Writing Classes

Kate Gregory

www.gregcons.com/kateblog @gregcons

A Class of Your Own

A class pulls together related data

- A customer's first name, last name, address, and phone number
- An account's number, balance, and list of transactions

It also adds functions that (generally) operate on the data

- Get a customer's full name
- Post a transaction to an account – including updating the balance
- Includes operator overloads if those make sense

Keeping these together makes the code easier to change and use

Simple Class System - Design

Account

- **Keeps track of a balance**
- **Holds a vector of Transaction objects**
- **Deposit and Withdraw member functions**
- **Report function – collection of strings that calling code can print**

Simple Class System - Design

Transaction

- **Should have a date, but we'll ignore that for now**
- **Holds an amount, and a transaction type (string for now)**
- **Report function – string describing amount and type**

Simple Class System - Design

Account member functions:

Deposit:

- Create a Transaction
- Add it to the vector
- Update the balance

Withdraw

- You can't take out more than you have
- Create a Transaction
- Add it to the vector
- Update the balance

Translating Design Into Code

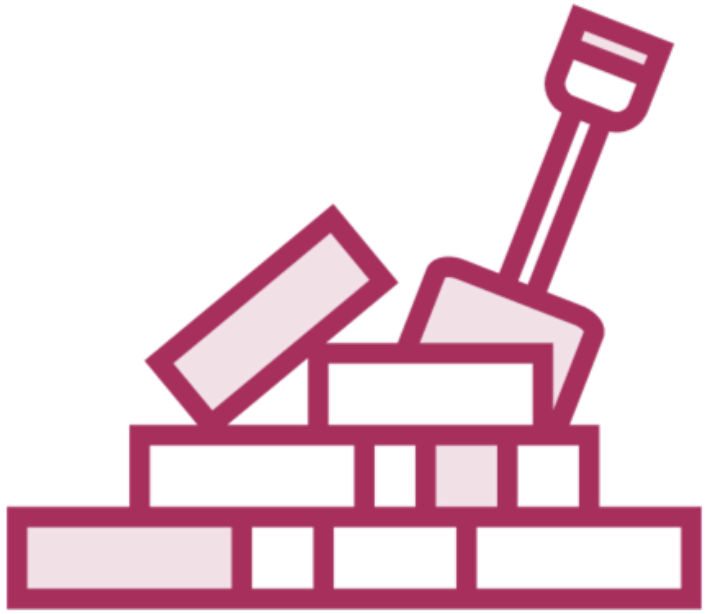
Generally, member variables are private

Encapsulation

Functions you think of early are usually public

Services the class offers

Constructors



Some classes need constructors to initialize variables

- Use special initializing syntax to initialize member variables

Name of the constructor function is name of the class

- Constructors have no return type
  - Not the same as returning void

Like any other function a constructor can take parameters

- Use them to initialize variables

Like any other function, constructors can be overloaded

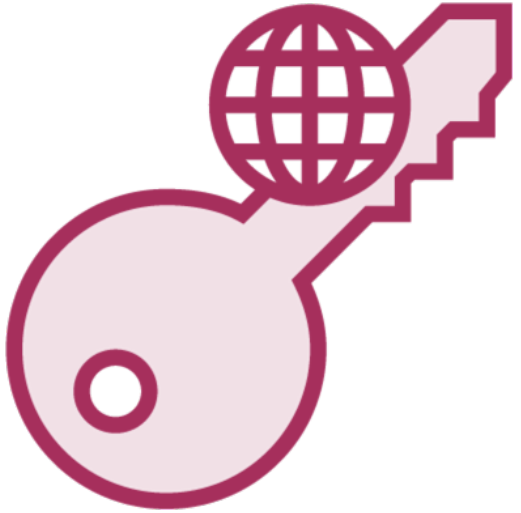
Structuring the Code

- Can define it all in one file, but usually don't
- One header file per class just declares what is in the class
- One .cpp file per class implements all the functions
- Any code that uses the class includes the header
- So does the .cpp file that implements the class

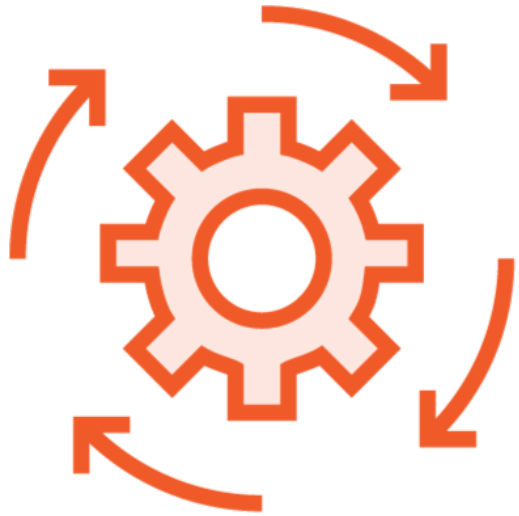
Keywords to Know

`class { …many lines … };`

`private:`



`public:`



Scope resolution operator `::`

```
class Account

{

private:

    int balance;
    std::vector<Transaction> log;

public:

    Account();

    std::vector<std::string> Report();

    bool Deposit(int amount);

    bool Withdraw(int amount);

    int GetBalance() {return balance;}

};
```

