KIE1008: Programming 2

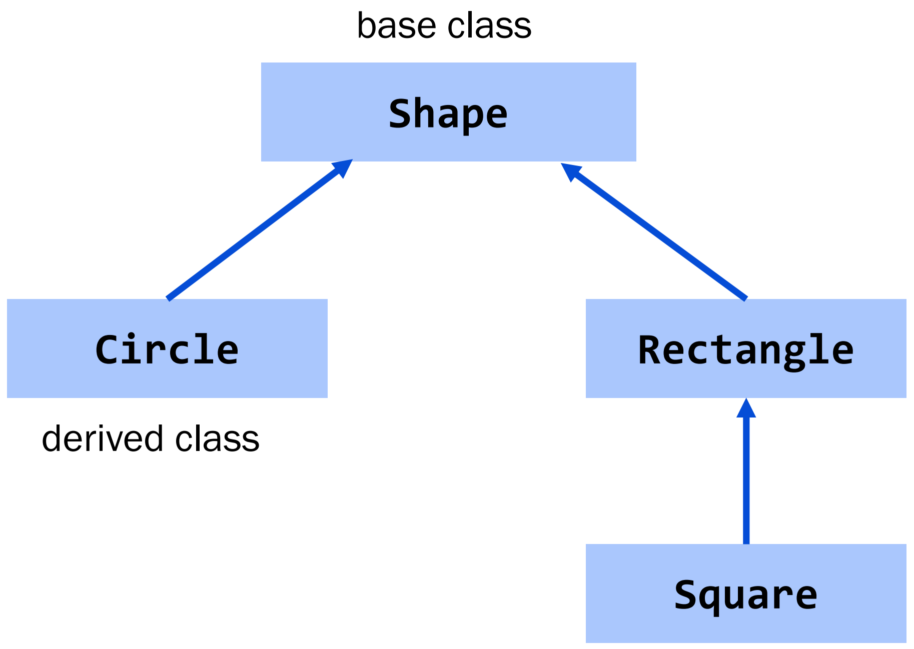
Object-Oriented Programming: Inheritance

Semester 1, 2025/2026

Inheritance

- Inheritance is a form of software reuse in which a class that absorbs an existing class's data and behaviors is created, and enhanced with new capabilities.
- The new class should inherit the members of an existing class.
- This existing class is called the base class (parent) and the new class is referred to as the derived class (child).
- C++ offers `public`, `protected` and `private` inheritance.
- Inheritance relationships form class hierarchies.

Inheritance



- Every object of a derived class is also an object of that derived class’s base class, but not vice versa.

e.g. c1 is an object of Circle, and also an object of Shape; s1 is an object of Shape, but not Circle.

