
Creative Software Programming

8 – Inheritance, Const & Class

Yoonsang Lee

Fall 2019

Midterm Exam Results

- Average: 68.80
- The highest score: 99 (1 students)
- If you have any questions about your score, ask the TA in the next lab session.
- Students who scored below 43 should study harder to avoid F!

Questions from Last Lecture

- As i know, queue and deque are both array-based, but why the random access is possible only in deque?

| | queue | deque |
|-------------------|---|---|
| Random access | - | <code>operator[]</code> <code>at()</code> |
| Sequential access | <code>front()</code> <code>back()</code> | <code>front()</code> <code>back()</code> |
| Iterators | - | <code>begin(), end()</code> <code>rbegin(), rend()</code> |
| Adding elements | <code>push()</code> | <code>push_front()</code> <code>push_back()</code> <code>insert()</code> |
| Deleting elements | <code>pop()</code> | <code>pop_front()</code> <code>pop_back()</code> <code>erase()</code> <code>clear()</code> |

- Because `std::queue` just provides interface of 'queue' data structure (FIFO behavior)
- 'queue and deque are both array-based' is incorrect.
- <https://en.cppreference.com/w/cpp/container/queue>

Today's Topics

- Inheritance
- Overriding
- Constructor, Destructor & Inheritance
- Multiple Inheritance
- const & Class

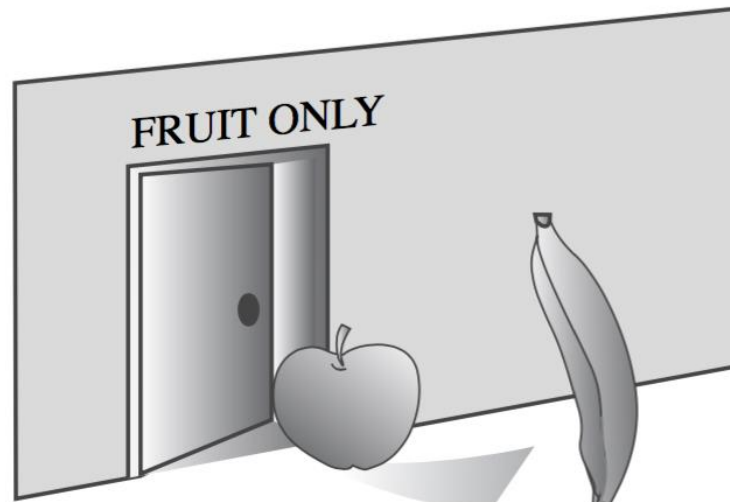
Fundamental Principle of Object Oriented Programming

- Encapsulation (already covered in 6-Class)
 - Binding the data with the code that manipulates it into a single unit (class) & hiding details of the unit (data hiding).
- Inheritance (Today's topic)
 - When a class inherits another class, it has the same behavior or characteristics of another.
- Polymorphism (Next lecture)
 - The ability to create a variable, a function, or an object that has more than one form.

Inheritance

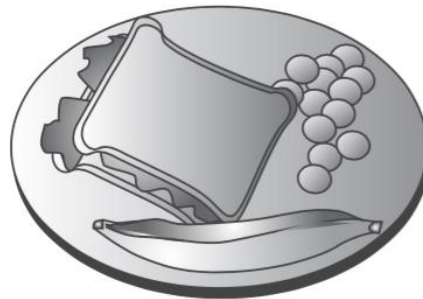
- Build a class on top of existing classes.
 - Minimize re-implementing similar functionalities.
 - Establish relations between classes/types.
 - Customized functionalities.

Inheritance: Is-a Relationship



Inheritance: is-a relation

A banana *is a* fruit,
but
a lunch *has a* banana.