- Private variables used only from member functions

- Account is a constructor for the class

- Report takes no parameters, returns a vector of strings
- Deposit and Withdraw each take an integer and return a bool

- GetBalance is implemented inline

Inline Functions

Some functions are really obvious

Added mostly to keep the data private

Makes sense to show the code right with the declaration of the function

Often called “inline”

```
Account::Account(): balance(0){}

vector<string> Account::Report()

{

        vector<string> report;

        report.push_back("Balance is " + to_string(balance));
        report.push_back("Transactions: ");

        for (auto t:log)
        {
            report.push_back(t.Report());
        }

    report.push_back("------------------");

    return report;

}
```

```
bool Account::Deposit(int amount)

{

    if (amount <= 0)
    {
      return false;
    }

    balance += amount;

    log.push_back(Transaction(amount,"Deposit"));
    return true;

}
```

```
bool Account::Withdraw(int amount)

{

        if (amount <= 0)
        {
      return false;
    }

    if (balance >= amount)
    {
      balance -= amount;
      log.push_back(Transaction(amount,"Withdraw"));
      return true;
    }

    return false;

}
```

Creating Instances

A constructor that takes no arguments is called a *default constructor*

Declare objects with default constructors the same as built in types:

- `Account acct;`

Declare objects with parameter-taking constructors using () or {}

- `Transaction t(amount, type);`
- `Transaction t{amount, type};`

Don't use = when declaring an object and initializing it

- There are exceptions, but this is a good general rule

This code doesn't do what you think it does:

- `Account acct(); //actually declares a function!`

Account acct{}; //same as `Account acct;`

Encapsulation

A well written class is changeable

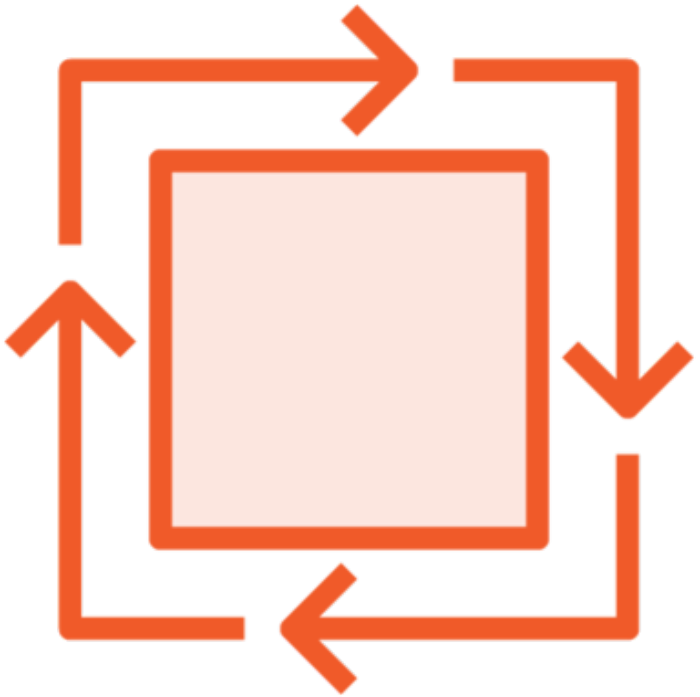
Make all your member variables private

Code outside the class can't count on their names or types

You can change name, type without breaking code outside the class

Code outside the class doesn't need to know the rules or remember them

Encapsulation



You can add public member functions as gatekeepers

- Eg `GetBalance()` to find out an account's balance
  - Never assume one `GetSomething()` for every member variable
- Don't always need a `SetSomething()`

The best public functions are things the class itself does, not ways to manipulate some of the data in the class

Encapsulation

- Add as few public member functions as you can
- Use private functions if you just want to keep from repeating code or give something a name
- The more that is encapsulated, the better
- Changes in one part of the code don’t affect other places
- Easier for the developer and less likely to cause bugs elsewhere

Lifetime Management

Some classes work with real-world things

- Open a file … and need to close it
- Open a database connection … and need to close it

When an object is out of scope, you can't ask it to close or clean up the things it held

C++ has great mechanisms to manage this

- Class can have a destructor that is called automatically
- RAII makes life simple
- You need to deal with copying
  - Two classes have copy of handle and one thinks it's done and closes the file?

Simple classes that just have local variables in them don't need to worry about lifetime management

Summary

Writing a class starts with design

A well designed class can be used like a built in type

A well designed class hides its implementation details

- Leaves the developer free to change them without breaking other code
- Saves those who use the class from having to remember to do things

Using one .h and one .cpp file per class is a good practice