INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Oriented-Object Programming

Labs
S7 EII

2017-2018

Summary

INSA

LAB 1

Playing Cards

Objectives : (*2h*)
- Create your first classes,
- Use inline functions,
- Use keywords to specify the class API (*explicit*, *const*, *static*,...)
- Use the STL library (vector),
- Study your first UML diagrams
- Take care of the coding policy,
- First study case of a copy constructor,
- Implement operator<<

1.1 Presentation

The purpose of this practical lesson is to provide classes to play cards.

You will have to code 2 classes : Card and Game. You will study a basic case of classes, with basic oriented-object programming tools. Take times to fully understand these latter.
The additionnal coding policy you will have to follow from now : add a symbol_ at the end of your class member name (for example, foo_).

1.2 Card class

A card is defined by
- one of the four suits : clubs, diamonds, hearts or spades
- its symbols : ace, king, queen, jack, ten, nine, eight or seven

Propose a Card class with at least these functionalities :
- constructors (which one is useful ?). Avoid implicit conversion by using the keyword *explicit*.
- access to the class members. Inline them as they are very simple and often used for a Card.
- function *display* to display the card (provided below)
- functions *equalsSuit*, *equalsSymbol* to compare the card with another one

Use enum type to define the cards symbols and the suits.

Which class members are supposed to be public ? Protected ? Private ? Why ?

Implement every function. You will find the code for function *display* hereafter (adapt it following your enum types and class member names) :

```
namespace {
const std::string suits_names[]={ "None", "Clubs", "Diamonds", "Hearts",
    "Spades" };
const std::string symbols_names[]={ "None", "Ace", "King", "Queen", "Jack",
    "Ten", "Nine", "Eight", "Seven"  };
}

void Card::display() const { std::cout<<suits_names[suit_]<<'
    '<<symbols_names[symbol_]; }
```

1.2.A Check Card API

At the end, check your Card API. It should match the UML diagram in Figure 1.1 (it has been designed with Visual Studio 2013 architecture tools). You will find in Annex A a reminder about UML symbols. If something is different or missing, ask and fix your code.

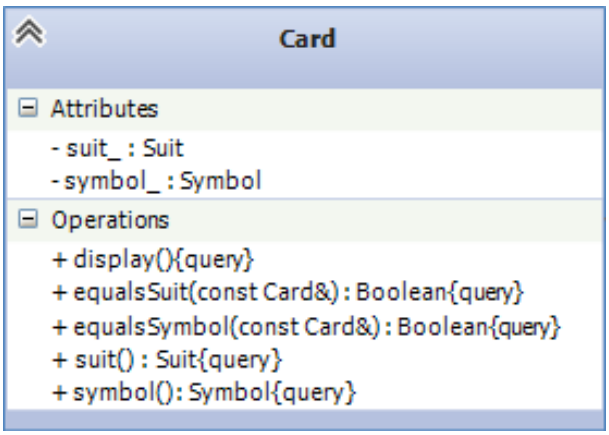

FIGURE 1.1 – UML diagram for the Card class

1.3 Game

A game is made of :
— a draw pile where the player can take some cards
— a discard pile with the extracted cards

The two piles are collectors of Cards. The question is to know whether an array, a list,... is the best choice. For example, in the C language style, you can use a $Card*$ member and manage the memory allocation of the dynamic array. However in OOP, C-style containers should be avoided to take benefit of the work of programmers that are far better than you[1] : the best solution here is to use the STL tools. The $std :: vector$ is the the most suitable container for this exercise. You will find the $std :: vector$ template class API[2] at `http://www.cplusplus.com/reference/vector/vector/`. You can note that complexity information is given, which is the case for every STL container. It helps you to find which is the most convenient container for a given problem.

1. and me
2. Application Programming Interface

Sometimes, to know the most convenient container in a complex project, you just need to try. In order to do it simply without recoding everything, you can take benefit of the STL library which ensures that its containers implement the same API. Therefore, you can define the type you will use and the rest of the code will be the same whatever the container is (vector, list, stack, queue,...). For example, for a collection of cards :

```
typedef std::vector<Card> Pile;   //change only this  if you want a list,...
Pile draw_pile_;
```

Which class members are supposed to be public ? Protected ? Private ? Why ?

