Lecture 3: Modularity & Information Hiding

A1 due Thursday — smoke testing available soon on CMSX

Scope

Names signify:

- local variables, parameters
- instance vars
- classes
- methods
- interfaces, packages

Scope: where name has meaning

Local vars: in scope from declaration to end of block.

```
{
    ⋮
    int i;
    ⋮
}
int x=i;
```

scope

// Static error (compile time) — out of scope

```
int i = i+1;
```

i not in scope

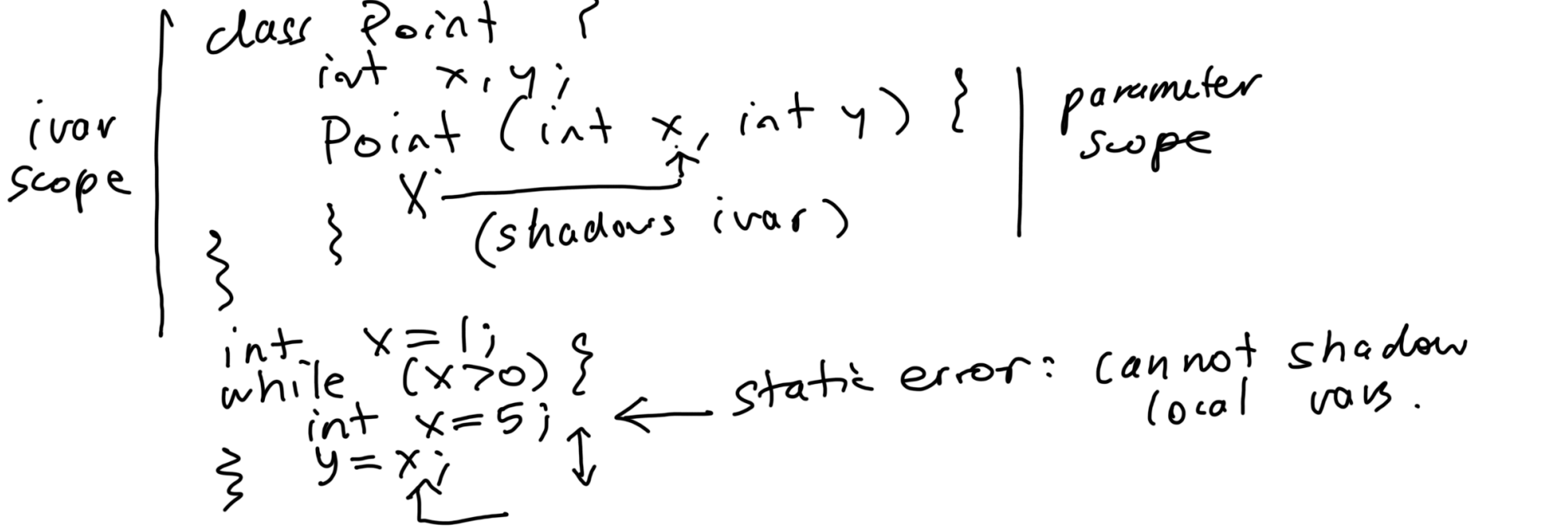
Methods ; Fields : in entire class and in selection expressions, e.g. this.x

Classes : in whole program

Rectangle r;
r.LL
selection expression

Overlapping scope?

1. Smaller scope wins – shadows outer scope
2. static error



Visibility

public: visible everywhere in containing scope (not for instance variables)

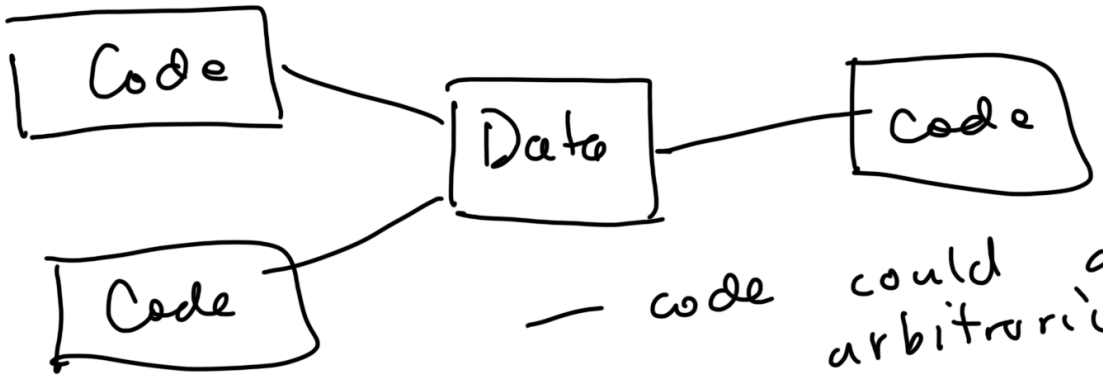
private: visible in same class (irons)

_ : " in same package

protected: " in subclasses.

