Encapsulation

- *Encapsulation* refers to the idea that an object should protect and manage its own state information.
  - In a way, each object should behave like its own entity.
  - Data security is enforced by the object definition itself.
  - This allows a programmer to make sure that the data being represented is always in a consistent form.
Encapsulation

• Generally speaking, objects should never make their fields public.
  – A public field can be accessed and modified at any time from code that has a reference to the object, which can invalidate its internal state.
Encapsulation

• Note that encapsulation is motivated by the desire to enforce *constraints* on our object.
  – How can we make sure our object is always in a proper, well-formed state if we can’t limit how others modify it?
Encapsulation

• In object-oriented languages, objects may set their fields to be inaccessible outside of the class.
  – To do this, one may use the access modifier `private`.
  – This restricts access to the “private” field or method to only code in the class in which said field or method is defined.
A First Object

// a very basic C++ object
class Person
{
    public:
        string name;
        int age;
}

• So, instead of marking the fields as public...
A First Object

// a very basic C++ object
class Person
{
    private:
        string name;
        int age;
}

• We want to mark them as private.
A First Object

// a very basic C++ object
class Person
{
    private:
        string name;
        int age;
}

• This creates a new problem, though.
  – How can we initialize our object?
Initialization

• By default, when no constructor exists for a class, C++ creates a “default” constructor with no internal code.
  – Note: this “default” constructor will initialize nothing within the class.
  – Java’s default constructor acts differently, setting values to zeros and nulls.
• Typically, we will need to create our own constructors to ensure the class is properly initialized.
// a very basic C++ object
class Person
{
    public:
        Person(string name, int age);

    private:
        string name;
        int age;
}

A First Object

Person::Person(string name, int age)
{
    this->name = name;
    this->age = age;
}

• Something interesting here: note the use of “->”.  
  - “this” is a pointer, and it refers to the instance of the class upon which the constructor/function has been called.
Initialization

• Once one constructor has been coded for a class, the default constructor no longer exists unless it is manually coded.
• A fully constructed and initialized class object can be called an *instance* of its class.
A First Object

// a very basic C++ object
class Person
{
    public:
        Person(string name, int age);

    private:
        string name;
        int age;
}

• We still have another problem.
  – How can we actually make use of the class’s data?
Encapsulation

• Since we’ve set the class fields to private, it is necessary to implement some way of accessing its information – one that does not expose the fields. – The solution? Accessor methods.
A First Object

```cpp
string Person::getName()
{
    return this->name;
}

int Person::getAge()
{
    return this->age;
}
```
A First Object

```cpp
string Person::getName()
{
    return this->name;
}
```

• Suppose we had a “Person p”. The line “p.getName()” would return the value for “name” from the object represented by “p”.