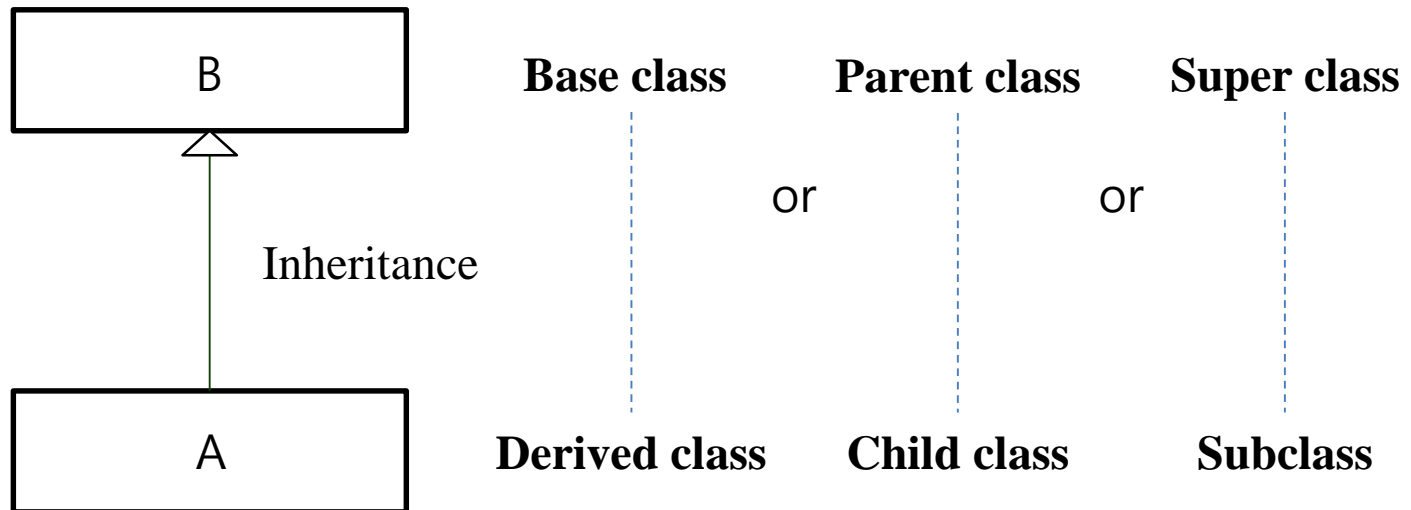
Composition: has-a relation

Inheritance: Is-a Relationship

- "Is-a" relationship: use (public) inheritance when A is a B.
 - A car is a vehicle.
A truck is a vehicle.
A cart is a vehicle.
...
 - A student is a person.
A professor is a person.
...
 - A person is an animal.
A dog is an animal.
...

Inheritance

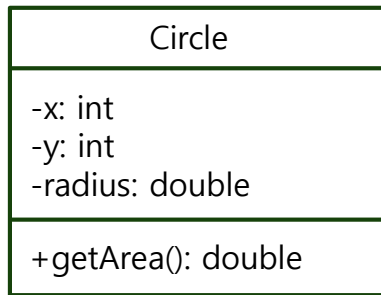
- If a class A inherits another class B,
 - Class A implicitly "has" the member variables and functions of class B.
 - Class A can have additional member variables and functions.



(by UML)

UML Class Diagram Example

Unified Modeling Language (UML): for visualize the design of a software system.



+: public

-: private

#: protected

variable: data type

method(parameter): return type

```
#include <iostream>
using namespace std;

class Circle {
private:
    int x, y;
    double radius;
public:
    Circle(int px, int py, double pradius) {
        x=px, y=py, radius=pradius;}
    double getArea() { return 3.14*radius*radius; }
};

int main()
{
    Circle c(2,3,4);
    cout << c.getArea() << endl;
}
```

An Inheritance Example

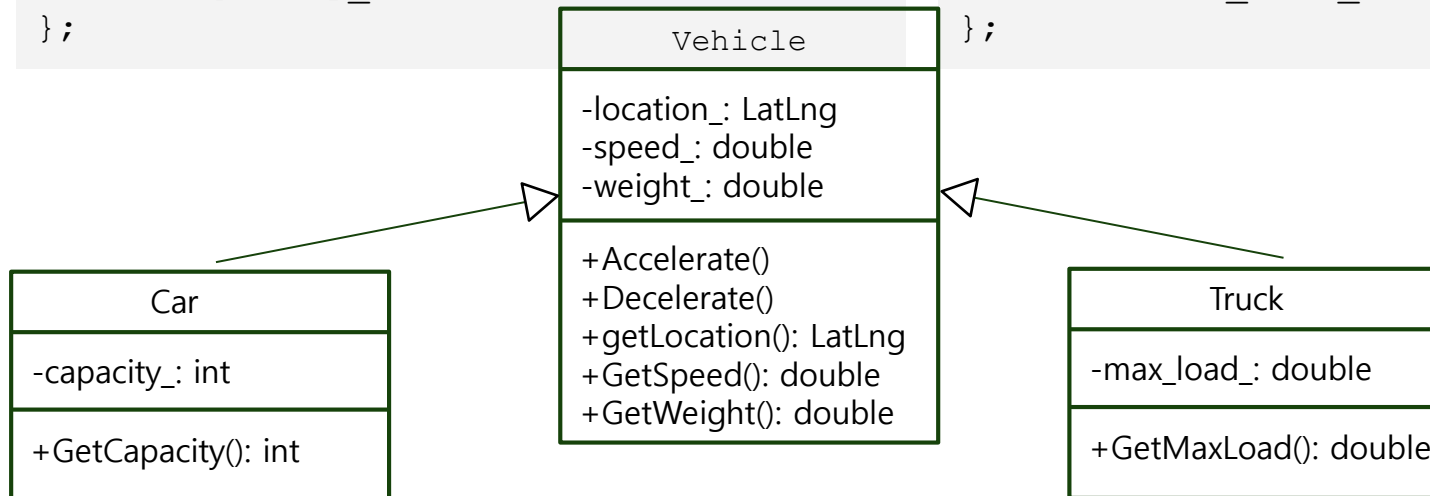
```
class Car {  
    public:  
        Car() {}  
        void Accelerate();  
        void Decelerate();  
        LatLng GetLocation();  
        double GetSpeed();  
        double GetWeight();  
        int GetCapacity();  
    private:  
        LatLng location_;  
        double speed_;  
        double weight_;  
        int capacity_;  
};
```

```
class Truck {  
    public:  
        Truck() {}  
        void Accelerate();  
        void Decelerate();  
        LatLng GetLocation();  
        double GetSpeed();  
        double GetWeight();  
        double GetMaxLoad();  
    private:  
        LatLng location_;  
        double speed_;  
        double weight_;  
        double max_load_;  
};
```

An Inheritance Example

```
class Car {  
public:  
    Car() {}  
    void Accelerate();  
    void Decelerate();  
    LatLng GetLocation();  
    double GetSpeed();  
    double GetWeight();  
    int GetCapacity();  
private:  
    LatLng location_;  
    double speed_;  
    double weight_;  
    int capacity_;  
};
```

```
class Truck {  
public:  
    Truck() {}  
    void Accelerate();  
    void Decelerate();  
    LatLng GetLocation();  
    double GetSpeed();  
    double GetWeight();  
    double GetMaxLoad();  
private:  
    LatLng location_;  
    double speed_;  
    double weight_;  
    double max_load_;  
};
```



