Topic 8: Types

2 basic questions: what / why

What??
bits are untyped!!!
most basic: a type defines how many, and how, to interpret bits.
also—the set of operations that are allowed it.
  primitive types “built in” — usually at hardware level
different from Java int, …
  composite types

Why?:
1. Types supply context — Useful for compiler as it specified what to do
2. Limit what is allowed to be done
3. Make the program more readable to user — effectively a form of documentation —
   especially useful when there are a lot of types (OO langs). So why type inference?
4. Compile time optimization

Type system:
  1. mechanism to define types
  2. Definition of
    type equivalence
    structural vs name
    type compatibility
    what is allowed with what
    for + suppose one is Int, what is the other allowed to be
    in a weakly typed anything
    Go, Java, Kotlin
    type inference (may not be available in some langs)

“primitive types” vs composite types
  composites in next chapter
  struct, array, set, pointers, list, file
  Primitive — int (at what precision?) should a lang care about precision?
  character? ASCII, 16-bit ascii? rune? UTF-8
  enums — primitive or composite. Why???? How??
  consecutive integers? Powers of two?

Do functions have types?
  Why?
  If they are first or second class, they do.

Strongly typed — language prohibits even trying to do something that is not allowed for a type.
Thrown out at compile
Weak — usually implies doing more work at run time — strong==fast
  for instance, to make the “+” work, javascript must do what?
  can interpreted language be strongly typed?
  realistically this is a spectrum. Language may have holes …
  weakly typed — ex language allows application of operators when it does not make
  necessarily make sense. For instance, javascript is weakly typed (and dynamically typed)
  f = some function
  q = 5 + f
Go? Kotlin? Javascript?
Statically typed — strong AND type checking is a compile time.

Polymorphism
Generics == “Explicit parametric polymorphism” implemented at compile time!!!
subtype polymorphism — common in OO languages — allow uses of subtype where base type is specified.

Lots of types
Basic type: integer, float ...
Integers
Java: byte, short, int, long. Also, Byte, Short, Integer, Long, BigInteger!!!
Kotlin: Byte, Int, Long, Short
What does kotlin get by dumping primitive types? Cost?
Go: [u]int[8,16,32,64]
Why so many int types???
char — what is a char?
one byte — ASCII
char in c
2 bytes — UNICODE16 — JAVA
char in Java
Up to 4 bytes — UTF8 — variable
0xxxxxxx — 1 byte — plain old ASCII
110xxxxx 10xxxxxx —-
1110xxxx 10xxxxxx 10xxxxxx
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
“rune” in Go

is String a basic type?
in Java? C? Go?
Java — NO…it is a class
(Are classes in java.lang really “basic” to Java??
You cannot do ANYTHING without java.lang.Object
To know would have to look at implementation of String class
C — definitely NOT
Go — from book “a string contains an array of bytes that, once created, is immutable”
This indicates that string is a composite type
Going further Go explicitly mirrors string functions with byte array functions
OTOH — “The underlying type of every constant is a basic type”
boolean, string or number”

Enumerated types
What: a type that has a specific, finite (usually small), and bounded set of possible values.
Why?
Go: enum_go/enum.go
They do not really exist like in other languages so you get little benefit
Kotlin: enum_kt/emun.kt
real enums
    checking and assignment
switch (when) expression

Composite (aggregate) types
Array

in Go array size is set at compile time!!
    Why???
    func t5() {
        ar := [3]int{1,2,3}
        fmt.Println(ar[5])
        for i:=0; i<5;i++ {
            fmt.Println(ar[i])
        }
    }

Arrays can be allocated on stack!! Faster.
In above example, bounds check at compile time???
Arrays contain objects — stick with value-model language
Java, Kotlin?

Slice / ArrayList
Go: “unlike array alements, the elements of a slice are indirect”
slices contain references!!!
slices are built on top of arrays! How, given that arrays have a
fixed size at compile time? (Trick reserved to language builders)

structs
sets
lists — no ordering
    traditionally heavily used in functional programming
    IMHO — because Lisp did it (Lisp == LISt Processing)
files

Type checking
obvious and handled by compiler in Java
Go,Kotlin often do not require explicit types
    type inference
        why have type inference?
            you lose the readability of the implicit documentation
what do you gain?
When are two types the same???
structural vs name equivalence
structural
    same order, or just same number and kind?
    what work needs to be done to get this?
    what does Go/Kotlin do?
        why not use structural equivalence?
name
what about type aliases?
see topic08/equals_kt/equals.kt
    cannot easily override == to give structural equivalence

what are Go, Kotlin, Java
Go: equiv_go/equiv.go
    strict name equivalence
Kotlin: equiv_kt/equiv.kt
loose name equivalence
    casting allowed
    Java: no typealias, otherwise like Kotlin

Casting — converting from one type to another
    in strongly types languages “weird” casts are not allowed
func t5() {
    str := "abc"
    fmt.Println(str)
    var num int64
    num=40
    fmt.Println(num)
    num = int64(str) // Compiler flags as not allowed
}

    Problem is that casting requires changing bits and you have to know how.
    what is the problem with changing bits???
    Some langns allow “non-converting” casts. That is, do not change bits just interpret
    bits differently. What is problem? (C does this)

type coercion
    allow 3+2.4 without explicit casing
    pros/cons

Type inference:
    kotlin, go does it:
        infer_go
        infer_kt
    Advantages / disadvantages of type inference (in a strongly typed language)???

When are two objects the same?
    Deep vs shallow checks?
        Java == vs equals
        Kotlin == vs ===
            Note: in Kotlin equals method overrides ==!
    Deep vs shallow assignment
        Only applied to reference model languages
            see copy_go
        Value languages effectively always deep copy
        Shallow
            copy and assign pointer
            make a new copy of object and assign.
        Kotlin,Java — shallow. Why???
            equal_kt
    KOTLIN: For values represented by primitive types at runtime (for example, Int), the ===
    equality check is equivalent to the == check.