EJERCICIO DE LA GUERRA

1.- OBJETIVO DEL EJERCICIO

Este ejercicio tiene por objetivo repasar los siguientes conceptos vistos en clase:

- definición de clases
- definición de clases abstractas
- definición de interfaces
- herencia
- polimorfismo
- números aleatorios

Y practicar con:

- LocalTime

2.- PLANTEAMIENTO DEL JUEGO

Vamos a implementar una aplicación Java que represente una batalla entre dos bandos: el bando Blanco y el bando Negro.

Cada bando tendrá:

- dos sanitarios: un médico y un cirujano
- un guerrero
- un sniper
- 4 civiles

En la aplicación habrá dos jugadores jugadorBlanco perteneciente al bando Blanco y que por lo tanto ataca al bando Negro, y el jugadorNegro que hace lo contrario. Cada jugador tendrá un turno y podrá hacer una acción en ese turno. El juego acaba cuando todos los personajes de un equipo han muerto.

3.- ACCIONES

Las acciones se ejecutan desde el programa principal y llaman a los métodos de las clases.

Antes de empezar el juego, hay que crear las dos listas de personajes correspondientes a los dos bandos. Y entonces empieza el juego. Supongamos que empiezan siempre los blancos.

Antes de seleccionar cualquier acción,

- se actualizará la cantidad de vida de todos los personajes que hayan sido heridos (acuchillados o disparados) en el pasado. Si ha pasado más de n minutos (n se determinará al principio de la partida como una constante MINUTOS), la vida es nula. Se considera que el personaje ha fallecido.
- se visualizará la información de los dos bandos para que el jugador pueda decidir qué jugada realizar.

Las acciones se mostrarán al usuario mediante un menú y son las siguientes:

1.- **Atacar**: La acción *Atacar* debe preguntar al jugador qué personaje del bando propio ataca y a qué personaje del bando contrario. Se comprobará que el atacante es guerrero o sniper ya que sólo ellos pueden atacar. Pueden atacar a cualquier personaje que no sea de su equipo, también habrá que comprobarlo. Se puede atacar a un fallecido? Lo que quiera el programador. Esta acción deberá llamar a los métodos:

- atacar de las clases Guerrero o Sniper.
- Huye de las clases Civil o Sanitario si el atacado pertenece a una de estas clases.

2.- **Curar**. La acción *Curar* debe preguntar al jugador qué personajes del bando propio cura y es curado. Sólo pueden curar los sanitarios, deberá comprobarse. Los médicos pueden curar los personajes acuchillados y los cirujanos curan todo: sacan balas y curan los acuchillados. Los sanitarios se pueden curar a sí mismos (como Rambo).
Esta acción deberá llamar a los métodos:

- curar de las clases Cirujano o Médico.

3.- **Robar Material Sanitario**. La acción "*Robar Material Sanitario*" debe preguntar al jugador qué personaje roba y a quién le roba. Los guerreros son los únicos que pueden robar material sanitario tanto a sanitarios de su propio equipo como del contrario cuando el sanitario ha fallecido. El material robado debe donarse a un sanitario del bando propio que esté vivo y que tenga la menor cantidad de material sanitario.

Habrá que comprobar que los personajes son correctos.
Esta acción deberá llamar a los métodos:

- robar de la clase Guerrero.

4.- **Robar Balas**. Un sniper puede robarle las balas a un sniper del bando contrario si ha fallecido. Entonces las balas robadas se añaden a las balas propias.

La función robar debe indicar a qué personaje se roba.

La acción "*Robar Balas*" debe preguntar al jugador qué personaje roba y a quién le roba. Los snipers son los únicos que pueden robar balas.