An Inheritance Example

```
// Vehicle class.
```

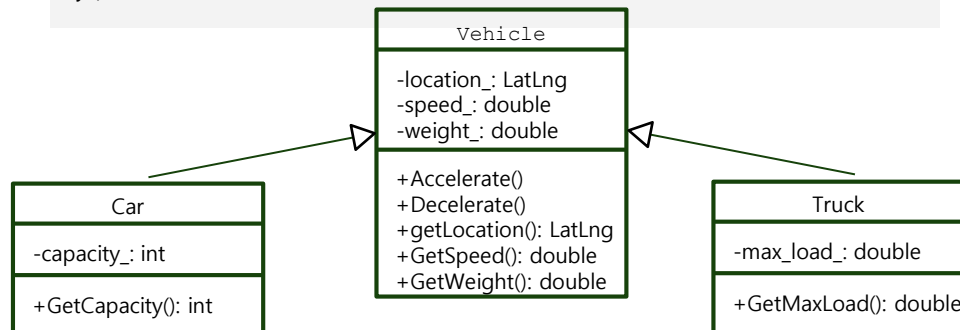
```
class Vehicle {  
public:  
    Vehicle() {}  
    void Accelerate();  
    void Decelerate();  
  
    LatLng GetLocation();  
    double GetSpeed();  
    double GetWeight();  
  
private:  
    LatLng location_;  
    double speed_;  
    double weight_;  
};
```

```
// Car class.
```

```
class Car : public Vehicle {  
public:  
    Car() : Vehicle() {}  
  
    int GetCapacity();  
  
private:  
    int capacity_;  
};
```

```
// Truck class.
```

```
class Truck : public Vehicle {  
public:  
    Truck() : Vehicle() {}  
  
    double GetMaxLoad();  
  
private:  
    double max_load_;  
};
```



An Inheritance Example

```
// Vehicle class.
```

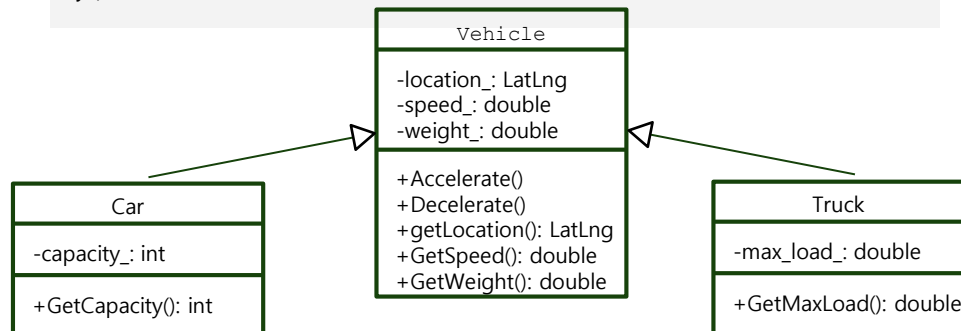
```
class Vehicle {  
public:  
    Vehicle() {}  
    void Accelerate();  
    void Decelerate();  
  
    LatLng GetLocation();  
    double GetSpeed();  
    double GetWeight();  
  
private:  
    LatLng location_;  
    double speed_;  
    double weight_;  
};
```

```
// Car class.
```

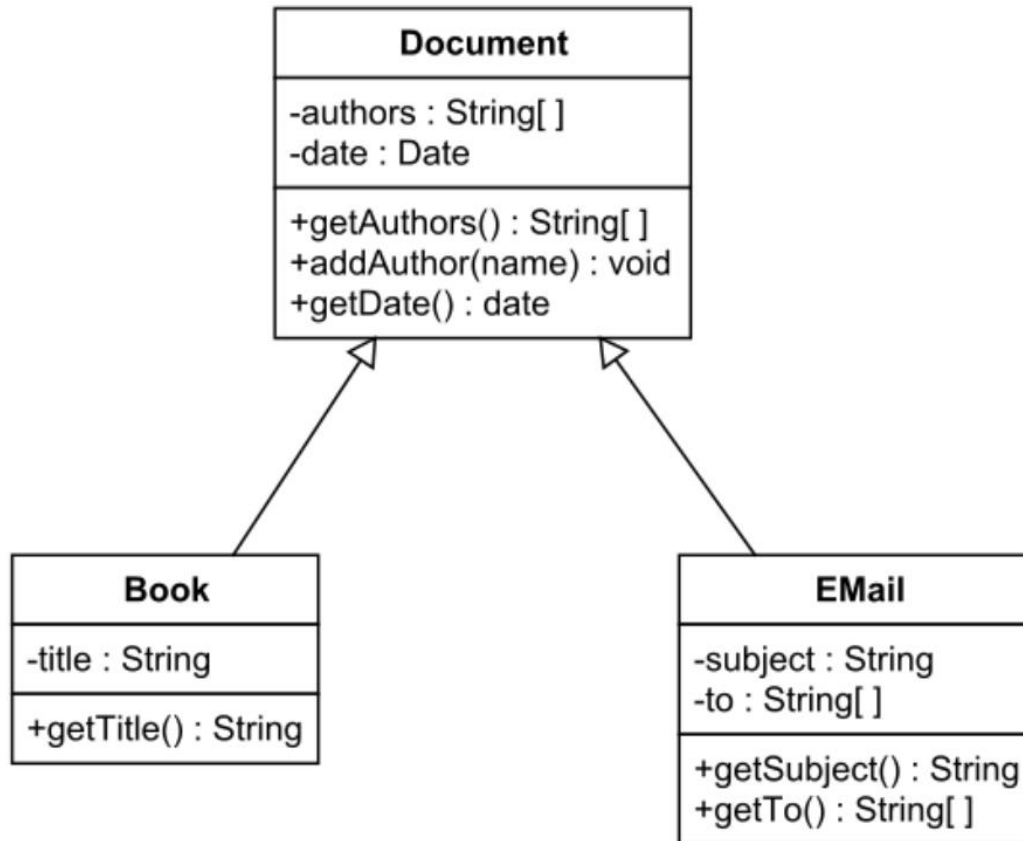
```
class Car : public Vehicle {  
public:  
    Car() : Vehicle() {}  
  
    int GetCapacity();  
  
private:  
    int capacity_;  
};
```

```
// Main routine.
```

```
int main() {  
    Car car;  
    cout << car.GetCapacity() << endl;  
    cout << car.GetSpeed() << endl;  
    cout << car.GetWeight() << endl;  
    return 0;  
}
```



