RETO 2: ABSTRACCIÓN

David Muñoz Sánchez
Juan Manuel Rodríguez Gómez

(Documentación del proyecto generada con Doxygen)

Doble Grado en Ingeniería Informática y Matemáticas

Estructuras de Datos

Curso 2020 – 2021

Clase Pieza

```
/**
   * @file pieza.h
   * @authors David Muñoz Sánchez, Juan Manuel Rodríguez Gómez
   * @date Octubre 2020
   */

#ifndef PIEZA_H
#define PIEZA_H

#include <iostream>

/**
   * @class Pieza
   * @brief Clase usada para representar una pieza del Tetris mediante una matriz dinámica
   */
class Pieza {

        private:
                bool ** pieza;          /**< Pieza del Tetris */
                int filas, columnas;    /**< Número de filas y columnas de la matriz de la
                                             pieza */

                /**
                   * @brief Reserva memoria y, si es necesario, libera la ya existente
                   * @param f Número de filas de la matriz de la pieza
                   * @param c Número de columnas de la matriz de la pieza
                   */
                void reservarMemoria(int f, int c);

                /**
                   * @brief Libera la memoria reservada
                   */
                void liberarMemoria();

                /**
                   * @brief Copia los datos de una pieza
                   * @ param p Pieza cuyos datos se copian
                   */
                void copiar(const Pieza& p);

                /**
                   * @brief Genera una pieza aleatoria
                   */
                void formarPieza();
```

```
public:
        /**
          * @brief Constructor por defecto
          */
        Pieza();

        /**
          * @brief Constructor con parámetros
          * @param f Número de filas de la matriz de la pieza
          * @param c Número de columnas de la matriz de la pieza
          */
        Pieza(int f, int c);

        /**
          * @brief Constructor de copia
          * @param p Pieza que se va a copiar
          */
        Pieza(const Pieza& p);

        /**
          * @brief Destructor
          */
        ~Pieza();

        /**
          * @brief Sobrecarga del operador de asignación
          * @param p Rvalue de la asignación
          * @return this Lvalue de la asignación
          */
        Pieza& operator=(const Pieza& p);

        /**
          * @brief Devuelve el número de filas de la matriz de la pieza
          * @return Número de filas de la matriz de la pieza
          */
        int getFilas() const;

        /**
          * @brief Devuelve el número de columnas de la matriz de la pieza
          * @return Número de columnas de la matriz de la pieza
          */
        int getColumnas() const;

        /**
          * @brief Devuelve el valor que contiene una celda de la matriz de la pieza
          * @param f Fila de la matriz de la pieza
          * @param c Columna de la matriz de la pieza
          * @return Valor que contiene la celda [f][c] de la matriz (1 si dicha
          * celda está ocupada y 0 si no lo está)
          */
        int get(int f, int c) const;
```

```
        /**
           * @brief Establece el valor de una celda de la matriz de la pieza
           * @param f Fila de la matriz de la pieza
           * @param c Columna de la matriz de la pieza
           */
        void set(int f, int c);

        /**
           * @brief Genera una pieza aleatoria dentro de los diferentes tipos de pieza
           * que hay
           * @return Pieza generada aleatoriamente
           */
        Pieza generarPiezaAleatoria();

        /**
           * @brief Rota una pieza
           * @param p Pieza que se quiere rotar
           * @return Pieza rotada
           */
        Pieza rotarPieza(Pieza p);

        friend std::ostream & operator<<(std::ostream& os, const Pieza& p);
        friend std::istream & operator>>(std::istream& is, const Pieza& p);
};

/**
   * @brief Sobrecarga del operador de salida
   * @param os Flujo de salida
   * @param p Pieza que devuelve la información
   * @return Referencia a un flujo de salida
   */
std::ostream & operator<<(std::ostream& os, const Pieza& p);

/**
   * @brief Sobrecarga del operador de entrada
   * @param is Flujo de entrada
   * @param p Pieza que recibe la información
   * @return Referencia a un flujo de entrada
   */
std::istream & operator>>(std::istream& is, const Pieza& p);

#endif /* PIEZA_H */
```

