FUNDAMENTOS DE INFORMÁTICA
Ingeniería Electrónica y Automática (18/19)

Práctica 6
Diseño de clases

Alumnos:		*Grupo:*

Esta práctica tiene como objetivo continuar familiarizándonos con los conceptos de POO que ya se aplicaron en la práctica anterior, esto es, el diseño de clases sencillas, sus componentes fundamentales (campos o atributos, constructores y métodos), el desarrollo de los mismos y la creación de nuevas clases utilizando otras para resolver problemas más complejos.

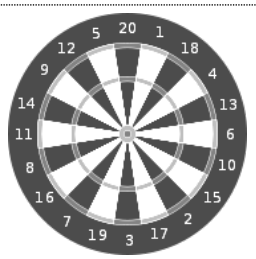
Ejercicio 1.- El objetivo es simular el desarrollo de una partida de dardos entre dos jugadores completando las partes inconclusas del proyecto `Dardos`.

En una partida de dardos los jugadores parten de una puntuación objetivo predeterminada, y los puntos que va obteniendo cada jugador en cada turno de tirada se van restando a su puntuación. Gana el primer jugador que llega exactamente a 0 puntos.

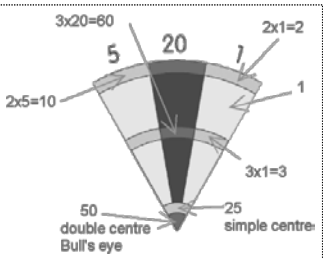
Se propone la siguiente especificación de clases para el proyecto que resolverá este problema.

La primera de las clases, ya construida, contiene los atributos, constructor y métodos necesarios para representar el **lanzamiento de un dardo**, con las características descritas a continuación.

– Los atributos permiten conocer, por un lado el valor de la zona donde se ha clavado el dardo (es decir, 0 si ha ido a parar fuera de la diana, 25 si lo ha hecho al centro simple, 50 si lo ha hecho al centro doble, y en otro caso un entero entre 1 y 20), y por otro el multiplicador que se aplica a dicho valor (un valor entero entre 1 y 3).



Lo que son el número y el multiplicador



– El constructor de la clase asigna valores aleatorios a los atributos haciendo uso de las funciones `valorAleatorio` y `multiplicadorAleatorio` incluidas en la clase `Utiles` que también se proporciona construida. Observa que cada vez que en el proyecto se necesite simular el lanzamiento de un dardo se hará mediante una llamada a este constructor.

– Los métodos realizan las siguientes operaciones:
  - Evidentemente, calcular los puntos del lanzamiento.
  - Mostrar al usuario los datos del lanzamiento (valor, multiplicador y puntuación)

La segunda de las clases, y primera que hay que construir, contendrá el atributo, el constructor y los métodos necesarios para representar un **jugador**, con las características que se describen a continuación.

– Su único atributo debe permitir conocer la puntuación actual del jugador.

– El constructor se encargará de inicializar adecuadamente el atributo descrito.

– Los métodos necesarios para luego poder implementar la partida son los que realizan las siguientes operaciones:
  - Averiguar si el jugador tiene la puntuación que le permite ganar la partida.
  - Calcular los puntos obtenidos por el jugador en un turno de tirada.
    Ten en cuenta que en cada turno de tirada un jugador lanza 3 dardos, salvo si con el 1er o el 2^{o} lanzamientos obtiene una cantidad de puntos mayor o igual que la puntuación del jugador. Los puntos obtenidos en el turno de tirada son la suma de los puntos de los lanzamientos, salvo que esta cantidad supere la puntuación que tiene el jugador, en cuyo caso la tirada se anula en su totalidad (se obtienen 0 puntos). Observa que este método simplemente ha de calcular los puntos obtenidos, es decir, **no debe modificar la puntuación del jugador** (de eso se encargará el método siguiente). Tras cada lanzamiento se mostrarán al usuario los datos del lanzamiento.
  - Actualizar la puntuación del jugador tras haber realizado un turno de tirada.
  - Mostrar al usuario la puntuación del jugador.

La última clase contendrá los atributos, el constructor y los métodos necesarios para poder desarrollar una **partida entre dos jugadores**.

– Los atributos deben permitir almacenar los dos jugadores y la puntuación objetivo a la que se juega la partida.

– El constructor debe inicializar la puntuación objetivo y crear los dos jugadores. Se considerarán dos modalidades de juego, 301 y 501, que son las respectivas puntuaciones objetivo del juego.

– Los métodos a implementar deben permitir:
  - Pedir al usuario por teclado la modalidad de juego (301 o 501).
  - Mostrar el marcador (la puntuación de los dos jugadores).

