# Module 15: Inheritance

## Learning Objectives

1. Design and implement classes that use inheritance.
2. Write derived class member functions that reuse base class member functions.
3. Identify the accessibility of member variables and member functions according to their access specifiers (i.e., private, protected, public).

## Process Skills

1. Information processing. Extract structural patterns from sample code.
2. Critical thinking. Use prior knowledge to interpret and explain related concepts.

Please fill in the roles for each member of your team. Take a look at the description of each role to see its responsibilities. If there are only three people in the group, please assign the same person to the **Presenter** and **Reflector** role. It is a good idea to select roles that you have not recently taken.

Team name: _____ Date: _____

| Role | Team Member Name |
|---|---|
| **Manager.** Keeps track of time and makes sure everyone contributes appropriately. | |
| **Presenter.** Talks to the facilitator and other teams. | |
| **Reflector.** Considers how the team could work and learn more effectively. | |
| **Recorder.** Records all answers and questions and makes the necessary submission. | |

For virtual activities: Once you select your roles, change your Zoom name using the format and example below.
*Format:* *Group X: First name, Last name initial / Role*
*Example:* *Group 1: Paul I / Presenter*

## Model 1. Inheritance (24 min) Start time: _____

```
// volunteer.h
#include <iostream>
#include <vector>

class Volunteer {
 public:
  Volunteer() : Volunteer("Anonymous", 0.0) {}
  Volunteer(const std::string& name, double hours_worked)
    : name_(name), hours_worked_(hours_worked) {}

  const std::string& Name() const {
    return name_;
  }

  void LogHoursWorked(double hours) {
    if (hours > 0) {
      hours_worked_ += hours;
    }
  }

  double HoursWorked() const {
    return hours_worked_;
  }

  void Display() const {
    std::cout << name_ << "(" << hours_worked_ << ")\n";
  }

 private:
  std::string name_;
  double hours_worked_;
 };

class VolunteerManager : public Volunteer {
 public:
  void AddVolunteer(const Volunteer& volunteer) {
    team_.push_back(volunteer);
  }

  int TeamCount() const { return team_.size(); }

 private:
  std::vector<Volunteer> team_;
```

```
};
```

```
// main.cc
#include <iostream>
#include "volunteer.h"

int main() {
  Volunteer p("Paul", 2.0);

  Volunteer j("JC", 3.0);
  VolunteerManager ch;
  ch.AddVolunteer(p);
  ch.AddVolunteer(j);

  p.Display();
  j.Display();
  ch.Display();
  std::cout << "Team count: " << ch.TeamCount() << "\n";
  return 0;
}
```

**Screen output:**
Paul(2)
JC(3)
Anonymous(0)
Team count: 2

1. Inheritance represents an "is-a" relationship. In our example, we can say that a VolunteerManager is a Volunteer because they can do everything a Volunteer can, but it can additionally store information about their team. Which class can perform more specialized actions than the other? Place a check (✓) beside your answer.

   a. Volunteers can do more actions than VolunteerManagers.
   b. VolunteerManagers can do more actions than Volunteers. ✓

2. *Derived classes* inherit all member variables and member functions from a *base class* without rewriting them. As a result, *derived classes* often contain more member variables and member functions than their *base class*. Identify the *base class* and *derived class* from our example. Place a check (✓) beside your answer.

   a. Volunteer is the base class and VolunteerManager is the derived class. ✓
   b. VolunteerManager is the base class and Volunteer is the derived class.

