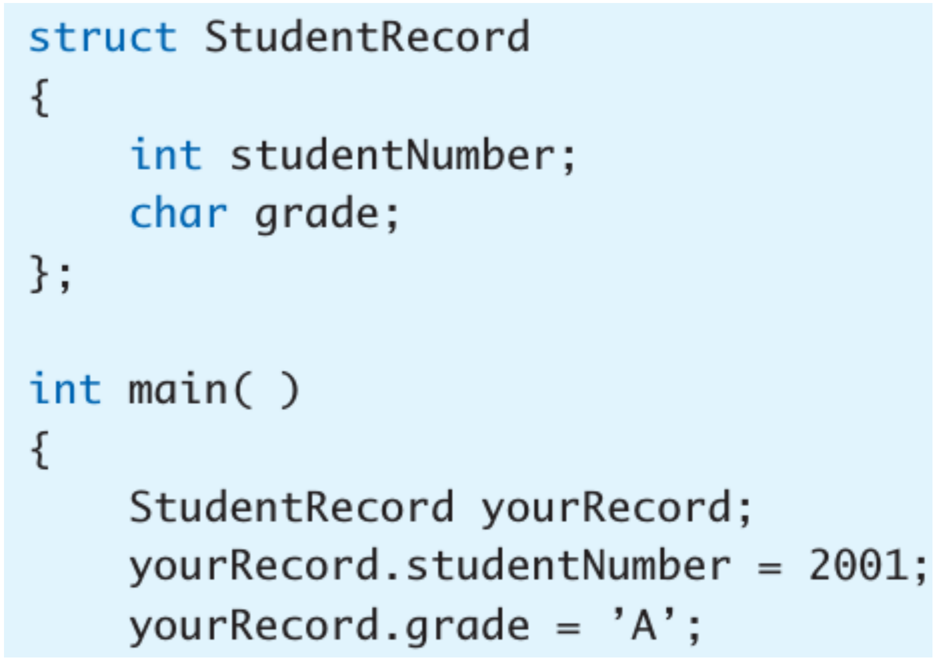
CSCI-2312 OOP

Structures vs. Classes

Textbook reference: Chapter 6

Structure

- A collection of different types of variables
  - That collection is under the same name
  - You can access <u>every</u> member variable using the Dot operator



```
struct StudentRecord
{
    int studentNumber;
    char grade;
};

int main( )
{
    StudentRecord yourRecord;
    yourRecord.studentNumber = 2001;
    yourRecord.grade = 'A';
```

Structure : reusing member names

```
struct FertilizerStock
{
    double quantity;
    double nitrogenContent;
};
```

```
struct CropYield
{
    int quantity;
    double size;
};
```

Hierarchical structures

- Sometimes it makes sense to have structures whose members are themselves smaller structures

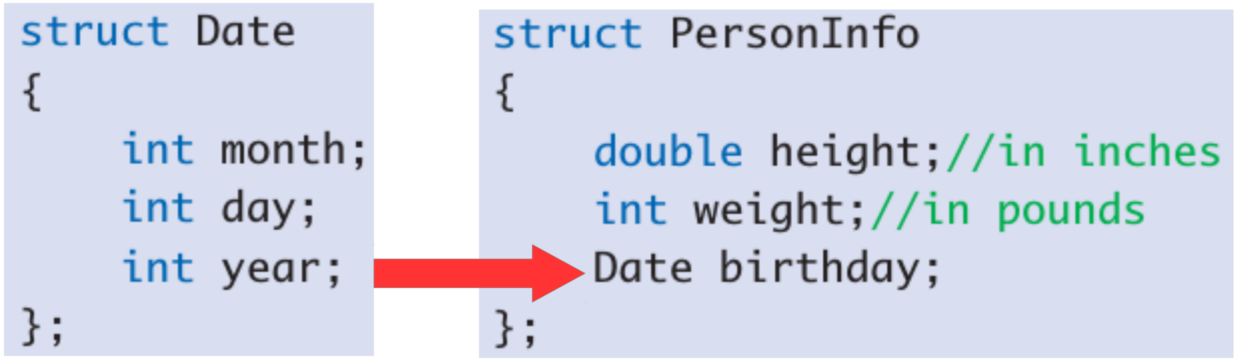
```cpp
struct Date
{
    int month;
    int day;
    int year;
};
```

```cpp
struct PersonInfo
{
    double height;//in inches
    int weight;//in pounds
    Date birthday;
};
```

```cpp
PersonInfo person1;
```

```cpp
cout << person1.birthday.year;
```

