UNIVERSIDAD DE GRANADA

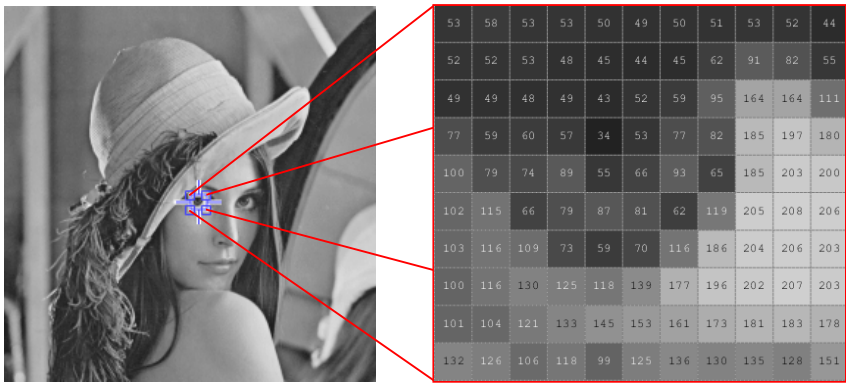
ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM

Curso 2021/2022



Guion de prácticas

Imaging1

Manejo de las primeras imágenes digitales

Marzo de 2022

Índice

1. Descripción

Como ya se indicó con anterioridad, las prácticas tienen como objeto principal las imágenes en blanco y negro. Una imagen todos la percibimos como una matriz de *(alto x ancho)* de píxeles. Por el hecho de ser en blanco y negro, cada píxel se puede representar mediante un byte, de ahí que centráramos nuestra atención sobre este tipo en la primera práctica.

Las funciones de bajo nivel que implementamos en Imaging0 para manipular los bytes, se van a encapsular dentro una clase Byte, junto con su espacio de datos, de modo que podamos usarla como componente base de una Imagen.

Una vez definido Byte, vamos a abordar la estructura interna de una imagen y su implementación dentro de una clase. Vamos a dotar a la clase con los primeros métodos para poderla manipular a bajo nivel. Para poner todo esto en práctica vamos a crear nuestras primeras imágenes inicialmente sin preocuparnos de formato y procederemos a su visualización con la ayuda de utilidades.

1.1. Arquitectura de la práctica

Como ya se indicó en la práctica anterior, la práctica `Imaging` se ha diseñado por etapas, las primeras contienen estructuras más sencillas, sobre las cuales se asientan otras estructuras más complejas y se van completando nuevas funcionalidades. A Imaging1 le corresponde la implementación de las clases Byte e Image, bloques **A'** y **B** respectivamente de la Figura 1.

A' Byte.cpp
: Implementa la clase Byte, incorporando y adaptando las funciones anteriores como métodos.

B Image.cpp
: Implementa la clase Image que contiene la información necesaria para su visualización. En una primera aproximación se usarán arrays estáticos.

2. Objetivos

El desarrollo de esta práctica Imaging1 persigue los siguientes objetivos:

- practicar el paso de parámetros por referencia
- practicar el paso de parámetros de tipo array
- practicar la definición de métodos de consulta y de modificación de clases,
- utilización de bibliotecas

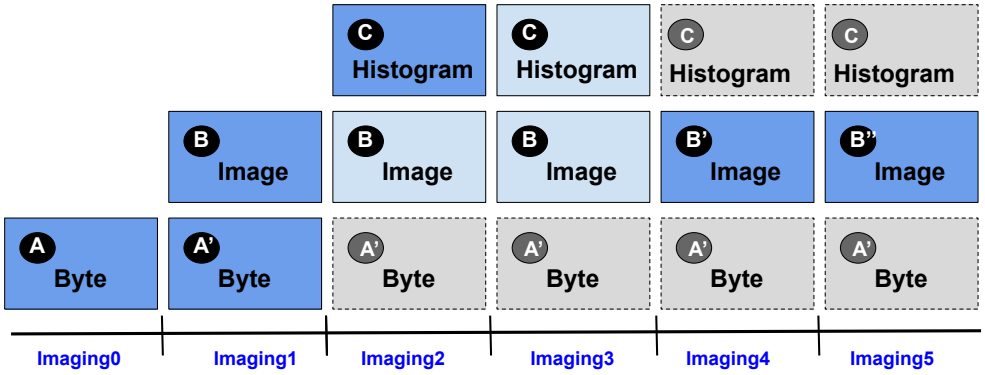


Figura 1: Arquitectura de las prácticas de MP 2022.

