CS 2334
Project 4: Graphical User Interfaces

October 28, 2017

Due: 1:29:00 pm on Monday, Nov 13, 2017

Introduction

For the last three projects, you have been focused on reading data from files and constructing large, efficient representations from the data. For this project, we will focus on presenting these data to a user, enabling the user to explore the statistics associated with specific field and subfield names, and weeks.

Your implementation from project 3 will continue to serve as the basis for data loading and representation (with minimal changes). What you will add is a graphical user interface that interacts with the user.

Your final product will:

1. Allow the user to specify an infant whose files can be loaded.
2. Allow the user to select one or more weeks of interest, a field and a subfield.
3. Report detailed information about the specified field/subfield, as well as the minimum, maximum and average over the specified weeks.

Learning Objectives

By the end of this project, you should be able to:

1. Create a menu that is attached to a frame.

2. Make use of JLists that present a set of options to a user and allow the user to select one or more of these options

3. Create a set of components that display textual data to a user

4. Create the listeners necessary to allow the GUI to respond to user input

5. Continue to exercise good coding practices for Javadoc and for testing

Note that this project relies heavily on your reading of the Java API documentation, and the examples. We have tried to provide you with a good set of hints, but, fundamentally, you have to pull the details out of the documentation.

Proper Academic Conduct

This project is to be done in the groups of two that we have assigned. You are to work together to design the data structures and solution, and to implement and test this design. You will turn in a single copy of your solution. Do not look at or discuss solutions with anyone other than the instructor, TAs or your assigned team. Do not copy or look at specific solutions from the net.

Strategies for Success

- The UML is a guide to the new classes and methods that you will implement.
- When you are implementing a class or a method, focus on just what that class/method should be doing. Try your best to put the larger problem out of your mind.
- We encourage you to work closely with your other team member, meeting in person when possible.
- Start this project early. In most cases, it cannot be completed in a day or two.
- Implement and test your project components incrementally. Don't wait until your entire implementation is done to start the testing process. Note that it is very challenging to write JUnit tests for GUIs – we do not expect you to provide these here. However, we do expect that you will provide unit tests for the "back-end" of your code and that you will test your GUI in person

(the **Driver** and **InfantFrame** classes and associated inner classes). These back-end tests must cover the code in your remaining classes.

- Write your documentation as you go. Don't wait until the end of the implementation process to add documentation. It is often a good strategy to write your documentation **before** you begin your implementation.

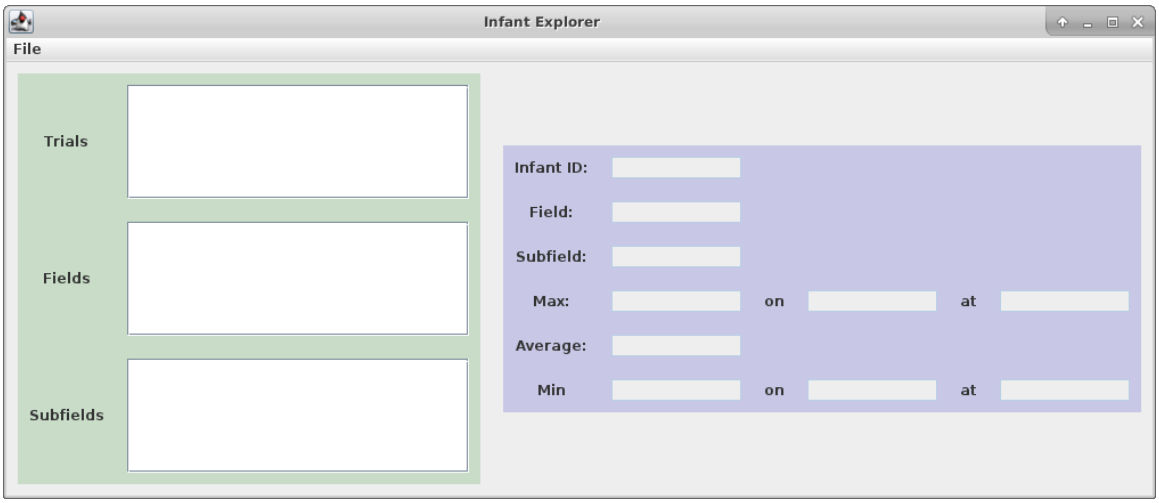
Preparation

- This description and supporting materials are available at:
  http://cs.ou.edu/~fagg/classes/cs2334/projects/project4
- We will be providing parts of our project 3 implementation on Canvas.
- In Eclipse, copy your *project3* folder to a new *project4* project.
- The data will be the same as for the last project. The data that we provided in project 3 must be located in the *data* directory. Any custom data that you use for testing must be in a different directory (e.g., *mydata*).
- Download project4.zip from the project directory. This zip file contains a partial implementation of the **InfantFrame** class. Copy this file into your *src* directory.

