

Subclassing and Subtyping

Dr. Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

Objectives

You should be able to ...

The idea of a subtype and a subclass are very closely related, but there is a subtle difference we would like to cover.

- ▶ Explain the difference between a subclass and a subtype.
- ▶ Explain the terms *covariant* and *contravariant*.
- ▶ Identify if two types have a subtyping relationship.

How do Types Relate?

- ▶ How can you tell if one type is a *subtype* of another?
 - ▶ Are integers subtypes of floats? (Or vice-versa?)
 - ▶ Characters / strings?
 - ▶ Squares / shapes?

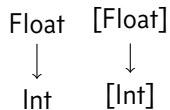
How do Types Relate?

- ▶ How can you tell if one type is a *subtype* of another?
 - ▶ Are integers subtypes of floats? (Or vice-versa?)
 - ▶ Characters / strings?
 - ▶ Squares / shapes?
- ▶ An integer is a kind of float, so we can say that integer is a subtype of float.

Float
↓
Int

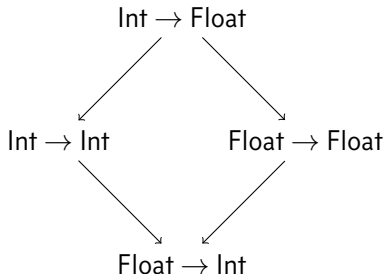
Covariance

- ▶ Some types take parameters, such as lists and trees.
- ▶ If the subtype relationship varies according to the input type, the type is said to be *covariant*.
- ▶ “Most” types containing parameters are covariant.



Functions

- ▶ Functions are an important exception!
 - ▶ The function type is covariant with respect to the output.
If we are expecting a function that outputs a float, I can give you a function that outputs an integer without breaking anything. The reverse is not true!
 - ▶ The function type is *contravariant* with respect to the input.
If we are expecting a function that takes a float, providing a function that takes an integer will fail or truncate the input.



The Trouble with Objects ...

Actually, there's more than just this one!

```
1   public class A {
2       public A foo(A x) { ... }
3       public A bar() { /* calls foo ... */ }
4   }
5   public class B : A {
6       public B foo(B x) { ... }
7   }
```

- ▶ B.bar inherits from A.
- ▶ But B.foo overwrites A.foo.
- ▶ When A.bar calls B.foo, what will happen?

Conclusions

- ▶ Objects have a lot of flexibility and allow us to create useful abstractions.
- ▶ They can be implemented using functions. Users of functional programming languages tend to avoid them.
- ▶ These are useful enough in practice, and difficult enough to implement, that most modern languages now include them, including OCaml. (That's where the O comes from.)
- ▶ Inheritance can be tricky.