Property Attributes Revisited
Setter Semantics
Setter Semantics

• Specifies the behavior of the setter
Setter Semantics

• Specifies the behavior of the setter
• `assign` - simple assignment (default)
Setter Semantics

• Specifies the behavior of the setter

• **assign** - simple assignment (default)

• **retain** - invoke retain on the object upon assignment
Setter Semantics

- Specifies the behavior of the setter
- **assign** - simple assignment (default)
- **retain** - invoke retain on the object upon assignment
- **copy** - assign a copy of the object
assign

object.propertyName = newValue;

gets converted to:

[object setPropertyName: newValue];
retain

```swift
object.propertyName = newValue;

effectively gets converted to:

[object setPropertyName: [newValue retain]];
```
object.propertyName = newValue;

effectively gets converted to:

[object setPropertyName: [newValue copy]];
retain and copy

• Since retain and copy both increase retain counts, they need to call release on the object if the property gets reassigned

• The property setters do this automatically
retain and copy

• However, the last object set will not get released automatically (since the setter isn't called again, which would release it)

• For retain/copy properties, the properties need to be released upon deallocation of the container object
Sidebar: Empty Class Definition

@interface MyClass : NSObject
@end

@implementation MyClass
@end
• Our objects *normally* return the correct retain count.
  • NSObjects and its decedents, on the other hand, typically return values like: 18446744073709551615
  • Autorelease complicates this even further.
-retainCount lesson:

• Don’t rely on retain counts for anything but debugging your own objects.
• Even then, take them with a grain of salt.
• Take the time to learn proper memory management!
-retainCount Abuse...
bad, bad, bad!

-(void)dealloc
{
    while( [myObject retainCount] != 0 )
    {
        [myObject release];
    }
    [super dealloc];
}
Avoiding Cyclic References

```
Parent is owned (retained) by "Some Object"

Parent

Array of children (retained)

Child

Child retains Parent locking two in cycle
```
Avoiding Cyclic References

To avoid cyclic references, a child object should never retain its parent. (A parent is the creator of the child or is an object holding the child as an instance variable.)

AutoRelease pools, when?
AutoRelease Pool Stack

Autoreleased objects go in the *latest* autorelease pool that was allocated (i.e., they’re stacked).
Garbage Collection

YAY!

...right?
Garbage Collection Downsides

1. **Performance**
   1. Many allocation overhead
   2. Working set larger
   3. Keeps memory “hot”
   4. Usually threaded
Garbage Collection
Downsides (con’t)

2. Object lifetime design considerations.

3. Must maintain a chain of “strong” references.
Common (retain/GC/ARC) Code?

“Don’t. *" -Apple

*(see memory management II lab)*