Universidad
Carlos III de Madrid

Industrial Informatics I Lab Session

Bachelor in Industrial Electronics and Automation

2016-2017

Lab 4

Teaching staff:

Juan Carlos Gonzalez Victores

Mohamed Abderrahim

Lab 4 – Inheritance

Inheritance is a feature of the OOP language that reuses and extends the code of existing class.

1. Inherit to extend

In C++ you can declare a class (called child class) that is derived from another class (called parent class). This implies that the child class retrieves all members of the parent class and can be extended by adding new members to it.

The following example shows how we can define a point class along with color (The color value is represented by an integer) by deriving it from the `Point` class, to reuse the code related to point coordinates that we have already implemented in the `Point` class.

Insert the following code into the `.h` file of `Point` class:

```
class Point {
      public:
            Point(int x, int y);

            void set(int x, int y);
            int getX() const {return _x;}
            int getY() const {return _y;}

            void display () const;

      private:
            int _x, _y;
}
```

And the following in the `.h` file of the new `ColorPoint` class:

```
class ColorPoint : public Point {
      public:
            ColorPoint(int x, int y, int c) : Point(x, y) {_color = c;}
            int getColor() const {return _color;}

      private:
            int _color;
}
```

Then try the following code in main:

```
int main() {
      ColorPoint CP(10, 20, 255);

      CP.getX();        // come from Point class
      CP.display();     // come from Point class, display x and y
      CP.getColor();    // come from PointColor class
}
```

Thanks to inheritance, the `ColorPoint` class only needs to define its additional member function (`getColor`), as it inherits the remaining members functions, relating to point coordinate, from the class `Point`. Note how the constructor of `ColorPoint` calls the constructor of the `Point` class, which accepts two parameters.

2. Redefining a member function

The above solution is still not fully satisfactory since the function `display`, inherited from the `Point` class does not show the color of the point. Therefore we can redefine the member function `display` in the child class, which would change the behaviour of this function.

Therefore, the class `Point` should be changed to the following:

```
class Point {
      public:
            Point(int x, int y);
            // set, getX, getY
            void display() const {cout << _x << _y;}

      private:
            int _x, _y;
};
```

And the class `ColorPoint` should be changed to:

```
class ColorPoint : public Point {
      public:
            ColorPoint(int x, int y, int c) : Point(x, y) {_color = c;}
            int getColor() const {return _color;}
            void display() const {cout << _x << _y << _color;}

      private:
            int _color;
};
```

Thus, you can change the behaviour of a child class by redefining its inherited method(s).

So the main function should be as follows:

```
int main() {
      ColorPoint CP(10, 20, 255);
      CP.display();     // comes from PointColor class, display x, y and color

      Point P(10, 20);
      P.display();      // comes from Point class, display x and y
}
```

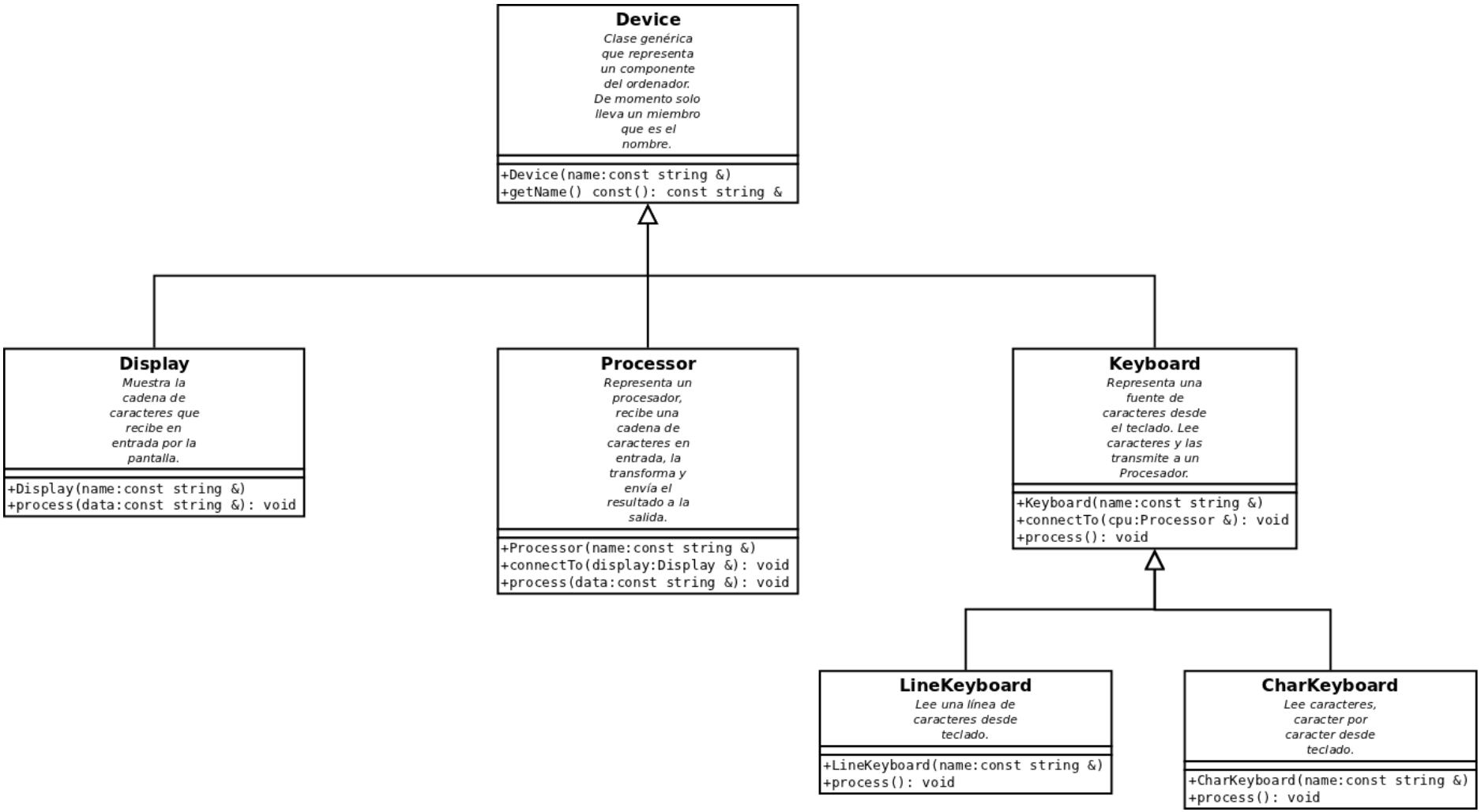
Note: The code in `ColorPoint` class would not compile. Find the error and propose a solution.

