Class

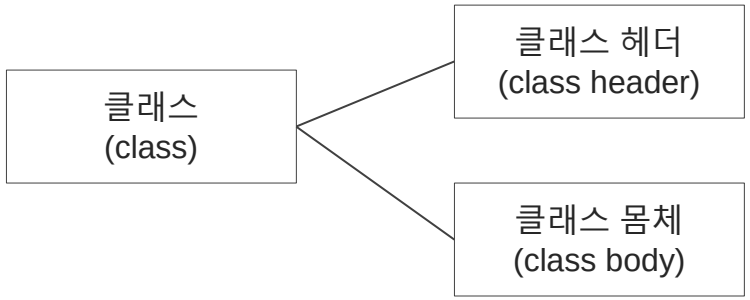
Using class on Arduino

Software Project II, 2024

클래스 (Class) (1)

- 객체지향 프로그래밍 (Object-Oriented Programming)
  - 객체 지향 프로그래밍에서는 모든 데이터를 객체 (object)로 간주
  - 이들 객체(object) 는OOP 의 핵심 구성품

```
class class_name
{

};
```




```
class Foo
{
  public:
   int data;
   std::string str;

   void foo() { std::cout << “hello” << “\n”; }
}
```

- 클래스 (Class)
  - 클래스(class)란 객체를 정의하는 틀 또는 설계도
  - 이들 클래스를 사용하여 여러 객체를 생성하여 사용

- 클래스로 무엇을 할 수 있는가?
  - 이름 공간 (name space) 제공
  - 새로운 인스턴스(instance)들을 생성
    - 프로그램에서의 object 는 instance 를 의미
  - 사용자 정의 데이터 타입을 제공

```
int v{100};

Foo foo;
```

  - int 는 ?
  - Foo 는?
  - 객체(instance) 는?
  - Foo 객체 타입의 인스턴스 foo 를 정적할당

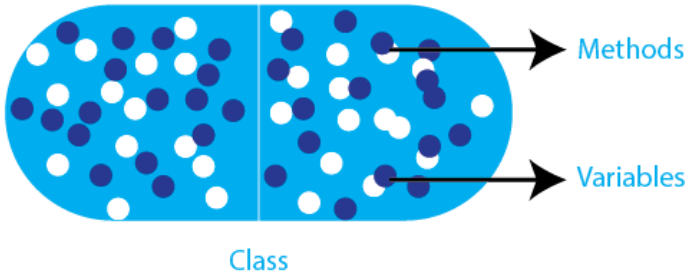
클래스 (Class) (2)

- 클래스 (class)
  - 상태 (States) + 동작 (Behaviors)
  - 속성 (Attributes/Properties) + 메서드(Methods)
  - 항목 (Fields) + 함수 (Functions)
  - 데이터 멤버 (data member) + 멤버함수 (member Function)

  - 위 용어는 모두 동일
    - 혼용해서 사용해두 무방

- 캡슐화 (Encapsulation)
  - 상태와 동작을 바인딩
    - 클래스

  - 정보 은닉 기능 제공
    - Information Hiding
    - Visibility



- 클래스

```
class class_name
{
        Data Members

        Member Functions

};
```

  - 데이터 중심의 설계가 먼저
    - 데이터 중심이란?
      - 무엇으로 객체를 대표하는가?
      - 객체를 표현하는 대표적인 속성이 무엇인가?

  - 예를 들어, 데이터베이스 (Database)
    - Table name: Person
    - Fields: ID
    - Name
    - Year (Birth year)

ID	NAME	YREAR
2023123	Robert	2003
...		
2023923	David	2002

클래스 (Class) (3)

- 데이터베이스 (Database)
  - Person

ID	NAME	YEAR
2023123	Robert	2003
...		
2023923	David	2003

  - Person Record 를 표현하는 대표적인 속성들이 무엇인가?

