COSC 2P03 – Assignment 1 – You couldn't find the manual if you had a map!

Objective:

This assignment will focus on two tasks: creating a simple abstract data type to create an arbitrary number of pairings between labels/keys and text entries, and then using that type to write a simple *man* suite, with *apropos* and *whatis* functionality.

…you're probably wondering what the heck that means, right?

man:

The *man* command is commonly found in Linux environments to act as the instruction manual for terminal commands.

- e.g. `man ls` brings up the instructions for how to use the `ls` command for directory listings
- Each such entry is a pretty decent length of text, but that's basically all it is: a selection of text

whatis:

The *whatis* command is available for providing brief summaries of commands. For the sake of brevity: it just displays the name of the command, followed by the first line of the man page.

- e.g. for `ls`, it says *ls - list directory contents*

apropos:

The *apropos* command is used when you know the basic functionality you're looking for, but can't recall the command.

- e.g. `apropos directory` lists several commands, including `ls` (as you'd expect)
- Its precise output is simply multiple lines: the same output as *whatis*, for each matching command

The Abstract Data Type: a searchable reference

This is going to be, more or less, very similar to the mapping we already did in lecture. In some ways, it's actually simpler! (In other ways, not quite so)

We'll call it a **`Lookup`**. Note that this is a mandatory name. Since we're talking about formal specifications for things, names *matter*! A tool that 'roughly does the job', but that you can't count on? Worthless.

So here's how a `Lookup` works:

- It's part of the `mappings` package
- You can `add` as many pairings of key and entry as you'd like
  - Each is mapping a *string* to a *string*
  - If you try adding an entry corresponding to an already-existing key? You simply replace the entry for that key (note: all keys are *stored as lowercase-only*)
- There is no way to *remove* an entry, but there's a catch to this (basically, keep reading)
- You can `lookup` a *key*
  - This simply returns the corresponding entry for that key (treated as lowercase)
  - If it doesn't exist, then that throws a `NoSuchCommandException` (part of the same package)
- You're implementing a search:
  - It receives a *search term*, which it will try to match within entries *and* keys themselves
  - It is *not* case-sensitive!
  - It returns an array, containing only the *keys* for matches
    - All searches are 'legal'; the array is properly-sized to the number of matches

- It has a `getKeys`, which gives an array of all the keys in the `Lookup`:
  - Obviously if it's empty, then the array simply has a length of zero
- It can return an `excerpt`: a *new* instance of a `Lookup`, constrained only to those keys specified (as an array) in its parameter
  - Technically, this can also act as a minimal substitute for being able to remove entries
  - If the provided keys has *any* entries not present, it's the same problem as looking up a single non-existent key (and must be handled the same way)
- It has the ability to return its total `size`, but it might not be what you're expecting!
  - What it returns is the total character count of *all keys and entries*!
  - (It's basically a way of approximating how big it would be to print everything out)
- It has the ability to generate 'one big `listing`' of its entire contents as a `String`:
  - (Obviously the formatting tells you the name; you've picked up on that convention, right?)
  - For each entry, it includes both the key and the corresponding entry
    - Have the key be the first term in the line, then a tab, then the entry (obviously some entries will have multiple lines; but that's not your concern)
  - Between each entry, have a 'break' of ten dashes
  - The listing should be sorted, by key
  - Also, though your interface can't do this, any concrete type will be expected to override `toString`, to simply return the same contents as the `listing`
- It can render and return itself `asArray`:
  - This simply yields a 2D array of *strings*: first column is keys; second is corresponding entries
- It can render itself (`save`) into an `ASCIIOutputFile`
  - It receives a (presumably newly-instantiated) ASCIIOutputFile as its only parameter
  - It does *not* close the file itself
  - The convention for precisely how it does this is mostly up to you, but *be reasonable*!
- It has three constructors:
  - The default constructor (duh)
  - One that can populate its contents from an ASCIIDataFile
    - The convention for this file is up to you; you'll be explaining it later in the assignment
  - One that can accept the array provided by the `asArray` function
    - An array of length zero is fine; but if the *convention* isn't followed (e.g. a ragged array), simply throw a general `RuntimeException`. Because... no

Of course, you should properly document your interface (with proper JavaDoc).

