Appendix E

Object-Orientated Data Structure

The following code was designed to allow effective and adaptive storage of eye-tracking information from multiple participant data.

Package: definingroi

topData.java

topData is the highest data abstraction, allowing manipulation of eye-tracking data from multiple experiments (see Figure 3.5). If uniquely defined, experiment objects can be added to an array list named 'experimentList'. If an experiment object has duplicated experiment name, determined as the sortKey, then data will be added. topData contains the high-level save function allowing all eye-tracking data to be exported to a text based file, which is subsequently used to ensure use of consistent data sets.

```
package definingroi;

import java.util.*;
import javax.swing.*;
import java.io.*;
import java.awt.*;

class topData {

/* topData is the highest object abstraction and allows
the input, manipulation and saving of data from multiple
experiments.

The experimentList array list may contain multiple
experiment objects, allowing the data from multiple
experiments to be manipulated and stored.
*/

private ArrayList experimentList = new ArrayList();

public void add(experiment exp) {

/* Add function adds a new experiment to the
experimentList array list, assuming that the experiment
name is unique */

  if (checkExperiment(exp) != true){
// if experiment is unque
    if (experimentList.isEmpty() == true) {
/* if no experiment objects are currently in the
experimentList array list, then add */
    experimentList.add(exp);}
    else if (experimentList.size() < 2) {
// single item
      experiment exp1 =
(experiment)experimentList.get(0);
/*create temporary experiment object from arraylist
object. The desired position of the new object is then
determined.*/
      if
(exp.sortKey().compareToIgnoreCase(exp1.sortKey()) <
0) {
        experimentList.add(0, exp);
      }
      else {
        experimentList.add(exp);
      }

    }
    else {
/* If there are more than one object then call the
compare function, which iterates through the list to
identify the correct new position.*/
      compare(exp);
    }
  }
  else {
    JOptionPane.showMessageDialog(null,"The overall
array definition is
invalid.","ERROR",JOptionPane.WARNING_MESSA
GE);
// in case object is not an experiment object
  }
 }

 public boolean checkExperiment(experiment exp) {
/* iterates through the arraylist to determine whether the
sortkey, which is definined as the experiment name is
duplicated. */
  for(int i=0; i<experimentList.size();i++) {
    experiment experimentInArray =
(experiment)experimentList.get(i);
    if ("" + exp.getExperimentName() == null) {
    return true;}
// true is no description given exists
    else if
(exp.sortKey().compareToIgnoreCase(experimentInArra
y.sortKey()) < 0)  {
      return true;
// true is duplicate exists
    }
  }
  return false;
 }

 public void compare (experiment exp) {
/* when adding an experiment to a populated arraylist,
compare iterates through the arraylist to identify the
correct location of the new experiment object. */
```

```
  int iterateVariable = 0;
  int nextExperiment = iterateVariable +1;
  int lastExperiment = experimentList.size()-1;
  boolean insert = false;
  while (iterateVariable < experimentList.size() &&
!insert) {
/* for as many objects as the number that exists in the
arraylist , whilst the object has not been added - create
three temporary experiment objects relating to the object
being tested and those either side in the array */
    experiment exp1 =
(experiment)experimentList.get(iterateVariable);
    experiment exp2 =
(experiment)experimentList.get(nextExperiment);
    experiment exp3 =
(experiment)experimentList.get(lastExperiment);
/* if position is 0 and new experiment object sortkey is
less then add as first object in the array */
    if ((iterateVariable == 0) &&
(exp.sortKey().compareToIgnoreCase(exp1.sortKey()) <
0)) {
      experimentList.add(0,exp);
      insert = true;
    }
/* if new experiment object sortKey is less than lesser
object, add in lessor position*/
    else if
(exp.sortKey().compareToIgnoreCase(exp1.sortKey()) <
0) {
      experimentList.add(iterateVariable, exp);
      insert = true;
   }
    else if
/* if the new experiment sortKey is the same as current
value (as in the case of frame rates) then add at the
current position */
(exp.sortKey().compareToIgnoreCase(exp1.sortKey())
== 0) {
      experimentList.add(iterateVariable, exp);
      insert = true;
    }
    else if
((exp.sortKey().compareToIgnoreCase(exp1.sortKey()) >
0) && (iterateVariable == experimentList.size())) {
      experimentList.add(exp);
/* if sortKey of new experiment object is greater than
the sortkey of objects in the arraylist then add at the end
of arrayList*/
      insert = true;
    }
    else if
((exp.sortKey().compareToIgnoreCase(exp1.sortKey()) >
0) &&
(exp.sortKey().compareToIgnoreCase(exp2.sortKey())
<= 0)) {
/* if new experiment object is equal or less than current
position sortkey, then add at position currently held by
nextExperiment*/
      experimentList.add(nextExperiment, exp);
      insert = true;
    }
/*if new experiment object sortKey is greater than
nextExperiment object sortKey then add after
nextExperiment object*/
    else if
(exp.sortKey().compareToIgnoreCase(exp3.sortKey()) >
0) {
      experimentList.add(exp);
      insert = true;
    }
    else iterateVariable++;
//iterates current position in arraylist
  }
 }

 public int getArrayLength()
 {
  return experimentList.size();
//return the number of experiments in arraylist
 }

 public ArrayList getExperimentList()
 {
  return experimentList;
/* returns contents of arrayList, which is vital to allow
searches through the arraylist structure. */
 }

 public String list() {
/* iterates through the arraylist and returns a string
containing a description of all experiments in the
arrayList*/
  String feedback = "";
  Iterator iterator = experimentList.iterator();
  while (iterator.hasNext()) {
    experiment nextExperiment =
(experiment)iterator.next();
    feedback = feedback+" "+
nextExperiment.description();
  }
  return feedback;
 }

  public void save() {
/* defines the location of the text file in which the data is
to be saved.*/
   Frame dialogInputFrame = new Frame();
      FileDialog outputFileName = new
FileDialog(dialogInputFrame, "Defining Output File");
      outputFileName.setVisible(true);
      String selectedItem;
      do
        selectedItem = outputFileName.getDirectory() +
outputFileName.getFile();
      while(selectedItem == null);
      dialogInputFrame.dispose();

      System.out.println(selectedItem);
/* facilitates the writing of text data to the chosen file */
      PrintWriter out;
   try {
     out = new PrintWriter(new
FileWriter(selectedItem));
   }
   catch (IOException ex) {
     throw new Error(ex.toString());
   }
/*iterates through all experiments. For each experiment a
string is produced {saveString()} that describes the
content of the experiment. This is then written to the
chosen text file.*/
   Iterator experimentIterator =
experimentList.iterator();
   for (int i=0; i<experimentList.size(); i++) {
     experiment toSave =
(experiment)experimentList.get(i);
     ArrayList output = new ArrayList();
     output = toSave.saveString();
     for (int j=0; j<output.size(); j++) {
       out.print(((String)output.get(j)));
     }

   }
   out.close();
//text file closed
  }
}
```

experiment.java

```
/* experiment contains information about experiment name and description. Additionally an arraylist (videoList) contains multiple video objects relating to the specific experiment.*/

package definingroi;

import java.util.*;
import javax.swing.*;

class experiment {

  private String experimentName;
  private String experimentDescription;
// experiment name and description
  private ArrayList videoList = new ArrayList();
//videoList contains related video information

  public experiment(String expName, String
expDescription)
  {
// assigns values when an experiment object is created.
