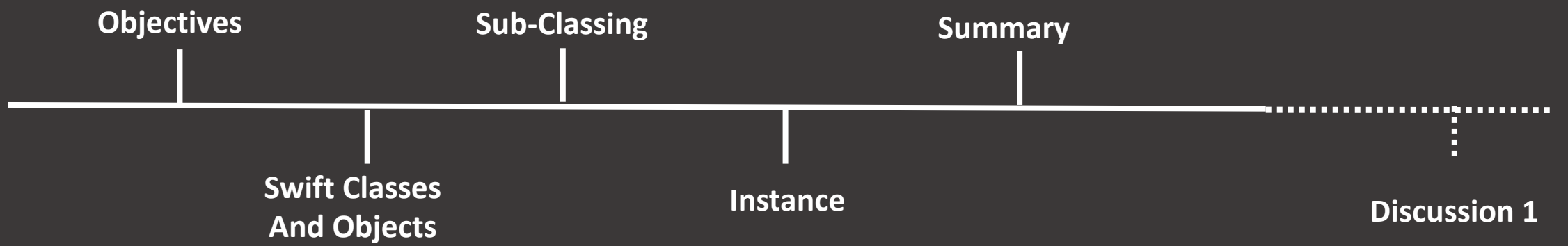
Mobile Applications Development 2 (MAD2)

Diploma in IT

Teaching Team:
Mr Ng Poh Oon
Mr Charles Keck

NGEE ANN
SCHOOL OF INFOCOMM TECHNOLOGY

Chapter 2
Swift Classes



Objectives

To be able to understand:

- Swift Language
  - Classes and Objects
  - Property
  - Subclassing
  - Instance

Swift Classes and Objects

NGEE ANN
SCHOOL OF INFOCOMM TECHNOLOGY

MAD2 Oct 2019

Object Oriented Programming in Swift

- OO programming is a fundamental programming paradigm
  - OO is the heart of most frameworks (you'll be working with)
  - *Objects can be used to model almost anything*
    - *coordinates on a map*
    - *touches on a screen*
    - *even fluctuating interest rates in a bank account*

Object Oriented Programming in Swift

- Swift classes demonstrate features of Object-Oriented Programming
  - Encapsulation
    - Properties and methods
  - Inheritance
  - Polymorphism – through dynamic message handling based on class of receiver

Self study: Swift's Overriding vs Overloading, Types vs Instances, Composition and Access Control

What is a Swift's Class or Structure?

- Classes and structures in Swift have a similar definition syntax:

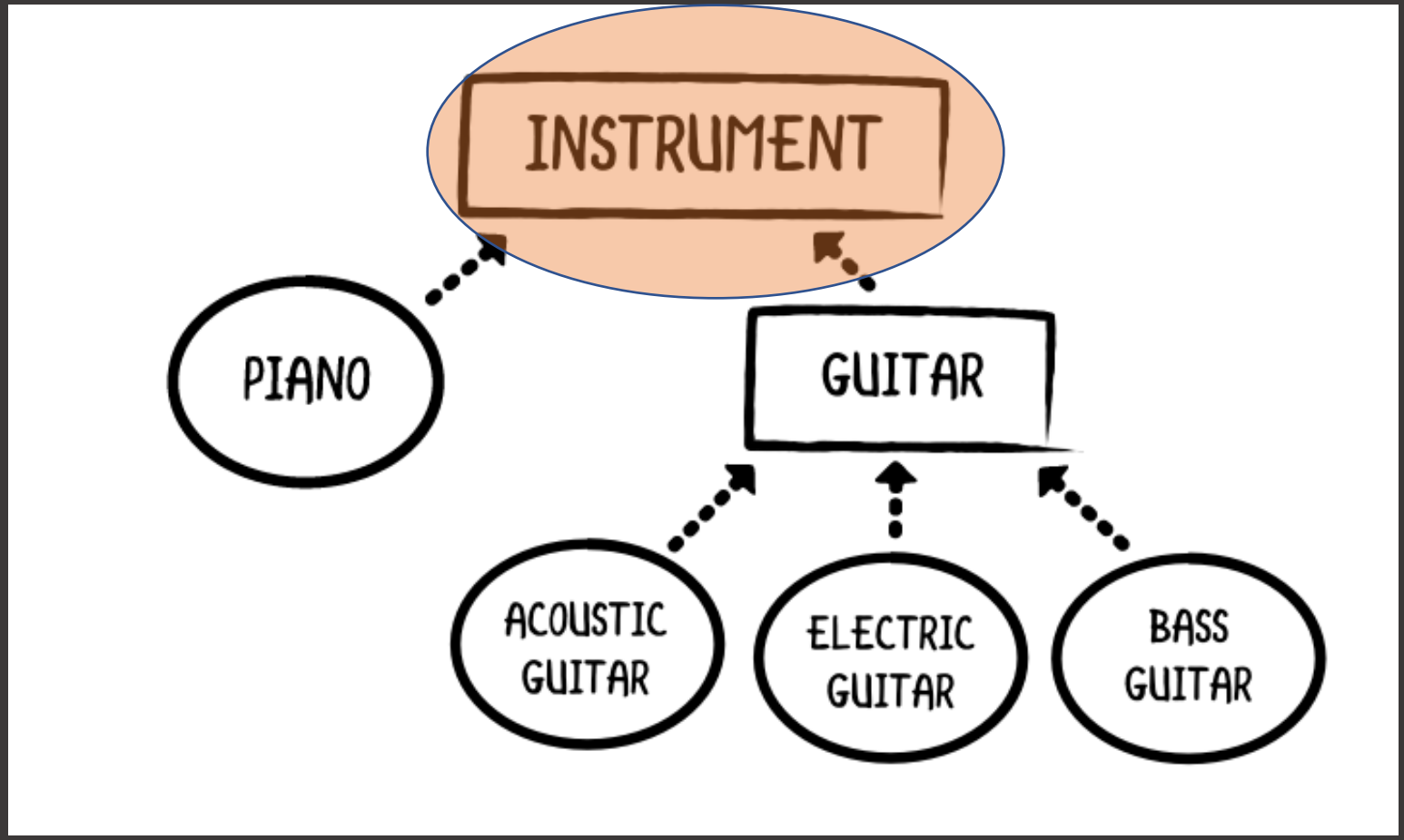
```
struct Resolution {
    var width = 0
    var height = 0
}

class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

The syntax for creating instances:

```
let someVideoMode = VideoMode()
```

Example,



https://www.raywenderlich.com/599-object-oriented-programming-in-swift

Instrument Class

```
class Instrument {
    let brand: String

    init(brand: String) {
        self.brand = brand
    }
}
```

base/root class

stored properties

constructor/initializer

Note 1: Initializer - purpose is to construct new instruments by initializing all stored properties
Note 2: Since the property and the parameter have the same name, the self keyword is used to distinguish between them.

Instrument Class

```
class Instrument {
    let brand: String

    init(brand: String) {
        self.brand = brand
    }

    func tune() -> String {
        fatalError("Implement this method for \(brand)")
    }
}
```

behavior

Music Class

```
class Music {
  let notes: [String]

