RELACIÓN DE PROBLEMAS 7
PROGRAMACIÓN ORIENTADA A OBJETOS II

Realizar los siguientes ejercicios controlando los errores con Excepciones.

1. Define una clase **Linea** con dos atributos: **puntoA** y **puntoB**. Son los dos puntos por los que pasa la línea en un espacio de dos dimensiones. La clase dispondrá de los siguientes métodos:

- **Linea():** Constructor predeterminado que crea una línea con sus dos puntos como (0,0) y (0,0).
- **Linea(Punto, Punto):** Constructor que recibe como parámetros dos objetos de la clase Punto, que son utilizados para inicializar los atributos.
- **gets y sets.**
- Debe redefinir el método **equals** para comparar líneas.
- **moverDerecha(double):** Desplaza la línea a la derecha la distancia que se indique.
- **moverIzquierda(double):** Desplaza la línea a la izquierda la distancia que se indique.
- **moverArriba(double):** Desplaza la línea hacia arriba la distancia que se indique.
- **moverAbajo(double):** Desplaza la línea hacia abajo la distancia que se indique.
- Método que nos permita mostrar la información de la línea de la siguiente forma: [puntoA,puntoB]. Por ejemplo: [(0.0,0.0),(1.0,1.0)].

Realiza un programa que inicialmente cree una línea solicitando los datos por teclado. Después debe mostrar un menú con las siguientes opciones:

1. Mover línea: Solicitará el movimiento (A-arriba, B-ABajo, I-Izquierda, D-Derecha) y realice el movimiento
2. Mostrar línea
3. Salir

2. Crear una clase que represente **Producto** con las siguientes características:

- Tienen un código que los identifica de manera única y que se asigna automáticamente en el momento de la creación.
- Guardan la descripción y el precio sin IVA.
- Todos los productos comparten el mismo IVA (supongamos el 20%), que puede variar en función de las decisiones del gobierno.

La clase **Producto** debe proporcionar los métodos adecuados:

- Constructor.
- Métodos para consulta y modificación de los atributos.
- Método para calcular el precio de venta del producto que se obtiene sumándole al precio el IVA correspondiente.

Realiza un programa principal que pruebe la clase anterior.

3. Crear una clase **Jarra** que utilizaremos para simular algunas de las acciones que podemos realizar con una jarra. Cada jarra tiene una determinada capacidad (en litros). Además una jarra dispondrá de una cantidad de agua que podrá ir variando a medida que realicemos operaciones con ellas. Las jarras se crearán siempre vacías.

Las acciones que podremos realizar sobre una jarra son:

- Llenar la jarra por completo desde un grifo.
- Vaciarla por completo.
- Volcar el contenido de una jarra en otra.

Por ejemplo: Disponemos de dos jarras A y B de capacidades 7 y 4 litros respectivamente. Podemos llenar la jarra A (no podemos echar menos del total de la jarra porque no sabríamos a ciencia cierta cuánta agua tendría). Luego volcar A sobre B (no cabe todo por lo que en A quedan 3 litros y B está llena). Ahora vaciar B. Después volver a volcar A sobre B. En esta situación, A está vacía y B tiene 3 litros.

Además se debe guardar en la clase Jarra el **total de agua** que se ha consumido llenando objetos Jarra.

Realizar un programa que cree dos jarras A y B (solicitando las capacidades por teclado) inicialmente vacías. Después se realizará un menú que permita:

1. **Llenar jarra**: Se solicitará con la pregunta "¿Qué jarra desea llenar (A/B)?" y se llenará la jarra correspondiente.
2. **Vaciar jarra**: Se realizará la pregunta "¿Qué jarra desea vaciar (A/ B)?" y se vaciará la jarra correspondiente.
3. **Volcar jarra A en B** .
4. **Volcar jarra B en A**.
5. **Ver estado de las jarras**: Se mostrará la capacidad y el agua que contiene, tanto para la jarra A como para la B.
6. **Salir**: Cuando salga debe mostrarse un mensaje que indique "El total de agua que se ha gastado llenando jarras es XXX litros".

5. Se desea desarrollar una aplicación que permita gestionar los datos de la Liga de fútbol profesional.

Deben tenerse en cuenta los siguientes aspectos:

- Un partido se juega en una jornada (entero entre 1 y 38) entre un equipo local y un equipo visitante. También interesa almacenar el número de goles del equipo local y del visitante, así como el resultado de la quiniela (un carácter con '1', 'X', '2').
- Ni el dato de la jornada, ni los equipos local y visitante de un partido son modificables.
- Cuando se crea un partido todavía no se ha jugado por lo que no se dispone de información de goles ni resultados. Debe tenerse en cuenta que los equipos que se

enfrentan en el partido sean diferentes. Se considera que dos equipos son iguales si tienen el mismo nombre.

- Debe realizarse el método **`ponerResultado`** en la clase Partido. El método recibirá una cadena de caracteres con el resultado en el formato "1-2" y actualizará los atributos goles local y visitante y el resultado de la quiniela, así como el número de partidos ganados de los equipos. Deben tratarse todos los posibles errores con excepciones.
- Debe programarse el método **`toString`** de la clase Partido.
  - Si el partido aún no se ha jugado aparecerá la siguiente información:

    "Partido entre equipo local … .. y equipo visitante ….. todavía no se ha jugado"

  - Si el partido ya está acabado aparecerá la siguiente información

    "Partido entre equipo local ….. y el equipo visitante ….. jugado en el estadio …… de la ciudad …… ha finalizado con .. goles de equipo local y .. goles de equipo visitante. Resultado quiniela= ..."
- No es necesario crear método **`equals`** de la clase **`Partido`**.

Realizar un programa principal para que funcione de la siguiente forma:

- Debe crear un partido entre dos equipos e intentar poner un resultado "2-1" (el resto de datos puedes inventarlo).
- Independiente de si el apartado anterior provoca o no un error, deben crearse otros dos partidos entre cualquiera de los equipos con resultados "0-0" y "1-2" asegurándose de que si se produce un error se vuelvan a intentar crear los dos partidos.
- Una vez se hayan creado los dos partidos se mostrarán su información por pantalla, así como la información de los equipos participantes.