Example Interactions

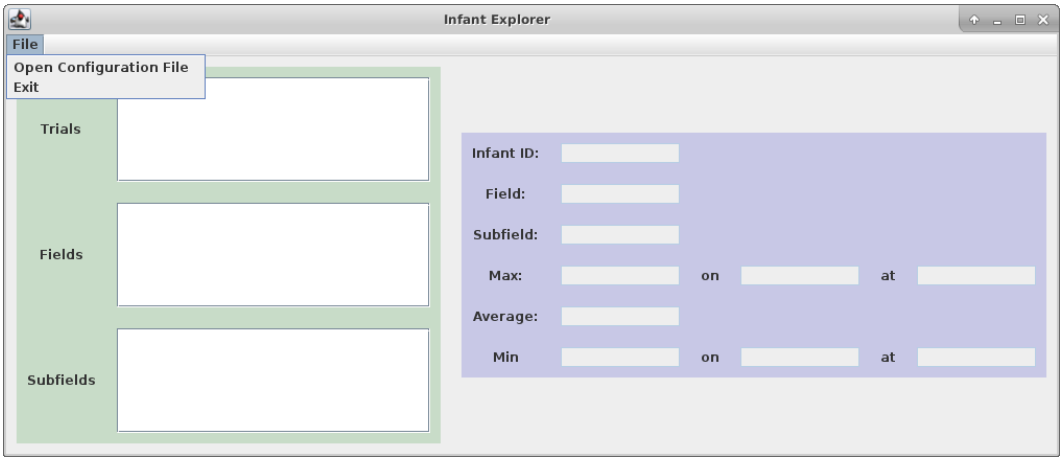
Below is a set of screen-shots for our implementation. Your implementation may have a different look. However, it must have the essential functionality, as described in the next section.

When your program starts up, it will create the GUI, but not populate it with any information:

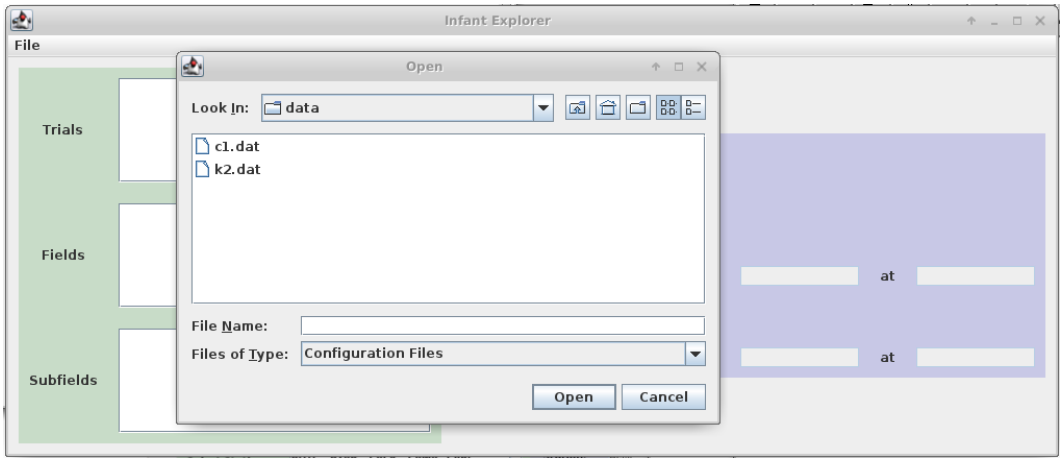


- A file menu is presented in the upper-left corner of the window.
- The green area contains three list interfaces that allow the user to select a set of trials, a field name and a subfield name. Only one field and subfield may be selected at any one time. However, any combination of trials.
- The purple area displays the infant ID, selected field, selected subfield name, and the maximum, average and minimum for the selected field, subfield and trials. For the minimum and maximum values, the week and time are also shown.

When the file menu is selected, the full menu opens:



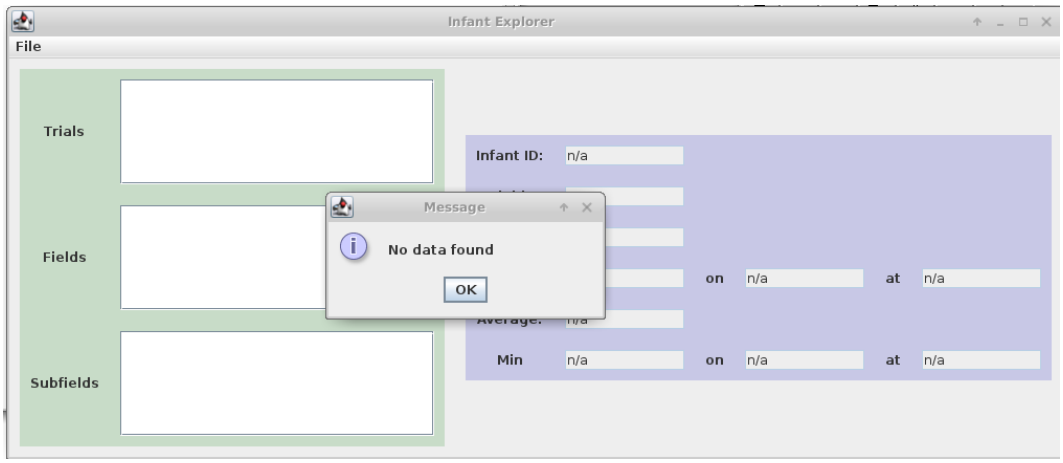
If *Exit* is selected, then the program exits (by calling System.exit(0)).
If *Open Configuration File* is selected, then a file chooser is opened:



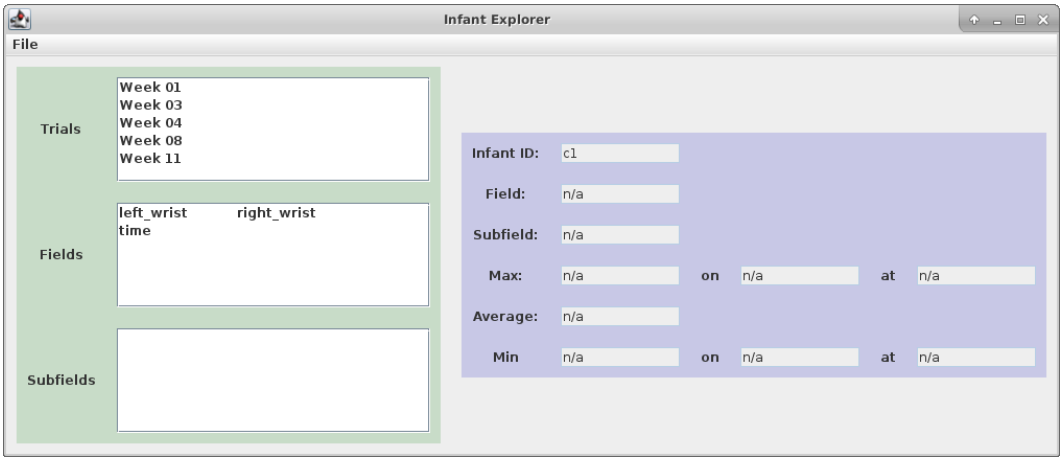
- The file chooser opens by default the *data* directory and shows only files with the *dat* extension.
- These *dat* files do not have any content; they only indicate the Infant IDs that are available in the corresponding directory.
- If the user selects one of these *dat* files, then your program will load the selected infant. While the data are loading, the cursor changes to an animated clock to indicate that your program is busy. This can be accomplished by setting the Frame's cursor to: **Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR)**.

- If the specified file does not correspond to an existing infant, your program should open an error window. This can be accomplished using **JOptionPane.showMessageDialog()**
- If an Exception is thrown while loading the file, then your program should also open an error window.

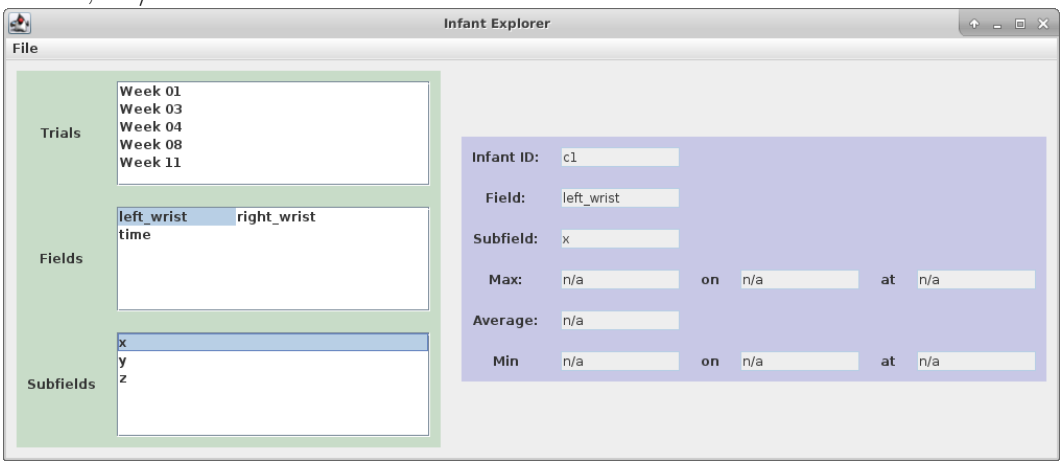
Here is one example of an error window:



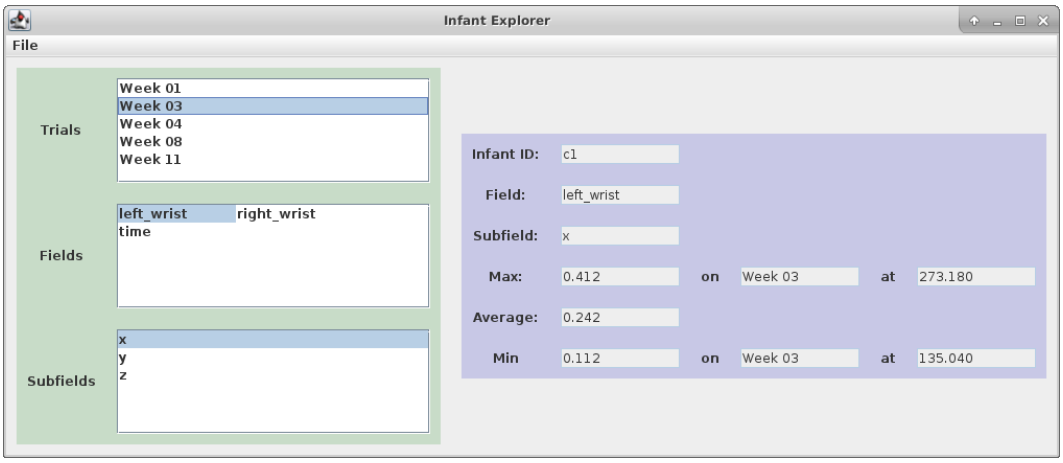
After loading, your program will display the set of available weeks and field names.



