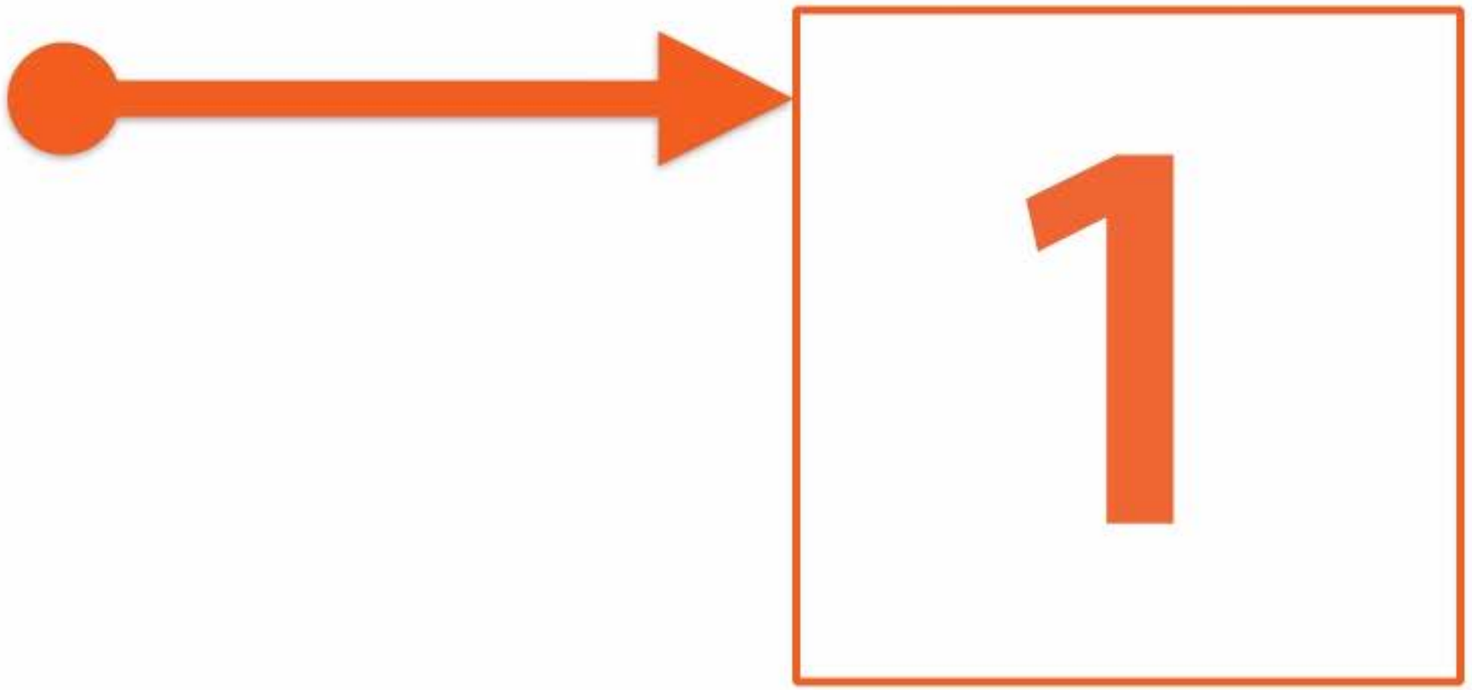
Memory Management

Allen Holub

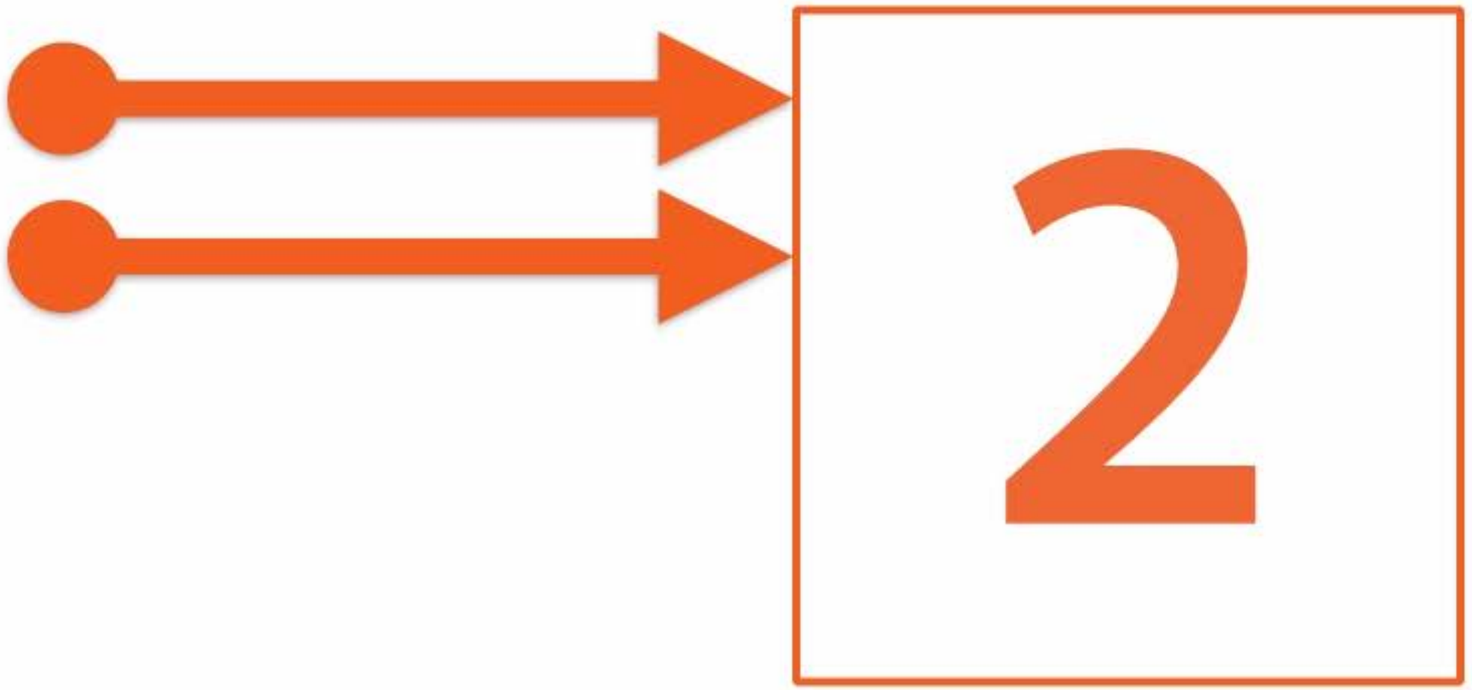
http://holub.com | Allen Holub | @allenholub