Clase ColaPiezas

```
/**
   * @file colapiezas.h
   * @authors David Muñoz Sánchez, Juan Manuel Rodríguez Gómez
   * @date Octubre 2020
   */

#ifndef COLAPIEZAS_H
#define COLAPIEZAS_H

#include <iostream>
#include "pieza.h"

/**
   * @class ColaPiezas
   * @brief Clase usada para almacenar una secuencia de piezas, donde cada pieza es un
   * objeto de la clase Piezas, mediante un vector dinámico
   */
class ColaPiezas {

        private:
                Pieza * cola;           /**< Cola de piezas del Tetris */
                int n;                  /**< Número de piezas de la cola*/

                /**
                   * @brief Reserva memoria y, si es necesario, libera la ya existente
                   * @param n Número de piezas de la cola
                   */
                void reservarMemoria(int n);

                /**
                   * @brief Libera la memoria reservada
                   */
                void liberarMemoria();

                /**
                   * @brief Copia los datos de una cola de piezas
                   * @ param cp Cola de piezas cuyos datos se copian
                   */
                void copiar(const ColaPiezas& cp);

                /**
                   * @brief Aumenta o disminuye el tamaño de la memoria reservada (según
                   * sea n un número positivo o negativo)
                   * @param n Cantidad de memoria a redimensionar
                   */
                void redimensionar(int n);
```

```
public:
        /**
          * @brief Constructor por defecto
          */
        ColaPiezas();

        /**
          * @brief Constructor con parámetros
          * @param n Número de piezas de la cola
          */
        ColaPiezas(int n);
        /**
          * @brief Constructor de copia
          * @param cp Cola de piezas que se va a copiar
          */
        ColaPiezas(const ColaPiezas& cp);

        /**
          * @brief Destructor
          */
        ~ColaPiezas();

        /**
          * @brief Sobrecarga del operador de asignación
          * @param cp Rvalue de la asignación
          * @return this Lvalue de la asignación
          */
        ColaPiezas& operator=(const ColaPiezas& cp);

        /**
          * @brief Sobrecarga del operador de asignación de la suma
          * @param cp Rvalue de la asignación de la suma
          * @return this Lvalue de la asignación de la suma
          */
        ColaPiezas& operator+=(const ColaPiezas& cp);

        /**
          * @brief Sobrecarga del operador de asignación de la resta
          * @param cp Rvalue de la asignación de la resta
          * @return this Lvalue de la asignación de la resta
          */
        ColaPiezas& operator-=(const ColaPiezas& cp);

        /**
          * @brief Sobrecarga del operador de indexación
          * @param i Índice del elemento a obtener
          * @return Elemento de la cola de piezas según el índice dado
          */
        ColaPiezas& operator[](int i);
```

```
    /**
     * @brief Sobrecarga del operador de indexación constante
     * @param i Índice del elemento a obtener
     * @return Elemento de la cola de piezas según el índice dado
     */
    const ColaPiezas& operator[](int i) const;

    /**
     * @brief Devuelve el tamaño de la cola de piezas
     * @return Tamaño de la cola de piezas
     */
    int getTamanio() const;

    /**
     * @brief Establece el tamaño de la cola de piezas
     * @param n Tamaño de la cola de piezas
     */
    void setTamanio(int n);

    /**
     * @brief Devuelve la pieza almacenada en una determinada posición de la
     * cola de piezas
     * @param pos Posición de la pieza almacenada en la cola de piezas
     * @return Pieza correspondiente a la posición pos de la cola de piezas
     */
    Pieza getPieza(int pos) const;

    /**
     * @brief Establece una pieza en una determinada posición de la cola de
     * piezas
     * @param p Pieza que queremos establecer en la cola de piezas
     * @param pos Posición de la cola de piezas en la que queremos establecer
     * la pieza
     */
    void setPieza(Pieza& p, int pos);

    /**
     * @brief Extrae la primera pieza de la cola de piezas
     * @return Primera pieza de la cola de piezas
     */
    Pieza extraer();

    /**
     * @brief Añade una pieza al final de la cola de piezas
     * @param p Pieza que se va a añadir al final de la cola de piezas
     */
    void aniadir(Pieza& p);

    friend std::ostream & operator<<(std::ostream& os, const ColaPiezas& cp);
    friend std::istream & operator>>(std::istream& is, const ColaPiezas& cp);
};
```

```
/**
   * @brief Sobrecarga del operador de salida
   * @param os Flujo de salida
   * @param cp ColaPiezas que devuelve la información
   * @return Referencia a un flujo de salida
   */
std::ostream & operator<<(std::ostream& os, const ColaPiezas& cp);
/**
   * @brief Sobrecarga del operador de entrada
   * @param is Flujo de salida
   * @param cp ColaPiezas que recibe la información
   * @return Referencia a un flujo de entrada
   */
std::istream & operator>>(std::istream& is, const ColaPiezas& cp);

#endif /* COLAPIEZAS_H */
```