Propose a Game class with at least these functionalities :

— constructors (which one is useful ?)
— access to the class members
— function $display$ to display the game
— function $shuffle$ that changes the order of the cards in the draw pile. See paragraph 1.3.A for more information.
— function $extractCard(int)$ that moves the i^{th} card of the draw pile and put it in the discard pile.
— function $putInDrawPile$ that puts every card in the discard pile in the draw pile

1.3.A Randomness

To shuffle cards, a random generator is required. In C, you would need to handle it by yourself :

— *srand(time(NULL)* has to be called once and only once to initialize the seed
— N calls to *rand()* to get a random number to swap cards

If you use STL library, guess what, you are not the first one that need a random shuffle, so it exists : *random_shuffle*. Find its documentation on the internet and use it to shuffle the draw pile in the *Game : :shuffle()* function.

1.3.B Check Game API

At the end, check your Game API. It should match the UML diagram in Figure 1.2 (it has been designed with Visual Studio 2013 architecture tools). If something is different or missing, ask and fix your code.

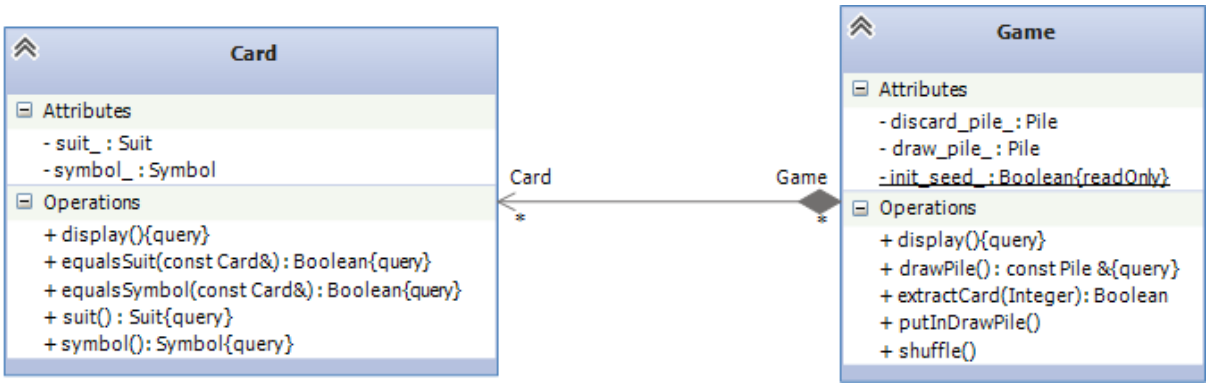

FIGURE 1.2 – UML diagram for the Game class

1.4 Assignment operator

Each time a card is moved in a pile, a copy is performed using the default assignment operator which copies the value of every member. To visualize these calls, add in the Card class the implementation of the assignment operator (operator=) with a display on the standard output to trace the function call.

Call the $shuffle$ function. What can you tell ?
Look at the documentation of *memcpy* function. When can it be useful ?

1.5 Operator$<<$

Add in the Card and Game classes the implementation of the operator$<<$ in order to be able to display a card this way :

```
Card c(Card::heart,Card::queen);
cout<<c<<endl;
```

What is the big difference between operator= and operator$<<$?

INSA

LAB 2

MyString class

Objectives : (*2h*)
- Memory management
- Implement a copy constructor and apply rule of 3
- Use pointers and references
- Implement operators using friend functions

2.1 Presentation

The practical lesson aims to provide a class that is similar to the string class of ISO C++. It will manage a dynamic char array and propose some basic functionalities. It is proposed for educational purpose however, in a real project, you would better use the standard classes that is optimized and safe [1].

2.2 MyString class

Propose a class to will manage such a char dynamic array. You can add others class members as long as your implementation complies with your choice.

For educationnal purpose, you will use the standard C functions on char arrays (strlen, strcpy,...) in your C++ functions for this practical lesson. However, as they are useful but dangerous (no memory management) and as the standard string class provides a more flexible and secure use, remind you should not use them in future C++ code.

