Classes Part 1:
Subclassing, Properties, and Initializers

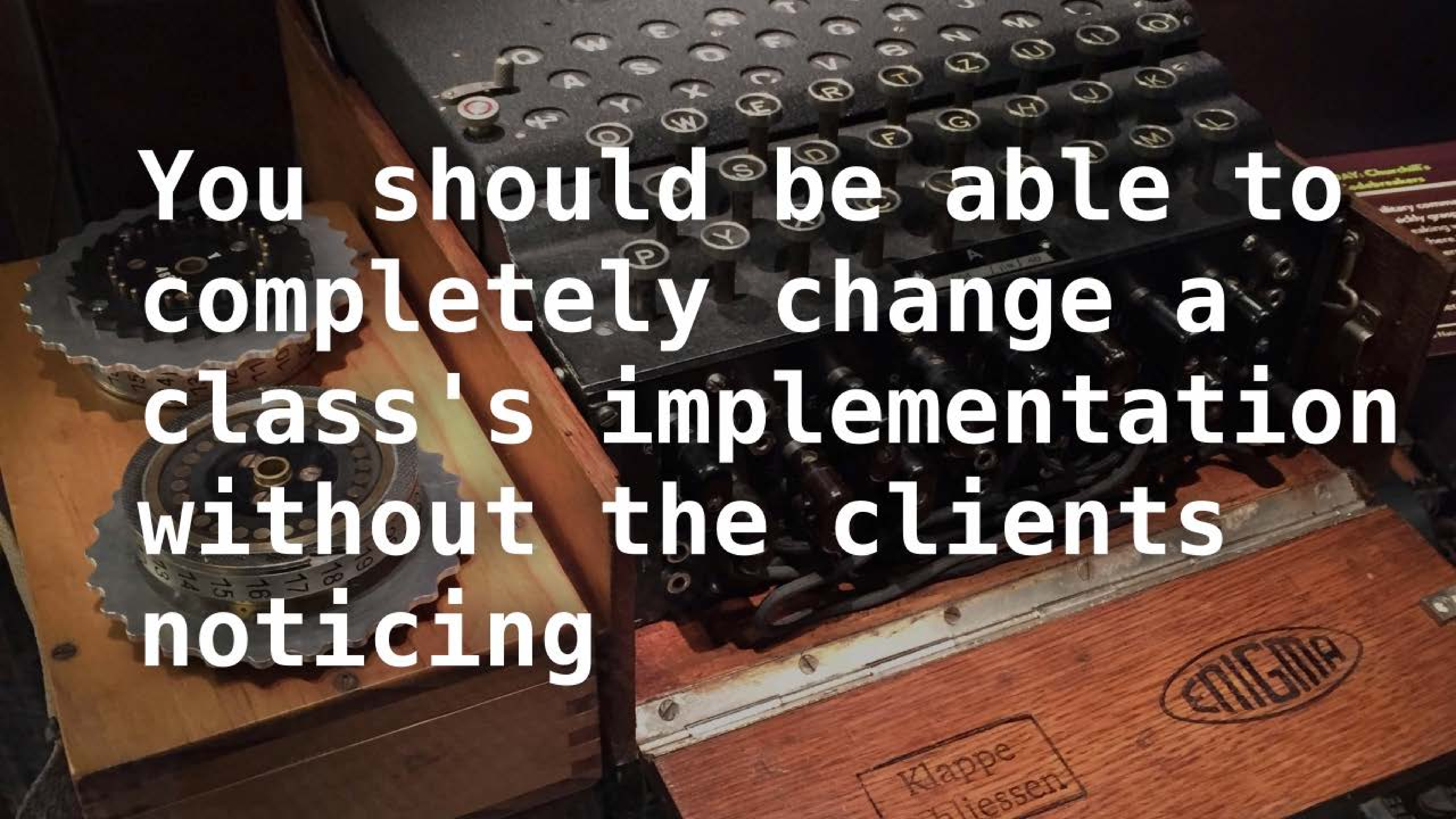
Allen Holub

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

```
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
  var email      = "fred@bedrockTileAndQuary.io"
  func changeEmailAddress( email: String ) {
      self.email = email
  }
  final func sendEmailTo( subject:String, body: String ){/*…*/}
}

class Employee: Person {
  let employeeID = 123456789

  override func changeEmailAddress(address:String){ /*...*/ }
  func          changeEmailAddress(name:String, domain:String){
      email = name + "@" + domain
  }
}
```

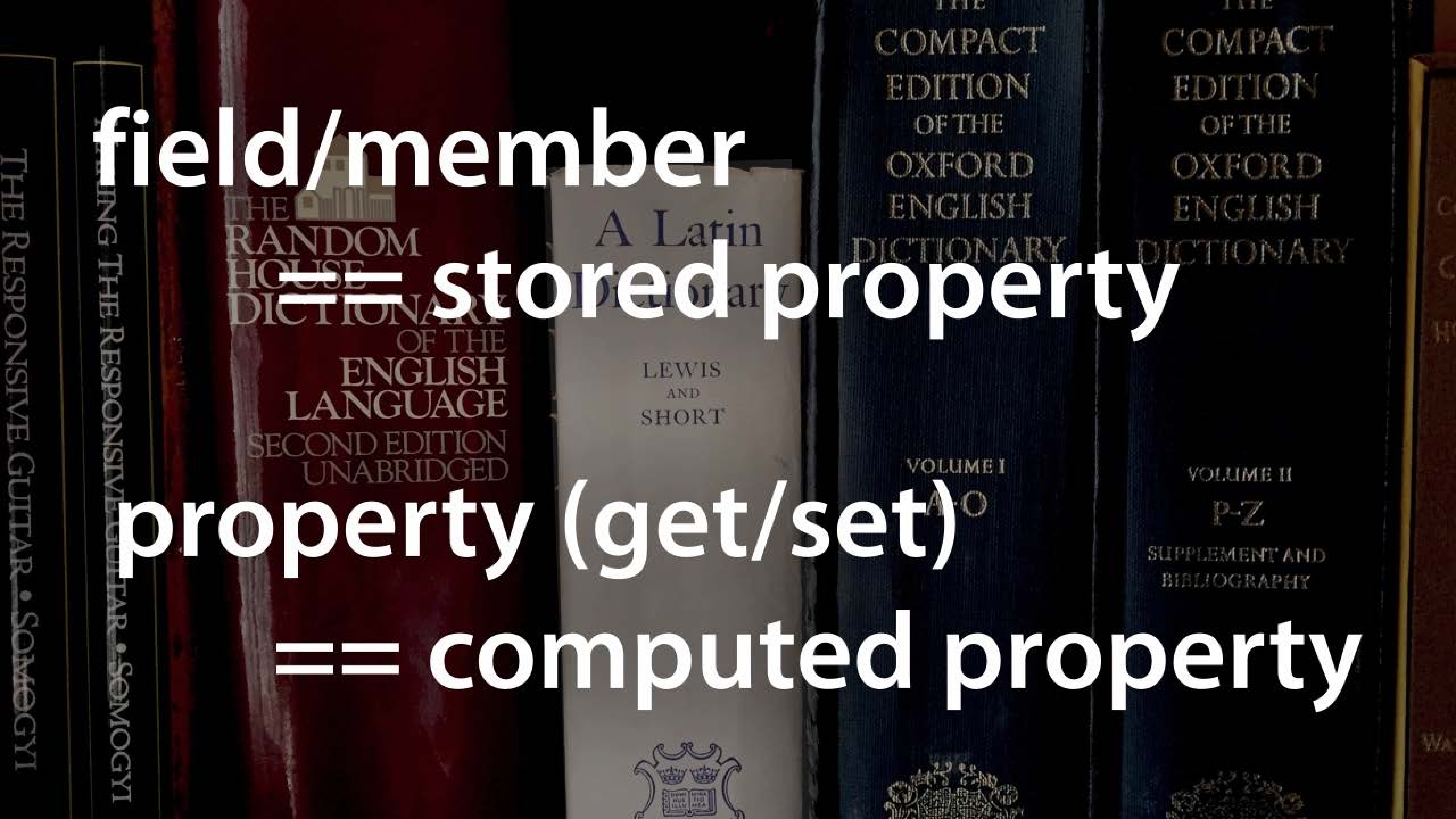
(Computed) Properties

Your implementation should be an enigma.



