Classes

ITP 165 – Fall 2015

Week 9, Lecture 2

Class

- A ***class*** can have both:
  - Member variables (aka properties)
  - Member functions that can operate on the member variables

- ***And***…we have the option to prevent other code from directly modifying member variables

- So this would solve our problems with the struct version of Clock!

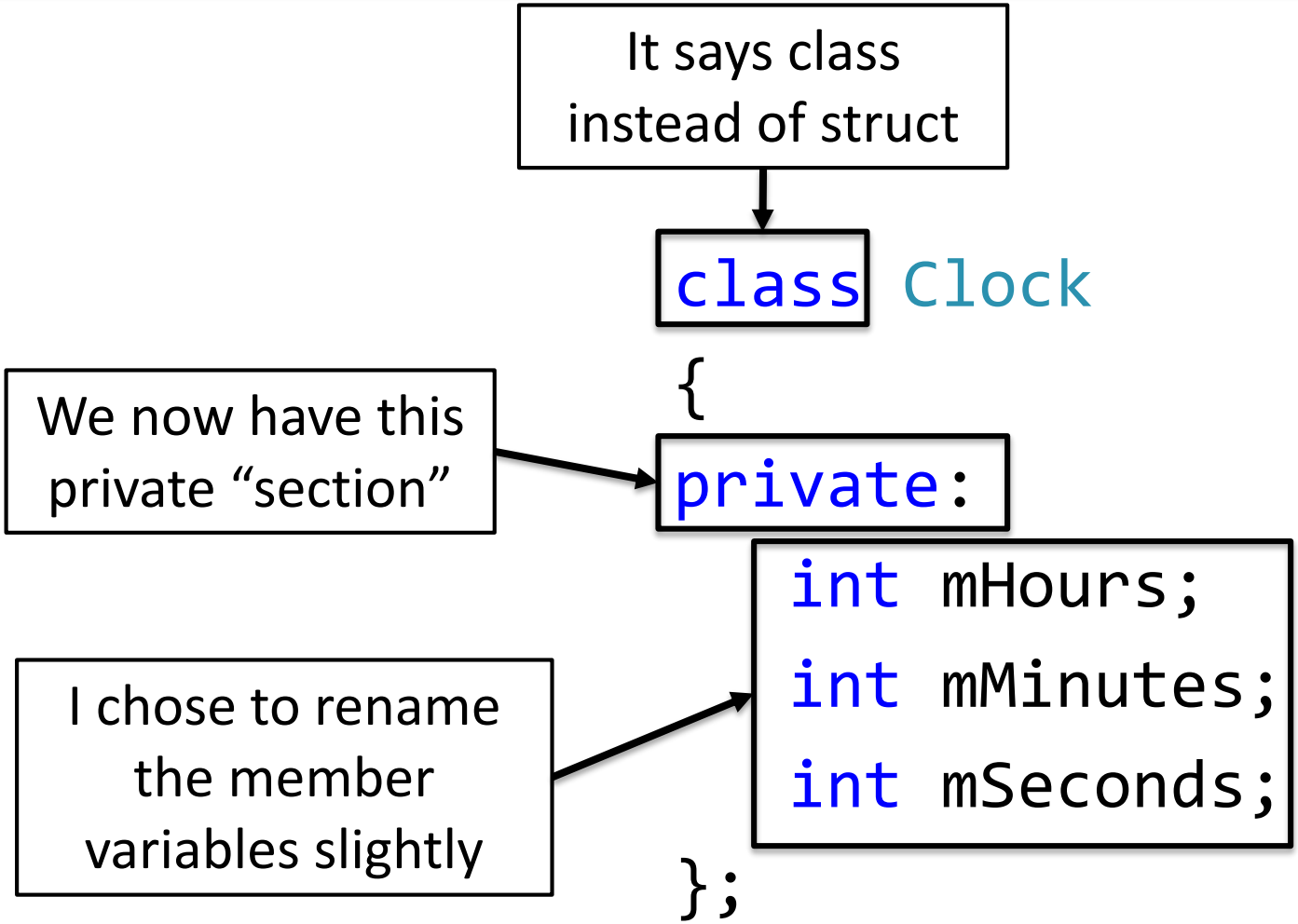
A Clock class

- Let's start out with just a Clock class that only has member data, and no member functions:

```
class Clock
{
private:
    int mHours;
    int mMinutes;
    int mSeconds;
};
```

- What's different from the struct?