2.2.A Constructors, destructors and copy assignment operator : rule of 3

There is a rule in C++98 called the *rule of 3* [2] that says that **IF a class defines its own copy constructor or copy assignment operator or destructor, it should probably explicitly define all three**.
Keep this in mind for future C++ code and apply this rule now (cf next page).

1. Remember : the guys that have written the STL library are better than you and me
2. This rule becomes the *rule of 5* for C++11

Implement at least these three constructors (and therefore destructor and copy assignment operator) :

```
MyString();
MyString(const char * s);
MyString(const MyString &c);

~MyString();

MyString &operator=(const MyString & s);
```

In this study case, explicit conversion is not wanted. Convert an array of char to a String implicitly is a usual feature.

Display on the standard output a trace. You will thus track the calls of each constructors/methods when running the program : it is very instructive. In order to enable/disable the trace, you can use such a macro :

```
#if 1
#define LOGME(x) std::cout<<x<<std::endl;
#else
#define LOGME(x)
#endif
```

Implement then the operator$<<$ to display the instance content on the standard output.

Test your code. More specifically, here are some special cases you should pay attention to :

Test of operator$=$

Test the following code :

```
MyString ch1("abc");
ch1 = ch1;
```

If it fails, find why and change your code to support self-assignment.

Constructors and assignment operator

What is the difference between the three following MyString initialization ?

```
MyString ch1("abc");
MyString ch2 = "abc";
MyString ch3;
ch3 = "abc";
```

You can watch the constructors, destructor and assignment operator calls to help you to understand what is going on.

2.2.B Operator[]

Implement the operator[] that enables to access to the i^{th}. Example of use :

```
MyString ch1("bbb");
MyString ch2 = "abc";
ch1[0] = ch2[0];
cout<<"ch1 = "<<ch1<<endl;
```

Test your code. If the output is not "abb" for the provided example, then fix your code.

Then add the *const* version of the operator[]. Why is it necessary to implement both of them ?

2.2.C Binary operators

For binary operators (comparison, concatenation of two strings), you have two choices :

— declare it as a method of the class with one parameter. For example :

```
class MyString
{
...
  bool operator==(const MyString &s) const;
};
```

— declare it as a friend method with two parameters. For example :

```
class MyString
{
...
  friend bool operator==(const MyString & s1, const MyString &s2);
};
MyString operator+(const MyString &s1,const MyString &s2);
```

Which one allows implicit conversion on both operands ? Which one is therefore the best solution ?

Implement these additional operators :

— comparison of two strings (operator$==$, $<$, $>$, $>=$, $<=$). It has to work as *strlen* for the return value.

— concatenation of two strings (operator$+$).

2.2.D Methods

Implement the $toUpper$ function which modifies the current instance by changing every char to the upper case one.
After your test, test the following code :

```
MyString ch1;
MyString ch2 = "abc";
(ch1 = ch2).toUpper();
cout<<"ch1 = "<<ch1<<endl;
```

The output is expected to be "ABC". If not, fix your code.

2.2.E Exception

When trying to access to a char out of the array in operator[](int), the error case can not be properly handled by the return value (what char is an error code ?). It is then better to throw an exception.

Modify this operator to throw a out_of_range exception and try to catch this exception in your main file.

2.2.F Some more methods

Implement the following methods that look for a substring in a string :

```
MyString subString(char begin, char end) const;
//begin is the first char of the wanted substring and end the last one

MyString subString(int begin, int s) const;
//to extract a substring of size s and beginning at index begin
```

2.3 ISO C++ string class

Change your main function to use the string class proposed in the standard C++ library instead of your MyString class.

INSA

LAB 3

Graphic objects

Objectives : (*2h*)
- Implement inheritance from an UML diagram
- Study polymorphism

3.1 Presentation

The practical lesson aims to provide a set of graphic objects that can be drawn, moved, erased. $Gdi+$ library is used for the drawing part. As a consequence, the new project to be created in Visual Studio has to be a "Win32" empty project. Once created, you will add the provided files and modify the project properties by adding "GdiPlus.lib" in the linker input.

3.2 Class diagram

The graphic objects diagram to implement are presented in Figure(3.1).
A graphic object is described by at least a colored point ($x\_$ and $y\_$ coordinates) that can move. Specific shapes may require additional attributes.

A list of all the created graphic objects is managed by the $first\_$ and $next\_$ pointers.

