Menu

Arduino Object Oriented Programming (OOP)

This tutorial is an introduction to Arduino Object Oriented Programming. If you're already programming using C++ and OOP, and want to start writing Arduino OOP code, you're in the right place.

I'll show you through some examples how to re-write some of the most common Arduino tutorials using the OOP way.

At the end of the tutorial you'll get the complete code so you can directly test on your own Arduino board.

>> I also made a 40 minute step by step YouTube tutorial on Arduino OOP, if you prefer learning with video:

After watching the video, subscribe to the Robotics Back-End Youtube channel so you don’t miss the next tutorials!

You want to go deeper into OOP for Arduino?

Check out this complete ***Arduino OOP*** *course.*

4. Class for a Button
5. Complete Arduino Object Oriented code
6. Reorganize your Arduino OOP code
    6.1. Led.h
    6.2. Led.cpp
    6.3. Button.h
    6.4. Button.cpp
    6.5. Main
7. Arduino Object Oriented: it’s already everywhere

Arduino Object Oriented Programming limitations

Even if Oriented Object Programming is possible with Arduino, you have to know the limitations.

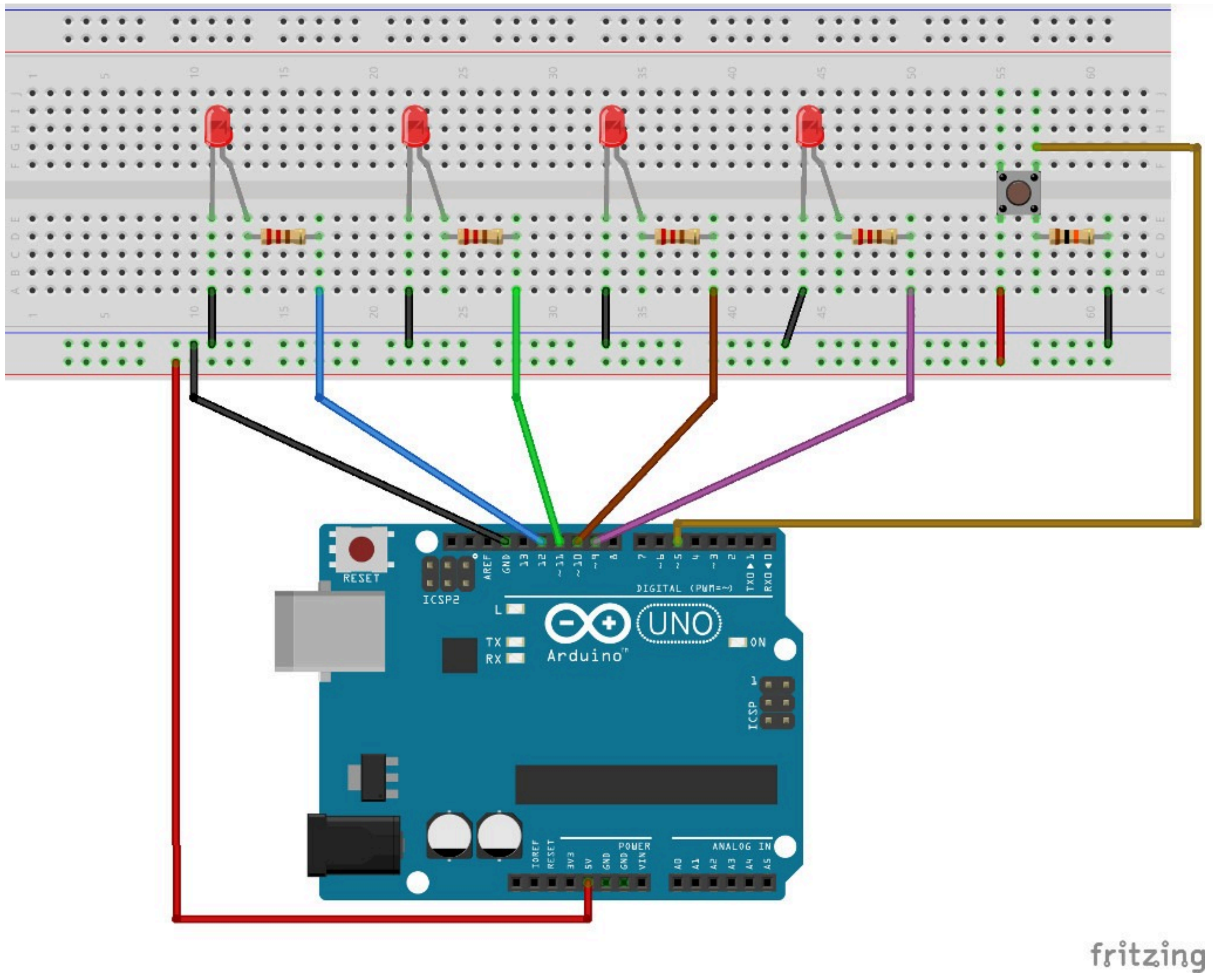
Basically, the Arduino language is a subset of C/C++. You can create classes, use inheritance, composition, and many other nice OOP functionalities, but:

- The STL library is not available (not natively, you can still use an external library to get most of the functionalities).
- You can’t use exceptions.