    experimentName = expName;
    experimentDescription = expDescription;
  }

  public void add(video vid) {
/* Add function adds a new video to the videoList array list, assuming that the video name is unique */
    if (checkVideo(vid) != true){
//if video is unique
      if (videoList.isEmpty() == true) {
/* if no vido objects are currently in the videoList array list, then add */
      videoList.add(vid);}
      else if (videoList.size() < 2) {
// single item
        video vid1 = (video)videoList.get(0);
/*create temporary video object from arraylist object. The desired position of the new object is then determined.*/
        if
(vid.sortKey().compareToIgnoreCase(vid1.sortKey()) <
0) {
          videoList.add(0, vid);
        }
        else {
          videoList.add(vid);
        }
      }
      else {
/* If there are more than one object then call the compare function, which iterates through the list to identify the correct new position.*/
        compare(vid);
      }
    }
    else {
      JOptionPane.showMessageDialog(null,"The
experiment definition is
invalid.","ERROR",JOptionPane.WARNING_MESSA
GE);
// in case object is not an video object
    }
  }

  public boolean checkadd(video vid) {
/* checks the content of the video before adding it to the arrayList */
    boolean feedback = false;
/* feedback used to identify duplicated objects */
    if (checkVideo(vid) == false) {
//if no fault is found in the video
      if (videoList.isEmpty() == true) {
//if arrayList is empty – then add
      videoList.add(vid);
      feedback = true;}
      else if (videoList.size() < 2) {
        if
(vid.sortKey().compareToIgnoreCase(vid1.sortKey()) <
0) {
          videoList.add(0, vid);
/* add new video object at position 0, if less sortKey is less than current object in list*/
          feedback = true;
        }
        else if
(vid.sortKey().compareToIgnoreCase(vid1.sortKey()) >
0){
          videoList.add(vid);
          feedback = true;
/* add new video object at end of list, if less sortKey is greater than current object in list*/
        }
      }
      else {
        compare(vid);
        feedback = true;
       // true if no duplicate exists
      }
    }
    return feedback;
  }

  public boolean checkVideo(video vid) {
/* iterates through the arraylist to determine whether the sortkey, which is definined as the video name is duplicated. */
    boolean feedback = false;
/* feedback used to identify video objects that should not be added  to the arrayList – false assumes no error */
    for(int i=0; i<videoList.size();i++) {
      video videoInArray = (video)videoList.get(i);
      if (vid.getVideoName() == null) {
        feedback = true;}
/* if the new video object has no contents then there is an error */
      else if
(vid.sortKey().compareToIgnoreCase(videoInArray.sort
Key()) == 0)  {
/* if the new video objects has a duplicated sortKey there is an error*/
        feedback = true;
      }
    }
    return feedback;
  }

  public void compare (video vid) {
/* when adding a video to a populated arraylist, compare iterates through the arraylist to identify the correct location of the new video object. */

    int iterateVariable = 0;
    int nextVideo = iterateVariable +1;
    int lastVideo = videoList.size()-1;
    boolean insert = false;
    while (iterateVariable < videoList.size() && !insert) {
/* for as many objects as the number that exists in the arraylist, whilst the object has not been added -  create three temporary video objects relating to the object being tested and those either side in the array */
      video vid1 = (video)videoList.get(iterateVariable);
      video vid2 = (video)videoList.get(nextVideo);
      video vid3 = (video)videoList.get(lastVideo);
/* if position is 0 and new video object sortkey is less then add as first object in the array */
```

```
    if ((iterateVariable == 0) &&
(vid.sortKey().compareToIgnoreCase(vid1.sortKey()) <
0)) {
      videoList.add(0,vid);
      insert = true;
    }
/* if new video object sortKey is less than lesser object,
add in lower position*/
    else if
(vid.sortKey().compareToIgnoreCase(vid1.sortKey()) <
0) {
      videoList.add(iterateVariable, vid);
      insert = true;
    }
    else if
/* if the new video sortKey is the same as current value
then add at the current position */
 (vid.sortKey().compareToIgnoreCase(vid1.sortKey())
== 0) {
      videoList.add(iterateVariable, vid);
      insert = true;
    }
    else if
((vid.sortKey().compareToIgnoreCase(vid1.sortKey()) >
0) && (iterateVariable == videoList.size())) {
      videoList.add(vid);
/* if sortKey of new video object is greater than the
sortkey of objects in the arraylist then add at the end of
arrayList*/
      insert = true;
    }
    else if
((vid.sortKey().compareToIgnoreCase(vid1.sortKey()) >
0) &&
    (vid.sortKey().compareToIgnoreCase(vid2.sortKey())
<= 0)) {
/* if new video object is equal or less than current
position sortkey, then add at position currently held by
nextVideo*/
      videoList.add(nextVideo, vid);
      insert = true;
    }
/*if new video object sortKey is greater than nextVideo
object sortKey then add after nextVideo object*/
    else if
(vid.sortKey().compareToIgnoreCase(vid3.sortKey()) >
0) {
      videoList.add(vid);
      insert = true;
    }
    else iterateVariable++;
//iterate current position in arraylist
   }
  }

  public String sortKey() {
   return experimentName;
/* defines the variable used by the experiment class as
the sortKey */
  }

  public String getExperimentDescription()
  {
   return experimentDescription;
// returns contents of string experimentDescription
  }

  public String getExperimentName()
 {
  return experimentName;
// returns contents of string experimentName
  }

  public int getArrayLength()
  {
   return videoList.size();
//return the number of videos in arraylist
  }

  public ArrayList getVideoList()
  {
   return videoList;
/* returns contents of video arrayList, which is vital to
allow searches through the arraylist structure. */
   }

  public String list() {
/* iterates through the arraylist and returns a string
containing a description of all videos in the arrayList*/
   String feedback = "";
   Iterator iterator = videoList.iterator();
   while (iterator.hasNext()) {
     video nextVideo = (video)iterator.next();
     feedback = feedback+" "+ nextVideo.description();
   }
   return feedback;
  }

  public String description() {
 /*prints one line description of current experiment
object*/
   String feedback = "Experiment:
"+experimentName+" Description:
"+experimentDescription;
   return feedback;
  }

  // TO ADD
   public ArrayList saveString() {
/* Iterates though all data associated with the specific
experiment and returns a string that reporting back all
experiment data – used in topData to save multiple sets
of experiment data to file. */

    ArrayList output = new ArrayList();
// add experimental name and description
//E TAG – see Appendix C
output.add("\ne\t"+experimentName+"\t"+experiment
Description);
   ArrayList videoSearch = new ArrayList();
   System.out.println(); //newline
   videoSearch = videoList;
   Iterator videoiterator = videoSearch.iterator();
//for all videos
   while (videoiterator.hasNext()) {
     video nextVideo = (video)videoiterator.next();
     ArrayList rateSearch = new ArrayList();
     rateSearch = nextVideo.getRateList();
/* for all videos: add videoname, frame length and
description (filename) */
//V TAG – see Appendix C
output.add("\nv\t"+nextVideo.getVideoName()+"\t"+
nextVideo.getVideoLength()+"\t"
            +nextVideo.getVideoDescription());
     Iterator rateiterator = rateSearch.iterator();
     while (rateiterator.hasNext()) {
// for video at all frame rates
       frameRate nextRate =
(frameRate)rateiterator.next();
       ArrayList frameSearch = new ArrayList();
       frameSearch = nextRate.getFrameList();
// add frame rate description
//R TAG – see Appendix C
       output.add("\nr\t"+nextRate.getFrameRate());
       Iterator frameiterator = frameSearch.iterator();
       while (frameiterator.hasNext()) {
//for all frames in the specific video
         videoFrame nextFrame =
(videoFrame)frameiterator.next();
         ArrayList coordinateSearch = new ArrayList();
         coordinateSearch =
nextFrame.getCoordinateList();
// add frame number
//F TAG – see Appendix C
output.add("\nf\t"+nextFrame.getFrameNumber());
        Iterator coordinateiterator =
coordinateSearch.iterator();
        while (coordinateiterator.hasNext()) {
// for complete data set
          xyCoordinates nextCoordinates =
(xyCoordinates)coordinateiterator.next();
/* add participant number, participant viewing order, as
well as x and y coordiate values */
//C TAG – see Appendix C
output.add("\nc\t"+nextCoordinates.getParticipant()+"
\t" + nextCoordinates.getOrder() + "\t" +
nextCoordinates.getXCoordinate() + "\t" +
nextCoordinates.getYCoordinate());
          }
        }
      }
    }
    return output;
// returns contents of the string
  }
}
```

video.java

```
/* video contains information about video name, video
description and length. Additionally an arraylist (rateList)
contains data relating to video shown at specific frame
rates.*/
package definingroi;
import java.util.*;
import javax.swing.*;

class video {
   private String videoName;
   private int videoLength;
   private String videoDescription;
// video name, length and description
   private ArrayList rateList = new ArrayList();
/* rateList contains data related to videos when shown at
a specific frame rate. */

   public video(String vidName, int vidLength, String
vidDescription)
   {
    videoName = vidName;
    videoLength = vidLength;
    videoDescription = vidDescription;
// assigns values when an video object is created.

   }

   public void add(frameRate rate) {
/* Add function adds new frame rate data to the rateList
array list, assuming that the video name is unique */
     if (checkRate(rate) != true){
//if frame rate is unique
       if (rateList.isEmpty() == true) {
/* if no video objects are currently in the videoList array
list, then add */
       rateList.add(rate);}
       else if (rateList.size() < 2) {
// single item
        frameRate rate1 = (frameRate)rateList.get(0);
/*create temporary frameRate object from arraylist
object. The desired position of the new object is then
determined.*/
        if ((rate.sortKey())<(rate1.sortKey())) {
          rateList.add(0, rate);
        }
        else {
          rateList.add(rate);
        }

       }
       else {
/* If there are more than one object then call the
compare function, which iterates through the list to
identify the correct new position.*/
        compare(rate);
       }
     }
     else {
      JOptionPane.showMessageDialog(null,"The video
object is
invalid.","ERROR",JOptionPane.WARNING_MESSA
GE);
// in case object is not an frameRate object
     }
   }

   public boolean checkadd(frameRate rate) {
/* checks the content of the frameRate object before
adding it to the arrayList */
      boolean feedback = false;
// feedback used to identify duplicated objects
      if (checkRate(rate) != true){
//if no fault is found in the framerate object
        if (rateList.isEmpty() == true) {
//if arrayList is empty – then add
        rateList.add(rate);
        feedback = true;}
        else if (rateList.size() < 2) {
         frameRate rate1 = (frameRate)rateList.get(0);
         if ((rate.sortKey())!=(rate1.sortKey())) {
          rateList.add(0, rate);
/* add new frameRate object at position 0, if less
sortKey is less than current object in list*/
          feedback = true;
         }
           else if
(rate.sortKey().compareToIgnoreCase(rate1.sortKey()) >
0){
         videoList.add(rate);
         feedback = true;
/* add new frameRate object at end of list, if less
sortKey is greater than current object in list*/
        }
        }
        else {
         compare(rate);
         feedback = true;
      // true if no duplicate exists
        }
      }
      return feedback;
   }


   public boolean checkRate(frameRate rate) {
/* iterates through the arraylist to determine whether the
sortkey, which is defined as the frame Rate is duplicated.