Another inheritance example



Quiz #1

- Go to <https://www.slido.com/>
- Join #csp-hyu
- Click "Polls"

- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as "attendance".

Overriding vs. Overloading

- Function overloading (함수 중복정의)

- provides **multiple definitions of function by changing signatures** (i.e. changing the number, order, or data type of parameters but leaving the function name the same)
- can be used without inheritance, in the same scope

```
int Print(int a) { ... }  
int Print(int a, int b) { ... }
```

- Function overriding (함수 재정의)

- **Redefinition of base class function** in the derived class with same signatures

Overriding Member Function

- You can override a member function to provide a custom functionality of the derived class.
- **Redefine** a member function with the same name as the inherited function.
 - All ancestor's member functions with the same name will be occluded.
 - To access the ancestor's member functions, use `Ancestor::MemberFunction()`.

An example of overriding

```
// Vehicle class.

class Vehicle {
public:
    Vehicle() {}
    void Accelerate();
    void Decelerate();

    LatLng GetLocation();
    double GetSpeed();
    double GetWeight();

private:
    LatLng location_;
    double speed_;
    double weight_;
};
```

```
// Car class.
class Car : public Vehicle {
public:
    Car() : Vehicle() {}

    int GetCapacity();

    // Override the parent's GetWeight().
    double GetWeight() {
        return Vehicle::GetWeight() + passenger_weight_;
    }

private:
    int capacity_;
    double passenger_weight_;
};
```

```
// Main routine.

int main() {
    Car car;
    cout << car.GetCapacity() << endl;
    cout << car.GetSpeed() << endl;
    cout << car.GetWeight() << endl;
    return 0;
}
```

An example of overriding

```
// Vehicle class.

class Vehicle {
public:
    Vehicle() {}
    void Accelerate();
    void Decelerate();

    LatLng GetLocation();
    double GetSpeed();
    double GetWeight();

private:
    LatLng location_;
    double speed_;
    double weight_;
};
```

```
// Car class.
class Car : public Vehicle {
public:
    Car() : Vehicle() {}

    int GetCapacity();

    // Override the parent's GetWeight().
    double GetWeight() {
        return Vehicle::GetWeight() + passenger_weight_;
    }
private:
    int capacity_;
    double passenger_weight_;
};
```

```
// Main routine.

int main() {
    Car car;
    cout << car.GetCapacity() << endl;
    cout << car.GetSpeed() << endl;
    cout << car.GetWeight() << endl;
    return 0;
}
```

An example of overriding

```
// Vehicle class.

class Vehicle {
public:
    Vehicle() {}
    void Accelerate();
    void Decelerate();

    LatLng GetLocation();
    double GetSpeed();
    double GetWeight();

protected:
    LatLng location_;
    double speed_;
    double weight_;
};
```

public: everyone can access.
private: only its member functions can access.
protected: its member functions and the member functions of descendant classes can access.

```
// Car class.
class Car : public Vehicle {
public:
    Car() : Vehicle() {}

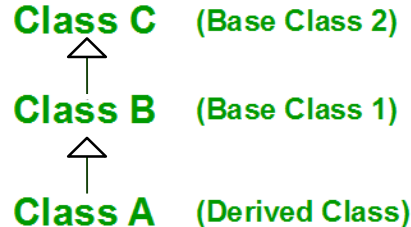
    int GetCapacity();
    // Override the parent's GetWeight().
    double GetWeight() {
        return weight_ + passenger_weight_;
    }
private:
    int capacity_;
    double passenger_weight_;
};
```

```
// Main routine.
int main() {
    Car car;
    cout << car.GetCapacity() << endl;
    cout << car.GetSpeed() << endl;
    cout << car.GetWeight() << endl;
    return 0;
}
```

Constructor, Destructor & Inheritance

- Constructor and destructor call order:
 - Constructors are called from base class to derived class.
 - Destructor are call in reverse order.