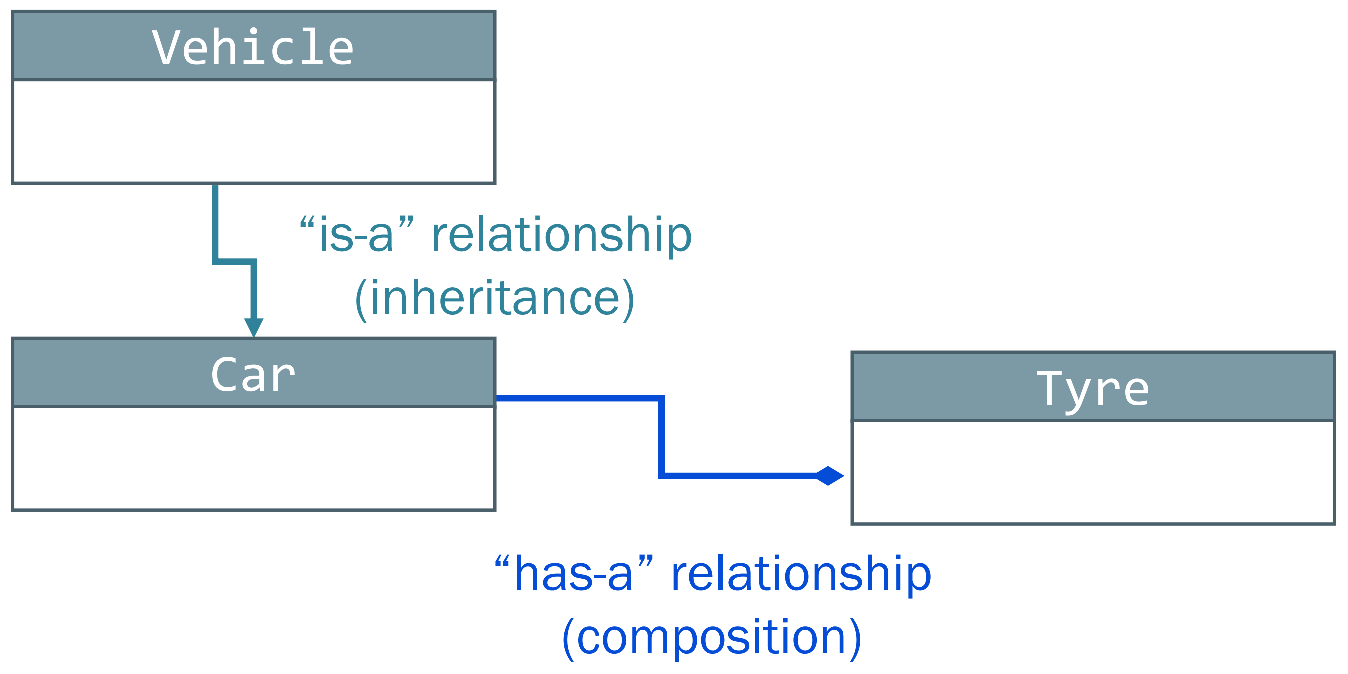
- A member function of a derived class may have the same name as a member function of a base class (polymorphism).

Base Classes and Derived Classes

- Although classes can exist independently, once they're employed in inheritance relationships, they become affiliated with other classes.
- A class becomes either a base class - supplying members to other classes, a derived class - inheriting its members from other classes, or both.

Base Class	Derived Classes
`Student`	`GraduateStudent, UndergradStudent`
`Shape`	`Circle, Triangle, Rectangle, Sphere`
`Loan`	`CarLoan, MortgageLoan`
`Employee`	`Faculty, AdminStaff, TechnicalStaff`

Inheritance vs Composition



Types of Inheritance

Base class member access specifier	**`public`** inheritance	**`protected`** inheritance	**`private`** inheritance
`public`	`public`	`protected`	`private`
`protected`	`protected`	`protected`	`private`
`private`	Not accessible (hidden)	Not accessible (hidden)	Not accessible (hidden)

- Use of `protected` and `private` inheritance is rare
- A base class's `private` members are never accessible directly from a derived class, but can be accessed through calls to the `public` and `protected` members of the base class.

`public` Inheritance

- most-commonly used
- Syntax:

```
class <derivedClass> : public <baseClass>
{
 …
};
```

Example 1: Vehicle and Car Classes

```
// Base class
class Vehicle {
public:
    string brand = "Ford";
    void honk() { cout << "Tuut, tuut! \n" ; }
};

// Derived class
class Car: public Vehicle { // public inheritance
public:
    string model = "Mustang";
};

int main() {
  Car myCar;
  myCar.honk(); // access base class’s member function
  cout << myCar.brand + " " + myCar.model; // access base class’s
                                              public data

}
```

