Sub types

A class describes an object and gives explicit code for the methods.

An interface only names (some of) the methods that an object implementing an interface.

TYPES describe properties of objects (and other data) and classifies data. The point of typechecking: to guarantee that the user of an object only invokes methods that are actually there. This can be checked at compile time (before the program is run).

Suppose I have a method called add

```
v.add(foo u) {
    value = this.value + u.getvalue();}
```

So foo names a class & u is an object of that class. I am expecting u to have a method called getvalue which returns a number.

foo is (in part) a type statement. The add method says "give me an object that has all the methods that a foo class object has." Now what if you pass an object that has all the methods declared in foo and more? This is not a problem.

We say A is a subtype of B if whenever you need an object of type A you can accept a type B object

A class declaration says many things:
(a) how to generate & initialize new objects
(b) the ~~are~~ names ~~of me~~ & signatures of methods
(c) the actual code of methods
(d) - - - -

(b) gives type information
(c) gives something much more detailed

If class ~~B~~ A extends class B by adding new methods then A has all the method names of class B so whenever you ask for something of class B you will be happy with ~~class~~ A. something of class A. So:

A is a subtype of B

It does not matter that some of the code for methods in B got altered.

~~Nothing~~ In Java ~~can~~ to enforce the restriction "give me something ~~in~~ of the ~~class~~ class B which is not of the class "A" you have to do awkward things with instanceof : Very bad style!

Now is there a way of specifying a pure type?
Interfaces : Just name the methods & give their signature NO CODE.

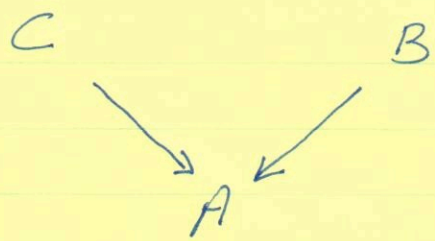
POLYMORPHISM : Having many types.

C
↓
B
↓
A

B extends C

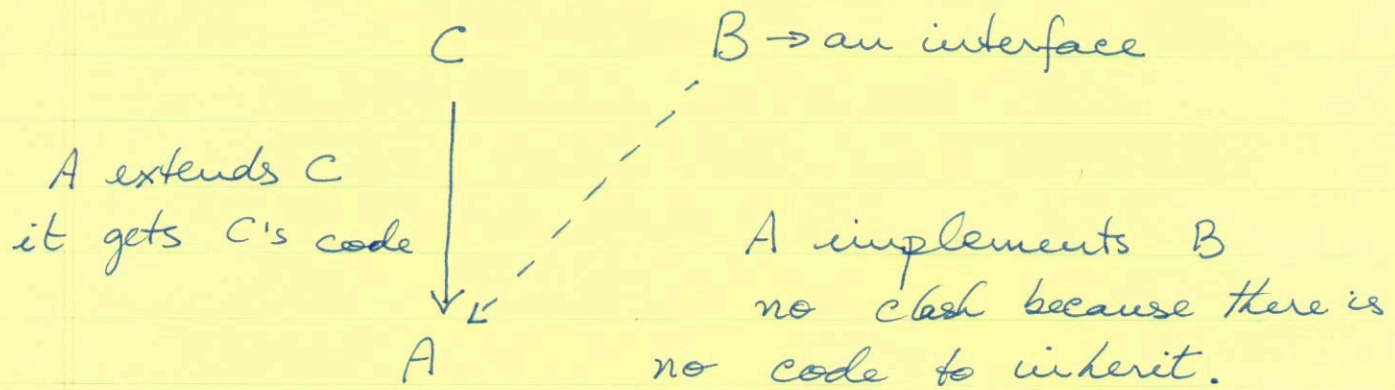
A extends B

What if we want ?




B, C may have clashing code for the same method name.

Instead




Why would we want <u>not</u> to inherit code?
Because we want <u>general</u> methods
that work on a variety of objects.

Plotting the graph of a function:

Suppose I have a function $f$: Reals → Reals

⓪ Choose the range you want to plot & scale

① Draw axes & mark $x_0, x_1, x_2, \ldots x_n$

② Calculate $f(x_0)$ & draw a point at $(x_0, f(x_0))$

③ Calculate $f(x_1)$ & draw a point at $(x_1, f(x_1))$

Hmm! Iteration!!

Set $i = 0$, choose step value $\delta x$

→ Calculate $f(x_0 + i * \delta x)$ & plot point at $(x_0 + i * \delta x, f(x_0 + i * \delta x))$

increment $i$ & check range

The point: This is <u>generic</u> in $f$. I do not want to write a new program for every $f$. The function $f$ has to be a <u>parameter</u>. We want an object with a method called "f" which takes a real & returns a real. You do not want to inherit the code for "f" you want to create many different f's.

```
public interface plottable
  { public double f(double x); }
public
  class Squarfun implements Plottable
  { public double f(double x){return x*x;}
    public String toString(){return "f(x)=x*x";}
  }
```