*/
     boolean feedback = false;
/* feedback used to identify frameRate objects that
should not be added  to the arrayList – false assumes no
error */
     for(int i=0; i<rateList.size();i++) {
      frameRate rateInArray = (frameRate)rateList.get(i);
      if ("" + rate.getFrameRate() == null) {
        feedback = true;}
/* if the new frameRate object has no contents then
there is an error */
      else if (rate.getFrameRate() ==
rateInArray.getFrameRate()) {
/* if the new frameRate object has a duplicated sortKey
there is an error*/
        feedback = true;
      }
    }
    return feedback;
  }

  public void compare (frameRate rate) {
/* when adding frameRate data to a populated arraylist,
the compare method iterates through the arraylist to
identify the correct location for the new framerate object.
*/
    int iterateVariable = 0;
    int nextRate = iterateVariable +1;
    int lastRate = rateList.size()-1;
    boolean insert = false;
    while (iterateVariable < rateList.size() && !insert) {
/* for as many objects as the number that exists in the
arraylist, whilst the object has not been added - create
three temporary framerate objects relating to the object
being tested and those either side in the array */
      frameRate rate1 =
(frameRate)rateList.get(iterateVariable);
      frameRate rate2 =
(frameRate)rateList.get(nextRate);
      frameRate rate3 = (frameRate)rateList.get(lastRate);
/* if current position is 0 and new frameRate object
sortkey is less then add as first object in the array */
      if ((iterateVariable == 0) &&
(rate.sortKey()<rate1.sortKey())) {
        rateList.add(0,rate);
        insert = true;
      }
/* if new frameRate object sortKey is less than lesser
object, add in lessor position*/
      else if (rate.sortKey()<rate1.sortKey()) {
        rateList.add(iterateVariable, rate);
        insert = true;
      }
/* if the new frameRate sortKey is the same as current
value then add at the current position */
      else if (rate.sortKey() == rate1.sortKey()){
        rateList.add(iterateVariable, rate);
        insert = true;
      }
/* if sortKey of new frameRate object is greater than the
sortkey of objects in the arraylist then add at the end of
arrayList*/
      else if ((rate.sortKey()>rate1.sortKey()) &&
(iterateVariable == rateList.size())) {
          rateList.add(rate);
          insert = true;
      }
/* if new frameRate object is equal or less than current
position sortkey, then add at position currently held by
nextRate*/
      else if ((rate.sortKey()>rate1.sortKey()) &&
              (rate.sortKey()<=rate2.sortKey())) {
        rateList.add(nextRate, rate);
        insert = true;
      }
/*if new frameRate object sortKey is greater than
nextRate object sortKey then add after nextRate object*/
      else if (rate.sortKey()>rate3.sortKey()) {
        rateList.add(rate);
        insert = true;
      }
      else iterateVariable++;
//iterate current position in arraylist
    }
  }

  public String sortKey() {
    return videoName;
// defines the variable used as the sortKey
  }

  public int getVideoLength()
  {
    return videoLength;
// return the number of frames in the video (int)
  }

  public String getVideoDescription()
  {
    return videoDescription;
// return string containing video description
  }

  public String getVideoName()
  {
    return videoName;
// return string containing video name
  }

  public int getArrayLength()
  {
  return rateList.size();
// return int representing size of the rateList array list
  }

  public ArrayList getRateList()
  {
    return rateList;
/* returns the rateList to a higher class abstraction. This
function is essential for iterating all data. */
  }

  public ArrayList search(int start, int stop) {
/* The search method returns an arraylist with data that
corresponds to frame rates with sortKey values between
start and stop values. */
    ArrayList matchedRateList = new ArrayList();
    for(int i=0;i<rateList.size();i++) {
      frameRate objectInList = (frameRate)rateList.get(i);
      if((objectInList.getFrameRate()>=start) &&
(objectInList.getFrameRate()<=stop))
        {matchedRateList.add(objectInList);
      }
    }
    return matchedRateList;
//returns arrayList
  }

  public String list() {
/* iterates through the arraylist and returns a string
containing a description of all data in the arrayList*/
    String feedback = "";
    Iterator iterator = rateList.iterator();
    while (iterator.hasNext()) {
      frameRate nextRate = (frameRate)iterator.next();
      feedback = feedback+" "+ nextRate.description();
    }
    return feedback;
  }

  public String description() {
// prints one line description of current video object
    String feedback = "Video: "+videoName+"
Length:"+videoLength+" Description:
"+videoDescription;
  return feedback;
 }
}
```

frameRate.java

```
/* frameRate contains information about the frame rate
used to play a video. Additionally an arraylist (frameList)
contains data relating to video specific frame data.*/

package definingroi;

import java.util.*;
import javax.swing.*;

class frameRate {

  private int testFrameRate;
// frame rate used to present video
   private ArrayList frameList = new ArrayList();
// frameList contains data relating specific frame data.


   public frameRate(int x)
   {
     testFrameRate = x;
// assigns values when an frameRate object is created.

   }

   public void add(videoFrame frame) {
/* Add function adds new frame data to the frameList
array list, assuming that the frame name is unique */
     if (checkVideoFrame(frame) != true){
//if frame is unique
       if (frameList.isEmpty() == true) {
/* if no identical frames are currently in the videoList
array list, then add */
       frameList.add(frame);}
       else if (frameList.size() < 2) {
// single item
         videoFrame frame1 =
(videoFrame)frameList.get(0);
/*create temporary frame object from arraylist object.
The desired position of the new object is then
determined.*/
         if ((frame.sortKey())<(frame1.sortKey())) {
           frameList.add(0, frame);
         }
         else {
           frameList.add(frame);
         }
       }
       else {
/* if there are more than one object then call the
compare function, which iterates through the list to
identify the correct new position.*/
         compare(frame);
       }
     }
     else {
       JOptionPane.showMessageDialog(null,"The frame
object is
invalid.","ERROR",JOptionPane.WARNING_MESSA
GE);
// in case object is not a frame object
     }
   }

   public boolean checkadd(videoFrame frame) {
/* checks the content of the frame object before adding
it to the arrayList */
     boolean feedback = false;
/* feedback used to identify duplicated objects */
    if (checkVideoFrame(frame) != true){
//if no fault is found in the frame object
      if (frameList.isEmpty() == true) {
//if arrayList is empty – then add
      frameList.add(frame);
      feedback = true;}
      else if (frameList.size() < 2) {
        videoFrame frame1 =
(videoFrame)frameList.get(0);
        if ((frame.sortKey())!=(frame1.sortKey())) {
          frameList.add(0, frame);
/* add new frame object at position 0, if less sortKey is
less than current object in list*/
          feedback = true;
        }
        else {
          frameList.add(frame);
          feedback = true;
/* add new frame object at end of list, if less sortKey is
greater than current object in list*/
        }

      }
      else {
        compare(frame);
        feedback = true;
// true if no duplicate exists
      }
    }
    return feedback;
   }

   public boolean checkVideoFrame(videoFrame frame) {
/* iterates through the arraylist to determine whether the
sortkey, which is defined as the frame number is
duplicated. */
     boolean feedback = false;
/* feedback used to identify frame objects that should
not be added to the arrayList – false assumes no error */
     for(int i=0; i<frameList.size();i++) {
       videoFrame frameInArray =
(videoFrame)frameList.get(i);
       if ("" + frame.getFrameNumber() == null) {
         feedback = true;}
/* if the new frame object has no contents then there is
an error */
       else if (frame.getFrameNumber() ==
frameInArray.getFrameNumber()) {
/* if the new frame object has a duplicated sortKey there
is an error*/
         feedback = true;
       }
     }
     return feedback;
   }

   public void compare (videoFrame frame) {
/* when adding frame data to a populated arraylist, the
compare method iterates through the arraylist to identify
the correct location for the new frame object. */
     int iterateVariable = 0;
     int nextFrame = iterateVariable +1;
     int lastFrame = frameList.size()-1;
     boolean insert = false;
     while (iterateVariable < frameList.size() && !insert) {
/* for as many objects as the number that exists in the
arraylist, whilst the object has not been added - create
three temporary frame objects relating to the object
being tested and those either side in the array */
       videoFrame frame1 =
(videoFrame)frameList.get(iterateVariable);
       videoFrame frame2 =
(videoFrame)frameList.get(nextFrame);
       videoFrame frame3 =
(videoFrame)frameList.get(lastFrame);
       if ((iterateVariable == 0) &&
(frame.sortKey()<frame1.sortKey())) {
/* if current position is 0 and new frame object sortkey
is less then add as first object in the array */
         frameList.add(0,frame);
        insert = true;
      }
/* if new frame object sortKey is less than lesser object,
add in lessor position*/
      else if (frame.sortKey()<frame1.sortKey()) {
        frameList.add(iterateVariable, frame);
        insert = true;
      }
/* if the new frame sortKey is the same as current value
then add at the current position */
      else if (frame.sortKey() == frame1.sortKey()){
        frameList.add(iterateVariable, frame);
        insert = true;
      }
/* if sortKey of new frame object is greater than the
sortkey of objects in the arraylist then add at the end of
arrayList*/
      else if ((frame.sortKey()>frame1.sortKey()) &&
(iterateVariable == frameList.size())) {
          frameList.add(frame);
          insert = true;
      }
/* if new frame object is equal or less than current
position sortkey, then add at position currently held by
nextFrame */
      else if ((frame.sortKey()>frame1.sortKey()) &&
            (frame.sortKey()<=frame2.sortKey())) {
        frameList.add(nextFrame, frame);
        insert = true;
      }
/*if new frame object sortKey is greater than nextFrame
object sortKey then add after nextFrame object*/
      else if (frame.sortKey()>frame3.sortKey()) {
        frameList.add(frame);
        insert = true;
      }
      else iterateVariable++;
//iterate current position in arraylist
    }
  }

  public int sortKey() {
    return testFrameRate;
// defines the variable used as the sortKey
  }

  public int getFrameRate()
  {
    return testFrameRate;
//returns int relating to the presentation frame Rate
  }

  public int getArrayLength()
  {
  return frameList.size();
// returns the number of different frame rates used
  }

  public ArrayList getFrameList()
  {
    return frameList;
/* returns the frameList to a higher class abstraction.