You should be able to completely change a class's implementation without the clients noticing

Use computed properties sparingly


THE RESPONSIVE GUITAR • SOMOGYI
THE RESPONSIVE GUITAR • SOMOGYI
THE RANDOM HOUSE DICTIONARY OF THE ENGLISH LANGUAGE
SECOND EDITION UNABRIDGED
A Latin Dictionary
LEWIS AND SHORT
THE COMPACT EDITION OF THE OXFORD ENGLISH DICTIONARY
VOLUME I
A-O
THE COMPACT EDITION OF THE OXFORD ENGLISH DICTIONARY
VOLUME II
P-Z
SUPPLEMENT AND BIBLIOGRAPHY

field/member

== stored property

property (get/set)

== computed property

```
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
}
```

```
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
  var fullName:  String {
  }
}
```

```
var barney = Person()
```

```
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
  var fullName:  String {
    set {
      var parts = split(newValue,isSeparator:{$0==" "})
      firstName = parts.count > 0 ? parts[0] : ""
      lastName  = parts.count > 1 ? parts[1] : ""
    }
  }
}
```

```
var barney = Person()
barney.fullName =
          "Barney Rubble"
```

```
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
  var fullName:  String {
    set {
      var parts = split(newValue,isSeparator:{$0==" "})
      firstName = parts.count > 0 ? parts[0] : ""
      lastName  = parts.count > 1 ? parts[1] : ""
    }
    get {
          return firstName + " " + lastName
    }
  }
}
```

```
var barney = Person()
barney.fullName =
          "Barney Rubble"
let name = barney.fullName
```

```
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
  var fullName:  String {
    set {
      var parts = split(newValue,isSeparator:{$0==" "})
      firstName = parts.count > 0 ? parts[0] : ""
      lastName  = parts.count > 1 ? parts[1] : ""
    }
    get {
          return firstName + " " + lastName
    }
  }
  var mailingAddress:String {
      return fullName + "\n"
                 + address
  }
}
```

```
var barney = Person()
barney.fullName =
          "Barney Rubble"
let name = barney.fullName
print("\(barney.mailingAddress)"
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address     = "Bedrock, CA"
  var fullName:  String {
    set {
      var parts = split(newValue,isSeparator:{$0==" "})
      firstName = parts.count > 0 ? parts[0] : ""
      lastName  = parts.count > 1 ? parts[1] : ""
    }
    get { precondition(lastName.characters.count > 0)
          return firstName + " " + lastName
    }
  }
  var mailingAddress:String {
      return fullName + "\n"
                + address
  }
}
```

```
var barney = Person()
barney.fullName =
          "Barney Rubble"
let name = barney.fullName
print("\(barney.mailingAddress)"
class Person {
  var firstName  = "Fred", lastName = "Flintstone"
  var address    = "Bedrock, CA"
  var fullName:  String {  set {/*…*/} get {/*…*/}  }
  var mailingAddress:String { /*…*/ }
}
class Employee: Person {
  override var fullName:  String { get{/*…*/} set{/*…*/} }
  override var mailingAddress: String {/*…*/}
  override var firstName: String { set{/*…*/} get{/*…*/} }
  override var lastName: String {
    willSet(newVal){ print("\(lastName) -> \(newVal)") }
    didSet (oldVal){ print("\(oldVal) is now \(lastName)")
                     lastName = lastName.uppercaseString
                    }
  }
}
```

```
class Person {
  var firstName = "Fred", lastName = "Flintstone"
  var address = "Bedrock, CA"
  var fullName: String {  set {/*…*/} get {/*…*/}  }
  var mailingAddress:String { /*…*/ }
}

class Employee: Person {
  override var fullName:  String { get{/*…*/} set{/*…*/} }
  override var mailingAddress: String {/*…*/}
  override var firstName: String { set{/*…*/} get{/*…*/} }
  override var lastName: String {
    willSet(newVal){ print("\(lastName) -> \(newVal)") }
    didSet (oldVal){ print("\(oldVal) is now \(lastName)")
                     lastName = lastName.uppercaseString
                   }
  }
}
```

```
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
ALLEN is now ALLEN
ALLEN -> ALLEN
```

```
class Person {
  var firstName  = "Fred"
  var address    = "Bedrock, CA"
  var fullName:  String {  set {/*…*/} get {/*…*/}  }
  var mailingAddress:String { /*…*/ }
  override var lastName: String = "Flintstone" {
    willSet(newVal){ print("\(lastName) -> \(newVal)") }
    didSet (oldVal){ print("\(oldVal) is now \(lastName)")
                     lastName = lastName.uppercaseString
                    }
  }
}
```

```
class MyClass {
    var x:Int?
    let const:Int = 10

    init(          ){              print("init()")   }
}
```

```
class MyClass {
    var x:Int?
    let const:Int

    init(         ){ const  = 10; print("init()")   }
}
```

```
class MyClass {
    var x:Int?
    let const:Int

    init(         ){ const  = 10; print("init()")    }
    init( _  x:Int){ self.x = x;  print("init(_)")   }
}



var c = MyClass(100)
```

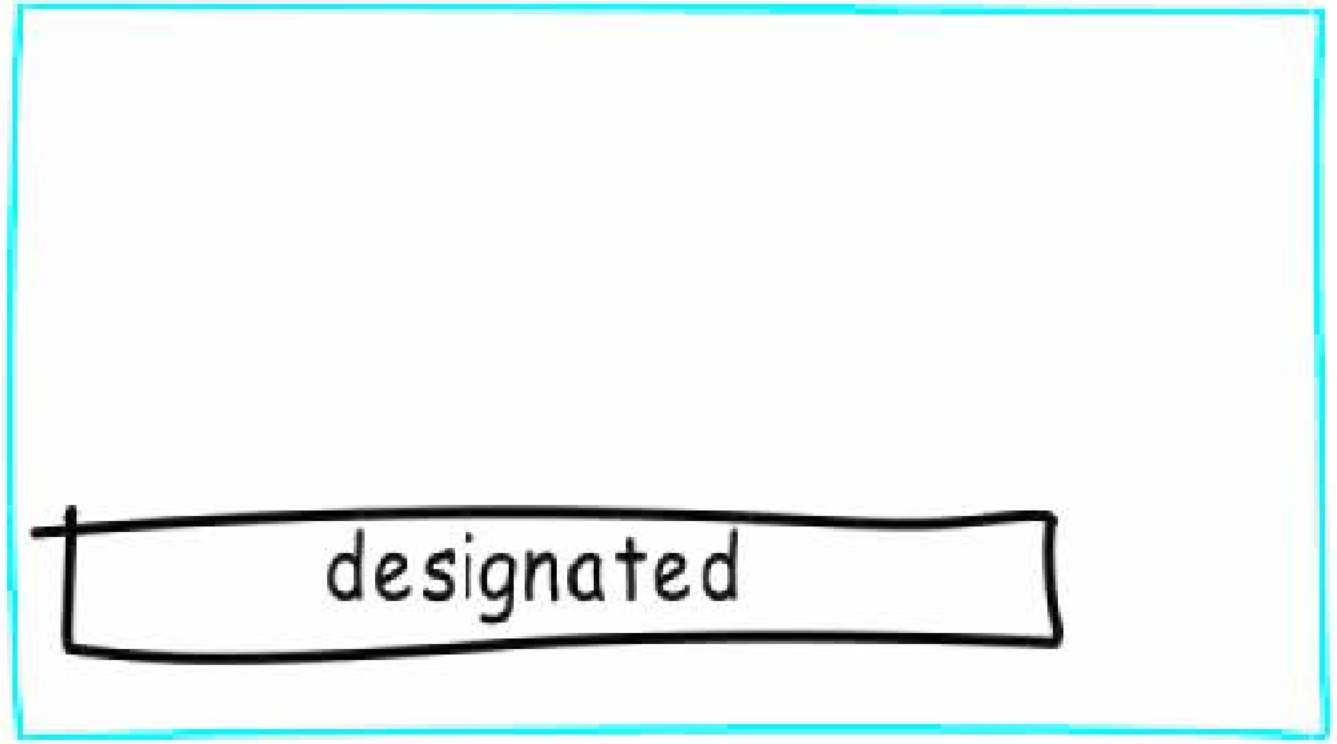
```
class MyClass {
    var x:Int?
    let const:Int

    init(         ){ const  = 10; print("init()")   }
    init( _  x:Int){ self.x = x;  print("init(_)")  }
    init(ext x:Int){ self.x = x;  print("init(ext)")}
}




var c = MyClass(ext:200)
```

```
class MyClass {
    var x:Int?
    let const:Int

    init(             ){ const  = 10; print("init()")   }
    init( _  x:Int){ self.x = x;  print("init(_)")  }
    init(ext x:Int){ self.x = x;  print("init(ext)")}
    init(    x:Int){ self.x = x;  print("init(x)")  }
}



var c = MyClass(x:300)
```

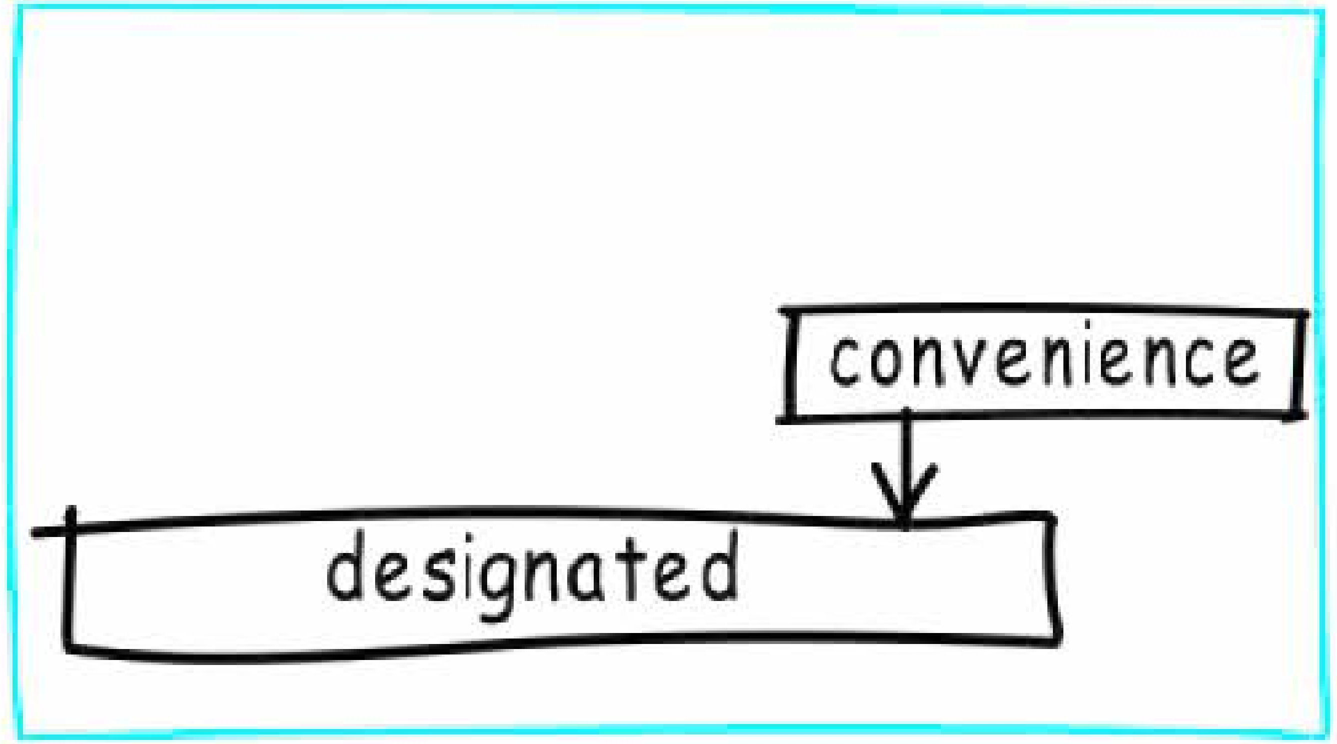
```
class MyClass {
    var x:Int?
    let const:Int

    init(         ){ const  = 10; print("init()")   }
    init( _  x:Int){ self.x = x;  print("init(_)")  }
    init(ext x:Int){ self.x = x;  print("init(ext)")}
    init(    x:Int){ self.x = x;  print("init(x)")  }
    init(    y:Int){ self.x = y;  print("init(y)")  }
}


var c = MyClass(y:400)
```

```
class MyClass {
    var x:Int?
    let const:Int

    init(             ){ const  = 10; print("init()")   }
    init( _  x:Int){ self.x = x;  print("init(_)")  }
    init(ext x:Int){ self.x = x;  print("init(ext)")}
    init(    x:Int){ self.x = x;  print("init(x)")  }
    init(    y:Int){ self.x = y;  print("init(y)")  }
    deinit         { print("destroying Cls")        }
}

var c = MyClass(y:400)

c = nil
```