- 클래스 Person

```
class Person
{
 public:
  std::string id;
  std::string name;
  int year;
}
```

- 클래스의 행동
  - 대표 속성들로 무엇을 서비스 할 수 있는가?
    - Person 클래스의 기능
      - 객체가 무엇을 서비스 할 수 있는가?
    - 나이 (age) 또한 필요하지 않는가?
      - 필드 age 필요?
    - Person 의 나이는 정적인가?
      - calculate_age

- 메서드 calculate_age 설계

```
int calculateAge(int current){
  return yearCurrent – year;
}
```

클래스 (Class) (4)

- 클래스 Person
- Example

```
#include <iostream>

class Person {
 public:

  std::string id;
  std::string name;
  int year;

 public:
  int calculate_age(int current){
    return current  - year;
  }
}
```

ID	NAME	YEAR
2023123	Robert	2003
...		
2023923	David	2002

- 객체 생성
  - 클래스 Person 로 부터 객체(instance)를 생성

```
#include <iostream>

class Person {
 public:
  std::string id;
  std::string name;
  int year;

 public:
  int calculate_age(int current) { return current - year; }
};

int main(int argc, char *argv[]) {
  Person p{"2003923", "David", 2002};
  std::cout << "p.id = " << p.id << "\n";
  std::cout << "p.name = " << p.name << "\n";
  std::cout << "p.year = " << p.year << "\n";
  std::cout << "p.age = " << p.calculate_age(2023) << "\n";

  return 0;
}
```

```
$ ./a.out
p.id = 2003923
p.name = David
p.year = 2002
p.age = 21
```

클래스 (Class) (5)

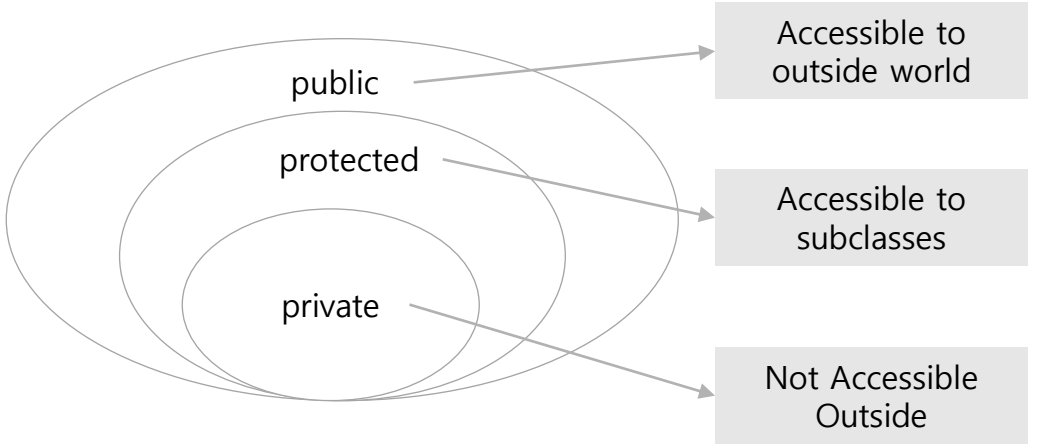
- Access Modifier (Access Specifier)
  - 접근 제한자(한정자): 접근 범위를 제한
    - Public:
    - Protected:
    - Private:
  - Encapsulation
- Example

```
#include <iostream>

class Foo {
 public:
  int data_pb;

 protected:
  int data_pt;

 private:
  int data_pr;
};

int main(int argc, char *argv[]) {
  Foo foo;  // static allocation
  std::cout << "foo.data = " << foo.data_pb << "\n"; ------------- <1>
  std::cout << "foo.data = " << foo.data_pt << "\n"; ------------- <2>
  std::cout << "foo.data = " << foo.data_pr << "\n"; ------------- <3>

  return 0;
}
```

함수 main 은 궁금해 하지 말자.

```
odo@iodolab:~/repo/cpp-tutorials/class/ex01$ g++ -Wall main.cpp
main.cpp: In function ‘int main(int, char**)’:
main.cpp:40:37: error: ‘int Foo::data_pt’ is protected within this
context
   40 |   std::cout << "foo.data = " << foo.data_pt << "\n";
...
main.cpp:43:37: error: ‘int Foo::data_pr’ is private within this
context
   43 |   std::cout << "foo.data = " << foo.data_pr << "\n";
```

- Access Modifier 기본



  - 멤버 데이터 및 멤버 함수 모두에 해당
    - friend 예외

클래스 (Class) (6)