3. What pattern do we use to express inheritance in C++? *Note: We can use private or protected instead of public, but in this course and most cases, we use public.* Place a check (✓) beside your answer.

   a. class <Base class name> : public <Derived class name> { <class body>};
   b. class <Derived class name> : public <Base class name> { <class body>}; ✓

4. Private member variables and member functions are only accessible from inside the class they were defined. Public member variables and member functions are accessible to any function or class, including derived classes. Which among the following VolunteerManager member functions would be valid? Place a check (✓) beside your answer. Select all that apply.

   a. void VolunteerManager::ShowHoursWorked() {
      std::cout << hours_worked_ << "\n";
      }
   b. void VolunteerManager::ShowTotalHours() { ✓
      std::cout << HoursWorked() << "\n";
      }

5. The sample code illustrates that we can call the Display function on a VolunteerManager object because a VolunteerManager inherits all member functions and member variables of its base class, Volunteer. Take note that we did not rewrite the Display member function inside VolunteerManager. What other member functions and member variables can we access from a VolunteerManager object? Place a check (✓) beside your answer. Select all that apply.

   a. Name (e.g., ch.Name()) ✓
   b. HoursWorked (e.g., ch.HoursWorked()) ✓
   c. LogHoursWorked (e.g., ch.LogHoursWorked(2.5)) ✓
   d. name_ (e.g., ch.name_)
   e. hours_worked_ (e.g., ch.hours_worked_)

6. Create a DonationManager class that is responsible for collecting money donated by customers or donors. It inherits from the Volunteer class and keeps track of the total donations it receives. You don't need to create a constructor; we will use the default constructor provided by the compiler.

   In the main function, create a DonationManager object. Donate $100.0, log 2 hours of work, call its Display member function, and display the total donations it received. In the main function, create a DonationManager object. Donate $100.0, log 2 hours of work, call its Display member function, and display the total donations it received.

```
// donationmanager.h

class DonationManager: public Volunteer {
 public:
  void Donate(double donation) {
    if (donation > 0.0) {
      total_donations_ += donation;
    }
  }
  double TotalDonations() const { return total_donations_; }
 private:
  double total_donations_;
};
```

```
// main.cc
#include "donationmanager.h"
int main() {
  DonationManager manager;
  manager.Donate(100.0);
  manager.LogHoursWorked(2.0);
  manager.Display();
  std::cout << "Total donations: " << manager.TotalDonations() << "\n";
 return 0;
}
```

**STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

## Model 2. Derived class constructors (12 min) Start time: _____

```
// refined VolunteerManager
class VolunteerManager : public Volunteer {
 public:
  VolunteerManager() : Volunteer() {}
  VolunteerManager(const std::string& name, double hours_worked)
    : Volunteer(name, hours_worked) {}

  void AddVolunteer(const Volunteer& volunteer) {
    team_.push_back(volunteer);
  }

 int TeamCount() const { return team_.size(); }

private:
 std::vector<Volunteer> team_;
};
```

```
// refined main.cc
#include <iostream>
#include "volunteer.h"
int main() {
  Volunteer p("Paul", 2.0);

  Volunteer j("JC", 3.0);
  VolunteerManager ch("Chang Hyun", 4.0);
  ch.AddVolunteer(p);
  ch.AddVolunteer(j);

  p.Display();
  j.Display();
  ch.Display();
  std::cout << "Team count: " << ch.TeamCount() << "\n";
  return 0;
}
```

**Screen output:**
Paul(2)

JC(3)
Chang Hyun(4)
Team count: 2

7. Recall what the Volunteer's default constructor does when it is called. Place a check (✓) beside your answer.

```
Volunteer() : Volunteer("Anonymous", 0.0) {}
```

   a. The default constructor calls the overloaded Volunteer constructor passing in "Anonymous" and 0.0 as arguments. ✓
   b. The default constructor creates a different Volunteer object passing in "Anonymous" as the argument.
   c. The default constructor displays "Anonymous" on the screen.

8. What do you think the VolunteerManager's constructor overload will do when it is called?

```
VolunteerManager(const std::string& name, double hours_worked)
    : Volunteer(name, hours_worked) {}
```

   a. It will call Volunteer's default constructor and set the name of the VolunteerManager to "Anonymous".
   b. It will call Volunteer's constructor overload, passing in the name and hours_worked parameters. Volunteer's constructor overload will then set the name and hours_worked for the VolunteerManager. ✓
   c. The code is syntactically incorrect. It should call the VolunteerManager constructor instead.

