



CLASSY

ADAM SWEENEY
CS 211

INTRODUCTION

- As we saw with structures, relevant data can be gathered together even if they are of different types
- But we ended up with some functions that now go with the structure wherever it goes, leaving us in the same predicament
 - Having to cart around disparate items
 - Keeping track of many distinct things
- Turns out we can package the functions as well

AGENDA

- Classes
- Defining & implementing classes
- Using classes
- Collecting the basic principles of classes

A decorative graphic on the left side of the slide consisting of two parallel, wavy vertical lines. The inner line is yellow and the outer line is white, both set against a dark brown background.

CLASSES

THE DIFFERENCE (-ISH)

- A class allows us to collect data AND functions under a common name
- The addition of functions changes the way we approach our code writing a bit

A BASIC MONEY CLASS

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```

- We can see some struct-ural similarities between a class and struct
- But there's a lot of new stuff going on as well

LOOKING AT THE OUTSIDE

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```

- We'll start on the outside
- If it said `struct` instead of `class`, it would be identical

LOOKING INSIDE

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```

- We see the data under the word private
- Indentation suggests that the data "belongs" to private
- The functions must then "belong" to public
- Two of the functions have a name but no return type

PUBLIC AND PRIVATE?

- In a nutshell, the public section is accessible outside of the class and the private section is not
- If data is worth keeping in a class, it is worth keeping safe
- Class data should only be accessed and manipulated in a controlled manner
 - This ensures that objects always contain meaningful data
 - Also known as ‘preserving the class invariant’
- The public section how the class can be used
 - Essentially the API (Application Programming Interface)

A BIT OF VOCABULARY

- A class definition is called the... *definition*
- The collected function bodies are called the *implementation*
- Any function that belongs to a class is called a *member*
 - Can have member data or member functions
- Classes are defined, and we call variables of a class type *objects*
- Recall procedural abstraction from functions
 - With classes, the same idea is called encapsulation
 - We have a new term because there are differences
 - Both in a what a class is vs. a function, and the scope of the term

A decorative wavy line in yellow and white on the left side of the slide.

DEFINING & IMPLEMENTING CLASSES

BUILDING A CLASS

- When we declare objects, we expect them to valid, usable, and consistent
- We can think of objects as being built when declared
- We can (and should) control that process

CONSTRUCTORS

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```

- The first two functions are called constructors
 - No return
 - Named after the class
- Constructors are special class member functions that build objects

CONSTRUCTORS, CONT.

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```

- The first constructor is also known as the default constructor
 - The default constructor never has parameters
- If we do not declare **any** constructors, the compiler will provide a default for us
- Generally better to define it ourselves

OTHER CONSTRUCTORS

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```

- The second constructor is called a parameterized constructor
- As the name implies, the parameter lists are populated
 - Allows us to initialize with specific values

MORE OTHER CONSTRUCTORS

- There are a couple other types of constructors, but they are beyond the scope of this course

CONSTRUCTORS, CONCLUDED

- It is likely that you will define more than one constructor for a class
- It is good practice to always provide the default constructor yourself
 - Made less tedious when using default member initialization and `= default`
- The amount of constructors you write depends on how many valid ways you wish to allow for initializing objects

OTHER CLASS FUNCTIONS

```
class Money {  
public:  
    Money();  
    Money(int d, int c);  
    void print();  
private:  
    int m_cents;  
};
```


- `print()` looks a lot more like a function that we're familiar with
- When writing class member functions, we need to think about them a little bit differently
- We need to consider whether it belongs in the class
- We need to keep in mind the calling-object

A decorative wavy line in yellow and white, resembling a stylized lightning bolt or a jagged edge, runs vertically along the left side of the image.

USING CLASSES

NOT SO BAD

- Once you've defined and implemented a class, using it is not so bad
- A class creates a new type you can use
- Not so different from using `string` or `vector` at this point



COLLECTING THE BASIC PRINCIPLES OF CLASSES

OOP

- Beginning to learn about Object-Oriented Programming (OOP) included a lot of new vocabulary and theory
- Up to this point, we've discussed some best practices when writing code
 - OOP takes this to a new level
- This section just gathers the OOP principles and advice we've seen so far
 - Adds one new practice, as well

ENCAPSULATION

- Encapsulation is to OOP as Procedural Abstraction is to functions
- Our classes should be self-contained, independent entities
- Encapsulation guides nearly all other OOP principles

DATA MEMBERS SHOULD BE PRIVATE

- If we are storing data, the class should retain 100% control
 - This means not allowing direct access to the data
- Some functions may also be private
 - Depends on how you want/need the class to be accessed/manipulated

SO HOW DOES THE DATA CHANGE?

- Another question, specific to our example, could be “How do I know how much money I have?”
- The answer is through specific public functions
- Accessors and Mutators
 - Also called getters and setters
- These are public functions that allow the private data to be directly manipulated
 - Being forced to go through a public function allows the class to exert control over the process
 - Validate changes, perform sanitization, etc.

WRITE THE DEFAULT CONSTRUCTOR

- Relying on the compiler-generated default can be dangerous
 - If we ever add another constructor, it's likely that we will forget about the default
 - Things will likely break
 - The compiler can't read our minds and doesn't know what good default values are
- Unless you require that all objects must be initialized, write the default constructor yourself

CALLING OBJECT

- You won't get far in OOP if this evades you
- Class functions can only be accessed through an object
- The object being used to call a class function is known as the calling object
- ...Since it's the object calling the function
- Understanding this allows us to write our classes