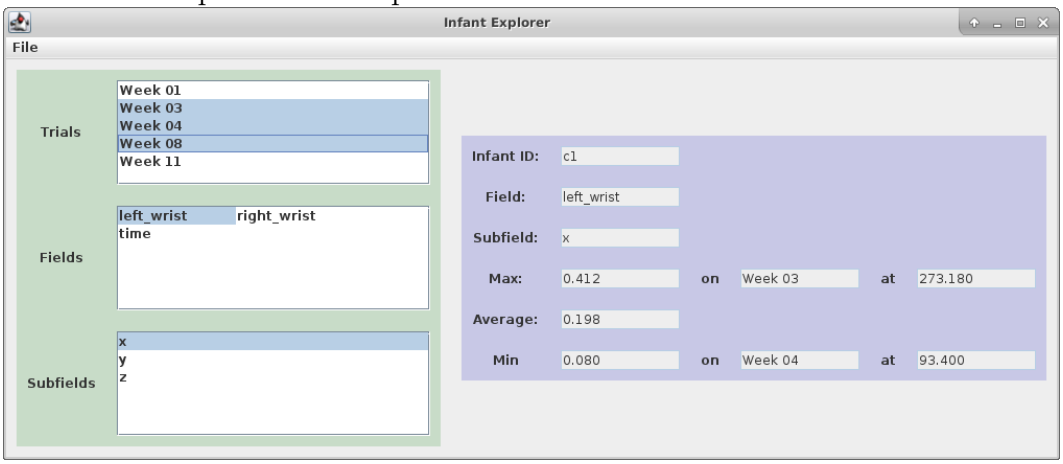
After selecting a field name, the available subfields are displayed. By default, the first subfield is selected. In the data panel, the selected field and subfield are displayed. However, since no trials are yet selected, there are no statistics that can be computed. Hence, "n/a" is shown for all statistics.



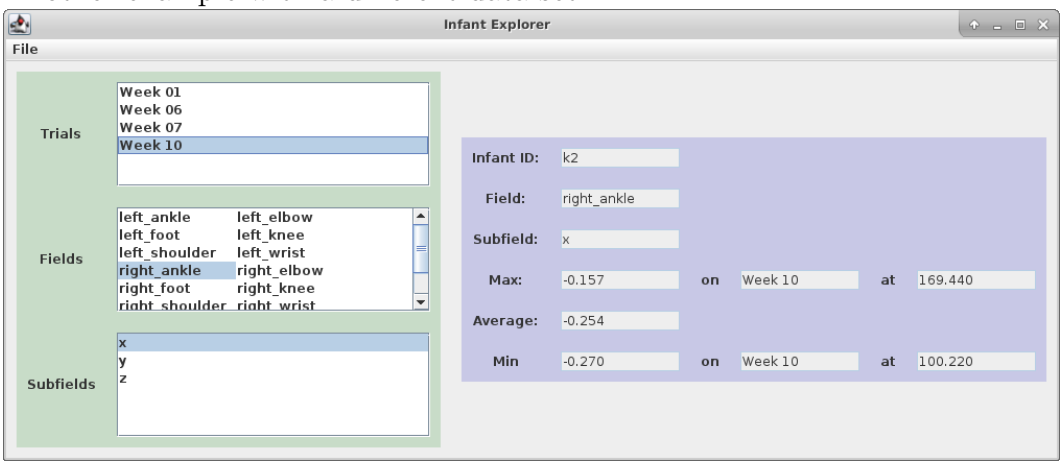
Specific trials can be selected. When this happens, then the statistics for the specified field and subfield names are shown for the selected trials. The displayed values are provided by the *toString()* methods that you have already implemented:



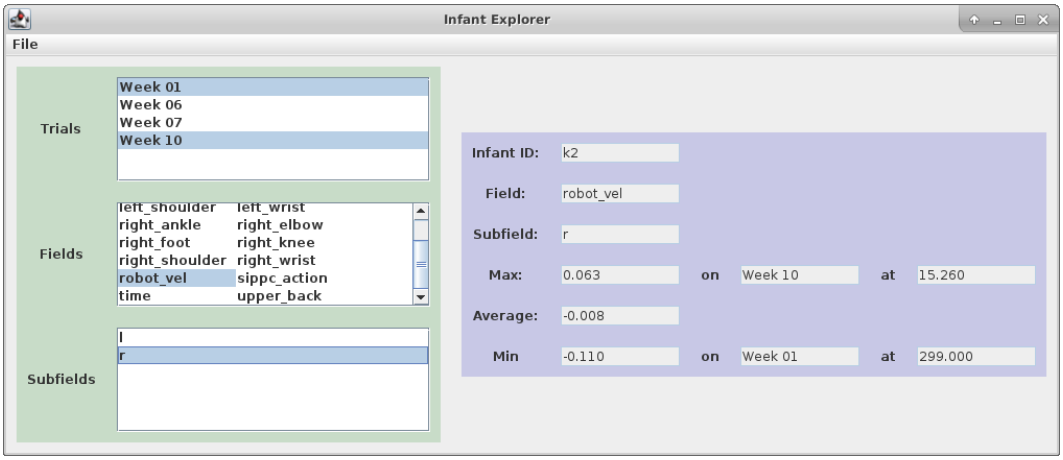
Another example with multiple trials selected:

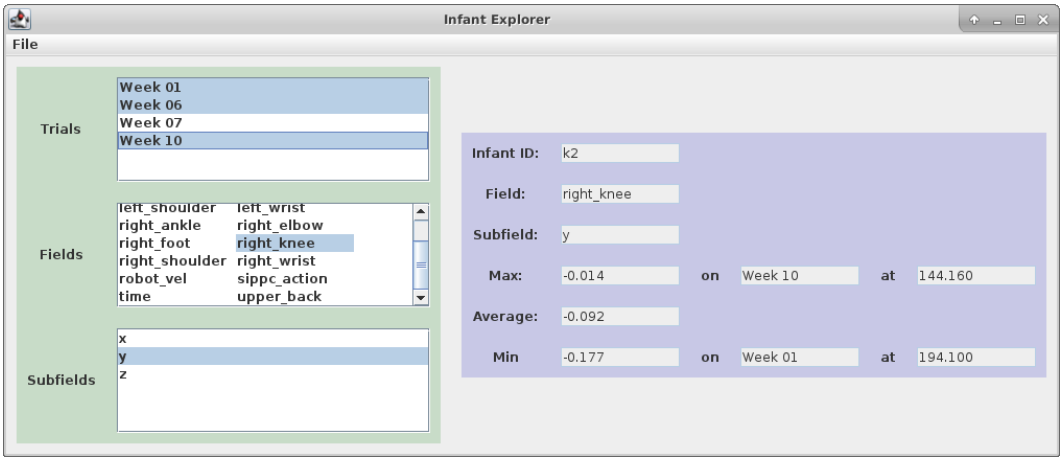


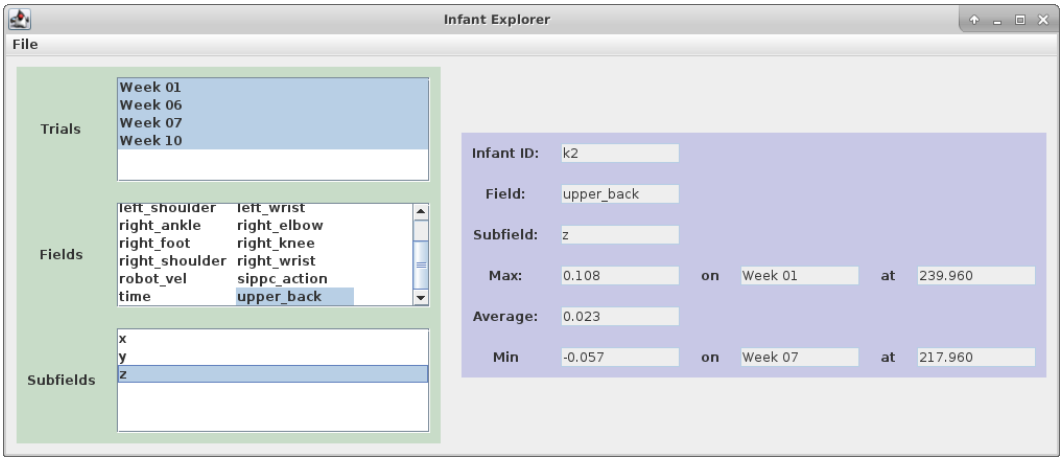
Another example with a different data set:



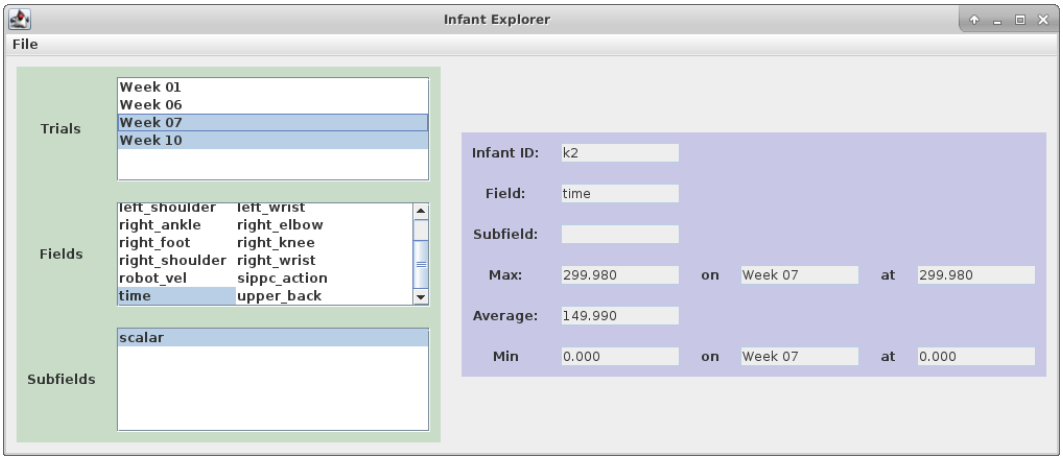
And a few other examples:




Infant Explorer
File
Trials
Week 01
Week 06
Week 07
Week 10
Fields
left_shoulder
left_wrist
right_ankle
right_elbow
right_foot
right_knee
right_shoulder
right_wrist
robot_vel
sippc_action
time
upper_back
Subfields
x
y
z
Infant ID: k2
Field: right_knee
Subfield: y
Max: -0.014 on Week 10 at 144.160
Average: -0.092
Min -0.177 on Week 01 at 194.100

