Lecture #6: Inheritance

“Inheritance is a way to form new classes using classes that have already been defined.”

Inheritance

- rids the need to rewrite/copy all code from first class into second class
- technique that enables us to define a **subclass** to have it **inherit** all of the functions and data of a **superclass**
- subclass cannot access the private variables of the parent class
- `class SubclassName : public SuperclassName`

```
class Robot {
public:
    void setLocation(int x, int y);
    int getX();
    int getY();
private:
    int m_x;
    int m_y;
}

class ShieldedRobot: public Robot {     // inheritance syntax
public:
    // can do everything a Robot can do
    int getShield() {
        return m_shield;
    }
    void setShield (int s) {
        m_shield = s;
    }
private:
    // cannot access private variables of an instance of a Robot
    int m_shield;
}
```

“Is a” vs. “Has a”

- a Student **is a type of** person (plus ID, GPA, etc.)
- a ShieldedRobot **is a type of** Robot (plus a shiled strength, etc.)
- a Student **has a** GPA
- **is a -** warrants inheritance (public inheritance)
- **has a -** warrants a member variable (private inheritance)

Class Heirarchy

- shows a tree of superclasses/subclasses in a system

Inheritance Terminology

- **base class/superclass -** serves as the basis for others classes
- **derived class/sublcass -** class that is derived from a base class

Three Uses of Inheritance

1. reuse
    a. write code once and reuse to avoid duplication
2. extension
    a. add new behaviors or data to derived class
3. specialization
    a. redefine an existing behavior in a subclass

Reuse

- every **public method** in the base class is automatically reused/exposed in the derived class
- only **public members** in the base class are **exposed/reused** in the **derived class(es)**
- **private members** in the **base class** are **hidden** from the **derived class(es)**
- if you want your **derived** class to be able to reuse one+ **private member functions** of the **base** class:
    - change **private** to **protected**
    - `protected: void canBeCalledFromSubClasses();`
    - still prevents rest of your program from seeing/using them
    - but **never ever** make your member variables protected (or public)
        - BREAKS ENCAPSULATION

Specialization/Overriding

- you can **override** or **specialize** existing functions from the base class in your derived class
- if you do this, you should always insert the **virtual** keyword in front of **both** the original and replacement functions
    - don't actually need **virtual** in subclass, but it's good practice
    - `virtual void thisFunctionAppearsInBaseAndSubclass();`
    - Don't include **virtual** keyword in non-inline function definition
- making a function **virtual** makes it slower, so don't make everything **virtual**
-

```
LL::swapLastTwo() {
    Node* temp = m_tail;    // last one
    temp->next = m_tail->prev;
    temp->prev = m_tail->prev->prev;
    temp->next->prev = temp;
    temp->next->prev = nullptr;
    m_tail = temp->next;
}
```