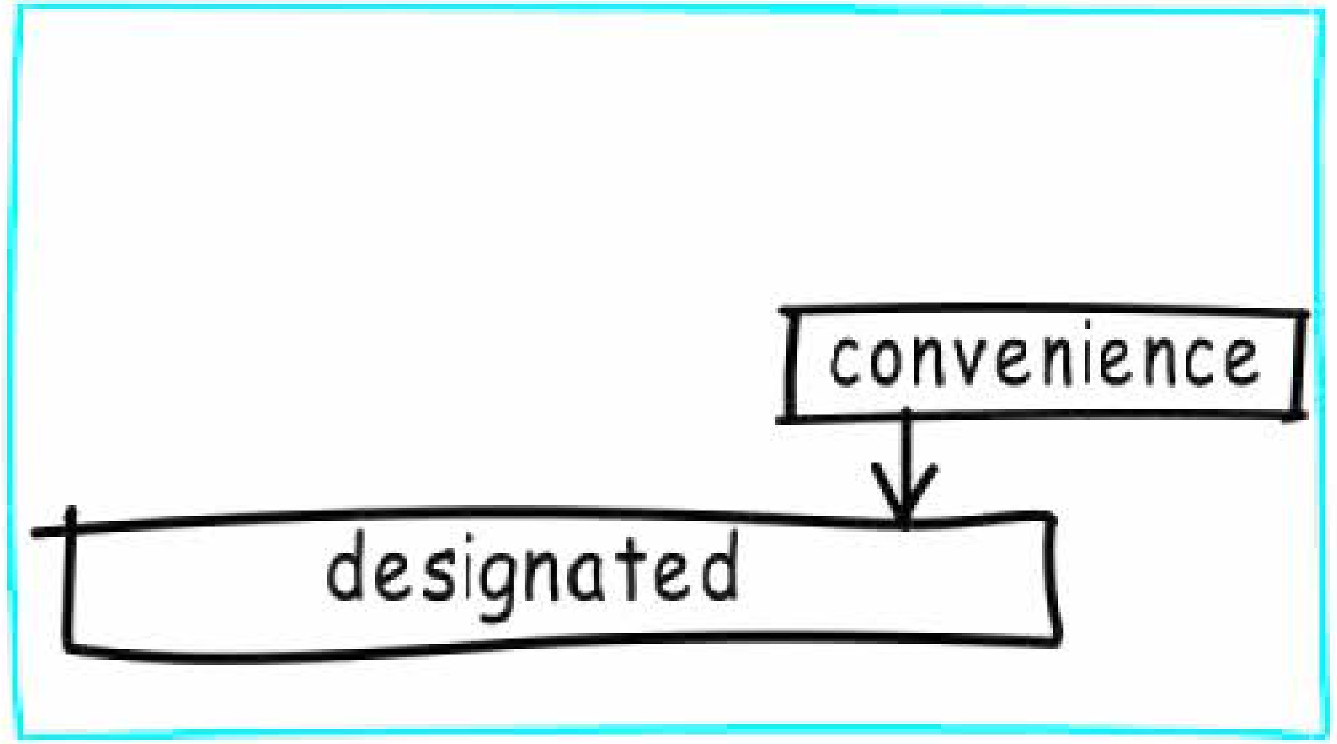
```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
}
```



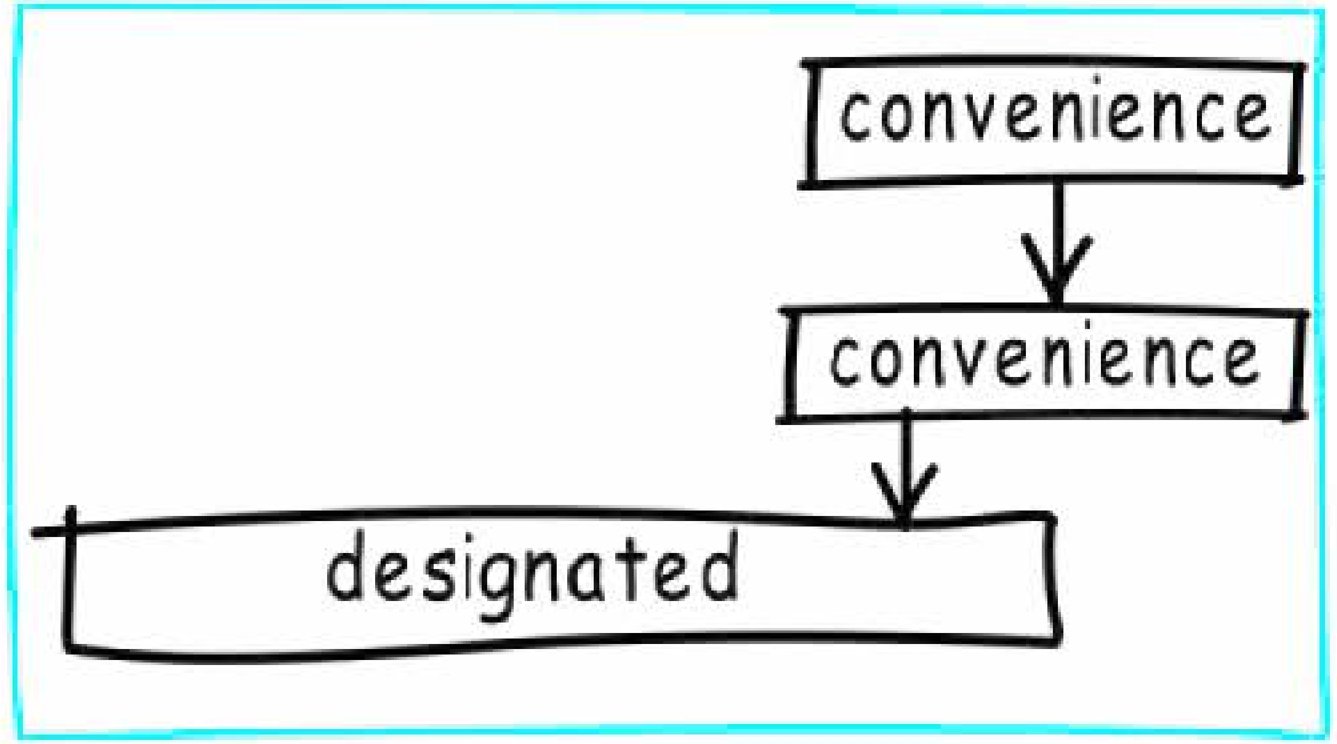
```
class Super {
  var d: Double
  init(d: Double){
       self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
}
```

