UNIVERSITY OF
Southampton
School of Electronics
and Computer Science

COMP2212
PROGRAMMING LANGUAGE CONCEPTS

Julian Rathke and Pawel Sobocinski

NOMINAL SUBTYPING

NOMINAL SUBTYPING

- Types are distinguished by their names - even if they represent the same structure!
- e.g.: `struct Foo = {x : int , y : int}` and `struct Bar = { x : int , y : int }` refer to *different* named types.
- A function that expects a **`Foo`** value cannot accept a **`Bar`** value, even though they are structurally identical.
- Type names hide the underlying structure.
- Consider the following example:
  - `type Address= { name : String , address : String }`
  - `type Email = { name : String , address : String }`
- These two types are conceptually different and are intended to be used differently.
- Distinct names for the same structure allows the programmer to enforce the distinction at the type level.

NOMINAL SUBTYPING RELATION

- Need to explicitly specify the relationships between the named types.
- A common approach is to provide the subtyping relation with the declaration of the type names themselves. e.g. `type Foo subtypes Bar = { ... }`
- Or as we know from OO languages: `class Foo extends Bar { ... }`
- As a type rule this looks something like (depending on syntax):

$$\frac{\text{type } T \text{ subtypes } U = \{..\}}{T <: U}\ \textsc{SubDecl}$$

- Such declarations alone aren't typically enough to define the subtyping relation
- We also ask that this relation is **reflexive** and **transitive**

$$\frac{}{T <: T}\ \text{SubRefl} \qquad \frac{T <: U \quad U <: V}{T <: V}\ \text{SubTrans}$$

- Java adds an extra rule in that states every type is a subtype of Object.

STRUCTURAL PROPERTIES VIA NOMINAL SUBTYPES

- Of course, just declaring that one type is a subtype of another may break the important subsumption property that for `T <: U` every value of type `T` can be considered as a value of type `U` also.
- e.g. if `T` is a pair type and `U` is a triple — which two of the three values do we take?
- Java overcomes such difficulties by only allowing subtyping between a single form of structuring - classes.
- Inheritance simply forces that every member in the supertype also exists in the subtype.
  - This approach is typical of nominal type systems.

NEXT LECTURE: STRUCTURAL SUBTYPING