Memory

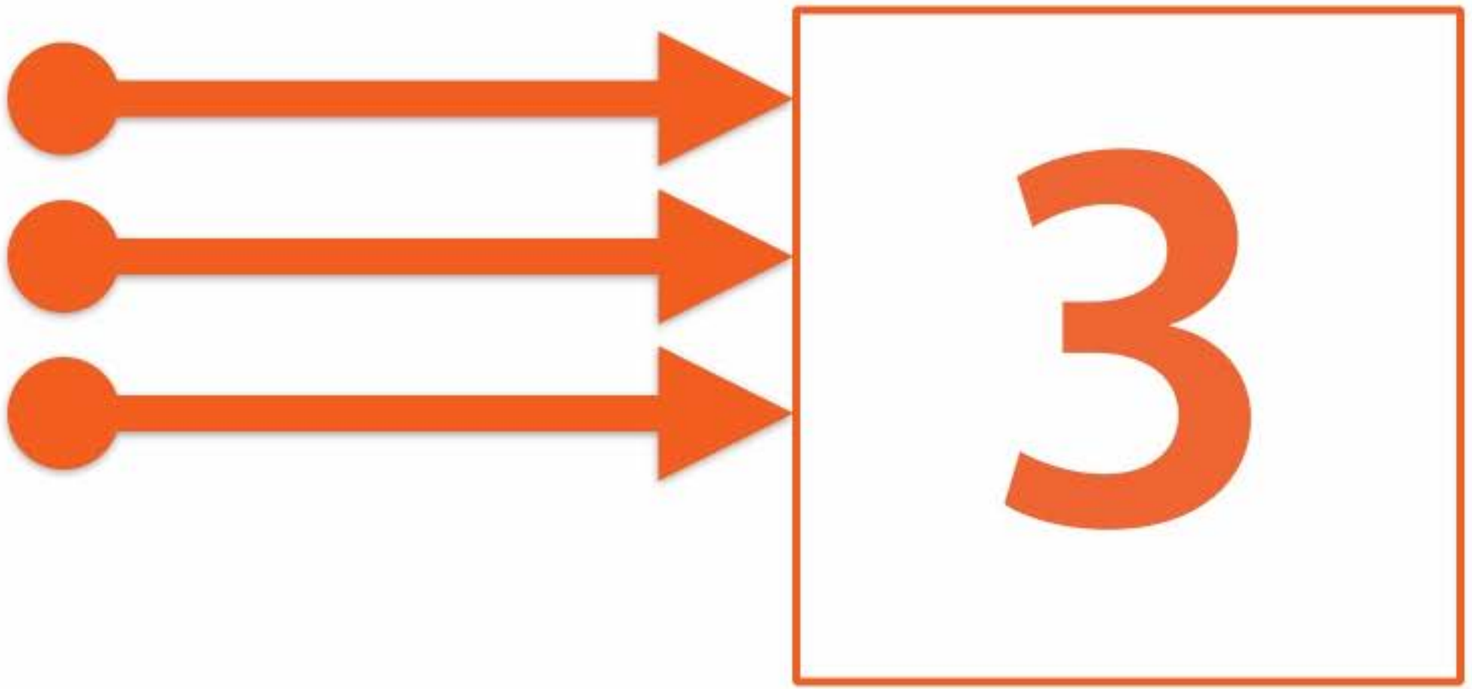
Reference Counting



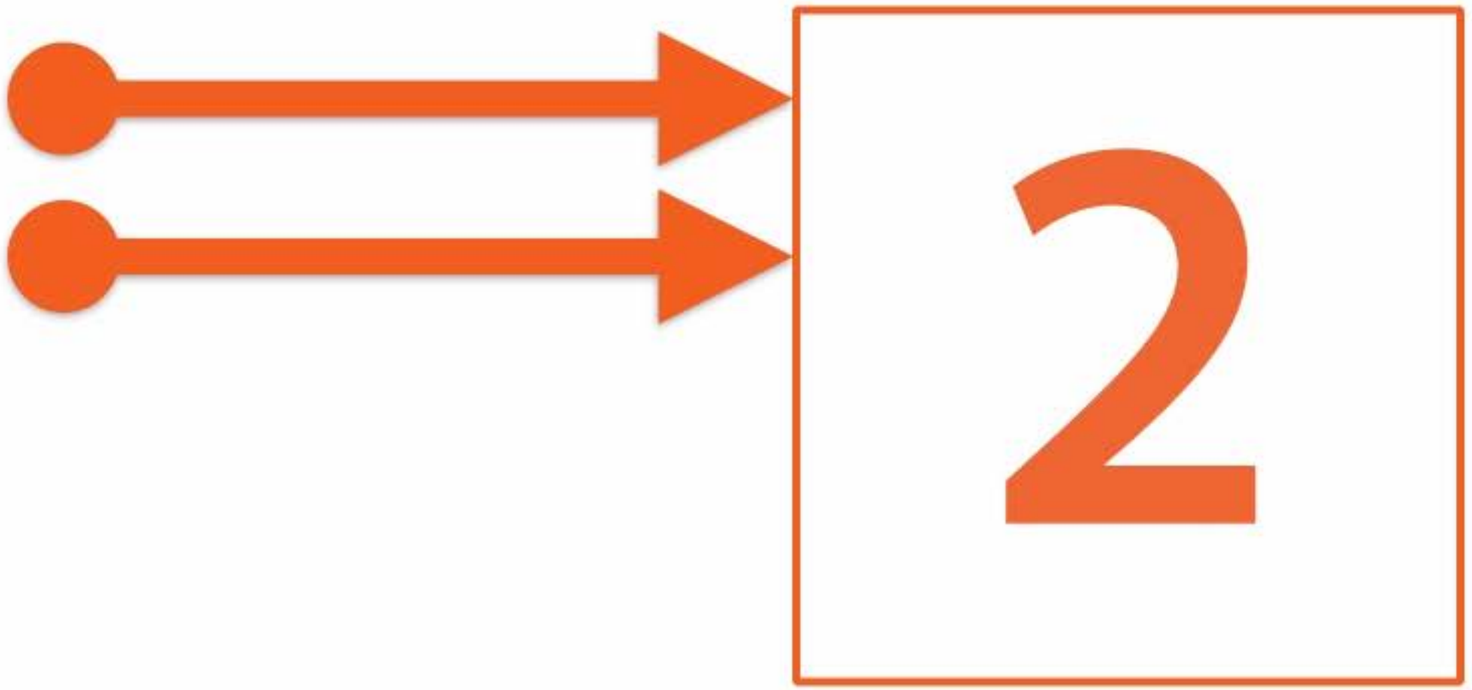
Reference Counting



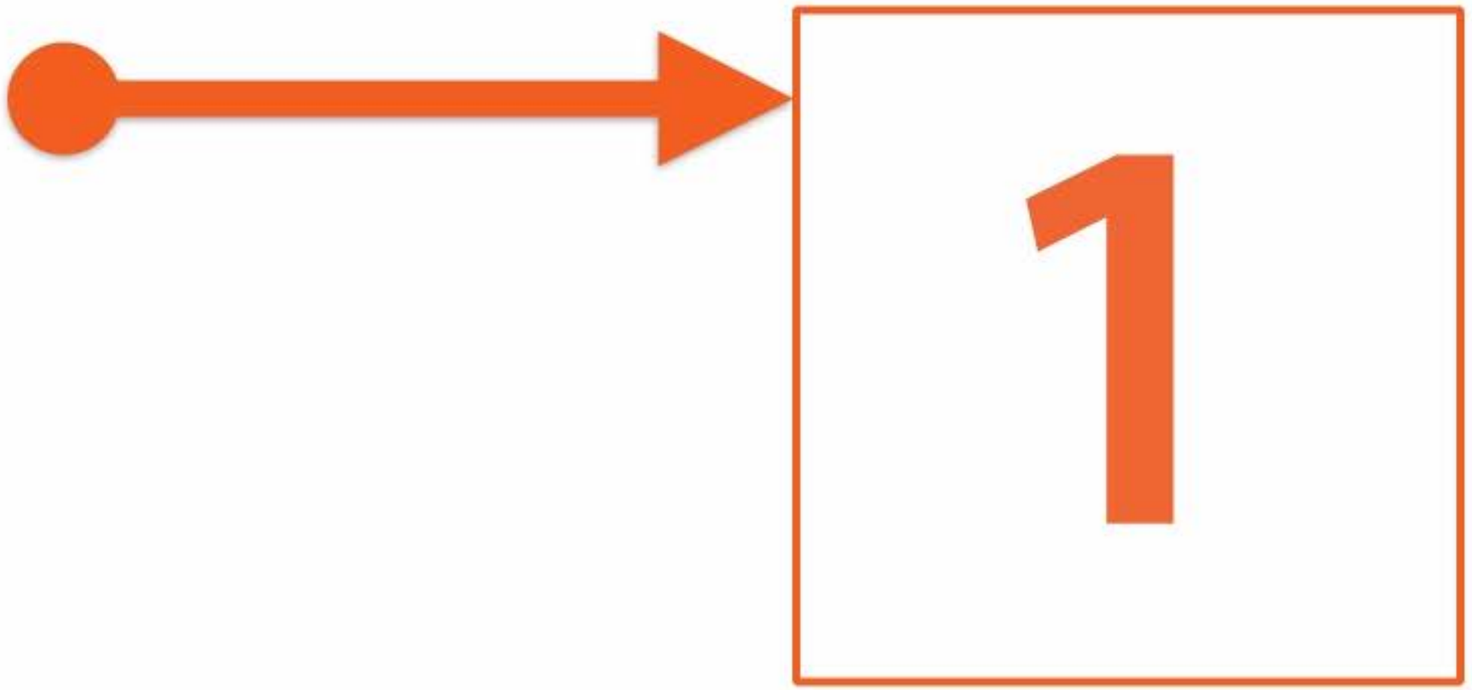
Reference Counting



Reference Counting



Reference Counting



Reference Counting

Reference Counting

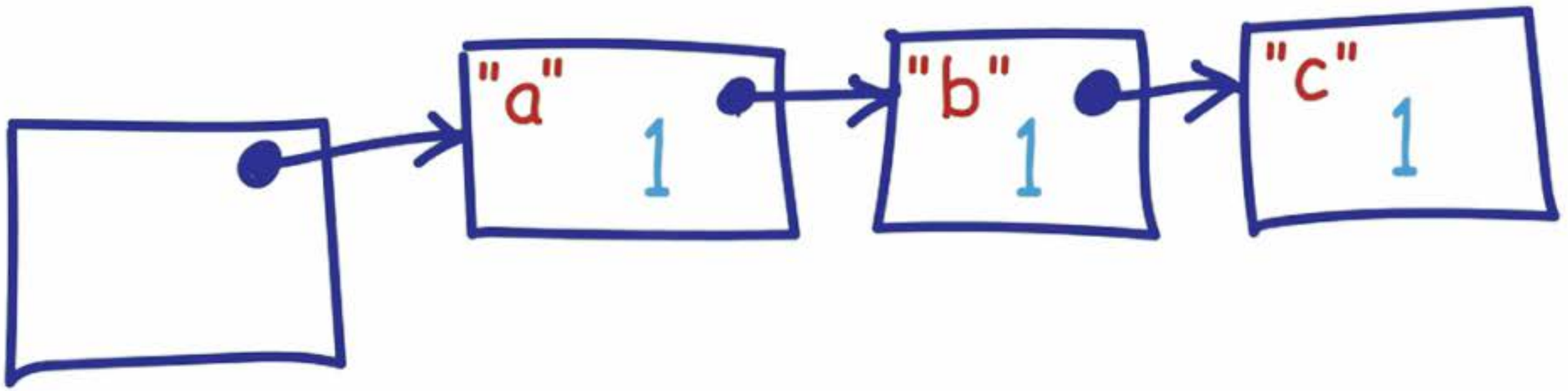
```
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}

public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new = Node(val)
    new.next = head
    head = new
  }
}
```

```
public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new  = Node(val)
    new.next = head
    head     = new
  }
}

func test() {
  let list = List<String>()
  list.insert("a")
  list.insert("b")
  list.insert("c")
  list.clear()
}
```

```
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}
```