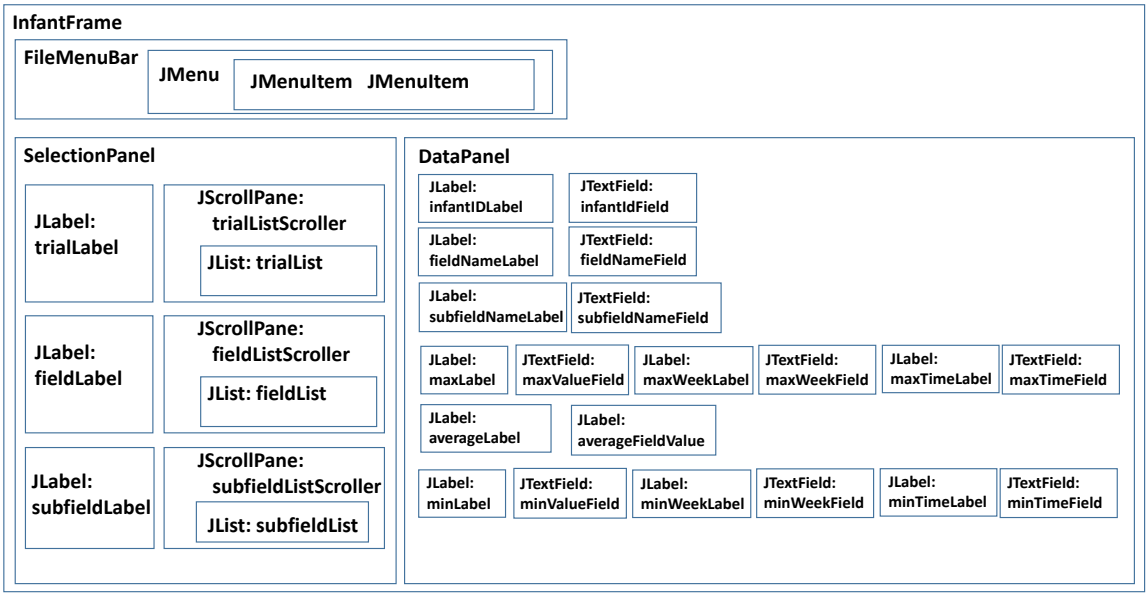

Infant Explorer
File
Trials
Week 01
Week 06
Week 07
Week 10
Fields
left_shoulder
left_wrist
right_ankle
right_elbow
right_foot
right_knee
right_shoulder
right_wrist
robot_vel
sippc_action
time
upper_back
Subfields
x
y
z
Infant ID: k2
Field: upper_back
Subfield: z
Max: 0.108 on Week 01 at 239.960
Average: 0.023
Min -0.057 on Week 07 at 217.960

When a scalar field (a field with one subfield) is selected, then the text "scalar" appears in place of the empty string. Note that this must be translated into the empty string within the statistics computation component of the code.



GUI Layout

Below is a sketch of our GUI layout. Here, we are describing the key GUI components and their approximate layout. Implicit in the way we have drawn things is also a *containment* relationship. Some of the relevant instance variables are also listed.



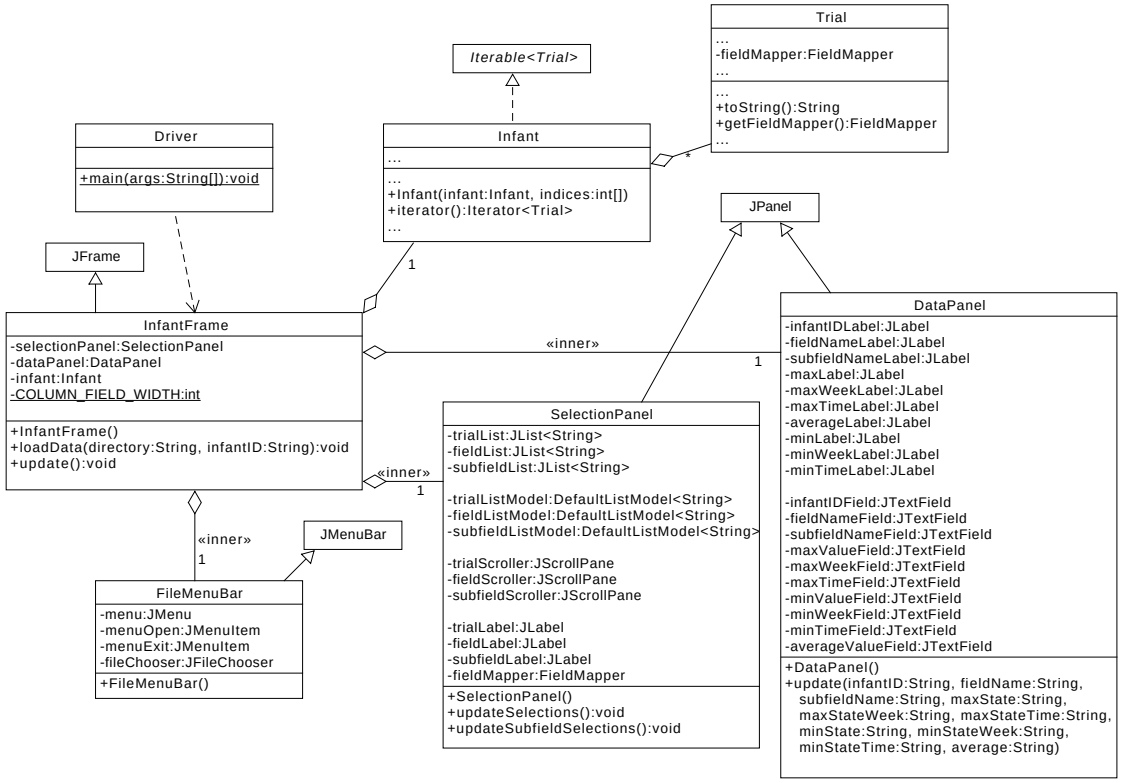
The different components in this layout are as follows:

- The **InfantFrame** contains three main components: a **FileMenuBar**, a **SelectionPanel** and a **DataPanel**.
- The **FileMenuBar** contains a single **JMenu**, which, in turn, contains two **JMenuItems**. The first menu item is used for opening files; the other is for exiting the program.
- The **SelectionPanel** contains a grid of sub-components: the rows correspond to the trial, field name and subfield name. The first column contains the labels, while the second column contains a set of **JLists** that we will use for selection. Note that each **JList** is contained within a **JScrollPane**. These scroll panes allow us to have a **JList** than will always fit within the allotted space. Should a list be too large, the **JScrollPane** will automatically show a scroll bar on the right hand side of the list.

- The **DataPanel** presents information according to what the user has selected in a grid format (though the figure above does not show the components in a grid). Labels are placed in odd columns in this grid, while data are presented in **JTextFields** in even columns.

UML Design

You will adopt your implementation from project 3 with minimal changes. Below are the new classes that you will be implementing/modifying for this project. In addition, changes to existing classes are highlighted.



Class Design Outline

Here are the key changes to your project 3 code:

- Add a new constructor to your **Infant** class that takes as input an existing

infant and an array of indices. This new **Infant** will have the same *infantID*, but will have a subset of the original **Infant's Trials**. This subset is defined by the array of indices. Any illegal indices should be ignored. Note: **do not clone the underlying trials**; instead, this new infant should reference the trials in the original infant.

- The **Infant** class now implements **Iterable<Trial>**.
- The **Trial** class now implements *toString()*. Here is an example of the required format:

```
"Week 03"
```

  This class also now explicitly stores a copy of the **FieldMapper** as an instance variable, which is returned by the *getFieldMapper()* method.

- Add JUnit tests for the above changes.

Below are the implementation notes for our Graphical User Interface. Note that any method called directly or indirectly by the graphics subsystem can be called at a time when the graphical Components are in a state in transition. For example, when a **DefaultListModel** is updated with a new list of items, the list is typically first *cleared* and then the new items are *added* one-by-one. It is possible for the associated listeners to be called in the middle of this process. It is therefore important these listeners and the methods that they call are robust to situations where the model may not be in its final state (e.g., a model could be empty). You will need to include code that detects these types of situations.

- For this project, we are using **GridBagLayout** as the layout manager for our frames and panels.
- **InfantFrame**: this class is-a **JFrame** and is the primary window of the interface.
  - Complete the implementation of the constructor
  - Complete the implementation of *loadData()*. This method creates a new **Infant** and then causes the *selectionPanel* to update with the new trial/-field/subfield options. Note that this method is declared as being *synchronized*. While this keyword is beyond the scope of this class, you should

leave it in place. In short, this keyword ensures that only one thread will call this method or the *update()* at any instant in time.

  - Complete the implementation of *update()*. This method extracts selection information from the selection panel, computes the statistics using the selected values, and informs the data panel that it needs to refresh its display.

- **FileMenuBar** is an inner class of **InfantFrame** that is-a **JMenuBar**.
  - Complete the menu creation process
  - Complete the implementation of the open menu listener

- **SelectionPanel** is an inner class that is-a **JPanel** that presents the elements through which the user will select the trials, field name and subfield name. This class contains a **JList** for each selection type.
  - In the constructor, complete the creation of the **JLists**. Each has its own **DefaultListModel** and **JScrollPane**
  - In the constructor, implement the layout of the components and the associated listeners
  - Complete the implementation of *updateSelections()*. This method is called any time the entire selection panel must be redrawn with new data from an **Infant**
  - Complete the implementation of *updateSubfieldSelections()*. This method is called any time the subfield selection section of the panel must be redrawn with new information.

- **DataPanel** is an inner class that is-a **JPanel** that displays the selected information and the associated statistics.
  - Complete the creation of the **JTextFields**
  - Implement the layout of the components
  - Complete the implementation of the **update()** method.

Notes

- Build your GUI incrementally. Focus on the "look and feel" of your GUI before you add functionality. Then, add functionality one piece at a time.

- The use of multiple classes to represent the GUI gives us the opportunity to logically partition the problem into smaller pieces. Because these pieces are largely independent of one-another, this allows us to keep the complexity down.

- By setting up all of these classes (but one) as **inner classes** of a larger frame class, this allows us to easily handle the dependencies between the various GUI classes. In particular, inner classes have the ability to access variables and methods of the outer class, even when they are private. For example, an inner class can refer to the outer class instance using:

  InfantFrame.this

  and, hence, access variables and call methods using:

  InfantFrame.this.selectionPanel

  or

  InfantFrame.this.setCursor()

- **JMenuItems** have **ActionListener**s attached to them to implement the functionality of selecting a menu item.