Your implementation of the `Lookup` will be a `SortedLookup`.

- Obviously also part of the `mappings` package

The man program:

You'll be creating a simple BasicIO program (using a BasicForm, in a `man` package) to allow the user to perform any number of *man*, *whatis*, and *apropos* queries. It's pretty straightforward; the reason for doing this is to ensure you're comfortable separating the *programs* that you create from the *tools* it uses.

There are some screenshots at the end, if needed.

The test harness:

You'll also be implementing a simple test harness, part of the `testing` package, to 'put your ADT through its paces'. Just demonstrate that it can support all of the required behaviours (since they're not actually all covered by the *man* program). Note that your TA might use an alternate testing program, so adherence to specifications is mandatory!

Additional requirements and tips:

- Note that there are three completely different packages here, and probably ~6 classes
- To confirm: when something is in `Courier`, that's indicating a **specific** name
- The only imports you're using are the BasicIO library, and the library you're creating
- You don't need regex. Don't be suspicious and use regex
- Note that it's up to *you* to define the convention for your 'data file' for the lookups, but whatever you pick needs to align with the rest of the requirements. Please don't overthink this
    - Remember that an entry may be comprised of multiple lines of text, so you can't simply alternate lines between key/entry/key/entry/etc.
    - Remember: you have an instructor. He likes questions
- You'll need to include a **writeup** containing two things:
    - A *brief* complexity analysis of your operations (for the `SortedLookup`); and
    - A quick explanation of your file convention for storing/loading `Lookup`s
    - Use .pdf format for this
- Make sure you don't 'overengineer' your solution, particularly to the point of diverging from the specification
    - e.g. even though we'd covered one approach to creating a mapping that was generic, that doesn't mean this `Lookup` can be! It connects strings to strings. Nothing more; nothing less
- You can probably guess this, but as this task includes components that rely *heavily* on specifications, the evaluation will necessarily be strict in this regard: follow instructions

Submission:

This course (and by extension, this assignment) uses IntelliJ. Do not use any other IDE.

- As a clarification, you aren't writing in something else then importing into IntelliJ. You aren't expecting the marker to fix your project for you. You're including a run *configuration* that explicitly includes the main class
- (If you have any questions, don't forget you have, like, an instructor you can ask?)

You are expected to adhere to certain minimal standards *before* true evaluation of your work even starts:

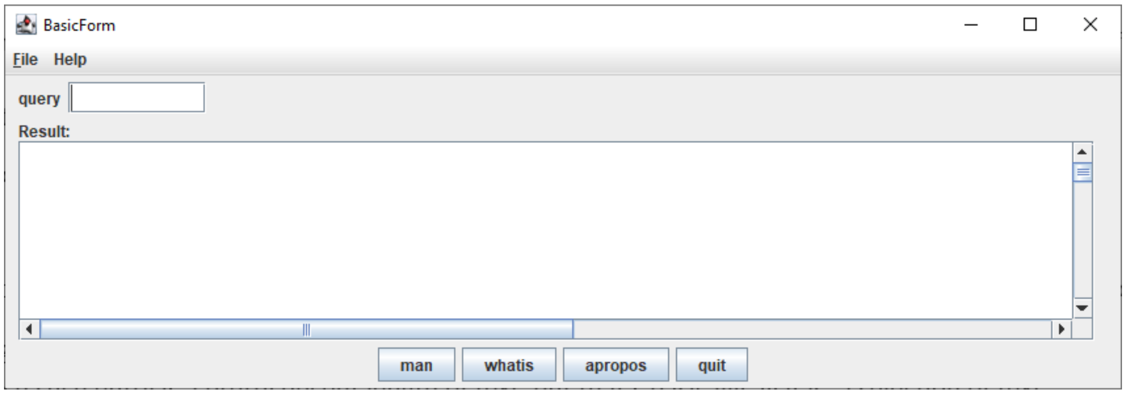
- Comment your code
    - Comment what it *does*; not what it *is*!
    - e.g. indicating that a 'for loop' is 'a loop' is worse than nothing. Don't do that
- Document your code
    - No, this is not the same thing
- Follow basic standards
    - Avoiding hardcoded literals
    - Obviously nothing 'static' other than the main method of the main class
    - (You get the idea!)
- Beyond BasicIO and your own library, no `import`s. At all. Seriously.
- Useful variable names. Organized. Indented correctly.
    - i.e. your submission will be readable, or you did not submit