3.2.A Class members

Which class members are public($+$) ? Protected($\#$) ? Private($-$) ? Static (underlined) or not ? Why ?

3.2.B Constructor and destructor

Why is the MyGraphicObject constructor protected ?
Why is it required to redefine a destructor in the children classes ?

3.2.C Methods

Which methods are abstract ? Virtual ? Why ?

3.3 Implementation

Add the provided files in your project :

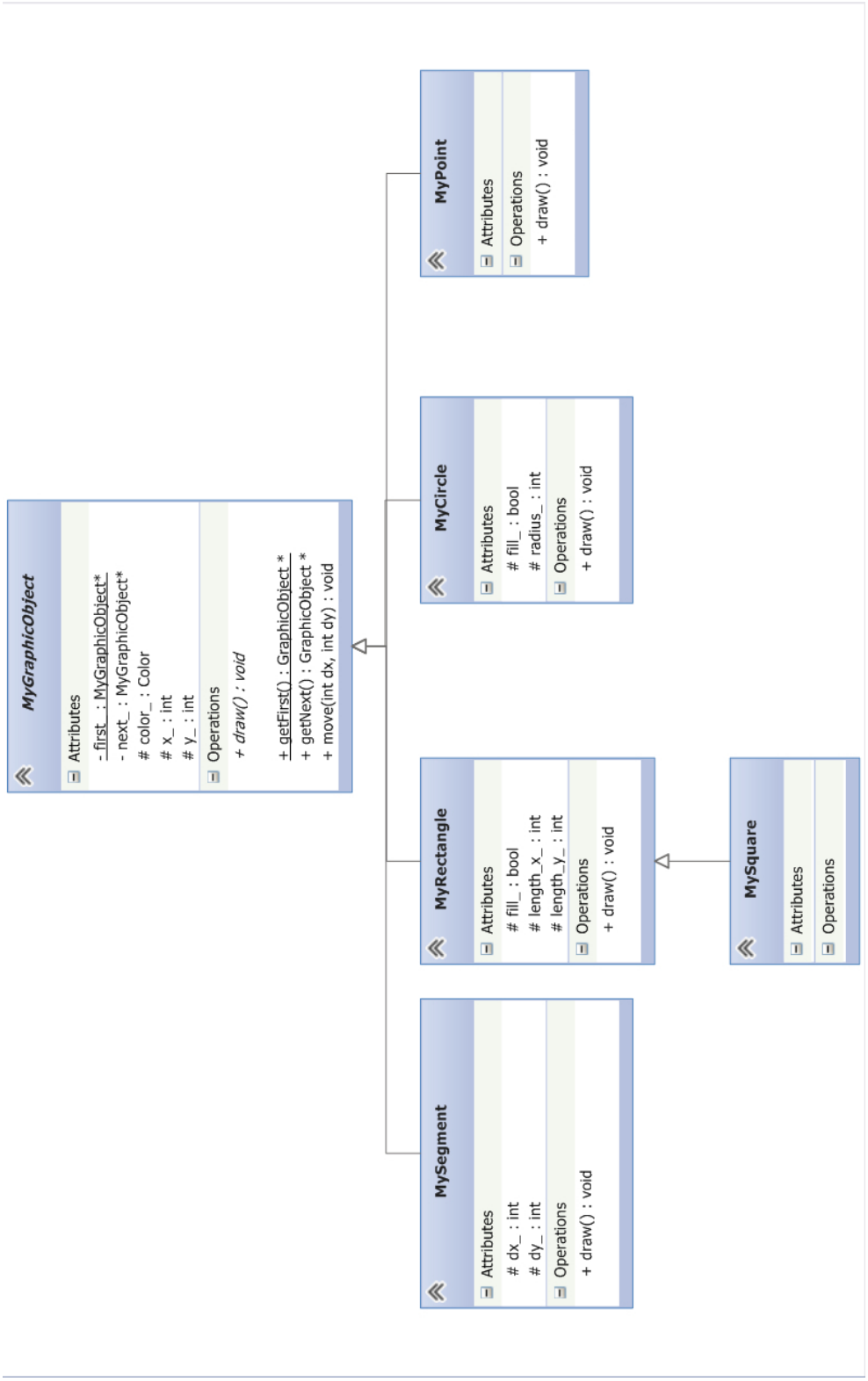

FIGURE 3.1 – Graphic Objects class diagram

— "MyGDIPlus.hpp" and "main_graphics_gdi.cpp" deal with the GDI+ library. You do not have to modify those files but you will have to use the provided functions for the drawing part.
— "game.cpp" provides the display test. You have to uncomment lines corresponding to the class you have implemented.