Order of Inheritance



Order of Constructor Call

1. **C()** (Class C's Constructor)
2. **B()** (Class B's Constructor)
3. **A()** (Class A's Constructor)

Order of Destructor Call

1. **~A()** (Class A's Destructor)
2. **~B()** (Class B's Destructor)
3. **~C()** (Class C's Destructor)

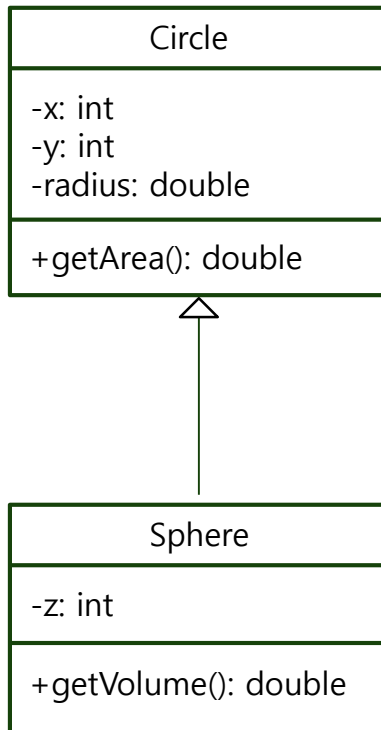
Constructor, Destructor & Inheritance: Example 1

```
class Parent {  
    public:  
    Parent() { cout << " Parent"; }  
    ~Parent() { cout << " ~Parent"; }  
};  
  
class Child : public Parent {  
    public:  
    Child() { cout << " Child"; }  
    ~Child() { cout << " ~Child"; }  
};  
  
class Test : public Child {  
    public:  
    Test() { cout << " Test"; }  
    ~Test() { cout << " ~Test"; }  
};
```

```
int main() {  
    {  
        Child child;  
        cout << endl;  
    }  
    cout << endl;  
    {  
        Test test;  
        cout << endl;  
    }  
    cout << endl;  
    return 0;  
}
```

```
Parent Child  
~Child ~Parent  
Parent Child Test  
~Test ~Child ~Parent
```

Constructor, Destructor & Inheritance: Example 2



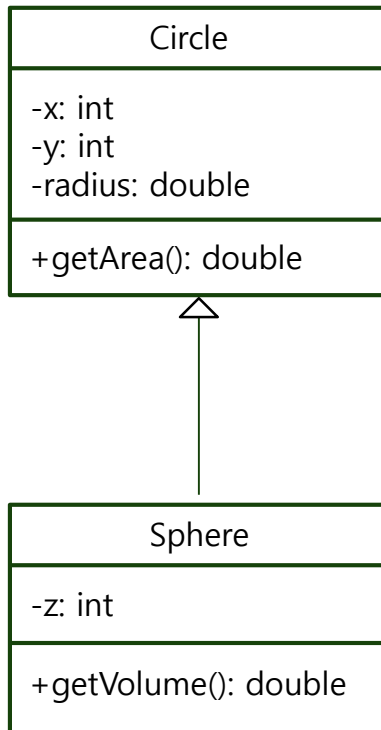
```
#include <iostream>
using namespace std;

class Circle {
private:
    int x, y;
    double radius;
public:
    Circle(int px, int py, double pradius) {
        x=px; y=py; radius=pradius;}
    double getArea() { return 3.14*radius*radius; }
};

class Sphere: public Circle{
private:
    int z;
public:
    Sphere(int px, int py, double pradius, int pz){
        cout << "Sphere" << endl;
        x=px; y=py; radius=pradius; z=pz;}
    double getVolumn(){
        return 4.0/3*3.14*radius*radius*radius;
    }
};

int main()
{
    Circle c(2,3,4.0);
    cout << c.getArea() << endl;
    Sphere s(2,3,4.0,5);
    cout << s.getVolumn() << endl;
    return 0;
}
```


Constructor, Destructor & Inheritance: Example 2



```
#include <iostream>
using namespace std;

class Circle {
private:
    int x, y;
    double radius;
public:
    Circle(int px, int py, double pradius) {
        x=px; y=py; radius=pradius;}
    double getArea() { return 3.14*radius*radius; }
};

class Sphere: public Circle{
private:
    int z;
public:
    Sphere(int px, int py, double pradius, int pz){
        cout << "Sphere" << endl;
        x=px; y=py; radius=pradius; z=pz;}
    double getVolumn(){
        return 4.0/3*3.14*radius*radius*radius;
    }
};
```

implicitly calls Circle's default constructor
which is not defined

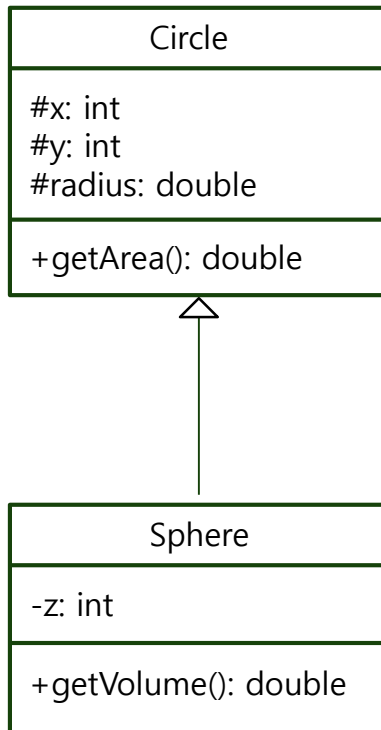
```
8_10.cc:18:5: error: constructor for 'Sphere' must explicitly initialize the
      base class 'Circle' which does not have a default constructor
      Sphere(int px, int py, double pradius, int pz){
      ^
```

```
8_10.cc:4:7: note: 'Circle' declared here
class Circle {
  ^
```

```
8_10.cc:20:9: error: 'x' is a private member of 'Circle'
      x=px; y=py; radius=pradius; z=pz;}
      ^
```

```
8_10.cc:6:9: note: declared private here
      int x, y;
      ^
```