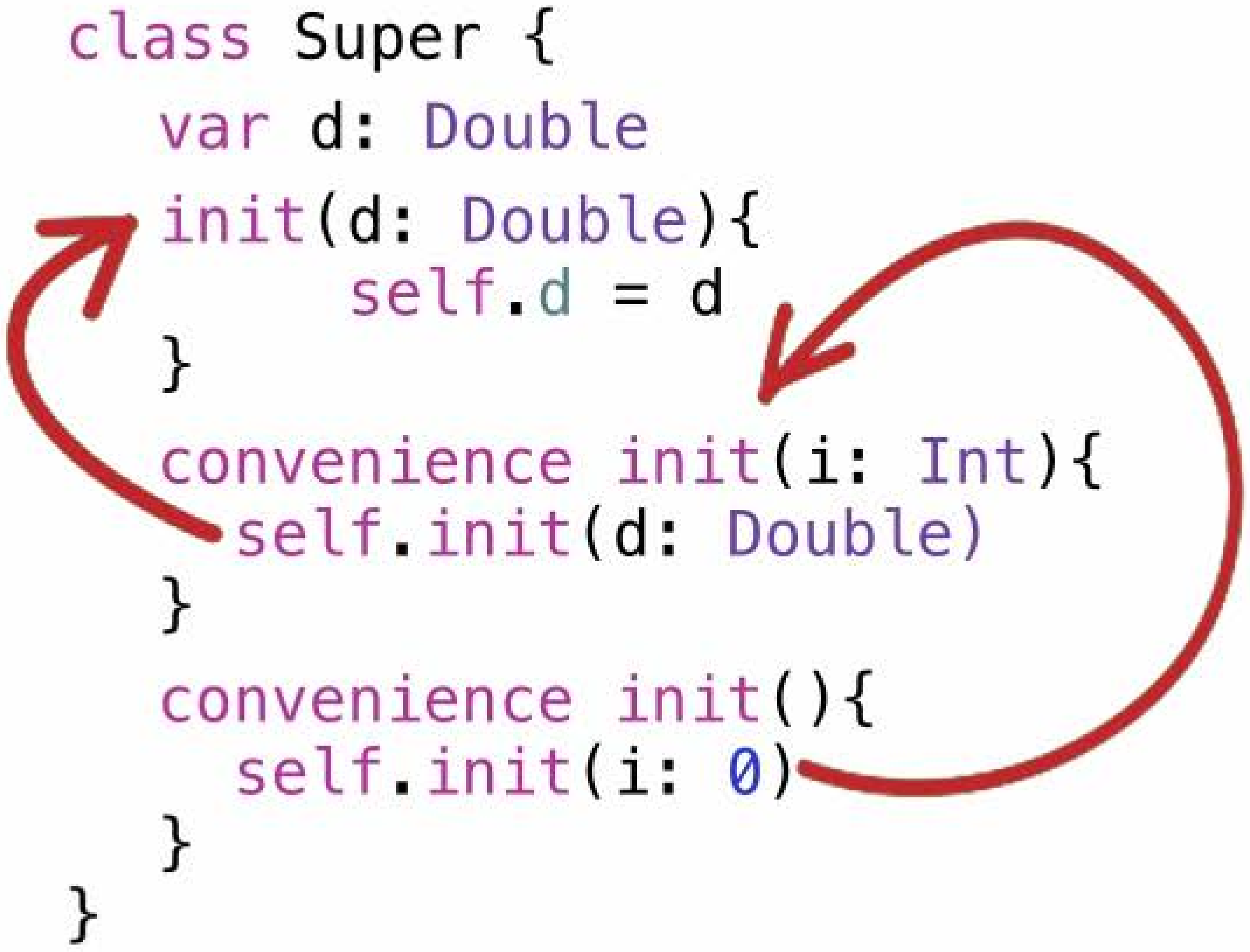


```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
}
```

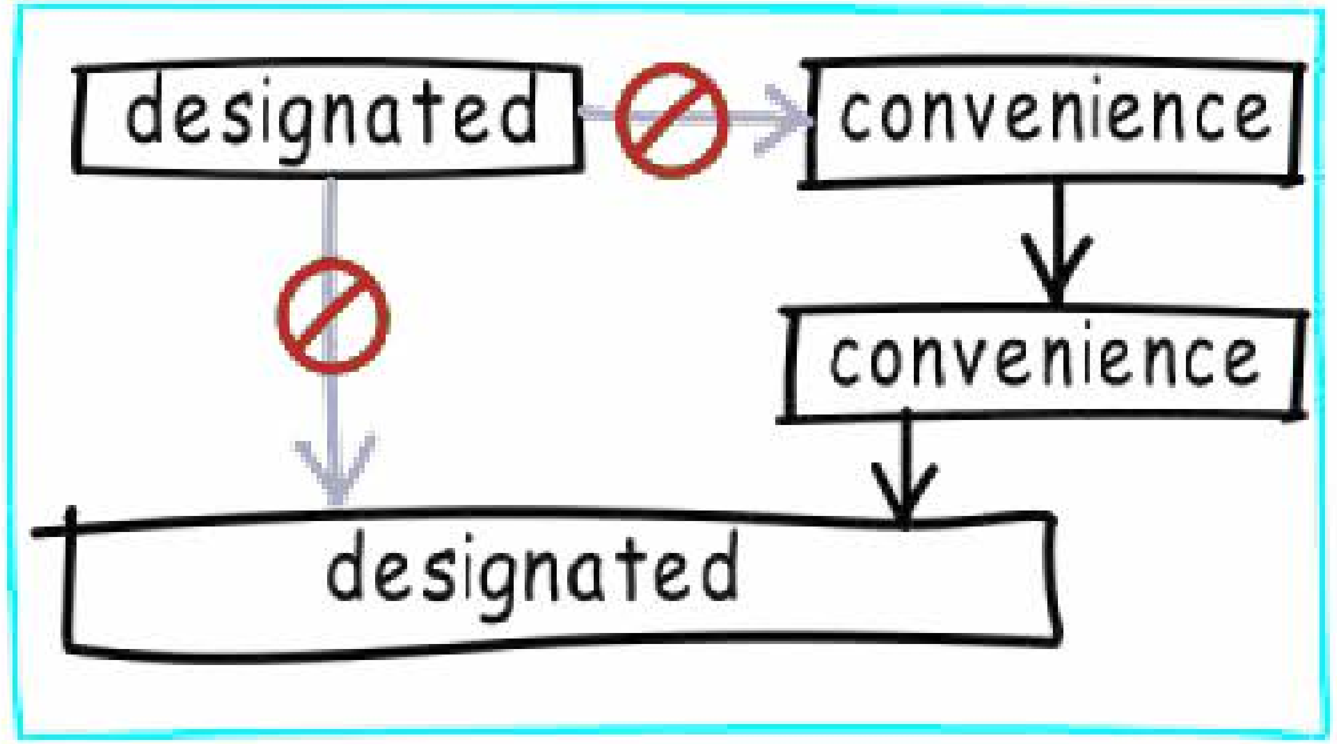


```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
  convenience init(){
    self.init(i: 0)
  }
}
```