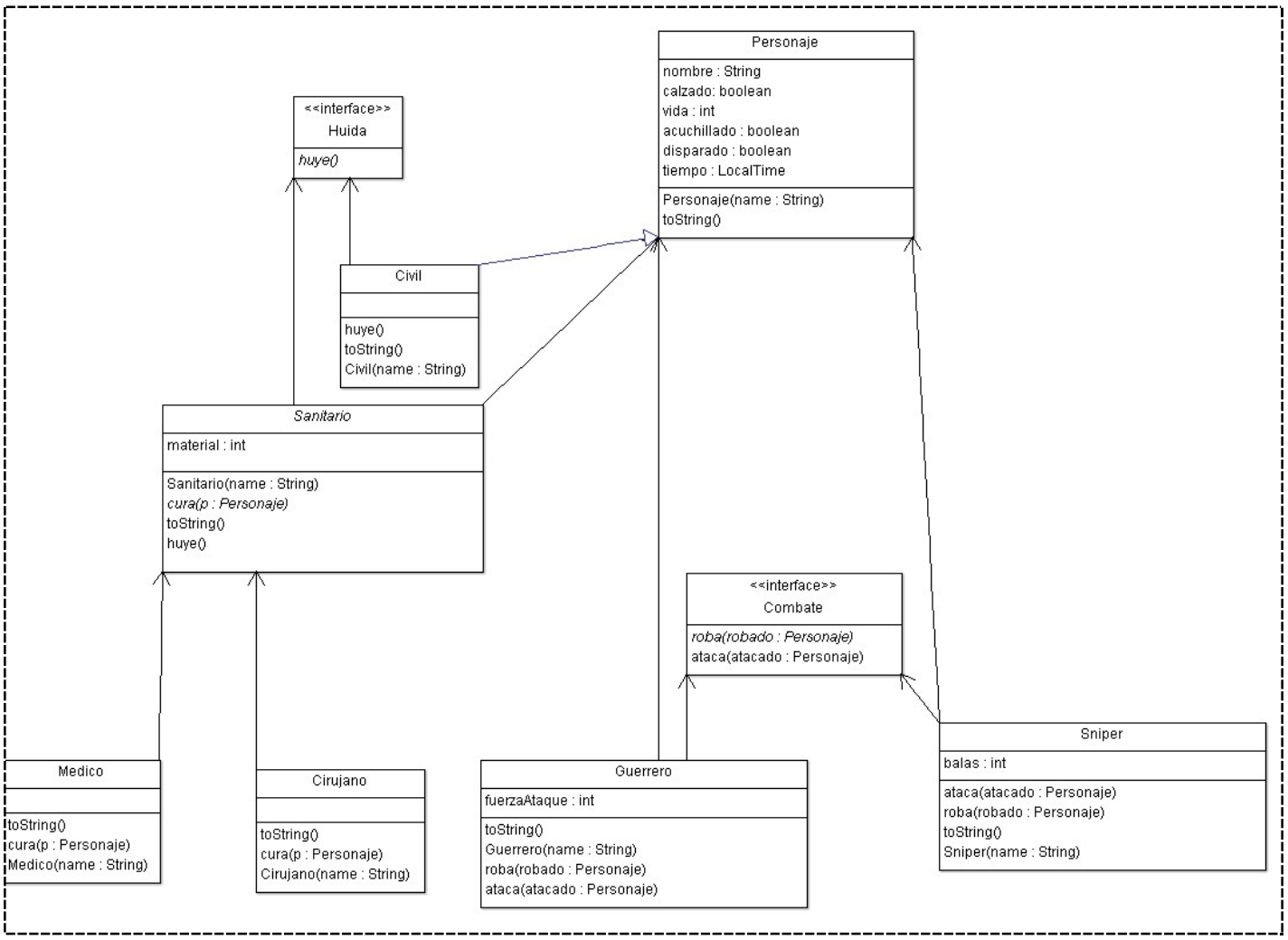
Habrá que comprobar que los personajes son correctos.
Esta acción deberá llamar a los métodos:

- robar de la clase Sniper.

0.- **Salir**. Se termina el programa.

4.- CLASES E INTERFACES

La estructura de clases e interfaces que intervienen en el juego es la siguiente:



Además de los métodos que se listan a continuación, se añadirán tantos get/set como se considere.

- La clase **PERSONAJE** es una clase abstracta
  - *Constantes de clase*
    - VIDA_INICIAL: vida inicial de un personaje
    - MINUTOS: tiempo que debe transcurrir desde que un personaje es herido hasta que muere (el atributo vida pasa a valer 0) si no le curan.
  - *Atributos*
    - Nombre: String
    - Calzado: boolean – Indica si el personaje lleva calzado o no
    - Vida: entero- Indica el nivel de vida que tiene el personaje
    - Acuchillado: boolean – Indica si el personaje ha sido acuchillado
    - Disparado: boolean – indica si el personaje ha sido disparado
    - Tiempo: LocalDate. Indica el momento en el que un personaje haya sido disparado o acuchillado. Sólo se tiene en cuenta si acuchillado o disparado es true.
  - *Constructor*: constructor al que se le pasa el nombre. Los demás atributos tienen el

siguiente valor por defecto. Calzado: true; acuchillado: false; disparado: false; tiempo: null; vida: VIDA_INICIAL (es una constante)
    - *Métodos*
        - toString(): devuelve una cadena de caracteres con la información de un personaje.

