Console Text I/O

15 Jan 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Outline for today

- Labeled blocks
- Console output:
  - System.out, print(), println()
  - Formatted output: printf()
  - Formatter objects: NumberFormat, DecimalFormat
- Console input:
  - Scanner class
  - .nextLine() and newlines

Labeled blocks

- Blocks can be named
- break/continue can specify a name:
  - Go to start/end of named block

```
main: {
        for (row=0; row<n_rows; row++) {
            for (col=0; col<n_cols; col++) {
                if (row+col == 12) break main;
            }
        }
}
```

Console output: System.out

- System.out is the standard output channel
  - Default is console
  - But can be redirected to a file
- Methods of System.out
  (and other output channels):
  - .print(*str*): output *str* exactly as-is
  - .println(*str*): output *str* plus a newline
  - .printf(*fmt, arg1, arg2, ...*):
    use a format string (with %d, %s, etc.)
- Or use formatter objects

Formatted output: printf

- .printf() uses a format string just like Python:
  - **System.out.printf("I have %3d apples\n", numApples);**
- Format specifiers (with optional field width):
  - %d: integer (%3d, %03d, %-3d, %-03d)
  - %f: float (%5f, %5.1f, %05.1f, %-05.1f)
  - %e: scientific-notation float, e.g., 1.23e4
  - %s: string (%12s)
  - %c: character (%-2c)

Formatter objects

- If using the same format string many times, try creating a formatter object:
  - Has a .format() method:
  - Parameters are the values to format
  - Returns a formatted string good for .print()
- There are several kinds of formatter classes in the java.text library:
  - **import java.text.NumberFormat;**
  - **NumberFormat moneyFmt = NumberFormat.getCurrencyInstance();**
  - moneyFmt is our new formatter object

Using formatter objects

- Now we can send all kinds of numbers to be formatted using our moneyFmt formatter:
  - **moneyFmt.format(105.248) // "$105.25"**
  - **moneyFmt.format(-12.3) // "-$12.30"**
  - **moneyFmt.format(17) // "$17.00"**
    - The .format() method returns a string
    - To output on the console, use:
  - **System.out.println(moneyFmt.format(…));**
- NumberFormat.getCurrencyInstance() returns the currency formatter for the current locale
    - Different for UK, Japan, etc.!

DecimalFormat

- NumberFormat has other kinds of formatters, and you can make your own, too.
- DecimalFormat is a subclass for creating your own formatter object on floats:
  - **Import java.text.DecimalFormat;**
  - **DecimalFormat myFmt =**
    new DecimalFormat(“000.0000”);
    - Declares a new DecimalFormat object and instantiates it to format a certain way:
    - “000.0000”: ≥3 digits before decimal point and exactly 4 digits after

Using DecimalFormat

- **System.out.print(myFmt.format(9.14038));**
  // outputs “009.1404”
- **System.out.print(myFmt.format(13849));**
  // outputs “13849.0000”

- Specify optional digits with '#':
  - **new DecimalFormat(“##00.0##”)**
    - Between 2-4 digits before decimal and between 1-3 digits after decimal
- Convert to percentage with '%' at end:
  - **new DecimalFormat(“00%”)**
    - 2-digit percentage: 0.7361 → “74%”

Console Input: Scanner

- System.in is the standard input channel
  - Yields raw text, like Python's raw_input()
- Parse the input using a Scanner object:
  - **import java.util.Scanner;**
  - **Scanner kbd = new Scanner(System.in);**
- Now we can read integers, floats, or words:
  - **kbd.nextInt()** **// returns an int**
  - **kbd.nextDouble()** **// returns a double**
  - **kbd.next()** **// returns next word (string)**

Dealing with newlines

- The Scanner's .nextLine() method reads from the current file postion to the next newline
  - Returns a string
- Remember to swallow newlines at end of input!
- Say our code does .nextInt(), then .nextLine()
- If the user's keyboard input is “12 apples”,
  - Then the .nextInt() gets 12,
    and .nextLine() gets “apples\n”
- If the user inputs just “12”, then
  - The .nextLine() gets just the newline!