Topic 6: Kotlin intro

Why Kotlin: Java is comically wordy, it requires lots of files, it has the weird "primitive types", it is almost paranoid about saftey, it is not friendly to functional programming (despite lambdas). But OTOH, take advantage of all of Java's class libraries and the fact that lots of people know Java. The fact that it starts with a K is NOT a mistake. (C was the successor to B)

Contrast Java and Kotlin Hello world

    see Topic06/hello/

To run a kotlin program — similar to java — compile into a target language then run using that target language system. Most commonly, compile to Java (like javac). Can also generate javascript and machine native code

To compile/run a kotlin program

```
kotlinc hw.kt -include-runtime -d hello.jar
```

where hw.ht contains the main function

this says to compile the program and put everything (and a lot more) into a jar file. jar is a java extension of tar. Major extension is that the jar file can include a file that says whent he main method to run is … MANIFEST.MF

```
java -jar hello.jar
```

a java jar file is basically a UNIX tar file with a special file that tells java what to do.

Note that this jar file is LARGE

kotlin shows its youth, a couple of 1.5 Meg is no biggie but it is a massive expansion from the 4 lines of code.

Object lives and bindings generally follow Java

- No "primitive type" everything is an object. (Why make this choice? Is it a good one)
- Has golbals that are statically allocated .. more or less given that programs run in JVM.
- Heap based allocation and Garbage collection. But does not have Java "new" to signal heap allocation
- Allows (encourages) nested subroutines — see below
- static scope
  - block scope
  - vars may have aliases
- functions are first class variables.

- function names may be overloaded, but not used as much in java thanks for optional and named parameters in functions
- has dynamic method dispatch

var and val.
var == variable. The thing is allowed to change
val == value. NOT allowed to change. equivalent to Java final

- for functional programming will almost exclusively use val
- function parameters in kotlin are always val!

Types:

Everything in kotlin is a class — does away with the java primitive types.
var x : String // IS THIS LEGAL // yes but causes problems — everything initializes to null

so the first use of x must be assignment

val aaa: String // LEGAL also, but cannot be legally used as it is null — more on null

var x : Int = 7
var x = 7

see initial/init.kt

Type inference:

for var/val much line Go
val aa = 7 will infer that aa is of type Int, etc
val aa: Int = 7
error val aa:Int = 7.0

Kotlin is a reference-model language — everything is a reference. Likewise Kotlin is pass-by-reference, EXCEPT that you are not allowed to change function params

initial/nc.kt

Note that you cannot change what is pointed to, but can change internals

also initial/nc.kt
Point mutable and immutable objects (mostly lists)

Go: immutable? No? (strings)?

Mutable Go objects:

- arrays and slices
- maps
- channels
- closures which are capturing at least 1 variable from the outer scope

Immutable Go objects:

- interfaces
- booleans, numeric values (including values of type int)
- strings
- pointers

How to test for immutability in Go?
Strings see go_immut/imm.go

Java: String, Numbers are immutable

for instance, String has getChar, but not setChar (same idea as Go)

Will be using Kotlin for functional programming so will largely ignore things like all loop constructs

Output

just “println”
To get formatted output use Java String.format(“FORMAT STRING”, args)

FORMAT STRING is much the same as Go fmt.Printf, (but no %v)

There is another way similar to Bash printing, use it if you want.

function and methods
- may be declared inside a class (a method) or outside (a function)
- fun xx(p1:type1, 22: type2) : returnType { … }
- fun yy(p1:type1) = expression
- for example:
  - fun lesser(p1:Int, p2:Int) = if (p1>p2) { p2 } else { p1 }
  - Here note
    - NO RETURN.
    - No return value type (kotlin infers it)
    - if statement can return a value and
      - when it does kotlin requires an else clause (so not return null)
    - for functions that are just an expression Kotlin will infer type
      - fun lesser(p1:Int, p2:Int) = if (p1>p2) { p2 } else { p1.toDouble() }
        - here return type is "Any", the Kotlin equivalent to Java

Object

will come back to functions as there is a LOT more

No "tuple assignment" as in Go.   When want to return multiple values from function, prefered approach is to use a "data class".

Classes
- do not require separate file if public
  - Note in class declaration may use var/val   (unlike functions)
- default is "public final" — unlike java where unspecified is "package" in kotlin it is public
- see classes/define.kt
  - GO THROUGH THIS EXAMPLE SLOWLY!!!!
- to make classes NOT final add "open" to declaration
- when overriding a function must say so. (Unlike java optional @Override annotation
- automatic constructor for items listed in the class declaration
  - can have other constructors
  - classes/twocon.kt

"Data classes"
- a standard class Plus
  - automatic generation of getters (and setters if properties are not val)
  - automatic useful equals (not pointer identity)
  - see classes/datacl.kt

Any — equivalent to "Object"

Null safety!
- "types default to non-nullable. However, if something can produce a null result, you must append a ? to the type name to explicitly tag the result as nullable
- elvis/elvis.kt
  - again, slow!

Operator overloading
    can define the behavior of + on a class
    can redefine == on a class!

Exceptions
    just like Java “try .. catch”
    We will largely ignore exceptions to the extent we can
    No required try/catch in Kotlin — is this good? Will come back to

Lists
    val ints = listOf(1,2,3,4,5)
    OR
    val ints = mutableListOf(1,2,3,4,5)
    ints.add(6)

functions
    fun name(varname:varType):rtnType { stuff }
    fun name(varname:varType) = statement or expression
        if expression Kotlin infers type
    val fff = fun(varname:vartype):returnType { stuff }
        anonymous function

    see funcs/funclist.kt
        uses tail recursion to print the elements of a list
        * Four things in this program .
            * first, a function using generics
            * second, a recursive function
            * third, tell the compiler to optimize for tail recursion
            * fourth, head and test functions on a list to create a recursive function to
step through a list

        Issues with this program
            1. depending on implementation of list, this could be very inefficient
            2. what happens when list is null?
            3. Kind of boring tail recursion as function does not return anything
    see funcs/funclist2.kt
        use tailrec to sum list
        * 1. Kotlin allows default values for function parameters
        * 2. Given default values, kotlin allows named parameters in function call
        * 3. The "elvis" operator. "?". This allows the parameter to
        * be null. It also REQUIRES that null be explicitly handled
        * This pairling allows Kotlin to be "null safe"

funcs/funclist3.kt
* This one, instead of gettign the head and recurring on
* the rest of the list, does everything positionally. Depending
* on the implementation of the list, this version may be much
* faster, or must slower, than the other version

funcs/funclist4.kt
Another list summer. This time use an internal recursive function so
* that you check base conditions before going into recursion.

* This should, this be a little quicker than the previous version

funcs/funclist5.kt
final version of summer. Here rather than returning sum return a function that sums the list.
* Note the use of a closure on b3.
* To return a function, it must be anonymous

funcs/funclist6.kt
Anonymous funs and recursion.
- same game as Go with predeclare does not quite work
  - because of Definite Assignment
  - Kotlin has an out “lateinit” tag on a variable
  - OTOH, because kotlin allows nesting of named functions this is not as much of an issue. (Still can be very useful if have recursive returned functions (Ouch and yuck)

Inheritance and extension
- class must be marked as “open” to do inheritance
- functions must be marked as open to be overridden

- inheritance and extension
  - see classes/clss.kt
- Extension function do NOT behave like member functions
  - inherited?
  - static dispatch!
  - see classes/extn.kt