This function is essential for iterating all data. */
  }

  public ArrayList search(int start, int stop) {
/* The search method returns an arraylist with data that
corresponds to frames with sortKey values between start
and stop values. */
    ArrayList matchedFrameList = new ArrayList();
    for(int i=0;i<frameList.size();i++) {
      videoFrame objectInList =
(videoFrame)frameList.get(i);
      if((objectInList.getFrameNumber()>=start) &&
(objectInList.getFrameNumber()<=stop))
          {matchedFrameList.add(objectInList);
      }
    }
    return matchedFrameList;
//returns arrayList
  }

  public String list() {
/* iterates through the arraylist and returns a string
containing a description of all data in the arrayList*/

    String feedback = "";
    Iterator iterator = frameList.iterator();
    while (iterator.hasNext()) {
      videoFrame nextFrame =
(videoFrame)iterator.next();
      feedback = feedback+" "+ nextFrame.description();
    }
    return feedback;
  }

  public String description() {
// prints one line description of current video object
   String feedback = "FrameRate "+testFrameRate+"
fps";
   return feedback;
  }

}
```

videoFrame.java

```
/* videoFrame contains information about frame
number. Additionally an arraylist (coordinateList)
contains data relating to x and y coordinate data for all
participants relating to a specific frame.
package definingroi;
import java.util.*;
import javax.swing.*;

public class videoFrame {
  private int frameNumber;
  private ArrayList coordinateList = new ArrayList();
// records frame number and XYcoordinate values


  public videoFrame(int x)
  {
    frameNumber = x;
// assigns values when an videoFrame object is created.
  }

  public void add(xyCoordinates coord) {
/* Add function adds new coordinate data to the
coordinateList array list, assuming that the participant
number is unique */
    if (checkxyCoordinates(coord) != true){
//if XYcoordinate data is unique
      if (coordinateList.isEmpty() == true) {
/* if no XYcoordinate objects are currently in the
coordinateList array list, then add */
      coordinateList.add(coord);}
      else if (coordinateList.size() < 2) {
// single item
        xyCoordinates coord1 =
(xyCoordinates)coordinateList.get(0);
/*create temporary XYcoordinate object from arraylist
object. The desired position of the new object is then
determined.*/
        if ((coord.sortKey())<(coord1.sortKey())) {
          coordinateList.add(0, coord);
        }
        else {
          coordinateList.add(coord);
        }
      }
      else {
/* If there are more than one object then call the
compare function, which iterates through the list to
identify the correct new position.*/
        compare(coord);
      }
    }
    else {
      JOptionPane.showMessageDialog(null,"The xy
coordinate object is
invalid.","ERROR",JOptionPane.WARNING_MESSA
GE);
// in case object is not a XYcoordinate object
    }
  }

  public boolean checkadd(xyCoordinates coord) {
/* checks the content of the XYcoordinate object before
adding it to the arrayList */
    boolean feedback =false;
// feedback used to identify duplicated objects
    if (checkxyCoordinates(coord) != true){
//if no fault is found in the XYcoordinate object
      if (coordinateList.isEmpty() == true) {
//if arrayList is empty – then add
      coordinateList.add(coord);
      feedback = true;}
      else if (coordinateList.size() < 2) {
        xyCoordinates coord1 =
(xyCoordinates)coordinateList.get(0);
        if ((coord.sortKey())!=(coord1.sortKey())) {
          coordinateList.add(0, coord);
/* add new XYcoordinate object at position 0, if less
sortKey is less than current object in list*/
          feedback = true;
        }
        else {
          coordinateList.add(coord);
/* add new frameRate object at end of list, if less
sortKey is greater than current object in list*/
          feedback = true;
        }

      }
      else {
        compare(coord);
       // true if no duplicate exists
        feedback = true;
      }
    }
    return feedback;
  }

  public boolean checkxyCoordinates(xyCoordinates
coord) {
/* iterates through the arraylist to determine whether the
sortkey, which is defined as the participant name is
duplicated. */
    boolean feedback = false;
/* feedback used to identify XYcoordinate objects that
should not be added  to the arrayList – false assumes no
error */
    for(int i=0; i<coordinateList.size();i++) {
      xyCoordinates partInArray =
(xyCoordinates)coordinateList.get(i);
      if ("" + coord.getParticipant() == null) {
        feedback = true;}
/* if the new coordinate object has no contents then
there is an error */
      else if (coord.getParticipant() ==
partInArray.getParticipant()) {
/* if the new XYcoordinate object has a duplicated
sortKey there is an error*/
        feedback = true;
      }
    }
    return feedback;
  }

  public void compare (xyCoordinates coord) {
/* when adding XYcoordinate data to a populated
arraylist, the compare method iterates through the
arraylist to identify the correct location for the new
coordinate object. */
    int iterateVariable = 0;
    int nextCoordinate = iterateVariable +1;
    int lastCoordinate = coordinateList.size()-1;
    boolean insert = false;
    while (iterateVariable < coordinateList.size() &&
!insert) {
/* for as many objects as the number that exists in the
arraylist, whilst the object has not been added -  create
three temporary XYcoordinate objects relating to the
object being tested and those either side in the array */
      xyCoordinates coord1 =
(xyCoordinates)coordinateList.get(iterateVariable);
      xyCoordinates coord2 =
(xyCoordinates)coordinateList.get(nextCoordinate);
      xyCoordinates coord3 =
(xyCoordinates)coordinateList.get(lastCoordinate);
/* if current position is 0 and new XYcoordinate object
sortkey is less then add as first object in the array */
      if ((iterateVariable == 0) &&
(coord.sortKey()<coord1.sortKey())) {
        coordinateList.add(0,coord);
        insert = true;
      }
/* if new XYcoordinate object sortKey is less than lesser
object, add in lessor position*/
      else if (coord.sortKey()<coord1.sortKey()) {
        coordinateList.add(iterateVariable, coord);
        insert = true;
      }
/* if the new XYcoordinate sortKey is the same as
current value then add at the current position */
      else if (coord.sortKey() == coord1.sortKey()){
        coordinateList.add(iterateVariable, coord);
        insert = true;
      }
/* if sortKey of new XYcoordinate object is greater than
the sortkey of objects in the arraylist then add at the end
of arrayList*/
      else if ((coord.sortKey()>coord1.sortKey()) &&
(iterateVariable == coordinateList.size())) {
          coordinateList.add(coord);
          insert = true;
      }
/* if new XYcoordinate object is equal or less than
current position sortkey, then add at position currently
held by nextCoordinate*/
      else if ((coord.sortKey()>coord1.sortKey()) &&
              (coord.sortKey()<=coord2.sortKey())) {
        coordinateList.add(nextCoordinate, coord);
        insert = true;
      }
/*if new XYcoordinate object sortKey is greater than
nextV object sortKey then add after nextCoordinate
object*/
      else if (coord.sortKey()>coord3.sortKey()) {
        coordinateList.add(coord);
        insert = true;
      }
      else iterateVariable++;
//iterate current position in arraylist
    }
  }

  public int getFrameNumber()
  {
    return frameNumber;
```

```
// returns the current frameNumber

  }

  public ArrayList getCoordinateList()
  {
    return coordinateList;
/* returns the CoordinateList to a higher class
abstraction. This function is essential for iterating all
data. */
  }

  public int getArrayLength()
  {
    return coordinateList.size();
/* returns a integer value representing size of the
coordinateList array list */
  }

  public ArrayList search(int x1, int x2, int y1, int y2) {
/* The search method returns an arraylist with
coordinate data with sortKey values between x1, x2, y1,
y2 coordinate square. */
    ArrayList matchedCoordinateList = new ArrayList();
    for(int i=0;i<coordinateList.size();i++) {
      xyCoordinates objectInList =
(xyCoordinates)coordinateList.get(i);
      if((objectInList.getXCoordinate()>x1) &&
(objectInList.getXCoordinate()<x2)
        && (objectInList.getYCoordinate()>y1) &&
(objectInList.getYCoordinate()<y2))
        { matchedCoordinateList.add(objectInList);
        }
      }
      return matchedCoordinateList;
//returns arrayList
    }

    public String list() {
/* iterates through the arraylist and returns a string
containing a description of all data in the arrayList*/
      String feedback = "";
      Iterator iterator = coordinateList.iterator();
      while (iterator.hasNext()) {
        xyCoordinates nextCoordinates =
(xyCoordinates)iterator.next();
        feedback = feedback+" "+
nextCoordinates.description();
      }
      return feedback;
    }

    public String description() {
// prints one line description of current video object
      String feedback = "FrameNumber
"+frameNumber+" - leaves : "+coordinateList.size();
      return feedback;
    }

    public int sortKey() {
     return frameNumber;
    }

}
```

XYCoordinates.java

```
/* XYCoordinate contains information about X and Y
coordinate values, participant number and viewing
order.*/
package definingroi;

public class xyCoordinates {
  private int xCoord;
  private int yCoord;
  private int part;
  private String videoOrder;

  public xyCoordinates(int participant, int x, int y)
  {
