9. Managing the Display

In this chapter

- Managing multiple display systems, including stereo
- Getting and modifying screen characteristics in FSEM

9.1. Multiple Displays

Each display connected your computer is represented by a GraphicsDevice object. getScreenDevices() method of GraphicsEnvironment class returns a list of all available displays

```
private static GraphicsEnvironment gEnvironment =
    GraphicsEnvironment.getLocalGraphicsEnvironment();
private static GraphicsDevice[] gDevices =
    gEnvironment.getScreenDevices();
```

having a list of available graphics devices allows us to implement a new constructor, which takes an integer argument representing the display id

```
public FullScreen2(int displayID) {
  super(gDevices[displayID].getDefaultConfiguration());
  //...
}
```

now an application can construct a FullScreen2 object on a particular display by invoking this constructor

```
int displayID = 1;
FullScreen2 fs = new FullScreen2(displayID);
```

If displayID+1 is larger than the number of available displays the constructor throws an ArrayIndexOutOf-Bound exception and terminates.

With multiple displays, the JFrame is not automatically palced on the display which switches to FSEM. It does have the correct dimensions but by default it is placed at (0,0), which is the upper left cornert of the default display device. To correctly place it on the display which swithed to FSEM we first get the coordinates of that display

```
gDevice = gDevices[displayID];
gConfiguration = gDevice.getDefaultConfiguration();
Rectangle gcBounds = gConfiguration.getBounds();
```

```
    int xoffs = gcBounds.x;
    int yoffs = gcBounds.y;
    int width = gcBounds.width;
    int height = gcBounds.height;
```

then use those coordinates to place the JFrame in correct position

```
    setBounds(xoffs, yoffs, width, height);
```

This line must come after gDevice.setFullScreenWindow(this), that is after swithing to FSEM.

There should still be a default constructor (Java insists on the default constructor - the constructor with no arguments - if your class has a non-default constructor). It is best that the default constructor uses the first available screen, displayID = 0, this way all the former examples of the book will still work with this new version of FullScreen class

```
    public FullScreen2() {
      this(0);
    }
```

Note for MacOS X: By default FSEM on a multi-display OS X system captures all displays, i.e. all displays turn blank once the FSEM is initiated. You can adjust this behavior and tell Java Virtual Machine not to capture all displays. You can add the following line in your code

```
    System.setProperty("apple.awt.fullscreencapturealldisplays”,” false”);
```

for instance in your main() method, or you can run your program with the following switch

```
    java –Dapple.awt.fullscreencapturealldisplays=false MyProgram
```

9.2. Screen characteristics

The simple example HelloPsychophysicist of Chapter 2 used default screen characteristics. But in FSEM you could do more, you could, for example change the screen resolution. In this section I will show how to safely change the characteristics of the secreen.

Why would you change the screen parameters? One reason is performance: rendering and presenting a 512 by 512 image on a 1280x960 screen is probably faster than rendering and presenting a 1024x1024 image on a 1920x1200 screen. Moreover, monitors usually have higher temporal resolution (refresh rate) at lower spatial resolutions. Another reason might be getting the desired visual angle in an experiment in situations where you cannot adjust the distance between the observer and the computer screen. Having said that, it is doubtful whether or not letting your experimental program adjust the screen characteristics is a very bright idea. A word of caution deserves space here: better use your OS's dedicated applications to adjust the screen characteristics on the computers that you run your experiments. Better yet, do it permanantly so that there is little room for making errors between different sessions of the same experiment.

You can set or get the width, height, bitDepth and refreshRate of our display through a DisplayMode object. However, before attempting to change the DisplayMode, you should first check whether or not FSEM and DisplayMode change is available by using the GraphicsDevice's isFullScreenSupported() and isDisplay-ChangeSupported() methods. They return true if the qurried functionality is supported, false if not. If FSEM or DisplayMode change is not available you can't do much other than using the current native mode. If FSEM and DisplayMode change is avialable there are probably more than one available DisplayModes, you can get a list of available DisplayModes by invoking the getDisplayModes() method of the GraphicsDevice class. But this wouldn't be a very useful piece of information to inspect, because you can't print it out. Here is a more eye-friendly way to see the available display modes

```
public String[] reportDisplayModes() {
  DisplayMode[] dms = getDisplayModes();
  String[] message = new String[dms.length];
  for (int i = 0; i < dms.length; i++) {
    StringBuilder m = new StringBuilder("("
        + dms[i].getWidth() + ","
        + dms[i].getHeight() + ","
        + dms[i].getBitDepth() + ","
        + dms[i].getRefreshRate() + ") ");
    message[i] = m.toString();
  }
  return message;
}
```

this method returns all the available display modes in a String array, wich can easily be printed. The getBitRate() returns BIT_DEPTH_MULTI if this mode supports multiple bit depths, and getRefreshRate() returns REFRESH_RATE_UNKNOWN if refreshRate is unknown or unconfigurable.