Procedural Programming (e.g. C)

Build programs as modules
- smaller, ~independent code fragments
- allow ~independent work by multiple devs.



- code could change data arbitrarily
- bug in one module can break others
- changes to a module propagate to other modules

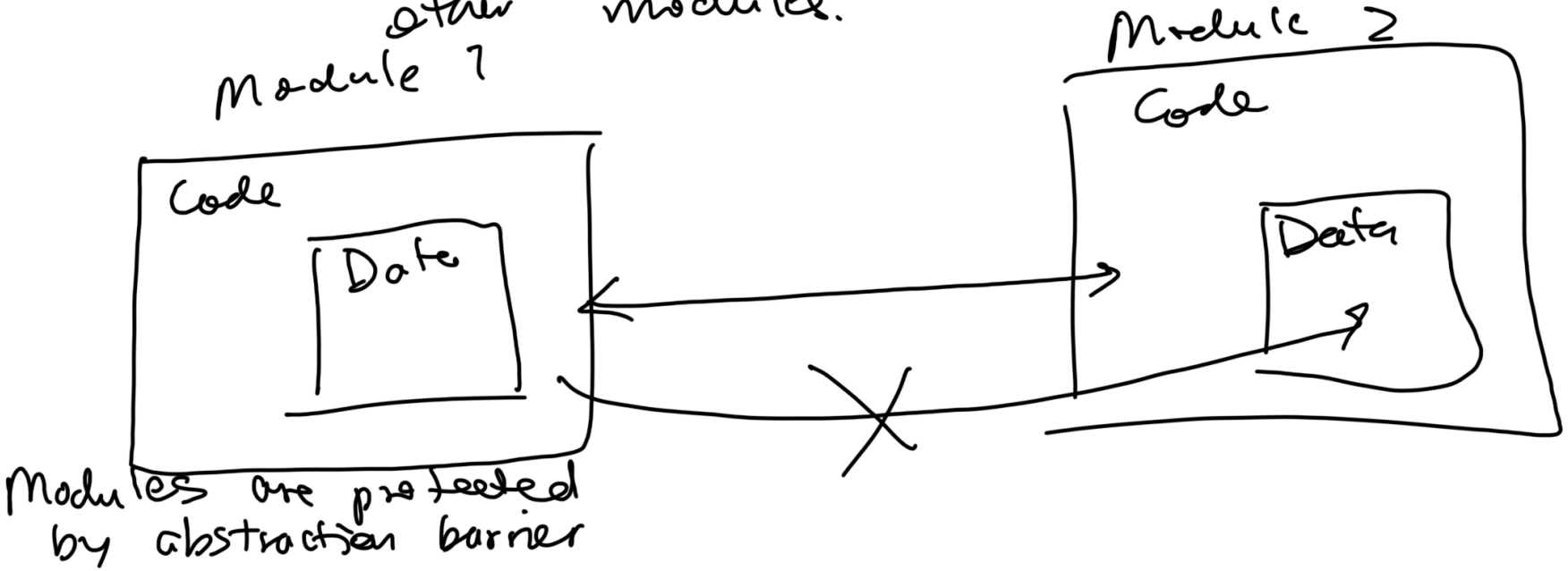
tight coupling

Information Hiding

Procedures hide (abstract) their implementation

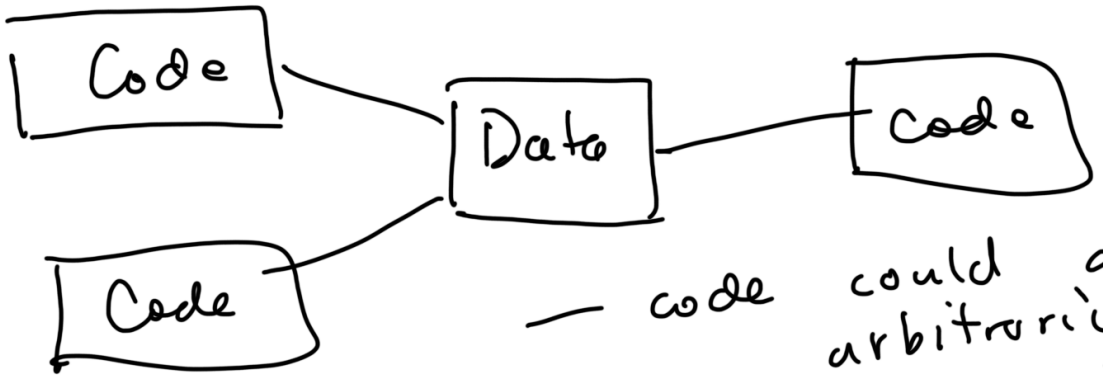
Idea: Modules can abstract contents
- hide procedures
- hide data representations (data abstraction)

Encapsulated data & procedures can't be used by other modules.