Constructor, Destructor & Inheritance: Example 2



```
#include <iostream>
using namespace std;

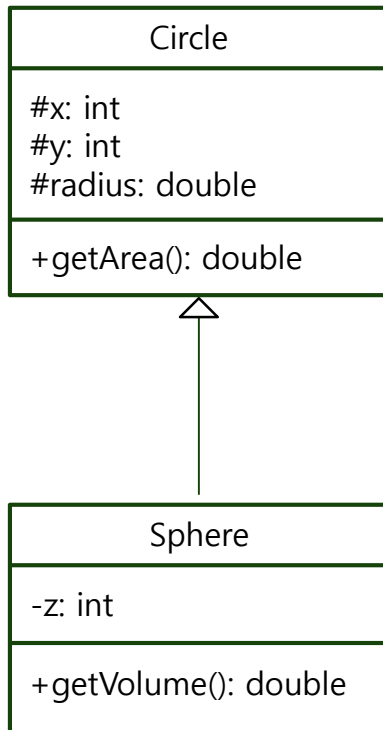
class Circle {
protected:
    int x, y;
    double radius;
public:
    Circle(){ cout << "Circle: no parameter" << endl; }
    Circle(int px, int py, double pradius) {
        cout << "Circle: with parameters" << endl;
        x=px; y=py; radius=pradius;}
    double getArea() { return 3.14*radius*radius; }
};

class Sphere: public Circle{
private:
    int z;
public:
    Sphere(int px, int py, double pradius, int pz){
        cout << "Sphere" << endl;
        x=px; y=py; radius=pradius; z=pz;}
    double getVolumn(){
        return 4.0/3*3.14*radius*radius*radius;
    }
};

int main()
{
    Circle c(2,3,4.0);
    cout << c.getArea() << endl;
    Sphere s(2,3,4.0,5);
    cout << s.getVolumn() << endl;
    return 0;
}
```

```
Circle: with parameters
50.24
Circle: no parameter
Sphere
267.947
```

Constructor, Destruct



```
#include <iostream>
using namespace std;

class Circle {
protected:
    int x, y;
    double radius;
public:

    Circle(int px, int py, double pradius) {
        cout << "Circle: with parameters" << endl;
        x=px; y=py; radius=pradius;}
    double getArea() { return 3.14*radius*radius; }
};

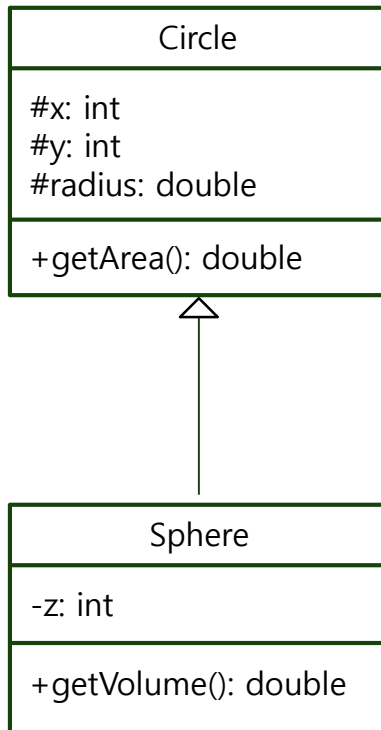
class Sphere: public Circle{
private:
    int z;
public:
    //Sphere(int px, int py, double pradius, int pz){
    // cout << "Sphere" << endl;
    //     x=px; y=py; radius=pradius; z=pz;}
    Sphere(int px, int py, double pradius, int pz):
        Circle(px, py, pradius), z(pz){
        cout << "Sphere" << endl;
    }
    double getVolumn(){
        return 4.0/3*3.14*radius*radius*radius;
    }
};

int main()
{
    Circle c(2,3,4.0);
    cout << c.getArea() << endl;
    Sphere s(2,3,4.0,5);
    cout << s.getVolumn() << endl;
    return 0;
}
```

explicitly calls Circle's constructor

```
Circle: with parameters
50.24
Circle: with parameters
Sphere
267.947
```

Constructor, Dest



```
#include <iostream>
using namespace std;

class Circle {
protected:
    int x, y;
    double radius;
public:
    //Circle(){ cout << "Circle: no parameter" << endl; }
    //Circle(int px, int py, double pradius) {
    //    cout << "Circle: with parameters" << endl;
    //    x=px; y=py; radius=pradius;}
    double getArea() { return 3.14*radius*radius; }
};

class Sphere: public Circle{
private:
    int z;
public:
    Sphere(int px, int py, double pradius, int pz){
        cout << "Sphere" << endl;
        x=px; y=py; radius=pradius; z=pz;}
    //Sphere(int px, int py, double pradius, int pz):
    //    Circle(px, py, pradius), z(pz){
    //    cout << "Sphere" << endl;
    // }
    double getVolumn(){
        return 4.0/3*3.14*radius*radius*radius;
    }
};

int main()
{
    //Circle c(2,3,4.0);
    // cout << c.getArea() << endl;
    Sphere s(2,3,4.0,5);
    cout << s.getVolumn() << endl;
    return 0;
}
```

Sphere
267.947

Quiz #2

- Go to <https://www.slido.com/>
- Join #csp-hyu
- Click "Polls"

- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as "attendance".

