A class is a data type whose variables are objects

- Some pre-defined classes in C++ include `int`, `char`, `ifstream`
- Of course, you can define your own classes too

A class definition says two basic things

- The kinds of values an object can hold
- A description of the member functions
Example: class DayOfYear

- Decide on the values to represent
- This example's values are dates such as July 4 using an integer for the number of the month
  - Member variable month is int (Jan = 1, Feb = 2, etc.)
  - Member variable day is int
- Decide on the member functions needed
- Just one member function named output in the first version of this class
Simplest version of DayOfYear

- Like a struct with an added method
  - All parts public
  - Clients access month, day directly

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

void DayOfYear::output() {
    cout << "month = " << month
         << ", day = " << day << endl;
}
```
Notes about '::' and '.

- '::' used with classes to identify a member
  ```
  void DayOfYear::output() { ... }
  ```
  - Also used with namespaces – identifies scope
  - Called scope resolution operator

- '.' used with variables to identify object
  ```
  DayOfYear birthday;
  birthday.output();
  ```
  - Object reference is passed to the method as an implicit parameter
What's wrong with DayOfYear?

- Most important: data are exposed to users
- Why is that a problem?
- Two major reasons:
  - No way to insure consistent object states – e.g. user could `birthday.month = 74;` // huh?
  - Developer can't change data names/meanings – e.g. can't change int to string for month, can't save Date instead of month, day, …
- What's the solution (in C++)?
An access specifier: `private`

- Private members of a class can only be referenced within the definitions of member functions (and `friends` – later)
  - If the program tries to access a private member, the compiler gives an error message
- Private members can be data or functions
  - Should have public set methods to change data
  - Need public get methods to access the data
- Btw: default for class is private (public for `struct`)
Better class DayOfYear

class DayOfYear {
public:
    void input( );
    void output( );
    void set(int new_month, int new_day);
    int get_month( );
    int get_day( );
private:
    void check_date( );
    int month;
    int day;
};
Creating and assigning (=) objects

- Declaring an object *creates the object*
  ```
  DayOfYear today, tomorrow;
  // two objects are created on stack
  ```

- Different if declaring pointers (or references)
  ```
  DayOfYear *soon, &r = today; // no object
  soon = new DayOfYear; // now object on heap
  ```

- Assignment operator *copies object’s data*
  ```
  r = *soon; // no new object—just copy on stack
  // original (today) object data overwritten
  ```
Another class example: BankAccount

- Has operations appropriate for a bank account (implemented with public member functions)
  - And a private utility function
- Stores an account balance and an interest rate

```cpp
The BankAccount Class (part 1 of 4)

//Program to demonstrate the class BankAccount.
#include <iostream>
using namespace std;

//Class for a bank account:
class BankAccount
{
public:
    void set(int dollars, int cents, double rate);
    //Postcondition: The account balance has been set to $dollars.cents;
    //The interest rate has been set to rate percent.
    void set(int dollars, double rate);
    //Postcondition: The account balance has been set to $dollars.00.
    //The interest rate has been set to rate percent.
    void update();
    //Postcondition: One year of simple interest has been
    //added to the account balance.
    double get_balance();
    //Returns the current account balance.
    double get_rate();
    //Returns the current account interest rate as a percentage.
    void output(ostream& outs);
    //Precondition: If outs is a file output stream, then
    //outs has already been connected to a file.
    //Postcondition: Account balance and interest rate have been written to the
    //stream outs.
private:
    double balance;
    double interest_rate;

    double fraction(double percent);
    //Converts a percentage to a fraction. For example, fraction(50.3) returns 0.503.
};

int main()
{
    BankAccount account1, account2;
    cout << "Start of Test:\n";
    
```
Method overloading – BankAccount::set

- A method's **signature** includes its name and its parameter list
- Can **overload** a name like set with a different parameter list
  - Number, types, order

```cpp
void BankAccount::set(int dollars, int cents, double rate)
{
    if ((dollars < 0) || (cents < 0) || (rate < 0))
    {
        cout << "Illegal values for money or interest rate.\n";
        exit(1);
    }
    balance = dollars + 0.01*cents;
    interest_rate = rate;
}
void BankAccount::set(int dollars, double rate)
{
    if ((dollars < 0) || (rate < 0))
    {
        cout << "Illegal values for money or interest rate.\n";
        exit(1);
    }
    balance = dollars;
    interest_rate = rate;
}
```

The BankAccount Class (part 2 of 4)
More implementing BankAccount

The BankAccount Class (part 3 of 4)