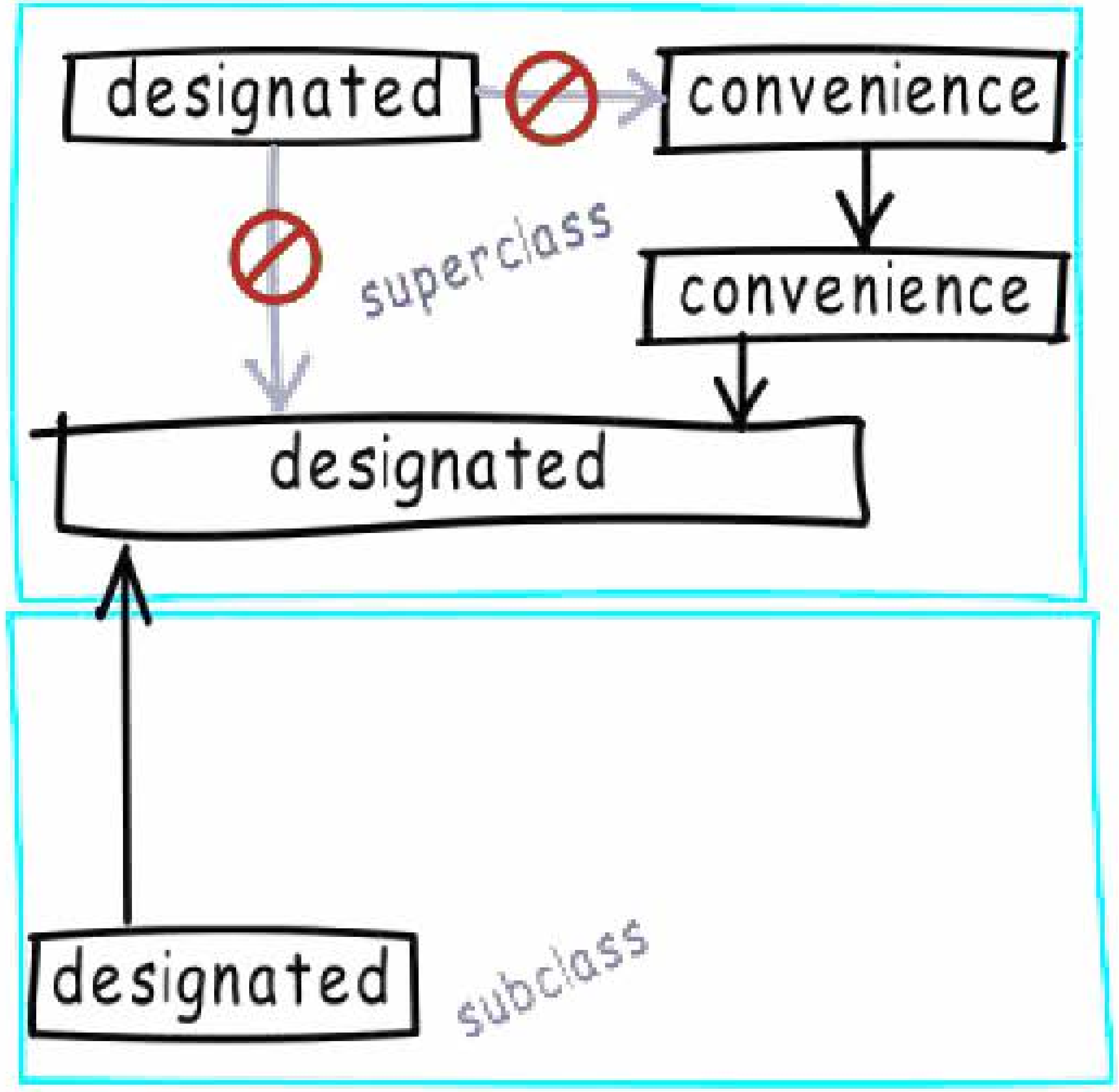


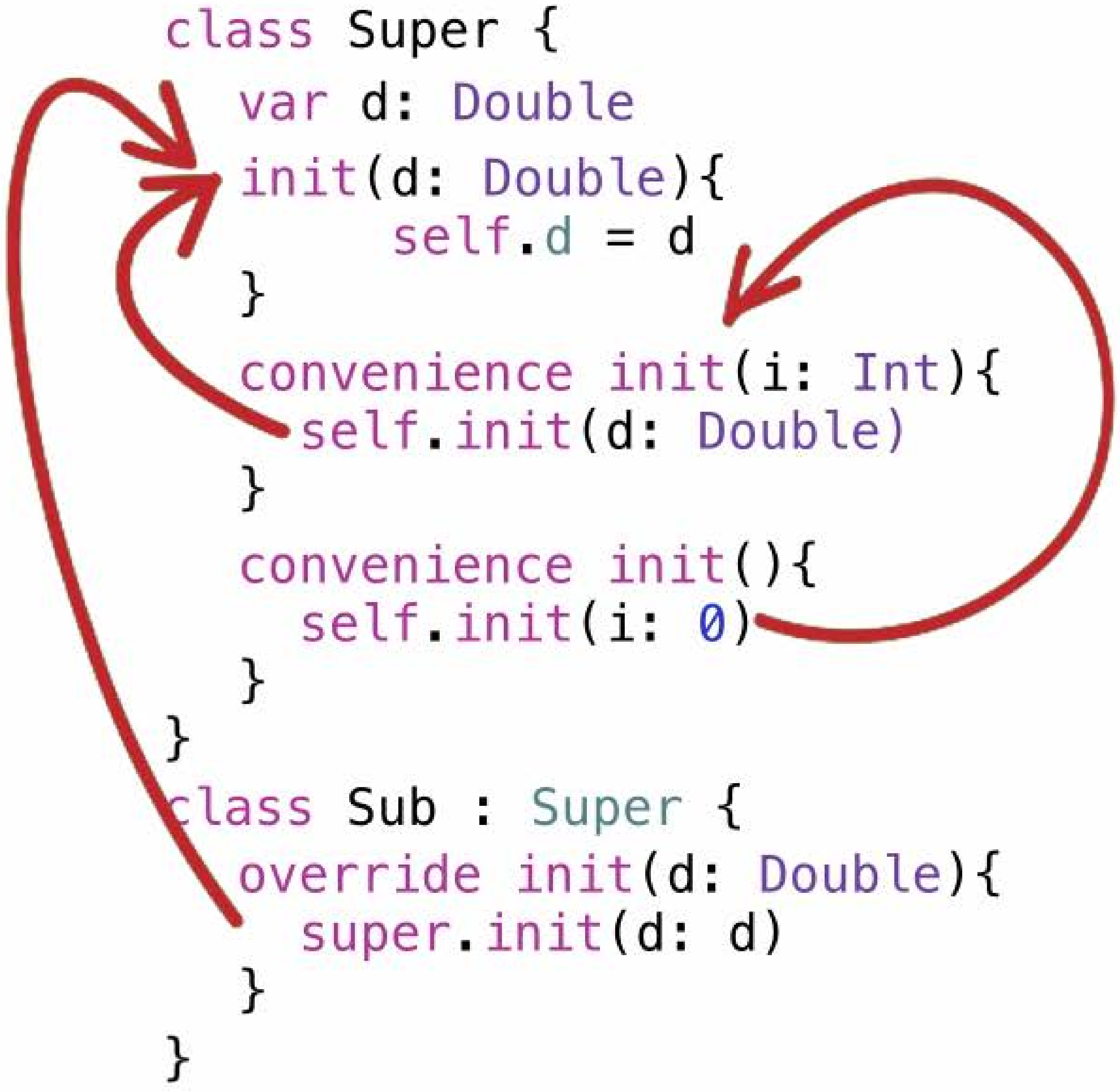


```
class Super {
   var d: Double
   init(d: Double){
         self.d = d
   }
   convenience init(i: Int){
     self.init(d: Double)
   }
   convenience init(){
     self.init(i: 0)
   }
}
```

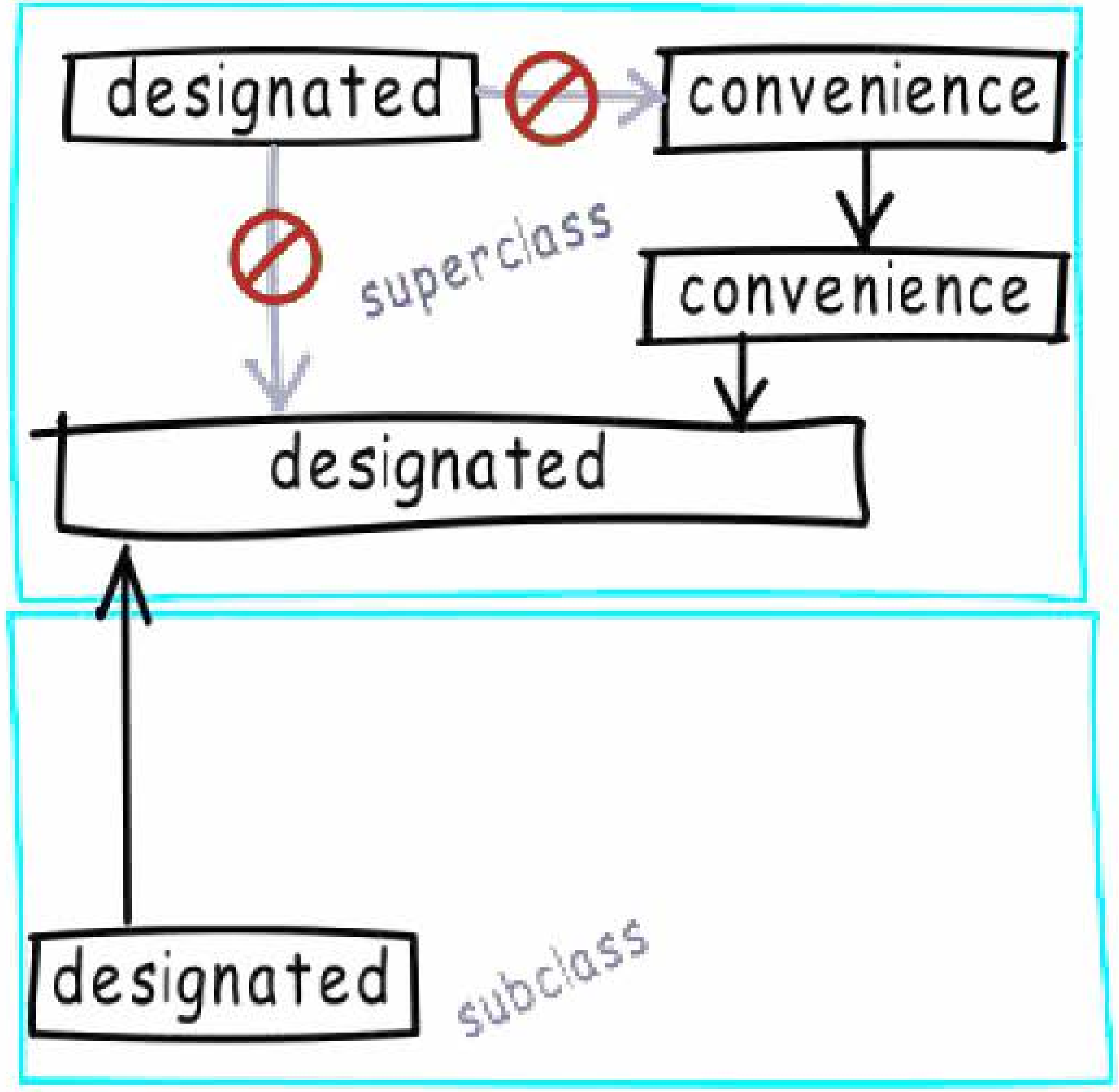


```
class Super {
  var d: Double
  init(d: Double){
       self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
  convenience init(){
    self.init(i: 0)
  }
}
class Sub : Super {
  override init(d: Double){
    super.init(d: d)
  }
}
```