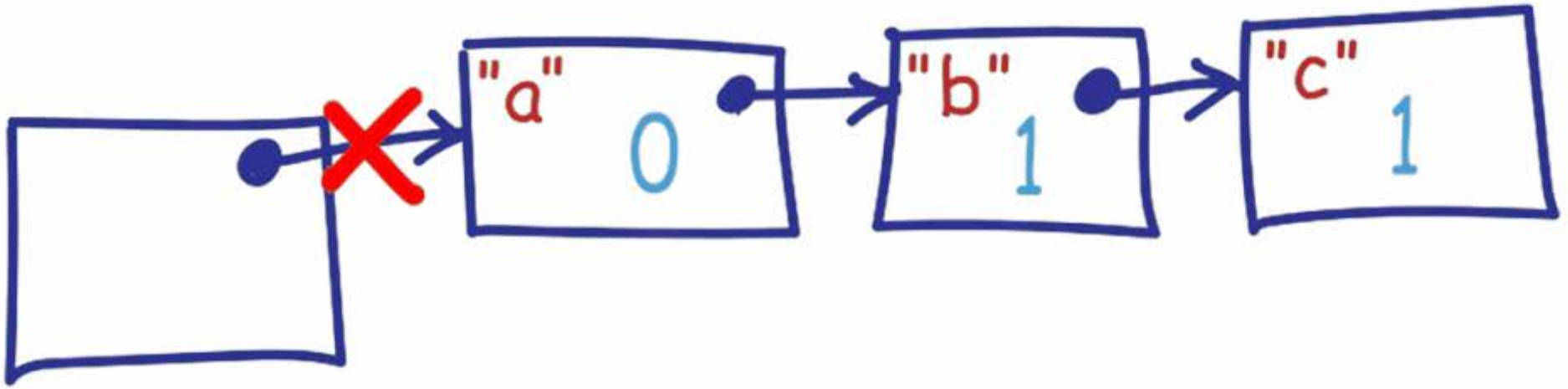
```
public class List<T> {
   private var head: Node<T>?
   public func clear(){ head = nil }
   public func insert(val: T) {
      let new  = Node(val)
      new.next = head
      head     = new
   }
}
```

```
private class Node<T> {
   var val:      T
   var next:     Node? = nil
   var previous: Node? = nil
   init(_ val: T){self.val = val}
}

func test() {
   let list = List<String>()
   list.insert("a")
   list.insert("b")
   list.insert("c")
   list.clear()
}
```



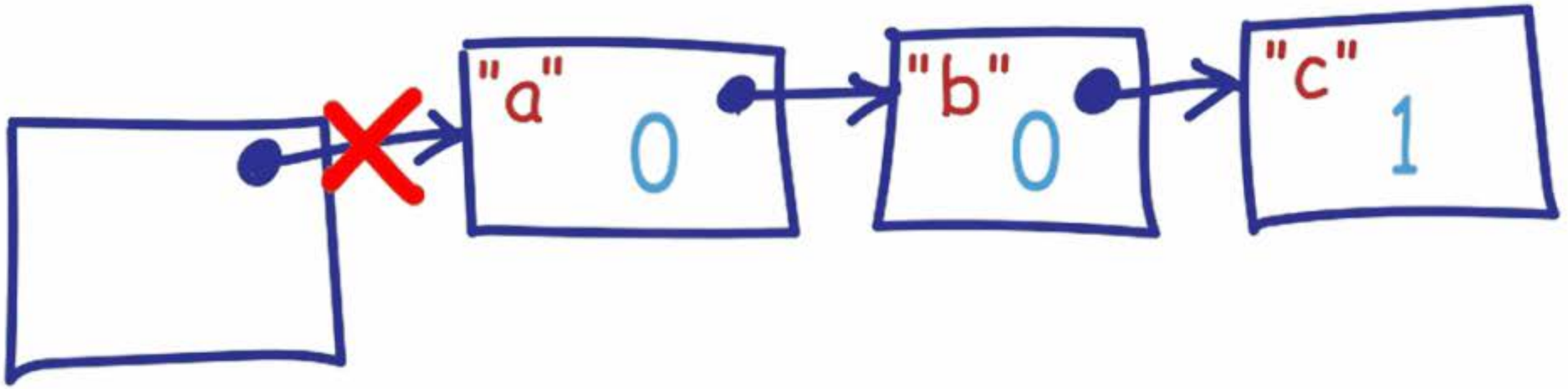
```
public class List<T> {
   private var head: Node<T>?
   public func clear(){ head = nil }
   public func insert(val: T) {
      let new  = Node(val)
      new.next = head
      head     = new
   }
}
```

```
