

# Designing Abstractions

```
/* C++ */
class Counter {
public:
    void increment();
    int total() const;
private:
    int value = 0;
};

void Counter::increment() {
    value++;
}

int Counter::total() const {
    return value;
}
```

```
/* Java */
public class Counter {
    public void increment() {
        value++;
    }

    public int total() {
        return value;
    }

    private int value = 0;
}
```

```
# Python
class Counter:
    def __init__(self):
        self.value = 0

    def increment(self):
        self.value += 1

    def total(self):
        return self.value
```

```
// JavaScript
class Counter {
    constructor() {
        this.value = 0;
    }

    total() {
        return this.value;
    }

    increment() {
        this.value++;
    }
}
```

# Designing Abstractions

# *ab·strac·tion*

[...]

the process of considering something independently of its associations, attributes, or concrete accompaniments.

Source: Google

Vector Map

Set Queue

Building a rich vocabulary of abstractions makes it possible to ***model and solve*** a wider class of problems.

## ***Question One:***

How do we create new abstractions to model ideas not precisely captured by the standard container types?

## ***Question Two:***

How do the abstractions we've been using so far work, and how can we use that knowledge to build richer abstractions?

# Classes in C++

# Classes

- Vector, Stack, Queue, Map, etc. are **classes** in C++.
- Classes contain
  - an **interface** specifying what operations can be performed on instances of the class.

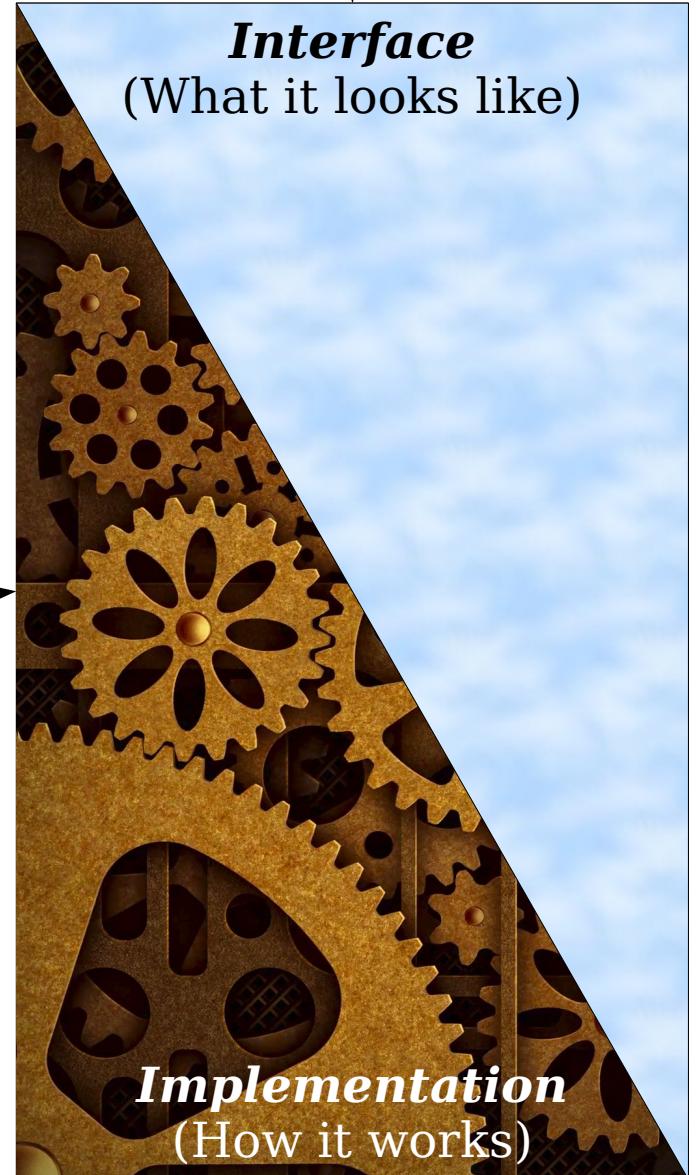
***Interface***  
(What it looks like)

# Classes

- Vector, Stack, Queue, Map, etc. are **classes** in C++.
- Classes contain
  - an **interface** specifying what operations can be performed on instances of the class, and
  - an **implementation** specifying how those operations are to be performed.

Where  
we've been

**Interface**  
(What it looks like)



Where we're  
going

**Implementation**  
(How it works)

# Creating our own Classes

# Random Bags

- A **random bag** is a data structure similar to a stack or queue. It supports two operations:
  - **add**, which puts an element into the random bag, and
  - **remove random**, which returns and removes a random element from the bag.
- Random bags have a number of applications:
  - Simpler: Shuffling a deck of cards.
  - More advanced: designing mazes. (*Curious how?  
Come talk to me after class!*)
- Let's go create our own custom RandomBag type!

# Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a ***header file*** (typically suffixed with .h) describing what operations the class can perform and what internal state it needs.
  - Create an ***implementation file*** (typically suffixed with .cpp) that contains the implementation of the class.
- Clients of the class can then include the header file to use the class.

# What's in a Header?

# What's in a Header?

`#pragma once`

This is called an ***include guard***. It's used to make sure weird things don't happen if you include the same header twice.

Curious how it works?  
Come talk to me after  
class!

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
};
```

This is a ***class definition***. We're creating a new class called RandomBag. Like a struct, this defines the name of a new type that we can use in our programs.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {
```

```
};
```

***Don't forget to add  
this semicolon!*** You'll  
get some Hairy Scary  
Compiler Errors if you  
leave it out.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:  
  
private:  
};
```



# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:  
  
private:  
};
```

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:
```



```
private:
```

```
};
```

The ***public interface*** specifies what functions you can call on objects of this type.

Think things like the `Vector`'s `.add()` function or the `string`'s `.find()`.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:  
};  
  
private:  
};
```

The ***public interface*** specifies what functions you can call on objects of this type.

Think things like the `Vector`'s `.add()` function or the `string`'s `.find()`.

The ***private implementation*** contains information that objects of the class type will need in order to do their job properly. This is invisible to people using the class.

# What's in a Header?

```
#pragma once
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
  
private:  
};
```

These are ***member functions*** of the RandomBag class. They're functions you can call on objects of the type RandomBag.

All member functions need to be declared in the class definition. We'll implement them in our .cpp file.

# What's in a Header?

```
#pragma once
```

```
#include "vector.h"

class RandomBag {
public:
    void add(int value);
    int removeRandom();

private:
    Vector<int> elems;
};
```

This is a ***data member*** of the class. This tells us how the class is implemented. Internally, we're going to store a `Vector<int>` holding all the elements. The only code that can access or touch this `Vector` is the `RandomBag` implementation.

# What's in a Header?

```
#pragma once

#include "vector.h"

class RandomBag {
public:
    void add(int value);
    int removeRandom();

private:
    Vector<int> elems;
};
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
  
private:  
    Vector<int> elems;  
};
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
    int size();  
    bool isEmpty();  
private:  
    Vector<int> elems;  
};
```

```
class RandomBag {  
public:  
    void add(int value);  
    int removeRandom();  
    int size() const;  
    bool isEmpty() const;  
private:  
    Vector<int> elems;  
};
```

# Your Action Items

- ***Rest and Recharge***
  - You've earned it!
- ***Read Chapter 6 of the textbook.***
  - There's a ton of goodies in there about class design that we'll talk about later on.
- ***Start Assignment 5***
  - Aim to complete the first two parts of the assignment by Wednesday.

# Next Time

- ***Dynamic Allocation***
  - Where does memory come from?
- ***Constructors and Destructors***
  - Taking things out and putting them away.
- ***Implementing the Stack***
  - Peering into our tools!