  init(notes: [String]) {
    self.notes = notes
  }

  func prepared() -> String {
    return notes.joined(separator: " ")
  }
}
```

`Instrument` Class

`tune()` method is a placeholder function that crashes at runtime

- Abstract Class
  - not intended for direct use
  - must define a subclass that overrides the method to do something sensible instead of only calling fatalError()
- Encapsulation
  - Class types are said to encapsulate data (e.g. stored properties) and behavior (e.g. methods)

Swift Subclassing

MAD2 Oct 2019

NGEE ANN
SCHOOL OF INFOCOMM TECHNOLOGY

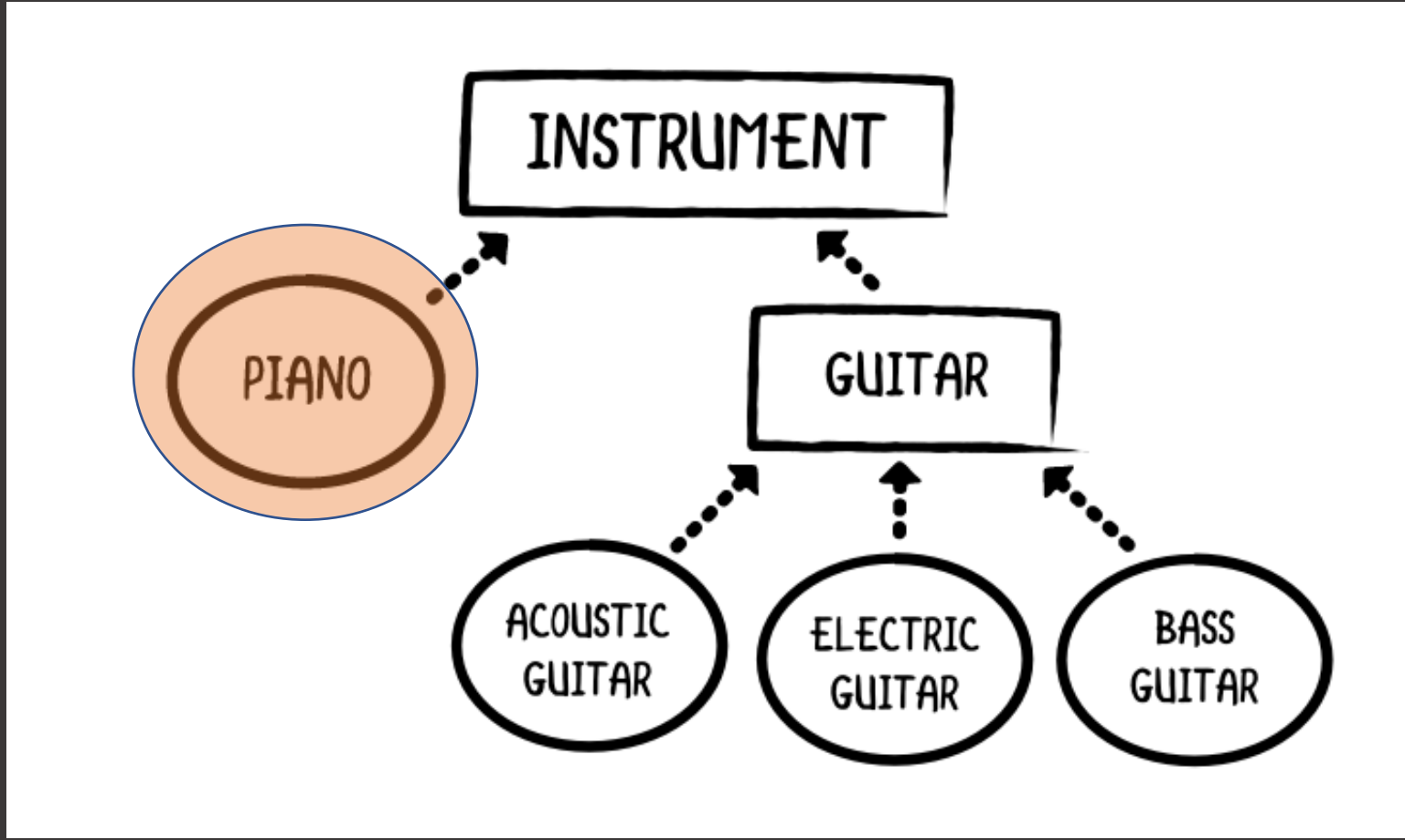
Sub-Classing

Subclassing is the act of basing a new class on an existing class.

The subclass inherits characteristics from the existing class, which you can then refine.

You can also add new characteristics to the subclass.

Example,



https://www.raywenderlich.com/599-object-oriented-programming-in-swift

Piano Class

