Programming and Software Development
COMP90041

Lecture 2

Input / Output

Dr. Thomas Christy
thomas.christy@unimelb.edu.au
Level: 08 Room: 11, Doug McDonell, Parkville.

Slides and materials were developed by Prof Peter Schachte

Review

- Hello World!
- Java
- Javac
- Class
- Servers

Operations for Number Types

- Each type has certain operations that apply to it
- For primitive number types: `+` `-` `*` `/` `%`
  - Type of result is same as type of operands
- Use operations to construct expressions, which have values that can be assigned or used as operands
  - E.g., `answer = (2 + 4) * 7;` `count = count + 1;`
- Comparison operations also work for number type:
  `<` `<=` `>` `>=` `==` `!=`
- NB: `==` is a comparison while `=` is an assignment.
- Comparisons return booean values
  - E.g., **`done = count >= answer;`**

Operations for booleans

- `&&` (AND) is true if both operands are true
  - E.g., `test1 && test2`
- `||` (OR) is true if either operand is true
  - E.g., `test1 || test2`
- Both are short circuit operations: they only evaluate the second operand if necessary
  - E.g., `dx 1= 0 && dy / dx > 0`
  - E.g., `dx == 0 || dy / dx <= 0`
- `&` and `|` are non-short-circuit versions: they always evaluate both operands (rarely needed)
- `!` (NOT) is true if its operant is false
  - E.g., `!test1`

Strings

- `String` is a class type, so strings are objects
- Specify a string constant by enclosing in double-quote (") characters
  - E.g., `String s = "example string";`
- Include double-quote in a string by preceding with backslash (\ )
- Include a backslash in a string by preceding with backslash
  - E.g., `"He said \"backslash (\\) is special!\""`
- Certain letters after backslash are treated specially
  - Most important: `\n` = newline (end current line), `\r` – return (go to start of line), `\b` = backspace, `\t` = tab character
- These work for character constants, like `'\n'`

String Operations

- You can use `+` to append two strings
  - E.g., `System.out.println("Hello " + "World");`
  - Prints "`Hello World`"

- `+` is clever: if either operand is a string, it will turn the other into a string
  - E.g., `System.out.println("a = " + a + ", b = " + b);`
  - If `a` = 1 and `b` = 2, this prints "`a = 1, b = 2`"

- But beware:
  `System.out.println("1 + 1 = " + 1 + 1);`
  actually prints "`1 + 1 = 11`"

More String Operations

- String class has many more operations, e.g.:

- Assume `String s, s2, int i, j;`
  - `s.length()` returns the length of the string
  - `s.toUpperCase()` returns ALL UPPER CASE version of string
  - `s.substring(i, j)` returns the substring of `s` from character `i` through `j`-1, counting the first char as 0
  - E.g., `"smiles".substring(1,5)`, returns `"mile"`
  - `s.equals(s2)` returns true if a `s` and `s2` are identical
  - `s.indexOf(s2)` returns the first position of `s2` in `s`

- Don't use ==, <, >= etc. to compare strings

- See String class in documentation for more
  - Java API 8
  - https://docs.oracle.com/javase/8/docs/api/

Special Assignment Operations

- It's common to perform an operation on a variable and store the results back in the same variable
  - E.g., `x = x +1; x = x * 2; x = x -2; ...`

- Java has a special form of assignment combined with each of these operations

- Just write the operation symbol followed by =
  - E.g., `x += 1; x *= 2; x -= 2; x/= 10; x %= y;`

- Can also use **+=** to append to a string carriable:

```
String msg = 'Hello, ";
msg += "World!";
System.out.println(msg);
```

- Prints "**Hello, World!**"

Pre/Post Increment/Decrement

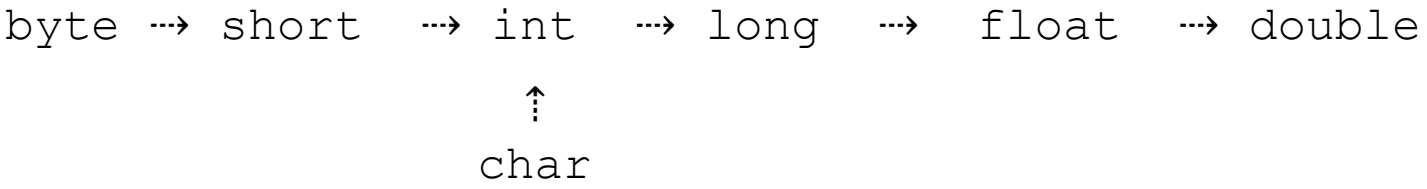
- ++x is a special expression that increments x (for any variable x) and returns the incremented value
  - E.g., if x is 7, ++x is 8, and after that, x is 8
  - Called "pre-increment" because it increments variables before returning it
- - -x (pre-decrement) is similar: it decrements x and returns it
- x++ (post-increment) returns x and then increments it
  - E.g., if x is 7, x++ is 7, and after that, x is 8
- x- - (post-decrement) returns x and then decrements it