Clase Tablero

```
/**
   * @file tablero.h
   * @authors David Muñoz Sánchez, Juan Manuel Rodríguez Gómez
   * @date Octubre 2020
   */

#ifndef COLAPIEZAS_H
#define COLAPIEZAS_H

#include <iostream>
#include "pieza.h"
#include "colapiezas.h"

/**
   * @class Tablero
   * @brief Clase usada para representar el tablero del Tetris mediante una matriz dinámica
   */
class Tablero {

        private:
                bool ** tablero;          /**< Tablero del Tetris */
                int filas, columnas;      /**< Número de filas y columnas del tablero*/

                /**
                   * @brief Reserva memoria y, si es necesario, libera la ya existente
                   * @param f Número de filas del tablero
                   * @param c Número de columnas del tablero
                   */
                void reservarMemoria(int f, int c);

                /**
                   * @brief Libera la memoria reservada
                   */
                void liberarMemoria();

                /**
                   * @brief Copia los datos de un tablero
                   * @ param t Tablero cuyos datos se copian
                   */
                void copiar(const Tablero& t);

                /**
                   * @brief Aumenta o disminuye el tamaño de la memoria reservada (según
                   * sea n un número positivo o negativo)
                   * @param n Cantidad de memoria a redimensionar
                   */
                void redimensionar(int n);
```

```
public:
        /**
          * @brief Constructor por defecto
          */
        Tablero();

        /**
          * @brief Constructor con parámetros
          * @param f Número de filas del tablero
          * @param c Número de columnas del tablero
          */
        Tablero(int f, int c);

        /**
          * @brief Constructor de copia
          * @param t Tablero que se va a copiar
          */
        Tablero(const Tablero& t);

        /**
          * @brief Destructor
          */
        ~Tablero();

        /**
          * @brief Sobrecarga del operador de asignación
          * @param t Rvalue de la asignación
          * @return this Lvalue de la asignación
          */
        Tablero& operator=(const Tablero& t);

        /**
          * @brief Devuelve el número de filas del tablero
          * @return Número de filas del tablero
          */
        int getFilas() const;

        /**
          * @brief Devuelve el número de columnas del tablero
          * @return Número de columnas del tablero
          */
        int getColumnas() const;

        /**
          * @brief Devuelve el valor que contiene una casilla del tablero
          * @param f Fila de la matriz del tablero
          * @param c Columna de la matriz del tablero
          * @return Valor que contiene la celda [f][c] de la matriz (1 si dicha
          * celda está ocupada y 0 si no lo está)
          */
        int get(int f, int c) const;
```

```
/**
   * @brief Establece el valor de una casilla del tablero
   * @param f Fila del tablero
   * @param c Columna del tablero
   */
void set(int f, int c);

/**
   * @brief Desplaza una pieza hacia la izquierda, hacia la derecha o hacia
   * abajo del tablero
   * @param p Pieza a desplazar
   */
void desplazarPieza(const Pieza& p);

/**
   * @brief Comprueba si una pieza se puede desplazar en el tablero
   * @param p Pieza a desplazar
   * @return 1 si la pieza se puede desplazar y 0 si no se puede
   */
bool poderDesplazarPieza(const Pieza& p);

/**
   * @brief Comprueba si una pieza se puede rotar en el tablero
   * @param p Pieza a rotar
   * @return 1 si la pieza se puede rotar y 0 si no se puede
   */
bool poderRotarPieza(const Pieza& p);

/**
   * @brief Comprueba si una pieza encaja o no en el tablero
   * @param p Pieza a rotar
   * @return 1 si la pieza encaja y 0 si no se puede
   */
bool piezaEncaja(const Pieza& p);

/**
   * @brief Coloca una pieza en el tablero
   * @param p Pieza a colocar
   */
void colocarPieza(const Pieza& p);

/**
   * @brief Comprueba si una casilla del tablero está ocupada o no
   * @param p Pieza a colocar
   * @return 1 si la casilla está ocupada y 0 si no lo está
   */
bool casillaOcupada(int f, int c);
```

```
        /**
           * @brief Comprueba si una línea del tablero está completa o no
           * @param f Fila del tablero
           * @return 1 si la fila está completa y 0 si no lo está
           */
        bool lineaCompleta(int f);

        /**
           * @brief Borra una línea del tablero
           * @param f Fila del tablero
           */
        void borrarLinea(int f);

        /**
           * @brief Añadir una línea al principio del tablero. También se puede usar
           * para añadir una línea aleatoria y así aumentar la dificultad del juego.
           * @param f Línea anterior a la posición donde se quiere añadir la línea
           * @pre La línea introducida como parámetro debe estar entre 0 y n, siendo
           * n el número de filas de la matriz
           */
        void aniadirLinea(int f);

        friend std::ostream & operator<<(std::ostream& os, const Tablero& t);
        friend std::istream & operator>>(std::istream& is, const Tablero& t);
};

/**
   * @brief Sobrecarga del operador de salida
   * @param os Flujo de salida
   * @param t Tablero que devuelve la información
   * @return Referencia a un flujo de salida
   */
std::ostream & operator<<(std::ostream& os, const Tablero& t);

/**
   * @brief Sobrecarga del operador de entrada
   * @param is Flujo de entrada
   * @param t Tablero que recibe la información
   * @return Referencia a un flujo de entrada
   */
std::istream & operator>>(std::istream& is, const Tablero& t);

#endif /* TABLERO_H */
```

