Laboratorul 5. Programarea orientată pe obiecte folosind placa de dezvoltare *Arduino*

Deoarece limbajul de programare pentru *Arduino Uno* este un limbaj de C++ simplificat care păstrează multe din proprietăţile unui limbaj de tip orientat pe obiecte, astfel că putem crea clase şi ierarhii de clase în codul sursă al aplicaţiei pentru *Arduino Uno*.

Pentru a crea clasele trebuie să respectăm exact acealeaşi regului pe care le avem în orice limbaj de tip orientat pe obiecte. Pentru a crea o clasă trebuie să folosim următoarea sintaxă:

```
class numeClasă
{
  public:
    void funcţie_1();
    void funcţie_2();
     int funcţie_3();
    virtual int read() {}; // funcţie virtuala
    virtual int readAnalog () = 0; // funcţie virtuala pura
  private:
    int _pin;
};
```

Orice clasă creată are nevoie de un constructor. Ca şi în orice limbaj de POO constructorul nu are tip şi are același nume ca și clasa:

```
numeClasă::numeClasa(int pin)
{
   pinMode(pin, OUTPUT);
   _pin = pin;
}
```

Sfaturi utile

Dacă clasa conţine o funcţie virtuală pură nu poate fi instanțiată direct! Apelarea unui constructor se poate face doar indirect prin constructorul clasei derivate.

Declararea funcţiilor din interiorul clasei se realizează exact ca în orice alt limbaj de tip POO:

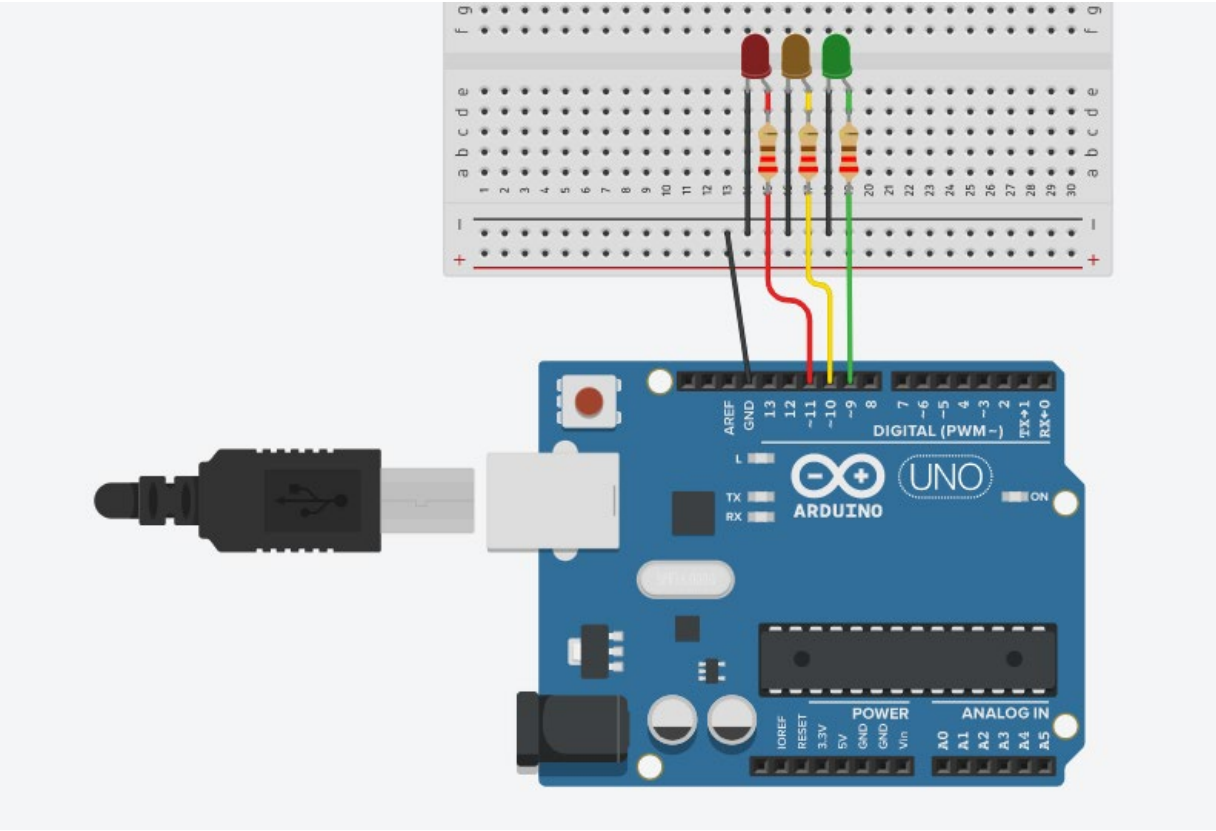
```
void numeClasa::funcţie_1()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

Limbajul de programare pentru modulele *Arduino* ne permit realizarea moştenirilor pentru extinderea claselor de bază. Sintaxa pentru realizarea moştenirii dintre clase este așa cum o știm din C++:

```
class numeClasaDerivata : public numeClasa
{
  public:
    inline funcţie_F1( byte userPin ) {
      pin = userPin;
    }
    //functia readAnalog() trebuie implementata ca este pura in clasa de baza
    virtual int readAnalog();
  private:
    byte pin;
};
```

Utilizând clase, funcţii virtuale şi clase derivate putem realiza foarte uşor o extindere a codului care rulează pe microcontroler în cazul în care se adaugă noi periferice la modulul de dezvoltare.

Exemplu



Codul sursa:

```
//enumerarea culorilor  verde 0, galben 1, rosu 2
enum color {VERDE, GALBEN, ROSU};

class Semafor
{
  /* variabilele declarate cu public pot fi apelate
  oricand este folosita clasa Semafor */
  public:
    Semafor(byte pinRosu, byte pinGalben, byte pinVerde);
    void Toggle(int color);
    void TurnOff(int color);
    void TurnOn(int color);
    void Go();
    void Careful();
    void Stop();
    bool GetState(int color);

  /*variabilele declarate cu private sunt
  folosite doar de catre clasa*/
  private:
  //declararea unui array gol pentru a stoca pinii
    byte pins[3];

    bool states[3]={0};
  //initializarea unui array pentru LEDuri cu zero. 0 -off; 1 -on
    void init();
};

void Semafor::init() {
  for(int i=0; i<3; i++) {
    pinMode(this->pins[i], OUTPUT);
  }
}

//constructorul pentru clasa Semafor
Semafor::Semafor(byte pinRosu, byte pinGalben, byte pinVerde) {
  this->pins[ROSU] = pinRosu;
  this->pins[GALBEN] = pinGalben;
  this->pins[VERDE] = pinVerde;
  this->init(); //aloca pinii pe pozitii
}

bool Semafor::GetState(int color) {
  return this->states[color];
}

```

```
47 //folosit pentru a schimba culorile semaforului
48 void Semafor::Toggle(int color) {
49   this->states[color] = !this->states[color];
50   if(this->states[color]) {
51     digitalWrite(this->pins[color], HIGH);
52   } else {
53     digitalWrite(this->pins[color], LOW);
54   }
55 }
56
57 //schimba starea pinului pe on
58 void Semafor::TurnOn(int color) {
59   if(!this->states[color]) {
60     this->Toggle(color);
61   }
62 }
63
64 //schimba starea pinului pe off
65 void Semafor::TurnOff(int color) {
66   if(this->states[color]) {
67     this->Toggle(color);
68   }
69 }
70
71 //doar LED-ul verde este aprins
72 void Semafor::Go() {
73   this->TurnOff(GALBEN);
74   this->TurnOff(ROSU);
75   this->TurnOn(VERDE);
76 }
77
78 //doar LED-ul galben este aprins
79 void Semafor::Careful() {
80   this->TurnOff(ROSU);
81   this->TurnOff(VERDE);
82   this->TurnOn(GALBEN);
83 }
84
85 //doar LED-ul rosu este aprins
86 void Semafor::Stop() {
87   this->TurnOff(GALBEN);
88   this->TurnOff(VERDE);
89   this->TurnOn(ROSU);
90 }
```

```
91
92  Semafor semaforMasini(11,10,9);
93
94  void setup() {
95  }
96
97  void loop() {
98    semaforMasini.Stop();
99    delay(3000);
100   semaforMasini.Go();
101   delay(4000);
102   semaforMasini.Careful();
103   delay(1000);
104 }
```

Tema

- De implementat un semafor pentru pietoni, *semaforPietoni*, folosindu-se de clasa *Semafor* deja definita. Semafoarele se vor sincroniza astfel încât atunci când pietonii au verde, vehiculele să aibă roșu (atenție la decalajele de timp).
- De implementat un *difuzor* semaforului de pietoni pentru oamenii nevazatori. Acesta va emite un sunet atunci cand semaforul este pe rosu si un alt sunet atunci cand semaforul pentru pietoni este verde.
- De implementat un *buton* pentru semaforul de pietoni. Odata apasat butonul, semaforul pentru masini va trece pe culoarea rosie, iar cel de pietoni se va face verde.

Pentru vehicule timpii vor fi:

- 4 secunde roşu;
- 6 secunde verde;
- 2 secunde galben;

Pentru pietoni timpii vor fi:

- 3 secunde verde;
- 9 secunde roşu;

Diagrama cu timpi a semafoarelor sunt prezentate în următorul tabel:

Semafor mașini	4 secunde Roșu	6 secunde Verde	2 secunde Galben	4 secunde Roșu	6 secunde Verde	2 secunde Galben
Semafor pietoni	3 secunde Verde	9 secunde Roșu		3 secunde Verde	9 secunde Roșu	

Tabel 1. Diagrama timpi