Abstraction and OOP

Tiziana Ligorio
tligorio@hunter.cuny.edu

Today's Plan

Announcements

Recap

Abstraction

OOP

Announcements

- Can’t use different email for Gradescope, Blackboard and Gradescope will link to the same hunter email, but you can forward it to your preferred email

- If you already have a Gradescope account from previous semesters you still need to make sure you have access to this course

- There is a handful of you who have never used Gradescope before. Please try to submit project 1 ASAP and seek help from TAs in lab if you have problems

Announcements

Application for the summer **2019 Tech Talent Pipeline Residency** is now live
Applications will be accepted through **Friday, March 1, 2019** at 11:59 PM; however, you are strongly encouraged to apply by the priority deadline of **Friday, February 15, 2019**.

What is it?: http://www.techtalentpipeline.nyc/cs-doubling/

To apply now, click the link below.

Apply Today →

https://cunyhunter.co1.qualtrics.com/jfe/form/SV_bNlA08EDYSsn03z

For more information about this opportunity, we encourage you to RSVP to attend an upcoming information session on
***Wednesday, February 13** or **February 20** from **1:30-3PM** in **Room 705, West Building**.*

Recap

Minimize complexity

Simplify complex program to manageable level

Break down into smaller problems

Isolate functionalities

Minimize and control interactions

So how do we do this?

Abstraction

Abstraction Example

Abstraction Example

You always use them, switch from one to another seamlessly and probably don’t think too much about them

Abstraction Example

Easy to use

Come in all shapes and sizes

Can have different complex mechanisms
(Laser, laserjet, Inkjet, Dot matrix …)

What is a printer?

What is a printer?

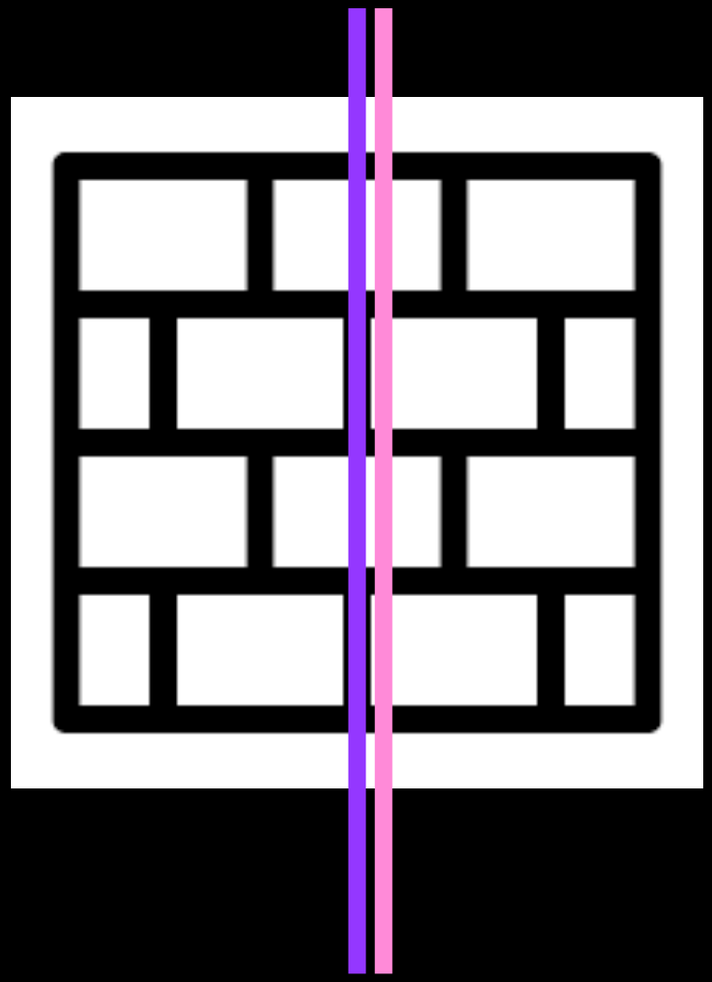
A printer reproduces graphics or text on paper

Separate functionality from implementation
(i.e. what can be done from how it's actually done)

Wall of Abstraction

Information barrier between device (program) use and how it works

Painstaking work to design technology and implement printers



Press button
Or
Send print job from application

Design and implementation

Usage

Abstractions are imprecise

A printer reproduces graphics or text on paper

Wall of abstraction between *implementer* and *client*

How does client know how to use it?