A Clock class: What's different?



Privacy

- If something is ***private***, that means it cannot be directly accessed from outside the class

- So by adding the `private:` before the variables, we are explicitly saying that everything following this is private.

- In the case of `Clock`…this means we can't directly modify the `mHours`, `mMinutes`, and `mSeconds` member variables from outside the class

Privacy, Cont'd

- So this would not work:

```
class Clock {
private:
    int mHours;
    int mMinutes;
    int mSeconds;
};

int main() {
    Clock myClock;

    myClock.mHours = 100; // Error: can't access private member

    return 0;
}
```

Privacy, Cont'd

- If something is ***public***, that means it can be directly accessed:

```
class Clock {
public:
    int mHours;
    int mMinutes;
    int mSeconds;
};

int main() {
    Clock myClock;

    myClock.mHours = 100; // Works now!

    return 0;
}
```

Privacy, Cont'd

- If you don't specify public or private, it will default to private:

```
class Clock {
    int mHours;
    int mMinutes;
    int mSeconds;
};

int main() {
    Clock myClock;

    myClock.mHours = 100; // Private error again :(

    return 0;
}
```

Privacy, Best Practices

- In general…
- Unless you have a good reason, all member variables (aka properties) should be ***private***!!

- Following this rule in a class is also called ***data encapsulation***

Why rename the variables?

- In the struct I called them hours, minutes, seconds

- In the class, I call them mHours, mMinutes, mSeconds

- This is just a naming convention where member variables always have a lowercase m at the start of them. I recommend following this convention.

Adding a Member Function

```
class Clock {
private:
    int mHours;
    int mMinutes;
    int mSeconds;
public:
    void reset() {
        mHours = 0;
        mMinutes = 0;
        mSeconds = 0;
    }
};
```

A member function called reset

Adding another Member Function

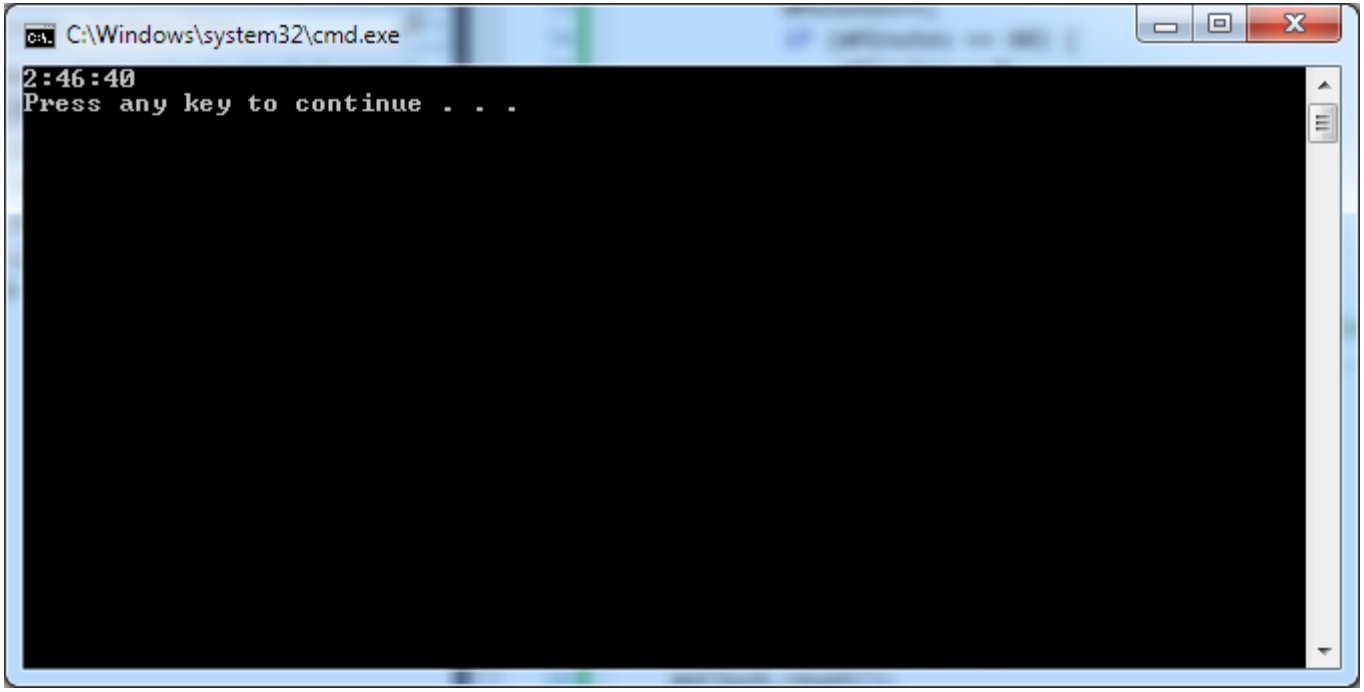
```
class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   void reset() {
      mHours = 0;
      mMinutes = 0;
      mSeconds = 0;
   }

   void print() {
      std::cout << mHours << ":";
      std::cout << mMinutes << ":";
      std::cout << mSeconds << std::endl;
   }
};
```

```
class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   void reset() {
      mHours = 0;
      mMinutes = 0;
      mSeconds = 0;
   }

   void print() {
      std::cout << mHours << ":";
      std::cout << mMinutes << ":";
      std::cout << mSeconds << std::endl;
   }

   void tick() {
      mSeconds++;
      if (mSeconds == 60) {
         mSeconds = 0;
         mMinutes++;
         if (mMinutes == 60) {
            mMinutes = 0;
            mHours++;
            if (mHours == 24) {
               mHours = 0;
            }
         }
      }
   }
};
```

