Chapter 4
Object-Oriented Techniques
for Microcontrollers

Object-oriented programs are built from various class objects that intuitively embody
the application through their actions and interrelations among each other. This
chapter introduces object-oriented real-time C++ methods using classes for LEDs,
PWM signal generators and communication interfaces.

4.1 Object Oriented Programming

Consider the application shown in Fig. 4.1.

![Diagram of LED application](image)

**Fig. 4.1** An application with four LEDs is shown
This application has four LEDs and two peripheral timers used as pulse-width modulated (PWM) signal generators. The LEDs L0 and L1 are connected to port pins P2.0 and P2.1, respectively. These LEDs have the same circuit as the one shown previously in Chap. 1, Fig. 1.1. They are controlled with bit manipulation of the microcontroller’s port P2, as introduced in the LED program of Sect. 1.1.

The LEDs L2 and L3 are connected to PWM signals generated from peripheral timers in the microcontroller. L2 is connected to timer0 and L3 is connected to timer1. Setting the duty cycle of the PWM signal to 0 or 100% switches the corresponding LED off or on, respectively. Intermediate duty cycles with values greater than 0% but less than 100% can be used for dimming the corresponding LED. Dimming is an additional feature that an LED on a simple digital I/O pin does not have.

We will now design a class hierarchy for the LED objects in Fig. 4.1. This is the class hierarchy that the developer in Fig. 4.1 is considering. The two types of LEDs can be represented with a base class and two derived classes. The base class is called led_base. The two derived classes are called led_port and led_pwm.

One potential implementation of the led_base class is shown below.

class led_base
{
public:
  virtual void toggle() = 0; // Pure abstract.
  virtual ~led_base() { } // Virtual destructor.

  // Interface for querying the LED state.
  bool state_is_on() const { return is_on; }

protected:
  bool is_on;

  // A protected default constructor.
  led_base() : is_on(false) { }

private:
  // Private non-implemented copy constructor.
  led_base(const led_base&) = delete;

  // Private non-implemented copy assignment operator.
  const led_base& operator=(const led_base&) = delete;
};