- We create a reference to your data directory this way:

  new File("./data")

- **JLists** present a list of items to the user and allow the user to select one (or possibly more). See the reference section below for a useful link that talks about many options.

  When the items in the list are known *a priori* and won't change, the simple way to create a **JList** is to hand it an array of Strings – one for each item. You can then tell the **JList** to select the first item in the list automatically:

  setSelectedIndex(0)

  A **SelectionListener** can then be added to respond to any change in what is selected. A change can be either the deselection of an item or the selection of an item (note that most "clicks" involve a sequence of deselection followed by selection). The currently selected element (if we assume that there is only one) can be read from the **JList** using **getSelectedValue()**.

  If a **JList** allows you to select more than one item, you can access the list of indices (in the presented list) using **getSelectedIndices()**.

When the items are not known *a priori* or will change with time (as is the case for all of our **JLists**), which we won't know until we have loaded the data), we must use some form of **ListModel**. The **DefaultListModel** class is a **List** to which items can be added or cleared from. Every time this list changes, the **DefaultListModel** will automatically inform the **JList** that the list has changed, which, in turn, will cause the display to be updated. To attach a **ListModel** to a **JList**, you include a reference to the model in your call to the **JList** constructor (there is an example in the code).

You will need to look carefully at the **JList** settings in order to get each of the **JLists** to display information in the correct way.

- Each **JList** is placed inside of a **JScrollPane**. This tells the GUI to use a fixed size pane to present the information, but to provide scroll bars if the information is too large to display in the fixed area. If the information fits, then the scroll bar is automatically hidden.
- **JTextField**s, by default, are about receiving text input from a user. However, they can be used as output-only components by setting their *editable* property to *false*. They are convenient for this because we can define their width in terms of the number of characters that they should hold. And, the text presented in the field can be selected and copied by a user through the use of mouse operations.
- Real-time grading will be much slower with this project and the next. Please plan accordingly.
- Depending on server load, real-time grading of submissions may be halted at any time. Our priority is to let groups submit solutions in a timely fashion. If we do halt online grading, we will attempt to reenable it at a time where the load on the server is low. This means that you should not expect feedback on solutions that are submitted near to the deadline.

Final Steps

1. Generate Javadoc using Eclipse for all of your classes.
2. Open the *project4/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that all of your classes are listed and that all of your documented methods have the necessary documentation.

Submission Instructions

- All required components (source code and compiled documentation) are due at 1:29:00 pm on Monday, November 13.
- Submit your project to Web-Cat using one of the two procedures documented in the Lab 1 specification.

Grading: Code Review

All groups must attend a code review session in order to receive a grade for your project. The procedure is as follows:

- Submit your project for grading to the Web-Cat server.
- Any time following the submission, you may do the code review with the instructor or one of the TAs. For this, you have two options:

    1. Schedule a 15-minute time slot in which to do the code review. We will use Doodle to schedule these (a link will be posted on Canvas). You must attend the code review during your scheduled time. Failure to do so will leave you only with option 2 (no rescheduling of code reviews is permitted). Note that schedule code review time **may not** be used for help with a lab or a project
    2. "Walk-in" during an unscheduled office hour time. However, priority will be given to those needing assistance in the labs and project

- Both group members must be present for the code review
- During the code review, we will discuss all aspects of the rubric, including:

    1. The results of the tests that we have executed against your code
    2. The documentation that has been provided (all three levels of documentation will be examined)
    3. The implementation. Note that both group members must be able to answer questions about the entire solution that the group has produced

- If you complete your code review before the deadline, you have the option of going back to make changes and resubmitting (by the deadline). If you do this, you may need to return for another code review, as determined by the grader conducting the current code review
- The code review must be completed by Monday, November 20th to receive credit for the project.

References

- The Java API: https://docs.oracle.com/javase/8/docs/api/
- JLists: https://docs.oracle.com/javase/tutorial/uiswing/components/list.html
- JFileChooser: https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html
- Menus: https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Correctness/Testing: 40 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is a product of the fraction of **our tests** that your code passes and the fraction of **your code** that is covered by *your tests*. In other words, your submission must perform well on both metrics in order to receive a reasonable grade.

Style/Coding: 25 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the project deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation (up to 10 points)
- Missing or inappropriate inline documentation (2 points per violation; up to 10 points)
- Inappropriate choice of variable or method names (2 points per violation; up to 10 points)
- Inefficient implementation of an algorithm (minor errors: 2 points each; up to 10 points)
- Incorrect implementation of an algorithm (minor errors: 2 points each; up to 10 points)

If you do not submit compiled Javadoc for your project, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every twelve hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose five points for every twelve hours that your assignment is submitted late. For a submission to be considered *on time*, it must arrive at the server by the designated minute (and zero seconds). For a deadline of 9:00, a submission that arrives at 9:00:01 is considered late. Assignments arriving 48 hours after the deadline will receive zero credit.

After 30 submissions to Web-Cat, you will be penalized one point for every additional submission.

Web-Cat note: 24 hours before the deadline, the server will stop giving hints about any failures of your code against our unit tests. If you wish to use these hints for debugging, then you must complete your submissions 24 hours before the deadline.