CSSE 152
Fundamentals of Computer Science II

Computer Science and Software Engineering
Seattle University

Spring 2009

Definition

- **Abstract Data Type:** data types generated by the programmer

- **Abstraction:** A representation or general model of something. The model removes details that are unimportant.

Example

- **Tree:** *(definition from Miriam-Webster)*
  - Pronunciation: \ trē\
  - Function: noun
  - 1 a: a woody perennial plant having a single usually elongate main stem generally with few or no branches on its lower part
- This is an ***abstract*** definition; a general definition
- A specific tree (e.g., the peach tree) is no longer a model; it is a ***concrete*** concept.

In C++

- Computer programs manipulate data
  - To facilitate this, C++ provides primitive data types
  - Data types are *abstractions*
    - Their own domain (or range) of data and set of permitted operations
- What are some primitive data types in C++? Corresponding Operations?

Example

- Data type
  - `integer`
- Concrete occurrence of the data type:
  - int count = 5;

C++ Data Types

- C++ also defines which operations are allowed on a particular data type
  - Example %
    - Allowed for ____________
    - Not permitted for ____________
- Programmers need to create their own data types as well
- **Abstract Data Type (ADT)**
  - A data type created by the programmer that consists of:
    - the data (primitive and other abstract data types)
    - allowable operations

ADTs

- Abstract Data Types ideally should match closely with the item they represent
- C++ provides some mechanisms for creating your own types.
  - Arrays
  - Structs

Structures

- A limitation of arrays is that all data must be of the same type.
- What about related data of different types?
  - Use a structure (`struct`)
- A structure defines the type but does NOT create a particular variable

Example

```
struct Person {   // the name of the type
                  //is person
   string name;
   int ssn;
   float salary;
};          // don’t forget the semicolon

person janitor; //name of the variable
```

“Dot” Operator

- Person: type (defined via the `struct`)
- Person janitor (variable defined based on the type)

janitor.name = “Russ”;
janitor.salary = 17.50;

Array of Structures

- Recall how we used a 2D array to represent grades.
- Structures can simplify our task

```
struct Assign {
   int assign1;
   int midterm;
   int assign2;
   int final;
};
Assign students[4];
```

To Use this Array

- Combine the array reference and dot operator
- Write the code to represent that student 2 received a 75 on the midterm.