private class Node<T> {
   var val:      T
   var next:     Node? = nil
   var previous: Node? = nil
   init(_ val: T){self.val = val}
}

func test() {
   let list = List<String>()
   list.insert("a")
   list.insert("b")
   list.insert("c")
   list.clear()
}
```



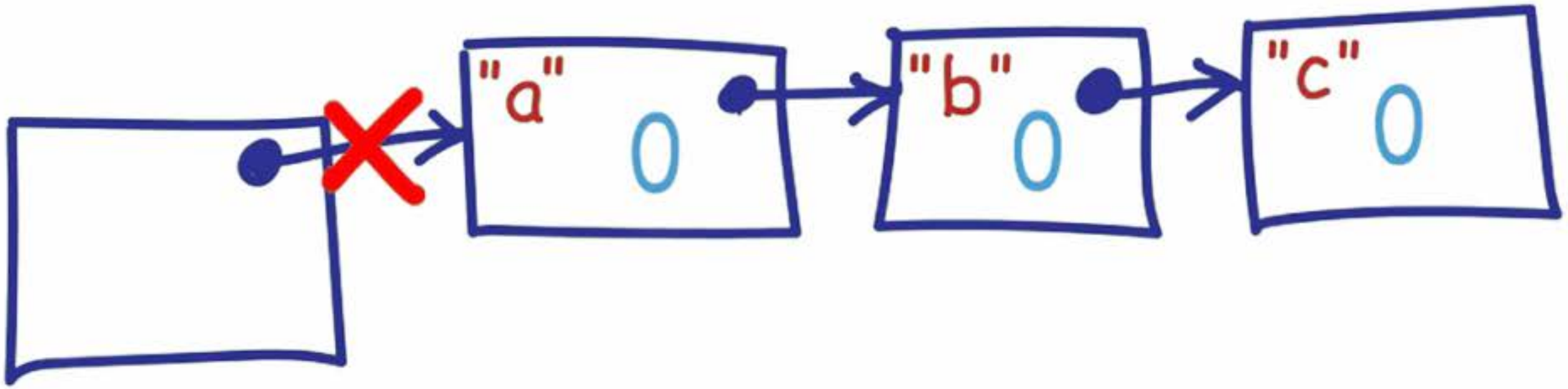
```
public class List<T> {
   private var head: Node<T>?
   public func clear(){ head = nil }
   public func insert(val: T) {
      let new  = Node(val)
      new.next = head
      head     = new
   }
}
```

```
private class Node<T> {
   var val:      T
   var next:     Node? = nil
   var previous: Node? = nil
   init(_ val: T){self.val = val}
}

func test() {
   let list = List<String>()
   list.insert("a")
   list.insert("b")
   list.insert("c")
   list.clear()
}
```



