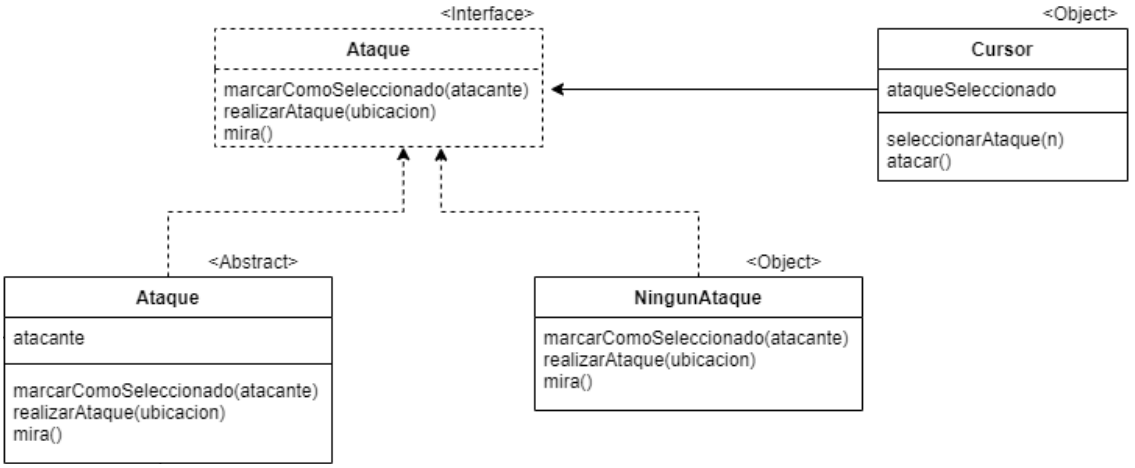
Explicación y aplicación de conceptos teóricos

- **Polimorfismo**: ¿Cuál es el mensaje polimórfico? ¿Qué objetos lo implementan? ¿Qué objeto se aprovecha de ello?

El cursor usa polimórficamente a cualquier objeto con la interfaz Ataque: es decir los mensajes “marcarComoSeleccionado”, “realizarAtaque” y “mira”.
Cuando seleccionamos un ataque, se guarda en el cursor una referencia a este (en la variable ataqueSeleccionado), y el cursor luego manda distintos mensajes a dicho ataque. Al realizar el ataque, se resetea el valor de ataqueSeleccionado a ningunAtaque.
La interfaz Ataque es implementada por cualquier objeto que herede la clase Ataque y también por el objeto ningunAtaque, que define ciertos comportamientos que debería tener el cursor cuando no hay ningún ataque seleccionado.



- **Colecciones**: ¿Qué operaciones de colecciones se utilizan? ¿Usaron mensajes con y sin efecto? ¿Para qué?

Uno de los métodos con uso de colecciones es *posicionesAtacables()*. Acá lo que hacemos es tomar la colección con las posiciones de las casillas y filtrar las posiciones con distancia menor al rango máximo del ataque, un atributo propio del mismo. Este mensaje no posee efecto, ya que no modifica la colección original, sólo nos retorna los valores de las posiciones.

```
override method posicionesAtacables() = tablero.posicionesCasillas().filter({ posicion => self.distanciaMenorA(posicion, rangoMaximo + 1) })
```

Una de las operaciones con colecciones es la de crear casilleros. Dada una colección de casillas, se utilizan los métodos de *crearFila* y *crearCasillas*, en forma de iteración para poder configurar el tablero de juego, siendo estos mensajes con efecto.

```
method crearFila(n) { tamanioHorizontal.times({i => casillas.add(new Casillero(coordenadas = new Coordenadas(x = i, y = n)))}) }
method crearCasillas() { tamanioVertical.times({i => self.crearFila(i)}) }
```

Una de las operaciones con colecciones, es la de agregar personajes. Cada jugador entiende el mensaje *agregarPersonajes(listaPersonajes)* que recibe una colección de personajes. De esta manera, podemos agregar a cada uno de los elementos de la *listaPersonajes*, mediante un forEach, a una colección *personajes*. A su vez, obteniendo el personaje podemos identificar a qué jugador pertenece.

```
method agregarPersonajes(listaPersonajes){
    listaPersonajes.forEach({personaje => self.agregarPersonaje(personaje)})
}

  jugador1.agregarPersonajes([soldadoNaziJp1, healerJp1])
  jugador2.agregarPersonajes([soldadoNaziJp2, healerJp2])
```

```
method agregarPersonaje(personaje) {
    personajes.add(personaje)
    personaje.jugador(self)
}
```

- **Clases**: ¿Usan clases? ¿Por qué? ¿Dónde o cuándo se instancian los objetos?

Sí, se usan clases porque se quieren modelar varias instancias de distintos conceptos que aparecen en el juego, como tropas para los jugadores, además de los ataques que pueden tener los personajes. Por ejemplo, se usa la clase Personaje porque un Jugador puede tener varias instancias de un personaje que tengan el mismo comportamiento, como varios soldados o healers. Estos objetos se instancian al comenzar el juego donde cada jugador va a tener una cantidad de tropas inicial.

- **Herencia**: ¿Entre quiénes y por qué? ¿Qué comportamiento es común y cuál distinto?