Exercise

- Implement a computer simulator:
    - To illustrate the concepts of object-oriented programming in C++, we will work on a small project, cumulating throughout the rest of the lab sessions. The objective of this project is to develop a computer simulator. The program should simulate keyboards, processors, software, monitor screen, etc. To keep the project simple, your computer simulator will only be able to handle strings. In this session, start by implementing the code that corresponds to the UML diagram provided in the Annex.
    - Specifically, the program should be able to read characters from the keyboard, transmit the same to the processor to be processed, and then send the processed string to be displayed on the monitor screen. The design should be modular, i.e. it should be relatively simple to add or modify elements of a class.
    - For each class, a header file and a source file, with naming format `class_name.h` and `class_name.cpp` respectively, must be created.
    - Description of the methods to be implemented can be found in the table in Annex.

Annex

Class	Methods	Description
`Device`	`Device::Device(const string & name)`	Initialize a device with the specified name.
	`const string & Device::getName() const`	Returns the name of the component.
`Display`	`Display::Display(const string & name)`	Initialize a monitor screen with the specified name.
	`void Display::process(const string & data)`	Displays the input data string on the screen using `cout`.
`Processor`	`Processor::Processor(const string & name)`	Initialize a processor with the specified name.
	`void Processor::connectTo(Display & display)`	Connect the `Processor` object to a monitor screen (`Display`) object.
	`void Processor::process(const string & data)`	Inverts the character string it receives as input, adds the prefix `"PROCESSED:"`, and then sends the processed string to the connected monitor screen (`Display`) object. For example, if the input string is `"hello"`, then the processed data string to be sent to the monitor screen is `"PROCESSED: olleh"`.
`Keyboard`	`Keyboard::Keyboard(const string & name)`	Initialize a keyboard with the specified name.
	`void Keyboard::connectTo(Processor & cpu)`	Connect the `Keyboard` object to a `Processor` object.
	`void Keyboard::process()`	Default implementation of a keyboard, reads a string of real characters from the keyboard using the flow operator `cin`. Then processes this string by calling the `process()` method of the processor object connected to this keyboard object.
`CharKeyboard`	`CharKeyboard::CharKeyboard(const string & name)`	Initialize a character keyboard with the specified name.
	`void CharKeyboard::process()`	This is similar to the `Keyboard::Process()` function, but it only reads a single character (To do this you can use the standard function `cin.get()`).
`LineKeyboard`	`LineKeyboard::LineKeyboard(const string & name)`	Initialize a line keyboard with the specified name.
	`void LineKeyboard::process()`	This is similar to the `Keyboard::process()` function, but reads a complete line, i.e, reads characters until `'\n'`, and send a full line to the processor.


Device
Clase genérica que representa un componente del ordenador. De momento solo lleva un miembro que es el nombre.
+Device(name:const string &)
+getName() const(): const string &
Display
Muestra la cadena de caracteres que recibe en entrada por la pantalla.
+Display(name:const string &)
+process(data:const string &): void
Processor
Representa un procesador, recibe una cadena de caracteres en entrada, la transforma y envía el resultado a la salida.
+Processor(name:const string &)
+connectTo(display:Display &): void
+process(data:const string &): void
Keyboard
Representa una fuente de caracteres desde el teclado. Lee caracteres y las transmite a un Procesador.
+Keyboard(name:const string &)
+connectTo(cpu:Processor &): void
+process(): void
LineKeyboard
Lee una línea de caracteres desde teclado.
+LineKeyboard(name:const string &)
+process(): void
CharKeyboard
Lee caracteres, caracter por caracter desde teclado.
+CharKeyboard(name:const string &)
+process(): void