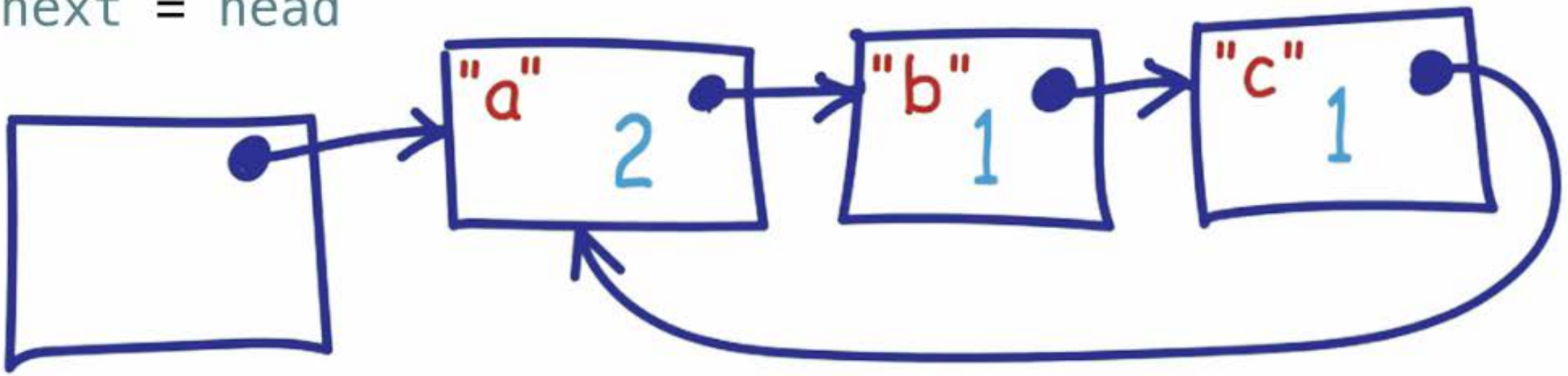
```
public class List<T> {
   private var head: Node<T>?
   public func clear(){ head = nil }
   public func insert(val: T) {
      let new  = Node(val)
      new.next = head
      head     = new
   }
}
```

```
private class Node<T> {
   var val:      T
   var next:     Node? = nil
   var previous: Node? = nil
   init(_ val: T){self.val = val}
}

func test() {
   let list = List<String>()
   list.insert("a")
   list.insert("b")
   list.insert("c")
   list.clear()
}
```



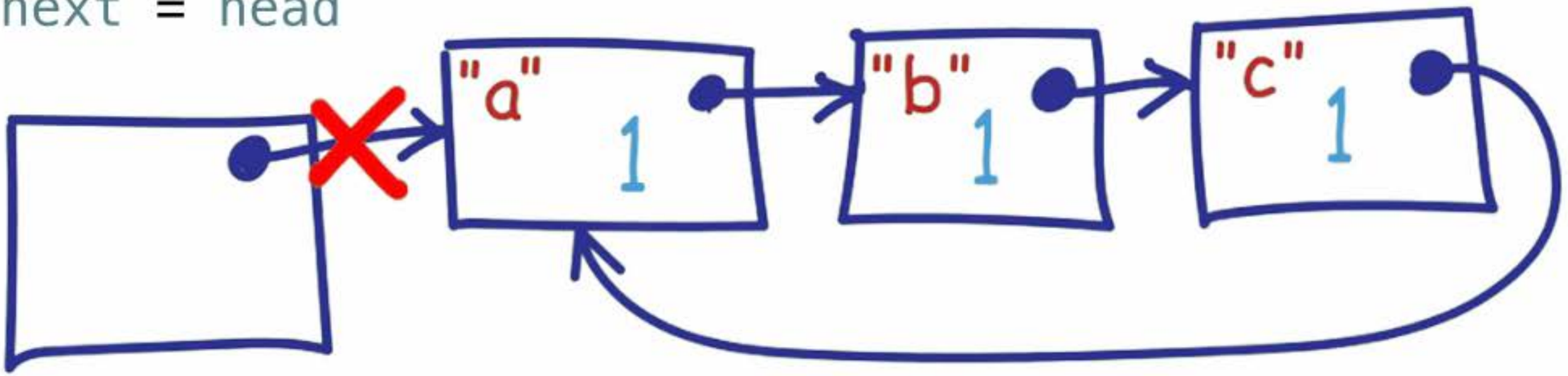
```swift
public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new  = Node(val)
    new.next = head
    head     = new
    var last = head!.next //find last
    while( last != nil ){
      last = last!.next
    }
    last!.next = head
  }
}
```

```swift
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}

func test() {
  let list = List<String>()
  list.insert("a")
  list.insert("b")
  list.insert("c")
  list.clear()
}
```