Full Clock class in Action

```
int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   // Call print member function
   myClock.print();

   return 0;
}
```

Full Clock class in Action



What if I want to get just the hours?

- We could write a member function that just returns the number of hours:

```
class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // reset, tick, and print just excluded from slide so this fits
   // but pretend the code for them is still here...
   int getHours() {
      return mHours;
   }
};
```

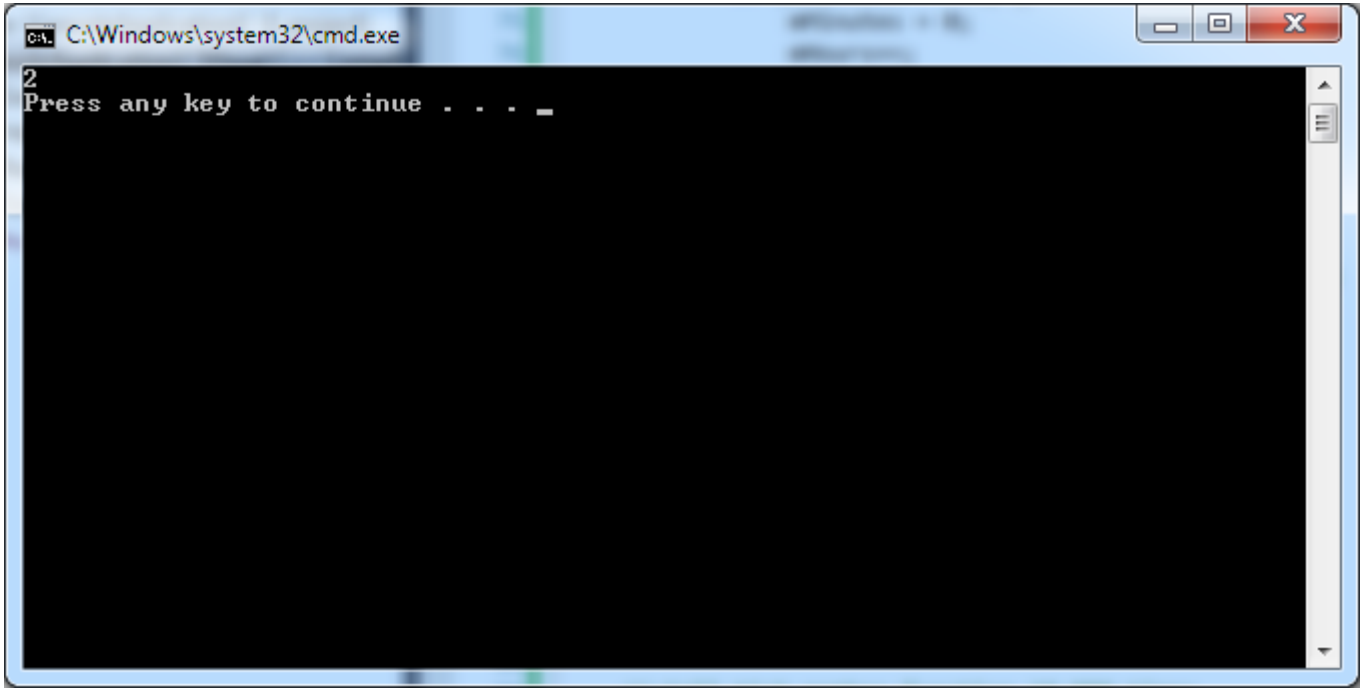
- This type of member function is called a ***getter function***

