Trabajo Especial para la cátedra de Programación 2

Autor: Franco Aller.
Nro de libreta:248090.

El diseño que creí adecuado para resolver e implementar el juego requerido, sobre el paradigma orientado a objetos , se explicará a continuación.
Primeramente comencé realizando el diagrama de clases, el cual está adjunto en la entrega.
En el diagrama puede verse que identifique 6 clases, de las cuales 2 son subclases.
Estas se listan a continuación:
1. Juego
2. Jugador
3. Mazo
4. Carta
5. Atributo (clase abstracta)
6. AtributoMayor (subclase de Atributo)
7. AtributoMenor (subclase de Atributo)

La decisión tomada ante la creación de cada una de estas clases fue a causa del rol que debían de prestar cada una de ellas según el enunciado del trabajo. Junto con esta clasificación de roles se logro definir las tareas(metodos) especificas que deberían de cumplir cada una. Con cada clase se quiso llevar a cabo una división eliminando las ambigüedades que podrían darse entre las mismas, y intentando clarificar y delegar correctamente su comportamiento.

Primero comenzare hablando de la clase Juego y posteriormente de las que le siguen, según el orden dado en la lista anterior.
La clase Juego tiene todas las funcionalidades necesarias para instanciar y manipular los objetos de las clases anunciadas. Mas precisamente realiza lo siguiente:

- Instancia los objetos requeridos para llevar a cabo el juego
  - Mazo, indicando su nombre, la cantidad de cartas.
  - Jugadores inicializa los jugadores necesarios para el juego, en este caso se crean tan solo 2 jugadores, y se les inicializa con nombre que se les asigne por consola.
  - Cartas, indicando sus nombres y cantidad de atributos.
  - Atributos, indicando sus nombres, valores , unidades y contienda.
    - AtributoMayor
    - atributoMenor
- En cuanto a su comportamiento:
  - Se encarga de aplicar las reglas y metodología del juego.
  - Define de que forma se va a interactuar con el mazo las cartas del mazo, como estas competirán y de que forma y en cuanto a que atributos o propiedades de las cartas se va a llevar una evaluación para definir cual de los dos jugadores gana, pierde o si se produce un empate entre ambos.

Se decidió el comportamiento mencionado anteriormente, ya que el juego deberá de modelar sus reglas sin que se vean afectadas las demás clases en cuanto a su comportamiento.
De esta forma es posible reutilizar estas últimas clases (mazo, jugador, carta y/o atributos) en otros tipo de juegos sin realizar cambios en las mismas.

Métodos de la clase juego:
armarMazoGenerico():
Crea un mazo genérico, en el que se especifica por consola, el tipo de mazo(nombre del mazo), cantidad de cartas, la cantidad de cartas tiene una restricción que impone que debe de ingresarse un número par de cartas para el juego. Se tomó esta decisión para que ninguno de los dos jugadores tuviera ventaja por sobre el otro.
También se especifica cantidad de atributos de cada carta, además de que por una única vez se podrán elegir el nombre de los atributos que se creen, su contienda, y su unidad. El valor del atributo se dará de forma random, pero el usuario podrá elegir el rango. En cuanto a la contienda ingresada se instanciará o un atributoMayor o un atributoMenor, cada cual resuelve una problemática distinta.
Ya una vez ingresado los parámetros anteriores, el método comienza con la creación de cada objeto necesario para el juego, usando estos parámetros para la instanciación de los mismos. Algunos parámetros se asignan de una forma genérica como por ejemplo el nombre de cada carta, los cuales tendrán la forma Personaje-x siendo x el número de cartas que se han creado. Al terminarse de ejecutarse este método, se habrá creado un mazo genérico.