Hardware setup

Note: it’s OK if you don’t have any Arduino or hardware component available. You can still follow this tutorial and get all the knowledge you need.

We’ll be using the following hardware setup:



Component list:

- 1* Arduino Uno board (Any Arduino board will do)
- 1* breadboard
- 4* LEDs (any color you want)
- 4* 220 Ohm resistors (one for each LED)
- 1* push button
- 1* 10k Ohm resistor (for the push button)
- A few male-to-male wires

A class for a LED

An LED is a very basic component. You setup a digital pin to OUTPUT mode, and then you just need to set its state to HIGH or LOW.

Let’s create a simple class to wrap the LED functionalities.

```
class Led {
  private:
    byte pin;
  public:
    Led(byte pin) {
      // Use 'this->' to make the difference between the
      // 'pin' attribute of the class and the
      // local variable 'pin' created from the parameter.
      this->pin = pin;
      init();
    }

    void init() {
      pinMode(pin, OUTPUT);
      // Always try to avoid duplicate code.
      // Instead of writing digitalWrite(pin, LOW) here,
      // call the function off() which already does that
      off();
    }

    void on() {
      digitalWrite(pin, HIGH);
    }

    void off() {
      digitalWrite(pin, LOW);
    }
}; // don't forget the semicolon at the end of the class
```

You can now use this class to create a Led object.

```
#define LED_1_PIN 9

class Led {
    // class definition
};

Led led1(LED_1_PIN);
```

With the Led class we can hide all the Arduino stuff about digital pins. Note that the object must be created in the global scope if you want to be able to use it in the setup() and loop() functions.

Class for a Button

Let's now write some OOP code for a push button! The button is a little bit more complex, because we need to add a debounce functionality if we want it to remove the mechanical noise.

```
class Button {
  private:
    byte pin;
    byte state;
    byte lastReading;
    unsigned long lastDebounceTime = 0;
    unsigned long debounceDelay = 50;
  public:
    Button(byte pin) {
      this->pin = pin;
      lastReading = LOW;
      init();
    }

    void init() {
      pinMode(pin, INPUT);
      update();
    }

    void update() {
      // You can handle the debounce of the button directly
      // in the class, so you don't have to think about it
      // elsewhere in your code
      byte newReading = digitalRead(pin);

      if (newReading != lastReading) {
        lastDebounceTime = millis();
      }

      if (millis() - lastDebounceTime > debounceDelay) {
        // Update the 'state' attribute only if debounce is checked
        state = newReading;
      }

      lastReading = newReading;
    }

}; // don't forget the semicolon at the end of the class
```

Here again, **all the complexity is hidden**. Once you've implemented the debounce functionality for the button inside the class, you don't need to think about it anymore.

In your program, you just need to create a Button object and check whether it's pressed or not.

```
#define LED_1_PIN 9
#define BUTTON_PIN 5

class Led {
    // class definition
};

class Button {
    // class definition
};

Led led1(LED_1_PIN);
Button button1(BUTTON_PIN);

void loop() {
  if (button1.isPressed()) {
    led1.on();
  }
  else {
    led1.off();
  }
}
```

When the button is pressed, we turn on LED 1. As you can see the code in the loop() is quite small and clean.

Complete Arduino Object Oriented code

OOP is great for reusability. Do you remember we added 4 LEDs at the beginning of the tutorial? Well, now that we have a class for a LED, we just need to create additional objects, all the implementation is already done.

```

#define BUTTON_PIN 5

class Led {
  private:
    byte pin;
  public:
    Led(byte pin) {
      // Use 'this->' to make the difference between the
      // 'pin' attribute of the class and the
      // local variable 'pin' created from the parameter.
      this->pin = pin;
      init();
    }

    void init() {
      pinMode(pin, OUTPUT);
      // Always try to avoid duplicate code.
      // Instead of writing digitalWrite(pin, LOW) here,
      // call the function off() which already does that
      off();
    }

    void on() {
      digitalWrite(pin, HIGH);
    }

    void off() {
      digitalWrite(pin, LOW);
    }
}; // don't forget the semicolon at the end of the class

class Button {
  private:
    byte pin;
    byte state;
    byte lastReading;
    unsigned long lastDebounceTime = 0;
    unsigned long debounceDelay = 50;
  public:
    Button(byte pin) {
      this->pin = pin;
      lastReading = LOW;
      init();
    }

    void init() {
      pinMode(pin, INPUT);
      update();
    }

