Operator =
Ch 11.4 & Appendix F

*me, a C/C++ developer learning java for the first time

Java for dummies

*the laptop

int x;

int foo[] = new int[100];

int foo[] = new int[100];
foo =

What have you done

int foo[] = new int[100];
foo = new int [50];
Highlights

- Overload equals

```cpp
classy x;
classy y;
y=x; // equals operator
```
Review: Copy constructor

To avoid double deleting (crashes program) or multiple pointers looking at the same spot...

We have to redefine the copy constructor if we use dynamic memory

The **copy constructor** is another special constructor (same name as class):

```cpp
Dynamic();
~Dynamic();
Dynamic(const Dynamic &d);
```
Copy constructor: arrays

How would you copy a dynamically created array inside a class?

```
class rng{
private:
  double* array;
public:
  rng();
  rng(const rng &original) //write me!
};
```

What if this was a normal array?

```cpp
rng::rng()
{
  array = new double[10];
  for(int i=0; i < 10; i++)
  {
    array[i] = rand()%100; /0-99
  }
}
```

(see: copyArray.cpp)
Copy constructor vs. '='

There is actually two ways in which you can use the '==' sign...

1. The copy constructor, if you ask for a box on that same line:
   ```
   classy x;
   classy y = x; // copy constructor
   ```

2. Operator overload, if you already have a box when using '==':
   ```
   classy x;
   classy y; // y gets box
   y = x; // equals operator
   ```

(See: copyVsEquals.cpp)
What is the difference between copy and '='?
What is the difference between copy and '='?

“copy” is a constructor, so it creates new boxes.

'=' is changing the value of an existing box.
(but the idea of not sharing is the same address)

The “proper” way to implement '==' is nuanced... see code comments if you care
(See: overloadEquals.cpp)
When using “new” in a constructor, you also should make:

1. Destructor
2. Copy constructor
3. Overload ‘=’ operator

Typically the built-in functions are not sufficient if you use a “new” or ‘*’
Consider the following code:

```cpp
BadPublic test;
test.x = 3;

int* intPtr = &(test.x);
intPtr = test.getX();

BadPublic* bpPtr = &test;
bpPtr = test.getMe();
```

```cpp
class BadPublic {
public:
    int x;
    int* getX();
    BadPublic* getMe();
};
```

How do we write `getX()` and `getMe()`?
Q: It seems you should have information about yourself, but how do you access that?

A: Inside every class, there is a this pointer, that points to yourself

(See: thisSelfPointer.cpp)
typedef

Side note: If you want to rename types, you can do that with `typedef` command:

```
typedef int DefinatelyNotAnInt;  // original name
DefinatelyNotAnInt x;
x=3;
int y = x;
cout << y;
```

If you have always been bothered that we use “double” instead of “real”, go ahead and fix it!