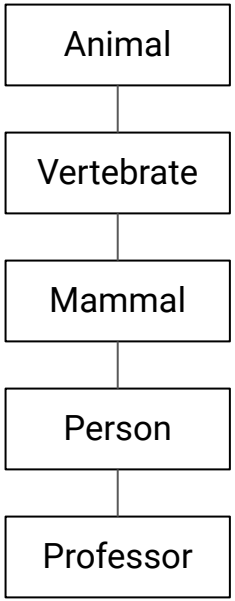
Type Hierarchy

Ken Q Pu, Faculty of Science, Ontario Tech University

Type hierarchy

Consider the IS-A relation:

- A professor is a person
- A person is a mammal
- A mammal is a vertebrate
- A vertebrate is an animal



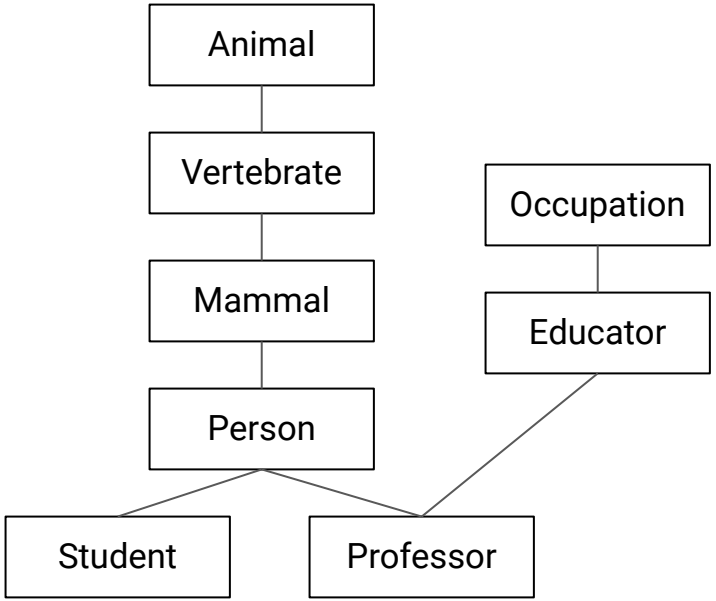
Type hierarchy

The types form a hierarchy based on the IS-A relation. The hierarchy is a general directed acyclic graph (**DAG**).

Subtypes:

- S is a subtype of T if all instances of S is also an instance T.
- Objects of S have all the methods of T, but may have more.

S <: T



Type checking is hard...

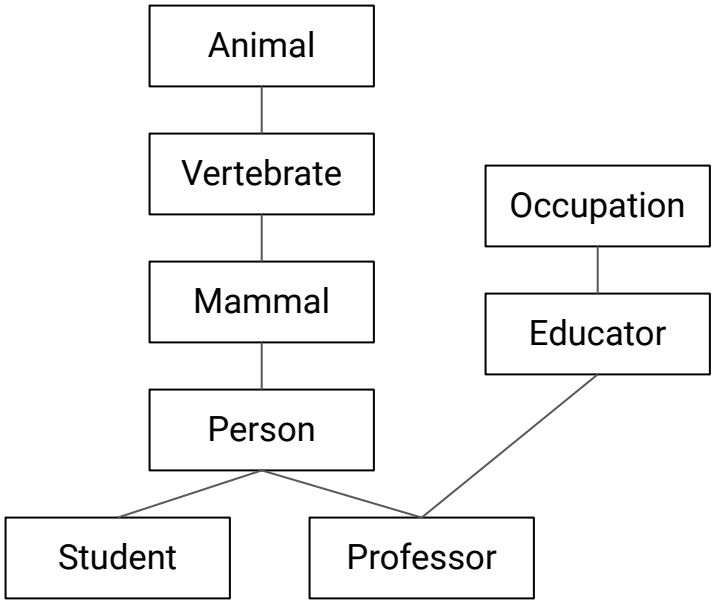
Consider a function:

```
findSalary: Occupation → Currency
```

Why is?

✅ `findSalary(obj : Professor)`

❌ `findSalary(obj: Student)`



Constrained generic types

A group of same creatures study in zoology?

❌ List<Animal> Allows *invalid* instances: [dog1, cat2, monkey3]

❌ List<Mammal> Misses *valid* instances: [frog1, frog2, frog3]

❌ List<T> Inappropriate parameter type: `List<Professor>`

We need to constrain the generic type parameter

`List<T>` where `Mammal <: T <: Animal`

Ken Q Pu, Faculty of Science, Ontario Tech University

Kotlin: a statically typed language

Kotlin

- Expressive type system
- Runs in JVM, Android and iOS
- Rich ecosystem

We will focus on:

- Type system of Kotlin
- Functional programming
- Novel language features