Structures as Function arguments



```
struct Date
{
    int month;
    int day;
    int year;
};
```

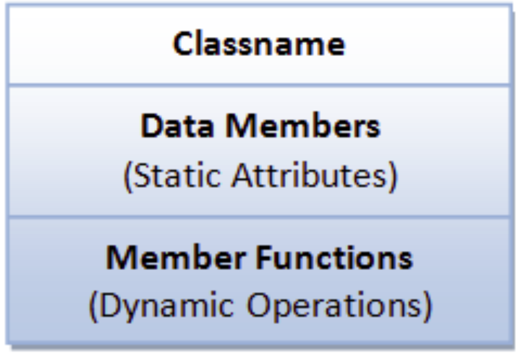
```
struct PersonInfo
{
    double height;//in inches
    int weight;//in pounds
    Date birthday;
};
```

```
void myFunction(PersonInfo p){
  cout << p.height << endl;
  cout << p.weight << endl;
  cout << p.month <<"/"<<p.day<<"/"<<p.year<<endl;

  p.weight++;
}
```

Class

- It's actually a extended structure
  - can accommodate **member variables** and **member functions**.
  - It's the key concept in Object Oriented Programming (OOP)
- Replace the keyword **struct** with **class** in the last slide.
- Every member inside a structure is **public** by default,
  - while every member inside a class is **private** by default.
  - *How to convert a class into a structure?*
- A variable of a class type is often called an object.