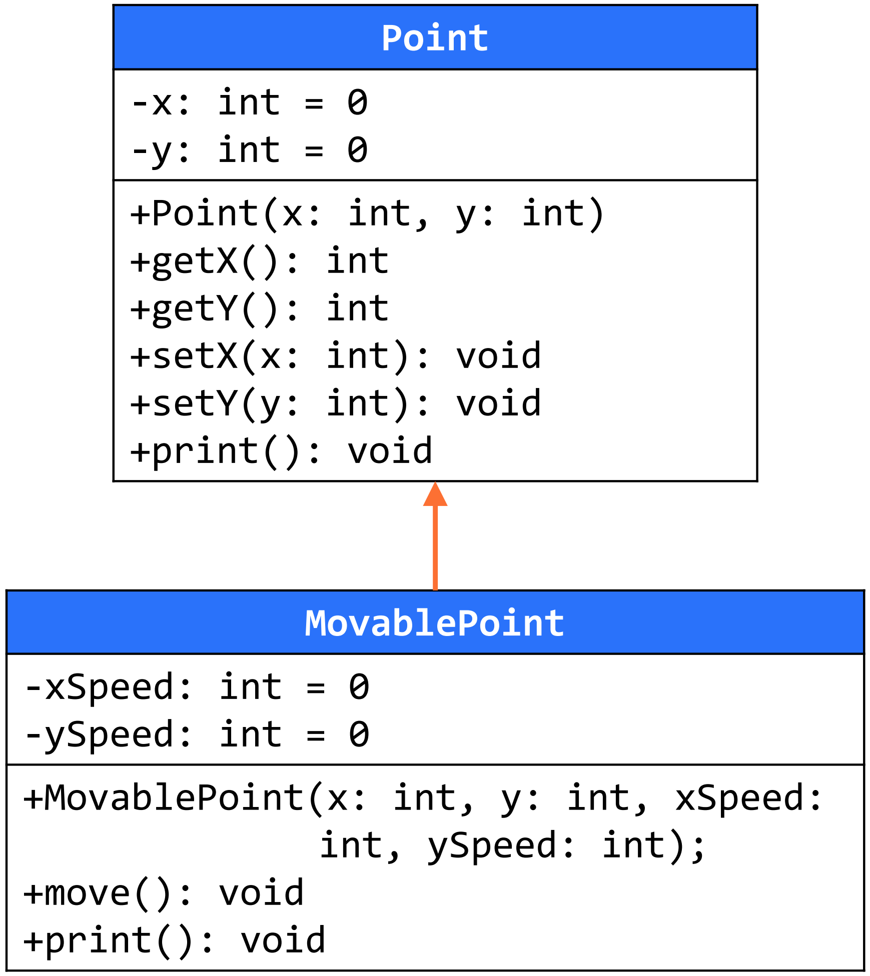
Constructors of Derived and Base Classes

- The constructors of a derived class can (directly) initialize only the `public` data members inherited from the base class of the derived class.
- When a derived class object is declared, it must also trigger the execution of one of the base class's constructors.

<u>Example:</u>

```
boxType::boxType(double l, double w, double h)
: rectangleType(l, w)
{
   height = h;
}
```

Example 2: `Point` **and** `MovablePoint` **Classes**



Example 2: `Point` and `MovablePoint` Classes (1/4)

```
class Point
{
private:
   int x, y;

public:
   Point(int x = 0, int y = 0); // Constructor
   int getX() const;
   int getY() const;
   void setX(int x);
   void setY(int y);
   void print() const;
};
```

Code: W05C01

Example 2:
`Point` and `MovablePoint` Classes (2/4)

```cpp
Point::Point(int x, int y) : x(x), y(y) { }

int Point::getX() const { return x; }
int Point::getY() const { return y; }

void Point::setX(int x) { this->x = x; }
void Point::setY(int y) { this->y = y; }

void Point::print() const
{
   cout << "(" << x << "," << y << ")" << endl;
}
```

Code: W05C01

Example 2:
Point and MovablePoint Classes (3/4)

```
#include "Point.h"

class MovablePoint : public Point
{
private:
   int xSpeed, ySpeed;

public:
   MovablePoint(int x, int y, int xSpeed = 0, int ySpeed = 0);
   void move();
   void print() const; // Member function overriding in inheritance
};
```

Code: W05C01

Example 2:
`Point` and `MovablePoint` Classes (4/4)

```
MovablePoint::MovablePoint(int x, int y, int xSpeed, int ySpeed)
      : Point(x, y), xSpeed(xSpeed), ySpeed(ySpeed) { }

void MovablePoint::print() const {
   cout << "Movable Point = ";
   Point::print(); // Invoke base class function via scope resolution
                      operator
   cout << "Speed = " << "(" << xSpeed << "," << ySpeed << ")";
}

void MovablePoint::move() {
   // Subclass cannot access private member of the superclass directly
   // Need to go thru the public interface
   Point::setX(Point::getX() + xSpeed);
   Point::setY(Point::getY() + ySpeed);
}
```