Include a sample output, along with your entire IntelliJ project and summary.
Bundle/compress everything into a single .zip file.

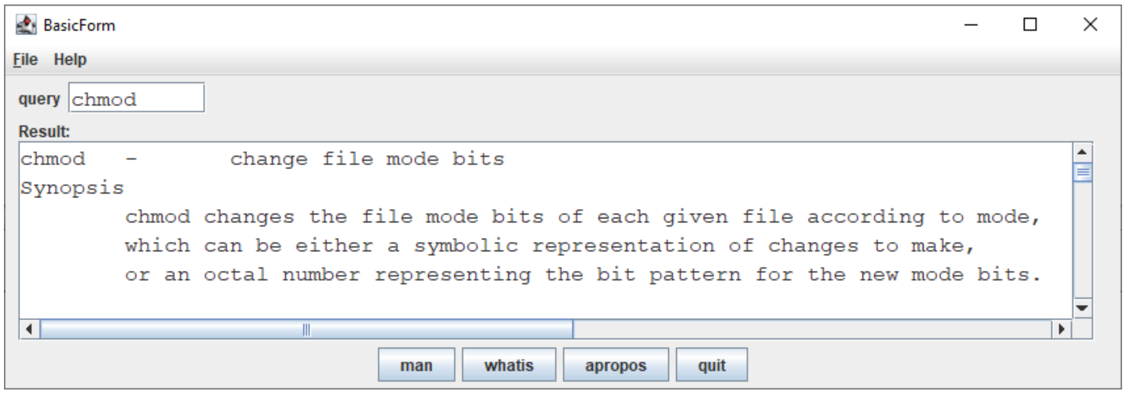
- Not a .rar
- Not a .7z
- Not a .tar.gz
- Please don't decide to earn a zero from something so trivial

Submit your .zip in Sakai.

Sample execution of the man program:

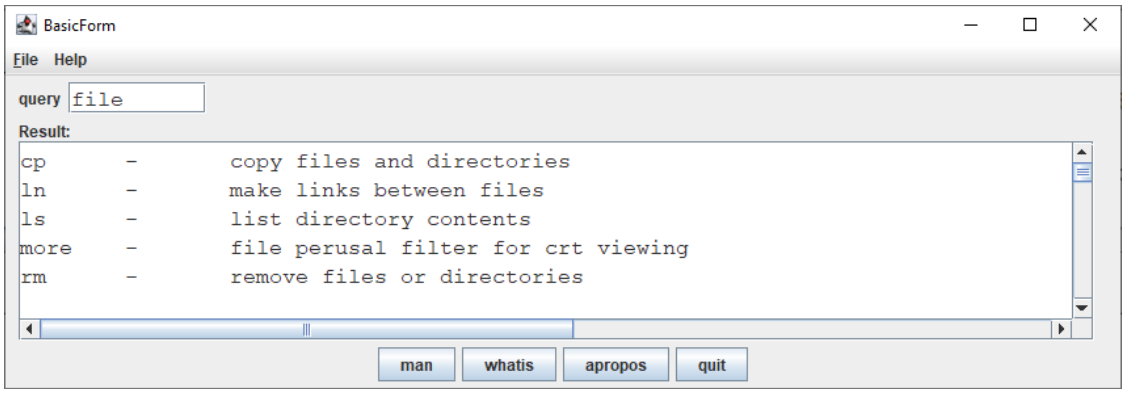


I just populated a file with a few bash commands. You can make something up, so long as the style is preserved. In this case, I typed *chmod* and clicked *man*:



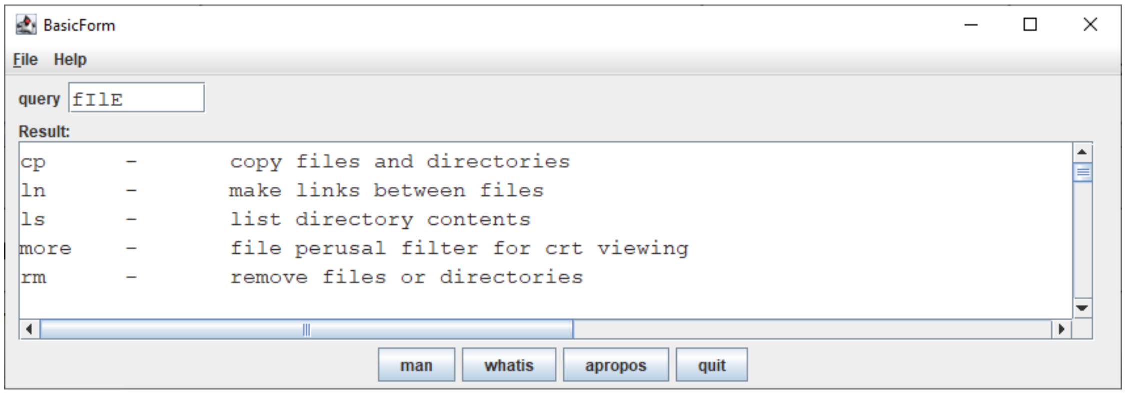
```
chmod   -       change file mode bits
Synopsis
        chmod changes the file mode bits of each given file according to mode,
        which can be either a symbolic representation of changes to make,
        or an octal number representing the bit pattern for the new mode bits.
```

Speaking of files, here's an *apropos* for the term *file*:



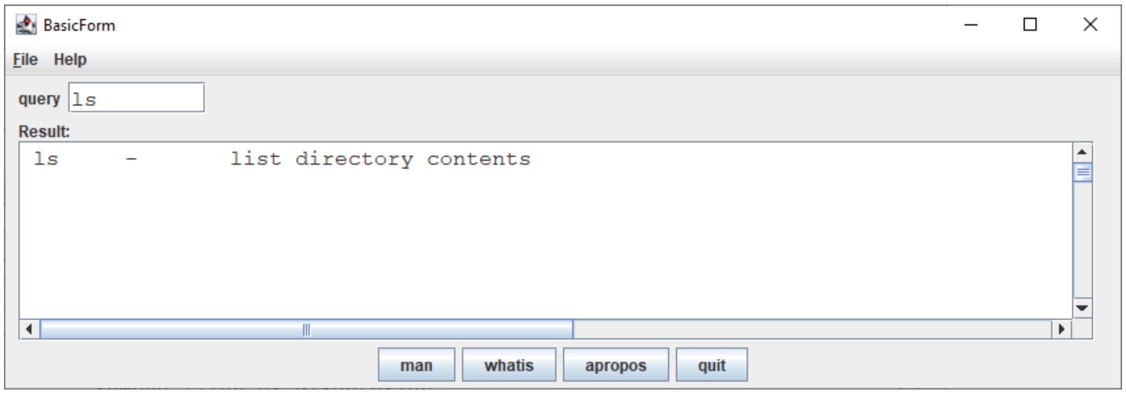
```
cp      -       copy files and directories
ln      -       make links between files
ls      -       list directory contents
more    -       file perusal filter for crt viewing
rm      -       remove files or directories
```

