CS 2334
Project 5: Java Graphics

November 13, 2017

Due: 6:00 pm on Friday, Dec 1, 2017

Introduction

In project 4, you created a graphical user interface that interacts with a user and displays statistics about infant kinematic data. This project will extend your experience into the realm of graphics. In particular, you will display the state of the infant using a skeletal representation, projected into three two-dimensional views. By changing the state in time, you will also animate this representation. Your implementation from the prior projects will serve as important components for this project.

Your final product will:

1. Allow the user to specify an infant, whose data are to be loaded,
2. Allow the user to select one of the available weeks associated with the infant,
3. Allow the user to specify a time within the week to render,
4. Render a top, side and rear view of the infant's skeleton, and
5. Animate the skeleton by rendering successive states.

Learning Objectives

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

1. Create a tree data structure for representing a kinematic tree,

2. Create a class for flexibly transforming 3D data into a 2D picture,
3. Use **JSliders** to accept input,
4. Use **Timers** to create animations, and
5. Continue to exercise good coding practices for Javadoc and for testing.

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

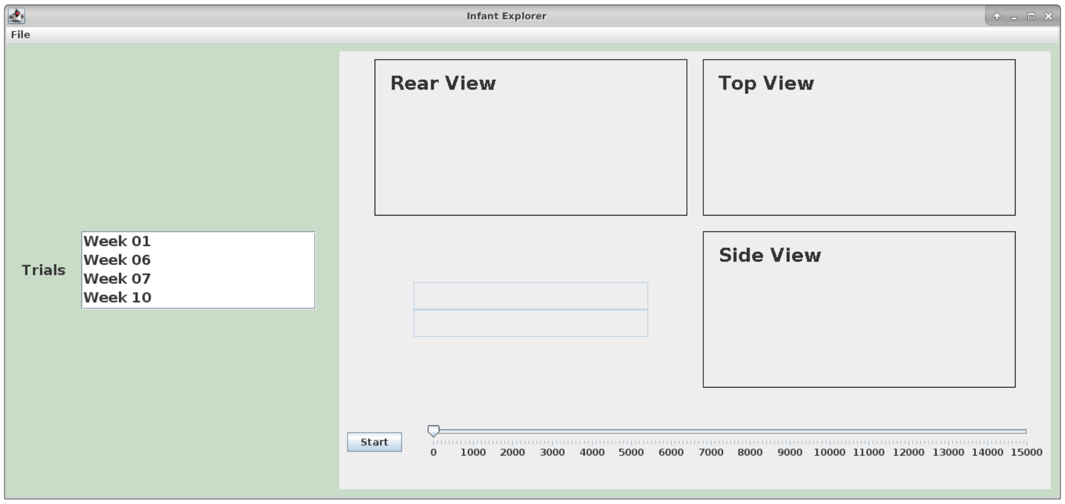
- The UML is a guide to the new classes and methods that you will implement.
- When you are implementing a class or a method, focus on just what that class/method should be doing. Try your best to put the larger problem out of your mind.
- We encourage you to work closely with your other team member, 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.
- 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

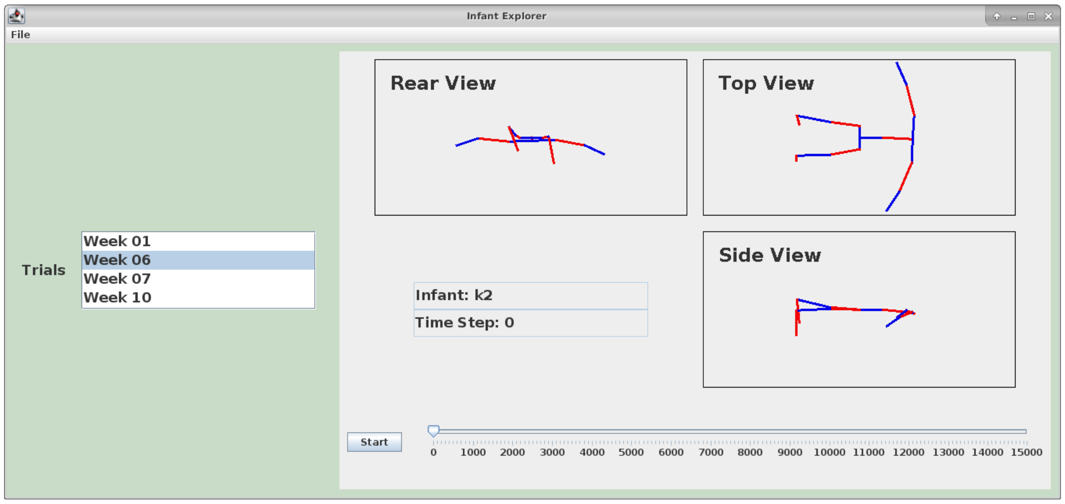
- This description and supporting materials are available at:
  `http://cs.ou.edu/~fagg/classes/cs2334/projects/project5`
- In Eclipse, copy your *project4* folder to a new *project5* project. Within this project, our data should be located in the *data* directory. Any custom data that you use for testing should be placed in the *mydata* directory.
- We provide a partial implementation of **KinematicPanel**. Place this java file in your default package.
- We will be providing a partial implementation of **InfantFrame** and its inner classes after the project 4 deadline +48 hours. Place this java file in your default package.