Code: W05C01

Example 2: `Point` and `MovablePoint` Classes

```cpp
#include "MovablePoint.h" // included "Point.h"

int main()
{
   Point p1(4, 5);  // base class
   p1.print();      // Point @ (4,5)
   cout << endl;

   MovablePoint mp1(11, 22, 33, 44); // derived class
   mp1.print();  // Movable Point=(11,22) Speed=(33,44)
   cout << endl;
   mp1.move();
   mp1.print();  // Movable Point=(44,66) Speed=(33,44)
   cout << endl;
}
```

Code: W05C01

`protected` Data Members

- Recall that a private data member in the base class is not accessible in the derived class.

```
void MovablePoint::move()
{
   x += xSpeed; // error: 'int Point::x' is private
   Point::setX(Point::getX() + xSpeed);
   Point::setY(Point::getY() + ySpeed);
}
```

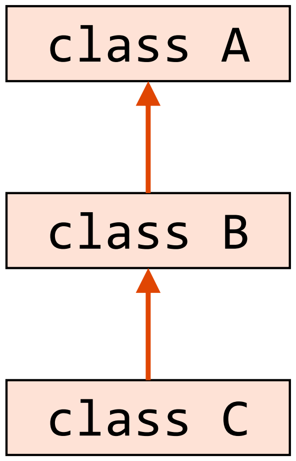
- However, if we make `x` `protected`, the derived class can access `x` directly.

`protected` **Data Members**

```
class Point {
protected:
   int x, y;
......
};

class MovablePoint : public Point {
   ......
}

void MovablePoint::move() {
   x += xSpeed;
   y += ySpeed;
}
```

Multilevel Inheritance

A derived class is created from another derived class.



```
class A  // parent
{
......
};

class B: public A  // child
{
......
};

class C: public B  // grandchild
{
......
};
```

Example: Multilevel Inheritance

```cpp
class A
{
private:
   int a = 10;
public:
   int get_a() { return a; }
   void disp_a()
   {
      cout << "Value of a: " << a << endl;
   }
};
```

```cpp
class B : public A
{
private:
   int b = 20;
public:
   int get_b() { return b; }
   void disp_b()
   {
      disp_a();
      cout << "Value of b: " << b << endl;
   }
};
```

Code: W05C02

Example: Multilevel Inheritance

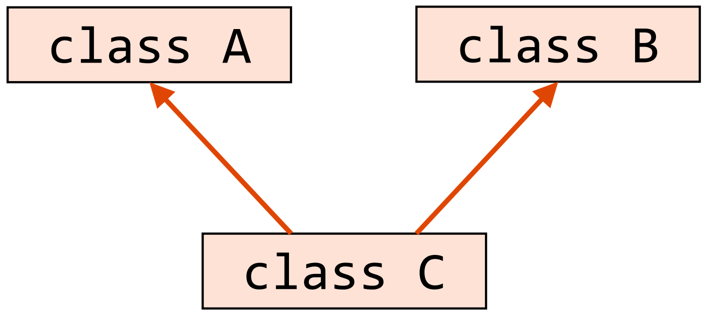
```cpp
class C : public B
{
private:
   int c = 30;
public:
   int sum() { return c + get_a() + get_b(); }
   void disp_c()
   {
      disp_b();
      cout << "Value of c: " << c << endl;
   }
};
```

```cpp
int main()
{
   C objC;

   objC.disp_c();

   cout << "a + b + c = " << objC.sum()
   << endl;

   return 0;
}
```

Code: W05C02

Multiple Inheritance

A class derived from more than one base class, using a comma-separated list.



```
class A
{
......
};

class B
{
......
};

class C: public A, public B
{
......
};
```

Example: Multiple Inheritance

```cpp
class Date {
public:
    Date(int = 1, int = 1, int = 2000);
    void printDate() const;
private:
    int day, month, year;
};

class Time {
public:
    Time(int = 0, int = 0, int = 0);
    void printTime() const;
private:
    int hour, minute, second;
};
```

Code: W05C03

Example: Multiple Inheritance

```cpp
Date::Date(int m, int d, int y) {
   if (m > 0 && m <= 12) // validate the month
      month = m;
   else
      throw invalid_argument("month must be 1-12");

   year = y;
   day = d;
}


void Date::printDate() const {
   cout << month << '/' << day << '/' << year;
}
```

