CS1007 lecture #5 notes

tue 17 sep 2002

- news
- boolean expressions
- logical operators
- truth tables
- relational operators
- the `if` branching statement
- flowcharts
- command line arguments
- `System.exit()` method
- reading: *ch 3.1-3.4*

news.

- homework #1 is due today
- homework #2 will be posted Thursday
- Note that we skipped section 2.6 — user input. This section describes the `tio` package, which we are not going to use for a few more weeks.

boolean expressions.

- boolean variables: `true` (1) or `false` (0)
- logical operators:

!	not
&&	and
||	or

example:

```
boolean a, b;
x = 1; // true
y = 0; // false
System.out.println( "x && y is false" );
System.out.println( "x || y is true" );
System.out.println( "x && !y is true" );
```

truth tables.

a	!a
false	true
true	false

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

relational operators.

==	equality
!=	inequality
>	greater than
<	less than
>=	greater than or equal to
<=	Less than or equal to

example:

```
int x, y;
x = -5;
y = 7;
```

some truths:

(x < y)	true
(x == y)	false
(x >= y)	false

the `if` branching statement.

```
if ( x < y ) {
  x = y;
}
```

```
if ( x < y ) {
  x = y;
}
else {
  x = 91;
}
```

the `if` branching statement (1).

there are four forms:

(1) simple if

```
if ( x < 0 ) {
  System.out.println( "x is negative\n" );
} // end if x < 0
```

(2) if/else

```
if ( x < 0 ) {
  System.out.println( "x is negative\n" );
} // end if x < 0
else {
  System.out.println( "x is not negative\n" );
} // end else x >= 0
```

the `if` branching statement (2).

(3) if/else if

```
if ( x < 0 ) {
  System.out.println( "x is negative\n" );
} // end if x < 0
else if ( x > 0 ) {
  System.out.println( "x is positive\n" );
} // end if x > 0
else {
  System.out.println( "x is zero\n" );
} // end else x == 0
```

the `if` branching statement (3).

(4) nested if

you can nest any kind/number of if's

```
if ( x < 0 ) {
  System.out.println( "x is negative\n" );
} // end if x < 0
else {
  if ( x > 0 ) {
    System.out.println( "x is positive\n" );
  } // end if x > 0
  else {
    System.out.println( "x is zero\n" );
  } // end else x == 0
} // end else x >= 0
```

flowcharts

- diagram for illustrating control flow of a program
- conventions:
  - rectangle = statement or method call
  - diamond = yes/no or true/false question

command line arguments (1).

- remember our model of a computer program from the 2nd lecture:
  $input \rightarrow \boxed{\text{CPU}} \rightarrow output$
- homework #1 was an *output only* program
- now we will learn how to get *input* into your program
- there are many ways to do this...
- we will start with *command line arguments*, which are a way of getting input into your program from the UNIX environment when you start up your program
- UNIX commands use *arguments* (arguments are also called *parameters*)
- for example, with the command:

  ```
  unix$ ls -l
  ```

  the `ls` part is the *command*; and
  the `-l` part is an *argument* (in this case, `-l` is a special type of argument, also called a "switch" in UNIX; it is an argument that starts with a `-`, and usually is used to indicate switching on or off some feature of the command being run)

command line arguments (2).

- the “hello world” program takes no arguments and is started up like this:

```
unix$ java hello
```

- here’s the source code:

```
public class hello {
    public static void main ( String[] args ) {
        System.out.println( "hello world!\n" );
    } // end of main()
} // end of class hello()
```

command line arguments (3).

- the “hello2” program that takes one argument and is started up like this:

```
unix$ java hello2 ringo
```

- here’s the source code:

```
public class hello2 {
    public static void main ( String[] args ) {
        System.out.println( "hello "+args[0] );
    } // end of main()
} // end of class hello2()
```

- in this example, the argument is `ringo`
- and the output of the program would be:

```
unix$ java hello2 ringo
hello ringo!

unix$
```

command line arguments (4).

- the argument `args` to the `main()` method is of type `String[]`
- which means it is a list of strings (i.e., Java class `String`)
- where a string is a list of characters (i.e., Java primitive data type `char`)
- `String` is something called a *wrapper class*
- we'll talk more about wrapper classes later
- a `String` value is defined using double quotes, e.g.,
  ```
  String x="ABC";
  ```
  or
  ```
  String y="A";
  ```
- a `char` value is defined using single quotes, e.g.,
  ```
  char z='A';
  ```

command line arguments (5).

- when a java program is invoked from the UNIX command line, any values after the program name are *passed into the program*, for use when the program is running.
- the `args` argument to `main` gives you access to these values, for free (i.e., you don't have to do anything special to get them), through the line of code that looks like this:
  `public static void main( String[] args )`
- you can see how many arguments were passed into the program by using the value of `args.length`
- you can see what the values of the arguments are by looking them up in the `args` list, using an *index*, i.e., a number which indicates which entry in the list you are referring to
- remember that in computer science, we start counting with 0
- so the first value in the argument list is referenced as `args[0]`, and so on

command line arguments (6).

- given the command line:
  `unix$ java ex60 A 12 DOG`
  then
  `args.length` would be equal to 3, and
  `args` would look like this:
  ```
  arg[0] "A"
  arg[1] "12"
  arg[2] "DOG"
  ```
- these are all `String`s!!
- if you want to use a command line argument as a number, then you have to convert it, just like we converted, or *coerced*, `int` to `char` and so forth
- for today, only worry about the syntax:

to go from	to	use the following
`String s`	`float f`	`f = (Float.valueOf(s)).floatValue();`
`String s`	`int i`	`i = (Integer.valueOf(s)).intValue();`

`System.exit()` (1)

- a *method* in class `java.lang.System`
- definition:

```
public static void exit( int status );
```

- terminates the currently running Java Virtual Machine
- the argument serves as a status code — by convention, a nonzero status code indicates abnormal termination
- use at the end of a program to exit cleanly or to terminate in the middle

System.exit() (2)

```
import java.lang.*;

public class ex_exit {

  public static void main ( String[] args ) {
    if ( args.length < 3 ) {
      System.out.println( "usage: java ex_exit <a> <b> <c>" );
      System.exit( 1 ); // abnormal termination
    }
    // ... rest of program goes here ...
    System.exit( 0 ); // normal termination
  } // end of main()

} // end of class ex_exit
```