Clase Imagen y Clase Tetris

```
/**
   * @file imagen.h
   * @authors David Muñoz Sánchez, Juan Manuel Rodríguez Gómez
   * @date Octubre 2020
   */

#ifndef IMAGEN_H
#define IMAGEN_H

#include <iostream>
#include "pieza.h"
#include "colapiezas.h"
#include "tablero.h"

/**
   * @class Imagen
   * @brief Clase usada para representar gráficamente en la terminal todos los elementos del
   * juego Tetris (tablero, cola de piezas y cuadros de texto con diversa información de la
   * partida)
   */
class Imagen {

        private:
                int ancho, alto; /**< Dimensiones de la ventaja de juego */
                int x, y;        /**< Coordenadas para posicionar los diferentes elementos*/
                int color;       /**< Color de los diferentes elementos del juego*/

        public:
                /**
                   * @brief Constructor por defecto
                   */
                Imagen();

                /**
                   * @brief Constructor con parámetros
                   * @param ancho Anchura de la imagen
                   * @param alto Altura de la imagen
                   */
                Imagen(int ancho, int alto);

                /**
                   * @brief Establece el tamaño de la imagen
                   * @param ancho Anchura de la imagen
                   * @param alto Altura de la imagen
                   */
                void establecerTamanioImagen(int ancho, int alto);
```

```
        /**
          * @brief Establece la posición de la imagen en la ventana de juego
          * @param x Coordenada eje X
          * @param y Coordenada eje Y
          */
        void establecerPosicionImagen(int x, int y);

        /**
          * @brief Dibuja texto con un color aplicado
          * @param x Coordenada eje X
          * @param y Coordenada eje Y
          * @param texto Texto a dibujar
          * @param color Color del texto
          */
        void dibujarTexto(int x, int y, std::string texto y, int color);

        /**
          * @brief Establece el color de cualquier imagen (tablero, pieza, fondos, etc.).
          * Como en la clase Tetris tenemos ColaPiezas, y en la clase ColaPiezas
          * tenemos el método get, el cual devuelve una pieza de la clase Piezas,
          * podemos cambiar el color de las piezas sin problema.
          * @param color Color que se va a aplicar
          */
        void establecerColor (int color);
};

/**
  * @class Tetris
  * @brief Clase hija de la clase Imagen. Implementa completamente el juego del Tetris
  */

class Tetris : public Imagen {

    private:
        Tablero tablero;        /**< Tablero del juego */
        ColaPiezas cola;        /**< Cola de Piezas del juego*/

        int puntuacion;         /**< Puntuación del juego*/
        int nivel;              /**< Nivel del juego*/
        int lineas;             /**< Líneas del juego completadas*/
        int piezas;             /**< Piezas del juego colocadas*/
        bool estado_partida;    /**< Estado de la partida*/

    public:
        /**
          * @brief Constructor por defecto
          */
        Tetris();
```

```
/**
   * @brief Obtiene la puntuación del juego