Pre/Post Increment/Decrement Use

- Pre/post increment/decrement can be confusing (like the last example!)
- They can also be used as statnements rather than expressions
  - E.g., **++x;** or **x++;**
  - Used as statements, these both just increment **x**
- This is the recommended way to use them

Type Conversions

- Primitive operations work on operands of the same type
- But Java can convert types for you automatically
- A widening conversion converts a number to a wider type (so the value can always be converted successfully)
- Automatic conversions in Java:



- bob

Casting

- Narrowing conversions are also possible
- But they must be performed explicitly using a cast
- Cast is specified by writing the name of the type to convert to in parentheses before the value to be converted
- Cast can be used to explicitly ask for a widening conversion

```
int sum;
int count;
// compute sum and count...
double average = (double) sum / count;
```

Precedence and Associativity

- Precedence of 2 operators, say $\odot$ and $\oplus$ determines whether $a \odot b \oplus c$ is read as:
  - $(a \odot b) \oplus c$ ($\odot$ has higher precedence), or
  - $a \odot (b \oplus c)$ ($\odot$ has lower precedence)
  - *E.g.*, `2+3*4` (`*` has higher precedence)
- Associativity determines whether $a \odot b \odot c$ is read as:
  - $(a \odot b) \odot c$ (left associativity), or
  - $a \odot (b \odot c)$ (right associativity)
  - *E.g.*, `3-2-1` (`-` associates left)
- Java's rules are mostly as you would expect
- When it doubt, just put in parentheses

Operators, High to Low Precedence

Symbol	Associativity
`.` (method invocation)	
`++ --`	
`-` (unary negation)	
`(type)` casts	
`* / %`	Left
`+ -`	Left
`< > <= >=`	Left
`== !=`	Left
`&&`	Left
`||`	Left
`= += *= ...`	Right

Formatted Output

- `printf` is like `print`, but it lets you control how data is formatted
- Form:
  `System.out.printf(format-string, args...);`
- *E.g.*:
  `System.out.printf("Average:  %5.2f", average);`
- `format-string` is an ordinary string, but can contain format specifiers, one for each of the `args`
  - Format specifier begins with `%`,
  - may have a number specifying how to format the next value in the `args...` list
  - ends with a letter specifying the type of the value
- Ordinary text in `format-string` is printed as is

Format Specifiers

- The (optional) number following % is interpreted:
  - The whole number part (before decimal point) specifies the minimum number of characters to be printed
  - The full number will be printed, even if takes more characters
  - If omitted, the value will be printed in its minimum width
  - If the number if negative, the value will be left-justified, otherwise right-justified
  - The part of the number after a decimal point specifies the number of digits of the value to print after the decimal point
  - If no decimal point, Java decides how to format

Format Letters

The final letter in a format specifier can be:

`d`	format an integer (no fractional part)
`s`	format a string (no fractional part)
`c`	format a character (no fractional part)
`f`	format a float or double
`e`	format a float or double in exponential notation
`g`	like either `%f` or `%e`, Java chooses
`%`	output a percent sign (no argument)
`n`	end the line (no argument)

- Good format for money: `$%.2f`

Formatted Output Example

```
public class PrintExample
{
    public static void main(String [] args)
    {
        String s = "string";
        double pi = 3.1415926535;
        System.out.printf("\"%s\" has %d characters%n", s, s.length());
        System.out.printf("pi to 4 places: %.4%n", pi);
        System.out.printf("Right>>%9.4f<<", pi);
        System.out.printf(" Left >>%-9.4f<<%n", pi);
    }
}
```

Generated Output

```
"string" has 6 characters
Pi to 4 places: 3.1416
Right>>   3.1416<< Left>>3.1416   <<
```