Procedural Programming (e.g. C)

Build programs as modules
- smaller, ~independent code fragments
- allow ~independent work by multiple devs.

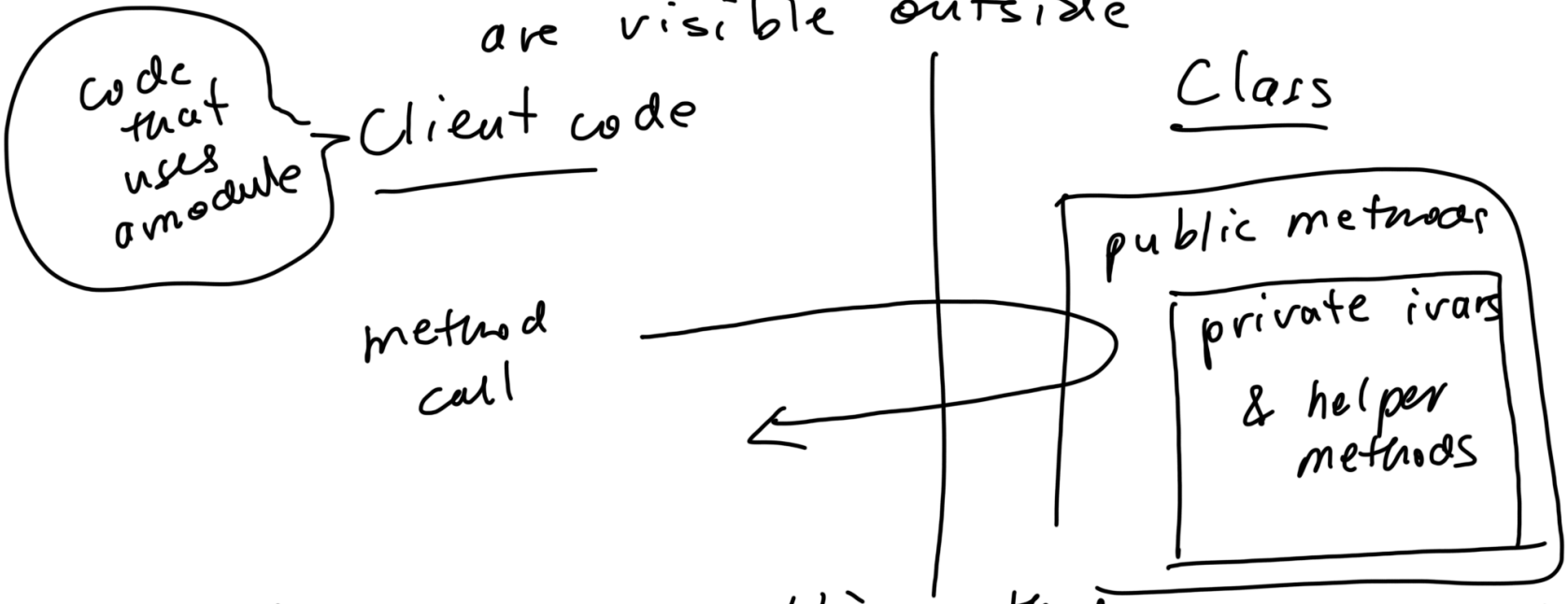


- code could change data arbitrarily
- bug in one module can break others
- changes to a module propagate to other modules

tight coupling

Java encapsulation

- Classes, packages: only "public" things are visible outside



Classes expose public methods = module interface ≠ Java "interface"

Visibility

public: visible everywhere in containing scope (not for instance variables)

private: visible in same class (irons)

_ : " in same package

protected: " in subclasses.

Preconditions & Postconditions

Requires clause in constructor: $q \neq 0$

= a precondition

Client call with $q = 0$?

- No check needed (but allowed)
- Client is wrong - must fix

Returns clause : postcondition

- must be true at return
- if not: implementer's fault.

Client specs: contract between clients & implementers

-