- El interfaz **HUIDA**
    - Atributos: no hay
    - Métodos abstractos:
        - Huye() : devuelve void y no recibe parámetros

- La clase **CIVIL** hereda de la clase PERSONAJE y del interfaz Huida
    - *Atributos*
        - Los que hereda de la clase PERSONAJE
    - *Constructor*: constructor al que se le pasa el nombre. Llama al constructor del padre
    - *Métodos*
        - toString():devuelve una cadena de caracteres con la información de un civil.
        - Huye(): Cuando un civil huye tiene un 50% de posibilidades de perder los zapatos. En ese caso, el atributo *Calzado* pasa al valor false.

- La clase **SANITARIO** es una clase abstracta y hereda de la clase PERSONAJE y del interfaz HUIDA

    - *Constantes de clase*
        - MATERIAL_INICIAL: cantidad de material inicial que tiene el sanitario.
    - *Atributos*
        - Los heredados de la clase PERSONAJE
        - Material: entero. Indica la cantidad de material que tiene el sanitario
    - *Constructor*: constructor al que se le pasa el nombre. Los demás atributos tienen valores por defecto. Material: MATERIAL_INICIAL. Y los demás atributos según el constructor de la clase Personaje
    - *Métodos*
        - toString():devuelve una cadena de caracteres con la información de un sanitario.
        - Huye(): Cuando un sanitario huye tiene un 33% de posibilidades de perder los zapatos. En ese caso, el atributo *Calzado* pasa al valor false.
        - Cura(): método abstracto. Recibe como parámetro un objeto de la clase Personaje y devuelve void.

- La clase **MEDICO** hereda de la clase SANITARIO
    - *Atributos*
        - Los que hereda de la clase SANITARIO
    - *Constructor*: constructor al que se le pasa el nombre. Llama al constructor del padre
    - *Métodos*
        - Método huye() que hereda de Sanitario
        - toString():devuelve una cadena de caracteres con la información de un médico.
        - Cura(): Un médico puede curar a un personaje acuchillado si tiene material (que no sea nulo). Cuando cura el médico pierde 1/3 de su material. Al curar a un personaje, éste deja de estar acuchillado, el atributo Tiempo pasa a ser nulo, y la vida se incrementa en 1.

- La clase **CIRUJANO** hereda de la clase SANITARIO
  - *Atributos*
    - Los que hereda de la clase SANITARIO
  - *Constructor*: constructor al que se le pasa el nombre. Llama al constructor del padre
  - *Métodos*
    - Método huye() que hereda de Sanitario
    - toString():devuelve una cadena de caracteres con la información de un cirujano.
    - Cura(): Un cirujano puede curar a un personaje disparado o acuchillado si tiene material (que no sea nulo). Cuando cura el médico pierde 1/3 de su material. Al curar a un personaje, éste deja de estar acuchillado o , el atributo Tiempo pasa a ser nulo, y la vida se incrementa en 1.

- El interfaz **COMBATE**
  - Atributos: no hay
  - Métodos abstractos:
    - Roba() : devuelve void y recibe como parámetro el personaje al que se va a robar
    - Ataca(): devuelve void y recibe como parámetro el personaje al que se ataca

- La clase **GUERRERO** hereda de la clase PERSONAJE y del interfaz Combate
  - *Constantes de clase*
    - FUERZA_ATAQUE: entero. Fuerza inicial que tiene el guerrero para atacar.
  - *Atributos*
    - Los que hereda de la clase PERSONAJE
    - fuerzaAtaque: entero – Indica qué poder tiene el guerrero para atacar
  - *Constructor*: constructor al que se le pasa el nombre. Los demás parámetros tienen valores por defecto. fuerzaAtaque se inicializa al valor de FUERZA_ATAQUE . Para el resto de atributos se llama al constructor del padre
  - *Métodos*
    - toString():devuelve una cadena de caracteres con la información de un guerrero.
    - Roba(): Un guerrero roba material sanitario a un sanitario del equipo contrario o del propio (si éste estuviera muerto) para dárselo a uno de su propio equipo que esté vivo y que tenga la menor cantidad de material sanitario. El sanitario robado se queda sin material (material=0). El sanitario que recibe el material debe actualizar su cantidad de material
    - Ataca(): un guerrero ataca a cuchillo. El impacto sobre el personaje atacado es
      - la vida del atacado se reduce (vida=vida-fuerzaAtaque del atacante)
      - el atacado pasa a estar acuchillado.
      - Se debe actualizar la hora del ataque a la hora actual