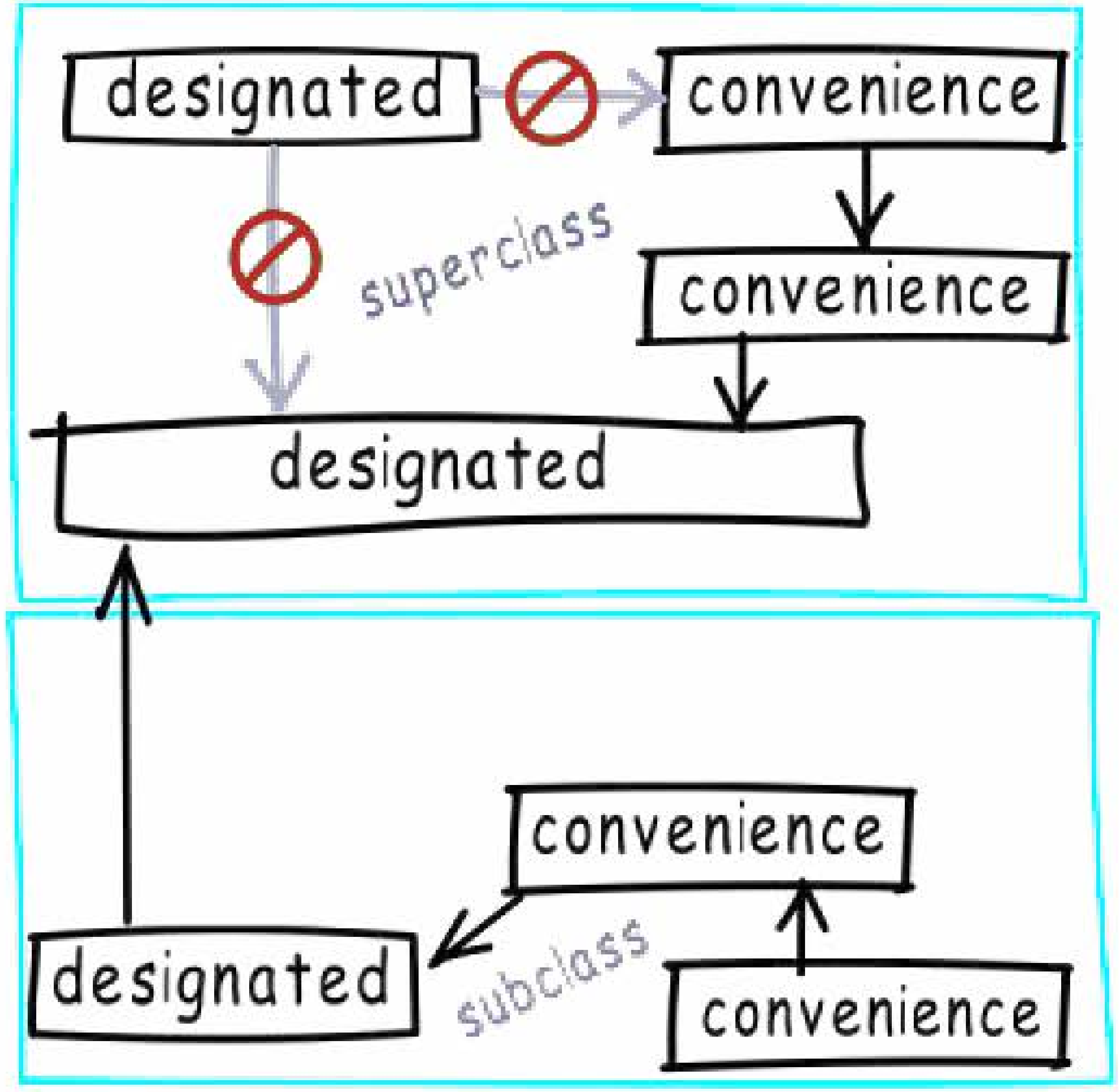


```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
  convenience init(){
    self.init(i: 0)
  }
}
class Sub : Super {
  override init(d: Double){
    super.init(d: d)
  }
}
```

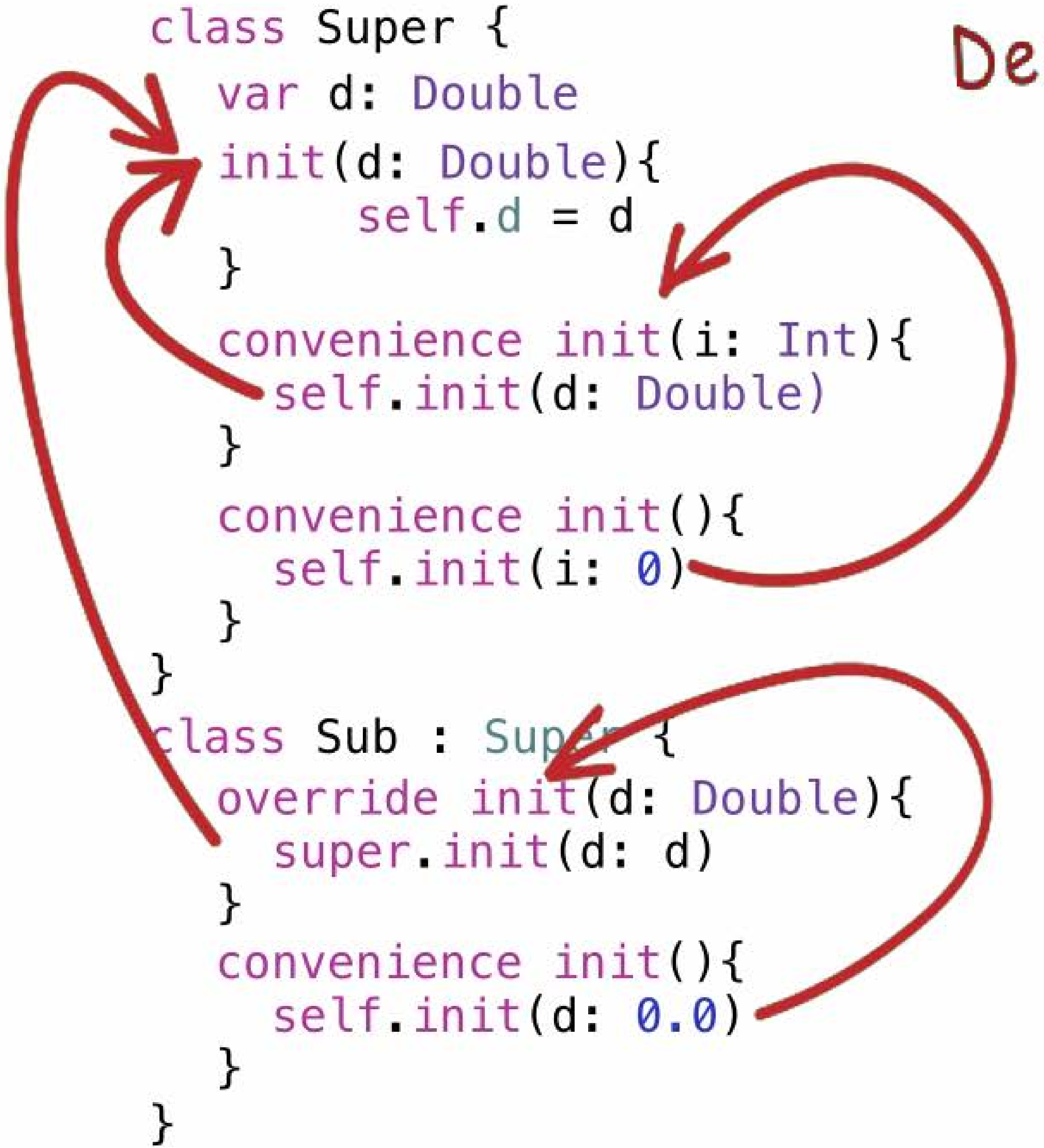


Designated chains up.

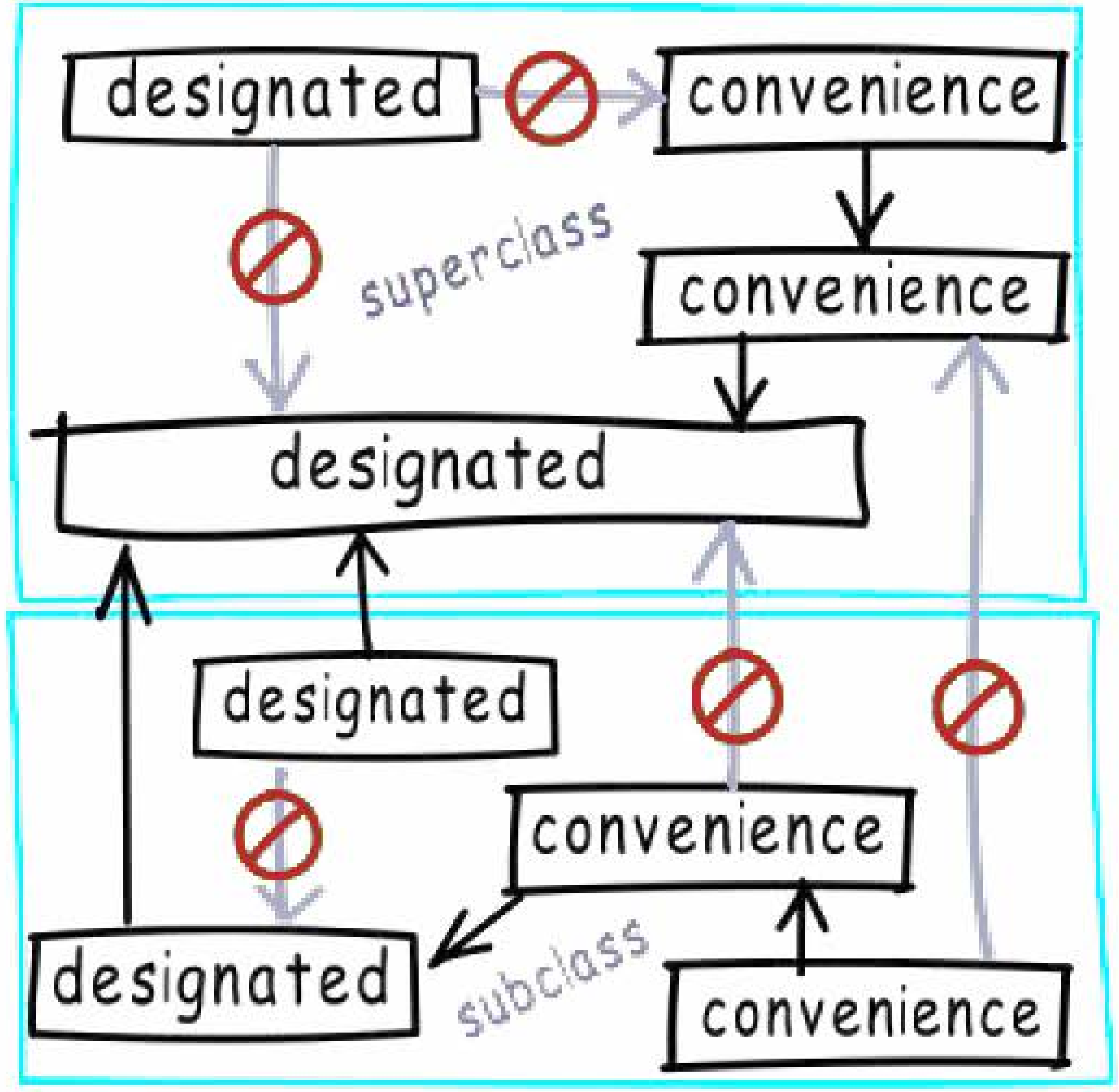
```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
  convenience init(){
    self.init(i: 0)
  }
}
class Sub : Super {
  override init(d: Double){
    super.init(d: d)
  }
  convenience init(){
    self.init(d: 0.0)
  }
}
```