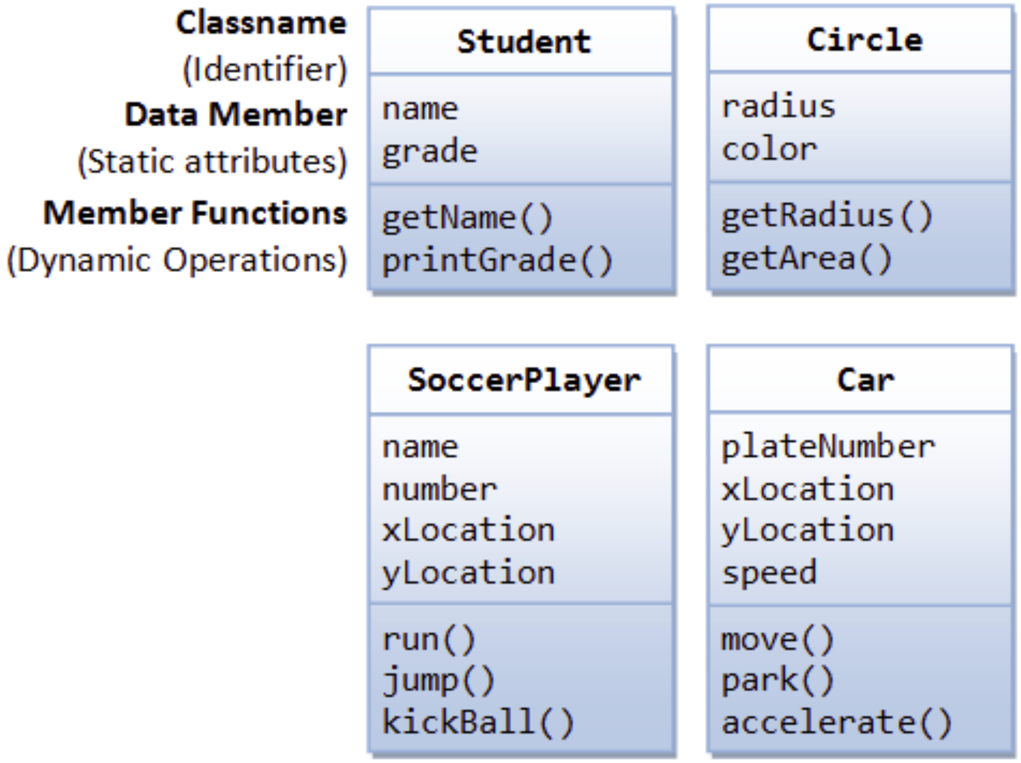
Demo: 3A.cpp & 3B.cpp



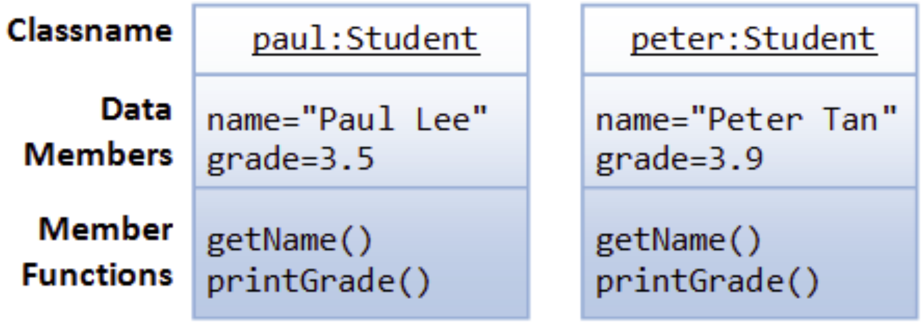
A class is a 3-compartment box encapsulating data and functions

- Classname / identifier
- Data members / member variables / attributes / fields
- Member functions / methods / behaviors / operations

Classes vs. instance of classes



Examples of classes



Two instances of the **Student** class

Class declaration

```
class DayOfYear
{
public:
    void output( );
    int month;
    int day;
};
```

Member function declaration

```
void DayOfYear::output( )
{

}
```

Member function definition

Scope resolution operator :: purpose is similar to the dot operator.
But :: is used with a class name,
dot **.** operator is used with objects/class-variables.

```
class DayOfYear
{
public:
    void output( );
    int month;
    int day;
};
```

- In the function definition for a member function-
  you can use the names of all members of that class.