9. What is the most likely reason why derived class constructors call their base class' constructor? *Take note that if you do not provide a constructor for a derived class, it will create one for you. The generated derived class default constructor will call the base class default constructor if available. It will produce an error if the base class does not have a default constructor.*

   a. After a derived class is instantiated, it creates a base class object and copies its member variable values to the derived class object.
   b. C++ syntax does not require derived classes to call the base class constructor. It is only used for readability.
   c. The base class must be constructed first so the derived class can inherit its member variables and member functions. ✓

10. Create a default constructor for your DonationManager class that calls Volunteer's default constructor.

    Then, create a constructor overload that takes in a name, hours worked, and an initial amount for total donations. Call Volunteer's constructor overload to set the name and hours worked. Assign the total donations using member initialization. Only provide code for the two constructors.

```
DonationManager(): Volunteer() { }

DonationManager(const std::string &name, double hours_worked, double total_donations)
  : Volunteer(name, hours_worked), total_donations_(total_donations) { }
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

## Model 3. Member function override (9 min) Start time: _____

```
// refined VolunteerManager
class VolunteerManager : public Volunteer {
 public:
  VolunteerManager() : Volunteer() {}
  VolunteerManager(const std::string& name, double hours_worked)
    : Volunteer(name, hours_worked) {}

  void AddVolunteer(const Volunteer& volunteer) {
    team_.push_back(volunteer);
  }

  int TeamCount() const { return team_.size(); }

  void Display() const {
    Volunteer::Display();
    if (team_.size() == 0) {
      std::cout << "No team members.\n";
    } else {
      std::cout << "Team members:\n----------\n";
      for (Volunteer v : team_) {
        v.Display();
      }
      std::cout << "----------\n";
    }
  }
```

```
 private:
  std::vector<Volunteer> team_;
};
```

```
// main.cc
#include <iostream>
#include "volunteer.h"
int main() {
 Volunteer p("Paul", 8.0);
 Volunteer j("JC", 10.0);
 VolunteerManager ch("Chang Hyun", 9.0);
 ch.AddVolunteer(p);
 ch.AddVolunteer(j);
 p.Display();
 j.Display();
 ch.Display();
 std::cout << "Team count: " << ch.TeamCount() << "\n";
 return 0;
}
```

**Screen output:**

```
Paul(2)
JC(3)
Chang Hyun(4)
Team members:
----------
Paul(2)
JC(3)
----------
Team count: 2
```

11. Compare Model 1's screen output with that of Model 3. What additional information was shown? Place a check (✓) beside your answer.

    a. The hours worked of the ch object.
    b. The list of team members assigned to the ch object. ✓
    c. The ch object's team count.

12. Which code block in the modified Display member function does your group think displayed the list of team members? Place a check (✓) beside your answer.

    a. Volunteer::Display
    b. if-else block ✓

13. Which code block in the modified Display member function does your group think displayed the VolunteerManager's name and hours worked? Place a check (✓) beside your answer.

    a. Volunteer::Display ✓
    b. if-else block

14. Derived class member functions can call base class member functions. According to the sample code, what does your group think is the syntax for calling a Base class member function? *Note: If the derived class has no member function with the same name as the base class member function, you can call it as if it were defined in the derived class.* Place a check (✓) beside your answer.

    a. <Derived class>::<Base class>::<member function name(<parameters>)
    b. <Derived class>::<member function name>(<parameters>)
    c. <Base class>::<member function name>(<parameters>) ✓

15. Create a Display member function for your DonationManager class. Reuse Volunteer's Display member function to display the name and hours worked of the DonationManager. In addition, display the total donations received.