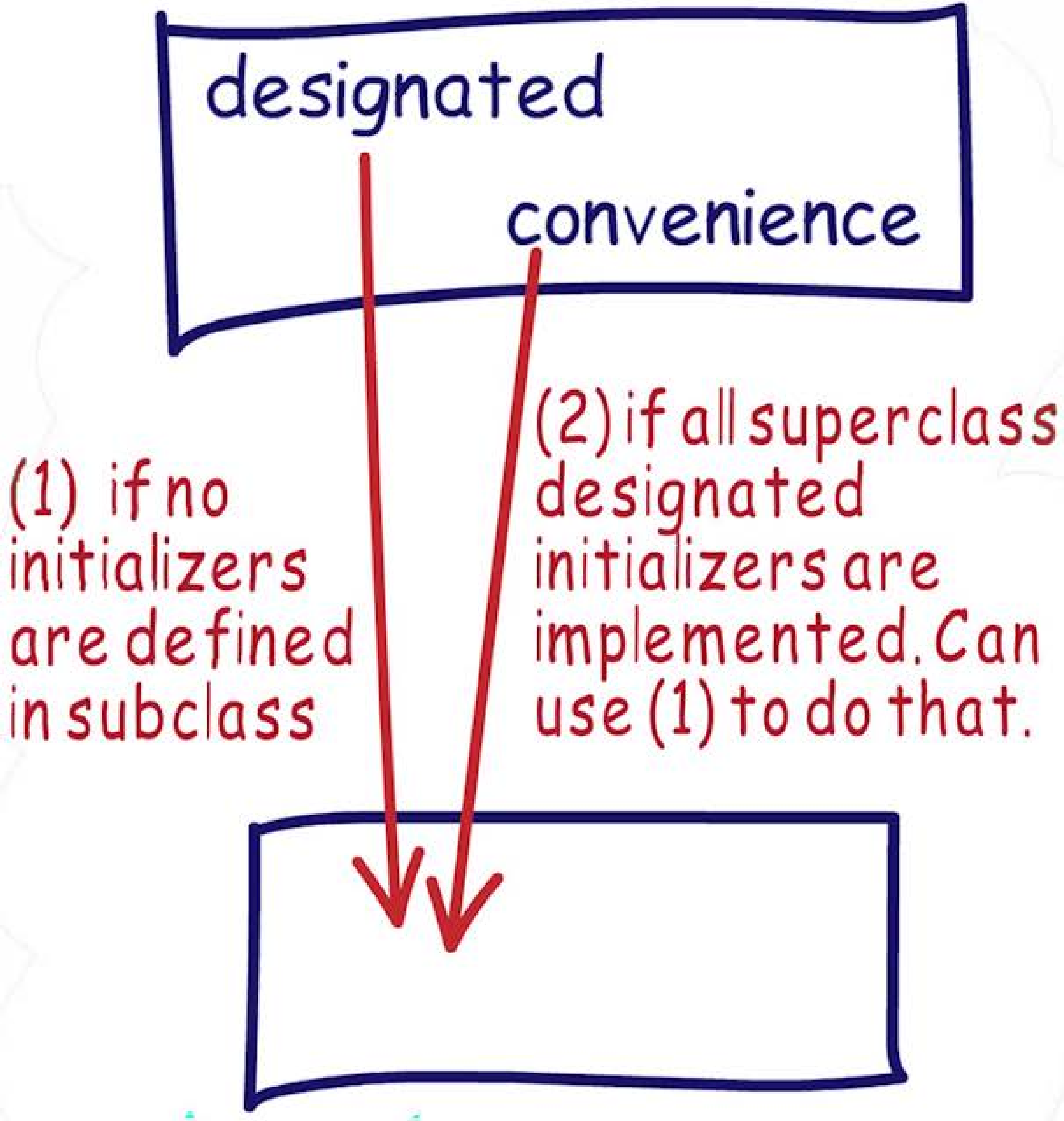
Designated chains up,
convenience across



```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
  convenience init(){
    self.init(i: 0)
  }
}
class Sub : Super {
  override init(d: Double){
    super.init(d: d)
  }
  convenience init(){
    self.init(d: 0.0)
  }
}
```

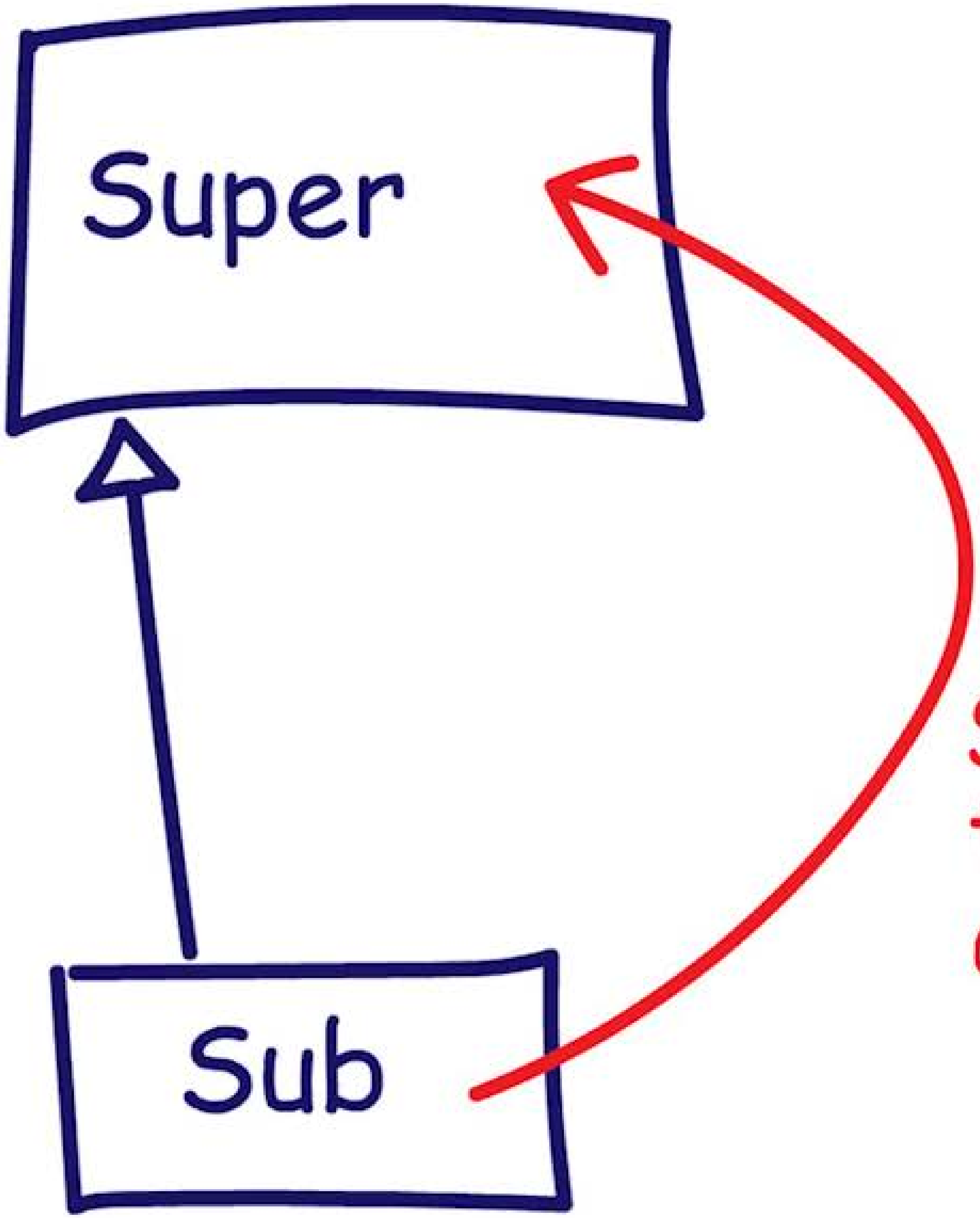


```
class Super {
  var d: Double
  init(d: Double){
      self.d = d
  }
  convenience init(i: Int){
    self.init(d: Double)
  }
  convenience init(){
    self.init(i: 0)
  }
}
class Sub : Super {
  override init(d: Double){
    super.init(d: d)
  }
  convenience init(){
    self.init(d: 0.0)
  }
}
```



(1) if no initializers are defined in subclass

(2) if all superclass designated initializers are implemented. Can use (1) to do that.



Subclass access is not safe unless the superclass is fully initialized!

