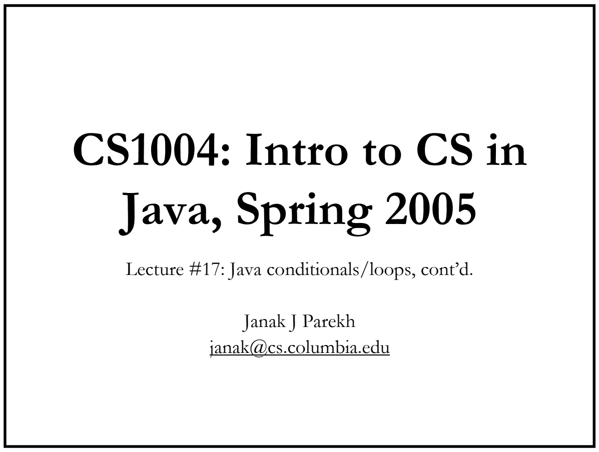
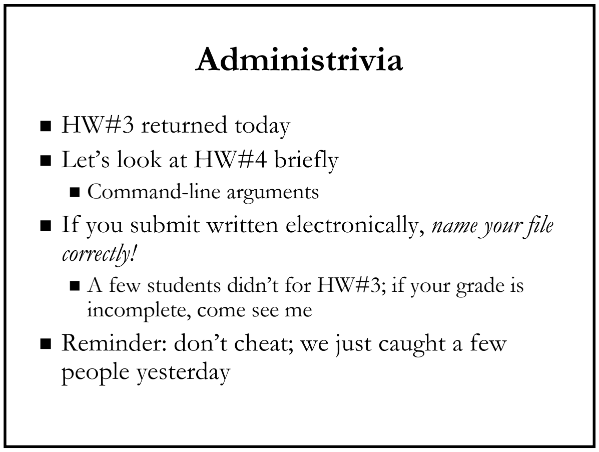


CS1004: Intro to CS in Java, Spring 2005

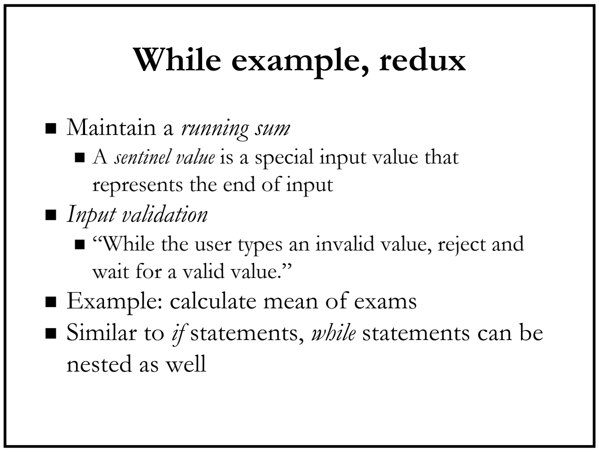
Lecture #17: Java conditionals/loops, cont'd.

Janak J Parekh
janak@cs.columbia.edu



Administrivia

- HW#3 returned today
- Let's look at HW#4 briefly
  - Command-line arguments
- If you submit written electronically, *name your file correctly!*
  - A few students didn't for HW#3; if your grade is incomplete, come see me
- Reminder: don't cheat; we just caught a few people yesterday



While example, redux

- Maintain a *running sum*
  - A *sentinel value* is a special input value that represents the end of input
- *Input validation*
  - "While the user types an invalid value, reject and wait for a valid value."
- Example: calculate mean of exams
- Similar to *if* statements, *while* statements can be nested as well

Infinite Loops

- The body of a `while` loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while(count <= 25) {
   System.out.println(count);
   count = count - 1;
}
```

- This loop will continue executing until interrupted or until an underflow error occurs

Nested Loops

- How many times will the string `"Here"` be printed?

```
count1 = 1;
while(count1 <= 10) {
   count2 = 1;
   while(count2 <= 20) {
      System.out.println("Here");
      count2++;
   }
   count1++;
}
```

break, version 2

- We saw `break` in the context of `switch`, but it can be used with `while` (and other loops) as well; for example,

```
while(true) {
      if(i > 10) break;
      else i++;
}
```

- What is this code equivalent to?
- Generally, you don't use break, but it's useful to have, especially if the while loop is very complex
- If you have nested loops, break only breaks out of the most immediate loop, not all of them
  - *return* can be used to break out of a bunch of loops, but avoid

The do Statement

- A *do statement* has the following syntax:

```
do {
   statement;
} while(condition);
```

- The `statement` is executed once initially, and then the `condition` is evaluated
- The statement is executed repeatedly until the condition becomes false

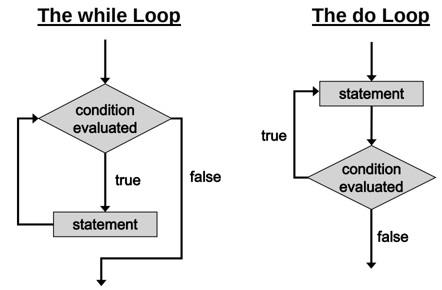
The do Statement

- An example of a `do` loop:

