CSE1710

Week 06, Lecture 12

Fall 2013 ◆ Thursday, Oct 17, 2013

YORK UNIVERSITÉ UNIVERSITY

Big Picture

The assigned reading was for today:

- ❑ read section 3.2 “A Development Walk-Through”
- ❑ review Ch 3 KC’s 7-14
- ❑ do Ch 3 RQ’s 13-25
- ❑ do Ch 3 Ex’s 3.12-3.16
- ❑ last week’s lab covered Lab Exercise L3.2 “A Software Project” (pp. 124-126), also listed as Ex 3.18

YORK UNIVERSITÉ UNIVERSITY

Checklist (for next time, Lecture 13)

What you should be doing to prepare for what comes next…

- ❑ read section 3.3
- ❑ review Ch 3 all key concepts, all review questions, remainder of exercises
- ❑ do Ch 3 Ex's 3.17.

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

- `println`, `print`, and `printf` : are all of these services of the same class?
- overloading:
  - is `printf` an overloaded method?
  - is `print` an overloaded method?
  - is `println` an overloaded method?
- How are `print` and `println` similar? How are they different?
- Is `printf` more similar to `print` or to `println`?

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

- Which class or classes offer the services `println`, `print`, and `printf`?
- overloading:
  - is `printf` an overloaded method?
  - is `print` an overloaded method?
  - is `println` an overloaded method?
- How are `print` and `println` similar? How are they different?
- In what key way is the signature of `printf` different from the signatures of `print` and `println`?
- Is `printf` more similar to `print` or to `println`?

YORK UNIVERSITÉ UNIVERSITY

5

Comprehension Q's: `printf`

- The method `printf` has a format specifier.
  What is the best general purpose specifier to use?
  - Suppose you want to replicate the result of `print(67.8)` or `print("hello")` but using `printf` instead?
- overloading:
  - is `printf` an overloaded method?
  - is `print` an overloaded method?
  - is `println` an overloaded method?

YORK UNIVERSITÉ UNIVERSITY

6

Comprehension Q's: `printf`

- In terms of behaviour, how are `print` and `println` similar? How are they different?
- In terms of being services within the `PrintStream` class, how are `print` and `println` similar? How are they different?
- In what crucial way is the signature of `printf` different from the signatures of `print` and `println`?
- Is `printf` more similar to `print` or to `println`?

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

The method `printf` has a format string.

The format string is a `String` which may contain fixed text and zero or more embedded *format specifiers*

Idea #1: The format specifiers are optional

Consider what happens when you leave out them out

- `stdOut.printf("hello");`
- `stdOut.printf("56");`
- `stdOut.printf("56\n");`
- `stdOut.printf("\t56\n");`

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

The format string is a `String` which may contain fixed text and zero or more embedded *format specifiers*. Each format specifier requires a conversion character.

Idea #2: The most basic *conversion* for the format specifier is `%s`.

The number of format specifiers must match the number of arguments (in addition to the format string)

Consider the following

- `stdOut.printf("hello%s",56f);`
- `stdOut.printf("56%s",4L);`
- `stdOut.printf("56%s%s%s\n",33,88.7,"hi");`
- `stdOut.printf("%s\t56\n", 33.567);`

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

Idea #3: The "`\n`" string is equivalent to the format specifier `%n`.

Consider the following:

- `stdOut.printf("hello\n");`
- `stdOut.printf("hello%n");`

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

Idea #4: The difference between the `d` and `f` conversion characters: `d` takes `int`/`long`, `f` takes `float`/`double`

Consider the following:

- `stdOut.printf("ans: %d%n",56);`
- `stdOut.printf("ans: %d%n",56L);`
- `stdOut.printf("ans: %f%n",56.67);`
- `stdOut.printf("ans: %f%n",56.67f);`

In case of mismatch, what happens?

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

Idea #5: The `width` component specifies the number of characters for the output. The output is right justified.

Consider the following:

- `stdOut.printf("ans: %10s%n", "hi");`
- `stdOut.printf("ans: %10d%n", 56);`
- `stdOut.printf("ans: %10f%n", 56.6798);`

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

Idea #6: The `precision` component cannot be used with `int/long`; only for `float/double`

Consider the following:

- `stdOut.printf("ans: %10.2s%n", "hi");`
- `stdOut.printf("ans: %10.2d%n", 56);`
- `stdOut.printf("ans: %10.2f%n", 56.6798);`

YORK UNIVERSITÉ UNIVERSITY

Comprehension Q's: `printf`

Idea #7: The `flag` component can be `,` and/or `0`. It can be used only with numerical types

Consider the following:

- `stdOut.printf("ans: %010d%n", 5666);`
- `stdOut.printf("ans: %,10d%n", 5666);`
- `stdOut.printf("ans: %0,10d%n", 5666);`
- `stdOut.printf("ans: %010.2f%n", 98956.6798);`
- `stdOut.printf("ans: %,10.2f%n", 98956.6798);`
- `stdOut.printf("ans: %0,10.2f%n", 98956.6798);`

YORK UNIVERSITÉ UNIVERSITY

Overview: `printf`

- The first parameter holds format specifiers
- Each specifier has the form:
  `%[flags][width][.precision]conversion`
  - flags can be: , and/or `0`
  - width: how many characters to be allocated
  - precision: # of decimal digits (for `f` conversion letter only)
  - conversion letter can be:
    - `d : int/long`
    - `f : float/double`
    - `s : string`
    - `n : new line`

YORK UNIVERSITÉ UNIVERSITY

About the Dev't Process

- What is the difference between the **requirements analysis** phase and the **design** phases?
- At what stage does coding take place?
- At what stage would the designer create UML class diagrams?
- In the **implementation** stage, how does the implementer know what functionality should be implemented?
- In the **testing** phase, a set of test cases are uses to assess correctness.
  What is the basis for the creation of these test cases?

