Anatomy of a Java program
(from Assignment 1, part 2)

/**
 * Calculates target heart rate using a standard formula.
 * CIS 210, University of Oregon, Assnmt 1 Winter 2012.
 * @author YOUR NAME HERE
 */
class THR {
    ... (more stuff here)
}

A Java program is divided into classes, one of which is the “main program”. Class THR is defined in file THR.java.

About names (identifiers)
The names of classes, methods, and variables
Start with a letter, followed by more letters, digits, or _
Conventions in Java:
ClassNamesAreCapitalized (camel case)
MethodsAndVariablesStartSmall
CONSTANTS_SHOUT

Two steps to run Java program

source code

```
class Foo {
    public Foo() { ... }
}
```

byte code

```
Foo.java Foo.class
```

```
javac java
```
**Translation**

Max.java

```java
int maxOfTwo(int x, int y) {
    int choice;
    choice = x;
    if (y > x) {
        choice = y;
    }
    return choice;
}
```

Max.class

```java
0: iload_1
1: istore_3
2: iload_2
3: iload_1
4: if_icmple 9
7: iload_2
8: istore_3
9: iload_3
10: ireturn
```

The `javac` compiler translates from source code to byte code or machine code.

---

**Procedural Abstraction (Methods)**

**Chunking**

Very limited working capacity

Nearly unlimited complexity

**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
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

Truth and Consequences

The type “boolean”
The “boolean” primitive type

Like int, float, etc.
but just two values: true, false

» Note to C and C++ programmers: “true” and “false” are not synonyms for 1 and 0 in Java

boolean b;
b = true;

boolean values

Typically from comparisons:
int i=7; int j = 8;
boolean b = ( i == j ); // false

Combine with “and” (&&), “or” ( || ), “not” (!)
boolean c = (i < j);
b = b && (i > j) || c;

Logic as Algebra

George Boole, 1815-1864
via Claude Shannon

Basic idea:
Treat “true” and “false” as values
Treat “and”, “or”, “not” as operations
true and x = x
true or x = true

Algebraic properties of “boolean”

“or” is like addition (commutative, associative)
(x || y) || z == x || (y || z) == x || (z || y)

“and” is like multiplication (commutative, associative)

Multiplication distributes over addition:
x && (y || z) == (x && y) || (x && z)
Using boolean values

We use boolean (true, false) values to make decisions

```java
if (cost > limit) {
    System.out.println("Sorry, out of money.");
}
```

Common Comparisons

- `==` “equal” or same as (the same value)
  (more subtle than it looks: later we will distinguish between “the same thing” and “equal value”)
- `>` “greater than” (7 > 8, “foo” > “bar”)
- `<” “less than”
- `>=, <=` for ≤, ≥
- `!=` for ≠

Boolean Operators

<table>
<thead>
<tr>
<th>Java</th>
<th>logic, math</th>
<th>electronics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;&amp;</code></td>
<td>and, ∧, ·</td>
<td>AND</td>
</tr>
<tr>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>!</code></td>
<td>not, ¬</td>
<td>NOT</td>
</tr>
</tbody>
</table>

```java
if ( ... ) { ... } else { ... }
```

```java
int limit = 1000;
int balance = 5000;
int cost = 1001;
if (cost > limit || cost > balance) {
    System.out.println("Sorry, out of money.");
} else {
    balance = balance - cost;
}
```
Nesting

```java
if (cost > limit) {
    System.out.println("Charge limit exceeded");
} else {
    if (cost > balance) {
        System.out.println("Sorry, out of money");
    } else {
        balance = balance - cost;
    }
}
```

else if ... (alternative form)

```java
if (cost > limit) {
    System.out.println("Limit exceeded");
} else if (cost > balance) {
    System.out.println("Sorry, out of money");
} else {
    balance = balance - cost;
}
```

but watch out for ...

```java
if (cost > limit)
    System.out.println("Charge limit exceeded");
else if (cost > balance)
    System.out.println("Limit exceeded");
else
    System.out.println("OK");
balance = balance - cost;
```

WRONG!

Summary: True or False

- boolean type represents logical values
- just two values, true and false
- comparisons create boolean values
- i < j, “Ben” < “Jerry”
- create expressions with &&, ||, !
- like arithmetic with numbers (almost)
- use in “if” and “if/else” statements
- control what the program does
Design and Build: Max

Given two numbers, x and y, print the larger of the two values

Given three numbers x, y, z, print the largest of the three values

Loops, part 1
or, more fun with booleans

'Round and around and around and around.

– Chubby Checker, 1960, “The Twist”

Example “for” loop

static void drawStars(int nStars) {

    for (int i=1; i <= nStars; ++i) {
        System.out.print('*');
    }
    System.out.println();
}

Pieces of the for loop (1)

    for (int i=1; i <= nStars; ++i) {
        do this some number of times
    }

Purpose: repeat the part in { ... } some number of times.
    Often a “counting” or “definite” loop.
Pieces of the for loop (2)

for (int i=1;  // create a counter
    i <= nStars;  // keep going?
    ++i)  // next counter value
{
    do this some number of times
}

More on Loops

If condition never becomes false, the loop will execute "forever"
An infinite loop will continue until some program fault is reached
The program may need to be externally terminated
Make sure when coding a loop that you provide logic to advance the
condition toward termination
Logic may be decrement, dividing by two, multiplying by 3, ...
  • Logic must guarantee that condition is eventually false

Loops are a type of statement
A loop can be selected by an if or if-else
Loop bodies may contain if, if-else, or other loop statements
So loops may be nested

Prospective: Control Flow

Power tools: We’re moving into algorithm design
  • We can make beautiful furniture, or cut off our hands

We’ll spend a little effort on syntax, and a lot of
effort on how to design and reason about
loops and conditions