Person Example - outline

```
// Person class.

class Person {
public:
    Person(const string& name);

    const string& name();
    const string& address();
    void ChangeAddress(const string& addr);
};

// Student class.

class Student : public Person {
public:
    Student(const string& name);

    void RegisterClass(int class_id);
    int GetNumClasses();
    int ComputeTuition();
};
```

```
// Employee class

class Employee : public Person {
public:
    Employee(const string& name, int salary);

    int salary();
    int ComputeIncomeTax();
    void SetSalary(int salary);
};

// Faculty class

class Faculty : public Employee {
public:
    Faculty(const string& name, int salary);

    void TeachClass(int class_id);
};
```

Person Example - implementation

person.h

```
#ifndef _PERSON_H_
#define _PERSON_H_

#include <string>

class Person {
public:
    Person(const std::string& name)
        : name_(name) {}

    const std::string& name() {
        return name_;
    }
    const std::string& address() {
        return address_;
    }

    void ChangeAddress(const std::string& addr) {
        address_ = addr;
    }

private:
    std::string name_, address_;
};

#endif
```

student.h

```
#ifndef _STUDENT_H_
#define _STUDENT_H_

#include <set>
#include "person.h"

class Student : public Person {
public:
    Student(const std::string& name)
        : Person(name) {}

    void RegisterClass(int class_id) {
        registered_classes_.insert(class_id);
    }

    int GetNumClasses() {
        return registered_classes_.size();
    }

    int ComputeTuition() {
        return registered_classes_.size() * 100
            + 500;
    }

private:
    std::set<int> registered_classes_;
};

#endif
```

Person Example - implementation

main.cc

```
#include "employee.h"
#include "faculty.h"
#include "student.h"
using namespace std;

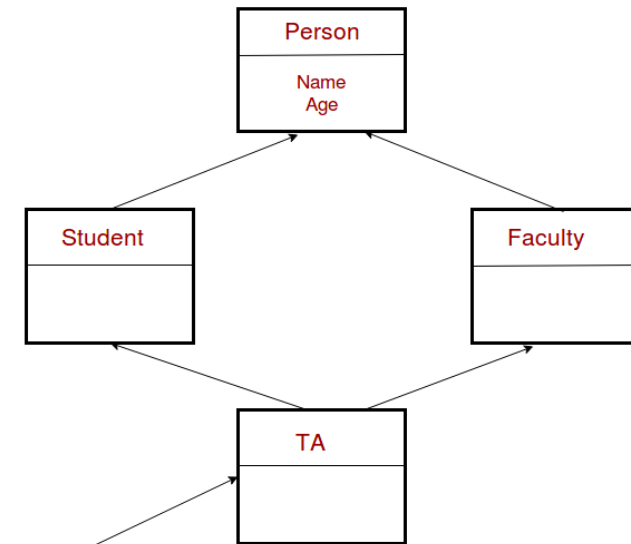
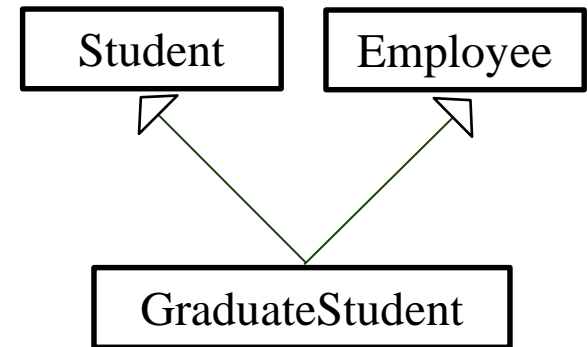
int main() {
    Student john("John"), david("David");
    Employee susan("Susan", 200);
    Faculty daniel("Daniel", 100);

    john.ChangeAddress("New York");
    david.RegisterClass(101);
    daniel.TeachClass(101);
    daniel.TeachClass(102);

    return 0;
}
```


Multiple Inheritance

- Inheriting from two or more base classes.
 - The derived class has all the members of base classes
- Issues
 - Ambiguity
 - What happens if base classes has same-named members?
 - The diamond problem
 - What happens if parent classes are derived from the same grandparent class?



Name and Age needed only once

Multiple Inheritance: Example

```
class Person {
    public:
    // ...
};
class Student : public Person {
    public:
    // ...
};
class Employee : public Person {
    public:
    // ...
};

// Multiple inheritance example.
class GraduateStudent
    : public Student, public Employee
{
    public:
    GraduateStudent(const string& name,
                    int salary)
        : Student(name),
          Employee(name + "*", salary) {
    }
};
```

```
int main() {
    GraduateStudent mark("Mark", 50);

    cout << mark.GetNumClasses() << endl;
    cout << mark.salary() << endl;
    return 0;
}
```

Multiple Inheritance: Example

```
class Person {
    public:
    // ...
};
class Student : public Person {
    public:
    // ...
    void DoSomething();
};
class Employee : public Person {
    public:
    // ...
    void DoSomething();
};
// Multiple inheritance example.
class GraduateStudent
    : public Student, public Employee {
public:
    GraduateStudent(const string& name,
                    int salary)
        : Student(name),
          Employee(name + "*", salary) {}
};
```

```
int main() {
    GraduateStudent mark("Mark", 50);

    // Error - ambiguous function DoSomething
    mark.DoSomething();

    return 0;
}
```