armarMazoDesdeArchivo():
Decidí dar la posibilidad de ingresar un mazo desde un archivo, ya que da muchas ventajas en cuanto a tener que crear un mazo indicando sus propiedades cada vez que se ejecuta el juego. Las ventajas mas notables de la decisión son: creación del mazo una sola vez, luego se rehusara cuantas veces se necesite. Posibilidad de cargar diferentes mazos, ya sean propios o ajenos, etc.
Lo que hace este método es crear los objetos, mazo, carta y atributos mediante la lectura secuencial de un archivo. Para esto se utilizo InputStream y BufferedReader y se lee renglon a renglon mediante readLine(). Al comenzar leyendo el primer renglón del archivo, se encuentra el nombre del mazo, luego en el siguiente renglón del archivo se encontrara la primer carta y así sucesivamente se ciclara hasta encontrar la frase finArchivo que especifica que no se ingresaran mas cartas al juego. Entonces, cada renglón del archivo donde se especifica una carta, comienza con el nombre de la carta, y le suceden el nombre del atributo y las variables del atributo(nombre, valor, contienda, unidad de medida) en un determinado orden.
Estos valores pertenecientes al atributo son enviados a un nuemo metodo que se encargara de la construccion de las cartas mediante estos valores, este nuevo metodo es armaCartaDeArchivo(). Entre los dos llevan a cabo la tarea de la instanciación de los objetos que necesita el juego y los cuales fueron cargados desde el archivo.
Este nuevo método recibirá el arreglo y desde el mismo tomara el valor que esta en la posición 0 para representar el nombre de la carta, entonces instancia una nueva carta con este nombre y en la liña siguiente comienza un ciclo for que ciclara de 4 en 4 a desde la posición 1. De este modo, en cada ciclo se toma los valores para instanciar un atributo, ya que se esperan 4 parámetros por cada atributo, los cuales son nombre, valor, contienda y unidad. Ya en cada ciclo se evalúa la contienda del atributo y según esta, se instanciara o un atributoMayor o un atributoMenor y se agregara este a la carta creada mediante el metodo carta.agregarAtributo(Atributo nuevoAtributo).
Una vez terminado el ciclo for e inicializados todos los atributos, entonces el metodo armaCartaDeArchivo() retornara la carta.
El metodo llamador agrega la carta al mazo y continua realizando el procedimiento anterior hasta que se encuentre la frase finArchivo en uno de los renglones devueltos por el readLine. Solo entonces el mazo y conjunamente con las cartas y atributos estaran instanciados.

El método jugar:
El método jugar define las reglas que tendrá el juego (como se jugará) , como se aplicarán y este retornará un ganador o un empate. La mayor parte del desarrollo se encuentra aquí.
Se comienza verificando que el mazo sea correcto, de ser así empieza la competencia. A continuación se dividen las cartas entre los jugadores, método que lleva a cabo el mazo ya que es quien se encarga de manejar las cartas(*baraja*.repartirMazo(p1, p2)).
En este momento realmente comienzan a jugar, se ingresa en la condición(while) que exigirá que mientras uno de los dos tenga cartas el juego seguirá en marcha. Este while engloba toda la lógica del juego. Las siguientes lineas son las que definirán el ganador del juego juego de al menos un ciclo del mismo estas retornan el ganador.
Hora comienza un ciclo do while en el que se especifica que mientras allá un empate y ambos jugadores tengan cartas se ciclara nuevamente. Este se da para el caso en que hay un empate y de esta forma se puede competir hasta que uno de los dos jugadores gana y entonces se dejara de ciclar o hasta que ambos jugadores se queden sin carta luego de uno o varias rondas empatadas en cuyo caso se dejara de ciclar. Esta condición tiene dentro el segundo bloque de logica que describirá a continuación.
La primera evaluación dentro de este bloque es el switch que comprueba quien tiene el próximo turno y obtiene el atributo a competir del jugador. Este también define la opción default dando el turno al primer jugador y a partir de este se obtiene un atributo por el cual competir (se utiliza en la primera jugada y en el caso de empate). El atributo es obtenido mediante un método randomAtributo
que devuelve un atributo de la primera carta del jugador que tiene el turno.
Ya entonces comienza la competencia mediante el método competencia(jugadorConTurno, jugadorPerdedor, atributoAcompetir) se compite entre este atributo elegido desde el jugador que tiene el turno contra el mismo atributo que tiene el jugador 2 en la primera de sus cartas. Este método a su vez llama a otro método competencia(), el cual se encuentra en la clase atributo y devuelve cual de los dos atributos gano, otro caso posible es devolver un empate. A partir de este ultimo método se sabe que atributo gano, y con el método padre se sabe que carta gano, y ya situados nuevamente al método juego, el método competencia devuelve 1(gana jugador 1), 2 (gana jugador 1) y 0(empate).
A continuación se ingresa al segundo switch en dentro del método jugar(). Este evalúa quien es el ganador o si ocurrió un empate a partir de la variable restacarta(variable previamente inicializada al comienzo del metodo jugar()) que contiene el valor del método competencia().
En el caso *GANA_PRIMERA(constante equivalente a 1) se llama a un método del mazo denominado* intercambiaCartas(jugador 1, jugador 2) este se encarga de transferir la carta del perdedor al ganador.
Luego corrobora si el arrayList *catasEnJuego* no esta vacio, en cuyo caso se transfiere todas las cartas de este arreglo a el suyo y se setea la variable empate a false. Este arrayList guarda las cartas que se han dado durante un empate.A continuación se setea la variable turno.
En el caso *GANA_SEGUNDA DEFAULT(constante equivalente a 2) se realiza exacamente lo mismo que el anterior caso, solo que se asigna al jugador 2.*
El caso DEFAULT setea el atributo a competir por el que ya estaba en juego, en vez de elegir uno random. Luego agrega las cartas de ambos jugadores al arrayList catasEnJuego que se encuentra en el mazo y borra estas cartas de los arreglos de cartas de los jugadores. Luego setea la variable empate a true.
Aquí termina el segundo bloque de lógica que es contenido por el do-while que se menciono liñas arriba.
En el caso de que no se cicle entonces se corrobora mediante un if si es que hay empate y ambos jugadores no tienen mas cartas para competir, en este caso se declara empate y entonces se finaliza la operación de el método juego().
En el caso de que no hubiera empate o de que al menos uno de los jugadores tenga cartas, se

