Java Lab 11: Memo Mania

This lab will re-use your solution to the Lab 10 project. Create a new project, Lab11, and copy your code into it, using the same package structure. When the problem says to change or replace some existing code, you should comment out the old line and add the new line, just in case you have to revert.

1. Add a static factory method in NoteCollection: **Note createNote(String type, String name, String body, String from, String to)**, where type is one of the concrete note types ("Memo", "TimedMemo", or "PoliteTimedMemo"). Change every constructor in the Note hierarchy to package private. Use createNote( ) in main( ) in the places you previously new'd up notes – change code like "Memo memo = new Memo()" to this: "Note note = new NoteCollection.createNote("Memo", etc.)". Make sure it all works.

2. In **Note**, add a getter for the **body** field and add **abstract** getters for the **to** and **from** fields (this is not a great solution, but go with it). Then add regular getters from the **to** and **from** fields in **Memo** and a getter for today in TimedMemo.

3. Create a class called **NoteDisplayer** that contains the following abstract methods:

public abstract void display(Note note).

4. Extend **NoteDisplayer** with these classes; override display( ) in each.

class SimpleNoteDisplay

display(Note) just prints Note.toString( )

class FancyNoteDisplay

display(Note) prints the note information as shown below

```
Enter your choice: 1
************************
* Number: 1            *
* Name  : Memo         *
* From  : Barrett      *
* To    : Jones        *
************************
*
* Body:
This is a regular memo
```

class ReallyFancyNoteDisplay

display(Note) adds the greeting and closing *if* the Note is a PoliteTimedMemo; otherwise, does the same as FancyNoteDisplay

5. Make these changes to Note: add a data field **NoteDisplayer noteDisplayer = null**. Add the method

public void display( ) { noteDisplayer.display(this) }

Also add a setter for noteDisplayer; this will be called in the factory method.

Change the factory method **createNote** to add one more parameter, printType, that is one of "Simple", "Fancy", or "Really". Create the appropriate NoteDisplayer object for each and set the Note's noteDisplayer field to it.

6. Find the places in MakeANote where the notes are created. For Memo, add “Simple” to the parameter list; for TimedMemo, add “Fancy”, and PoliteTimedMemo, add “Really”.

What pattern is being used, and what are its parts?

Find the calls to createNote for print(toString) and change them to display( ).

7. This problem is practice with the Builder method, but just for the **PoliteTimedMemo** class. Following the pattern outlined in the notes:

- create a **public static Builder** class *inside* the **PoliteTimedMemo** class;

- since all four parts will be required, Builder needs only a default constructor;

- create four methods in Builder, one each for the four fields name, body, from, to; the return type for each should be Builder, and each should return this;

- create a **build( )** method in Builder that returns a new PoliteTimedMemo with this as the parameter using the next method;

- add a private PoliteTimedMemo constructor taking a Builder as a parameter; it should set this's fields base on the parameters fields. Call super( ).

Now test the Builder: in NoteCollection's factory method, replace the call to new PoliteTimedMemo to one using the PoliteTimedMemo.Builder, calling the four methods (in any order – you should test it with different orderings to check that it works) and build( ).

Was it worth it? Seems like a lot of trouble!