Example Interactions

Below is a set of screen-shots for our implementation. Your implementation will look very similar. And, the data that you display must match our implementation. Here is the initial state of the interface:



After loading infant *k2*, and selecting *Week 06*, the interface will appear as follows:



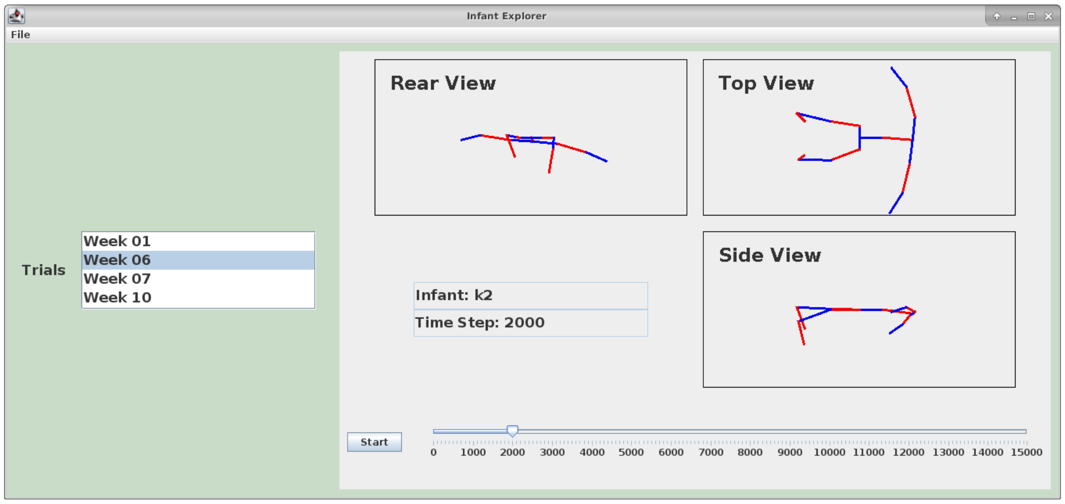
Infant: k2
Time Step: 0

In the State representation that you have developed, you represent the x/y/z position of many points on the infant body. The top view shows the locations of these points in the x/y plane. In this view, the arms are displayed to the right of the panel, with the left arm pointing toward the top of the panel. The ankles and tips of the feet are shown to the left of the panel.

The side view displays the 3D points in the x/z plane. This view also renders the arms to the right of the panel, but shows the feet pointing to the bottom of the panel.

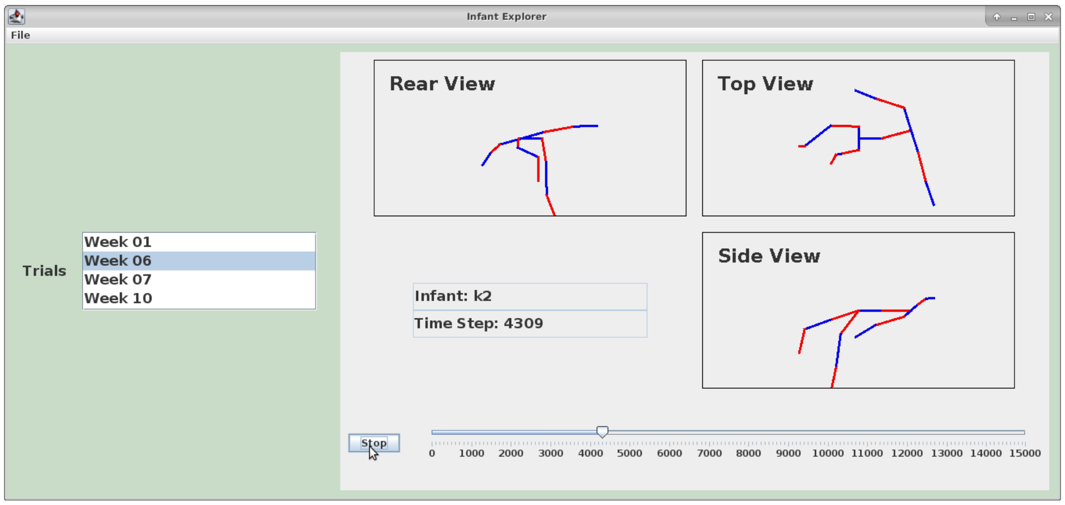
The rear view displays the 3D points in the y/z plane. This view displays the left arm and leg on the left of the panel and the right arm and leg on the right.