Introduction to Classes

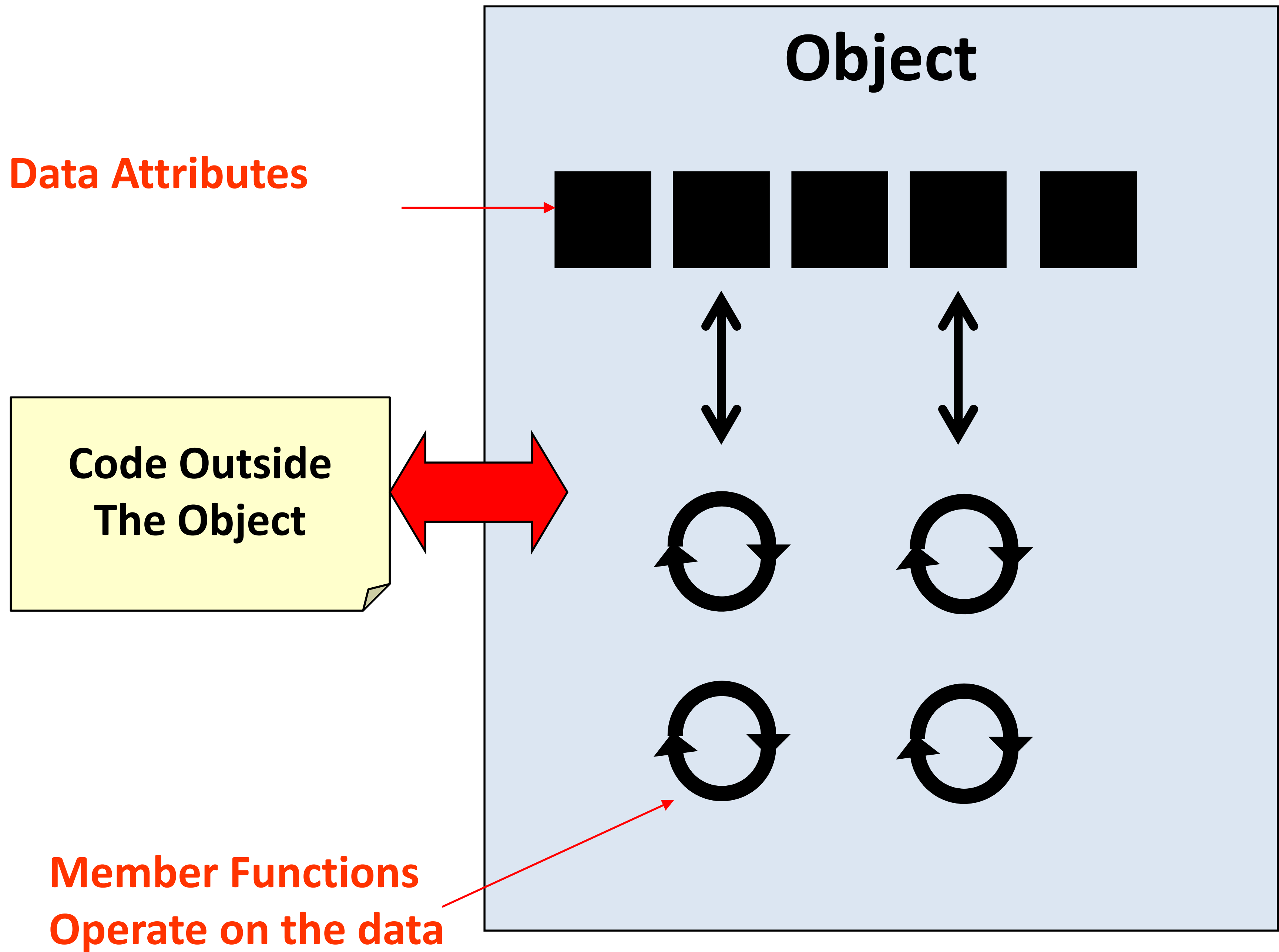
Chapter 13

Introduction to Classes (ch 13)

- Procedural Programming
  - Focused on set of functions (aka procedures) that manipulate data
- Object Oriented Programming
  - Centers on creating and manipulating *objects*

Objects

- **Object:** A software entity that contains both data and procedures
- It contains two parts:
  – **Attributes:** the data contained in the object (also called *data members*).
  – **Member functions:** The functions that an object performs (also called *methods* in other languages).