```
int count = 0;
do {
   count++;
   System.out.println(count);
} while (count < 5);
```

- The body of a `do` loop executes at least once
- What's the result of this code fragment?
- *do* is particularly useful for "interactive repetition"

Comparing while and do


The while Loop

The do Loop

The for Statement

- A *for statement* has the following syntax:

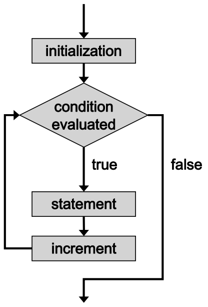
The *initialization* is executed once before the loop begins

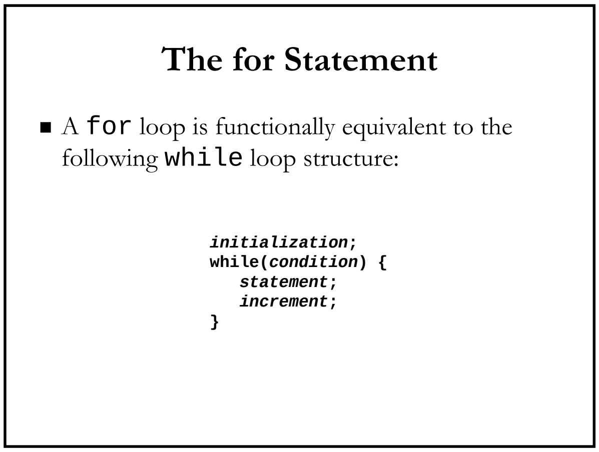
The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

Logic of a for loop

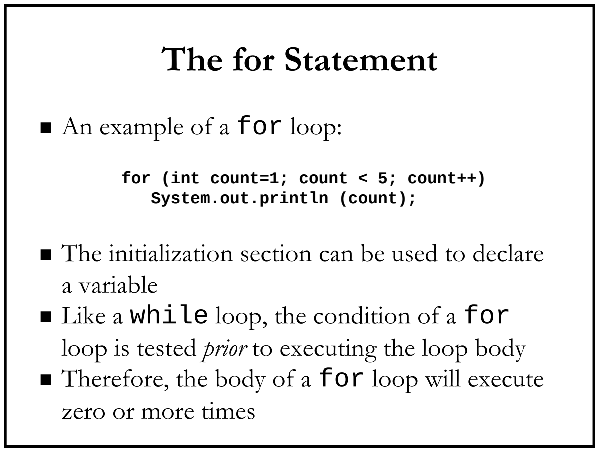




The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while(condition) {
   statement;
   increment;
}
```

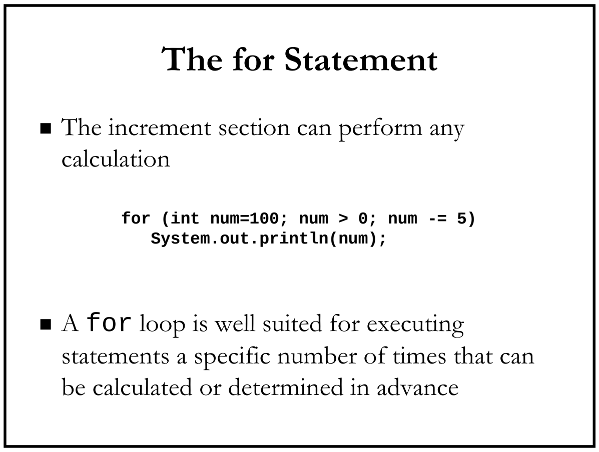


The for Statement

- An example of a `for` loop:

```
for (int count=1; count < 5; count++)
   System.out.println (count);
```

- The initialization section can be used to declare a variable
- Like a `while` loop, the condition of a `for` loop is tested *prior* to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times