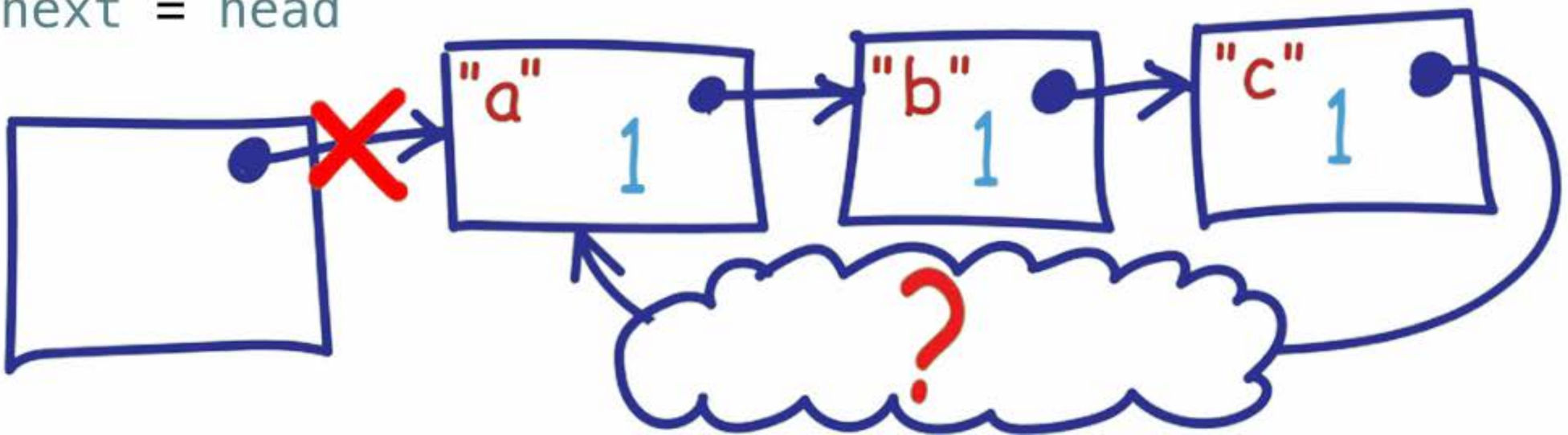
```
public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new  = Node(val)
    new.next = head
    head     = new
    var last = head!.next //find last
    while( last != nil ){
      last = last!.next
    }
    last!.next = head
  }
}
```

```
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}

func test() {
  let list = List<String>()
  list.insert("a")
  list.insert("b")
  list.insert("c")
  list.clear()
}
```



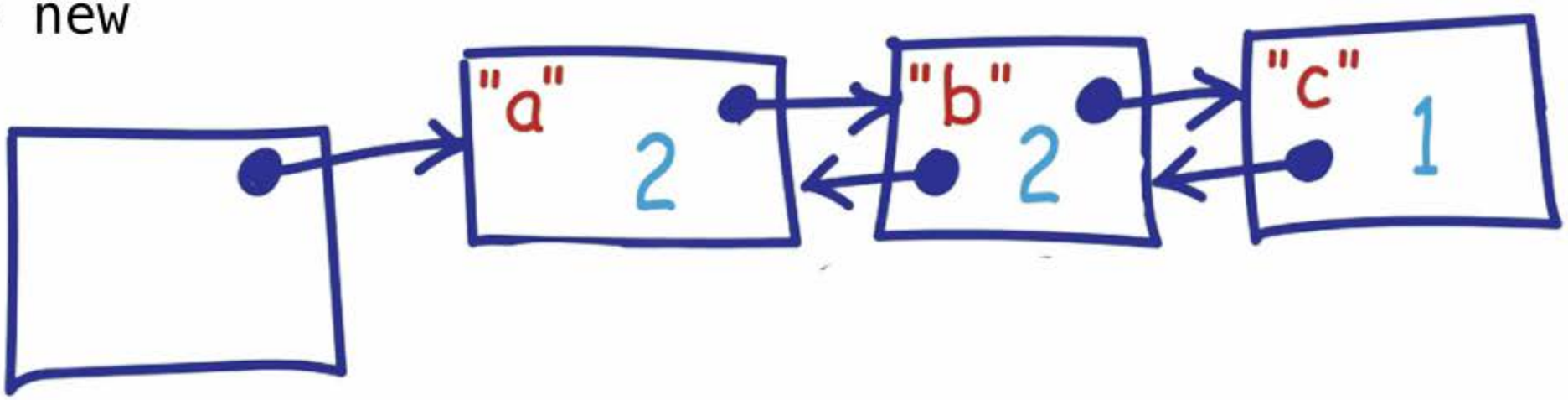
```
public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new  = Node(val)
    new.next = head
    head     = new
    var last = head!.next //find last
    while( last != nil ){
      last = last!.next
    }
    last!.next = head
  }
}
```

```
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}

func test() {
  let list = List<String>()
  list.insert("a")
  list.insert("b")
  list.insert("c")
  list.clear()
}
```