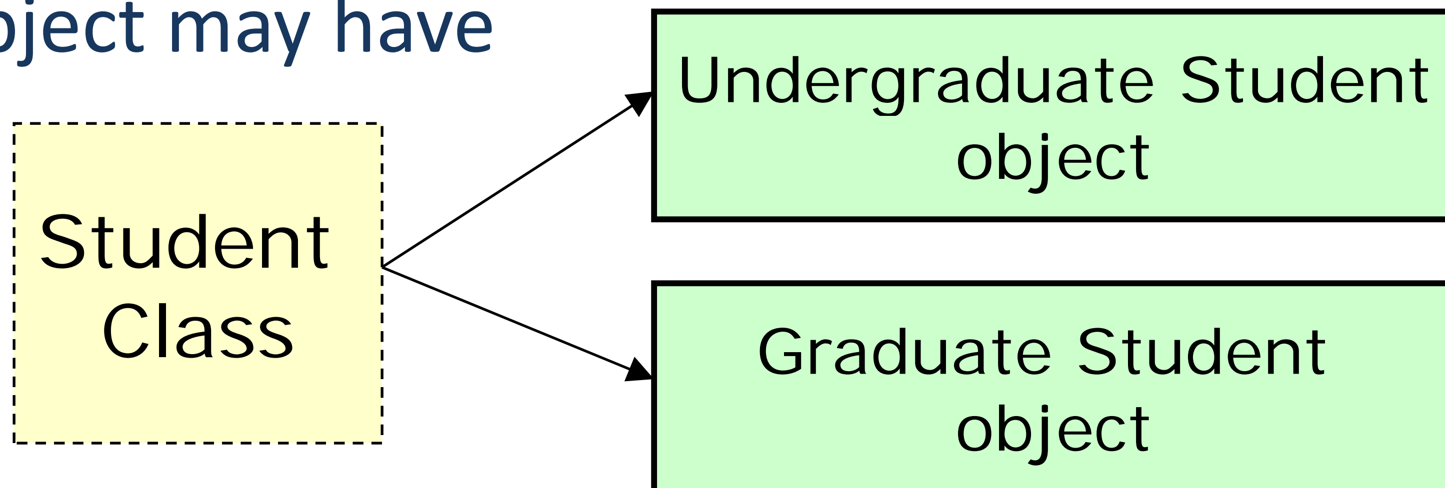

Object
Data Attributes
Code Outside
The Object
Member Functions
Operate on the data

Two Key Principles of OOP

- **Encapsulation**
  - Combining of data and code into a single object
- **Data Hiding**
  - An object's ability to hide its data from code that is outside the object
- Results:
  - protection of data from accidental corruption; making the code more *modifiable*
  - Making the code (object) reusable

Objects and Classes

- The programmer must design the objects
  - Determine attributes and functions
  - Create a `class`
- A class is code that specifies the attributes and member functions that a particular type of object may have



13.2: Introduction to Classes

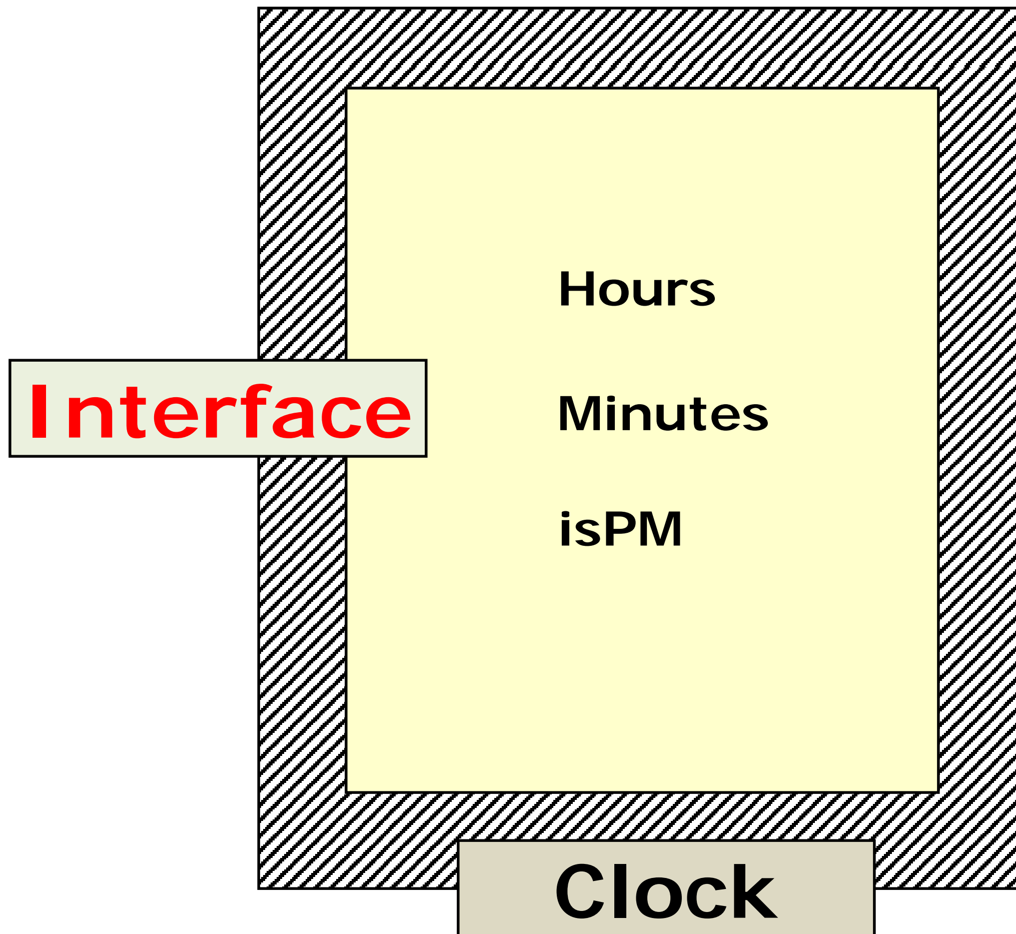
- In C++, a **class** is used to describe an object
- A `class` is similar to a `struct`
- Write a structure that represents the time of the day:

The `Time` Class

Access Specifiers

- **Private section of a class** consists of:
  - Data members, Member functions, and Constants
  - Only visible within the member functions of the class
  - Not visible to functions outside of the class
  - Similar to how local variables in functions are only visible within the function
- **Public section of a class** consists of
  - Variables, Member functions, and Constants
  - Visible outside of the class
  - Also known as the *class interface*

Visualization – the Wall



Class=
Structure +
Wall around it

The private data members are inside the wall and can only be accessed by member functions that act as guards to the wall.

Clock.hours = 5;

Defining Member Functions

- Let’s complete the class description with the definition of the member functions
  - Write the member function for `set`
  - Write the member function for `display`

Class Problem

- Write the member function for `increment()`

Review of Classes (ch 13.2)

- A **class** is used to describe an object
  - In OO programming, the object typically represents a physical object in real world
    - e,.g, Time, Date, Employee, …
- Class definition is similar to a struct

```
Class Time {
…
};
```

The `Time` Class

```
class Time
{
  public:
    void display() const;
    void set(int hoursArg, int minutesArg, bool
  isPMArg);
    void increment();

  private:
    int hours;
    int minutes;
    bool isPM;
};
```

Member Functions

- `void Time::set(int hoursArg, int minutesArg, bool isPMArg);`

- `void Time::display() const;`

- `void Time::increment();`

Notes

- Data members must be private
- Local variables vs. data members
- By convention, names of classes begin with a capital letter.
  - Software companies may have their own convention.
- keyword `const` should be used when a member function does not modify data members

Defining an Instance of a Class

- Defining a class essentially creates a type
- Once a type is defined, you can then create an instance of it
  - E.g.,

```
int count;
Time mealTime;
Time classTime;
```

How to Call Member Functions?

- Calling member functions is done using the dot operator just like **structs**.
- The big difference is that you cannot directly access private member using the dot operator

Why Have Private Data Members? (ch 13.4, 13.11)

- In most classes and programs (including ALL of the programs for this course):
  - All data members are private. Access can be obtained via member functions.
  - Most member functions are public.
  - It may be helpful to have internal member functions that are private.
    - Private member functions can only be called from other member functions. Make member functions you do not want users of the class to execute.

Security Alert!!

- Who is the user of a class?
  - Person executing the program
  - Also could be another programmer
    - You, someone else in your organization, some other person
- The **private** designation is used to make sure the programmer does not do something stupid (like create an illegal time value).
- It has nothing to do with the security of the data elements (use encryption / decryption).

Advantages of this Setup

- Programs cannot create illegal data (such as `t1.hours =45`).
  - Error checking can be built into the class.
- Programmers learn only what they need to know.
  - Interface is the only thing that other programs or programmers need to know. No need to worry about the fields within the structure.
- The private section can be changed without affecting programs that use the class. Imagine changing the name of a field in a structure.

More Advantages

- Better separation
  - The operations are self-contained within the class. Without member functions, portions of the algorithms may need to be replicated in several parts of the code. Changes to code do not have to be replicated across many functions.
- Make the code more readable for future programmers (includes yourself if you go 3 months without looking at your code).

More Tenets of OOP

- **Encapsulation:** combining of data and code into a single object.
- **Data hiding**: Hiding the inner details of a software component.
  - E.g., automobiles
- **Object reusability**: The ability to reuse a class for multiple programs.
  - E.g., string class, graphics, and windows libraries

Types of Member Functions

- **Accessors:** functions that retrieve member data (get).
- **Mutators:** functions that modify member data (set)
  - Do accessors and mutators violate the spirit of having data members private?
- **Constructors:** functions to instantiate a class