Implement in the following order the classes : MyGraphicObject, MyPoint, MySegment, MyRectangle, MySquare, MyCircle. Implicit conversion is not wanted here : avoid it. Test a class you coded before the next one. Once you have added the MyGraphicObject class, you can uncomment the functions *step*, *draw* and *desinit* in the "game.cpp" file.

In the loop of the game functions, only the MyGraphicObject type is used but various shapes are moving. Why does it work ?

3.4 Discussion

The management of the list is not secure : allocation and deallocation of the list elements are not managed alltogether. A MyGameElement can be allocated/deallocated anywhere, so the user has to take care with its memory management in a C-like style, which is bad.
A good solution is to provide a MyGameElement manager (create a list : constructor, delete the list : destructor, add an element or get its value : member functions,...). This solution will be provided in the next practical lesson.
Generally speaking, to avoid memory leaks, you should :
— respect the rule : the owner should deal with the memory management
— study smart pointers (available in C++11)

INSA

LAB 4

"Game"

Objectives :	(*4h*)
— Propose a solution using inheritance	
— Study polymorphism	

4.1 Presentation

The practical lesson aims to build a game made of a frame in which a rocket and a missile are moving. When the missile hits the rocket, this latter explodes. As it relies on a set of graphic objects that can be drawn and moved, you will need $Gdi+$ library for the drawing part as for the previous practical lesson. As a consequence, the new project to be created in Visual Studio has to be a "Win32" empty project. Once created, you will add $GdiPlus.lib$ in the linker inputs of your project.

4.1.A Required files

You will need the files provided on the Moodle site and the classes you have coded at the last session (MyGraphicObject and so on). Remove **before the session** the attributes linked to the list management ($first\_$ and $next\_$) as well as the code related to them since the list is no more useful. Instead, a MyGraphicalObjectManager is provided, which a safer solution to handle the list.

4.2 Game

A game is made of a frame into which a rocket and a missile evolve. The $MyGame$ class is provided. Study the code to understand how the game evolves.

In order to maintain the game easily, every component of the game should derived from a base class, $MyGameElement$. The $MyGameElement.hpp$ is provided. As you can see in it, a $MyGameElement$ is defined by its bounding box, its color and the figures required to draw it. The constructors in this practical lesson are strongly typed to make the code more readable. The *explicit* keyword is used to prevent from implicit conversion which could lead to unwanted behavior.
These components are divided into two groups, the mobile ones and the immobile ones. To detect when they cross, you can check if their bounding box intersect.

As a result, you have to implement :

1. the $MyMobile$ and $MyImmobile$ classes derived from the $MyGameElement$ class,
2. then the $MyFrame$, $MyRocket$ and $MyMissile$ classes derived from one of these two classes. Each time you add one of those classes, uncomment the related code (and adapt it if necessary) in each function of the "game.cpp" file.

You have to think about which methods are virtual or not, which classes are instantiable or not, etc. Hint and tips : if you have to override the *draw* function in the MyGameElement family, you are wrong.

For the drawing part :

— you have to use the graphic objects implemented in the previous practical lesson,
— the missile should be represented by a filled circle,
— the rocket by two rectangles, one filled and one not
— when the rocket explodes, it should be represented by three concentric circles (yellow for the smallest one, orange and red for the biggest one).

A few questions :

— Why $MyGameElement$ class is not instantiable ? Are there some other classes which are not instantiable ? Why ? What is the consequence on the constructors ?
— Why the $draw()$ method of $MyGameElement$ class is not virtual ?

Up to you now.