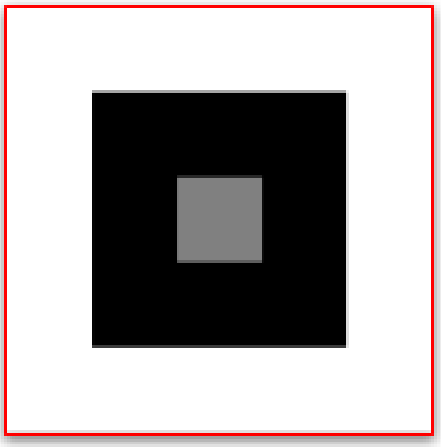
3. Imágenes

Para imágenes en blanco y negro, cada píxel se suele representar con un byte[1] (8 bits). El valor del píxel representa su tonalidad de gris que va desde el negro (0) hasta el blanco (255). Un píxel con valor 128 tendrá un gris intermedio entre negro y blanco. En la siguiente imagen se puede observar el valor de los píxeles para una pequeña porción de la imagen (Lena) ya presentada.

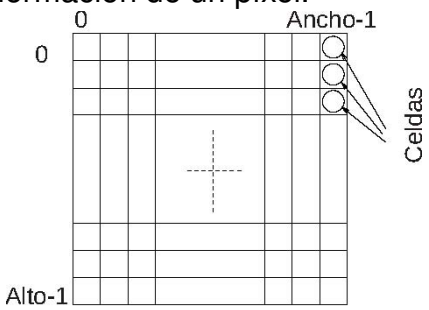
o esta otra imagen de 5x5 píxeles

[1] Recuerde que en C++ un ”unsigned char” almacena exactamente un byte.

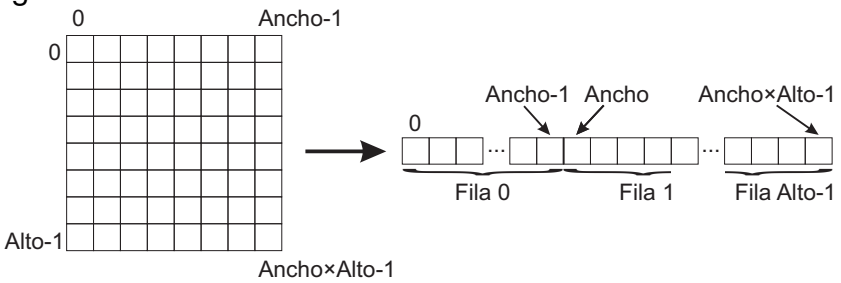
255	255	255	255	255
255	0	0	0	255
255	0	127	0	255
255	0	0	0	255
255	255	255	255	255



Desde un punto de vista de estructura de datos, una imagen se puede considerar como una matriz bidimensional de celdas. Cada celda de la matriz almacena la información de un píxel.



Pese a que una imagen se trata habitualmente como una matriz bidimensional de bytes, es usual representarla internamente como un vector en el que las filas se van guardando una tras otra, almacenando consecutivos todos los bytes de la imagen. Así, la posición 0 del vector tendrá el píxel de la esquina superior izquierda, la posición 1 el de su derecha, y así hasta el píxel de la esquina inferior derecha, como se muestra en la figura siguiente:



Así, se puede acceder fácilmente a las posiciones de la imagen de forma consecutiva pero, para acceder a cada píxel (x, y) de la imagen es necesario convertir las coordenadas (x, y) de la imagen en la coordenada (i) de un vector. Para ello se aplicará la siguiente fórmula:

$$i = y * Ancho + x.$$

3.1. Image

```
/**
@file Image.h
@brief Manejo de imÃ¡genes digitales en formato PGM blanco y negro
@author MP-DGIM - Grupo A
 **/

#ifndef _IMAGE_H_
#define _IMAGE_H_

#include <istream>
#include <fstream>
#include "Byte.h"

/**
@brief A black and white image
 */
class Image {
public:
    static const int IMAGE_MAX_SIZE=300000; ///< Max number of bytes allowd for
    static const int IMAGE_DISK_OK=0; ///< Image read/write successful
    static const int IMAGE_ERROR_OPEN=1; ///< Error opening the file
    static const int IMAGE_ERROR_DATA=2; ///< Missing data in the file
    static const int IMAGE_ERROR_FORMAT=3; ///< Unknown image format
    static const int IMAGE_TOO_LARGE=4; ///< The image is too large and does not fit into memory

    /**
     * @brief It builds an empty, image
     */
    Image();
    /**
     * @brief It builds a fully black image with @a width columns and @a height rows
     * @param height number of rows
     * @param width number of columns
     */
    Image(int width, int height);
    /**
    @brief It gives the number of rows of the image
    @return number of rows
     */
    int height() const;
    /**
    @brief It gives the number of columns of the image
    @return The number of rows
     */
    int width() const;
    /**
     * @brief It assigns the value @a v to the position(x,y) of the image.It must check that
     * the values x and y are valid, otherwise, it does not do anything.
     * @param x The column
     * @param y the row
     * @param v The new value
     */
    void setPixel(int x, int y, Byte v);
    /**
     * @brief It returns the value of the requested (x,y) position. It must check that
     * the values x and y are valid, otherwise, it returns a negative value. Please note that
     * the value returned is a int in order to allow negative values
     * @param x The column
     * @param y the row
     * @return The value of the pixel in [0-256] or -1 if there is an access error
     */
    int getPixel(int x, int y) const;
    /**
     * @brief It assigns the value @a v to the linear position i of the image. It must check that
     * the values i is valid, otherwise, it does not do anythig.
     * @param i The linear position
     * @param v The new value
     */
    void setPos(int i, Byte v);
    /**
     * @brief It returns the value of the requested linear position. It must check that
     * the value i is valid, otherwise, it returns a negative value. Please note that
     * the value returned is a int
     * @param i The linear position
     * @return The value of the pixel in [0-256] or -1 if there is an access error
     */
    int getPos(int i) const;
    /**
     * @brief  It sets all pixels of the image to the value given
     * @param b The value
     */
    void flatten(Byte b);

    /**
     * @brief It shows an image in an external window, ready for inspection. It uses
     * the program display (ImageMagick) to display every image. For an easier identification
     * process of all images shown are labeled with a title
     * @param title The title on top of the window
     */
    void showInWindow(std::string title) const;
    /**
     * @brief It calculates the hash value of the image and returns it inside a string,
     * together with its dimension.
     * @return a string that contains the dimension and the hash value of the image
     */
    std::string inspect() const;    /**
     * @brief It calculates the histogram of the image, and returns it into the array values such
```

```
98          * that values[i] = number of pixels whose tone is i
99          * @param values The array of pixel counts. The sum of all these values must be, exactly
100         * width()*height()
101         */
102        void getHistogram(int values[]) const;
103    private:
104        Byte _data[IMAGE_MAX_SIZE]; ///< Bytes of the image
105        int _height; ///< number of rows
106        int _width; ///< number of columms
107    };
108    #endif
```

4. Imaging1, práctica a entregar

Se deberá crear un proyecto Netbeans compuesto por los siguientes ficheros fuente incompletos.

- `Byte.h`
  Revisar y completar las declaraciones de los métodos, respetando el número y tipo de los parámetros, pero estableciendo una adecuada comunicación entre módulos, esto es: métodos const o no const, parámetros const o no const, parámetros por valor o por referencia etc. de acuerdo con las especificaciones dadas en la cabecera de cada método, para que pueda funcionar correctamente.

- `Byte.cpp`
  Completar la definición de todos los métodos de la clase, de acuerdo con las especificaciones dadas en el fichero de cabeceras.

- `Image.h`
  Revisar y completar las declaraciones de los métodos, respetando el número y tipo de los parámetros, pero estableciendo una adecuada comunicación entre módulos (paso por copia o por referencia).