- Desarrollar la simulación de la partida según el siguiente esquema:
  Mientras no acaba la partida (es decir, mientras no ha ganado ninguno de los jugadores), se repite que el jugador 1 haga un turno de tirada y, si no ha ganado, después lo haga el jugador 2. Tras cada turno de tirada se mostrará el marcador y a quién le corresponde el siguiente turno, y se hará una pausa (véase el contenido de la clase `Utiles`) para que el usuario pueda visualizarlo. Al finalizar se mostrará un mensaje indicando quién ha ganado la partida. Para evitar que la partida se alargue en exceso se considerará un máximo de 20 turnos de tirada por jugador, por lo que una vez agotados la partida finalizará sin ganador.

Ejercicio 2.- El objetivo es implementar el juego "Pares o nones" entre el usuario y el computador. Es decir, habrá dos jugadores, de los que uno será el usuario y el otro el computador. La ejecución del programa desarrollará una partida a 5 victorias. Es decir, ganará el primer jugador que obtenga ese número de victorias.

Los atributos de un jugador permitirán almacenar el nombre, la puntuación (el número de jugadas que ha ganado) y la jugada en cada turno (que son dos valores, el número de dedos que saca y si ha elegido pares o nones). El nombre del usuario, lógicamente, se le preguntará al inicio del juego, mientras que el del ordenador será simplemente "Ordenador".

El constructor de un jugador establecerá los valores adecuados que permitan iniciar la partida.

Los métodos de un jugador permitirán:

- Mostrar su jugada elegida
- Mostrar la puntuación
- Saber si su jugada gana a la de otro jugador (sumando los números sacados, comprobando la paridad de ese número y si coincide con la elección efectuada de pares o nones)
- Incrementar su puntuación
- Elegir su jugada. Observa que la jugada del usuario y la del ordenador han de obtenerse de forma diferente. Al usuario hay que preguntársela mientras que para el ordenador hay que calcularla aleatoriamente (fíjate en la implementación de `multiplicadorAleatorio` y `valorAleatorio` incluidas en la clase `Utiles` proporcionada para el ejercicio 1)

Quizá el diseño de clases más adecuado resulte de aplicar herencia, definiendo una clase para representar a un jugador genérico (es decir, que contenga todos los elementos comunes tanto para el usuario como el ordenador), otra que represente al usuario que solo contenga los elementos particulares de este jugador específico, y una tercera que represente del mismo modo al ordenador.

Adicionalmente, hay que definir una última clase que permita desarrollar una partida entre un usuario y un ordenador. Los atributos permitirán representar a los dos jugadores. El constructor tendrá que crear los jugadores para poder iniciar la partida. Los métodos permitirán:

- Saber si la partida ha terminado (es decir, alguno de los jugadores ha alcanzado el número de victorias objetivo)
- Actualizar la puntuación de los jugadores
- Mostrar el marcador de la partida
- Desarrollar una partida conforme al siguiente esquema: Mientras no acaba la partida, se repite que el usuario hace su jugada, después lo hace el computador, se muestran las jugadas elegidas por ambos jugadores, se actualizan sus puntuaciones y se muestra el marcador actual. Al finalizar se mostrará un mensaje indicando quién ha sido el ganador.

Ejercicio 3.- En un sistema centralizado de gestión de citas sanitarias, se quiere generar las clases Java suficientes para representar la información necesaria que permita gestionar las reservas o citas para que los médicos atiendan a los pacientes. Una Cita Médica estará compuesta por la información del centro sanitario donde se va a realizar, por el médico colegiado que pasará consulta, por el paciente que será atendido y por el momento de la cita. Interesa conocer el tipo de centro sanitario (consultorio, centro de salud, centro especializado u hospital) junto con el nombre del mismo y la dirección donde se encuentra. También interesará conocer el servicio o departamento que se encarga de la atención. Del médico que atiende interesa conocer su nombre y su número de colegiado, así como los instantes de inicio y de final de su consulta. Del paciente interesa su nombre y su número NSS. Por último, el momento de la cita queda determinado por una fecha y un instante de tiempo de la jornada. Todas las citas se supondrán de una cierta duración fija preestablecida. Por simplicidad, se supondrán también que los instantes de inicio y de final de la consulta de un médico son válidos para todas las fechas (es decir, no se considerarán festivos).

Diseña las clases necesarias y, una vez implementadas, genera un programa con el código necesario para realizar al menos las siguientes tareas:

- Leer una cita de teclado. Úsala para poder generar objetos de clase cita y realizar pruebas de las operaciones subsiguientes
- Comprobar si una cita es anterior a otra temporalmente
- Comprobar si una cita colisiona con otra en algún aspecto (es decir, si no pueden darse ambas simultáneamente)
- Comprobar si el momento de una cita es correcto, lo que supone que su instante no debe ser anterior al inicio de la consulta del médico ni tampoco posterior al final, y además el número de minutos de diferencia con el instante inicial debe ser un múltiplo exacto de la duración de las citas (por ejemplo, si las citas duran 7 minutos y el instante de inicio son las 9:00, un instante de cita válido sería 9:21 y no lo sería 9:30)
- Otras operaciones que estimes necesarias para gestionar las citas