And capitalization doesn't matter:



```
cp      -       copy files and directories
ln      -       make links between files
ls      -       list directory contents
more    -       file perusal filter for crt viewing
rm      -       remove files or directories
```

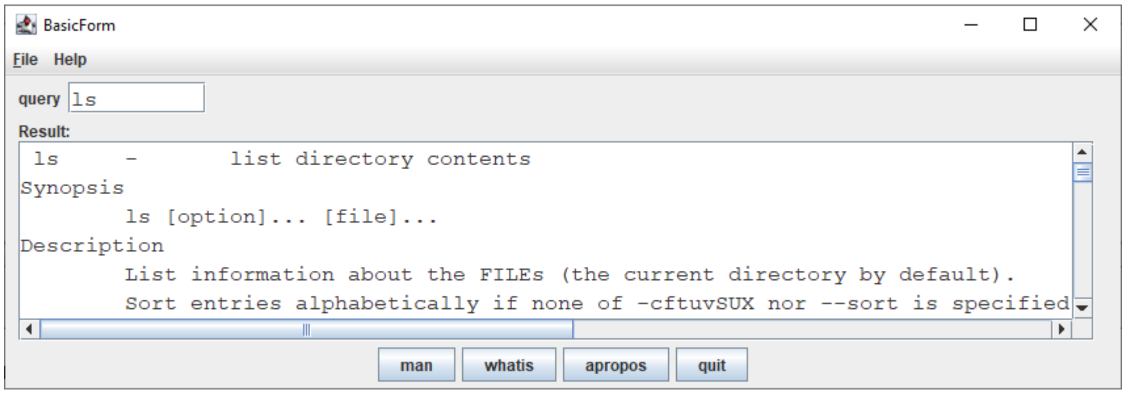
I can find out what *ls* is for:



```
ls
```

```
 ls       –        list directory contents
```

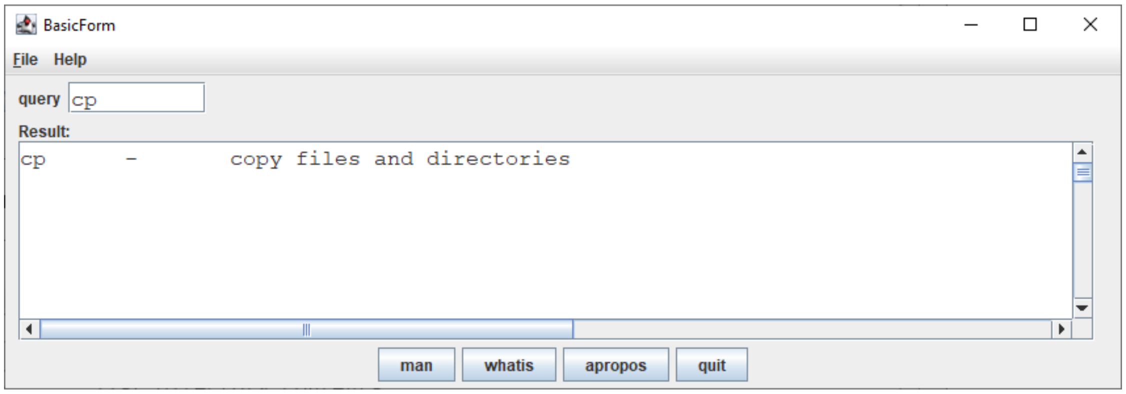
or get its full *man* page:



```
ls
```

```
 ls       –        list directory contents
Synopsis
        ls [option]... [file]...
Description
        List information about the FILEs (the current directory by default).
        Sort entries alphabetically if none of -cftuvSUX nor --sort is specified
```

Since the lookup's *search* includes both entries *and* keys, apropos matches on commands as well as instructions:



```
cp
```

```
cp       –        copy files and directories
```