The time step slider can be used to select the State to be rendered (here, we are showing time step 2000):



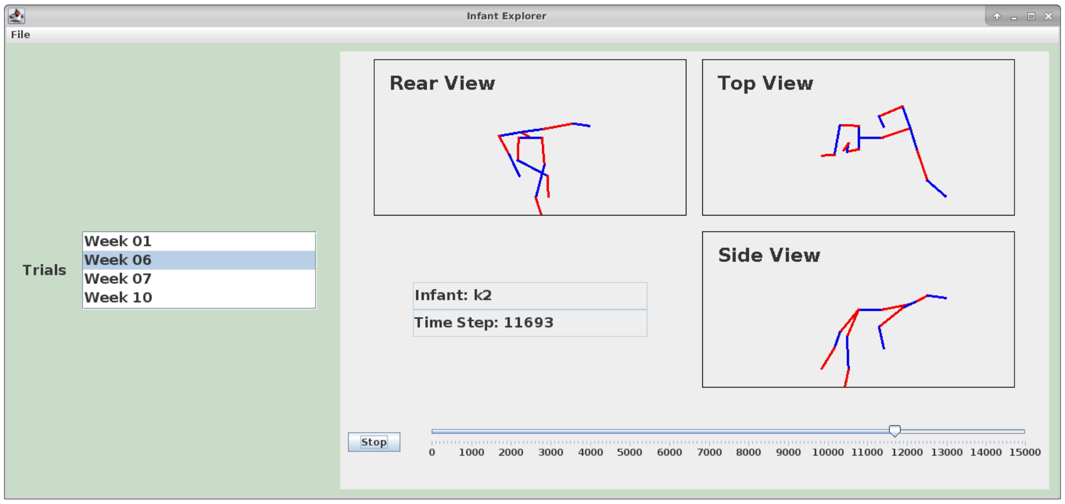
Infant: k2
Time Step: 2000

Here is time step 4309:



Infant: k2
Time Step: 4309

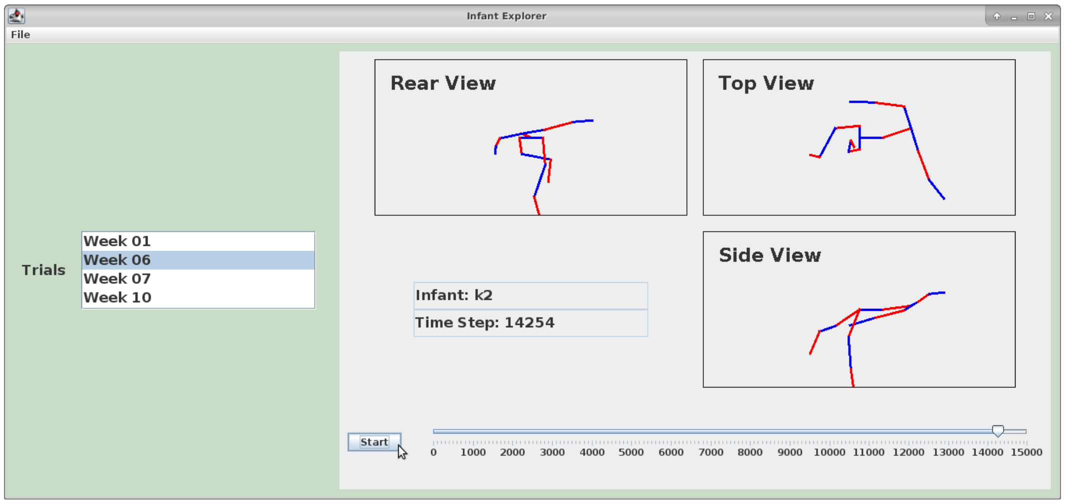
And time step 11693:



Week 01
Week 06
Week 07
Week 10

Infant: k2
Time Step: 11693

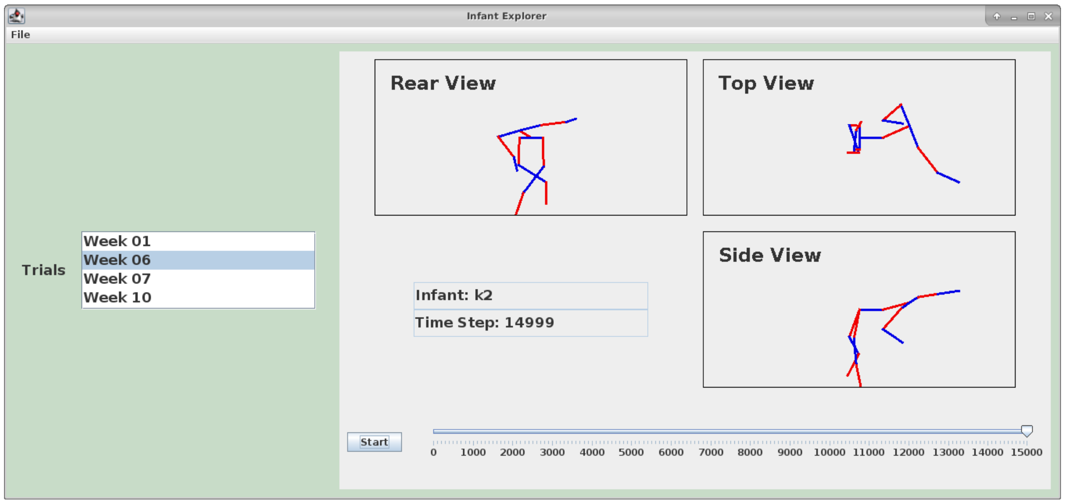
And time step 14254:



Week 01
Week 06
Week 07
Week 10

Infant: k2
Time Step: 14254

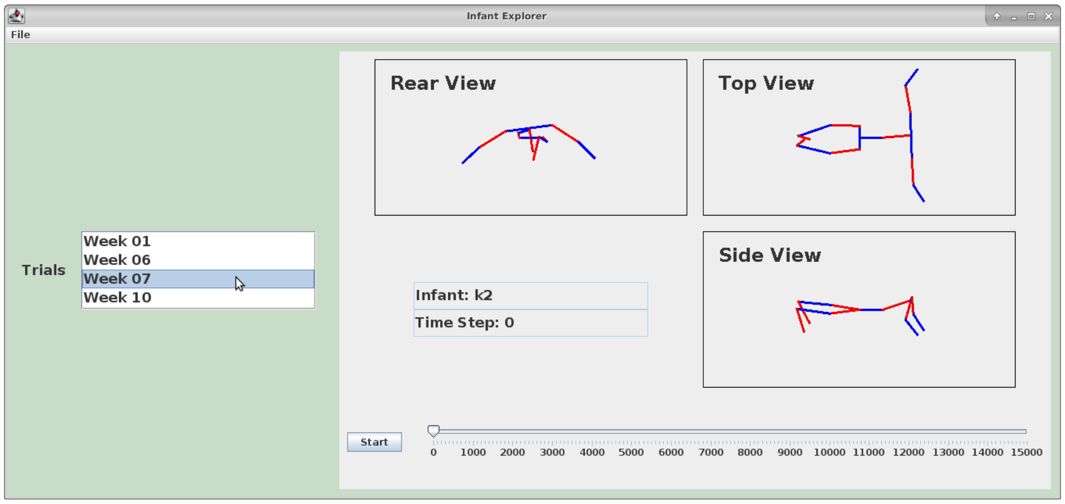
And time step 14999 (the last available step):



Infant: k2

Time Step: 14999

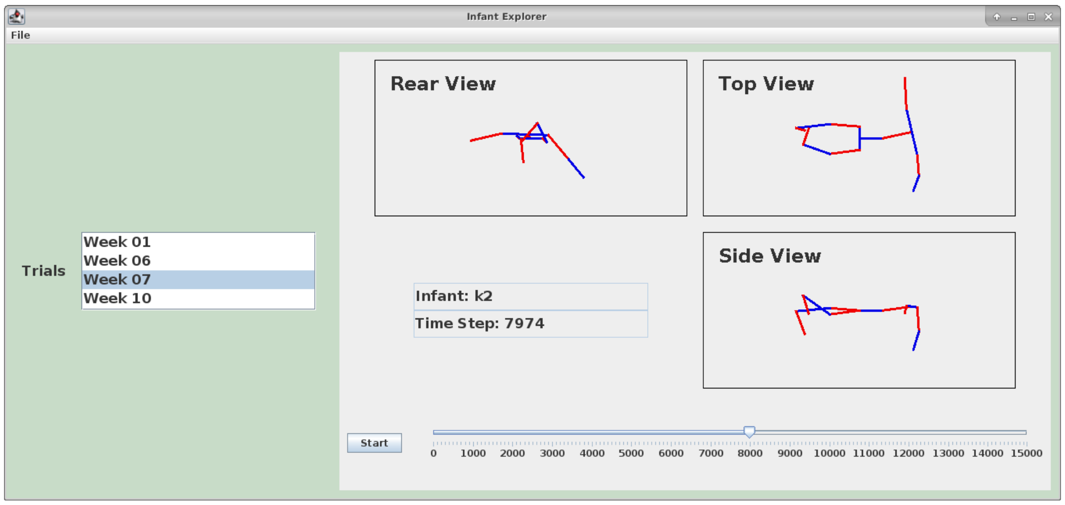
When *Week 7* is selected, the rendered trial changes and the current time changes automatically to time step 0:



Infant: k2

Time Step: 0

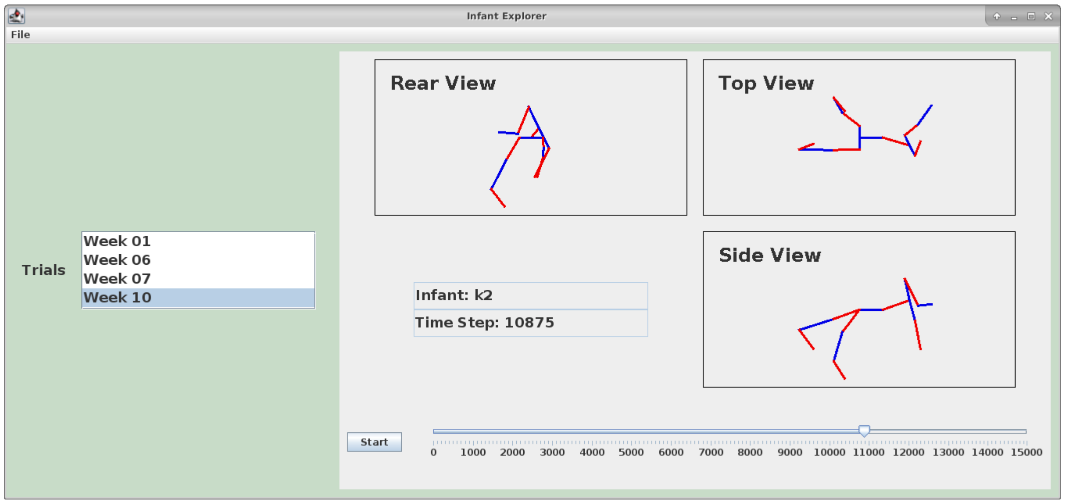
Here is time step 7974:



Infant: k2

Time Step: 7974

And here is *Week 10*, time step 10875:



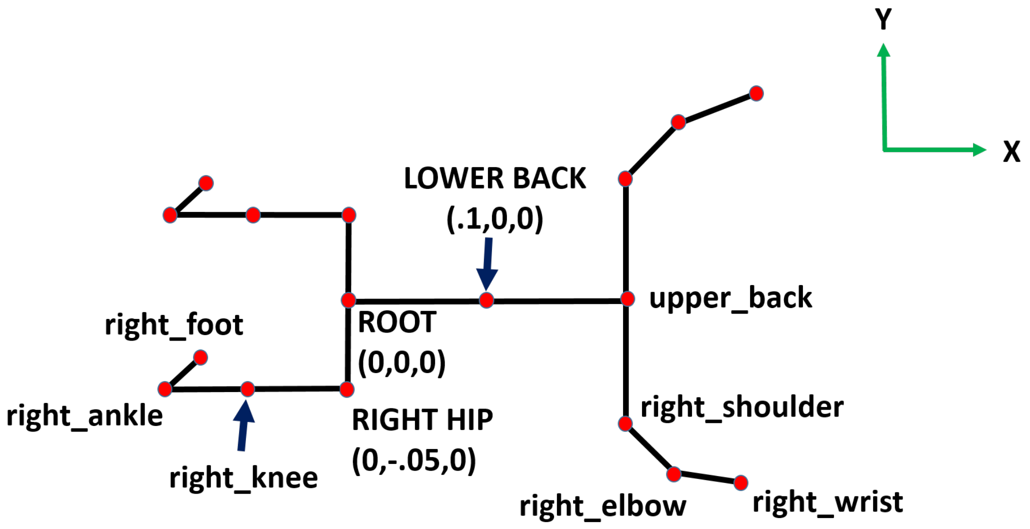
Infant: k2

Time Step: 10875

When the *Start* button is pressed, the button text switches to *Stop*, and time begins to increment automatically once every 5 milliseconds. With each change of time, the display is updated, giving us an animation of how the infant behaved during the trial. If the *Stop* button is pressed **or** time reaches the end of the trial, time ceases to increment and the button text returns to *Start*.

