Name: Nguyễn Văn Cường – Student ID: 20215006

Report – Lesson 3

[Demo Hello World 2](#_Toc178632580)

[3.1 Using Classes and Objects in Kotlin 3](#_Toc178632581)

[Create a class 3](#_Toc178632582)

[Add class constructors 4](#_Toc178632583)

[Learn about visibility modifiers 6](#_Toc178632584)

[Learn about subclasses and inheritance 7](#_Toc178632585)

[Compare abstract classes and interfaces 8](#_Toc178632586)

[Use interface delegation 10](#_Toc178632587)

[Create a data class 11](#_Toc178632588)

[Learn about singletons and enums 12](#_Toc178632589)

[3.2 Pairs/triples, collections, constants, and writing extension functions 13](#_Toc178632590)

[Create a Companion Object 13](#_Toc178632591)

[Learn about pairs and triples 13](#_Toc178632592)

[Learn more about collections 14](#_Toc178632593)

[Organize and define constants 16](#_Toc178632594)

[Understand extension functions 17](#_Toc178632595)

# Demo Hello World

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# 3.1 Using Classes and Objects in Kotlin

## Create a class

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

## Add class constructors

Step 1: Create a constructor

A screenshot of a computer

Description automatically generated

Step 2: Add init blocks

A screenshot of a computer program

Description automatically generated

Step 3: Learn about secondary constructors

A screenshot of a computer program

Description automatically generated

Step 4: Add a new property getter

A screenshot of a computer program

Description automatically generated

Step 5: Add a property setter

A screenshot of a computer program

Description automatically generated

## Learn about visibility modifiers

In Kotlin, classes, objects, interfaces, constructors, functions, properties, and their setters can have *visibility modifiers*:

* private means it will only be visible in that class (or source file if you are working with functions).
* protected is the same as , but it will also be visible to any subclasses.private
* internal means it will only be visible within that module. A [module](https://kotlinlang.org/docs/reference/visibility-modifiers.html#modules) is a set of Kotlin files compiled together, for example, a library, a client or application, a server application in an IntelliJ project. Note the usage of "module" here is unrelated to Java modules that were introduced in Java 9.
* public means visible outside the class. Everything is public by default, including variables and methods of the class.

## Learn about subclasses and inheritance

Step 1: Make the Aquarium class open

A computer screen shot of a program

Description automatically generated

Step 2: Create a subclass

A screenshot of a computer program

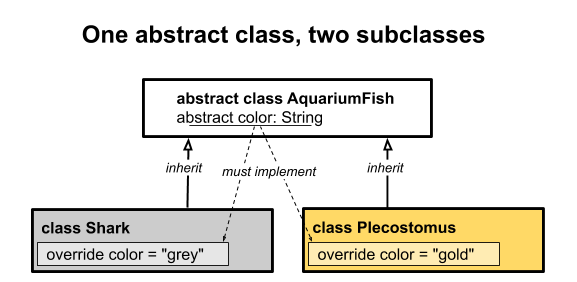
Description automatically generated

## Compare abstract classes and interfaces

Step 1. Create an abstract class

A screenshot of a computer

Description automatically generated



Step 2: Create an interface

A screenshot of a computer

Description automatically generated

A diagram of fish action

Description automatically generated

When to use abstract classes versus interfaces

* Use an abstract class any time you can't complete a class. For example, going back to the AquariumFish class, you can make all AquariumFish implement FishAction, and provide a default implementation for eat while leaving color abstract, because there isn't really a default color for fish.

