Designing Abstractions

Apply to Section Lead!

Application Online
Due Thursday, February 16th, 11:59PM

Want to check out Treehacks?
A little nervous about it?
Don't know anyone else who's doing it?

Come to HACK 101!

Learn how to be successful at a hackathon!
Meet teammates for Treehacks!
Start the ideation process for your project!

RSVP **HERE**
hosted by Black in CS

Assignment 4

- Assignment 4 is due a week from today.
  - Aim to be done with Doctors Without Orders and Disaster Planning by the end of the evening.
  - Try to complete DNA Detective by Monday.
- ***Remember:*** The midterm is right after Assignment 4 comes due, so *using late days here is a Really Bad Idea!*

Midterm Logistics

- Our Midterm is ***Tuesday, February 21st*** from 7:00PM – 10:00PM, location TBA.
- Topic coverage:
  - Lectures 00 – 12 (up through and including sorting and big-O notation).
  - Assignments 0, 1, 2, 3, and 4.
- Email Anton by Monday at 5:00PM to arrange for an alternate time. ***No alternate exam time requests will be accepted after that point.***
- Students with OAE accommodations: if you haven't yet reached out to us, please do so ASAP.

Midterm Logistics

- Exam is closed-book, closed-computer, and limited-note. You can have a single 8.5” × 11” sheet of notes with you when you take the exam.
  - It can be hand-written, typed, calligraphed, mimeographed, etc.
  - ***Recommendation:*** hand-write your own notes sheet. Start off by writing notes without regard to length, then pare it down to a single sheet.
- We’ll provide a **C++ library reference sheet** with the exam, so you shouldn’t need to cram all that into your notes sheet.

Practice Midterm

- We will be holding a practice midterm exam next Monday, February 13th here in Hewlett 200 from 7:00PM – 10:00PM.
- ***You should plan to attend this practice exam unless you have an immovable conflict.*** The first time you write code on paper should not be during the exam itself.
- Can't make it? The practice exam will be posted on the course website, along with solutions.

Extra Practice

- Need some extra practice?
  - ***Work through the section handouts***. We deliberately put way more questions on them than you can handle in section so that you can use them as a study resource.
  - ***Work through the textbook practice problems***. The chapter exercises are a great way to sharpen your skills.
  - ***Revisit old assignments***. It'll be a lot easier to code them up the second time around!
- Still not enough practice? Contact us and we can try to put some more materials together.

Onward and Forward!

Designing Abstractions

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 we can use to model and solve larger problems?

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.
  - An ***implementation*** specifying how those operations are to be performed.
- To define our own classes, we must define both the interface and the implementation.

Random Bags

- A ***random bag*** is a data structure similar to a stack or queue. It supports two operations:
  - ***Add***, which adds an element to 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: Training self-driving cars to park and change lanes. *(Curious how? Come talk to me after class!)*

Let's Code it Up!

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?

```
#ifndef RandomBag_Included
#define RandomBag_Included




#endif
```

This boilerplate code 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?

```
#ifndef RandomBag_Included
#define RandomBag_Included



class RandomBag {



};

#endif
```

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?

```
#ifndef RandomBag_Included
#define RandomBag_Included



class RandomBag {









};

#endif
```

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?

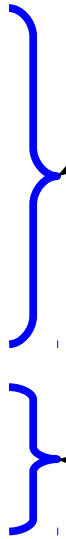
```
#ifndef RandomBag_Included
#define RandomBag_Included


class RandomBag {
public:

private:

};

#endif
```



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

Think things like the `Vector`'s `.add()` function or the `TokenScanner`'s `.nextToken()`.

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?

```
#ifndef RandomBag_Included
#define RandomBag_Included



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



private:

};

#endif
```

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?

```
#ifndef RandomBag_Included
#define RandomBag_Included

#include "vector.h"

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




private:
   Vector<int> elems;
};

#endif
```

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?

```
#ifndef RandomBag_Included
#define RandomBag_Included

#include "vector.h"

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




private:
   Vector<int> elems;
};

#endif
```

```
#include "RandomBag.h"
```

If we're going to implement the `RandomBag` type, the .cpp file needs to have the class definition available. All implementation files need to include the relevant headers.

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



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

```cpp
#include "RandomBag.h"


void RandomBag::add(int value) {

}
```

The syntax

`RandomBag::add`

means "the `add` function defined inside of `RandomBag`." The `::` operator is called the ***scope resolution operation*** in C++ and is used to say where to look for things..

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



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

```
#include "RandomBag.h"


void RandomBag::add(int value) {

}
```

If we had written something like this instead, then the compiler would think we were just making a free function named add that has nothing to do with `RandomBag`'s version of `add`. That's an easy mistake to make!

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



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

```cpp
#include "RandomBag.h"


void RandomBag::add(int value) {
   elems += value;
}
```

We don't need to say what `elems` is. The compiler knows we're inside `RandomBag`, and so it knows that this means "the current `RandomBag`'s collection of elements."

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



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

```cpp
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (elems.isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, elems.size() - 1);
   int result = elems[index];
   elems.remove(index);

   return result;
}
```

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



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

```cpp
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (elems.isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, elems.size() - 1);
   int result = elems[index];
   elems.remove(index);

   return result;
}
```

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

   bool isEmpty();
   int  size();

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

```cpp
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (elems.isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, elems.size() - 1);
   int result = elems[index];
   elems.remove(index);

   return result;
}

int RandomBag::size() {
   return elems.size();
}
```

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

   bool isEmpty();
   int  size();

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

```
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (elems.isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, elems.size() - 1);
   int result = elems[index];
   elems.remove(index);

   return result;
}

int RandomBag::size() {
   return elems.size();
}

bool RandomBag::isEmpty() {
   return size() == 0;
}
```

This code calls our own `size()` function. The class implementation can use the public interface.

```
mBag {

(int value);
oveRandom();

   bool isEmpty();
   int  size();

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

```cpp
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, size() - 1);
   int result = elems[index];
   elems.remove(index);

   return result;
}

int RandomBag::size() {
   return elems.size();
}

bool RandomBag::isEmpty() {
   return size() == 0;
}
```

That's such a good idea, let's do this up here as well.

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

   bool isEmpty();
   int  size();

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

```cpp
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, size() - 1);
   int result = elems[index];
   elems.remove(index);

   return result;
}

int RandomBag::size() {
   return elems.size();
}

bool RandomBag::isEmpty() {
   return size() == 0;
}
```

```cpp
class Random
public:
   void add(int value);
   int  removeRandom();

   bool isEmpty() const;
   int  size() const;

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

This use of the `const` keyword means "I promise that this function doesn't change the object."

```cpp
#include "RandomBag.h"
#include "random.h"

void RandomBag::add(int value) {
   elems += value;
}

int RandomBag::removeRandom() {
   if (isEmpty()) {
      error("Aaaaahhh!");
   }

   int index = randomInteger(0, size() - 1);
   int result = elems[index];
   elems.rem

   return re
}

int RandomBag::size() const {
   return elems.size();
}

bool RandomBag::isEmpty() const {
   return size() == 0;
}
```

We have to remember to put it here too as well!

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

   bool isEmpty() const;
   int  size() const;

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

Your Action Items

- Read Chapter 6 of the textbook.
  - There's a ton of goodies in there about class design that we'll talk about later on.
- Aim to complete the first two parts of Assignment 4 by the end of today.
  - It's probably not a good idea to fall behind on this assignment.
- Aim to complete the first three parts of Assignment 4 by Monday.
  - Proactivity!

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!