Lecture 5.1

Object Oriented Concepts

Relationships between classes

Three main types:

- Association
  - “has a” relationship.
  - Two or more classes that interact in some manner.
- Aggregation
  - “has a part” relationship.
  - One class is **constructed from** another.
- Generalization (Inheritance)
  - “is a” relationship.
  - One class is **derived from** another (base class)

Association (“has a”) relationship

- **Association** represents the ability of one instance to send a message to another instance. This is typically implemented with a pointer or reference instance variable, although it might also be implemented as a method argument, or the creation of a local variable.
- Example:

  A house "has" furniture. Association

Aggregation (“has a part”) relationship

- **Aggregation** is the typical whole/part relationship. One class is **constructed from** another. There is not much **difference** in the way that the association and aggregation are implemented.
- Example:

  A house "has a part" roof. Aggregation

Generalization (Inheritance)

- Derived class has more specialization than the base class.
- May **override** methods of the base
- May **add new** methods.

Example of Inheritance

```
class Animal {
    private:
        int itsAge;
        float itsWeight;
    public:
        //Accessor methods
        void setAge (int value) { itsAge=value; }
        void setWeight (float value) { itsWeight = value; }
        int getAge ( ) { return itsAge; }
        float getWeight ( ) { return itsWeight; }
        // General methods
        void move () { cout << "Animal Moving\n“ ; }
        void speak() { cout << "Animal Speaking\n“ ; }
        void eat() { cout << "Animal Eating\n“ ; } };
```


Example of Inheritance (cont)

```
class Duck: public Animal {
    private:
        int beakSize;
    public:
        // Accessor methods are not included for simplification
        // this method is said to override the move
        // method from Animal
        void move() { cout << "Waddle\n"; }
        //this method overrides speak from Animal
        void speak() { cout << "Quack\n"; }
};
```


Example of Inheritance (cont2)

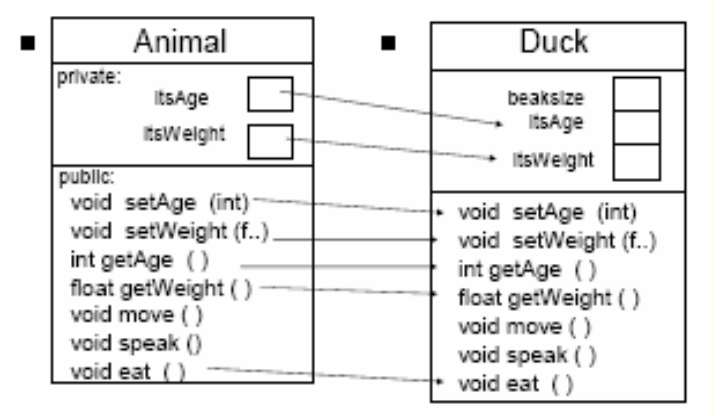
- The Duck still has a weight, an age and can still eat, because it **inherits** these from Animal.

```
int main ( ) {
    Duck ducky;
    cout << "ducky age before setting = " << ducky.getAge() <<endl;
    ducky.setAge (2);
    cout << "ducky age after setting = " << ducky.getAge() <<endl;
    cout << "EAT inherited from animal = ";
    ducky.eat ( );
    cout << "overriden MOVE = " ;
    ducky.move ( );
    cout << "overriden SPEAK = " ;
    ducky.speak ( ); }
```

Output

```
ducky age before setting   = 0
ducky age after setting    = 2
EAT inherited from anmal   = Animal Eating
overriden MOVE             = Waddle
overriden SPEAK            = Quack
Press any key to continue . . .
```


What's happening?



Private vs Protected

```
class Animal {……
    private:
        float itsWeight;
};

class Duck { …..
void print_weight ()
{ cout<<itsWeight;}
};
```

Can't

```
class Animal {……
    protected:
        float itsWeight;
};

class Duck{ …..
void print_weight ()
{ cout<<itsWeight;}
};
```

Can

Inheritance (Another Example)

```
class Pet {
    public:
        Pet ( ) { weight = 1; food = "Pet Chow";}
        ~Pet ( ) { }
        void setWeight (int w) { weight = w; }
        int getWeight ( ) { return weight; }
        void setfood (string f) { food = f; }
        string getFood ( ) { return food; }
        //General Methods
        void eat ( ) {cout<<"eating "<<food<<endl;}
        void speak ( ) {cout<<"Growl"<<endl;}
    protected: // these data can be seen by derived classes
        int weight;
        string food;
};
```