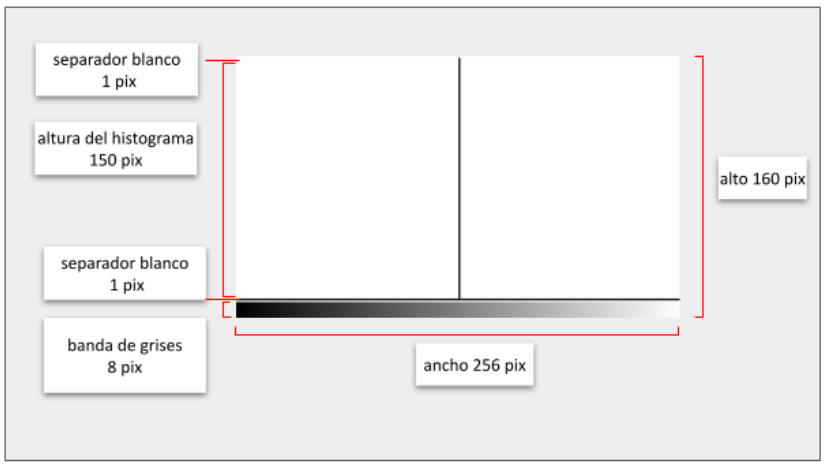
- `Image.cpp`
  Se deben implementar las funciones incluidas en el fichero `Image.h`.

- `main.cpp`
  Completar el código para realizar el programa que se describe en los comentarios.

  1. Construir una imagen de 256x256 totalmente plana (imagen de un único color) al valor 128.
  2. Calcular y reajustar su histograma.

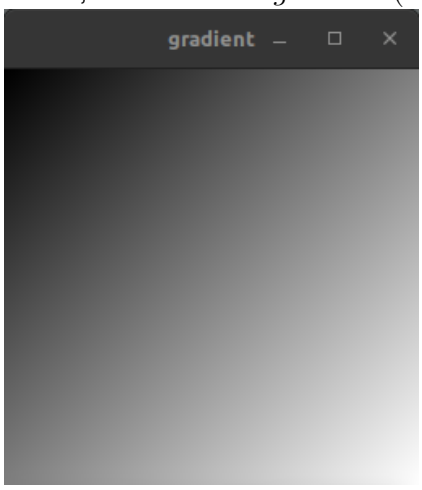
     El histograma de una imagen es un vector de 256 posiciones (una para cada uno de los tonos) de números enteros que representan el número de píxeles por cada tono, encontrados en la imagen. Así, la posición i contiene el número de píxeles de la imagen que tienen el valor i. El histograma de la imagen plana, contiene todo a valor 0 excepto la posición 128 cuyo valor se corresponde con el número de píxeles totales de la imagen.

Para visualizar un histograma, se hará uso de una imagen auxiliar como se muestran a continuación y que se deberá construir a partir del vector de conteo, del histograma. Indicar aquí que, las dimensiones de la imagen del histograma serán siempre fijas independientemente de las dimensiones de la imagen sobre la que se calcule. Dado que la altura disponible para el histograma es de 150 píxeles (como s ilustra en la imagen) el valor de las frecuencias no puede superar este valor, por lo que se debe ponderar mediante una sencilla proporción lineal.



3. Visualizar las dos imágenes, la plana y su histograma.
4. A continuación nueva imagen, histograma y visualización. En detalle, debe crear una imagen 256x256 con un relleno degradado en diagonal tal y como muestra la figura siguiente, de forma que la posición (0,0) tenga el valor 0 y la posición (255,255)

tenga el valor 255. El resto se distribuyen como una proporción lineal, de la forma: $gradient(x, y) = ((x + y) * 255)/(255 + 255)$



El histograma de esta imagen degradada es muy característico

5. Para comprobar de forma analítica que las imágenes se han generado correctamente, se debe generar su hash, un código biunívoco que las identifica en base a su contenido, haciendo uso de la función `inspect`. Así, además de visualmente, podrá cotejar sus valores obtenidos con las salidas esperadas.

Tanto la utilidad para lanzar el visualidor de imágenes `showInWindow()` como el hashing `inspect()` son funciones ya programadas que se encuentran en la librería MTools.

4.1. Ejemplo de ejecución

Se muesta la salida, la cual contiene únicamente los código hash de las imágenes generadas y sus correspondientes histogramas

```
[flat] 256x256 1468792869

[histogram] 256x160 1519143717

[gradient] 256x256 2590034725

[histogram] 256x160 1617958612
```

5. TESTS DOCUMENTATION FOR PROJECT Imaging1

5.1. _01_Basics

5.1.1. UnitByte_Constructor

1. Declaring a Byte gives 0 by default
2. Declaring a Byte(1) gives 1
3. Declaring a Byte(128) gives 128

5.1.2. UnitByte_getValue

1. Declaring a Byte gives 0 by default
2. Declaring a Byte(1) gives 1
3. Declaring a Byte(128) gives 128

5.1.3. UnitByte_setValue

1. Declaring a Byte and setting its value to 0 gives 0 by default
2. Declaring a Byte and setting its value to 1 gives 1
3. Declaring a Byte and setting its value to 128 gives 128