Code: W05C03

Example: Multiple Inheritance

```
Time::Time(int h, int m, int s)
: hour(h), minute(m), second(s) {}

void Time::printTime() const
{
    cout << ((hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
    << setfill('0') << setw(2) << minute << ":" << setw(2) <<
    second << (hour < 12 ? " AM" : " PM" ) << endl;
}
```

Code: W05C03

Example: Multiple Inheritance

```
class DateTime : public Date, public Time {
public:
   DateTime();
   DateTime(int, int, int, int, int, int);
};

DateTime::DateTime() {
    cout << "Date is ";
    printDate();
    cout << "; Time is ";
    printTime();
}
```

Code: W05C03

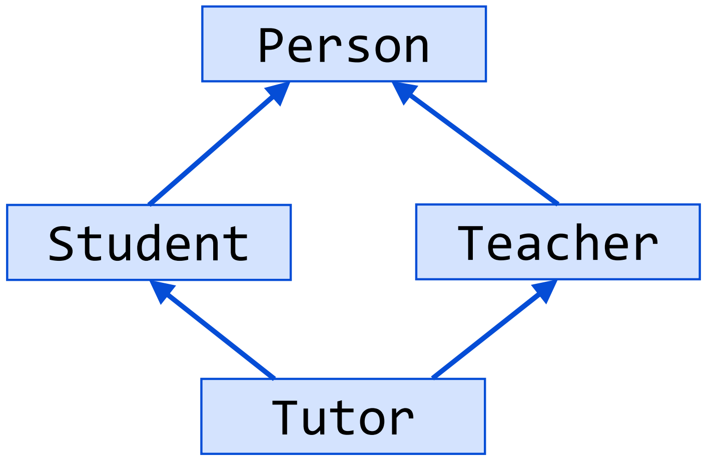
Example: Multiple Inheritance

```
DateTime::DateTime(int dy, int mon, int yr, int hr, int mt, int
sc)
: Date(dy, mon, yr), Time(hr, mt, sc) {
    cout << "Date is ";
    printDate();
    cout << "; Time is ";
    printTime();
}

int main()
{
   DateTime Day1;
   DateTime Day2(31, 10, 2021, 15, 32, 27);
}
```

Code: W05C03

Multiple Inheritance: The diamond problem

An ambiguity arises when two classes B and C inherit from A, and class D inherits from both B and C.



Constructor of `Person` is called two times, so object `Tutor` has two copies of all members of `Person`

If there is a method in `Person` that `Student` and `Teacher` have overridden, and `Tutor` does not override it, then which version of the method does `Tutor` inherit: that of `Student`, or that of `Teacher`?

The solution to this problem is `virtual` keyword.

Example: Virtual Inheritance (1/4)

```cpp
class Person
{
public:
    Person(string n)
    {
        name = n;
    }
protected:
    string name;
};
```

Code: W05C04

Example: Virtual Inheritance (2/4)

```
class Teacher : virtual public Person
{
public:
    Teacher(string n): Person(n) {
        srand(time(0));
        staffID = rand() % 100000;
    }
    void printInfo() const {
        cout << "Name: " << name << "; Staff ID: " << setfill('0')
        << setw(5) << staffID << endl;
    }

protected:
    int staffID;
};
```

Code: W05C04

Example: Virtual Inheritance (3/4)

```
class Student : virtual public Person
{
public:
    Student(string n): Person(n) {
        srand(time(0));
        studentID = rand() % 100000;
    }
    void printInfo() const {
        cout << "Name: " << name << "; Student ID: " <<
        setfill('0') << setw(5) << studentID << endl;
    }

protected:
    int studentID;
};
```

Code: W05C04

Example: Virtual Inheritance (4/4)

```cpp
class Tutor : public Teacher, public Student {
public:
    Tutor(string n): Person(n), Teacher(n), Student(n) { }
    void printInfo() const {
        cout << "Name: " << name << "; Staff ID: " << setfill('0')
        << setw(5) << staffID << "; Student ID: " << setfill('0')
        << setw(5) << studentID << endl;
    }
};

int main() {
    Tutor s1("Ali");
    s1.printInfo();
}
```

Code: W05C04