   * @return Puntuación del juego
   */
int getPuntuacion() const;

/**
   * @brief Obtiene el nivel del juego alcanzado
   * @return Nivel del juego alcanzado
   */
int getNivel() const;

/**
   * @brief Obtiene el total de líneas del juego completadas
   * @return Líneas del juego completadas
   */
int getLineasCompletadas() const;

/**
   * @brief Obtiene el total de piezas del juego colocadas
   * @return Piezas del juego colocadas
   */
int getPiezasColocadas() const;

/**
   * @brief Obtiene el estado de la partida
   * @return Estado de la partida (1 si se está jugando y 0 si no se está jugando)
   * @pos Si devuelve 0, puede ser por diversos motivos que hay que
   * diferenciar (partida pausada o finalizada)
   */
int getEstadoPartida() const;

/**
   * @brief Establece la puntuación del juego
   */
void setPuntuacion();

/**
   * @brief Establece el nivel del juego alcanzado
   */
void setNivel();

/**
   * @brief Establece el total de líneas del juego completadas
   */
void setLineasCompletadas();

/**
   * @brief Establece el total de piezas del juego colocadas
   */
void setPiezasColocadas();
```

```
    /**
     * @brief Establece el estado de la partida (1 si se está jugando y 0
     * si no se está jugando)
     */
    void setEstadoPartida();

    /**
     * @brief Calcula la puntuación de la partida
     * @return Puntuación de la partida
     */
    int calcularPuntuacion();

    /**
     * @brief Calcula el nivel de la partida alcanzado
     * @return Nivel de la partida alcanzado
     */
    int calcularNivel();

    /**
     * @brief Establecerá la partida del Tetris con interfaz gráfica (para ello se
     * hará uso de los métodos de la clase padre Imagen)
     * @return Tetris con interfaz gráfica
     */
    Tetris establecerTetris();

    /**
     * @brief Comprueba si la partida del Tetris ha finalizado
     * @return 1 si ha finalizado y 0 si no ha finalizado
     */
    bool finalizarPartida() const;

    /**
     * @brief Lee la partida de Tetris
     * @param is Flujo de entrada
     * @param archivo_entrada Archivo de entrada, si no se pone nada, se usa
     * la consola
     */
    void leerPartida(std::ostream& is, std::string archivo_entrada = "");

    /**
     * @brief Guarda la partida de Tetris
     * @param os Flujo de salida
     * @param archivo_salida Archivo de salida, si no se pone nada, se usa la
     * consola
     */
    void guardarPartida(std::ostream& os, std::string archivo_salida = "");

    friend std::ostream & operator<<(std::ostream& os, const Tetris& tetris);
    friend std::istream & operator>>(std::istream& is, const Tetris& tetris);
};
```

```
/**
   * @brief Sobrecarga del operador de salida
   * @param os Flujo de salida
   * @param tetris Partida de Tetris que devuelve la información
   * @return Referencia a un flujo de salida
   */
std::ostream & operator<<(std::ostream& os, const Tetris& tetris);

/**
   * @brief Sobrecarga del operador de entrada
   * @param is Flujo de entrada
   * @param tetris Partida de Tetris que recibe la información
   * @return Referencia a un flujo de entrada
   */
std::istream & operator>>(std::istream& is, const Tetris& tetris);

#endif /* IMAGEN_H */
```