    xCoord = x;
    yCoord = y;
    part = participant;
/* assigns values when a XYCoordinate object is created,
yet no viewing order is given. */
  }

  public xyCoordinates(int participant, String order, int x,
int y)
 {
   xCoord = x;
   yCoord = y;
   part = participant;
   videoOrder = order;
/* assigns values when a XYCoordinate object is created,
and viewing order information is available. */
  }

  private void setXCoordinate(int x)
  {
    xCoord = x;
// specifically assigns a value to the X coordinate
  }

  public int getXCoordinate()
  {
    return xCoord;
//returns the current X value
  }

  private void setYCoordinate(int y)
  {
    yCoord = y;
// specifically assigns a value to the Y coordinate

  }

  public int getYCoordinate()
  {
    return yCoord;
//returns the current Y value
  }

  public int getParticipant()
  {
    return part;
//returns the participant value (int)
  }

  public String getOrder()
  {
    return videoOrder;
//returns the viewing order (String)
  }

  public int sortKey() {
   return part;
//defined the sortKey for XYCoordinate objects
  }

  public String description() {
// prints one line description of current video object

    String feedback = "Part: "+part+" x:"+xCoord+"
y:"+yCoord;
    return feedback;
  }
}
```

extractingData.java – used to extract data from data file

```
/* extractingData takes the output files from the
Arrington Research Viewpoint eyetracker and extracts
and synchronises participant data. It is worth spending
time looking at Appendix B to make sure the reader is
familiar with the format of data taken from the Viewpont
eye-tracker. */

package definingroi;

import java.io.*;
import java.awt.*;
import java.util.*;
import java.lang.*;

public class extractingData {

  File videoFile;
  String VideoMainDirectory;

  public void extractingData(){
  }

  public int videoDuration(String VideoName, File
inputFile){
/* As the number of frames is consistent videoDuration
information is used to synchronise eye-tracking data and
specific video frames. Returns a integer relating to the
number of frames of the stated video file. Video files
must be renamed BA05.mpg, BA15.mpg, BA25.mpg,
BD05.mpg, etc. to ensure consistency. */
  File file;
  int frames=0;
  boolean print= false;
  file = inputFile;
  if (file.getName().startsWith(VideoName)) {
// if file has name
  try {
//define the number of frames for a specific video
      if (file.getName().startsWith("BD")) {frames = 908;
};
      if (file.getName().startsWith("BA")) {frames = 640;
};
      if (file.getName().startsWith("CH")) {frames = 834;
};
      if (file.getName().startsWith("DA")) {frames = 813;
};
      if (file.getName().startsWith("FC")) {frames =
1128; };
      if (file.getName().startsWith("LN")) {frames = 789;
};
      if (file.getName().startsWith("NA")) {frames = 918;
};
      if (file.getName().startsWith("NW")) {frames =
954; };
      if (file.getName().startsWith("OR")) {frames = 982;
};
      if (file.getName().startsWith("RG")) {frames = 859;
};
      if (file.getName().startsWith("SN")) {frames = 845;
};
      if (file.getName().startsWith("SP")) {frames = 918;
};
  }
  catch (Exception e) {
    System.err.println("Failed to create a processor from
the given url: " + e);
// if file reading error occurs
  }
  }
  return frames;
// return frame number to the user
}

 public boolean exists(String VideoName){
/*The exists() method checks that the current video file
actually exists. */
   boolean feedBack = false;
/* Feedback is true if there is a match.The following
code allows the user to defined the directory, where the
specific video may exist. If this is the first call of the
method this value is stored in the Video MainDirectory
variable. However, if this is not the first time that this
method has been called then the directory value is
restored from the VideoMainDirectory variable.*/
   File file;
   String videoDirectory = VideoMainDirectory;
   if (videoDirectory == null) {
    Frame dialogInputFrame = new Frame();
    FileDialog inputFileName = new
FileDialog(dialogInputFrame, "Video Directory");
    inputFileName.setVisible(true);
    String selectedItem;
    do {
     videoDirectory = inputFileName.getDirectory();
     VideoMainDirectory
=inputFileName.getDirectory();
    }
    while(videoDirectory == null);
    dialogInputFrame.dispose();
   }

   File temporyDirectory = new File(videoDirectory);
   String fileList[] = temporyDirectory.list();
//create a list of the files in the selected directory

   for (int currentFile = 0; currentFile < fileList.length ;
currentFile++){
/* for all the files in the directory compare the URL of
the current file to check whether the name matched the
wanted file. */
    String selectedItem = videoDirectory +
fileList[currentFile];
    String inputString = videoDirectory + VideoName;
    file = new File(selectedItem);
    if ((videoDuration(VideoName,file))>0){
     System.out.println("Number of
Video:"+currentFile+"/"+fileList.length);
/* if current video being compared has a duration match
then print then duration to screen. */
     System.out.println("Video
Compare:"+file.getAbsolutePath());
     System.out.println("Video Input
String:"+inputString);
     if
((file.getAbsolutePath()).compareToIgnoreCase(inputStri
ng)==0){
/* if file URL is matched then print confirmation and
appropriately assign the feedback (true to show a match)
and videoFile variables. */
        System.out.println("Number of Frames:
"+videoDuration(VideoName,file));
        System.out.println("VIDEO EXISTS");
        feedBack = true; videoFile = file;
     }
    }
   }

   return feedBack;
 }

 public experiment extract(String experimentName,
String experimentDescription){
```

```
/* The extract method goes through a directory of
Arrington Research eye-tracker user data files and
extracts and synchonises participant data. In addition, the
extract method created the object-oreiented data
structure from the extracted data.*/
  String inputString;
/* inputString used to manipulate single lines from data
files. The following code allows the user to select the
data directory – named directory.*/
  Frame dialogInputFrame = new Frame();
  FileDialog inputFileName = new
FileDialog(dialogInputFrame, "Data Directory");
  inputFileName.setVisible(true);
  String directory, selectedItem;
  experiment exp = new
experiment(experimentName,experimentDescription);
//creates a new experiment object
  do
// while(count < outputVector.size())

    directory = inputFileName.getDirectory();
    while(directory == null);
    dialogInputFrame.dispose();


    if (exists(" ")){}
    File temporyDirectory = new File(directory);
//defines data folder
    int participant=1;
//sets participant value for extraction batch
    String fileList[] = temporyDirectory.list();
//creates a list of files in the data folder
    for (int currentFile = 0; currentFile < fileList.length ;
currentFile++){
// for all files in the data folder, select file, define length
      File f = new File(fileList[currentFile]);
      int size = fileList.length ;
      selectedItem = directory + fileList[currentFile];
//define file URL
      Vector inputVector = new Vector();
// used to store input data
      try{
// loads data from file into inputVector
        FileReader inputFile = new
FileReader(selectedItem);

        BufferedReader reader = new
BufferedReader(inputFile);
        String bufferInput;
        while((bufferInput = reader.readLine()) != null){
// for all lines in data file, import into inputVector
          inputVector.add(bufferInput);
        }
      catch(Exception e){
      System.err.println(e.getMessage());
    }

    Vector outputVector = new Vector();
// outputVector only contains text with useable data
    String currentLine = "";
    String outputLine = "";
    int numOfLines = inputVector.size();
    int sectionCount = 0;
/* The sectionCount variable relates to the statelogic –
see Appendix D. Only data between 2B and 2C tags is
relevant */
    for(int lineNum = 0; lineNum < numOfLines;
lineNum++) {// for all input lines
      currentLine = (String)inputVector.get(lineNum);
//Get the current string from the input Vector
      StringTokenizer currentLineTokenizer = new
StringTokenizer(currentLine);
      if (currentLineTokenizer.hasMoreTokens()){
        outputLine = currentLineTokenizer.nextToken();
      }
      while(currentLineTokenizer.hasMoreTokens()){
        outputLine = outputLine + " " +
currentLineTokenizer.nextToken();

/* if the line starts with 2A then we are waiting for the
video to be started.There sectionCount =1. */
        if (outputLine.startsWith("2 A") == true) { //if we
are section A, set tag
          sectionCount = 1;
        }
          if (outputLine.startsWith("2 B") == true) {
sectionCount = 2;
          }
/* if the line starts with 2B then we now expect valid
data.Therefore sectionCount = 2. */
          if (outputLine.startsWith("3 :") == true) {
//if title line add line to outputVector
          while(currentLineTokenizer.hasMoreTokens()){
              outputLine = outputLine + " " +
currentLineTokenizer.nextToken(); // add all tokens
            }
            outputVector.add(outputLine);
          }
          if (outputLine.startsWith("2 C") == true) {
/* if the line starts with 2C then we now expect valid
data to stop.Therefore sectionCount =0 –  i.e. onto the
next video. */
            sectionCount = 0;
          }
          if (sectionCount == 3) { // if found data
            if (outputLine.startsWith("2 +") == true) {
              // yet multiple 2 + exist.
