1	Un profesor necesita sumar dos números enteros. Se pide crear un programa con el siguiente menú: 1.- Leer los números 2.- Calcular la suma 3.- Mostrar el resultado q.- Salir empleando la arquitectura MVC ya conocida.
2	Una compañía tiene cinco almacenes, cada uno de los cuales vende dos productos. Se dispone de una tabla con las ventas por unidades de los dos productos en todos los almacenes. Se dispone también de los precios que tiene cada unidad de los dos productos en todos los almacenes. ¿Cuál es el beneficio total de la empresa? Pedir información por teclado, y mostrar como resultado los datos en forma tabular y el total de beneficios. Construir `makefile` y `manifest`. Resolver el problema usando listas ([]): y la arquitectura MVC ya conocida. El menú puede ser como el siguiente.
3	Pleno al quince. Escribir un programa que pida a través del teclado los nombres de los equipos y el número de goles de cada equipo. El programa debe mostrar la quiniela, pleno al 15, con los equipos en columnas de igual longitud ajustadas a la izquierda y los resultados alineados a la derecha (1, X, 2) Resolver el problema usando listas ([]) y clases (`Equipo` con nombre y goles, `Partido` con 2 `Equipos` y `resultado()`). Los datos fijos (los nombres de los equipos de cada jornada) residen en la carpeta datos del escritorio, en un archivo llamado `equipos.txt` que se lee automáticamente en el arranque. El programa muestra el listado de equipos y pide el número de goles de cada uno, para dp mostrar una quiniela con resultados 1X2. Crear un menú con las opciones necesarias. Resolver el problema usando listas ([]): y la arquitectura MVC ya conocida.
4	Una compañía tiene un archivo de `Facturas`, con formato de texto delimitado por tabuladores (CSV). Cada línea del archivo contiene el `nombre`, `apellidos`, `DNI` y `teléfono` del cliente, así como el `importe` de la factura, el tipo de `IVA` y el `descuento` aplicado (que puede ser 0 y se expresa como un número menor que 1 y con 2 decimales).Se pide construir un programa al que se le pueda proporcionar un valor mínimo de factura, que se calculará en la forma `precio_base*(1-descuento)*IVA` para cada factura, y que proporcione el estado de clientes que tengan facturas con valor total mayor que el dado. Resolver el problema usando colecciones y clases, incluso bibliotecas (como `biblioteca.jar`). Resolver el problema la arquitectura MVC ya conocida.

5	Crear un programa basado en la clase `Persona` (nombre, teléfono y peso), y en otra clase `Principal` que lee de disco una colección de personas y muestra sus datos de forma tabular. Los datos residen en un archivo en el escritorio llamado datos.txt. El programa debe mostrar una opción de menú que permita leer (importar) los datos, otra que permita mostrarlos en forma tabular y una tercera que permita al usuario salir de programa tras preguntarle si desea hacerlo. El archivo datos.txt tiene formato delimitado por tabuladores. Resolver el problema usando colecciones y la arquitectura MVC ya conocida.

6	Crear el programa `ListaDeClase`. Está basado en la clase `DatosDeAlumno`,
	que tiene un atributo de tipo `Direccion` y otro de tipo `DatosPersonales`.
	La clase `DatosDeDireccion` tiene como atributos
	
	una calle (`String`),
	un numero (`int`)
	una letra (`String`).
	un DNI (`String`)
	
	La clase `DatosPersonales` tiene como atributos:
	
	un atributo llamado nombre (`String`)
	un atributo llamado apellidos (`String`)
	un atributo llamado edad (`int`)
	un atributo llamado `NIF`, alfanumérico y no repetido.
	
	Tanto la clase `DatosPersonales` como `DatosDeDirección` poseen
	un método de factoría que crea ejemplares con valores
	aleatorios razonables para los atributos.
	
	El programa debe crear una colección de 20 alumnos
	y ofrecerá un menú con las opciones siguientes:
	
	`1.- Crear lista de alumnos`
	`2.- Mostrar tabla de alumnos`
	`3.- Exportar datos personales`
	`4.- Exportar direcciones`
	`q.- Salir`
	
	La tabla de alumnos mostrada en la opción 2 debe contener tanto los datos personales como los datos de dirección.
	
	Construir `makefile` y `manifest`

7	Un profesor necesita informatizar la asistencia del alumnado a su clase, y para ello diseña el programa `Asistencia`, que está basado en la clase `Alumno.` Esta clase tiene como campos el `nombre`, `primerApellido`, `segundoApellido`, `DNI` y `asistencia.El` atributo `asistencia` es una lista de `boolean`, `boolean`[] con quince elementos. Todos todos atributos son privados. Se pide construir la clase `Alumno`, dotándola de un constructor que genera valores aleatorios razonables para todos los campos. En particular, el campo de asistencia recibe valores `true` o `false` aleatoriamente. El programa genera 30 alumnos y muestra sus atributos (los valores de los campos, esto es) en pantalla,en forma de tabla. Es posible exportar a disco toda la información, incluidas las asistencias, con formato delimitado por tabuladores. Además, el programa calcula el promedio de asistencia a clase para cada alumno, y muestra aquellos alumnos que no han superado el 50% (tienen solo siete asistencias o menos). El programa mostrará un menú como este: `1.- Generar Lista 2.- Mostrar lista 3.- Mostrar alumnos con poca asistencia 5.- Exportar q.- Salir` que permitirá al usuario generar la lista, mostrar la lista general, mostrar la lista de alumnos con poca asistencia, o salir del programa. Resolver el problema usando `biblioteca.jar` y la arquitectura MVC habitual.
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	