Handling Command Line Inputs

- When your program is run, it can be given arguments on the command line
- Allows the user to give information to the program
- For the boilerplate we've been using, the command line arguments can be referred to as:
  - first command line argument: `args[0]`
  - second command line argument: `args[1]`
  - third command line argument: `args[2]`, *etc.*.
- Each of these is a `String`
- This will be explained in detail in a few weeks, but this will be enough for now

Command Line Input Example

```
// print out a friendly greeting
public class Hello2 {
    public static void main(String[] args) {
        System.out.println("Hello, " + args[0] + "!");
    }
}
```

Program Use

```
frege% java Hello2 Peter
Hello, Peter!
frege% java Hello2
Exception in thread "main" java.lang.ArrayIndexOutOfBo
undsException: 0
        at Hello2.main(Hello2.java:4)
```

Handling Command Line Inputs

- To converts string to int:
  `Integer.parseInt(`*`string`*`)`

```
// Print double the command line integer
public class Hello3 {
    public static void main(String[] args) {
        System.out.println("Twice your number is "
                           + 2 * Integer.parseInt(args[0]));
    }
}
```

Program Use

```
 frege% java Hello3 4
Twice your number is 8
```

Reading Console Input

- Interactive programs get input while running
- Java 5 introduces the `Scanner` class for this
- To use `Scanner`:
  - Add this near top of source file:

```
import java.util.Scanner;
```

  - Create scanner: add this in `main` before reading input:

```
Scanner keyboard = new Scanner(System.in);
```

  - Use `keyboard` as needed to read input
  - *E.g.*, this reads (rest of) current line as a string:

```
String line = keyboard.nextLine();
```

Reading Console Input

- Other methods to read from a `Scanner` variable `keyboard`:

What	Type	Expression
One word	`String`	`keyboard.next()`
One integer	`int`	`keyboard.nextInt()`
One double	`double`	`keyboard.nextDouble()`

- Similar methods to read other types; see documentation
- These all skip over whitespace and read one "word"
- Whitespace includes spaces, tabs, and newlines
- Error if text is not of expected type

Command Line and Scanner Example

```
import java.util.Scanner;
public class ScannerExample {
    public static void main(String[] args) {
        int num1 = Integer.parseInt(args[0]);
        Scanner kbd = new Scanner(System.in);
        System.out.print("Enter second number: ");
        int num2 = kbd.nextInt();
        System.out.println(num1 + " * " + num2 +
                           " = " + num1*num2);
    }
}
```

```
frege% java ScannerExample 6
Enter second number: 7
6 * 7 = 42
```

Pitfall: Mixing with nextLine

- `nextLine()` reads up to and including newline
- Others do not read after the next word
- After `next`, `nextInt`, or `nextDouble`, `nextLine` just reads rest of current line (maybe nothing!)
- To read a number on one line followed by the next whole line:

```
int num = keyboard.nextInt();
keyboard.nextLine(); // throw away rest of line
String line = keyboard.nextLine();
```

- Ideally, avoid mixing `nextLine` with the others

Pitfall: Multiple Scanners

- A scanner reads a chunk of input at a time
- Holds onto it until it is requested
- If you create multiple scanners, each will hold onto some of the input
- You may get input in the wrong order or even lose some input
- Simple solution: only ever create one scanner in a program

Multiple Scanners Go Wrong

```
Scanner kbd1 = new Scanner(System.in);
Scanner kbd2 = new Scanner(System.in);
System.out.print("Enter two numbers: ");
int num1 = kbd1.nextInt();
int num2 = kbd2.nextInt();
System.out.print("Enter two numbers: ");
int num3 = kbd1.nextInt();
int num4 = kbd2.nextInt();
System.out.println(num1 + " * " + num2 +
                   " = " + num1*num2);
System.out.println(num3 + " * " + num4 +
                   " = " + num3*num4);
```

(Showing only the body of the method.)

Multiple Scanners Go Wrong

Behaviour of previous example

```
frege% java MultiScanner
Enter two numbers: 6 7
3 4
Enter two numbers: 6 * 3 = 18
7 * 4 = 28
```

Summary

- Use `+` to append strings
- `+` converts one operand to string if the other is a string
- `System.out.printf` does formatted output
- `args[`n`]` is the nth command line argument
- Use `java.util.Scanner` to read from the console
- Never create multiple `java.util.Scanner` objects

THE UNIVERSITY OF
MELBOURNE