              //There reset tag to sectionCount = 2
              sectionCount = 2;
            }
            else
            { if (outputLine.startsWith("9") == false) {
/* this section removes timing lines (start 9), yet allows
all data whilst sectionCount = 3 (data found) */
          while(currentLineTokenizer.hasMoreTokens()){
                outputLine = outputLine + " " +
currentLineTokenizer.nextToken();
              } //while
              outputVector.add(outputLine); // add all
other lines to output vector
              }
            }
          }
          if (sectionCount == 2) {
/* if we find 2 + (+ pressed, therefore video starting) we
set sectionCount to 3 (data found) */
            if (outputLine.startsWith("2 +") == true) {
              sectionCount = 3;
            }
          }
        }
      }
      numOfLines = outputVector.size();
//finds size of outputVector and prints to screen

      for(int lineNum = 0; lineNum < numOfLines;
lineNum++)
      {
        System.out.println(outputVector.get(lineNum));
      }
      int count = 0, lineCount = 0;
// used to count the line numbers
      int past = 0, dtSum = 0, currentdt = 0, frame = 0;
// used to synchronise frames and timings
      String xPosition = "", yPosition = "";
//coordinate values
      String dtTemp = "", fileName = "";
      boolean beginFile = false;
      do {
outputLine = ((String)outputVector.get(count));
        if ((count + 1) == outputVector.size()) {
          outputLine = "3 :";
        }
```

```
// so that last video labelled before data arrives
          if (outputLine.startsWith("3 :") == true) {
           if (beginFile == false) {
//define the start of the video
             past = count;
             beginFile = true;
             fileName =
outputLine.substring(20,24)+currentFile;
/* extracts video name - currentFile equals data file */
             System.out.println(fileName);
           }
           else {
            if ((count + 1) < outputVector.size()) {
             fileName =
outputLine.substring(20,24)+currentFile;
// so that last video labelled before data arrives
}
           try{
/* while (past < count), i.e. – for all lines relating to a
specific video frame. The following code corrects the
frame count. We therefore know the duration of the
video. We can calculate the delta delay shown on the data
file and accordingly work out the delay due to loading
the video file. We can therefore associate the frames and
XY coordinates. */
            ArrayList correctFrame = new ArrayList();
            int frameCorrection;
            int LastFrame=0;
            frame = (currentdt/40);
//defines the beginning of dataset
            String videoName="";
            int rate=0;
            do { //do 3
            outputLine = (String)outputVector.get(past);
//read line from outputVector
            System.out.println("OUTPUTLINE:
"+outputLine);
            StringTokenizer stringInput = new
StringTokenizer(outputLine);
            if (lineCount == 0) {
// if title line, write title line to xy file
             videoName = outputLine.substring(20,22) ;
// extracts BA, BD, CH, etc
             rate =
Integer.parseInt(outputLine.substring(22,24));
//extract 05, 15, 25
             System.out.println("Frame Rate: "+rate);
             System.out.println("Video Name:
"+videoName);
            } else {
// if not at the title line
             stringInput.nextToken();
// ignore first token (data tag)
             xPosition = stringInput.nextToken();
//read x Position
             yPosition = stringInput.nextToken();
//read y Position
             dtTemp = stringInput.nextToken();
/* read dt (delta time) between the pressing of + to
current data reading. */
             try {
/* dtSum determines the sum of all delays and allows us
to synchorise frames to eye-tracking data */
              dtSum = dtSum +
Integer.parseInt(dtTemp.trim()); // add to int sum
// add delta time to total delta Sum
              do { //writes x,y position at frame intervals
               int frameNumber = ((currentdt/40)-frame);
               int x = Integer.parseInt(xPosition) ;
               int y = Integer.parseInt(yPosition) ;
               System.out.println("TEMP
FRAME:"+frameNumber+" X:"+x+"Y: "+y);
               frameCorrect NewCorrection = new
frameCorrect(frameNumber,participant,x,y);
// create XYCoordinate object for all frames
               correctFrame.add(NewCorrection);
               LastFrame = frameNumber;
//used to determine the require frame shift
               currentdt = currentdt + 40;
/* increments frame dt whilst currentdt is less than the
actual dtSum- where sample data is no longer valid */
              } while (currentdt <= dtSum);
             }
             catch (NumberFormatException e) {}
             }
             lineCount++;
             past++;
            } while (past < count);
            //System.out.println("END OF THE FRAME
DESCRIPTIONS");
            System.out.println("Last Frame :
"+LastFrame);
            String tempName;
            if (rate < 10) {
             tempName = videoName+"0"+rate+".mpg";
            } else {
             tempName = videoName+rate+".mpg";
            }
// defined video name from title
            System.out.println("NAME: "+tempName);
            if (exists(tempName)) {
             System.out.println(tempName+" EXISTS");
//check that file actually exists
             //System.out.println("EXIST CLAUSE
FILE: "+videoFile.getName());
//defines video duration
             int duration =
videoDuration(tempName,videoFile);
             System.out.println("Frames
Calculated:"+duration);
             frameCorrection = duration-LastFrame;
// accordingly defines and displays the frame shift
             System.out.println("Correction :
"+frameCorrection);
             video vid = new video(videoName, duration,
tempName);
//creates a video object and adds to experiment object
             if (exp.checkadd(vid)) {
              System.out.println("VIDEO ADDED:
"+tempName);
             }
/* The following code finds the appropriate video
object, creates a framerate object (defined by extracted
data and adds it to the identified video object) */
             Iterator videoiterator =
exp.getVideoList().iterator();
             int videocount = 0;
             while (videoiterator.hasNext()) {
              video nextVideo =
(video)videoiterator.next();
              if
(nextVideo.getVideoName().compareToIgnoreCase(vide
oName)==0) {
               frameRate testRate = new frameRate(rate);
               if
(((video)exp.getVideoList().get(videocount)).checkadd(tes
tRate)) {
                System.out.println("ADDED
FRAMERATE: "+testRate);
               }
              }
              videocount++;
             }
             int finalFrameNumber=0;
             int xCoord=0; int yCoord=0; int part=0;
              int generatedFrame;
             boolean firstFrame = true;
/* for all frame video coordinates in correctFrames
arrayList */
             for(int ArrayPosition = 0; ArrayPosition <
correctFrame.size(); ArrayPosition++) {
//export data into variables that can be manipulated
```

```
            xCoord =
((frameCorrect)correctFrame.get(ArrayPosition)).getXCo
ordinate();
            yCoord
=((frameCorrect)correctFrame.get(ArrayPosition)).getYC
oordinate();
            part =
((frameCorrect)correctFrame.get(ArrayPosition)).getParti
cipant();
            videoiterator = exp.getVideoList().iterator();
            videocount = 0; int ratecount = 0;
            while (videoiterator.hasNext()) {
            video nextVideo =
(video)videoiterator.next();
            if
(nextVideo.getVideoName().compareToIgnoreCase(vide
oName)==0) {
             Iterator rateiterator =
(((video)exp.getVideoList().get(videocount)).getRateList()
).iterator();
             frameRate nextRate =
(frameRate)rateiterator.next();
// find correct position in the object struction
             if ((nextRate.getFrameRate())==rate) {
              videoFrame testFrame = new
videoFrame(((((frameCorrect)correctFrame.get(ArrayPos
ition)).getFrameNumber())+frameCorrection));
//create a frame that takes into account of frame shift
              if (testFrame.getFrameNumber()>0){
               if
(((frameRate)(((video)exp.getVideoList().get(videocount))
.getRateList()).get(ratecount)).checkadd(testFrame)) {
//if not duplicated, add the frame to the object structure
                System.out.println("ADDED FRAME:
"+((((frameCorrect)correctFrame.get(ArrayPosition)).get
FrameNumber())+frameCorrection));
                if (firstFrame){
                 for (int i=frameCorrection; i>0; i--){
/* To ensure that all frames exist we need to add the
number of frames defined by the frame shift. All have
the same starting coordinate values. If there is no error –
add to frame object */
                  videoFrame correctionFrame = new
videoFrame(i);
                  xyCoordinates testxyCoordinates =
new xyCoordinates(part,xCoord,yCoord);
                  if
(correctionFrame.checkadd(testxyCoordinates)) {
                   System.out.println("ADDED XY: "
+ xCoord+ " " + yCoord);
                  }
                  if
(((frameRate)(((video)exp.getVideoList().get(videocount))
.getRateList()).get(ratecount)).checkadd(correctionFrame
)) {
                   System.out.println("ADDED
FRAME: "+i);
                 }
                 firstFrame = false;
//only true for first added frame
                }
               }
              }
             }
            }
             ratecount++;
            }

            videocount++;
           }
// the following code finds the appropriate video frame
            videoiterator = exp.getVideoList().iterator();
            videocount = 0; ratecount = 0; int
framecount = 0;
            while (videoiterator.hasNext()) {
             video nextVideo =
(video)videoiterator.next();
            if
(nextVideo.getVideoName().compareToIgnoreCase(vide
oName)==0) {
             Iterator rateiterator =