Using getHours

- Then I could use it in main, like this:

```
int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   // Output only the hours
   std::cout << myClock.getHours() << std::endl;

   return 0;
}
```

Using getHours



What if I want to set the number of hours?

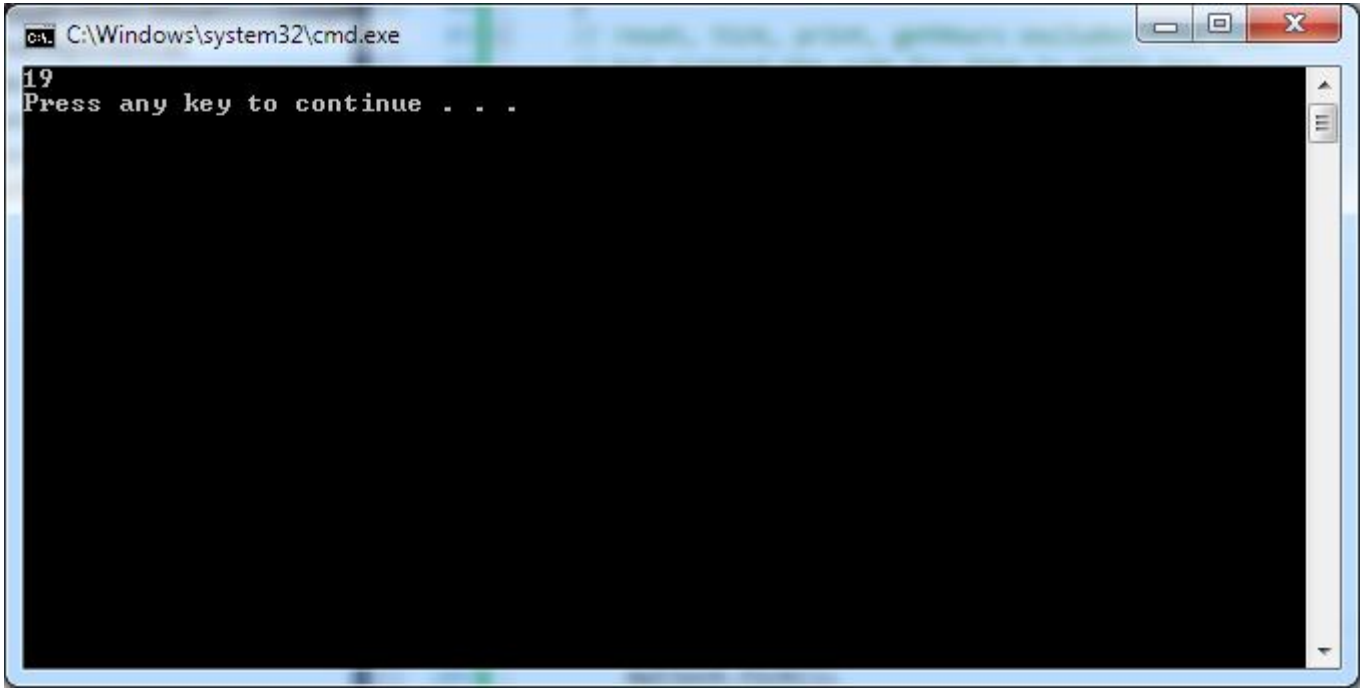
- A ***setter function*** is a member function that allows you to set member variables

```
class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // reset, tick, print, getHours excluded from slide
   // but pretend the code for them is still here...
   void setHours(int newHours) {
      if (newHours >= 0 && newHours <= 23) {
         mHours = newHours;
      }
   }
};
```

Using setHours

```
int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   myClock.setHours(19);

   // Output only the hours
   std::cout << myClock.getHours() << std::endl;

   return 0;
}
```

Using setHours



Lab Practical #15