Constructors

- A constructor (ctor) initializes a newly-instantiated object
  - A class can have multiple constructors that differ in parameters
    - Which one is invoked depends on how the object is instantiated
  - A constructor is always invoked when creating a new instance of an object.

- Written with the class name as the method name:
  ```cpp
  Point(const int x, const int y); // constructor
  ```
  - C++ will automatically create a synthesized default constructor if you have no user-defined constructors created for you
    - Takes no arguments and calls the default ctor on all non-"plain old data" (non-POD) member variables
    - Synthesized default ctor will fail if you have non-initialized const or reference data members

Copy Constructors

- C++ has the notion of a copy constructor (cctor)
  - Used to create a new object as a copy of an existing object
  ```cpp
  Point::Point(const Point& copyme) {
    x_ = copyme.x_;  
    y_ = copyme.y_;  
  }
  ```

- Initializer lists can also be used in copy constructors (preferred)
When Do Copies Happen?

- The copy constructor is invoked if:
  - You initialize an object from another object of the same type:
    ```cpp
    Point x; // default ctor
    Point y(x); // copy ctor
    ```
  - You pass a non-reference object as a value parameter to a function:
    ```cpp
    void foo(Point x) { ... }
    Point y; // default ctor
    foo(y); // copy ctor
    ```
  - You return a non-reference object value from a function:
    ```cpp
    void Point:: operator=(const Point& rhs) { 
        if(this != &rhs) { // (1) always check against this
            x_ = rhs.x_; 
            y_ = rhs.y_; 
        } 
        return *this; // (2) always return *this from op=
    }
    Point a; // default constructor
    a = b = c; // works because = return *this
    (a = b = c); // equiv. to above (= is right-associative)
    (a = b = c); // "works" because = returns a non-const
    ```

Overloading the "=" Operator

- You can choose to define the "=" operator
  - But there are some rules you should follow:
    ```cpp
    void foo(Point x) { ... }
    Point y; // default ctor
    foo(y); // copy ctor
    ```

Destructors

- C++ has the notion of a destructor (dtor)
  - Invoked automatically when a class instance is deleted, goes out of scope, etc. (even via exceptions or other causes!)
  - Place to put your cleanup code – free any dynamic storage or other resources owned by the object
  - Standard C++ idiom for managing dynamic resources
    - Slogan: "Resource Acquisition Is Initialization" (RAII)

```cpp
Point::~Point() { // destructor
    // do any cleanup needed when a Point object goes away
    // (nothing to do here since we have no dynamic resources)
}
```