(((video)exp.getVideoList().get(videocount)).getRateList()
).iterator();
             while (rateiterator.hasNext()) {
              frameRate nextRate =
(frameRate)rateiterator.next();
              if (nextRate.getFrameRate()==rate) {
               Iterator frameiterator =
((((frameRate)(((video)exp.getVideoList().get(videocount)
).getRateList()).get(ratecount)).getFrameList()).iterator());
               while (frameiterator.hasNext()) {
               videoFrame nextFrame =
(videoFrame)frameiterator.next();
               finalFrameNumber =
((((frameCorrect)correctFrame.get(ArrayPosition)).getFra
meNumber())+frameCorrection);
//create new final XYCoordinate object
               if
(nextFrame.getFrameNumber()==finalFrameNumber) {
                xyCoordinates testxyCoordinates =
new
xyCoordinates(((frameCorrect)correctFrame.get(ArrayPo
sition)).getParticipant(),

((frameCorrect)correctFrame.get(ArrayPosition)).getXCo
ordinate(),

((frameCorrect)correctFrame.get(ArrayPosition)).getYCo
ordinate());
                if
(((videoFrame)(((frameRate)(((video)exp.getVideoList().g
et(videocount)).getRateList()).get(ratecount)).getFrameLi
st()).get(framecount)).checkadd(testxyCoordinates));
/* add to frame if duplicate participant number does not
already exist */
                System.out.println("ADDED XY:
"+((frameCorrect)correctFrame.get(ArrayPosition)).getX
Coordinate()+"
"+((frameCorrect)correctFrame.get(ArrayPosition)).getY
Coordinate());
               }
               framecount++;
               }
              }
              ratecount++;
             }
            }
            videocount++;
           }
          }

/* the following creates and adds all other frames and
links XYCoordinate objects. Note that instead of having
the value of the first frame being equal to 0, the values
are shifted to make the first frame number 1. */
          for(int ArrayPosition = 0; ArrayPosition <
(finalFrameNumber-1); ArrayPosition++) {
            videoiterator = exp.getVideoList().iterator();
// search through videos
            videocount = 0; int ratecount = 0;
            while (videoiterator.hasNext()) {
            video nextVideo =
(video)videoiterator.next();
            if
(nextVideo.getVideoName().compareToIgnoreCase(vide
oName)==0) {
             Iterator rateiterator =
(((video)exp.getVideoList().get(videocount)).getRateList()
).iterator();
//search through framerates
             frameRate nextRate =
(frameRate)rateiterator.next();
             if ((nextRate.getFrameRate())==rate) {
                  videoFrame testFrame = new
videoFrame(ArrayPosition+1);
                  if
(((frameRate)(((video)exp.getVideoList().get(videocount))
.getRateList()).get(ratecount)).checkadd(testFrame)) {
                  }
                 }
                ratecount++;
               }
               videocount++;
               }
               videoiterator = exp.getVideoList().iterator();
//search through all videos
               videocount = 0; ratecount = 0; int
framecount = 0;
               while (videoiterator.hasNext()) {
                video nextVideo =
(video)videoiterator.next();
                if
(nextVideo.getVideoName().compareToIgnoreCase(vide
oName)==0) {
                 Iterator rateiterator =
(((video)exp.getVideoList().get(videocount)).getRateList()
).iterator();
                 while (rateiterator.hasNext()) {
                  frameRate nextRate =
(frameRate)rateiterator.next();
                  if (nextRate.getFrameRate()==rate) {
                   Iterator frameiterator =
((((frameRate)(((video)exp.getVideoList().get(videocount)
).getRateList()).get(ratecount)).getFrameList()).iterator());
//search all frames
                   while (frameiterator.hasNext()) {
                   videoFrame nextFrame =
(videoFrame)frameiterator.next();
                   if
(nextFrame.getFrameNumber()==(ArrayPosition+1)) {
                    xyCoordinates testxyCoordinates =
new xyCoordinates(part,xCoord,yCoord);
//create new XYcoordinate object
                    if
(((videoFrame)(((frameRate)(((video)exp.getVideoList().g
et(videocount)).getRateList()).get(ratecount)).getFrameLi
st()).get(ArrayPosition)).checkadd(testxyCoordinates));
//add XYobject to associated frame
                   }
                   framecount++;
                   }
                  }
                  ratecount++;
                 }
                }
                videocount++;
               }
              }
// if file does not exist then display message
          } else {System.out.println("VIDEO DOES
NOT EXIST");}
          lineCount = 0;
         } catch(Exception e)  {
          System.err.println(e.getMessage());
         }
         past = count;
//next video
        }
       }
//move to next text line
      count = count + 1;
     } while(count < outputVector.size());
     participant++;
// move to next participant, i.e. next data file
   }// end for
   return exp;
  }// end main
}
```

testextractRoI.java – for importing, cleaning, displaying and saving multiple data sets

```
package definingroi;
import java.util.*;
import java.io.*;
import java.awt.*;

public class testextractRoI {
/* textextract is a runnable program to setup and test the
extractData class. It creates a topData object and then
extracts data from three experiments ext1, ext2 and ex3.
Finally it iterates through the whole data structure and
prints all leaf nodes.*/
  public static void main(String[] args) {
    topData top = new topData();
    extractingData ext1 = new extractingData();
    extractingData ext2 = new extractingData();
    extractingData ext3 = new extractingData();
    top.add(ext1.extract("CONTROL", "CONTROL"));
    top.add(ext2.extract("JITTER", "JITTER"));
    top.add(ext3.extract("DELAY", "DELAY"));
// experiments created and extracted from data files
    ArrayList experimentSearch = new ArrayList();
    experimentSearch = top.getExperimentList();
    Iterator experimentiterator =
experimentSearch.iterator();
//for all experiments
    while (experimentiterator.hasNext()) {
      experiment nextexperiment = (experiment)
experimentiterator.next();
      ArrayList videoSearch = new ArrayList();
      videoSearch = nextexperiment.getVideoList();
      Iterator videoiterator = videoSearch.iterator();
//for all videos
      while (videoiterator.hasNext()) {
        video nextVideo = (video) videoiterator.next();
        ArrayList rateSearch = new ArrayList();
        rateSearch = nextVideo.getRateList();
        Iterator rateiterator = rateSearch.iterator();
//for all frame rates
        while (rateiterator.hasNext()) {
          frameRate nextRate = (frameRate)
rateiterator.next();
          ArrayList frameSearch = new ArrayList();
          frameSearch = nextRate.getFrameList();
          Iterator frameiterator = frameSearch.iterator();
//for all video frames
          while (frameiterator.hasNext()) {
            videoFrame nextFrame = (videoFrame)
frameiterator.next();
            ArrayList coordinateSearch = new ArrayList();
            coordinateSearch =
nextFrame.getCoordinateList();
            Iterator coordinateiterator =
coordinateSearch.iterator();
//for all XY coordinate values
            while (coordinateiterator.hasNext()) {
              xyCoordinates nextCoordinates =
(xyCoordinates)
                  coordinateiterator.next();
//Display contents
                System.out.println("Video:" +
nextVideo.getVideoName() + " Rate:" +
nextRate.getFrameRate() + " Frame:" +
nextFrame.getFrameNumber() + " Part:" +
nextCoordinates.getParticipant() + " x:" +
nextCoordinates.getXCoordinate() + " y:" +
nextCoordinates.getYCoordinate());
          }
        }
      }
        }
      }
      top.save();
//saves the content structure to a user defined file
    }
  }
```

loadData.java – load data back from data file

```
package definingroi;
/* loadData takes a saved object structure file and
reimports the data. This is useful as it means that data is
not being freshly imported from data files, which leads to
unnecessary delay and error. */
import java.util.*;
import javax.swing.*;
import java.io.*;
import java.awt.*;

class loadData {

  topData dataStore = new topData();
//create topData object
  public void load() {

/* The following code allows the user to define the file
location of the saved data. */
    String inputString = "";
    Frame dialogInputFrame = new Frame();
    FileDialog inputFileName = new
FileDialog(dialogInputFrame, "Loading Data");
    inputFileName.setVisible(true);
    String selectedItem;
    do
      selectedItem = inputFileName.getDirectory() +
inputFileName.getFile();
    while(selectedItem == null);
// repeat process until a file has been selected
    dialogInputFrame.dispose();

/* Create a new inputVector and for all lines of data file
import the contents into the inputVector */
    Vector inputVector = new Vector();
    try{
      FileReader inputFile = new FileReader(selectedItem);
    BufferedReader reader = new
BufferedReader(inputFile);
    String bufferInput;
    while((bufferInput = reader.readLine()) != null)
      inputVector.add(bufferInput);
    }
    catch(Exception e){
      System.err.println(e.getMessage());
    }

    int numOfLines = inputVector.size();
//defines the number of lines in the data file
    ArrayList experimentvalues = new ArrayList();
    ArrayList videovalues = new ArrayList();
    ArrayList ratevalues = new ArrayList();
    ArrayList framevalues = new ArrayList();
    ArrayList coordinatevalues = new ArrayList();
//Arraylists required to contain unlinked created objects
    for(int lineNum = 0; lineNum < numOfLines;
lineNum++)
/* for all lines in the data file (see Appendix C for
structure) */
    {
      inputString = (String)inputVector.get(lineNum);
      String lineToken;
/* if line starts with an E tag (a textual description of the
experiment, e.g. control experiment) */
      if (inputString.startsWith("e") == true) {
        String experimentName = "";
        String experimentDescription = "";
        String currentLine =