- La clase **SNIPER** hereda de la clase PERSONAJE y del interfaz Combate
  - *Constantes de clase*
    - NUMERO_BALAS: entero. Número inicial de balas que tiene el sniper para disparar.
  - *Atributos*
    - Los que hereda de la clase PERSONAJE
    - numeroBalas: entero – Indica cuántas balas tiene
  - *Constructor*: constructor al que se le pasa el nombre. Los demás parámetros tienen valores por defecto. numeroBalas se inicializa al valor de NUMERO_BALAS. Para el

resto de atributos se llama al constructor del padre

- *Métodos*
  - toString():devuelve una cadena de caracteres con la información de un sniper.
  - Roba(): Un sniper puede robarle las balas a un sniper del bando contrario si ha fallecido. Entonces las balas robadas se añaden a las balas propias. Ell sniper robado se queda sin balas.
  - Ataca(): un sniper dispara. Cada vez que dispara pierde una bala. El impacto sobre el personaje atacado es
    - la vida del atacado se reduce (vida=vida/2)
    - el atacado pasa a estar disparado.
    - Se debe actualizar la hora del ataque a la hora actual

5.- RECAPITULATIVO DE LAS CONSTANTES

- La vida inicial de un personaje (VIDA_INICIAL), el material de un sanitario (MATERIAL_INICIAL), las balas de un sniper (NUMERO_BALAS), la fuerza de ataque de un guerrero (FUERZA_ATAQUE) y el tiempo transcurrido desde que un personaje es atacado hasta que fallece (MINUTOS) son constantes.
- Se deben declarar en sus correspondientes clases
- Sus valores se dejan a la elección del programador. Las sugerencias son: VIDA_INICIAL: 10: MINUTOS: 5; MATERIAL_INICIAL: 25; FUERZA_ATAQUE: 4; NUMERO_BALAS: 4

6.- PASOS

paso 1: montar la estructura de clases e interfaces. Añadir para cada clase los constructores, los getter/setter y el método toString

paso 2: desarrollar la función menu que visualiza las posibles acciones

paso 3: desarrollar el método actualizarVida que se encarga de actualizar el atributo vida para todos los personajes que han sido heridos, en función del tiempo que ha pasado desde que ocurrió.

Paso 4: desarrollar el método visualizarInformacion que recorre los dos bandos visualizando todos los datos. Deberá llamar al método toString

Paso 5: desarrollar el método atacar en las clases Guerrero y Sniper

Paso 6: desarrollar el método robar en las clases Guerrero y Sniper

Paso 7: desarrollar el método huye en las clases Civil y Sanitario.

Paso 8: desarrollar el método curar en las clases Medico y Cirujano.

Paso 9 : desarrollar la gestión del menú que llamará a los métodos anteriores.

7.- ANEXOS

Números aleatorios

Os recuerdo lo que hacíamos para utilizar los números aleatorios.

1.- Definir un objeto de la clase Random
```
public static Random rnd = new Random();
```

2.- Generar un número aleatorio entre 0 y n
```
int numero=rnd.nextInt(n);
```

Si necesitamos una probabilidad del 50%, n será 2 , y si es del 33% n será 3.
Se utilizará en el método huye de los civiles y de los sanitarios.

LocalTime es una clase predefinida de Java.

Lista de métodos que nos vienen bien para este ejercicio :

1.- El método **of** permite generar un objeto de tipo LocalTime a partir de una hora y unos minutos.

```
Static LocalTime     of(int hour, int minute)
```

Ejemplo de uso:

```
LocalTime horaAtaque;
horaAtaque=LocalTime.of(12, 30);
```

2.- El método **plusMinutes** añade una cantidad de minutos a un objeto de tipo LocalTime . Hay que pasarle los minutos a sumar

```
LocalTime              plusMinutes(long minutesToAdd)
```

Ejemplo de uso:

```
LocalTime horaAtaque;
horaAtaque=LocalTime.of(12, 30);
horaPosterior=horaAtaque.plusMinutes(20); //horaPosterior es 20 minutos posterior a
                                          //horaAtaque
```

3.- Los métodos **isAfter e isBefore permiten comparar dos tiempos**

```
boolean              isAfter(LocalTime other)

boolean              isBefore(LocalTime other)
```

Ejemplo de uso:

```
if (  horaA.isAfter( horaB ))  System.out.println(" horaA es posterior a horaB");
```