- Example

```
#include <iostream>

class Foo {
  int data;
};

struct Bar {
  int data;
};

int main(int argc, char *argv[]) {
  Bar bar;
  std::cout << "bar.data = " << bar.data << "\n";

  Foo foo;
  std::cout << "foo.data = " << foo.data << "\n";

  return 0;
}
```

함수 main 은 궁금해 하지 말자.

```
$ g++ -Wall main.cpp
main.cpp: In function ‘int main(int, char**)’:
main.cpp:93:37: error: ‘int Foo::data’ is private within this context
   93 |   std::cout << "foo.data = " << foo.data << "\n";
```

- Access Modifier
  - Structure
    - public
  - Class
    - private

```
#include <iostream>

class Foo {
 public:
  int data;
};

struct Bar {
  int data;
};

int main(int argc, char *argv[]) {
  Bar bar;
  std::cout << "bar.data = " << bar.data << "\n";

  Foo foo;
  std::cout << "foo.data = " << foo.data << "\n";

  return 0;
}
```

함수 main 은 궁금해 하지 말자.

클래스 (Class) (7)

- In C++, how to access members of a class
  - 접근 (Access)
    - 값을 할당/변경하거나 값을 가져오려 할 때

- Example

```
#include <iostream>

class Foo {
 public:
  void get_data_protected() {
    std::cout << "foo::data_pr = " << data_pt << "\n";
  }
  void get_data_private() {
    std::cout << "foo::data_pt = " << data_pr << "\n";
  }

 public:
  int data_pb;

 protected:
  int data_pt;

 private:
  int data_pr;
};
```

```
int main(int argc, char *argv[]) {
  Foo foo;

  std::cout << "foo.data = " << foo.data_pb << "\n";

  foo.get_data_private();
  foo.get_data_protected();

  return 0;
}
```

함수 main 은 궁금해 하지 말자.

  - 외부에서 접근 가능한 함수를 만들어 제공
    - public
      - get_data_private
      - get_data_protected

```
$ ./a.out
foo.data = 1858244232
foo::data_pt = 28670
foo::data_pr = 1857153002
```

  - 데이터 멤버의 초기화

클래스 (Class) (8)

- Default Constructor
  - Zero-argument constructor
  - 기본 생성자
    - 매개변수를 가지지 않는 생성자
  - 생성자를 정의 하지 않은 경우 컴파일 타임에서 자동 삽입
    - Implicit generated constructor
  - 명시적으로 생성자를 정의하는 경우 기본 생성자를 추가

```
#include <iostream>

class Foo {
 public:
  void print_attr() {
    std::cout << "Foo::data01 = " << data01 << "\n";
    std::cout << "Foo::data02 = " << data02 << "\n";
  }
 private:
  int data01;
  int data02;
};

int main(int argc, char *argv[]) {
  Foo foo{};
  foo.print_attr();

  Foo foo1{100, 200}; ------------------------------------------ <1>
  foo1.print_attr();
  return 0;
}
```

- Example

```
#include <iostream>

class Foo {
 public:
  Foo() {
    std::cout << "Foo::Foo() default "
              << "\n";
  }

 public:
  void print_attr() {
    std::cout << "Foo::data01 = " << data01 << "\n";
    std::cout << "Foo::data02 = " << data02 << "\n";
  }

 private:
  int data01;
  int data02;
};

int main(int argc, char *argv[]) {
  Foo foo;
  foo.print_attr();

Foo foo1{};
  foo.print_attr();

  return 0;
}
```

함수 main 은 궁금해 하지 말자.

클래스 (Class) (9)

- Example

```
#include <iostream>

class Foo {
 public:
  Foo() {
    std::cout << “Foo::Foo() default “
              << “\n”;
  }

 public:
  void print_attr() {
    std::cout << "Foo::data01 = " << data01 << "\n";
    std::cout << "Foo::data02 = " << data02 << "\n";
  }

 private:
  int data01;
  int data02;
};

int main(int argc, char *argv[]) {
  Foo foo;
  foo.print_attr();

  Foo foo1{};
  foo.print_attr();

  return 0;
}
```