```
class Piano: Instrument {
    let hasPedals: Bool
    static let whiteKeys = 52
    static let blackKeys = 36

    init(brand: String, hasPedals: Bool = false) {
        super.init(brand: brand)
        self.hasPedals = hasPedals
    }

//continue ...
```

Piano Class

```
//continue ...

    override
    func tune() -> String {
        return "Piano standard tuning for \(brand)."
    }

    override
    func play(_ music: Music) -> String {
        let preparedNotes = super.play(music)
            return "Piano playing \(preparedNotes)"
    }
}
```

Sub-Classing

- Piano class is created as a subclass of the Instrument parent class. All the stored properties and methods are automatically inherited by the Piano child class and available for use
- The associated values of their corresponding properties don't change dynamically, so the properties is marked as static in order to reflect this
- The initializer provides a default value for its hasPedals parameter
- The super keyword is used to call the parent class initializer after setting the child class stored property hasPedals
- The inherited tune() method's implementation is "over-rided" with the override keyword
- The inherited play(_:) method is "over-rided". This method, the super keyword is used to call the Instrument parent method in order to get the music's prepared notes and then play on the piano

Swift Initialization

- Swift classes use an initialization process called two-phase-initialization to guarantee that all properties are initialized before you use them.

Swift Instances

MAD2 Oct 2019

Instance

Instantiation is the creation of an instance

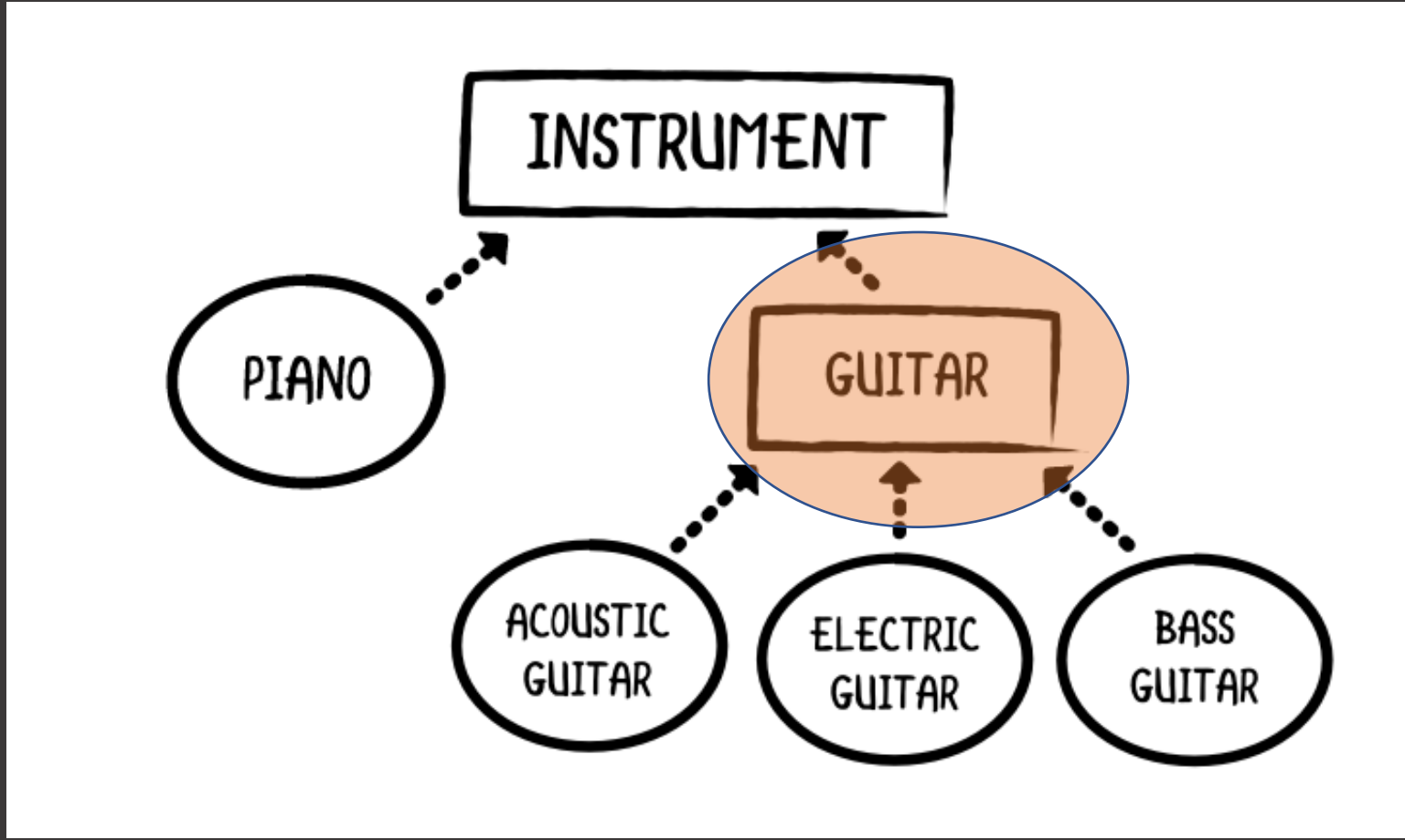
```
let piano = Piano(brand: "Yamaha", hasPedals: true)
piano.tune()