```cpp
void BankAccount::update()
{
    balance = balance + fraction(interest_rate)*balance;
}

double BankAccount::fraction(double percent_value)
{
    return (percent_value/100.0);
}

double BankAccount::get_balance()
{
    return balance;
}

double BankAccount::get_rate()
{
    return interest_rate;
}

//Uses iostream:
void BankAccount::output(ostream& outs)
{
    outs.setf(ios::fixed);
    outs.setf(ios::showpoint);
    outs.precision(2);
    outs << "Account balance $" << balance << endl;
    outs << "Interest rate " << interest_rate << "%" << endl;
}
```

- In the definition of a member function, you call another member function like this.
- Stream parameter that can be replaced with either cout or with a file output stream.
- Using stream manipulators.
Sample BankAccount results

The BankAccount Class (part 4 of 4)

Sample Dialogue

- Start of Test:
  - account1 initial statement:
    - Account balance $123.99
    - Interest rate 3.00%
  - account1 with new setup:
    - Account balance $100.00
    - Interest rate 5.00%
  - account1 after update:
    - Account balance $105.00
    - Interest rate 5.00%

account1.set(123, 99, 3);
  // called with all 3 arguments

account1.set(100, 5);
  // called other version of set

account1.update();

account2 = account1;

Q: What if account2.update()?
Constructors

- A constructor (a.k.a. \texttt{ctor}) is a member function
  - Usually declared public
- One is always called when an object is created
- Main purpose – initialize \texttt{instance} variables
  - Also useful to allocate resources if needed
- Constructor's name must be the name of the class
- A constructor cannot return a value
  - No return type, not even \texttt{void}
A BankAccount constructor

- Declare in public part of class definition
  
  ```
  BankAccount(int dollars, int cents, double rate);
  ```

- Implement essentially like other methods
  
  ```
  BankAccount::BankAccount(int dollars, int cents, 
                          double rate) {
      if ((dollars < 0) || (cents < 0) || (rate < 0)) {
        cout << "Illegal values for money or rate\n";
        exit(1);
      }

      balance = dollars + 0.01 * cents;
      interest_rate = rate;
  }
  ```
Constructor call is automatic

- **May not** `invoke` (i.e., call) it directly:
  
  ```
  account1.BankAccount(10, 50, 2); // ERROR
  ```

- **Instead invoke indirectly**
  
  - **On stack:** `BankAccount account1(10, 50, 2);
  ```
  - **Or free store:** … `new BankAccount(10, 50, 2);
  ```

- **But class must have a matching constructor**
  
  - *e.g.*, `BankAccount()` if just `new BankAccount;
  ```
    - Default constructor is called – but oops: ERROR if explicit constructor is defined and not overloaded!
Overloading and the default ctor

- Another possible BankAccount ctor:
  BankAccount (double balance, double interest_rate);

- Or can have either one of the following. Why not both?
  BankAccount (double balance);
  BankAccount (double interest_rate);

- Also either explicitly define default ctor:
  BankAccount ( );

  Or implicitly via default arguments in other ctors:
  BankAccount (double balance = 0.0);

- Tip: good idea to always include a default ctor even if there is no need to initialize variables
  - So clients can: BankAccount checking, savings;
  - Important for inheritance reasons too (a future topic)
Base/member initialization list

- An initialization section in a constructor definition provides an alternative way to initialize member variables
  ```cpp
  BankAccount::BankAccount() : balance(0), interest_rate(0)
  {
  } // still need a body (even if intentionally empty like this case)
  - Can use parameter names too – even if same name as member!

- Note: order of initialization matches the order in which the variables are declared in the class, not their order in the list

- *Must* use such a list for constants and reference variables (since references are always constant)
  - Also must use to initialize private data in a base class (later topic)

- *Should* always use for user-defined types if default ctor not appropriate – to avoid extra ctor (and destructor) calls