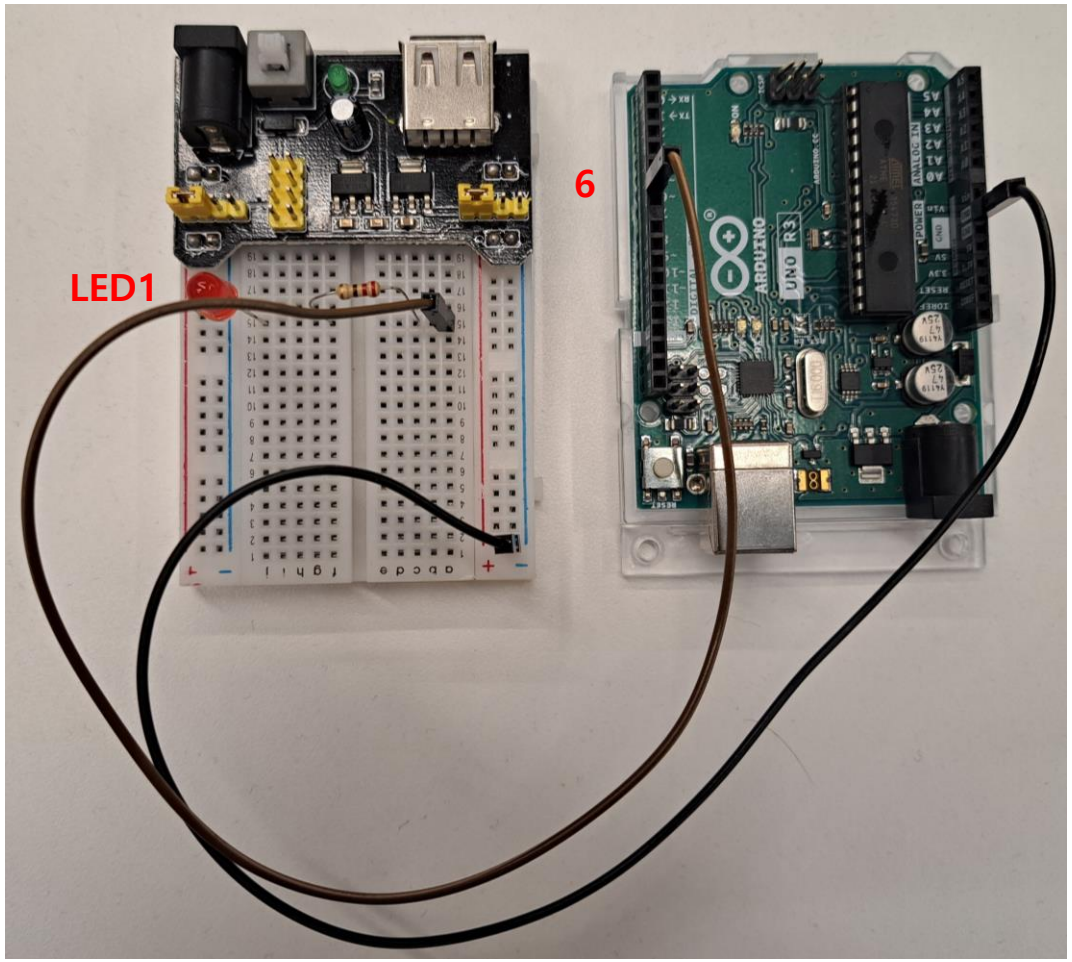
함수 main 은 궁금해 하지 말자.

```
$ ./a.out
Foo::Foo() default
Foo::data01 = 117608072
Foo::data02 = 30768
Foo::Foo() default
Foo::data01 = 117608072
Foo::data02 = 30768
```

- 기본 생성자를 명시적으로 구현
  - 멤버들의 초기화를 직접 수행

회로-01 (1-Single LED)

- 회로구성
  - LED (극성 주위), 적적갈금 저항
  - Arduino PIN-PORT 6, GND



- 목표
  - 500 msec 간격으로 LED 를 toggle 하도록 코드를 작성한다.
  - File: toggle_led_01.ino

```
#define PIN_LED 6
#define INTERVAL 500

void setup() {
  // put your setup code here, to run once:
  pinMode(PIN_LED, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(PIN_LED, 0);
  delay(INTERVAL);

  digitalWrite(PIN_LED, 1);
  delay(INTERVAL);
}
```

회로-01 (2-Single LED)

