

Compare/contrast to what our "theory" allows:

1. Types are class names and subtyping are explicit subclasses

2. A subclass can add fields and methods

 A subclass can override a method with a covariant return type

 (No contravariant arguments; instead makes it a nonoverriding method of the same name)

(1) Is a subset of what is sound (so also sound)

(3) Is a subset of what is sound and a different choice (adding method instead of overriding)

Autumn 2019

CSE341: Programming Languages

4

## Classes vs. Types

- A class defines an object's behavior

   Subclassing inherits behavior and changes it via extension and overriding
- A type describes an object's methods' argument/result types
- A subtype is substitutable in terms of its field/method types
- These are separate concepts: try to use the terms correctly
- Java/C# confuse them by requiring subclasses to be subtypes
   A class name is both a class and a type

5

Confusion is convenient in practice

Autumn 2019 CSE341: Programming Languages

### **Optional:** More details

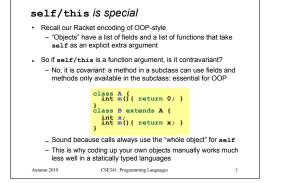
Java and C# are sound: They do not allow subtypes to do things that would lead to "method missing" or accessing a field at the wrong type

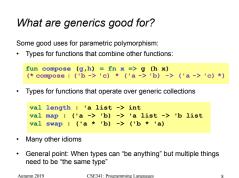
Confusing (?) Java example:

- Subclass can declare field name already declared by superclass
- Two classes can use any two types for the field name
- Instances of subclass have two fields with same name
- "Which field is in scope" depends on which class defined the method

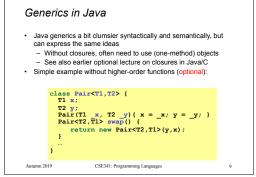
Autumn 2019 CSE341: Programming Languages

6





Autumn 2019 CSE341: Programming Languages



# Subtyping is not good for this · Using subtyping for containers is much more painful for clients

- Have to downcast items retrieved from containers
- Downcasting has run-time cost
- Downcasting can fail: no static check that container holds the type of data you expect
- \_ (Only gets more painful with higher-order functions like map)

#### class LamePair { Object x; Object y; LamePair(Object \_x, Object \_y) { x=\_x; y=\_y; } LamePair swap() { return new LamePair(y,x); } // error caught only at run-time: String s = (String) (new LamePair("hi",4).y); Autumn 2019 CSE341: Programming Languages 10

# What is subtyping good for?

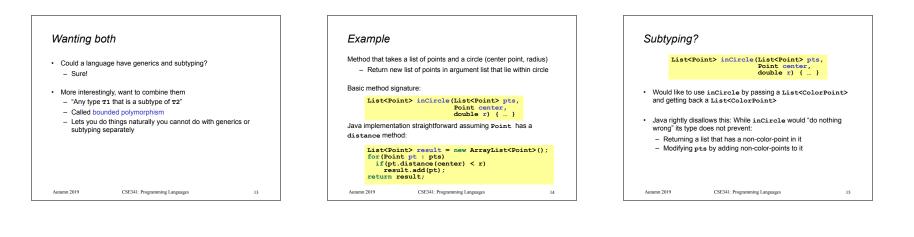
#### Some good uses for subtype polymorphism:

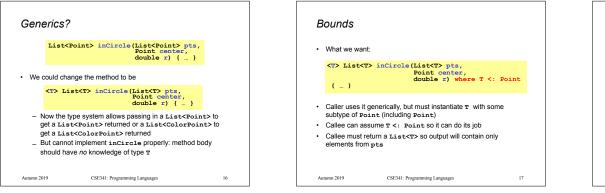
- · Code that "needs a Foo" but fine to have "more than a Foo"
- · Geometry on points works fine for colored points
- · GUI widgets specialize the basic idea of "being on the screen" and "responding to user actions"

11

Autumn 2019 CSE341: Programming Languages

Awkward in ML
ML does not have subtyping, so this simply does not type-check:
<pre>(* {x:real, y:real} -&gt; real *) fun distToOrigin ({x=x,y=y}) =     Math.sqrt(x*x + y*y)</pre>
<pre>val five = distToOrigin {x=3.0,y=4.0,color="red"}</pre>
Cumbersome workaround: have caller pass in getter functions:
<pre>(* ('a -&gt; real) * ('a -&gt; real) * 'a -&gt; real *) fun distToOrigin (getx, gety, v) =     Math.sqrt((getx v)*(getx v)</pre>
<ul> <li>And clients still need different getters for points, color-points</li> </ul>
Autumn 2019 CSE341: Programming Languages 12





Real Ja	va	
The actua	I Java syntax:	
Li fo	<pre>tends Pt&gt; List<t> inCircle(List<t> pts,</t></t></pre>	
Java there static che	backward-compatibility and implementation reasons, in is actually always a way to use casts to get around the cking with generics r without bounded polymorphism	
Autumn 2019	CSE341: Programming Languages	18