```

```

      if (millis() - lastDebounceTime > debounceDelay) {
        // Update the 'state' attribute only if debounce is checked
        state = newReading;
      }

      lastReading = newReading;
    }

    byte getState() {
      update();
      return state;
    }

    bool isPressed() {
      return (getState() == HIGH);
    }
}; // don't forget the semicolon at the end of the class

// Create your objects in the global scope so you can
// get access to them in the setup() and loop() functions
Led led1(LED_1_PIN);
Led led2(LED_2_PIN);
Led led3(LED_3_PIN);
Led led4(LED_4_PIN);
Button button1(BUTTON_PIN);

void setup() { }

void loop() {
  if (button1.isPressed()) {
    led1.on();
    led2.off();
    led3.on();
    led4.off();
  }
  else {
    led1.off();
    led2.on();
    led3.off();
    led4.on();
  }
}
```

Reorganize your Arduino OOP code

The previous code works well, but everything is in the same file. **We want to use OOP for reusability, modularity, readability, etc, but it's impossible if we write all the code in**

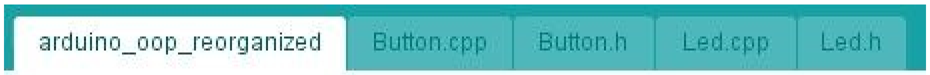
Each class will be on its own independent file. In fact, for one class we'll have 2 files: one Cpp file (.cpp) and one header file (.h). Your code will become much more readable. The classes you create will be also more reusable as you can include them in every file where you need them.

Note that creating other files for an Arduino program is quite tricky. You must create all your files inside your Arduino program folder. Example: if your program is named Test.ino, then it will be automatically saved on a Test/ folder (the Arduino IDE does that). You'll have to put all your files in the Test/folder as well, so the Arduino IDE can find them.

Go into the folder of your current Arduino program. Create 4 files:

- Led.h
- Led.cpp
- Button.h
- Button.cpp

The files won't appear in the Arduino IDE right away. You'll need to close the project, and open it again so the files will show up in tabs on the top.



>> Additional help: Step by Step process to split your Arduino program into different files:

And now, here is the complete code for all 5 files. The classes and functionalities are exactly the same as the code we just wrote before.

Led.h

```
#ifndef MY_LED_H
#define MY_LED_H

#include <Arduino.h>

class Led {

  private:
    byte pin;

```

In this file we just write the class declaration, which is the interface that any client using this class will use. This makes the class much easier to understand and use.

The header guards (first 2 lines, and last line) will make sure that the Led class will not be included more than once, if for example you have multiple `#include "Led.h"` in other parts of your program.

Also note that I've added `#include <Arduino.h>` at the beginning. Why? **This include is necessary to use the specific Arduino functions and types** (think of pinMode(), digitalWrite(), byte). In the "main" file we don't need to write it because it's automatically added when you compile your code. But in any other file, you need to add it by yourself.

Led.cpp

```
#include "Led.h"

Led::Led(byte pin) {
  // Use 'this->' to make the difference between the
  // 'pin' attribute of the class and the
  // local variable 'pin' created from the parameter.
  this->pin = pin;
  init();
}

void Led::init() {
  pinMode(pin, OUTPUT);
  // Always try to avoid duplicate code.
  // Instead of writing digitalWrite(pin, LOW) here,
  // call the function off() which already does that
  off();
}

void Led::on() {
  digitalWrite(pin, HIGH);
}

void Led::off() {
  digitalWrite(pin, LOW);
}
```

Button.h

```
    byte state;
    byte lastReading;
    unsigned long lastDebounceTime = 0;
    unsigned long debounceDelay = 50;

  public:
    Button(byte pin);

    void init();
    void update();

    byte getState();
    bool isPressed();
};

#endif
```

Same warning as for the Led.h file. Don’t forget to include the Arduino library at the beginning of the file. Also put all your class declaration inside header guards so you won’t include it twice in other parts of your code.