- toggle_led_02.ino

```
#define PIN_LED 6
#define INTERVAL 500
unsigned int status_led; // on/off = 0/1

void setup() {
  // put your setup code here, to run once:
  pinMode(PIN_LED, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(PIN_LED, status_led);
  status_led ^= 1; // using XOR (Exclusive OR) bit operator
  delay(INTERVAL);
}
```

- LED class
  - Behaviors for LED class
    - Turn on
    - Turn off
    - turn

- LED class
  - toggle_led_03.ino

```
#define PIN_LED 6
#define INTERVAL 500
unsigned int status_led; // on/off = 0/1

class Led {
  public:
  Led() {
    pinMode(PIN_LED, OUTPUT);
  }
  public:
  void turn(unsigned int status) {
    digitalWrite(PIN_LED, status);
    Serial.println("led mode: " + (String)status);
  }
};

Led led;

void setup() {
  Serial.begin(115200);
}

void loop() {
  led.turn(status_led);
  status_led ^= 1; // using XOR (Exclusive OR) bit operator
  delay(INTERVAL);
}
```

회로-01 (3-Single LED)

- LED class
  - toggle_led_04.ino
  - 선언부와 구현부를 분리

```
#define PIN_LED 6
#define INTERVAL 500

unsigned int status_led; // on/off = 0/1

class Led {
  public:
    Led();
  public:
    void turn(unsigned int status);
};

Led::Led() {
  pinMode(PIN_LED, OUTPUT);
}

void Led::turn(unsigned int status) {
  digitalWrite(PIN_LED, status);
  Serial.println("led mode: " + (String)status);
}
```

```
Led led;

void setup() {
  Serial.begin(115200);
}

void loop() {
  led.turn(status_led);
  status_led ^= 1; // using XOR (Exclusive OR) bit operator
  delay(INTERVAL);
}
```

  - Method toggle 을 구현하려면
    - 매번 외부에서 toggle logic 을 기술하기 싫다.

```
void loop() {
  led.toggle();
  delay(INTERVAL);
}
```

회로-01 (4-Single LED)

- LED class
  - toggle_led_05.ino
  - Method toggle 구현

```
#define PIN_LED 6
#define INTERVAL 500

class Led {
  public:
    Led(unsigned int status);
  public:
    void turn(unsigned int status);
    void toggle();
  private:
    unsigned int status;
};

Led::Led(unsigned int status): status{status} {
  pinMode(PIN_LED, OUTPUT);
}

void Led::turn(unsigned int status) {
  digitalWrite(PIN_LED, status);
  Serial.println("led mode: " + (String)status);
}

void Led::toggle() {
  digitalWrite(PIN_LED, status);
  Serial.println("toggle-led: " + (String)status);
  status ^= 1; // using XOR (Exclusive OR) bit operator
}
```

```
Led led{1};

void setup() {
  Serial.begin(115200);
}

void loop() {
  led.toggle();
  delay(INTERVAL);
}
```

  - Method toggle 을 구현하려면
    - 매번 외부에서 toggle logic 을 기술하기 싫다.

회로-01 (5-Single LED)

- LED class
  - toggle_led_06.ino, led.h, led.cpp
  - 헤더와 구현을 분리하여 관리

- toggle_led_06.ino

```
#include "led.h"
#define INTERVAL 500


Led led{1};

void setup() {
  Serial.begin(115200);
}

void loop() {
  led.toggle();
  delay(INTERVAL);
}
```

- led.h

```
#ifndef LED_H
#define LED_H

class Led {
  public:
    Led(unsigned int status);
  public:
    void turn(unsigned int status);
    void toggle();
  private:
    unsigned int status;
};
#endif
```

  - header guard
    - 헤더파일의 중복 포함을 피하기 위함
    - 즉 코드 상에서 헤더파일이 여러 번 include 되어도 실제로 한 번만 include 됨을 보장하기 위함

