ICS 111
Java Comparisons, Booleans, Problem-Solving Techniques

- Review: Java Arithmetic Comparisons
- Java String Comparisons
- Boolean Operators
- Problem Solving

Review:
Java Arithmetic Comparisons

- `a == b` is `true` if a has the same numeric value as b
- `a != b` is `true` if a does not have the same numeric value as b
- `a < b`, `a <= b`, `a > b`, `a >= b` are `true` if a is less than b, less than or equal to b, greater than b, or greater than or equal to be
- and otherwise, each of these is `false`
- reminder: = is assignment, == is comparison

Precedence of Comparisons

- These comparison operators are called **relational** operators
  - because they relate one value to another
- relational operators have lower precedence than arithmetic operators:

`(a + 1 > b)` means `((a + 1) > b)`

String Equality Comparison

- You can compare strings with ==
- However, `a == b` is true only if the strings a and b are at the same location in memory
  - useful sometimes
  - but not at this stage in your career
- Instead, compare with `String.equals`:

```
String hello = “Hello world”;
if (hello.equals(“Hello world”)) { ...
```

- self-test exercise: which of these are true?

```
hello.equals(“hello”)
hello.substring(6).equals(“world”)
hello.substring(6,7).equals(“w”)
```

String Equality Self-Test: Results

- self-test exercise: which of these are true?

```
hello.equals(“Hello”)
hello.substring(6).equals(“world”)
hello.substring(6,7).equals(“w”)
```

- The second and the third are true. The first is not, because “Hello world” is not equal to “Hello”

Alphabetic Comparisons

- We all know how to alphabetize strings
- Intuitively, “a” < “b”
- But which of “A” and “a” is less?
- There is a standard called the American Standard Code for Information Interchange, or ASCII (pr. ask-key)
- digits < uppercase < lowercase
  - the whole set at wikipedia
- The international equivalent is Unicode
  - including UTF-8, UTF-16
- alphabetic comparisons only make sense between characters in the same language

String Alphabetic Comparisons in Java

- Just as we usually use String.equals instead of ==, we use `String.compareTo(s)` instead of < to compare strings

```
  if (String.compareTo(s) < 0) ...
```

- String.compareTo(s) returns an integer
  - an integer < 0 if String comes before s
  - an integer > 0 if String comes after s
  - 0 if String.equals(s)
- So we can use any arithmetic relational operator, with String.compareTo(s) on the left, and zero on the right to compare two strings
  - instead of saying `if (a >= b)`
  - we say `if (a.compareTo(b) >= 0)`

String Comparison Examples

`String hello = “hello, world”;`

- `hello.compareTo(“world”) < 0` is true
  - “hello, world” is alphabetically before “world”
- `hello.compareTo(“hello”) < 0` is false
  - because “hello” is a shorter string than hello, so “hello” comes first

`String abc = “abc”;`

- `abc.compareTo(hello) < 0` is true
- `hello.compareTo(abc) < 0` is false

Boolean Operator Review

- We have already seen the basic boolean operators && (and), || (or), ! (not)
- the result of `a && b` is only true if a is true and b is true
- the result of `a || b` is true if a is true, b is true, or both are true
- the result of `!a` is true if a is false

Boolean Operators: A different Perspective

- In Java, true and false are separate values, they are not integers
- But some programming languages do not have a separate boolean type
- Instead, they use 0 for false, and 1 or any other integer for true
- Then, && is the same as multiplication:
  - true * true = true, but true * 0, 0 * true, 0 * 0 = 0
- || is the similar to addition
  - false + false = false, but true + 0, true + true, 0 + true = true

Boolean Operator Precedence

- The precedence of the boolean operators is less than that of relational operators
  - `a + 1 > 3 && a < 4` means
    ((a + 1) > 3) && (a < 4)
- && has greater precedence than ||
  - so a && b || c means
    (a && b) || c
- This precedence is modeled on the analogy of && to multiplication, and || to addition
- ! has high precedence, like the negation operator -

Boolean Operators: Short-Circuit Evaluation

```
int x = 3
if (x > 0 || x++ > 1) {
}
```

- What is the value of x after this execution?
- Java evaluates expressions left-to-right
- If the left operand of an || is true, Java knows it does not have to evaluate the right operand
- Similarly if the left operand of an && is false
- So Java does not evaluate x++ > 1, and x remains at three

