Subclasses & Inheritance

by Ash Dreyer & Donald Pinckney

Table of Contents

Review of Classes

Classes

- Group together:
  - State / Data (properties)
  - Behaviors (methods)

Subclasses

Structure in Similar Objects

- We need different code for different classes
- But the different classes are otherwise similar

Structure in Similar Objects

Knight

```
let color: Bool
var x: Int
var y: Int
func moves() -> [ChessMoves]
```

Queen

```
let color: Bool
var x: Int
var y: Int
func moves() -> [ChessMoves]
```

Structure in Similar Objects

ChessPiece

```
let color: Bool
var x: Int
var y: Int
```

Knight

```
func moves() -> [ChessMoves]
```

Queen

```
func moves() -> [ChessMoves]
```

Subclasses

ChessPiece

```
let color: Bool
var x: Int
var y: Int
```

Superclass of Knight

Knight

```
func moves: () -> [ChessMoves]
```

Subclass of ChessPiece

Writing Subclasses - Sort of Right

```
class ChessPiece {



}
```

Writing Subclasses - Sort of Right

```
class ChessPiece {
  let isWhite: Bool
  var x: Int
  var y: Int
  init(isWhite: Bool, x: Int, y: Int ) { … }
}
```

Writing Subclasses - Sort of Right

```
class Knight {
   func moves() -> [ChessMoves] { … }
}

class Queen {
   func moves() -> [ChessMoves] { … }
}
```

Writing Subclasses - Sort of Right

```
class Knight: ChessPiece {
   func moves() -> [ChessMoves] { … }
}

class Queen: ChessPiece {
   func moves() -> [ChessMoves] { … }
}
```

Writing Subclasses - Sort of Right

```
let k = Knight(isWhite: false, x: 0, y: 2)
let q = Queen(isWhite: true, x: 0, y: 2)

print(k.x)
k.y += 2

print(“Is the queen white = \(q.isWhite)”)

print(“Knight moves: \(k.moves())”)
print(“Queen moves: \(q.moves())”)
```

Overriding Methods

Overriding Methods

- Subclasses can **override**, or change, the behavior of superclass methods

Overriding Methods

TextLabel

```
func display() { … }
```

EditableLabel

```
func display() { … }
```

```
class TextLabel {
   var text: String = “”
   func display() {
      // Code to actually display text…
      print(“Displaying the text in the label”)
   }
}
class EditableLabel: TextLabel {
   override func display() {
      // Code to actually display text…
      print(“Displaying the text in the label”)
      // Code to display blinking cursor
      print(“Displaying blinking cursor thingy”)
   }
}
```

Overriding Methods

```
let label = TextLabel()
label.text = “16 Unread Emails”
label.display()

let editable = EditableLabel()
editable.text = “Subject…”
editable.display()
```

```
class TextLabel {
   var text: String = “”
   func display() {
      // Code to actually display text…
      print(“Displaying the text in the label”)
   }
}
class EditableLabel: TextLabel {
   override func display() {
      super.display()
      // Code to display blinking cursor
      print(“Displaying blinking cursor thingy”)
   }
}
```

Polymorphism

Is-A Relationship

- Suppose Queen is a subclass of ChessPiece
- Then, a Queen **is-a** ChessPiece
- But, a ChessPiece is **NOT** a Queen
- This should make sense when you say it out loud for all your subclasses!

Is-A Relationship

ChessPiece

```
let color: Bool
var x: Int
var y: Int
```

Knight

```
func moves() -> [ChessMoves]
```

Queen

```
func moves() -> [ChessMoves]
```

Is-A Relationship

```
let queen = Queen(isWhite: true, x: 0, y: 2)

let piece: ChessPiece = Queen(isWhite: false, x:
3, y: 1)

print(queen.x) // 0
print(piece.x) // 3

print(queen.moves()) // [ … ]
print(piece.moves()) // Compiler error!
```

Writing Subclasses - Better!

```
class ChessPiece {
  let isWhite: Bool
  var x: Int
  var y: Int
  init(isWhite: Bool, x: Int, y: Int ) { … }
  func moves() -> [ChessMove] {
    // This does nothing, because subclasses will
override it!
    return []
  }
}
```

Writing Subclasses - Better!

```
class Knight {
   override func moves() -> [ChessMoves] { … }
}

class Queen {
   override func moves() -> [ChessMoves] { … }
}
```

Writing Subclasses - Better!

```
let queen = Queen(isWhite: true, x: 0, y: 2)

let piece: ChessPiece = Queen(isWhite: false, x:
3, y: 1)

print(queen.x) // 0
print(piece.x) // 3

print(queen.moves()) // [ … ]
print(piece.moves()) // This calls the Queen move
method.
```