Procedural Abstraction (Methods)

Modular Structure

Brain-size chunks
- Not too many details. Not too many interactions.
  Something to focus on.

Units of work
- Good size for a work assignment. Good size to build and test before moving on.

Units of change
- Something that can be replaced

Classes and Methods

A class is a typical module in Java
- So far we have only written single-class programs
- Later you will write programs with tens or hundreds of classes ... but not for a while

A method is a smaller chunk, using
- A “chunk” of steps in processing
  That can be described simply
  And often reused

Chunking

Very limited working capacity

Nearly unlimited complexity
Java method signature

```java
static int min (int t1, int t2) {
    if (t1 <= t2) {
        return t1;
    } else {
        return t2;
    }
}
```

Method return type (output)

```java
static int min (int t1, int t2) {
    The output of this method is an int.
    I can write  int x = min(7, 9);
}
```

Arguments (input)

```java
static int min (int t1, int t2) {
    The inputs of this method are two integers.
    ex:  int a =1; int b=2;
        int x = min(a,b);  // note names!
```

O, be some other name! – *Juliet*

```java
static int diff (int a, int b) {
    return a – b;
}
... then in main ...
int a =5; int b = 7;
int c = diff( b, a );
   // ??? What happens
```
static int diff (int a, int b) {
    return a - b;
}

... then in main ...
int a = 5; int b = 7;

int c = diff( b, a );

static int diff (int a, int b) {
    return a - b;
}

... then in main ...
int a = 5; int b = 7;

int c = diff( b, a );

“Pass by value”

foo( x, y, 37.489, “e tu, Brute?” );

static void foo( int a, int b, double c, String d ) { ... }

What does it print?

static void foo(int x, int y) {
    x = 17;
    y = 19;
}

...
int x = 3; int m = 22;
foo(x, m);
System.out.println(“Now ” + x + “ and ” + m);
procedures and functions

static void foo( ... ) { ... }
   "void" type (procedure): Doesn’t return anything
   must do something (e.g., printing)

static int foo( ... ) { .... }
   "int" type (function): Returns a value
   should usually be a "pure" function, no side
effects

What makes a good method?

Simplifies the code that calls it
Isolates a design decision (easier to change)
Used more than once
Can be tested separately
...

A good method may have only some of these properties. Few have all.

Bad method smells

Complicated description
   If the simplest description is "blah blah and blah and blah except blah or
   blah", maybe it shouldn’t be a method

Have to keep looking back at it
   I should be able to use the method
   without remembering details of how it works

Let’s design some methods ...

Is YYYY a leap year?
Is MM/DD a valid date?
How many days between MM/DD/YYYY and (the
next occurrence of) MM/DD?

(How do they smell? You may use these in
assignment 3)