COMP 401 Final: Question Part

Thursday, Dec 14, 2017 12pm-3pm

Instructions

1. **If you believe you have an A without the final, please indicate so in the space provided to you in the answer book**.
2. Please spread out and try and sit in alternate seats.
3. This is a closed book exam.
4. You will not be penalized for errors in Java syntax.
5. Do not provide your answer in this question document. A separate answer document is also being distributed.
6. The question document has:
   - 4 numbered pages including this one and any marked blank pages.
   - 2 questions.
7. The answer document has 4 numbered pages including marked blank pages.
8. Point values appear in brackets next to each question. **Points have been allocated for writing the onyen, name and email clearly in capitals.**
9. You are not required to comment or annotate any code you write, but may get partial credit if you write appropriate comments/annotations but incorrect code.
10. If you need to make any assumptions to clarify a problem, *write your assumptions down.* Only reasonable assumptions get full credit.
11. Please inform the proctor of anything in the exam that you think is a mistake.
12. Your code will be evaluated not only for correctness, but also for time and space efficiency and *style*. Certain relevant stylistic rules, covered in class, are not explicitly stated. *Y*ou must adhere to the same requirements that you would if this was a class assignment (with a perfect human grader!), except where explicitly stated otherwise. This implies, for instance, if you are asked to implement a class or subclass, you must define any additional types (classes and interfaces), if necessary, to follow the design principles, pattern and other concepts covered in class. Similarly, if you are asked to implement certain methods, you should write other methods, if necessary to follow stylistic rules.
13. If a class or method header is given to you, do not change it.
14. You cannot use any Java capabilities not covered in class.
15. If you do not understand some English word, do not hesitate to ask the proctor. Naturally, you are expected to know the computer science terms defined in class.
16. **Walk over to the proctor to ask questions in private without disturbing others**, do not raise your hand.
17. Write clearly using a pen/pencil with a dark color – we will be scanning your exams. You will lose points if we cannot understand what you wrote. Please try and write in the allocated space on the answer document, as that will reduce mistakes in grading of scanned exams. **We strongly recommend that you answer on scratch paper before committing your solution to the answer sheet. Do not unstaple your answer part** – we may not grade it if it is unstapled
18. Put your initials on the front page of the answer document and on each page of the document, in case the pages are separated.
19. If your answer overflows to the extra page, please indicate that explicitly in the space provided for the answer – otherwise we will miss it.

Study and understand the types - classes and interfaces - below. This program is used in all questions. All types are declared in package final_F17. Package declarations and imports are omitted to conserve space. The program uses some standard classes and interfaces studied in class. The relevant aspects of them are given in Q2. They are not needed in Q1.

```
public interface TwoItemSpreadsheet<ItemType> extends PropertyListenerRegisterer {
      public ItemType getItem();
      public ItemType getDependentItem();
      public void setItem(ItemType newValue);
}
```

```
public abstract class ATwoItemSpreadsheet<ItemType> implements TwoItemSpreadsheet<ItemType>
{
      PropertyListenerSupport propertyListenerSupport = new APropertyListenerSupport ();
      ItemType item;
      public ItemType getItem() {
            return item;
      }
      public void setItem(ItemType newValue) {
            item = newValue;
      }
      protected abstract ItemType calculateDependentItem();
      public ItemType getDependentItem() {
            return calculateDependentItem();
      }
      public void addPropertyChangeListener(PropertyChangeListener aListener) {

      }
}
```

```
public class ATwoStringSpreadsheet extends ATwoItemSpreadsheet<String> {
      public static final String INIT_ITEM = "Change Me";
      public ATwoStringSpreadsheet() {
            item = INIT_ITEM;
      }
      protected String calculateDependentItem() {
            return getItem().toLowerCase();
      }
}
```

```
public class ATwoStringSpreadsheetView extends Component implements
PropertyChangeListener {
      public static final int ITEMS_X = 50; // x coordinate of both model properties
      public static final int ITEM_Y = 50; // y coordinate of independent item
      public static final int DEPENDENT_ITEM_Y = 100; // y of dependent item
      protected TwoItemSpreadsheet<String> model;
      public ATwoStringSpreadsheetView(TwoItemSpreadsheet<String> aModel) {
            model = aModel;
      }
      // implements sole method in PropertyChangeListener
      public void propertyChange(PropertyChangeEvent evt) {
            System.out.println(evt);
      }
}
```

```
public class TwoStringSpreadsheetDriver {
	public static final int FRAME_WIDTH = 300;
	public static final int FRAME_HEIGHT = 300;
	public static final String EXAMPLE_STRING = "Hello";
	public static void main(String[] args) {
		JFrame aFrame = new JFrame();
		TwoItemSpreadsheet<String> aModel = new ATwoStringSpreadsheet();
		Component aView = new ATwoStringSpreadsheetView(aModel);
		aFrame.add(aView);
		aFrame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
		aFrame.setVisible(true);
		aModel.setItem(EXAMPLE_STRING);
	}
}
```

