Last time: encapsulation/modularity

interface: defines how clients interact with a module

Java "interface": language feature for making interfaces for classes, signatures for public methods

Today: subtyping

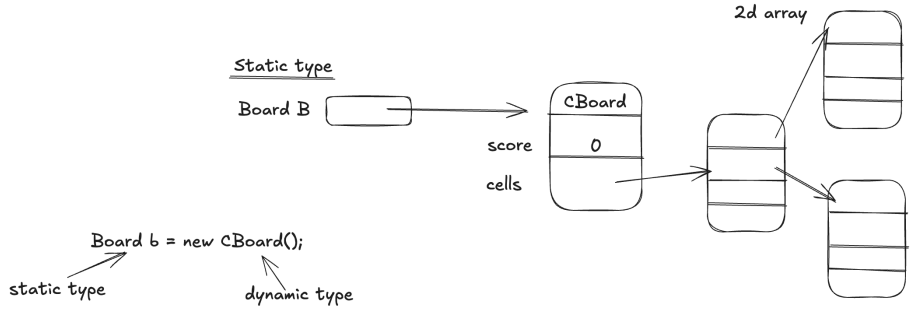
Abstraction barrier

clients can't tell which implementation is being used

example

```
void display(Board b){
    .
    .
    b.tile(r,c)
    .
    .
}
```

calls CBoard.tile or TBoard.tile depending on what b refers to



determines what methods are called

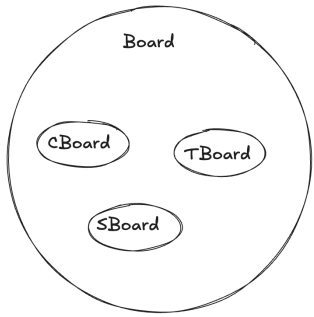
Subtyping

a CBoard can be used wherever a Board is expected

CBoard is a subtype of Board

CBoard <: Board

means subset



clients expect to use board

subtype relationships

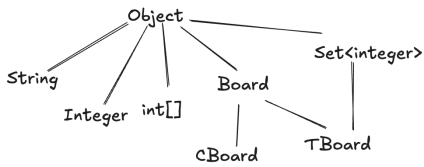
C Implements I, C <: I

I1 extends I2, I1 <: I2

C1 extends C2, C1 <: C2 ← inheritance

A class can implement >1 interface

Subtype hierarchy




```
class TBoard implements Board, Set<Integer>{
    boolean contains(Integer i){
        //loop over tiles
    }
}
```

must implement Set<Integer> method

Sound typing

Dynamic type must be a subtype of the static type

Casts

(t)e view value of e as a t

up-cast:

```
CBoard c = ...
Board b = (Board) c;
//can't fail, equiv. b = c
```

down-cast: supertype -> subtype

```
CBoard d = (CBoard) b;
```

- can fail with ClassCastException

  eg. if b is actually a TBoard

- exposes CBoard

  subtyping /= encapsulation

```
interface SolvableBoard extends Board{
    Direction[] solve();
}
```

SolvableBoard <: Board