5.1.4. UnitByte_onBit

1. Given a byte 00000000, activating the 0-bit gives 1
2. Given a byte 00000000, activating the 1-bit gives 2
3. Given a byte 00000000, activating the 7-bit gives 128

5.1.5. UnitByte_offBit

1. Given a byte 11111111, deactivating the 0-bit gives 254
2. Given a byte 11111111, deactivating the 1-bit gives 253
3. Given a byte 11111111, deactivating the 7-bit gives 127

5.1.6. UnitByte_getBit

1. Given a byte 11111111, querying any bit always give true
2. Given a byte 00000000, querying any bit gives false

5.1.7. UnitByte_to_string

1. A byte 11111111 prints as it is
2. A byte 00000000 prints as it is

5.1.8. UnitByte_shiftRByte

1. A byte 11111111 shifted to the right gives 127
2. A byte 11111111 shifted twice to the right gives 63
3. A byte 00000001 shifted to the right gives 0

5.1.9. UnitByte_shiftLByte

1. A byte 11111111 shifted to the left gives 254
2. A byte 11111111 shifted twice to the right gives 252
3. A byte 00000001 shifted to the right gives 2

5.1.10. Image_Constructor

1. and empty data
2. and empty data
3. and empty data

5.1.11. Image_Width

1. gives width
2. gives width
3. gives width

5.1.12. Image_Height

1. gives height
2. gives height
3. gives height

5.1.13. Image_setPixel

1. but should have been
2. but should have been

5.1.14. Image_getPixel

1. but should have been
2. but should have been

5.1.15. Image_getPos

1. but should have been
2. but should have been

5.2. _02_Intermediate

5.2.1. UnitByte_onByte

1. Activating a Byte gives 255

5.2.2. UnitByte_offByte

1. Deactivating a Byte gives 0

5.2.3. Image_flatten

1. is wrong
2. is wrong

5.2.4. Image_getHistogram

1. but it should have been
2. but it should have been

5.2.5. INTEGRATION_Image

1. The output of the main program must match that of the assignment

5.3. _03_Advanced

5.3.1. UnitByte_encodeByte

1. Activating bits 0,1 and 7 gives 131

5.3.2. UnitByte_decodeByte

1. A byte 131 gives true only in bits 0,1 and 7
2. A byte 131 gives true only in bits 0,1 and 7
3. A byte 131 gives true only in bits 0,1 and 7
4. A byte 131 gives true only in bits 0,1 and 7
5. A byte 131 gives true only in bits 0,1 and 7

5.3.3. UnitByte_decomposeByte

1. Decomposing byte 131 gives 3 active bits
2. Decomposing byte 131 gives 3 active bits
3. Decomposing byte 131 gives 3 active bits
4. Decomposing byte 131 gives 3 active bits