Short-Circuit Evaluation Practical Examples

- Division by zero in Java causes an error
  - really, an exception, but for now they look like errors
- We can test for the quotient being non-zero, then divide by that quotient in the same expression, without fear of triggering the exception

```
if (q != 0 && (2222 / q) == z) {
```

- When we talk about arrays (around Oct 7th), we may want to test for a valid array index, then use that index to access an array element
  - We can do all this in a single boolean expression!

```
if (index < a.length && a [index] > 0) ...
```

  - again, this code will make more sense once we learn arrays

More Java

- Dangling `else`
- Enumeration types

Dangling `else`

- Suppose you have a nested if,
- and it is so simple you don't want to use braces

```
if (a)
  if (b)
    System.out.println("a and b");
  else
    System.out.println("not sure!");
```

- Which of the two if statements does the else belong to? Instead of "not sure", we could print:
  - "a and not b", if the else belongs to the second if or
  - "not a", if the else belongs to the first if
- In Java, else matches the nearest if, so "a and not b" is correct
- However, this is confusing!

Dangling `else` Solutions

- Always use the curly braces for if
  - remember that we are trying to write clear code
  - this is part of the program structure
- If you are reading someone else's code, remember that an else goes with the innermost matching if

Enumeration Types in Java Motivation

- When a variable can only have a few different values, we can represent it using an int or a string
- For example, if using a variable to represent animal, vegetable, or mineral,

```
int category = 1; // 1 animal, 2 vegetable, 3 mineral
String category = “animal”;
```

- If we do this, the Java compiler doesn't notice when we assign a “wrong” value such as 5 or “food”
- To have the compiler check our work, we can create a special Java data type that only has the values we plan to use
  - For example, we may call it Group

```
Group category = ANIMAL;
```

Enumeration Types in Java

- Enumeration types only work with a finite number of values
- All of these must be listed in the declaration of the type

```
public enum Group { ANIMAL, VEGETABLE, MINERAL }
```

(listing all possible values is called **enumerating**)

- The values are constant, so we write them in all upper-case
- Then we can compare with == or switch

```
switch (category) {
case ANIMAL: System.out.println (“meeow”); break;
case MINERAL: System.out.println (“thud”); break;
default: break;
}
```

Problem Solving in Java

- Tracing Programs
- Test Cases
- Logging
- Flowcharts

Tracing Programs

- Humans can do anything a computer can do
  - only more slowly, and not as accurately
- When we are confused by a program (or part of a program), we can execute it by hand
- We can record the values of variables on paper or on a computer
  - don't delete the old values -- just write the new values near the old ones, so it is clear which value is current
  - should also record what the program prints
- When we see an “if”, we must evaluate the condition, and only execute the relevant part
- This is a very useful learning tool too!
- See Programming Tip 3.5 in the textbook for a detailed example

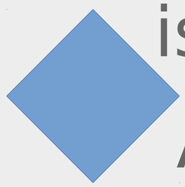
Test Cases

- A real program has inputs that affect the computation
- Conditionals may do different things depending on the value of these inputs
- Ideally, we test the program:
  - in such a way that every branch of every conditional is executed at least once
  - for all values that are near the boundary of a condition
  - e.g. if the condition is x > 3, test for values of 2, 3, and 4
  - also test for conditions that the programmer might have forgotten about, especially 0 and -1
- When we test, we have to verify that the result is correct, so we have to know what results to expect from the program
- You can develop test cases before writing code!

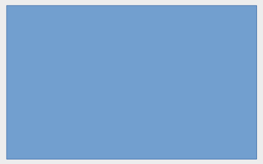
Logging

- Print statements are useful to try to see what your program is doing
- But when running the program in daily life, it is better not to have such print statements
- java.util.logging.Logger lets you turn such print statements on and off just once for your entire program
  - rather than having to find and fix each print statement
- Full documentation