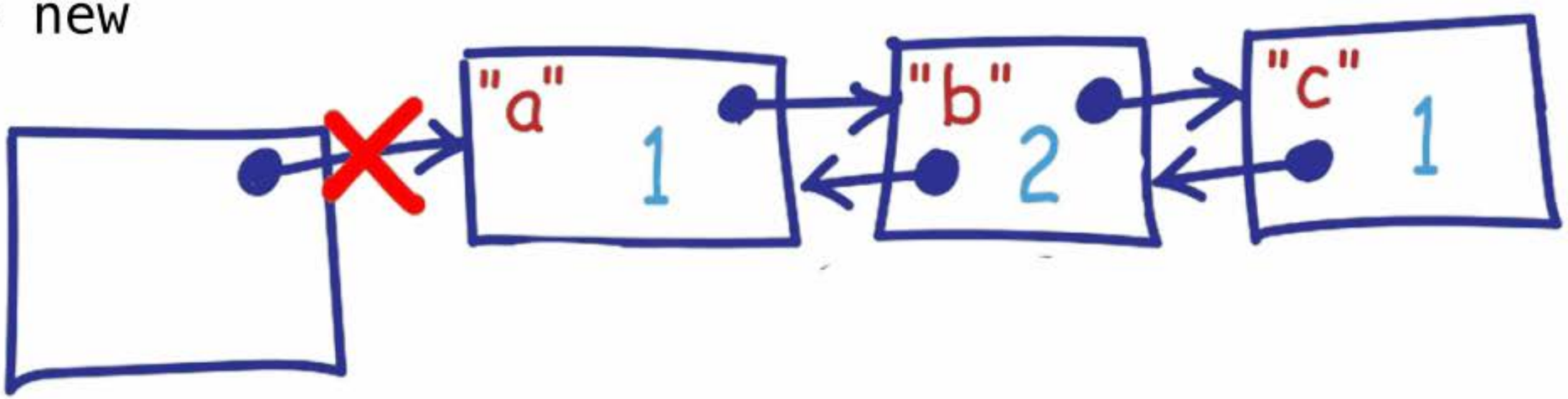
```
public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new      = Node(val)
    new.next     = nil
    new.previous = nil
    if let first = head {
      first.previous = new
      new.next = first
    }
    head = new
  }
}
```

```
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}

func test() {
  let list = List<String>()
  list.insert("a")
  list.insert("b")
  list.insert("c")
  list.clear()
}
```



```
public class List<T> {
  private var head: Node<T>?
  public func clear(){ head = nil }
  public func insert(val: T) {
    let new      = Node(val)
    new.next     = nil
    new.previous = nil
    if let first = head {
      first.previous = new
      new.next = first
    }
    head = new
  }
}
```

```
private class Node<T> {
  var val:      T
  var next:     Node? = nil
  var previous: Node? = nil
  init(_ val: T){self.val = val}
}

func test() {
  let list = List<String>()
  list.insert("a")
  list.insert("b")
  list.insert("c")
  list.clear()
}
```



```
class Committee {
  var members: [Person]=[]
  func join (newMember: Person) {
    members.append(newMember)
  }
}
class Person {
  weak var myCommittee: Committee?
  func join( committee: Committee ){
    myCommittee = committee
    committee.join( self )
  }
}
```

```
class Committee {
  var members: [Person]=[]
  func join (newMember: Person) {
    members.append(newMember)
  }
}
class Person {
  weak var myCommittee: Committee?
  func join( committee: Committee ){
    myCommittee = committee
    committee.join( self )
  }
}
```

objects have independent lifetimes

nil is reasonable

```
class Order {
  unowned var placedBy: Customer
  init( placedBy: Customer ) {
    self.placedBy = placedBy
  }
  func asString()->String{
    return "Placed by:\(placedBy)"
  }
}
class Customer {
  var myOrders: [Order] = []
  //...
}
```

Objects have the same lifetime

nil is NOT a reasonable value

```swift
class HTMLElement {
  let tag: String
  let content: String?
  init(_ tag:String, content:String){
    self.tag     = tag;
    self.content = content
  }
  lazy var asHTML: ()->String = {
    [unowned self] in
    return self.content == nil
      ? "<\(self.tag) />"
      : "<\(self.tag)>\(self.content!)</\(self.tag)>"
  }
}
let title = HTMLElement("title", content: "Hello")
print( title.asHTML() )
```