Button.cpp

```
#include "Button.h"

Button::Button(byte pin) {
  this->pin = pin;
  lastReading = LOW;
  init();
}

void Button::init() {
  pinMode(pin, INPUT);
  update();
}

void Button::update() {
    // You can handle the debounce of the button directly
    // in the class, so you don't have to think about it
    // elsewhere in your code
    byte newReading = digitalRead(pin);

    if (newReading != lastReading) {
      lastDebounceTime = millis();
    }

    if (millis() - lastDebounceTime > debounceDelay) {
      // Update the 'state' attribute only if debounce is checked
}

bool Button::isPressed() {
  return (getState() == HIGH);
}
```

Main

```
#include "Led.h"
#include "Button.h"

#define LED_1_PIN 9
#define LED_2_PIN 10
#define LED_3_PIN 11
#define LED_4_PIN 12

#define BUTTON_PIN 5

Led led1(LED_1_PIN);
Led led2(LED_2_PIN);
Led led3(LED_3_PIN);
Led led4(LED_4_PIN);
Button button1(BUTTON_PIN);

void setup() { }

void loop() {
  if (button1.isPressed()) {
    led1.on();
    led2.off();
    led3.on();
    led4.off();
  }
  else {
    led1.off();
    led2.on();
    led3.off();
    led4.on();
  }
}
```

Well, as you can see, the code is now much clearer and readable. Plus, you could import the Led and Button class in any other part of your application.

For example you could create a class named LedPanel. This class would contain an array of Led objects and handle them. In your "main", you could just import the LedPanel header

Arduino Object Oriented: it's already everywhere

If it's the first time you use Object Oriented Programming with Arduino, well… Don't think you're doing something new! In fact, many of the Arduino already use OOP.

A few OOP library examples:

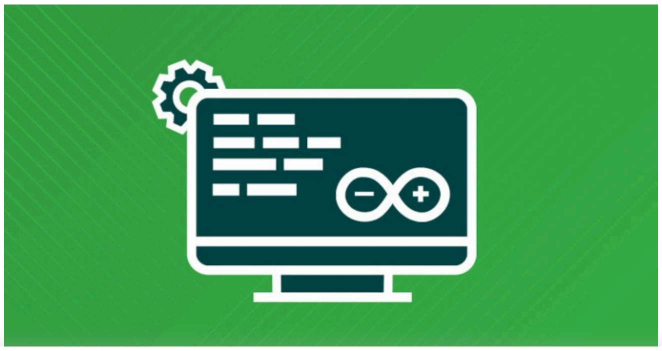
- Servo: control a servo motor
- Serial: communicate between your Arduino board and other devices
- Wire: communicate though the I2C protocol
- SevSeg: control a digit 7-segment display
- EEPROM: store values permanently in your Arduino board

If you're already familiar with OOP and want to use it in your Arduino programs, don't hesitate and do it! OOP is certainly not the answer to everything, but if you know how to use it right, there is no reason you couldn't get its benefits with Arduino.

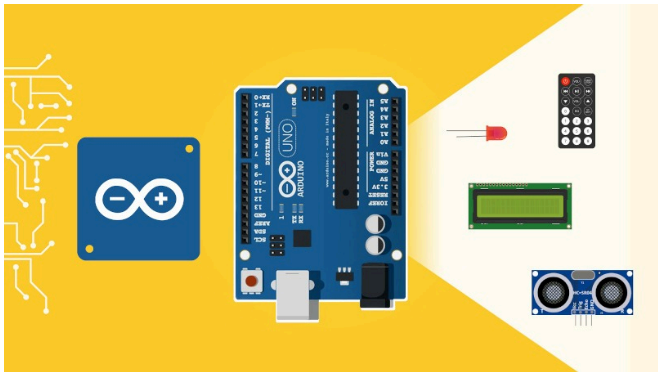
Arduino Tutorials

Did you find this tutorial useful?

If yes, this course is for you:

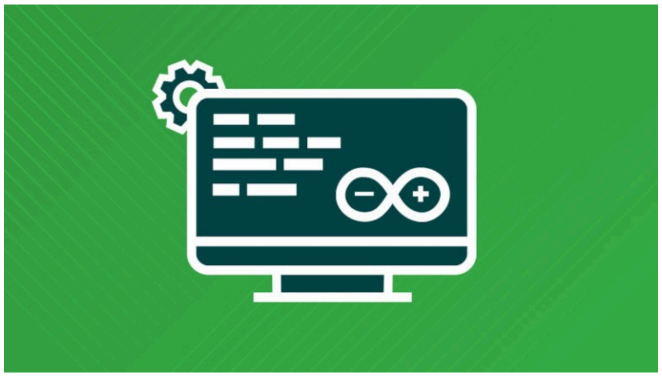


Want to learn Arduino programming?



Arduino Programming For Beginners

Learn Arduino OOP with a Complete Course



Arduino OOP