5.4. Tests run

```
[==========] Running 23 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 15 tests from _01_Basics
[ RUN      ] _01_Basics.UnitByte_Constructor
[       OK ] _01_Basics.UnitByte_Constructor (0 ms)
[ RUN      ] _01_Basics.UnitByte_getValue
[       OK ] _01_Basics.UnitByte_getValue (1 ms)
[ RUN      ] _01_Basics.UnitByte_setValue
[       OK ] _01_Basics.UnitByte_setValue (1 ms)
[ RUN      ] _01_Basics.UnitByte_onBit
[       OK ] _01_Basics.UnitByte_onBit (1 ms)
[ RUN      ] _01_Basics.UnitByte_offBit
[       OK ] _01_Basics.UnitByte_offBit (0 ms)
[ RUN      ] _01_Basics.UnitByte_getBit
[       OK ] _01_Basics.UnitByte_getBit (4 ms)
[ RUN      ] _01_Basics.UnitByte_to_string
[       OK ] _01_Basics.UnitByte_to_string (1 ms)
[ RUN      ] _01_Basics.UnitByte_shiftRByte
[       OK ] _01_Basics.UnitByte_shiftRByte (1 ms)
[ RUN      ] _01_Basics.UnitByte_shiftLByte
[       OK ] _01_Basics.UnitByte_shiftLByte (1 ms)
[ RUN      ] _01_Basics.Image_Constructor
[       OK ] _01_Basics.Image_Constructor (5 ms)
[ RUN      ] _01_Basics.Image_Width
[       OK ] _01_Basics.Image_Width (5 ms)
[ RUN      ] _01_Basics.Image_Height
[       OK ] _01_Basics.Image_Height (5 ms)
[ RUN      ] _01_Basics.Image_setPixel
[       OK ] _01_Basics.Image_setPixel (2 ms)
[ RUN      ] _01_Basics.Image_getPixel
[       OK ] _01_Basics.Image_getPixel (2 ms)
[ RUN      ] _01_Basics.Image_getPos
[       OK ] _01_Basics.Image_getPos (2 ms)
[----------] 15 tests from _01_Basics (31 ms total)

[----------] 5 tests from _02_Intermediate
[ RUN      ] _02_Intermediate.UnitByte_onByte
[       OK ] _02_Intermediate.UnitByte_onByte (1 ms)
[ RUN      ] _02_Intermediate.UnitByte_offByte
[       OK ] _02_Intermediate.UnitByte_offByte (0 ms)
[ RUN      ] _02_Intermediate.Image_flatten
[       OK ] _02_Intermediate.Image_flatten (2 ms)
[ RUN      ] _02_Intermediate.Image_getHistogram
[       OK ] _02_Intermediate.Image_getHistogram (1 ms)
[ RUN      ] _02_Intermediate.INTEGRATION_Image
[       OK ] _02_Intermediate.INTEGRATION_Image (33 ms)
[----------] 5 tests from _02_Intermediate (37 ms total)

[----------] 3 tests from _03_Advanced
[ RUN      ] _03_Advanced.UnitByte_encodeByte
[       OK ] _03_Advanced.UnitByte_encodeByte (0 ms)
[ RUN      ] _03_Advanced.UnitByte_decodeByte
[       OK ] _03_Advanced.UnitByte_decodeByte (1 ms)
[ RUN      ] _03_Advanced.UnitByte_decomposeByte
[       OK ] _03_Advanced.UnitByte_decomposeByte (1 ms)
[----------] 3 tests from _03_Advanced (2 ms total)

[----------] Global test environment tear-down
[==========] 23 tests from 3 test suites ran. (70 ms total)
[  PASSED  ] 23 tests.
```

5.5. Configuración de la práctica

Para esta práctica se puede seguir con la misma configuración de la práctica anterior y hacer una copia del proyecto en uno nuevo con las siguientes novedades:

- Se deben eliminar los antiguos tests de la carpeta **tests** antes de introducir los nuevos tests correspondientes a las clases `Byte` y `Image`, así como los nuevos tests de integración.
- Se eliminarán los antiguos `Byte*` y se incluirán los nuevos ficheros correspondientes a la clase `Byte` en las carpetas **include** y **src**.
- Se incluirán los ficheros correspondientes a la clase `Image` a las carpetas **include** y **src**.
- Comprobar en **Project Properties - C++ Compiler - Include Directories** que, están los directorios **include** siguientes:
  - del propio proyecto **./include**
  - el de la librería MPTools **../MPTools/include**
  - las de testeo **../MPTools/googletest-master/googletest/include**
- Se modificará el fichero **main.cpp** de acuerdo a las instrucciones en la documentación del fichero de la práctica.
- Recordar añadir desde la vista lógica del proyecto.
  1. En `Header Files`, `Add existing item` añadir el fichero `Image.h`.
  2. En `Source Files`, `Add existing item` añadir el fichero `Image.cpp` y el nuevo fichero `main.cpp`.
  3. En `Test Files`, `Add existing item` añadir los nuevos ficheros de los tests.

La práctica deberá ser entregada en Prado, en la fecha que se indica en cada entrega, y consistirá en un fichero ZIP del proyecto en su estado actual (es decir con todas las clases implementadas).

5.6. Entrega de la práctica

Una vez terminada la práctica que, al menos, haya superado los tests básicos, se debe hacer un zip (se sugiere utilizar la script `runZipProject.sh`) excluyendo las carpetas `./dist/`, `./build/`, `./nbproject/private/`, `./doc/html/` y `./dos/latex/` y subirla a Prado antes de la fecha de cierre de la entrega.