The for Statement

- The increment section can perform any calculation

```
for (int num=100; num > 0; num -= 5)
   System.out.println(num);
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance

The for Statement

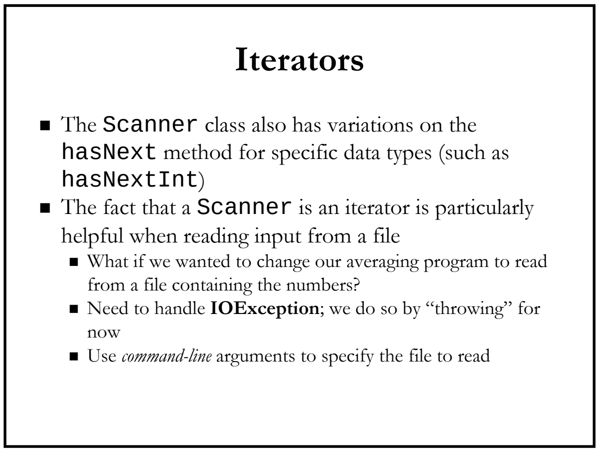
- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

In-class extra credit

- Here's how it works:
  - I'll outline the problem on the board in class
  - At the *beginning* of *next class*, hand in a printout containing:
    - Your name
    - The code
    - Execution of the code
    - A few sentences explaining what you found out
- *No* electronic submission for this
- This will *not* affect the grade of those that don't do it
- Goal is for people to get opportunities to practice concepts more frequently than homeworks

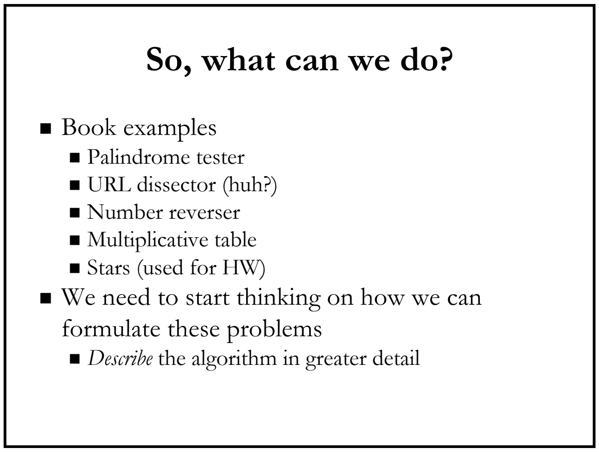
Iterators

- An *iterator* is an object that allows you to process a collection of items one at a time
- Step through each item in turn and process it as needed
  - The `hasNext` method that returns true if there is at least one more item to process
  - The `next` method returns the next item
- Several classes in Java, including `Scanner`, are iterators
  - The `hasNext` method returns true if there is more data to be scanned
  - The `next` method returns the next scanned token as a string



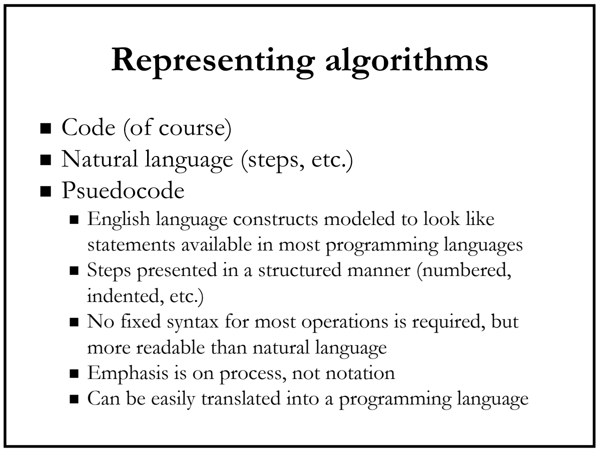
Iterators

- The `Scanner` class also has variations on the `hasNext` method for specific data types (such as `hasNextInt`)
- The fact that a `Scanner` is an iterator is particularly helpful when reading input from a file
  - What if we wanted to change our averaging program to read from a file containing the numbers?
  - Need to handle **IOException**; we do so by “throwing” for now
  - Use *command-line* arguments to specify the file to read



So, what can we do?

- Book examples
  - Palindrome tester
  - URL dissector (huh?)
  - Number reverser
  - Multiplicative table
  - Stars (used for HW)
- We need to start thinking on how we can formulate these problems
  - *Describe* the algorithm in greater detail



Representing algorithms

- Code (of course)
- Natural language (steps, etc.)
- Psuedocode
  - English language constructs modeled to look like statements available in most programming languages
  - Steps presented in a structured manner (numbered, indented, etc.)
  - No fixed syntax for most operations is required, but more readable than natural language
  - Emphasis is on process, not notation
  - Can be easily translated into a programming language

How do we come up with algorithms?

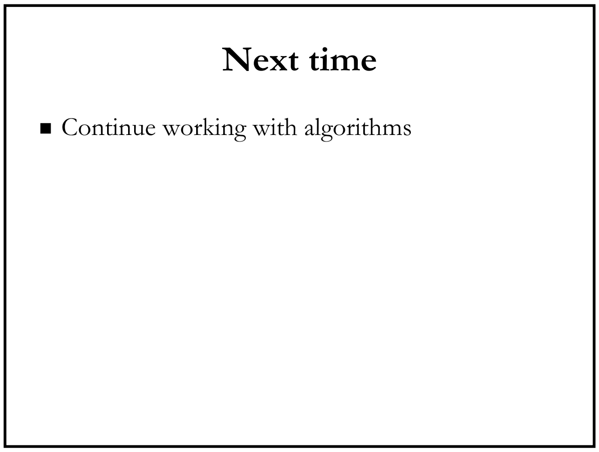
- An imprecise science at best: problem-solving
  - Understand the problem
  - Get an idea of how/which algorithm might solve the problem
  - Formulate the algorithm and represent as a program
  - Evaluate the program for accuracy and potential to solve other problems
- This is not much help, is it?

"Get a foot in the door"

- Try doing the first (few) step(s) by hand
  - Look at what you had to do to accomplish it
  - See if you can reapply this to continue solving the problem
- Reapply another solution
- Stepwise refinement
  - Look at the problem from a very high level
  - Break it down repeatedly into smaller pieces, until we get a set of algorithmic steps

Board examples

1. Palindrome checker (see book for code)
2. Print out the first *n* Fibonacci numbers
3. Search for a number in a list
4. Reverse a list (array) of numbers



Next time

- Continue working with algorithms