Podemos ver herencia entre los objetos *jugador1* y *jugador2*, que heredan el comportamiento de la clase jugador. Lo único que diferencia su comportamiento es la imagen que cada uno posee para indicar su victoria mediante *pantallaGanadora.*

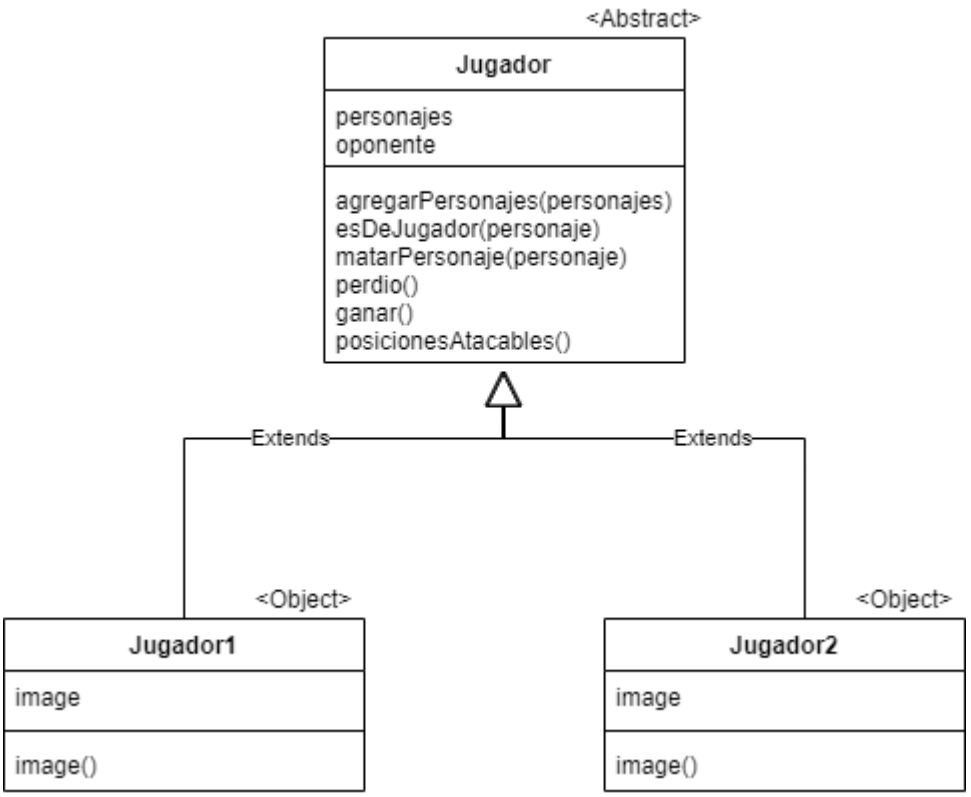
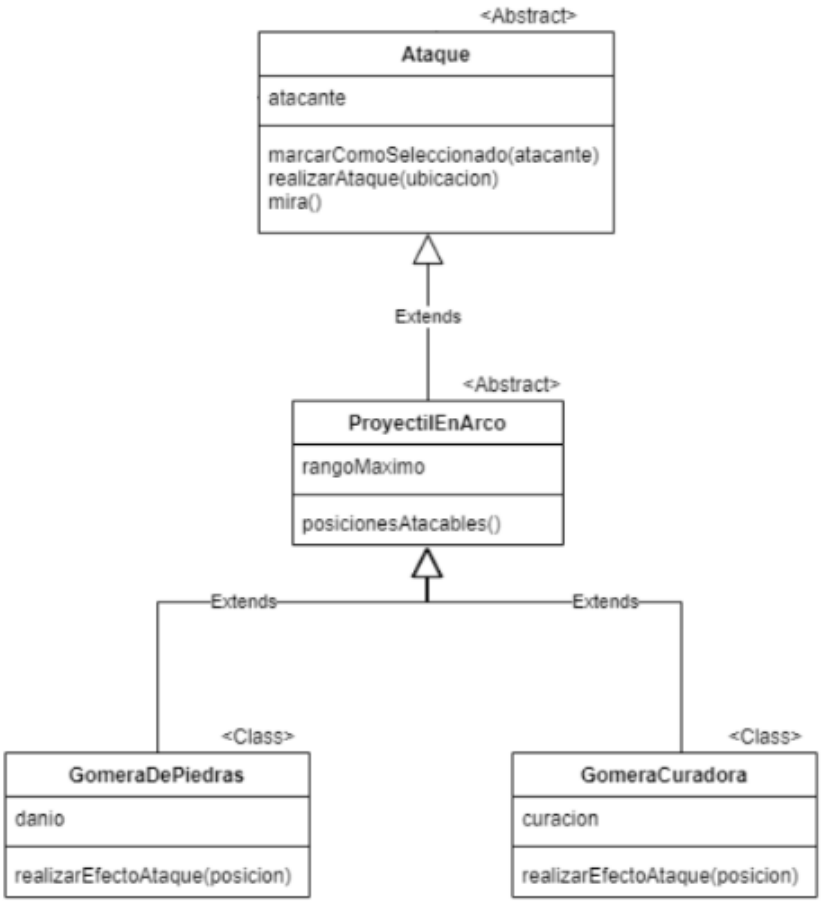


Diagrama de clases mostrando la herencia en los jugadores

También hay herencia con los distintos tipos de ataques. Todos los ataques van a heredar la clase abstracta Ataque, que tiene los comportamientos comunes a todos los ataques. Por ejemplo cambiar la mira cuando el cursor está sobre un casillero que se encuentra fuera de rango, o guardar en una variable el atacante que lo va a utilizar. Hay comportamiento distinto en los efectos y los rangos de los ataques.



En este diagrama mostramos dos clases abstractas, Ataque, que es heredada por todos los ataques, y ProyectilEnArco, que es un tipo de ataque. ProyectilEnArco es a su vez heredada por GomeraDePiedras y GomeraCuradora, que son clases concretas. Estas sí se instancian y son ataques que utilizan los jugadores. Es necesario tener ProyectilEnArco porque este se encarga del comportamiento del rango (define los casilleros que se pueden atacar), que es compartido por ambos tipos de ataque.