EJERCICIO 1

Queremos representar con programación orientada a objetos, un aula con estudiantes y un profesor.  
Tanto de los estudiantes como de los profesores necesitamos saber su nombre, edad y sexo. De los estudiantes, queremos saber también su calificación actual (entre 0 y 10) y del profesor que materia da.  
Las materias disponibles son matemáticas, filosofía y física.  
Los estudiantes tendrán un 50% de hacer novillos, por lo que si hacen novillos no van a clase pero aunque no vayan quedara registrado en el aula (como que cada uno tiene su sitio).  
El profesor tiene un 20% de no encontrarse disponible (reuniones, baja, etc.)  
Las dos operaciones anteriores deben llamarse igual en Estudiante y Profesor (polimorfismo).  
El aula debe tener un identificador numérico, el número máximo de estudiantes y para que esta destinada ( Piensa que más atributos necesita.  
Un aula para que se pueda dar clase necesita que el profesor esté disponible, que el profesor de la materia correspondiente en el aula correspondiente (un profesor de filosofía no puede dar en un aula de matemáticas) y que haya más del 50% de alumnos.  
El objetivo es crear un aula de alumnos y un profesor y determinar si puede darse clase, teniendo en cuenta las condiciones antes dichas.  
Si se puede dar clase mostrar cuantos alumnos y alumnas (por separado) están aprobados de momento (imaginad que les están entregando las notas).  
NOTA: Los datos pueden ser aleatorios (nombres, edad, calificaciones, etc.) siempre y cuando tengan sentido (edad no puede ser 80 en un estudiante o calificación ser 12).

EJERCICIO 2

Nos piden hacer un programa orientado a objetos sobre un cine (solo de una sala) tiene un conjunto de asientos (8 filas por 9 columnas, por ejemplo).  
Del cine nos interesa conocer la película que se está reproduciendo y el precio de la entrada en el cine.

De las películas nos interesa saber el título, duración, edad mínima y director.

Del espectador, nos interesa saber su nombre, edad y el dinero que tiene.

Los asientos son etiquetados por una letra (columna) y un número (fila), la fila 1 empieza al final de la matriz como se muestra en la tabla. También deberemos saber si está ocupado o no el asiento.

8 A 8 B 8 C 8 D 8 E 8 F 8 G 8 H 8 I  
7 A 7 B 7 C 7 D 7 E 7 F 7 G 7 H 7 I  
6 A 6 B 6 C 6 D 6 E 6 F 6 G 6 H 6 I  
5 A 5 B 5 C 5 D 5 E 5 F 5 G 5 H 5 I  
4 A 4 B 4 C 4 D 4 E 4 F 4 G 4 H 4 I  
3 A 3 B 3 C 3 D 3 E 3 F 3 G 3 H 3 I  
2 A 2 B 2 C 2 D 2 E 2 F 2 G 2 H 2 I  
1 A 1 B 1 C 1 D 1 E 1 F 1 G 1 H 1 I

Realizaremos una pequeña simulación, en el que generaremos muchos espectadores y los sentaremos aleatoriamente (no podemos donde ya este ocupado).

En esta versión sentaremos a los espectadores de uno en uno.

Solo se podrá sentar si tienen el suficiente dinero, hay espacio libre y tiene edad para ver la película, en caso de que el asiento este ocupado le buscamos uno libre.

Los datos del espectador y la película pueden ser totalmente aleatorios.

EJERCICIO 3

Realizar un programa en Java que permita calcular el pedido que realiza un cliente en un restaurante.

Los pedidos de un restaurante están conformados por las siguientes partes:

1. Un primer plato.
2. Un segundo plato.
3. Una bebida.
4. Un postre.

Cada uno de dichas partes tiene un nombre y un valor. Se requiere definir métodos sobrecargados para calcular el valor del pedido dependiendo si el cliente solicita:

* Un primer plato y una bebida.
* Un primer plato, un segundo plato y una bebida.
* Un primer plato, un segundo plato, una bebida y un postre.

Implementar un método main que utilice los tres métodos sobrecargados en tres diferentes pedidos.

EJERCICIO 4

Excepciones Cuenta corriente del banco

1. Crearemos la clase InsufficientFundsException (no tienes suficiente dinero en la cuenta), la clase CheckingAccount (cuenta corriente) y una clase tester, que será el main. En la clase CheckingAccount tendremos:

* Un método toString() que mostrará el número de cuenta y saldo.
* Un constructor al que le pasaremos el número de cuenta y nos pondrá el saldo a cero.
* Un método para ingresar dinero y otro para extraer dinero. Lanzaremos la excepción InsufficienteFundsException cuando sea debido. Y, la clase tester la capturará e imprimirá “Sorry, but you don’t have that much money”.

1. Haced una nueva versión donde la clase InsufficientFundsExpection tenga un atributo que sea cuantos Euros le falta para poder retirar el dinero que ha pedido. Para ello, al constructor se le tendrá que pasar dicho valor y tendrá que tener un setter que devuelva el valor de dicho atributo. Esto es para que interioricemos que la Exceptions son clases y, como tales, puedes tener los atributos y métodos que consideremos necesarios.

EJERCICIO 5

Implementa la clase Dado. Por defecto el dado tendrá 6 caras. Tendremos 3 construcciones (Uno al que no se le pasa nada e inicializar el dada al azar, otro al que sólo se le qué número tiene el dado en la cara superior y otro con el número del dado en la cara superior y el número de caras del dado). Implementa los getters, el método tirarDado() que tirará el dado al azar y el toString()de acuerdo con el ejemplo de ejecución. Implementa un tester que tenga un vector de 4 datos y los lance una serie de veces.