    - #pragma once eh 사용가능

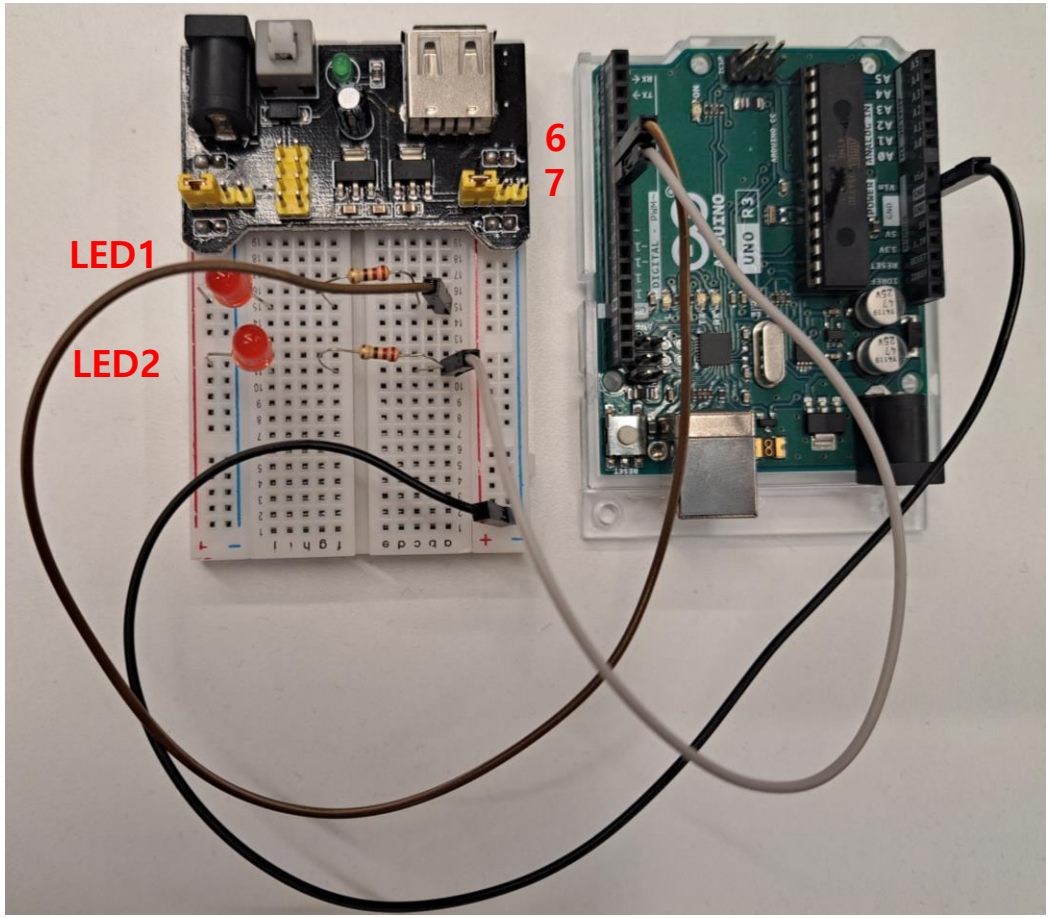
회로-01 (6-Single LED)

- led.cpp

```
#include <Arduino.h>

#include "led.h"
#define PIN_LED 6

Led::Led(unsigned int status): status{status} {
  pinMode(PIN_LED, OUTPUT);
}

void Led::turn(unsigned int status) {
  digitalWrite(PIN_LED, status);
  Serial.println("led mode: " + (String)status);
}

void Led::toggle() {
  digitalWrite(PIN_LED, status);
  Serial.println("toggle-led: " + (String)status);
  status ^= 1; // using XOR (Exclusive OR) bit operator
}
```

회로-02 (1-Two LEDs)

- 회로구성
  - LEDx2 (극성 주위), 적적갈금 저항x2
  - Arduino PIN-PORT 6, 7, GND



- 두 개의 LED 객체를 운용
  - LED01 은 켜짐과 동시에 LED02 는 꺼짐을 500 ms 간격으로 toggle 한다.
  - toggle_led_07.ino

```
#include "led.h"
#define INTERVAL 500
#define PIN_LED_01 6
#define PIN_LED_02 7

Led led1{PIN_LED_01, 0};
Led led2{PIN_LED_02, 1};

void setup() {
  Serial.begin(115200);
}

void loop() {
  led1.toggle();
  led2.toggle();
  delay(INTERVAL);
}
```

  - led.h 와 led.cpp 를 작성하세요.