

Unit 2—Lesson 4:

Classes, Inheritance

```
class Person {  
    let name: String  
  
    init(name: String) {  
        self.name = name  
    }  
  
    func sayHello() {  
        print("Hello there!")  
    }  
}  
  
let person = Person(name: "Jasmine")  
print(person.name)  
person.sayHello()
```

Jasmine

Hello there!

Inheritance

Base class

Vehicle

Subclass

TandemBicycle

Superclass

Bicycle

Inheritance

Defining a base class

```
class Vehicle {  
    var currentSpeed = 0.0  
  
    var description: String {  
        return "traveling at \$(currentSpeed) miles per hour"  
    }  
  
    func makeNoise() {  
        // do nothing – an arbitrary vehicle doesn't necessarily make a noise  
    }  
}  
  
let someVehicle = Vehicle()  
print("Vehicle: \$(someVehicle.description)")
```

Vehicle: traveling at 0.0 miles per hour

Inheritance

Create a subclass

```
class SomeSubclass: SomeSuperclass {  
    // subclass definition goes here  
}
```

```
class Bicycle: Vehicle {  
    var hasBasket = false  
}
```

Inheritance

Create a subclass

```
class Bicycle: Vehicle {  
    var hasBasket = false  
}  
  
let bicycle = Bicycle()  
bicycle.hasBasket = true  
  
bicycle.currentSpeed = 15.0  
print("Bicycle: \(bicycle.description)")
```

```
Bicycle: traveling at 15.0 miles per hour
```

Inheritance

Create a subclass

```
class Tandem: Bicycle {  
    var currentNumberOfPassengers = 0  
}
```

Inheritance

Create a subclass

```
class Tandem: Bicycle {  
    var currentNumberOfPassengers = 0  
}
```

```
let tandem = Tandem()  
tandem.hasBasket = true  
tandem.currentNumberOfPassengers = 2  
tandem.currentSpeed = 22.0  
print("Tandem: \(tandem.description)")
```

```
Tandem: traveling at 22.0 miles per hour
```


Inheritance

Override methods and properties

```
class Train: Vehicle {  
    override func makeNoise() {  
        print("Choo Choo!")  
    }  
}
```

```
let train = Train()  
train.makeNoise()
```

Choo Choo!

Inheritance

Override methods and properties

```
class Car: Vehicle {  
    var gear = 1  
    override var description: String {  
        return super.description + " in gear \ (gear)"  
    }  
}
```

Inheritance

Override methods and properties

```
class Car: Vehicle {  
    var gear = 1  
    override var description: String {  
        return super.description + " in gear \$(gear)"  
    }  
}
```

```
let car = Car()  
car.currentSpeed = 25.0  
car.gear = 3  
print("Car: \$(car.description)")
```

Car: traveling at 25.0 miles per hour in gear 3

Inheritance

Override initializer

```
class Person {  
  let name: String  
  
  init(name: String) {  
    self.name = name  
  }  
}  
  
class Student: Person {  
  var favoriteSubject: String  
}
```



Class 'Student' has no initializers

Inheritance

Override initializer

```
class Person {  
    let name: String  
  
    init(name: String) {  
        self.name = name  
    }  
}  
  
class Student: Person {  
    var favoriteSubject: String  
    init(name: String, favoriteSubject: String) {  
        self.favoriteSubject = favoriteSubject  
        super.init(name: name)  
    }  
}
```

References

- When you create an instance of a class:
 - Swift returns the address of that instance
 - The returned address is assigned to the variable
- When you assign the address of an instance to multiple variables:
 - Each variable contains the same address
 - Update one instance, and all variables refer to the updated instance

```
class Person {  
    let name: String  
    var age: Int  
  
    init(name: String, age: Int) {  
        self.name = name  
        self.age = age  
    }  
}
```

```
var jack = Person(name: "Jack", age: 24)
```

```
var myFriend = jack
```

```
jack.age += 1
```

```
print(jack.age)
```

```
print(myFriend.age)
```

25

25

```
struct Person {  
    let name: String  
    var age: Int  
}
```

```
var jack = Person(name: "Jack", age: 24)  
var myFriend = jack
```

```
jack.age += 1
```

```
print(jack.age)  
print(myFriend.age)
```

25

24

Memberwise initializers

- Swift does not create memberwise initializers for classes
- Common practice is for developers to create their own for their defined classes

Class or structure?

- Start new types as structures
- Use a class:
 - When you're working with a framework that uses classes
 - When you want to refer to the same instance of a type in multiple places
 - When you want to model inheritance

Unit 2, Lesson 4

Lab: Classes.playground



Open and complete the exercises in Lab – `Classes.playground`

