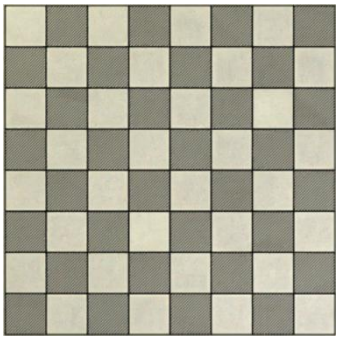
Ajedrez

El ajedrez consiste en un tablero de 8x8 casillas blancas y negras dispuestas de manera alternada

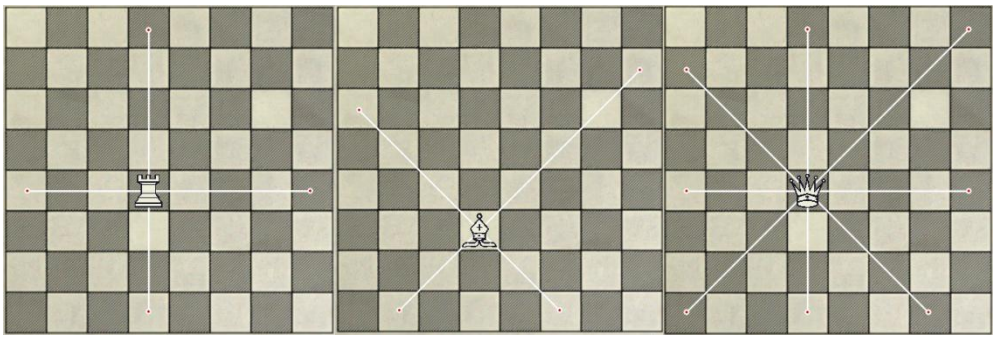


De las piezas del ajedrez considere solo los alfiles, torres y damas (puede haber cualquier cantidad de estas).

Los alfiles se mueven en diagonal (note que un alfil solo podrá moverse por casillas de del mismo color).

Las torres se mueven en dirección horizontal o vertical.

Las damas se mueven lo mismo como alfiles que como torres, es decir en dirección diagonal, horizontal y en vertical.



Diremos que una pieza puede alcanzar una casilla siempre que dicha casilla este vacía, esté en la trayectoria que puede recorrer la pieza y no tenga ninguna otra pieza en el camino.

Se dice que una pieza puede comerse a otra si la posición que la otra ocupa es una casilla alcanzable por la primera y son de colores diferentes.

Una pieza es amenazada, si puede ser “comida” por alguna otra.

Implemente la interfaz siguiente:

```
public interface ITableroAjedrez
{
    /// <summary>
    /// Coloca una pieza en un casilla específica del tablero.
    /// Si la casilla está ocupada lanza InvalidOperationException.
    /// </summary>
    void ColocaPieza(Pieza pieza, Casilla casilla);

    /// <summary>
    /// Devuelve si la pieza que está en una casilla de origen
    /// puede moverse a una casilla destino.
    /// Si no existe una pieza en la casilla origen
    /// lanza InvalidOperationException.
    /// Note que una pieza puede moverse a una casilla ocupada si
    /// se la puede comer.
    /// </summary>
    bool PuedeMoverse(Casilla casillaOrigen, Casilla casillaDestino);

    /// <summary>
    /// Mueve la pieza de una casilla origen a una destino.
    /// De existir una pieza de distinto color en la casilla destino se
    /// comerá a ésta
    /// Si no existe una pieza en la casilla de origen o no se
    /// puede efectuar el movimiento lanza InvalidOperationException.
    /// </summary>
    void MuevePieza(Casilla casillaOrigen, Casilla casillaDestino);

    /// <summary>
    /// Devuelve todas las casillas en las que hay piezas que pueden
    /// moverse hacia una casilla
    /// determinada.
    /// Note que puede que sea vacío el iterador.
    /// </summary>
    IEnumerable<Casilla> PuedenMoverseHacia(Casilla casilla);


    /// <summary>
    /// Devuelve la cantidad de piezas de un color determinado
    /// que son amenazadas.
    /// </summary>
    int CantidadDeAmenazadas(ColorPieza color);

    /// <summary>
    /// Devuelve las casillas en las que hay piezas que son amenazadas.
    /// Note que el iterador puede ser vacío
```

```
        /// una colección vacia.
        /// </summary>
        IEnumerable<Casilla> Amenazadas { get; }

        /// <summary>
        /// Devuelve el conjunto de casillas en las que están las piezas
        /// que amenazan la mayor cantidad
        /// de piezas enemigas.
        /// Esto es, el conjunto de piezas sobre el tablero tal que
        /// no exista otra que amenace a más piezas que las del conjunto.
        /// Note que si ninguna pieza amenaza a otra,
        /// el resultado serían todas las piezas sobre el tablero,
        /// ya que todas amenazan a 0 piezas.
        /// De no existir piezas en el tablero el iterador será vacío.
        /// </summary>
        IEnumerable<Casilla> PiezasQueMasAmenazan { get; }

        /// <summary>
        /// Devuelve la pieza situada en la casilla especificada.
        /// Si no existe una pieza en la casilla, devuelve null.
        /// </summary>
        Pieza this[Casilla casilla] { get; }
}
```

Las coordenadas de una casilla esta dada por `Fila` y `Columna`, donde la esquina superior izquierda tiene coordenadas `(0,0)`, la esquina superior derecha `(0,7)`, la esquina inferior izquierda `(7,0)` y la esquina inferior derecha `(7,7)`. Todas las casillas que se pasarán como parámetro serán válidas (interiores al tablero), no se necesita que usted chequee su validez.

Ejemplo

Si tenemos el siguiente tablero:

La llamada al método `PuedenMoverseHacia(new Casilla(4,5))` devuelve una secuencia que contiene las casillas `(2,5)`,`(6,3)` y `(4,3)` que contiene a las dos damas y una de las torres.

La llamada al método `CantidadDeAmenazadas(ColorPieza.Blanca)` devuelve el valor 2, ya que solo dos torres de las blancas son amenazadas.

La propiedad `PiezasQueMasAmenazan` devuelve una secuencia que contiene las casillas `(2,5)` y `(6,3)` que contienen en este caso a las dos damas, ya que cada una de ella amenaza a dos torres (el máximo de amenazadas en este ejemplo).

Las definiciones de las clases `Casilla`, `Pieza` son

```
public class Casilla
{
    public Casilla(int fila, int columna) { ... }

    public int Fila { get; set; }

    public int Columna { get; set; }
}

public class Pieza
{
    public Pieza(TipoPieza tipo, ColorPieza color) { ... }

    public TipoPieza Tipo { get; }

    public ColorPieza Color { get; }
}
```

Estas definiciones, la de la interfaz, así como los enumerativos `ColorPieza` y `TipoPieza` se encuentran en el ensamblado `Weboo.Examen.dll` y que ya está agregado como referencia dentro de la plantilla de solución. Dentro de la plantilla de solución suministrada, usted debe completar la definición de la clase `TableroAjedrez` que implementa la interfaz antes descrita. La implementación de esta clase debe tener al menos un constructor sin parámetros.