Abstractions are imprecise

A printer reproduces graphics or text on paper

Wall of abstraction between *implementer* and *client*

How does client know how to use it?

Provide an ***interface***

In Software Engineering typically a set of ***attributes*** (or properties) and a set of ***actions***

Lecture Activity

Attributes (data):

Designing the interface:
think about what the user needs to do / know about

Actions (operations):

Interface for Printer

Attributes (data):

Ink level

Paper level

Error codes

Actions (operations):

Print

Rotate (landscape/portrait)

Color / Black & White

Information Hiding

Later
it will always mean
software

Interface —> ***client*** doesn't have to know about the inner workings

Actually client shouldn't know of or *have* access to implementation details

It is dangerous to allow clients to bypass interface and directly modify *objects*

Safe Programming

Reasons for Information Hiding

Harmful for client to tamper with someone else's implementation (*code*)

Reduces flexibility and modifiability by locking implementation in place

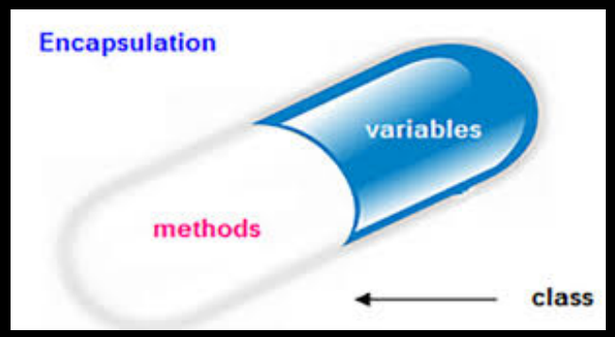
Increases complexity of interactions between modules

Object Oriented Analysis and Design

Principles of Object Oriented Programming (OOP)

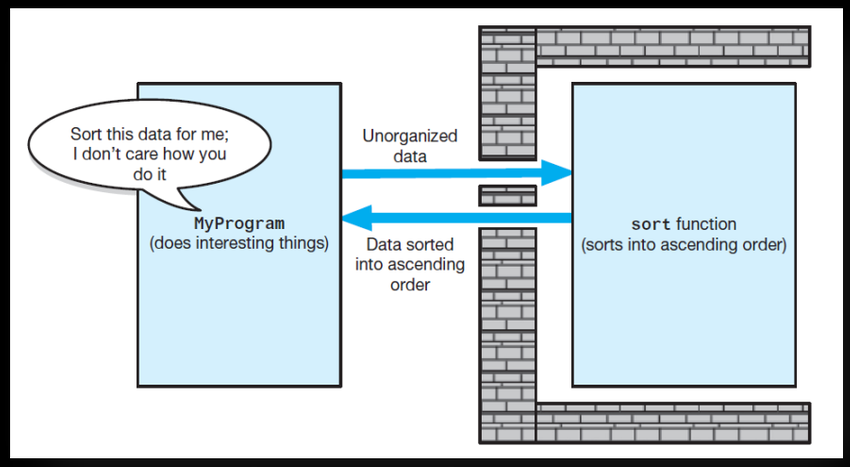
Encapsulation

Objects combine data and operations



Information Hiding

Objects hide inner details



Inheritance

Objects inherit properties from other objects

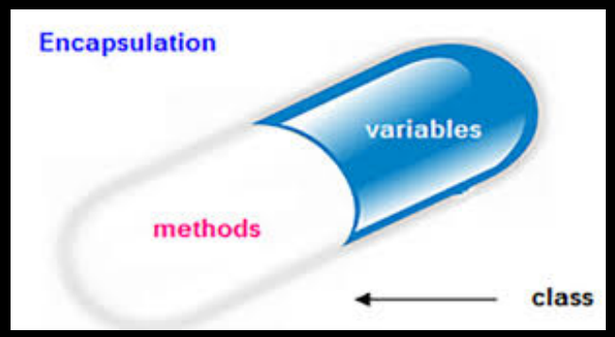
Polymorphism

Objects determine appropriate operations at execution

Principles of Object Oriented Programming (OOP)

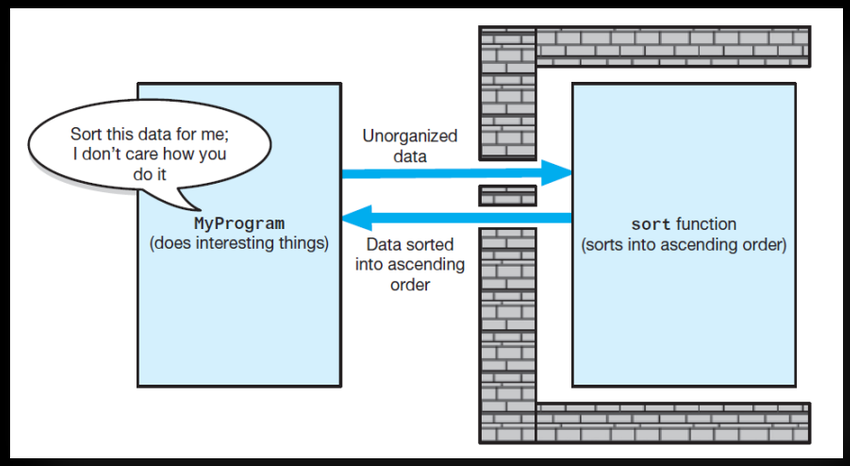
Encapsulation

Objects combine data and operations



Information Hiding

Objects hide inner details



Inheritance

Objects inherit properties from other objects

Polymorphism

Objects determine appropriate operations at execution

Coming soon

Object-Oriented Solution

Use classes of objects

Combine attributes and actions

data members + member functions

Create a good set of modules

Self contained unit of code

Class

Language's mechanism for

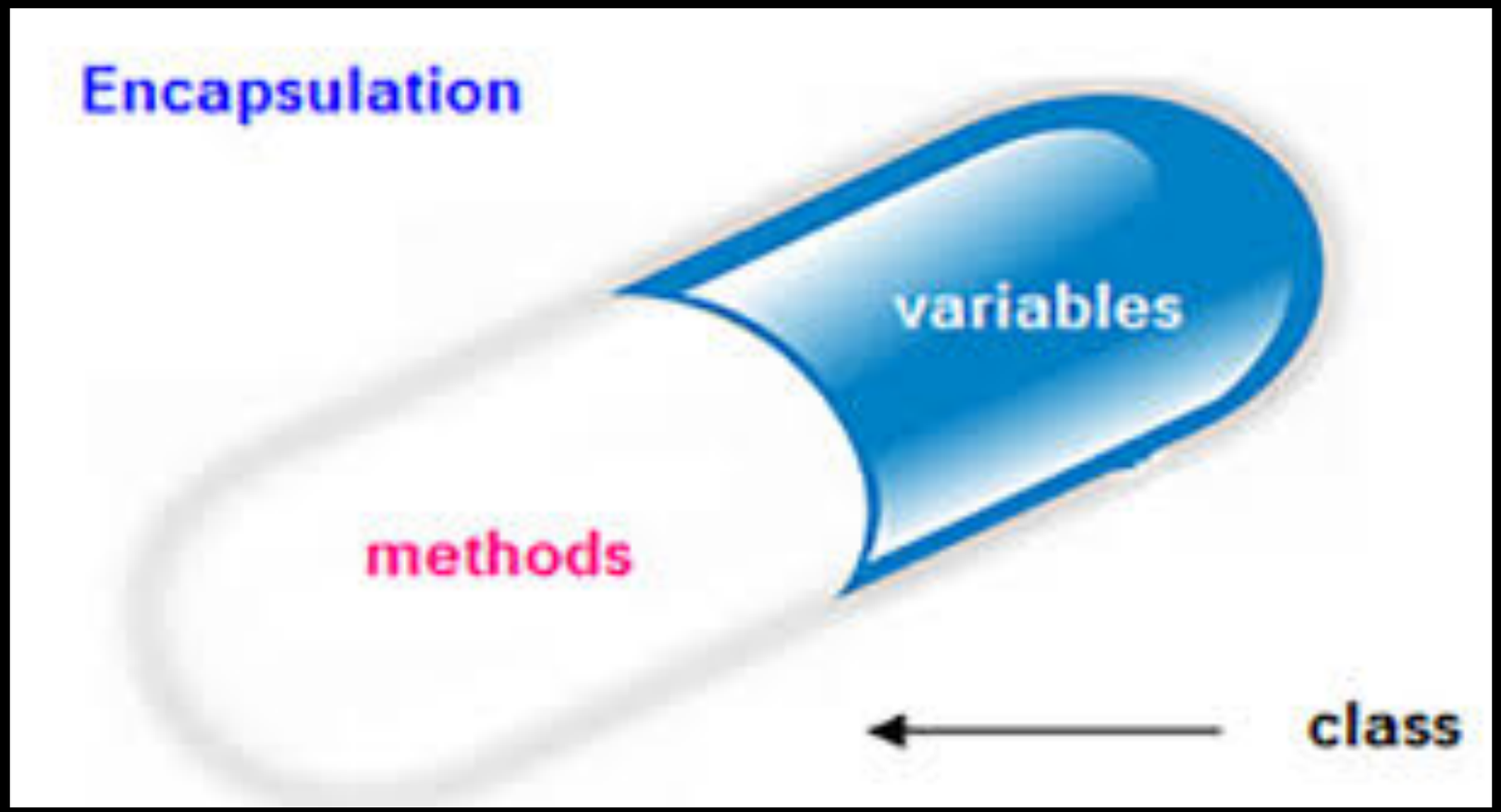
Encoding abstraction

Enforce encapsulation

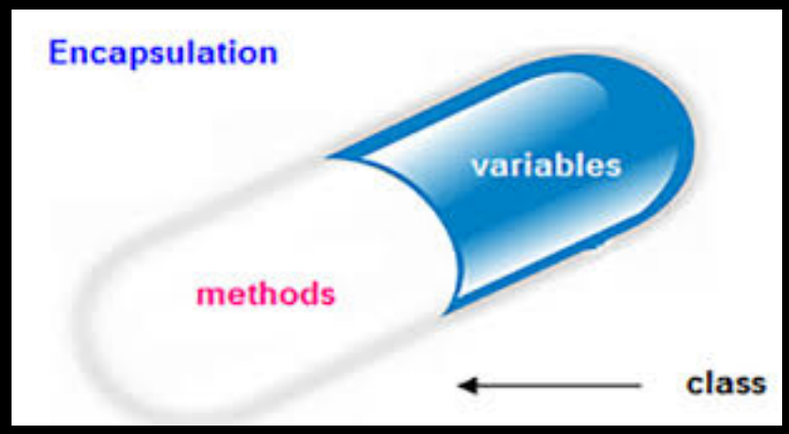
Separate interface from implementation

A ***user-defied* data type** that bundles together data and operations on the data

Encapsulation


Encapsulation
variables
methods
class

Class




```
class SomeClass
{
    access_specifier    // can be private, public or protected

        data_members        // variables used in class

        member_functions    // methods to access data members

}; // end SomeClass
```

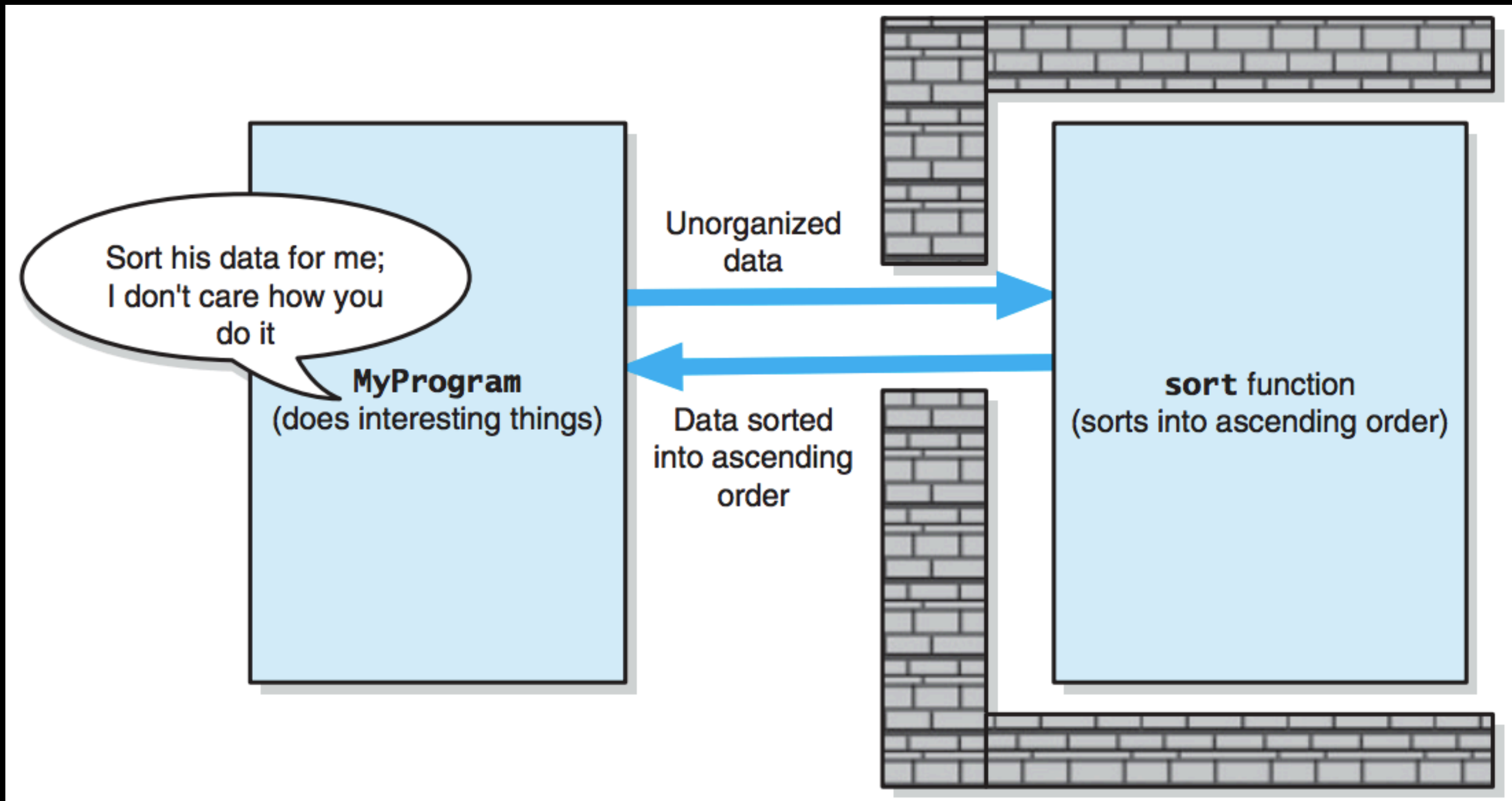
Information Hiding

Class


```
class SomeClass
{
    public:
        // public data members and member functions go here

    private:
        // private data members and member functions go here

}; // end SomeClass
```

Access specifier

Access specifier


Sort his data for me;
I don't care how you
do it
MyProgram
(does interesting things)
Unorganized
data
Data sorted
into ascending
order
sort function
(sorts into ascending order)

Interface

SomeClass.hpp
(same as SomeClass.h)

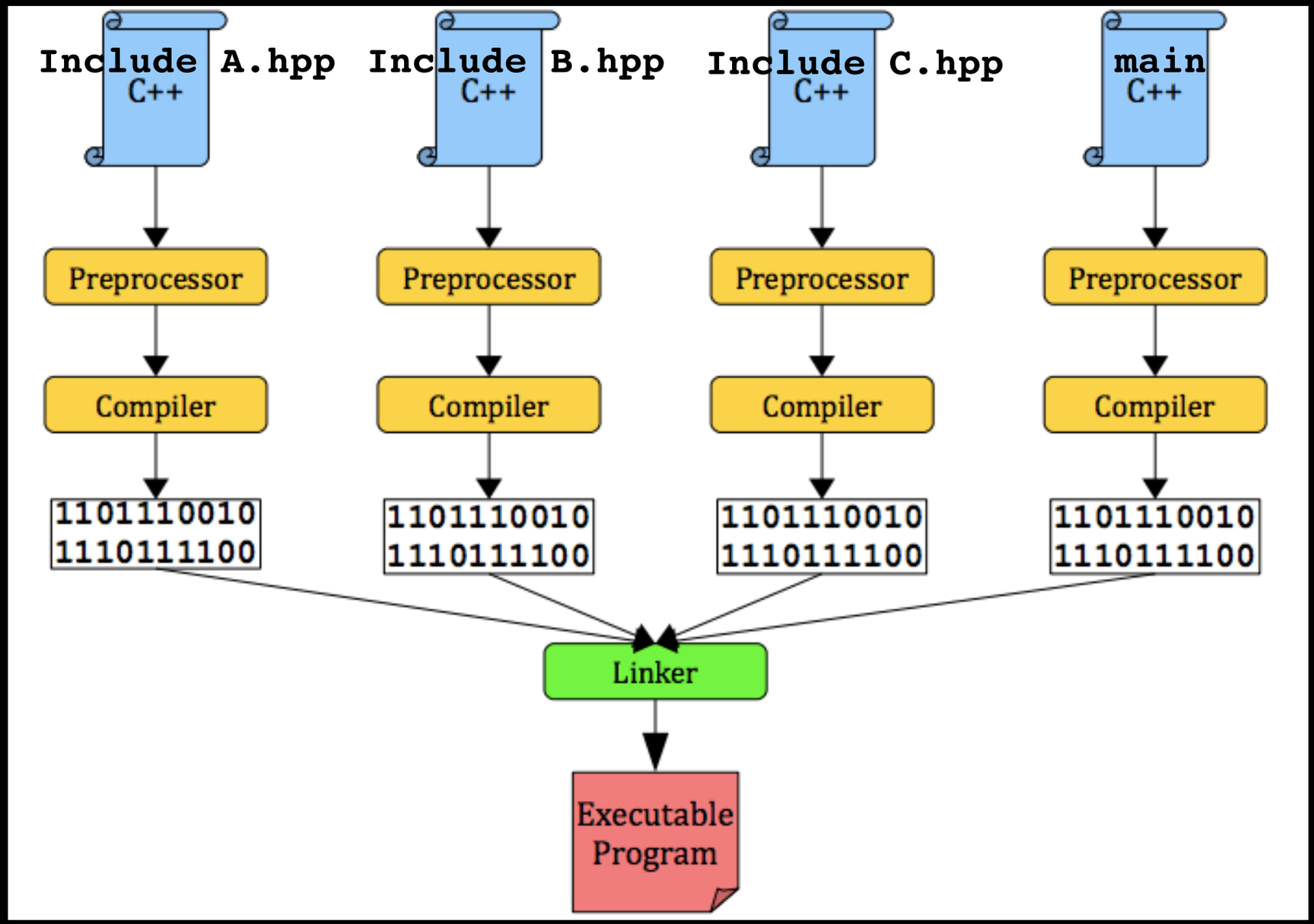
```
#ifndef SOME_CLASS_H_
#define SOME_CLASS_H_

#include <somelibrary>
#include "AnotherClass.h"


class SomeClass
{

public:
    SomeClass(); //Constructor
    int methodOne();
    bool methodTwo();
    bool methodThree(int
                     someParameter);


private:
    int data_member_one_;
    bool data_member_two_;

};    //end SomeClass

#endif
```

Implementation

SomeClass.cpp

```
#include "SomeClass.hpp"

SomeClass::SomeClass()
{
    //implementation here
}


int SomeClass::methodOne()
{
    //implementation here
}


bool SomeClass::methodTwo()
{
    //implementation here
}


bool SomeClass::methodThree(int

someParameter)
{
    //implementation here
}
```

Separate Compilation




```
g++ -o program_name A.cpp B.cpp C.cpp main.cpp
```

Class Recap

Access specifiers: determines what data or methods are public, private or protected (more on protected later)

Data members: the attributes/data

Member functions: the operations/actions available on the data

- **Mutator functions**: modify data members

- **Accessor functions**: retrieve the value of data members

  Use `const` to enforce/indicate it **will not modify the object**

  e.g. `string getName() const;`

Constructor(s)

Destructor

Take care of what happens when object goes in/out of scope

Class / Object

A class is a ***user-defied*** **data type** that bundles together data and operations on the data

Class: type (like `int`)

Object: instantiation of the class (like `x` - as in `int x`)

Just like variables, objects have a ***scope***

- they are born (instantiated/constructed)

- they are killed (deallocated/destroyed)

Object instantiation and usage

```
#include “SomeClass.h”


int main()
{


    SomeClass new_object(); /instantiation of SomeClass calls constructor


    int my_int_variable = new_object.methodOne();
    bool my_bool_variable = new_object.methodTwo();


    return 0;


}  //end main
```

object (dot) method
calls the member function for this object

Constructors

DECLARATION / INTERFACE:

```
class SomeClass
{
    public:
        SomeClass();                        //default constructor
        SomeClass( parameter_list );        //parameterized constructor
        // public data members and member functions go here

    private:
        // private members go here

};// end SomeClass
```

Default Constructor automatically supplied by compiler if not provided.

If only Parameterized Constructor is provided, compiler WILL NOT supply a Default Constructor and class MUST be initialized with parameters

Executed when object is declared.
Initializes member variables and does whatever else may be required at instantiation

Constructors

DECLARATION / INTERFACE:

```
class SomeClass
{
    public:
      SomeClass();                          //default constructor
      SomeClass( parameter_list );          //parameterized constructor
        // public data members and member functions go here


    private:
        // private members go here

};// end SomeClass
```

IMPLEMENTATION:

```
SomeClass::SomeClass()
{
}// end default constructor
```

OR:

```
SomeClass::SomeClass():
member_var1_(initial value),
member_var2_(initial value)
{
}// end default constructor
```

```
SomeClass::SomeClass(type parameter_1, type parameter_2):
member_var1(parameter_1), member_var2(parameter_2)
{
}//end parameterized constructor
```

Member Initializer List

Destructor

Default Destructors automatically supplied by compiler if not provided.

Must provide Destructor to free-up memory when SomeClass does dynamic memory allocation

```
class SomeClass
{
    public:
        SomeClass();
        SomeClass( parameter_list );//parameterized constructor
        // public data members and member functions go here
        ~SomeClass(); // destructor

    private:
        // private data members and member functions go here

};// end SomeClass
```

Executed when object goes out of scope. Does mostly clean-up work, usually necessary to free-up dynamically allocated memory

Lecture Activity

Write the interface for a printer class:

```
class Printer
{
    access_specifier    // can be private, public or protected

        data_members          // variables used in class

        member_functions     // methods to access data members

}; // end Printer
```

Interface as Operation Contract

Documents use and limitations of a class and its methods

Function Prototype and Comments **MUST** specify:

- Data flow
  - Input => parameters
  - Output => return
- Pre and Post Conditions

Operation Contract

In Header file:

```
/** sorts an array into ascending order
// @pre 1 <= number_of_elements <= MAX_ARRAY_SIZE
// @post an_array[0] <= an_array[1] <= ...
//       <= an_array[number_of_elements-1];
//       number_of_elements is unchanged
// @param an_array of values to be sorted
// @param number_of_elements contained in an_array
// @return true if an_array is sorted, false otherwise
*/
bool sort(const int& an_array[], int number_of_elements);
```

Function prototype

Function prototype

Back to some principles of Software Engineering

Unusual Conditions

Values out of bound, null pointer, inexistent file…

How to address them (strive for fail-safe programming):

State it as precondition

Return value that signals a problem

Typically a boolean to indicate success or failure

Throw an exception (later in semester)

Solution guidelines

Many possible designs/solutions

Often no clear best solution

"Better" solution principles:

High cohesion

Loose Coupling

Cohesion

Performs one well-defined task

Well named => self documenting

e.g. `sort()`

SORT ONLY!!!
E.g. If you want to output, do that in another function

Easy to reuse

Easy to maintain

Robust (less likely to be affected by change)

Coupling

Measure of ***dependence (interactions)*** among modules

i.e. share data structures or call each other's methods

Minimize but cannot eliminate

Objects must collaborate!!!

Reduce Coupling

Methods should only call other methods:

- defined within same class
- of argument objects
- of objects created within the method
- of objects that are data members of the class

Control Interaction

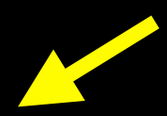
Pass-by-value

```
bool my_method(int some_int);
```

Pass-by-reference if need to modify object

```
bool my_method(ObjectType& some_object);
```

Pass-by-constant-reference if function doesn't modify object



```
bool my_method(const ObjectType& some_object);
```

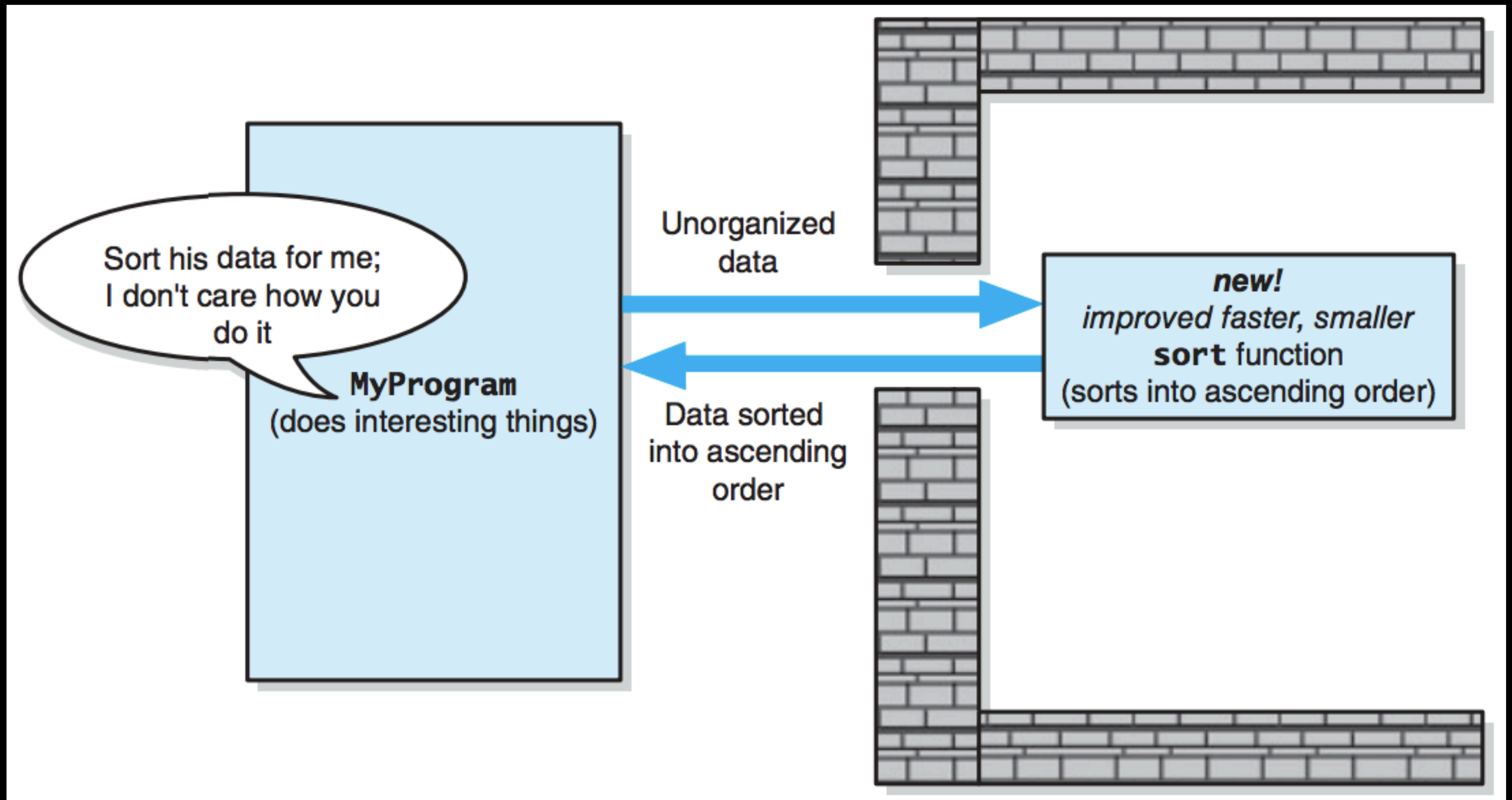
Modifiability

No global variables EVER!!!

Named Constants

```
const int NUMBER_OF_MAJORS = 160;
int scores [NUMBER_OF_MAJORS];
for(index = 0 through NUMBER_OF_MAJORS - 1)
        Process
```

Modifiability



Readability

Write self-commenting code

Important to strike balance btw readable code and comments

- don't write the obvious in comments

```
x  += m * v1; //multiply m by v1 and add result to x
```

Use descriptive names for variables and methods

```
/**@return: the average of values in scores*/
double getAverage(double* scores, int size)
{
   double total = 0;

   for (int i = 0; i < size; ++i)
   {
      total += scores[i];
   }

   return ( total / (double)size );
}
```

Naming Conventions

https://google.github.io/styleguide/cppguide.html

http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rl-comments

```
string my_variable;
```

or

```
string myVariable;
```

Classes ALWAYS start with capital

```
MyClass
```

In this course I will strive for:

```
class MyClass
MyClass class_instance;
string my_variable;
string my_member_variable_;
void myMethod();
int MY_CONSTANT;
```

Be consistent!!!

Project 1

Find it on the course schedule

Trivial — write a simple class: ClassMember (a person in this course)

Review / Establish Baseline

Submit on Gradescope

2 files total

I give you the interface (CourseMember.hpp)

You write and submit the implementation (CourseMember.cpp)

Testing: write your own main function to test each method

- **INCREMENTALLY!!!!**
- **Think about test cases / edge cases when appropriate**

Submit only 1 file

Multi-file compilation with g++ in Programming Rules document