Representing Infant Kinematics

A *kinematic tree* represents a robot or biological body using rigid links (lines) connected by (possibly) movable joints. The figure below shows a top view of our infant kinematic model. Each of the red points correspond to specific points on the infant body. When two or more links are connected by a point, then the point is a proper joint.



The root of our kinematic tree is located at the origin of our three dimensional space ($x = 0; y = 0; z = 0$). This root point is connected to three other points through fixed relationships: the *lower back* is always located at $x = 0.1; y = 0; z = 0$, and the left and right hip socket joints are located at $x = 0; y = 0.05; z = 0$ and $x = 0; y = -0.05; z = 0$, respectively. We refer to these three points as the *children* of the root point.

The lower back is connected to one other point: the *upper_back*. Hence, the lower back has one child point. The location of this point is variable, and is captured in a single **State** object.

The upper back has two children: the *left_shoulder* and *right_shoulder*.

The right shoulder has one child: the right_elbow. This point has one child: the *right_wrist*. The right wrist has no children.

Rendering a Kinematic Tree

In this project, we will not worry about rendering our three-dimensional kinematic tree in three dimensions. Instead, your interface will present a set of simple two-dimensional projections of the three-dimensional data. For the top view of the infant, we will “throw away” the z dimension of the data and use only the x and y dimensions. Specifically, we will use the x dimension of the data to determine the x coordinate on the screen and the y dimension of the data to determine the y coordinate on the screen.

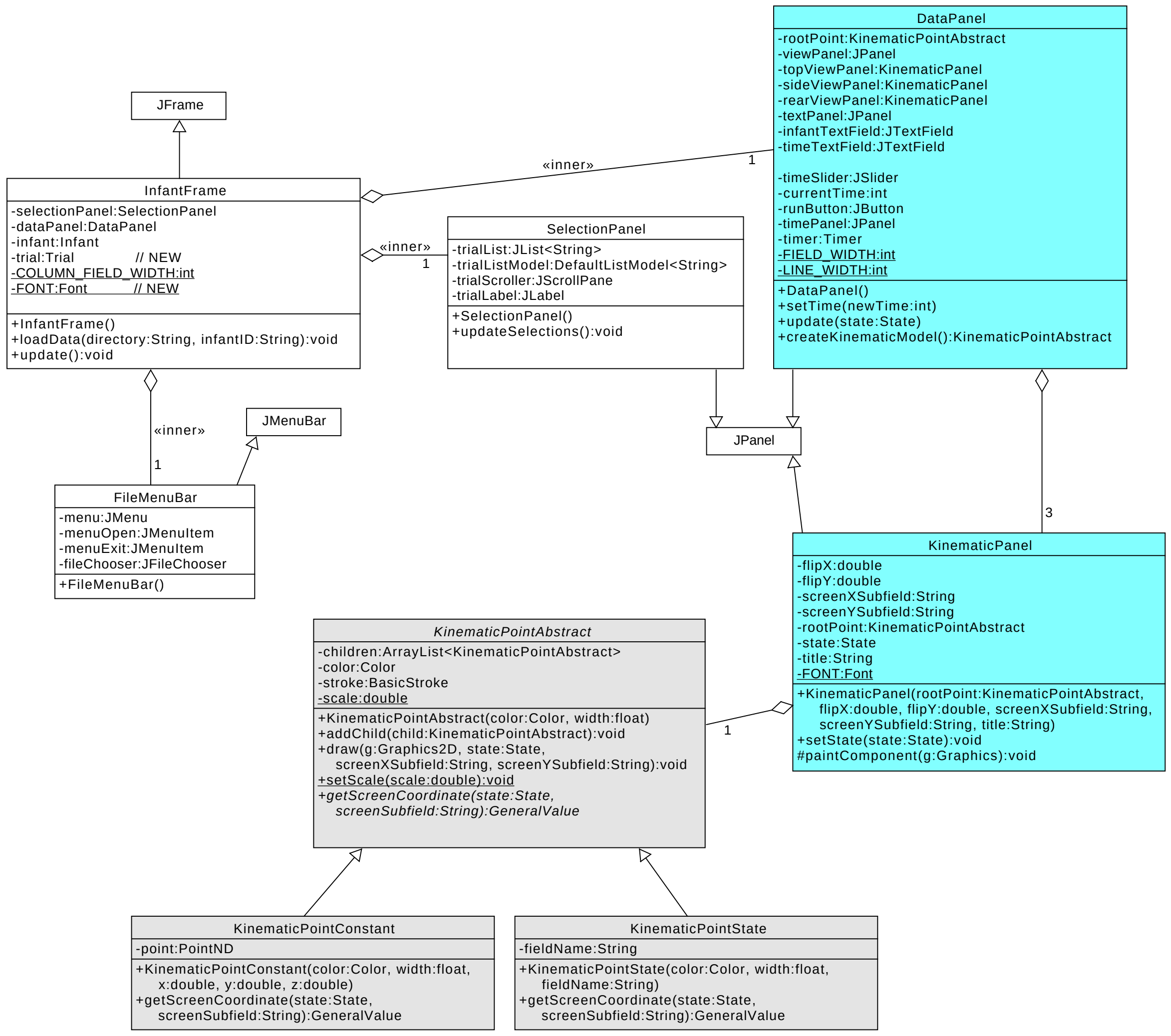
Note that in making this transformation from the three-dimensional data (which are measured using meters), we must also account for 1) the transformation from meters to pixels, and 2) we must flip the sign of the y pixel coordinate in order to account for the fact that we are moving from a right-handed coordinate frame to a left-handed one.