Multiple Inheritance

- Actually, you can avoid these problem by using `virtual inheritance` in C++.
- General advice: Avoid using multiple inheritance as much as possible.
 - It is commonly believed that multiple inheritance tends to mass things up.
 - That's why Java forbids multiple inheritance.
- Note that multiple inheritance from *interfaces* (pure abstract classes in C++) can be very helpful.
 - Java only allows multiple inheritance from *interfaces* (“implements” multiple interfaces in Java)

const: review

- Const variables

```
const int MAX = 100;
```

- Const parameters

```
int sum(const int x, const int y) { . . . }
```

- Pointer to const and const pointer

```
const int *pNum = &num
```

```
int* const p = &a;
```

const & Class

- Const member variables
 - should be initialized in *member initializer list* of constructor
- **Const member functions**
 - Can read the value of member variables
 - Cannot change the value of member variables
- Const object
 - Cannot change the value of member variables on a const object
 - **Cannot call non-const member functions on a const object**

const: member variables

```
#include <iostream>
using namespace std;

class Circle {
private:
    double Radius;
    const double PI;
public:
    Circle(double r=0, double p){Radius = r; PI=p;}
    void SetRadius(double r) { Radius = r;}
    double GetArea() const { return PI*Radius*Radius;}
};

int main(){
    Circle cir(2,4);
    cout << cir.GetArea() << endl;
    return 0;
}
```

???

const: member variables

```
#include <iostream>
using namespace std;

class Circle {
private:
    double Radius;
    const double PI;
public:
    //Circle(double r=0, double p){Radius = r; PI=p;}
    Circle(double r, double p): Radius(r), PI(p){}
    void SetRadius(double r) { Radius = r;}
    double GetArea() const { return PI*Radius*Radius;}
};

int main(){
    Circle cir(2,4);
    cout << cir.GetArea() << endl;
    return 0;
}
```

- Const member variables
 - should be initialized in *member initializer list* of constructor

const: member function

```
#include <iostream>
using namespace std;

class Circle {
private:
    double Radius;
    const double PI;
public:
    //Circle(double r=0, double p){Radius = r; PI=p;}
    Circle(double r, double p): Radius(r), PI(p){}
    void SetRadius(double r) const { Radius = r;}
    double GetArea() const { return PI*Radius*Radius;}
};

int main(){
    Circle cir(2,4);
    cir.SetRadius(5.0);
    cout << cir.GetArea() << endl;
    return 0;
}
```

???

const: member function

```
#include <iostream>
using namespace std;

class Circle {
private:
    double Radius;
    const double PI;
public:
    //Circle(double r=0, double p){Radius = r; PI=p;}
    Circle(double r, double p): Radius(r), PI(p){}
    void SetRadius(double r) { Radius = r;}
    double GetArea() const { return PI*Radius*Radius;}
};

int main(){
    Circle cir(2,4);
    cir.SetRadius(5.0);
    cout << cir.GetArea() << endl;
    return 0;
}
```

- Const member functions
 - Can read member variables, cannot update member variables

const: object

```
#include <iostream>
using namespace std;

class Circle {
private:
    double Radius;
    const double PI;
public:
    Circle(double r = 0): Radius(r), PI(3.14){ }
    void SetRadius(double r) {Radius = r;}
    double GetArea() const { return (PI*Radius*Radius);}
};

int main()
{
    Circle cir(2);
    cout << cir.GetArea() << endl;

    const Circle cir2(3);
    cout << cir2.GetArea() << endl;
    //cir2.SetRadius(5);    //compile error

    return 0;
}
```

- Const object
 - Cannot update member variables
 - **Cannot call non-const member functions**

Quiz #3

- Go to <https://www.slido.com/>
- Join #csp-hyu
- Click "Polls"

- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked as "attendance".

Class Inheritance Types

- Types of inheritance: `public`, `protected`, and `private`.
 - Depending on the inheritance types, the parent's member has different access control IN the child class.
 - Most commonly used is **public inheritance** (and probably it's the only useful inheritance).

| Type of inheritance | Parent's public member | Parent's protected member | Parent's private member |
|------------------------|------------------------|---------------------------|-------------------------|
| <code>public</code> | <code>public</code> | <code>protected</code> | x (not accessible) |
| <code>protected</code> | <code>protected</code> | <code>protected</code> | x (not accessible) |
| <code>private</code> | <code>private</code> | <code>private</code> | x (not accessible) |

Example of Private Inheritance

```
class A {
public:
    void APublic() {}
protected:
    void AProtected() {}
private:
    void APrivate() {}
};

// Private inheritance.
class CA : private A {
public:
    void CAPublic() {
        APublic();    // OK.
        AProtected(); // OK.
        APrivate();   // Error.
    }
    void CAPublic2() {}
protected:
    void CAProtected() {}
private:
    void CAPrivate() {}
};
```

```
class Client : public CA {
    void Function() {
        APublic();    // Error.
        AProtected(); // Error.
        APrivate();   // Error.

        CAPublic();   // Error.
        CAPublic2();  // OK.
        CAProtected(); // OK.
        CAPrivate();  // Error.
    }
};
```

```
// Main routine.

int main() {
    CA ca;
    ca.APublic(); // Error.
    ca.CAPublic(); // Error
    ca.CAPublic2(); // OK.
    ...
}
```

Next Time

- Labs in this week:
 - Lab1: Assignment 8-1
 - Lab2: Assignment 8-2

- Next lecture:
 - 9 - Polymorphism 1