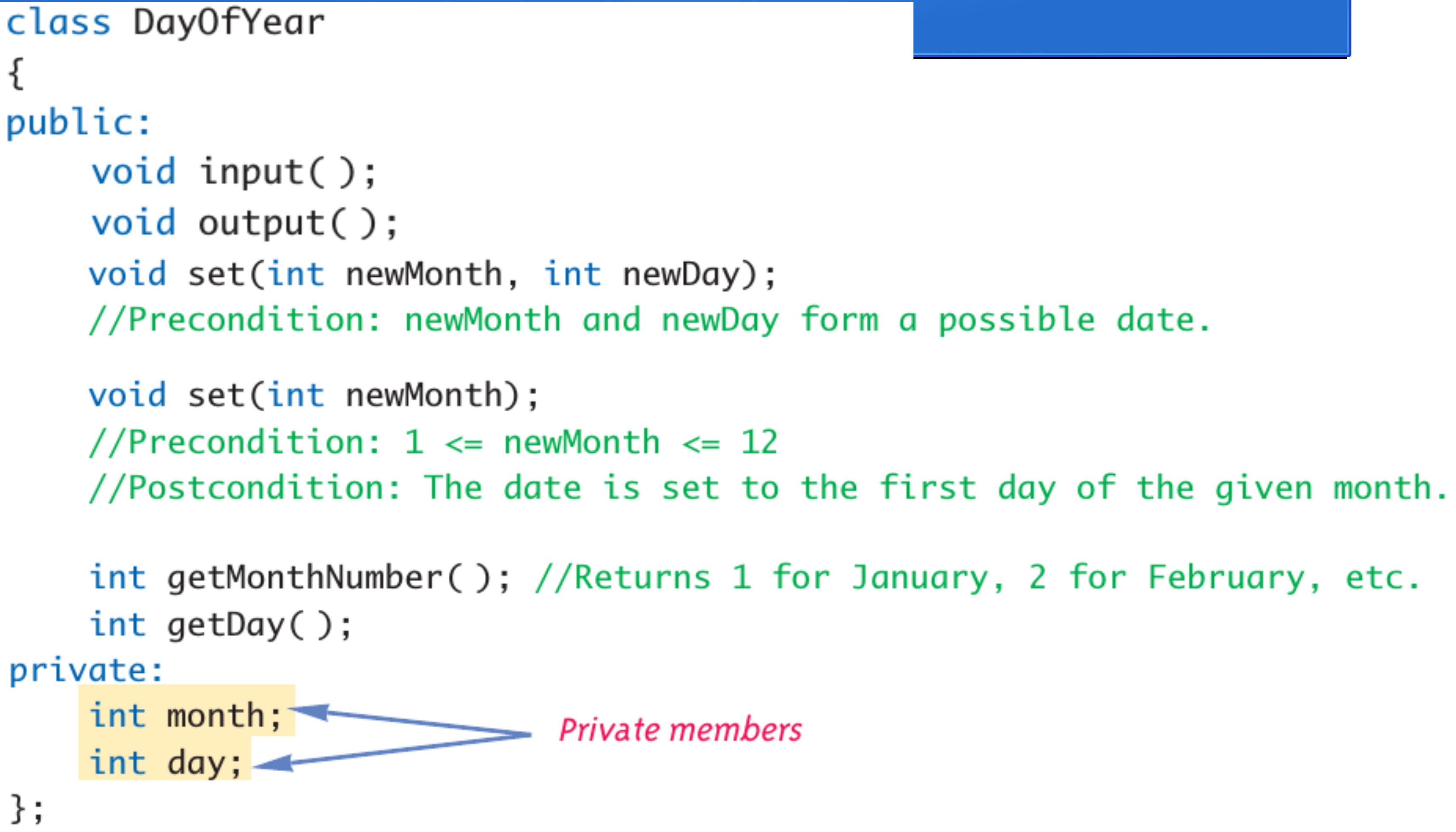
```
void DayOfYear::output( )
{
    switch (month)
    {
        case 1:
            cout << "January "; break;
        case 2:
            cout << "February "; break;
        case 3:
            cout << "March "; break;
        case 4:
            cout << "April "; break;
        case 5:
            cout << "May "; break;
        case 6:
            cout << "June "; break;
        case 7:
            cout << "July "; break;
        case 8:
            cout << "August "; break;
        case 9:
            cout << "September "; break;
        case 10:
            cout << "October "; break;
        case 11:
            cout << "November "; break;
        case 12:
            cout << "December "; break;
        default:
            cout << "Error in DayOfYear::output. Contact software vendor.";
    }

    cout << day;
}
```

Member function definition

Abstract Data Types (ADT)

- A data type consists of a collection of values together with a set of basic operations defined on these values.
  - Values: ..., -3, -2, -1, 0, 1, 2, 3, ...
  - Operations: +, -, *, /, %
- A data type is an ADT if the programmer who uses the type has access to some values or operations.
  - but, does not have access to the implementation details/ definitions of the all the values and the operations, etc.
- Classes are also ADT – programmers who use them need not be concerned with the details of how the class is implemented
  - They only need to know the rules for how to use the class.
  - These set of rules are known as **interface** in C++:
    - Comments at the beginning of the class definition
    - public member functions along with the comments that tell how to use the them.
  - This is called **encapsulation / information hiding / data abstraction**

Public & Private members


Private members

```
class DayOfYear
{
public:
    void input( );
    void output( );
    void set(int newMonth, int newDay);
    //Precondition: newMonth and newDay form a possible date.

    void set(int newMonth);
    //Precondition: 1 <= newMonth <= 12
    //Postcondition: The date is set to the first day of the given month.

    int getMonthNumber( ); //Returns 1 for January, 2 for February, etc.
    int getDay( );
private:
    int month;
    int day;
};
```

Public & Private members

```
class DayOfYear
{
public:
     void input( );
     void output( );
     void set(int newMonth, int newDay);
     //Precondition: newMonth and newDay fo

     void set(int newMonth);
     //Precondition: 1 <= newMonth <= 12
     //Postcondition: The date is set to th

     int getMonthNumber( ); //Returns 1 for
     int getDay( );
private:
     int month;
     int day;
};
```

Private members

```
DayOfYear today; //This line is OK.
today.month = 12;//ILLEGAL
today.day = 25;//ILLEGAL
cout << today.month;//ILLEGAL
cout << today.day;//ILLEGAL
if (today.month == 1) //ILLEGAL
    cout << "January";
```