(String)inputVector.get(lineNum);
        StringTokenizer lineTokenizer = new
StringTokenizer(currentLine,"\t");
        if (lineTokenizer.hasMoreTokens()){ lineToken =
lineTokenizer.nextToken();}
        if (lineTokenizer.hasMoreTokens()){
experimentName =  lineTokenizer.nextToken();}
// export experiment name
        if (lineTokenizer.hasMoreTokens()){
experimentDescription = lineTokenizer.nextToken(); }
// export experiment description
        if (lineTokenizer.hasMoreTokens() == false){
          experiment exp = new
experiment(experimentName,experimentDescription);
// create experiment object
// add to experiment arrayList
          experimentvalues.add(exp);
          video vid = new video("NEW VIDEO",0,"NEW
VIDEO");
// create a marker video, add to the video ArrayList
          videovalues.add(vid);
          //System.out.println(currentLine+" - INPUT
CONFIRMED");
        } else {
          JOptionPane.showMessageDialog(null,"The
experiment object loaded
incorrectly.","ERROR",JOptionPane.WARNING_MES
SAGE);
        }
      }
      if (inputString.startsWith("v") == true) {
/* if line starts with an V tag (provides a video
description containing video code, number of frames and
video filename) */
        String videoName = "";
        String videoLength = "";
        String videoDescription = "";
        String currentLine =
(String)inputVector.get(lineNum);
        StringTokenizer lineTokenizer = new
StringTokenizer(currentLine,"\t");
        if (lineTokenizer.hasMoreTokens()){ lineToken =
lineTokenizer.nextToken();}
        if (lineTokenizer.hasMoreTokens()){ videoName =
lineTokenizer.nextToken();}
// define video name
        if (lineTokenizer.hasMoreTokens()){ videoLength
=  lineTokenizer.nextToken();}
// define number of frames
        if (lineTokenizer.hasMoreTokens()){
videoDescription = lineTokenizer.nextToken(); }
// define video description
        if (lineTokenizer.hasMoreTokens() == false){
          video vid = new
video(videoName,Integer.parseInt(videoLength),videoD
escription);
//create new video and add to video ArrayList
          videovalues.add(vid);
//add a marker frame rate object to arrayList
          frameRate rate = new frameRate(0);
          ratevalues.add(rate);
          //System.out.println(currentLine+" - INPUT
CONFIRMED");
        } else {
```

```
        JOptionPane.showMessageDialog(null,"The video
object loaded
incorrectly.","ERROR",JOptionPane.WARNING_MES
SAGE);
      }
    }
    if (inputString.startsWith("r") == true) {
/* if line starts with an R tag ( describes the specific
video frame rate) */
      String frameNumber = "";
      String currentLine =
(String)inputVector.get(lineNum);
      StringTokenizer lineTokenizer = new
StringTokenizer(currentLine,"\t");
      if (lineTokenizer.hasMoreTokens()){ lineToken =
lineTokenizer.nextToken();}
      if (lineTokenizer.hasMoreTokens()){ frameNumber
=  lineTokenizer.nextToken();}
// extract frame rate number
      if (lineTokenizer.hasMoreTokens() == false){
        frameRate rate = new
frameRate(Integer.parseInt(frameNumber));
//create new frameRate object and add to Arraylist
        ratevalues.add(rate);
//create marker frame object and add to arraylist
        videoFrame frame = new videoFrame(0);
        framevalues.add(frame);
        // System.out.println(currentLine+" - INPUT
CONFIRMED");
      } else {
        JOptionPane.showMessageDialog(null,"The rate
object loaded
incorrectly.","ERROR",JOptionPane.WARNING_MES
SAGE);
      }
    }
      if (inputString.startsWith("f") == true) {
/* if line starts with an F tag (contains information about
the video frame, which allows the system to relate a
specific video frame to participant data) */

        String frameNumber = "";
        String currentLine =
(String)inputVector.get(lineNum);
        StringTokenizer lineTokenizer = new
StringTokenizer(currentLine,"\t");
        if (lineTokenizer.hasMoreTokens()){ lineToken =
lineTokenizer.nextToken();}
        if (lineTokenizer.hasMoreTokens()){
frameNumber =  lineTokenizer.nextToken();}
//extract frame number
        if (lineTokenizer.hasMoreTokens() == false){
          videoFrame frame = new
videoFrame(Integer.parseInt(frameNumber));
//create new frame and add to arrayList
          framevalues.add(frame);
          xyCoordinates coord = new
xyCoordinates(0,"NEW FRAME",0,0);
//create and add marker coordinate object to arrayList
          coordinatevalues.add(coord);
          //System.out.println(currentLine+" - INPUT
CONFIRMED");
        } else {
          JOptionPane.showMessageDialog(null,"The
frame object loaded
incorrectly.","ERROR",JOptionPane.WARNING_MES
SAGE);
        }
      }
      if (inputString.startsWith("c") == true) {
/* if line starts with an C tag (contains information
concerning participant number, participant order, as well
as X and Y coordinate values) */
        String participant = "";
        String order = "";
        String xCoord = "";
        String yCoord = "";
        String currentLine =
(String)inputVector.get(lineNum);
        StringTokenizer lineTokenizer = new
StringTokenizer(currentLine,"\t");
        if (lineTokenizer.hasMoreTokens()){ lineToken =
lineTokenizer.nextToken();}
         if (lineTokenizer.hasMoreTokens()){ participant
=  lineTokenizer.nextToken();}
//extract participant number
         if (lineTokenizer.hasMoreTokens()){ order =
lineTokenizer.nextToken();}
//extract order information
         if (lineTokenizer.hasMoreTokens()){ xCoord =
lineTokenizer.nextToken();}
//extract x coordinate data
         if (lineTokenizer.hasMoreTokens()){ yCoord =
lineTokenizer.nextToken();}
// extract y coordinate data
         if (lineTokenizer.hasMoreTokens() == false){
           xyCoordinates coord = new
xyCoordinates(Integer.parseInt(participant),
           order, Integer.parseInt(xCoord),
Integer.parseInt(yCoord));
//create new coordinate object and add to arrayList
           coordinatevalues.add(coord);
           //System.out.println(currentLine+" - INPUT
CONFIRMED");
         } else {
           JOptionPane.showMessageDialog(null,"The
frame object loaded
incorrectly.","ERROR",JOptionPane.WARNING_MES
SAGE);
         }
       }
   }

/* The following is a recursive loop that iterates through
the object structure adding relevant objects until a
marker is found – which implies that following data is
from another branch of the tree structure.*/
   int videocount=1;
   int ratecount=1;
   int framecount=1;
   int coordinatecount=1;
   topData top = new topData();
// create new topData
   for (int experimentcount=0;
experimentcount<experimentvalues.size();
experimentcount++) {
/* for all experiments in arrayList, create experiment
object */
     experiment exp =
((experiment)experimentvalues.get(experimentcount));
     top.add(exp);
// take add to topData object
     System.out.println(exp.description());

while((((video)videovalues.get(videocount)).getVideoLen
gth()>0) && ((videocount)<videovalues.size())){
/*while considering appropriate video (i.e. before the
marker)*/
       video vid = ((video)videovalues.get(videocount));
//create new video
       exp.add(vid);
//add to the experient.
       System.out.println(vid.description());
while((((frameRate)ratevalues.get(ratecount)).getFrameRa
te()>0)
         && ((ratecount)<ratevalues.size())){
/*while considering appropriate frame rate (i.e. before
the marker)*/
         frameRate rate =
((frameRate)ratevalues.get(ratecount));
//create new framerate object
         vid.add(rate);
//add to the video object
         System.out.println(rate.description());
```

```
while((((videoFrame)framevalues.get(framecount)).getFra
meNumber()>0)
        && ((framecount)<framevalues.size())){
/*while considering appropriate frame (i.e. before the
marker)*/
          videoFrame frame =
((videoFrame)framevalues.get(framecount));
//create new frame object from arrayList
          rate.add(frame);
//add to the framerate object
          System.out.println(frame.description());
while(((xyCoordinates)coordinatevalues.get(coordinateco
unt)).getParticipant()>0){
/*while considering appropriate coordinate objects (i.e.
before the marker object)*/
             xyCoordinates coord =
((xyCoordinates)coordinatevalues.get(coordinatecount));
//create new XYCoordinate object from arrayList

/* the following code updates the state of videocount,
ratecount, framecount and coordinatecount, which
defines the current position in building up the object
structure. */
                    if
((coordinatecount)<coordinatevalues.size());{frame.add(c
oord);};
             System.out.println(coord.description());
             if
((coordinatecount)<coordinatevalues.size()){coordinatec
ount++;}
             if (coordinatecount ==
(coordinatevalues.size())) {break;};
            }
         if ((coordinatecount)<coordinatevalues.size())
{coordinatecount++;}
         if ((framecount)<framevalues.size())
{framecount++;}
         if (framecount == (framevalues.size())) {break;};
        }
        if ((framecount)<framevalues.size())
{framecount++;}
        if ((ratecount)<ratevalues.size()) {ratecount++;}
        if (ratecount == (ratevalues.size())) {break;};
      }
      if ((ratecount)<ratevalues.size()) {ratecount++;}
      if ((videocount)<videovalues.size())
{videocount++;}
      if (videocount == (videovalues.size())) {break;};
    }
    if ((videocount)<videovalues.size()) {videocount++;}
  }
  dataStore = top;
// writes temporary top to main dataStore
}
  public topData getTopDataList()
// runs load() and returns full topData object
  {
    load();
    return dataStore;
  }
}
```

testloadData.java – used to facilitate loadData.java

```
package definingroi;
/* testloadData is a runnable program that test the
loadData class. */
import java.util.*;
import java.io.*;
import java.awt.*;

class testloadData {
  public static void main(String[] args) {

   //extracts data and saves to file
    loadData loading = new loadData();
    topData top = new topData();
    top = loading.getTopDataList();
  }
}
```