So subclass init() must chain to superclass init() before it can use self.

```
class Sub: Super {
  var j: Int
  func f(){}
  init(j: Int){
    ~~i=0; f(); g(self)~~
    self.j = j
    print("\(self.j)")
    super.init(i:j)
    i=0; f(); g(self)
  }
}
```

phase 1

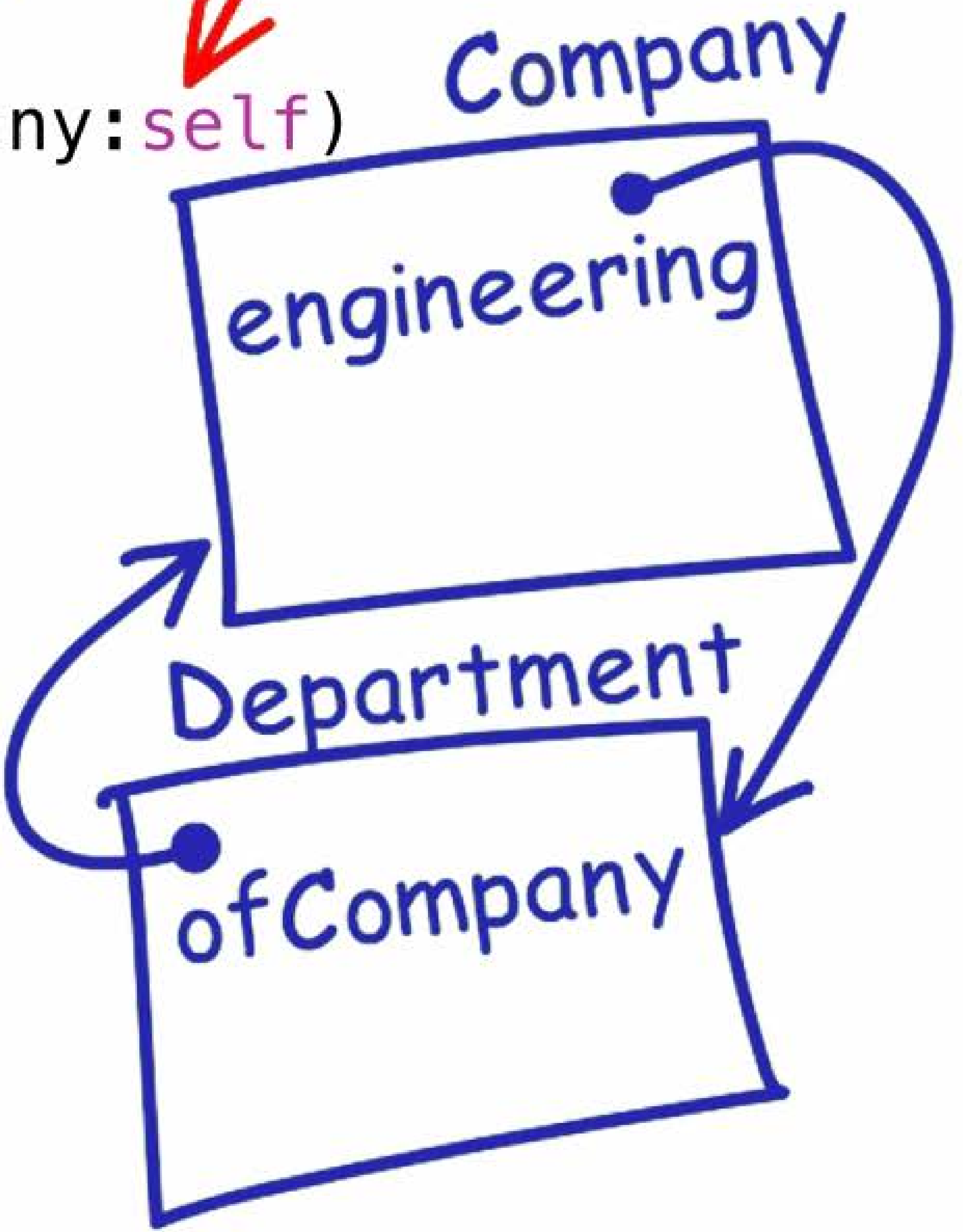
phase 2

```
class Super {
  var i: Int
  init(i:Int){self.i=i}
}
func g(s:Sub){}
```

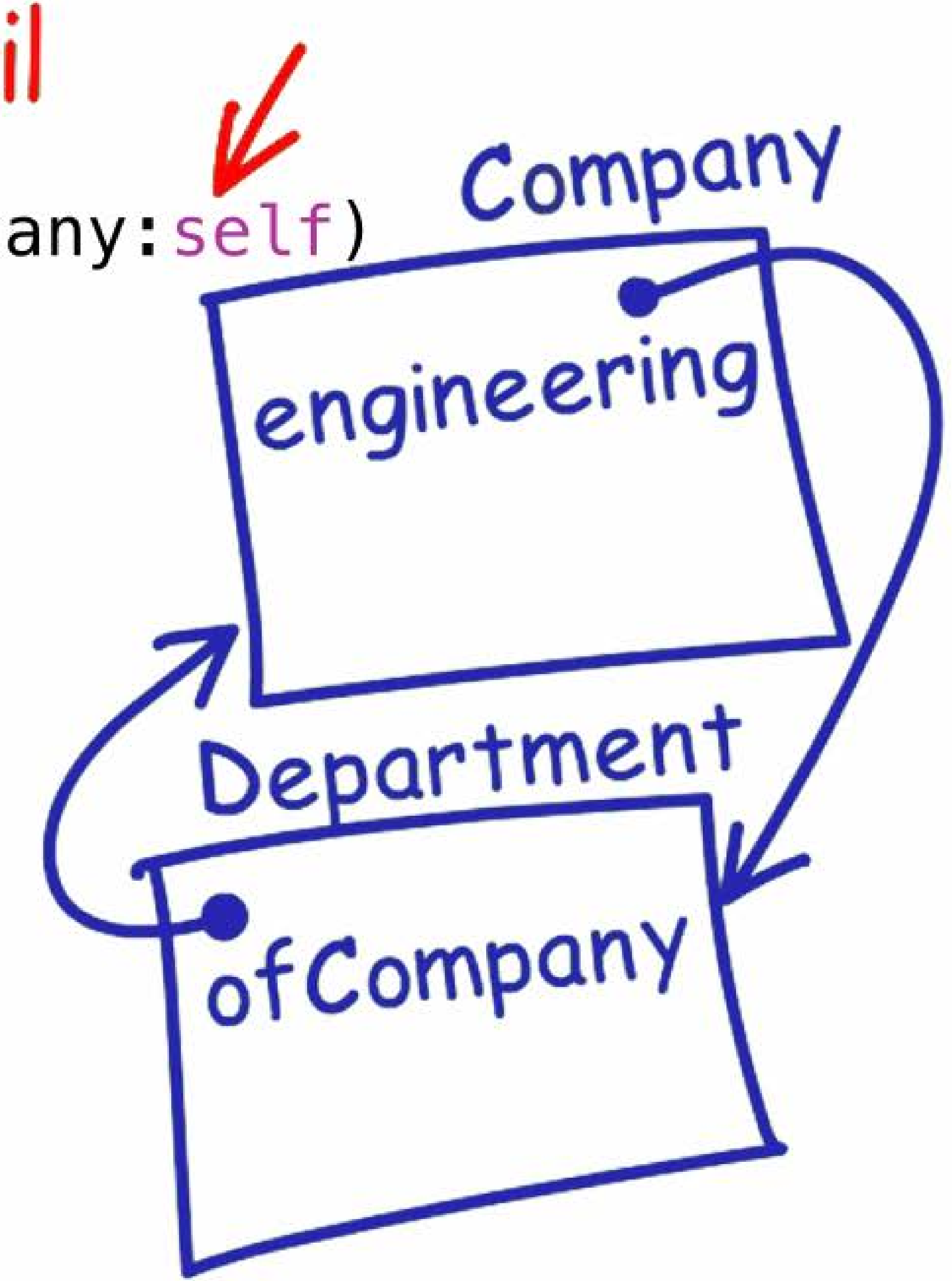
```
class Sub: Super {
  var j: Int
  func f(){}
  init(j: Int){
            ~~i=0; f(); g(self)~~
            self.j = j              ✓
phase 1     print("\(self.j)")      ✓
            super.init(i:j)
phase 2     i=0; f(); g(self)
  }
  convenience init() {
            print("hello")
            ~~j = 10; f()~~         phase 1
            self.init(j:10)
            j = 10; f()             phase 2
  }
}
```

```
class Super {
  var i: Int
  init(i:Int){self.i=i}
}
func g(s:Sub){}
```

```
class Company {
  var engineering: Department? = nil
  init() {
    engineering = Department(ofCompany:self)
    engineering!.f(self)
  }
  func g(){/* Dangerous */}
}

class Department {
  unowned let ofCompany: Company
  init( ofCompany: Company ){
    self.ofCompany = ofCompany }
  func f( c: Company ){
    ofCompany.g()
  }
}
```



```
class Company {
  var engineering: Department? = nil
  init() {
    engineering = Department(ofCompany:self)
    engineering!.f(self)
  }
  func g(){/* Dangerous */}
}

class Department {
  unowned let ofCompany: Company
  init( ofCompany: Company ){
    self.ofCompany = ofCompany }
  func f( c: Company ){
    ofCompany.g()
  }
}
```



```
struct Failable {
    init?(_ x:String) {
        if x.isEmpty {
            return nil
        } //...
    }
}
if let someF = Failable("hello") {
    /*worked!*/
}
```

```swift
enum DistanceUnit: String {
    case Feet="ft", Meters="m"
}

if let unit =
        DistanceUnit(rawValue:"ft")
```

```swift
enum TempUnit {
    case Celsius, Fahrenheit
    init?(_ symbol :Character ) {
        switch symbol  {
        case "C": self = .Celsius
        case "F": self = .Fahrenheit
        default: return nil
        }
    }
}
if let unit = TempUnit("C") {/*…*/}
```

```swift
class Super {
    required init() {
        assert(false,"not implemented")
    }
}
class Sub: Super {
    required init() {
        //...
    }
}
```