Accessor & Mutator Functions
Get & Set Functions

```
class DayOfYear
{
public:
    void input( );
    void output( );
    void set(int newMonth, int newDay);
    //Precondition: newMonth and newDay form

    void set(int newMonth);
    //Precondition: 1 <= newMonth <= 12
    //Postcondition: The date is set to the first day of the given month.

    int getMonthNumber( ); //Returns 1 for
    int getDay( );
private:
    int month;
    int day;
};
```

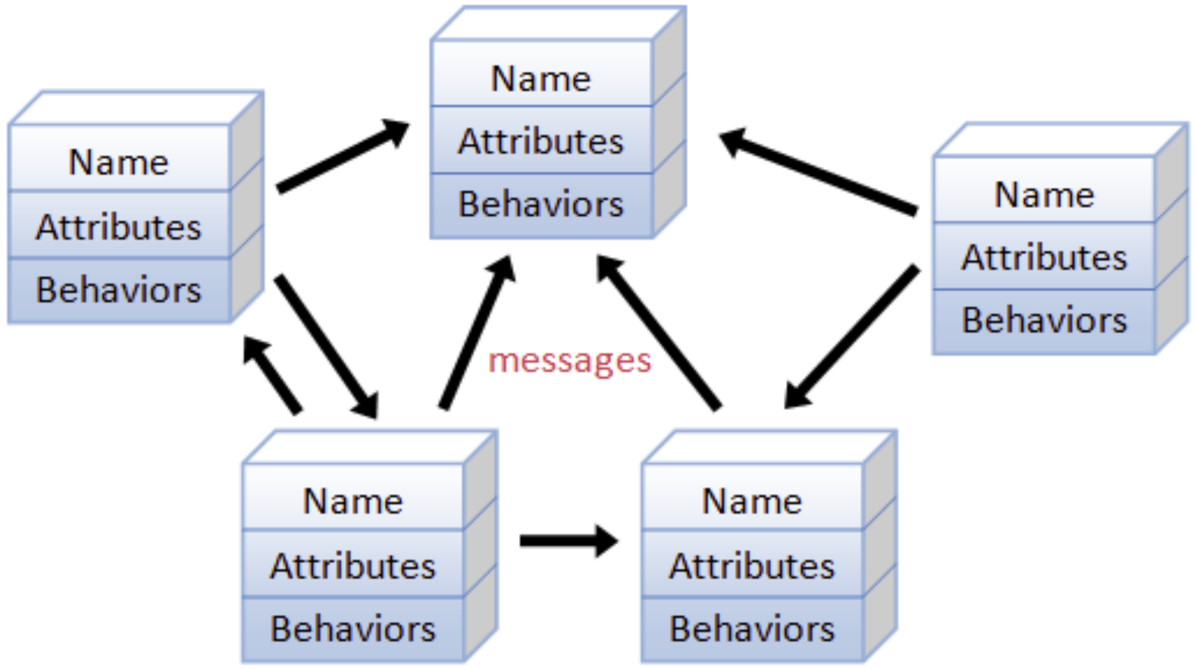
MUTATOR FUNCTIONS

ACCESSOR FUNCTIONS

Private members

Convention is to make all member variables in a class private

This is the big picture of OOP



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Exercise 1

```
class Automobile
{
public:
    void setPrice(double newPrice);
    void setProfit(double newProfit);
    double getPrice( );
private:
    double price;
    double profit;
    double getProfit( );
};
```

```
int main(){
    Automobile hyundai, jaguar;
}
```

Which of the following are allowed in the main function?

```
hyundai.price = 4999.99;
jaguar.setPrice(30000.97);
double aPrice, aProfit;
aPrice = jaguar.getPrice( );
aProfit = jaguar.getProfit( );
aProfit = hyundai.getProfit( );
hyundai = jaguar;
```

Exercise 2

- Create a `Temperature` class that internally stores a temperature in degree Kelvin. Create functions named `setTempKelvin`, `setTempFahrenheit`, and `setTempCelsius` that take an input temperature in the specified temperature scale, convert the temperature to Kelvin, and store that temperature in the class member variable. Also, create functions that return the stored temperature in degrees Kelvin, Fahrenheit, or Celsius.

  Write a `main` function to test your class.