let music = Music(notes: ["C", "G", "F"])
piano.play(music, usingPedals: false)
piano.play(music)

Piano.whiteKeys
Piano.blackKeys
```


Overloading
(not shown in Piano class)

Example,



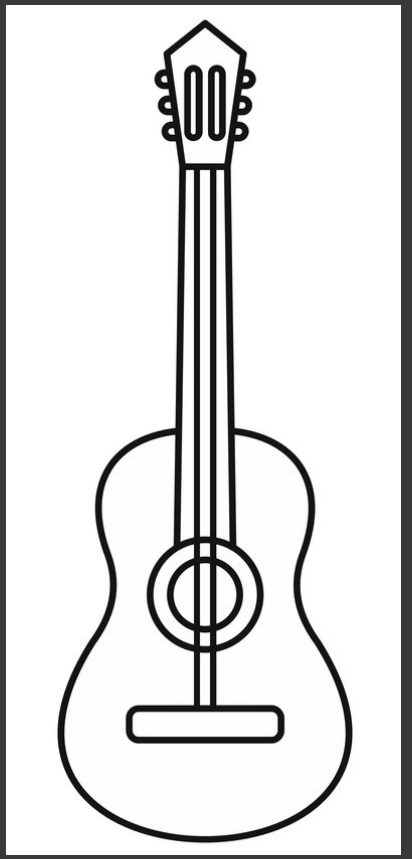
https://www.raywenderlich.com/599-object-oriented-programming-in-swift

Guitar Class

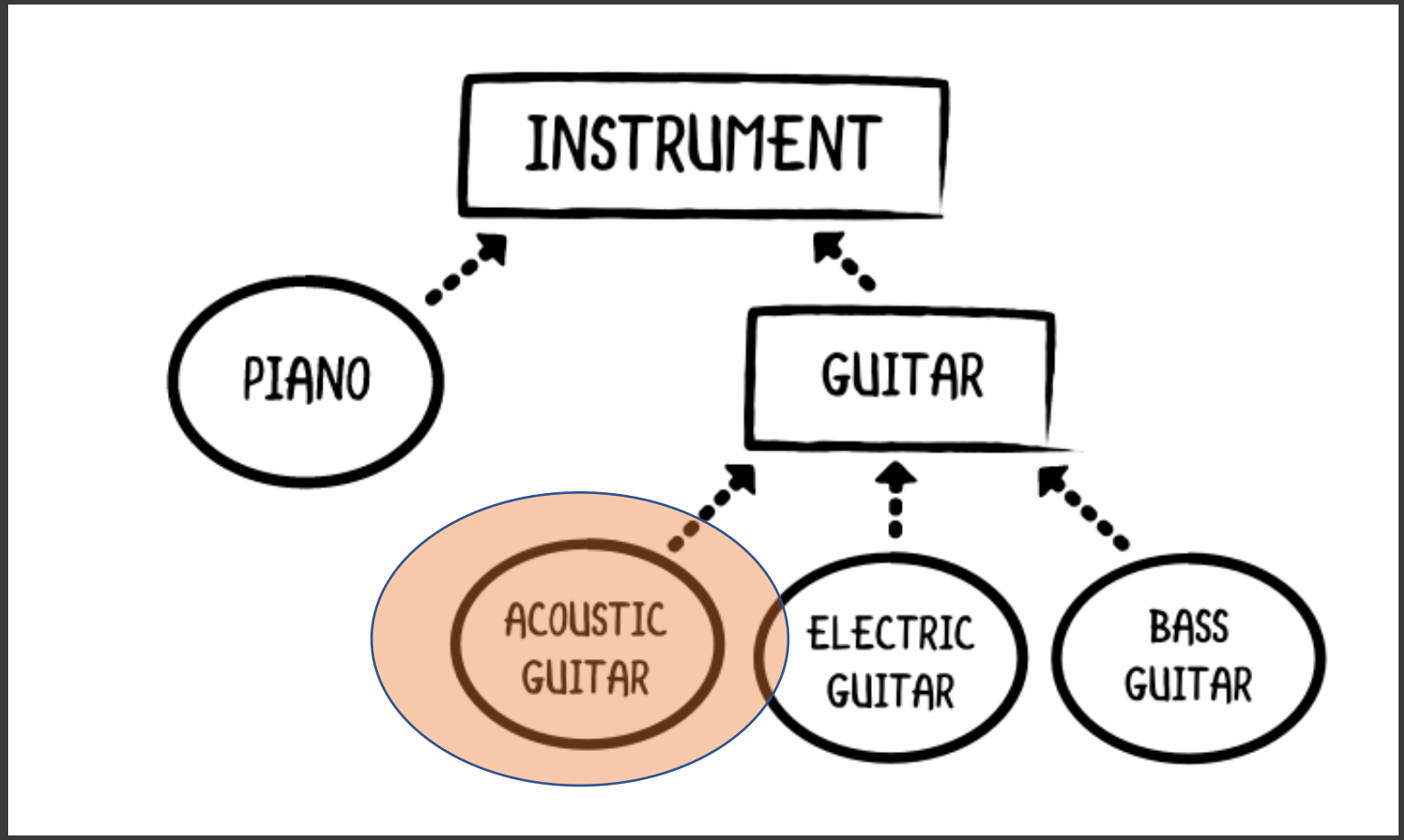
Intermediate abstract base class

```
class Guitar: Instrument {
    let stringGauge: String

    init(brand: String, stringGauge: String = "medium") {
    super.init(brand: brand)
        self.stringGauge = stringGauge
    }
}
```



Example,



https://www.raywenderlich.com/599-object-oriented-programming-in-swift

AcousticGuitar Class

```
class AcousticGuitar: Guitar {
    static let numberOfStrings = 6
    static let fretCount = 20

    override func tune() -> String {
        return "Tune \(brand) acoustic with E A D G B E"
    }

    override func play(_ music: Music) -> String {
        let preparedNotes = super.play(music)
        return "Play folk tune on frets \(preparedNotes)."
    }
}
```

Instance

```
let acousticGuitar = AcousticGuitar (brand: "Roland",
                                     stringGauge: "light")

acousticGuitar.tune()
acousticGuitar.play(music)
```

Summary

To be able to understand:

- Swift Language
  - Classes and Objects
  - Property
  - Subclassing