INSA

LAB 5

Template image

Objectives : (*2h*)
— code a template class in C++
— use libjpeg library
— use exceptions
— use STL vectors

5.1 Presentation

The practical lesson aims to implement a class MyImage. This class has 3 members :
— $width\_$ the number of columns
— $height\_$ the number of rows
— $bitmap\_$ a one-dimensional pixel vector of size $width\_ \times height\_$
This class should handle grey level images, color images,... As a consequence, the pixel type is not pre-defined. Since every type of image should handle the same functionalities (open an image in a file, write an image, image modifications, filtering,...), the generic code can be obtained using a template class. Frames are provided but uncomment code only when needed, compilation will fail otherwise.
The project is a Win32 console application.

5.2 MyImage class

Copy every provided files in the project directory. Add only "MyImage.hpp" file in your project.

Implement the MyImage class. Provide at least a constructor setting the size of the image. Since STL vector is used, no need for a copy constructor, a destructor and the copy assignment operator (rule of 3).
Code the operators required in the API of MyImage class (except the $read$ and $write$ functions for the moment).

To test your class, use first grey level images (the type of a pixel is therefore an *unsigned char*).

For a nicer use of this class, it would be interessant to open existing images ("test-bw.jpg" is provided) and write the results in files. These IO functionalities are provided in the MyImageIO namespace. You do not have to study the code of MyImageIO for this lab, it just provides useful IO functionalities. However it required the jpeg library and the next section will enable you to use these IO functionalities.

5.3 MyImageIO class

For the image reading/writing, the MyImageIO namespace is provided. To read/write a JPEG image, the *libjpeg* library is required.

5.3.A libjpeg setup

Here are the steps to install and use the *libjpeg* library for Visual Studio :

1. Download the *jpegsr9.zip* on the Moodle site and unzip it. Comment the line *!include <win32.mak>* in the *makefile.vc* file [1].
2. Launch Visual Studio Command Prompt (in Visual Studio Tools). Go to the uncompressed *jpeg-9* directory and generate Visual project files by typing : nmake -f makefile.vc setup-v10
3. Open *jpeg.sln* and build the library (click "Yes" for project upgrade if necessary)
4. In your project, modify the project properties to add the *libjpeg* headers directory, library and library path :
   (a) *jpeg-9* added in "Configuration Properties > C/C++ > General > Additional Include Directories"
   (b) *jpeg.lib* in "Configuration Properties > Linker > Input > Additional Dependencies"
   (c) *jpeg-9/Release* in "Configuration Properties > Linker > General > Additional Library Directories"

Now you will be able to compile MyImageIO namespace. Do not forget to uncomment code to read/write grey level images in *MyImageIO.cpp* file.

5.3.B Back to MyImage class

Add the MyImageIO and myRGBa files in your project.
To be able to open an image, you can now code in the MyImage class the following functions using the MyImageIO functionalities [2] :

— $read$ function that opens an image stored in a jpeg file
— $write$ function that writes the image in a jpeg file

5.4 RGB images

The MyImage template should be able to work with other types of pixel. What functionalities should be provided to use another type of pixel ?

We will now study a color image. The color will be represented by the RGBa representation (Red, Green, Blue, Alpha). A MyRGBa class is provided. Does the class match the requirements to be used with MyImage template ? If not, complete the class. Then uncomment code in MyImageIO class to make it work.

Test the use of a color image. "test-color.jpg" is provided if needed.

1. Do not comment this line for older versions than Visual Studio 2012.
2. If you want some information about the greylevel - color conversion, you can read : `http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html` for example

5.4.A Color image test

Follow this process :

1. In MyImage template, add the $operator - (const\ Pixel\ \&\ val)$ that withdraws the value *val* to every pixel of the image ;
2. Test it on the "mask.jpg" image by withdrawing the RGBa value (100,100,100,0) to this image ;
3. Modify the modified image by setting the background to red. The background are the pixel values greater than 128. Set the others pixels to blue (use the iterator functionalities provided in the MyImage API) ;
4. Save the result in a file and look at it.

You should quickly understand if you have succeed.

5.5 IO exceptions

What happen when you try to open an image file that does not exist ? Modify the provided functions to throw an exception when the file can not be opened properly(*ios_base : :failure* exception) [3]

What happen when you try to open a bitmap file (.bmp) ? Modify the provided functions to throw an exception when the file format is not handled (*invalid_argument* exception).

Modify your main program to catch these exceptions.