volverá a ciclar el while que contiene toda la logica que compete a las reglas del juego. Se evaluaran nuevamente las condiciones para saber si hay un ganador y de no ser asi comenzara un nevo ciclo de competencia.

Clase Mazo:
Esta tendrá a cargo el funcionamiento del mazo, como corroborar que se haya construido correctamente, de acuerdo con diferentes criterios. También se encargara de transferir las cartas entre los jugadores, entre otras cosas.
Esta clase contiene o esta compuesto por :

- El constructor Mazo(String nombre, **int** cantidad) utilizado en la construcción de un mazo genérico.
- El constructor Mazo(String nombre) utilizado en la construcción de un mazo cargado desde archivo.
- Contara con una arrayList llamado “cartas” en el cual tendrá todas las cartas instanciadas para el juego. También provee de métodos para obtener diferentes valores de este arrayList, y de algunos otros para la agregación y trasferencia de las cartas, otro método mas especifico es el de obtenerCarta(string nombreCarta) el cual devuelve una carta de acuerdo con un string pasado como parámetro.
- Contara con una arrayList llamado cartasEnJuego que contendrá las cartas que se estén disputando en un empate

El metodo mazoCorrecto() verifica la cantidad de cartas seteada en el mazo se corresponda con la cantidad de catas dentro del arrayList cartas. También utiliza la primer carta del mazo para comparar con todas las demás mediante un ciclo for, comprara mediante el método equals que es redefinido en la clase carta y realiza una comparación bajo ciertos criterios que se mencionaran cuando se hable de esta ultima clase.

Clase Carta:
Con esta clase se modelara el comportamiento necesario para representar las cartas en el juego.Albergara un conjunto de atributos propios y métodos como por ejemplo el método equals(), método redefinido con el cual se compara una carta con otra.
Esta clase contiene o esta compuesto por :

- El constructor Carta(nombre, cantdadAtributos) usado para la construcción de un mazo genérico.
- El constructor Carta(nombre) usado para la construcción de un mazo cargado desde archivo.
- un arrayList denominado atributos, el cual contiene todos los objetos atributos que posee la carta. La principales funcionalidades de esta clase están relacionados con la forma en que se podrán evaluar sus atributos, obtenerlos, etc.

