con anotaciones
- Las versiones v4, v5 y v6 son evoluciones del ejercicio
- Incluye siempre un web.xml aunque esté vacío (no hará falta con Spring Boot)