```
void DonationManager::Display() {
  Volunteer::Display();
  std::cout << "Total donations received: " << total_donations_ << "\n";
}
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

## Model 4. Protected access specifier (6 min) Start time: _____

```
// refined Volunteer and VolunteerManager
#include <iostream>
#include <vector>

class Volunteer {
 public:
  Volunteer() : Volunteer("Anonymous", 0.0) {}
  Volunteer(const std::string& name, double hours_worked)
    : name_(name), hours_worked_(hours_worked) {}

  const std::string& Name() const {
    return name_;
  }
```

```
  void LogHoursWorked(double hours) {
    if (hours > 0) {
      hours_worked_ += hours;
    }
  }

  double HoursWorked() const {
    return hours_worked_;
  }

  void Display() const {
    std::cout << name_ << "(" << hours_worked_ << ")\n";
  }

  double Salary() {
    return RegularHoursWorked() * 15 + OvertimeHoursWorked() * 17;
  }

 protected:
  double OvertimeHoursWorked() {
    double overtime = hours_worked_ - 8;
    if (overtime < 0) {
      overtime = 0;
    }
    return overtime;
  }

  double RegularHoursWorked() {
    return hours_worked_ - OvertimeHoursWorked();
  }

 private:
  std::string name_;
  double hours_worked_;
};

class VolunteerManager : public Volunteer {
 public:
  VolunteerManager() : Volunteer() {}
  VolunteerManager(const std::string& name, double hours_worked)
    : Volunteer(name, hours_worked) {}

  void AddVolunteer(const Volunteer& volunteer) {
    team_.push_back(volunteer);
  }

  int TeamCount() const { return team_.size(); }
```

```
  void Display() const {
    Volunteer::Display();
    if (team_.size() == 0) {
      std::cout << "No team members.\n";
    } else {
      std::cout << "Team members:\n-----------\n";
      for (Volunteer v : team_) {
        std::cout << v.Name() << "(" << v.HoursWorked() << ")\n";
      }
      std::cout << "-----------\n";
    }
  }

  double Salary() {
    return RegularHoursWorked() * 17 + OvertimeHoursWorked() * 20;
  }

 private:
  std::vector<Volunteer> team_;
};
```

```
// revised main.cc
#include <iostream>
#include "volunteer.h"
int main() {
 Volunteer p("Paul", 8.0);
 Volunteer j("JC", 10.0);
 VolunteerManager ch("Chang Hyun", 9.0);
 ch.AddVolunteer(p);
 ch.AddVolunteer(j);

 std::cout << "Paul salary: $" << p.Salary() << "\n";
 std::cout << "JC salary: $" << j.Salary() << "\n";
 std::cout << "Chang Hyun salary: $" << ch.Salary() << "\n";

 // The std::cout statement will produce an error:
 // 'OvertimeHoursWorked' is a protected member of 'Volunteer'
 std::cout << "Overtime hours: " << p.OvertimeHoursWorked() <<
"\n";

 return 0;
```

```
}
```

**Screen output:**
Paul salary: $120
JC salary: $154
Chang Hyun salary: $156

16. Member functions and member variables under the protected section can be accessed by member functions of the class and any of its derived classes. However, they are not accessible to any external function or class. Review the Volunteer class' Salary member function. Why is it valid to call RegularHoursWorked? Place a check (✓) beside your answer.

    a. Protected member functions can be accessed by the class' member functions. ✓
    b. Protected member functions can be accessed by derived class' member functions.
    c. Protected member functions cannot be accessed by external functions or classes.

17. Review the VolunteerManager class' Salary member function. Why is it valid to call OvertimeHoursWorked? Place a check (✓) beside your answer.

    a. Protected member functions can be accessed by the class' member functions.
    b. Protected member functions can be accessed by derived class' member functions. ✓
    c. Protected member functions cannot be accessed by external functions or classes.

18. Review the main function. Why is it invalid to call OvertimeHoursWorked on the Volunteer object p? select answer  Place a check (✓) beside your answer.

    a. Protected member functions can be accessed by the class' member functions.
    b. Protected member functions can be accessed by derived class' member functions.
    c. Protected member functions cannot be accessed by external functions or classes. ✓

The Google C++ style guide suggests we only use protected for member functions that are used in derived classes.

Member variables should remain private. Derived classes can use accessors and mutators to manage them.

# Reflector questions

1. What was the most useful thing your team learned during this session?

2. What concept about inheritance did you find most confusing? What helped you understand the concept?

3. What previous topic discussed in class helped you understand inheritance?