El método equals, ya mencionado . Es uno de los métodos mas importantes de esta clase, se ecarga de llevar conjuntamente con la clase carta el trabajo de comprobar si un mazo es correcto, se utiliza este método para comparar la carta actual con otra pasada por parámetro. En esta primeramente verifica si el objeto es una instancia de una carta mediante un if, y de ser así entran en juego una serie de if anidados que realizaran comprobaciones de acuerdo con diferentes criterios para tomar la decisión de que si una carta es igual o no.
El primero de estos if corrobora que la carta cuente con mas 3 o menos de 8 atributos, que es una de los requerimientos impuestos para el juego.

El segundo if corrobora que la cantidad de atributos sea igual a la definida en su variable cantidadAtributos, para corroborar que esta carta este con la cantidad adecuada y seteadas para todas las cartas.
El tercer if corrobora que la cantidad de atributos sea igual a la carta con la cual se esta comparando. De ser verdad se comprobara mediante dos for anidados que recorrerán los arreglos atributos de ambas cartas y verificaran que todos los atributos de la primera se encuentren en la segunda y que los mismos no se encuentren repetidos, además se comprobaran que todos los atributos con el mismo nombre posean la misma contienda. Esta ultima comprobación también en un requerimiento impuesto por el juego. De cumplirse todas las condiciones entonces se retornara true indicando que ambas cartas son iguales, en caso contrario se retornara false.

Clase Atributo:
Esta superclase es abstracta y además define un método template. Esta decisión fue a causa de que hay un método que ejecuta una operación de forma similar pero con una pequeña variación de acuerdo a la contienda del atributo. Por esta razón se decidió de hacer un método template e implementar la funcionalidad que varia en una subclase especifica, dando lugar a dos nuevas clases, AtributoMayor y atributoMenor que a la hora de realizar una operación de contienda reescribirán un método abstracto que incorpora esa variación y reutilizara el resto del método template que es común a ambas clases.
El método template es “competencia” y utiliza el método abstracto “compitePorContienda” con este método se podrá saber si el atributo local es mayor al atributo que se paso como parámetro.
Esta clase contiene o esta compuesto por :

- El constructor Atributo(String nombre, double valor, String operacion, String unidad) con el mismo se instanciaran todos los valores de un nuevo atributo.
- competencia(Atributo atriCarta2) descripto anteriormente. Comparar 2 atributos y devolver el ganador o un empate.

Clase AtributoMayor:
Esta es una subclase de Atributo. La decisión que tome ante su creación fue la de que esta permite crear un atributo según la contienda “>” o “<” y de que de esta forma se puede reutilizar todo lo implementado en Atributo y además dejo a cargo a esta clase de realizar una operación correspondiente a su contienda, de esta forma cuando se compite al por mayor esta clase reutiliza un método template heredado “competencia”, sobrescribiendo el método compitePorContienda() en se cual modifica el calculo para evaluar si es mayor o no a otro atributo pasado por parámetro. Retornando el ganador de ambos o un empate.
De esta forma se hace mas simple la competencia ya que no hay que hacer diferencia entre cuando un atributo es al por mayor o cuando es al por menor, mediante el binding dinámico y el polimorfismo se podrá encontrar el método correspondiente para llevar a cabo la evaluación delatributo de acuerdo con su contienda.

Clase AtributoMenor:
Esta subclase de Atributo es prácticamente igual a la clase AtributoMenor solo que en esta se sobrescribe compitePorContienda() en se cual modifica el calculo para evaluar si es menor o no a otro atributo pasado por parámetro. Retornando el ganador de ambos o un empate.

Clase Jugador:
Esa clase se creo para englobar el comportamiento de los jugadores, los cuales son principalmente manejar su arrayList de objetos cartas, las cuales recibe una a una al comienzo del juego mediante

el metodo repartirMazo correspondiente al mazo. Ya una vez con sus cartas entonces brindara la posibilidad de obtener una carta especifica para su post evaluación durante una competencia.
Esta clase contiene o esta compuesto por :

- El constructor Jugador(String nombre) el cual además de inicializar el nombre, también inicializa el arrayList cartas
- arrayList cartas el cual ya se menciono, y el cual posee las cartas del jugadores.

Algunos de sus metodos mas importantes pueden ser :

- obtenerCarta(nombre) al cual se le pasa un string y según este string se busca la carta cuyo nombre fue pasado como parámetro. Para esto se recorre mediante un ciclo for cada carta de su arreglo, en caso de no encontrarse se retorna null.