Rendering of a kinematic tree begins at its root point. The algorithm is as follows:

1. For the current point, extract the dimensions that correspond to the x and y screen coordinates.
2. For every child point:
   (a) Extract its dimensions that correspond to the x and y screen coordinates.
   (b) If all four of the values are valid (two values each from this point and the child), then transform the values into pixel coordinates and draw a line between the two points. For our work, we will use a **BasicStroke** to draw the line segment.
   (c) Recursively draw the child point (Step #1).

UML Design

Below is the UML diagram that emphasizes the classes that have changed since project 4.



DataPanel
-rootPoint:KinematicPointAbstract
-viewPanel:JPanel
-topViewPanel:KinematicPanel
-sideViewPanel:KinematicPanel
-rearViewPanel:KinematicPanel
-textPanel:JPanel
-infantTextField:JTextField
-timeTextField:JTextField
-timeSlider:JSlider
-currentTime:int
-runButton:JButton
-timePanel:JPanel
-timer:Timer
-FIELD_WIDTH:int
-LINE_WIDTH:int
+DataPanel()
+setTime(newTime:int)
+update(state:State)
+createKinematicModel():KinematicPointAbstract
JFrame
«inner»
1
InfantFrame
-selectionPanel:SelectionPanel
-dataPanel:DataPanel
-infant:Infant
-trial:Trial // NEW
-COLUMN_FIELD_WIDTH:int
-FONT:Font // NEW
+InfantFrame()
+loadData(directory:String, infantID:String):void
+update():void
«inner»
1
SelectionPanel
-trialList:JList<String>
-trialListModel:DefaultListModel<String>
-trialScroller:JScrollPane
-trialLabel:JLabel
+SelectionPanel()
+updateSelections():void
«inner»
1
JMenuBar
JPanel
3
FileMenuBar
-menu:JMenu
-menuOpen:JMenuItem
-menuExit:JMenuItem
-fileChooser:JFileChooser
+FileMenuBar()
KinematicPanel
-flipX:double
-flipY:double
-screenXSubfield:String
-screenYSubfield:String
-rootPoint:KinematicPointAbstract
-state:State
-title:String
-FONT:Font
+KinematicPanel(rootPoint:KinematicPointAbstract, flipX:double, flipY:double, screenXSubfield:String, screenYSubfield:String, title:String)
+setState(state:State):void
#paintComponent(g:Graphics):void
KinematicPointAbstract
-children:ArrayList<KinematicPointAbstract>
-color:Color
-stroke:BasicStroke
-scale:double
+KinematicPointAbstract(color:Color, width:float)
+addChild(child:KinematicPointAbstract):void
+draw(g:Graphics2D, state:State, screenXSubfield:String, screenYSubfield:String):void
+setScale(scale:double):void
+getScreenCoordinate(state:State, screenSubfield:String):GeneralValue
1
KinematicPointConstant
-point:PointND
+KinematicPointConstant(color:Color, width:float, x:double, y:double, z:double)
+getScreenCoordinate(state:State, screenSubfield:String):GeneralValue
KinematicPointState
-fieldName:String
+KinematicPointState(color:Color, width:float, fieldName:String)
+getScreenCoordinate(state:State, screenSubfield:String):GeneralValue

The colors denote classes that: we provide in their entirety (*white*), we provide partial implementations for (*blue*) and that you implement yourself (*gray*). As the **InfantFrame** class and its inner classes will not be made available until after the project 4 deadline + 48 hours, we suggest that you start your implementation with the **KinematicPoint*** classes.