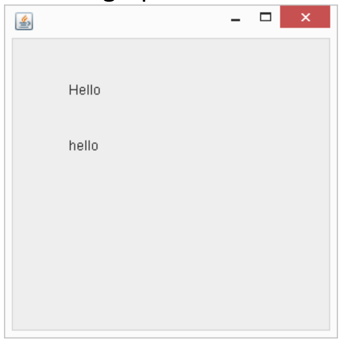
1. Answer questions about the program asked in the Q1 answer part.

2. Your task in this question is to change two of the classes given above, `ATwoItemSpreadsheet` and `ATwoStringSpreadsheetView`, with the following related goals:
   - `ATwoItemSpreadsheet` should behave like a model in the MVC design pattern.
   - `ATwoStringSpreadsheetView` should behave like a view in the MVC design pattern that displays the two properties of the model in a graphics window and the console. In the graphics window, the two properties are displayed by showing the string values at the locations specified by the named constants in the class. In the console window, the properties are displayed by printing the corresponding property notification events.

At the end of the execution of the main method (of `TwoStringSpreadsheetDriver`), which now uses the modified versions of these two classes, the console window should show the following two property changes:

```
java.beans.PropertyChangeEvent[propertyName=Item; oldValue=Change Me; newValue=Hello;
propagationId=null; source=final_F17.ATwoStringSpreadsheet@1efed156]
java.beans.PropertyChangeEvent[propertyName=DependentItem; oldValue=change me; newValue=hello;
propagationId=null; source=final_F17.ATwoStringSpreadsheet@1efed156]
```

and the graphics window should display the ending property values.



Thus, the MVC protocol should result in the program displaying the two property change events in the console and the model property values in the graphics window.

Some of the code needed to implement the MVC protocol is the program is given to you including named constants for positions of the two properties in the graphics window. You must make necessary changes to this code to implement the full protocol. You should change only `ATwoItemSpreadsheet` and `ATwoStringSpreadsheetView`. *No other interface or class, including the main class, should be changed*.

You can use the following method of Graphics2D:

```
drawString(String str, int x, int y);
```

Renders the text of the specified `String`, using the current text attribute state in the `Graphics2D` context.

You can call/override all methods of the class Component – if you do not know the exact syntax of methods you need, invent appropriate syntax.

The header of the constructor of `PropertyChangeEvent` is given below.

```
public PropertyChangeEvent(Object source, String propertyName,
                                    Object oldValue, Object newValue);
```

Your code can use the following familiar types, presented in praxes.

```
public interface PropertyListenerSupport {
      public void add(PropertyChangeListener l);
      public void notifyAllListeners(PropertyChangeEvent event);
}
```

```
public class APropertyListenerSupport implements PropertyListenerSupport {
      List<PropertyChangeListener> observers = new ArrayList<PropertyChangeListener>();
      public void add(PropertyChangeListener l) {
            observers.add(l);
      }
      public void notifyAllListeners(PropertyChangeEvent event) {
            for (int index = 0; index < observers.size(); index++) {
                  observers.get(index).propertyChange(event);
            }
      }
```

```
public interface PropertyListenerRegisterer {
      public void addPropertyChangeListener(PropertyChangeListener aListener);
}
```