3. You can find the standard exception hierarchy at `http://en.cppreference.com/w/cpp/error/exception` or at `http://www.cplusplus.com/reference/exception/exception` for example

ANNEXE A

UML

The Unified Modeling Language (UML) is a general-purpose modeling language used in software engineering. It provides semantics to describe the structure and the behaviour of a system, through :

— component diagram
— class diagram
— activity diagram
— use case diagram
— sequence diagram
— communication diagram
— ...

As it provides a standard way to visualize the design of a system, it is not related to a specific language. Therefore its keywords and symbols are different from those of C++. As we deal only with class diagrams, here is a reminder on the main notations used in class diagrams [1].

A class is represented by a rectangle divided in 3 or 4 parts :

— the first one is for the name
— the second one is for the member variables : attributes
— the third one is for the member functions : methods
— the forth one is not mandatory, it is a description of the responsibilities of the class.

The visibility (*public*, *protected* or *private*) of attributes and methods are represented by a character before the name :

1. public : +
2. protected : #
3. private : -

A *static* attribute or method is underlined.

A method can have special properties indicated in between {}. For example, a method that does not change the state of the class will have {query} after its prototype.
A pure virtual method is in italic or marked as {abstract}.

The Figure A.1 sums up all these inner class notations.

1. See books or on-line documentation for a complete description for each diagram.

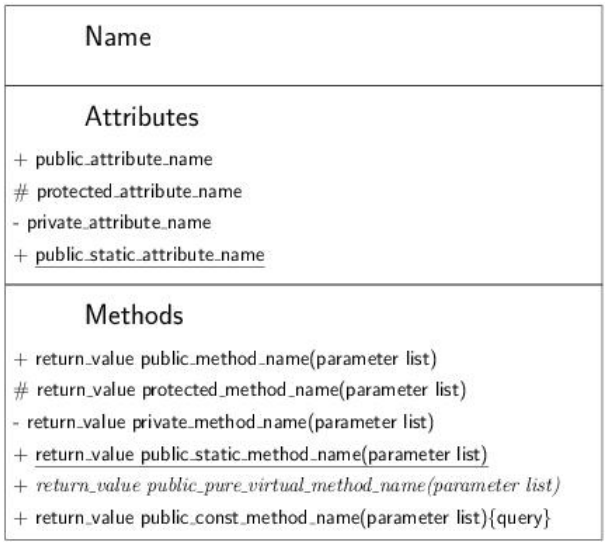

FIGURE A.1 – Basic class diagram notations

For class diagrams, UML provides links between classes to describe their relationship. The basic ones (inheritance, aggregation and composition) are shown in Figure A.2.

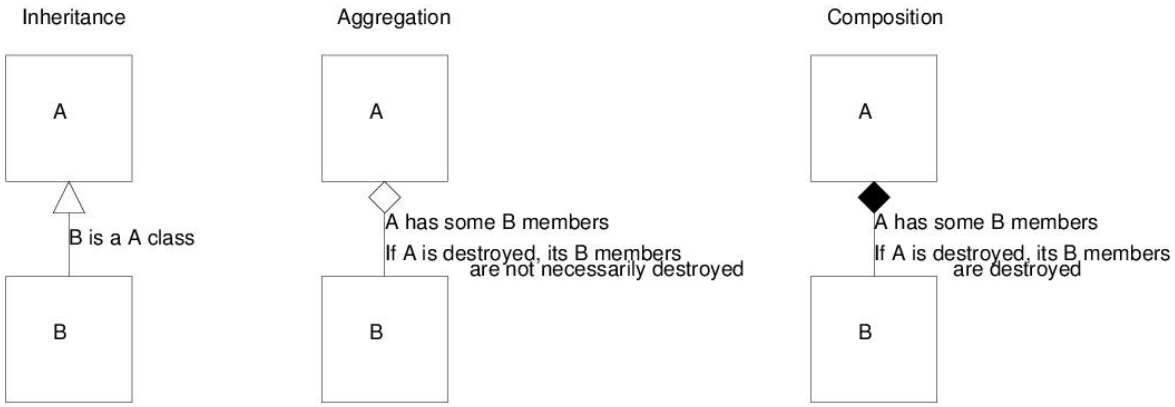

FIGURE A.2 – Basic class relationship

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE