Deep Copy, Shallow Copy, Clone

- The default semantics in C++ is *copy semantics*

```cpp
Foo a, b;
a.doThis();
b = a;
b.doThis();
// how do a and b compare?
```

- We expect assignment to be a copy, it is for ints and other assignment/copy operators should mirror this.
  - Repercussions for pointer-based classes?
- By default C++ supports assignment and copy, *but shallow*

```cpp
const Foo& operator = (const Foo& f);
Foo(const Foo& f);
```
Assignment and copy in C++

- If copies are expensive (time, memory, development) we can prevent them
  - Make copy constructor and assignment operator private, see hmap.h, for example
  - What does this do for pointers to objects, copyable?

- How does this work in C++? Java analog?

```cpp
vector<string> vs(3, "hello");
const vector<string> cvs(3, "hello");
cout << cvs[2] << endl;
cvs[2][0] = 'j';
```
Copy/Assignment in Java

- The default semantics in Java is non-copy for objects, copy for primitive types

```java
Foo a, b;
a = new Foo();
b = a;
b.doThis();
```

- Suppose we want a copy of a, what do we do?

- Java has an interface Cloneable, implementers must have:
  ```java
  public Object clone() {...}
  ```
  ```java
Possible to call Object.clone(), default member assignment---shallow copy, not deep
```Object.clone() is protected, but Cloneable classes must make clone public```
Gates, Wires, OO Design

- Building circuits from gates. What’s the difference between a gate and a circuit? What is a circuit?
  - Composite design pattern used for part-is-a-whole

- How do we “wire” a circuit, how do signals flow?
  - Observer/Observable: wire is the observable, gate is the observer, who notifies whom?

- Wire creation and cloning: where is it done, who’s responsible?
  - Factory pattern (WireFactory) and factory method (clone)
  - What about cloning a composite?