## Use interface delegation

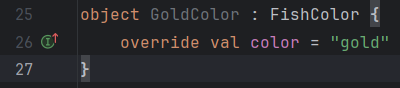
Step 1: Make a new interface

A screen shot of a computer program

Description automatically generated

Step 2: Make a singleton class

1. In **AquariumFish.kt**, create an object for GoldColor. Override the color.



Step 3: Add interface delegation for FishColor

A computer screen with white text

Description automatically generated

Step 4: Add interface delegation for FishAction

A computer screen with text and images

Description automatically generated

A screenshot of a computer

Description automatically generated

## Create a data class

Step 1: Create a data class

A screenshot of a computer

Description automatically generated

Step 2. Use destructuring

A screenshot of a computer

Description automatically generated

## Learn about singletons and enums

Step 1: Recall singleton classes

Step 2: Create an enum

A screenshot of a computer program

Description automatically generated

# 3.2 Pairs/triples, collections, constants, and writing extension functions

## Create a Companion Object

A screenshot of a computer program

Description automatically generated

## Learn about pairs and triples

Step 1: Make some pairs and triples

1. Create a pair

A screen shot of a computer

Description automatically generated

1. Create a triple

A screenshot of a computer

Description automatically generated

1. Create a pair where the first part of the pair is itself a pair.

A screen shot of a computer code

Description automatically generated

Step 2: Destructure some pairs and triples

1. Destructure a pair and print the values.

A screenshot of a computer

Description automatically generated

1. Destructure a triple and print the values.

A screenshot of a computer

Description automatically generated

## Learn more about collections

Step 1: Understand more about lists

1. Complete listings in the Kotlin documentation for both [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html) and [MutableList](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-mutable-list/index.html)

A screenshot of a white box

Description automatically generated

1. Sums up all the elements.

A screenshot of a computer program

Description automatically generated

1. Create a list of strings and sum the list.

A screenshot of a computer program

Description automatically generated

1. Using .sumBy() with a lambda function

A screenshot of a computer program

Description automatically generated

1. There's a lot more you can do with lists. One way to see the functionality available is to create a list in IntelliJ IDEA, add the dot, and then look at the auto-completion list in the tooltip. This works for any object. Try it out with a list.

A screenshot of a computer program

Description automatically generated

1. Choose listIterator() from the list, then go through the list with a for statement and print all the elements separated by spaces.

A screen shot of a computer program

Description automatically generated

Step 2: Try out hash maps

1. Create a hash map



1. Retrieve the scientific name value based on the common fish name key, using get(), or even shorter, square brackets [].

A screen shot of a computer program

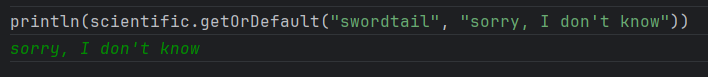
Description automatically generated

1. Try specifying a fish name that isn't in the map.

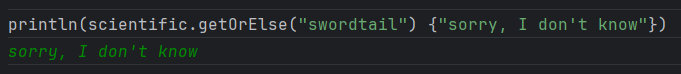
A screen shot of a computer screen

Description automatically generated

1. Try looking up a key that has no match, using getOrDefault().



1. Change your code to use getOrElse() instead of getOrDefault().



## Organize and define constants

Step 1: Learn about const vs. val

The value for const val is determined at compile time, whereas the value for val is determined during program execution, which means, val can be assigned by a function at run time.

That means val can be assigned a value from a function, but const val cannot.

A screen shot of a computer program

Description automatically generated

Step 2: Create a companion object

The basic difference between companion objects and regular objects is:

* Companion objects are initialized from the static constructor of the containing class, that is, they are created when the object is created.
* Regular objects are initialized lazily on the first access to that object; that is, when they are first used.

There is more, but all that you need to know for now is to wrap constants in classes in a companion object.

## Understand extension functions

Step 1: Write an extension function

1. String is a valuable data type in Kotlin with many useful functions. But what if we needed some additional String functionality that wasn't directly available? For example, we might want to determine if a String has any embedded spaces.

A screen shot of a computer

Description automatically generated

1. You can simplify the hasSpaces() function. The this isn't explicitly needed, and the function can be reduced to a single expression and returned.

A black screen with white text

Description automatically generated

Step 2: Learn the limitations of extensions

1. Try adding extension functions that call a property marked private.

A black background with white text

Description automatically generated

**Note:** Extension functions are resolved statically, at compile time, based on the type of the variable.

1. Examine the code below and figure out what it will print.

A screen shot of a computer program

Description automatically generated

Step 3: Add an extension property

1. Add an extension property isGreen to AquariumPlant, which is true if the color is green.
2. Print the isGreen property for the aquariumPlant variable and observe the result.

A screen shot of a computer program

Description automatically generated

Step 4: Know about nullable receivers

1. Define a pull() method that takes a nullable receiver.

This is indicated with a question mark ? after the type, before the dot. Inside the body, you can test if this is not null by using ?.apply.

A screen shot of a computer program

Description automatically generated

1. In this case, there is no output when you run the program. Because plant is null, the inner println() is not called.