Flowcharts

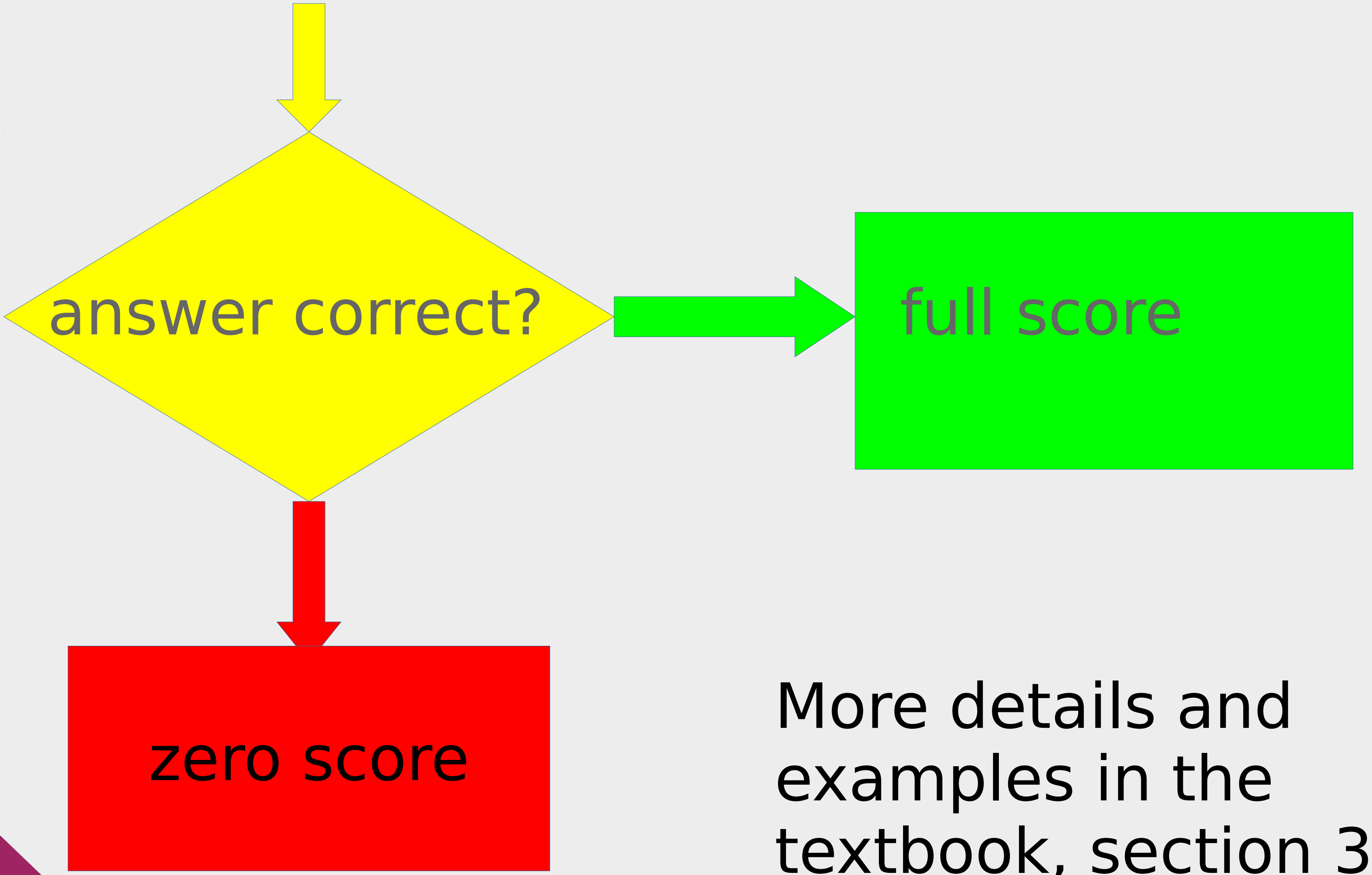
- Flowcharts are a graphical way of showing the structure of a program
-  is used to show a decision point.

  A true arrow leads out of one corner,

  a false arrow out of another corner
-  is used to show any other task

  arrows come in from above, leave below
- Flowcharts are good for people who think visually!

Flowchart Example



More details and examples in the textbook, section 3.5

Summary

- Comparing:
  - Numbers: <, <=, ==, !=, >, >=
  - Strings:
    - String.equals()
    - String.compareTo() combined with <, <=, ==, !=, >, >=
- Boolean expressions, short-circuit evaluation
- Dangling else, enumeration types
- Problem Solving: Tracing, test cases, logging, flowcharts