YORK UNIVERSITÉ UNIVERSITY

3.2.4 Relational Operators

- Numeric operands: **< <= > >=**
- Numeric/boolean operands (any type): **== !=**

- All relational operands **violate closure**
- No matter what the operand type is, the result type is always **boolean**.

YORK UNIVERSITÉ UNIVERSITY

Operator Precedence

Precedence	Operator	Operands	Syntax	`true` if
-7 ➔	<	numeric	`x < y`	`x` is less than `y`
	<=	numeric	`x <= y`	`x` is less than or equal to `y`
	>	numeric	`x > y`	`x` is greater than `y`
	>=	numeric	`x >= y`	`x` is greater than or equal to `y`
	`instanceof`	`x instanceof C` is true if object reference `x` points at an instance of class `C` or a subclass of C.		
-8 ➔	==	any type	`x == y`	`x` is equal to `y`
	!=	any type	`x != y`	`x` is not equal to `y`

YORK UNIVERSITÉ UNIVERSITY

Relational Operators & Non-Primitive Types

```
Rectangle r1 = new Rectangle(10, 10);
Rectangle r2 = new Rectangle(10, 10);
Rectangle r3 = new Rectangle(20, 20);
boolean isEqual1 = r1==r2;
boolean isEqual2 = r1==r3;
```


YORK UNIVERSITÉ UNIVERSITY

Coming up: Working with Images

To work with images, we need to:

- work with the file system
- work with the operating system's window manager and the platform's graphics hardware
- understand colour models and representation formats
- iterate and construct conditions

YORK UNIVERSITÉ UNIVERSITY

File pathnames are system dependent

- Windows Local File System (LFS):
  - `C:\USER\DOCS\LETTER.TXT`
- Windows Uniform Naming Convention (UNC)
  - `\\Server\Volume\File`
- Unix-like OS
  - `/home/user/docs/Letter.txt`

Which details are system dependent?

What can be abstracted away?

- separator (e.g., `/`, `\`) `File.separator`
- system prefix (e.g., `/`, `\\`, `C:\`)

also **lists** of pathnames are system dependent

- Windows Local File System (LFS):
  - `C:\USER\DOCS\;C:\BIN`
- Unix-like OS
  - `/home/user/docs/:/usr/bin/:/sbin/`

Which details are system dependent?

What can be abstracted away?

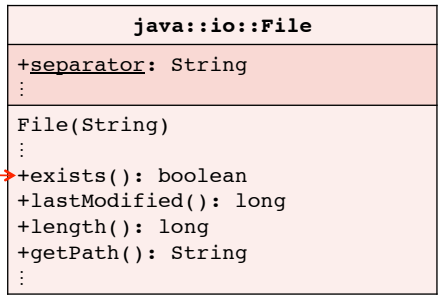
- path separator (e.g., `;`, `:`) `File.pathSeparator`

Useful class: `java.io.File`

- The class `java.io.File` encapsulates a *file* on the platform's file system
  - a *file* in this context can be
    - a directory
    - a "normal file" (i.e., not a directory)
  - files constructed from *pathnames*
- the class File is **not** utility
  - it provides *some* static features to encapsulate system-dependent elements
    - separator, path separator
    - demo: `File_Example01`

Recap: the `File` class

just because an object is instantiated for a pathname doesn't necessarily mean that there is an actual file corresponding to that pathname



The `File` class encapsulates information about and operations on either potentially-existing files and already-existing files.

Services provided by the `File` class

- provides constructor for object creation, given a pathname
  - demo: `File_Example01`
- provides delegation of file-related tasks:
  - does **this** file exist?
  - is **this** file a *directory* or a *normal file*?
  - what is the size of **this** file?
  - can I write to **this** file?
  - which files are in **this** directory, if any?
    - assumes this file is a directory
  - make a directory, as specified by **this** file
    - assumes pathname is not already in use and operation is allowed

Services provided by the `File` class

- Additional services
  - can I write to **this** file?
  - which files are in **this** directory, if any?
    - assumes this file is a directory
  - make a directory, as specified by **this** file
    - assumes pathname is not already in use and operation is allowed

The encapsulation of a `File`…

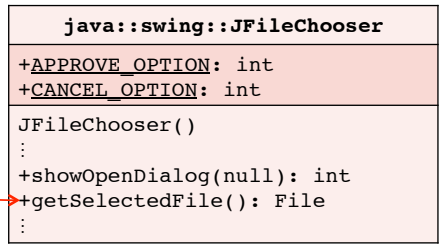
- provides services to ask whether the file is **writable**
  - this tells you whether the permissions and other conditions are favourable
- the `File` class **does not** provide the means to *write* to the file object ☹
  - for this, you need the services of `FileWriter`
  - a `FileWriter` object encapsulates all of the working of writing content to a `File` object
  - defer this aspect for the time being

Another way to interface with the file system

- Use the services of the Swing package, which has a class called `JFileChooser`
- let the user **specify** one for you
  - `FileChooser_Example01`

the `JFileChooser` class

`getSelectedFile()` will always return *something*, even before the dialog is even opened



`JFileChooser` encapsulates information about and operations on a file choice dialogue.