Not shown are the "back-end" classes that you developed for project 3 (and made slight modifications for project 4).

Class Design Outline

KinematicPointAbstract is an abstract class that provides a majority of the machinery for rendering two-dimensional projections of three-dimensional kinematic trees. Each instance of this class is defined by a list of *children*, and a *color* and a *stroke* to be used for drawing line segments from the point to its children. There is also a *scale* class variable that defines the relationship between meters (infant coordinates) and screen pixel size. The key methods are:

- *addChild()* adds a new **KinematicPointAbstract** object as a child object
- *draw()* takes as input a **Graphics2D** context, a **State** to render and the *subfield names* that are mapped to the screen x and y dimensions. The algorithm for rendering by this method is described above
- The abstract *getScreenCoordinate()* method requires implementing classes to provide a method that will return the **GeneralValue** that corresponds to the specified *subfield name*

KinematicPointConstant represents a point whose location does not change with time (i.e., it is independent of **State**). The constructor takes as input the constant values for the x, y and z dimensions.

KinematicPointState represents a point whose location is defined by a specific field of a **State** object.

KinematicPanel is a **JPanel** that renders a single view of a kinematic tree. The properties are:

- *screenXsubfield* and *screenYsubfield* are **Strings** that define the subfields to be used for the screen x direction and the screen y direction, respectively
- *flipX* and *flipY* can take on the values $+1$ or -1, and indicate whether positive values in infant coordinates correspond to positive ($+1$) or negative (-1) values in screen coordinates
- *rootPoint* is the root of the kinematic tree to be rendered by the panel
- *state* is the **State** object that is used for rendering
- *title* is the title of the panel that is placed in the upper left corner of the panel

- *font* is the font to be used to render the title

The key methods are:

- *setState()* sets the state to be rendered and then forces a repaint of the panel
- *paintComponent()* renders the entire panel, including the title and the kinematic tree. Note that in the partial implementation that we provide, the origin of the kinematic panel is translated from the upper left corner to the center of the panel and the directions for drawing are flipped according to *flipX* and *flipY*

InfantFrame.DataPanel displays information based on the selected *trial* and the *currentTime*. The key properties are:

- *rootPoint* is the root of the kinematic tree
- *viewPanel* is the panel that contains all kinematic views
- **ViewPanel* are specific **KinematicPanels**
- *textPanel* is a panel that contains information about the loaded infant ID (*infantTextField*) and current time (*timeTextField*)
- *timePanel* is a panel that contains the start/stop button (*runButton*) and a slider for specifying the current time (*timeSlider*). We will also use the slider to show the current time
- *currentTime* is an int that represents the current time step that is being rendered
- *timer* is a **Timer** object that drives the animation of the kinematic figure by regularly incrementing the *currentTime*
- *FIELD_WIDTH* is the width of the text fields
- *LINE_WIDTH* is the width of the line segments in the kinematic panels.

The key methods/pieces that you must implement are:

- *runButton* action listener. In response to a button press and the current start/stop state of the button, change the state of the button and the *timer*

- *timer* action listener. Sets the *currentTime* to one value higher than its current value
- *setTime()* takes as input a potentially new current time. If a trial exists and the new time falls within the valid range, then change the *currentTime* and cause the *dataPanel* to update with the new state. If the new time falls outside of the valid range, then ensure that the *timer* is not running and that the button is in the appropriate state
- *update()* takes as input a new **State** and forces all of the components of the *dataPanel* to redraw with the new state information
- *createKinematicModel()* creates the full kinematic model for the infant

Testing

- You must continue to test all of your “back-end” classes from project 4, but no new testing of the back-end is required.
- Because the new classes that you will implement involve graphics, you are not required to test these classes using JUnit tests. However, you must test these classes by interacting with your GUI.
- We will test both your front-end and back-end code automatically on Web-Cat.

Notes

- Spend some time with the Java API for the following classes: **Graphics2D**, **Line2D.Double**, **BasicStroke**, and **Timer**.

Final Steps

1. Generate Javadoc using Eclipse for all of your classes.
2. Open the *project5/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 6:00:00 pm on Friday, December 1st.
- 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 Friday, December 8th to receive credit for the project.

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, the fraction of **your code** that is covered by *your tests* and the fraction of **your tests** that your code passes. In other words, your submission must perform well on all three metrics in order to receive a reasonable grade.

Note that Web-Cat only provides an estimate of the correctness score. If your code is throwing Exceptions, then the correctness score can be overestimated.

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.