Tip: Why using StringBuilder, why not String? A String object is *immutable*, which means that we cannot modify a String object once it is created, for example we cannot append any more characters to an existing String. It is designed in this way for better performance. But a StringBuilder can be modified after it is constructed.

To change the DisplayMode, java API provides GraphicsDevice.setDisplayMode() method. However this method is known to lack robustness. I will instead use the following more robust method to change the DisplayMode

```
public void setDisplayMode(DisplayMode dm) {
  if(displayMode.equals(dm))
    return;
  else if (!gDevice.isDisplayChangeSupported() || dm == null) {
    System.err.println("Exception in FullScreen.setDisplayMode(): "
        + "Display Change not Supported or DisplayMode is null");
    closeScreen();
    System.exit(0);
  }
  else if (!isDisplayModeAvailable(dm)) {
    System.err.println("Exception in FullScreen.setDisplayMode(): "
        + "DisplayMode not available");
    System.err.println(" ");
    System.err.println("Supported DisplayModes are:");
    String[] dms = reportDisplayModes();
    for (int i = 0; i < dms.length; i++) {
      System.err.print(dms[i]);
      if ((i + 1) % 4 == 0)
        System.err.println();
    }
    closeScreen();
    System.exit(0);
  }
  else {
    try {
      gDevice.setDisplayMode(dm);
      displayMode = dm;
      setBounds(0, 0, dm.getWidth(), dm.getHeight());
      Thread.sleep(200);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    } catch (RuntimeException e) {
      System.err.println("Exception in FullScreen.setDisplayMode(): "
          + "DisplayMode not available or " + "Display Change not Supported");
      System.err.println("");
      System.err.println("Supported DisplayModes are:");
      String[] dms = reportDisplayModes();
      for (int i = 0; i < dms.length; i++) {
        System.err.println(dms[i]);
        if ((i + 1) % 4 == 0)
          System.err.println();
      }
      closeScreen();
      System.exit(0);
    }
  }
}
```

The method first compares the current DisplayMode to the requested one, if they are equal it returns. Next the method checks whether or not display mode change is available at all. If it isn't available it terminates the program. I chose to terminate the program in case the method fails to change to the requested DisplayMode, because you wouldn't want to present the stimulus in a wrong resolution by mistake in an actual experiment. After that we check whether the requested DisplayMode is among the available ones with a call to the following isDisplayModeAvailable() method

```
  private boolean isDisplayModeAvailable(DisplayMode dm) {

    DisplayMode[] mds = gDevice.getDisplayModes();
    for (int i = 0; i < mds.length; i++) {
      if (mds[i].getWidth() == dm.getWidth()
          && mds[i].getHeight() == dm.getHeight()
          && mds[i].getBitDepth() == dm.getBitDepth()
          && mds[i].getRefreshRate() == dm.getRefreshRate())
        return true;
    }
    return false;
  }
```

If the requested DisplayMode is not among the available ones, we warn the user, produce a list of available DisplayModes, then terminate. Finally we invoke the GraphicsDevice.setDisplayMode() method, and wait for one second, to let that method finish its job - much like in the createBufferStrategy() method we came across in Chapter 2. Nothe how I set the dimensions of FullScreen (JFrame) to fit exactly those of the new DisplayMode by invoking the setBounds() method of JFrame. The GraphicsDevice.setDisplayMode() method may throw a RuntimeException. We also catch this exception and terminate the program after printing out a list of available DisplayModes.

Finally I will include another getter to get the current DisplayMode

```
public DisplayMode getDisplayMode() {
  return displayMode;
}
```

and its acompanying reportDisplayMode() method

9.3. Stereo display systems

Coming soon...

Here is the entire FullScreen3 class

```
/*
 * chapter 9: FullScreen3.java
 *
 * Provides methods to manage (multiple) displays
 *
 */
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.font.FontRenderContext;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;
import javax.swing.JFrame;
public class FullScreen3 extends JFrame implements KeyListener{
  private static final GraphicsEnvironment gEnvironment = GraphicsEnvironment
      .getLocalGraphicsEnvironment();
  private static final GraphicsDevice[] gDevices = gEnvironment
      .getScreenDevices();
  private GraphicsDevice gDevice;
  private GraphicsConfiguration gConfiguration;

  private int nBuffers = 1;
  private Color bgColor = Color.BLACK;
  private Color fgColor = Color.LIGHT_GRAY;
  private final Font defaultFont = new Font("SansSerif", Font.BOLD, 36);
  private DisplayMode oldDisplayMode;
  private DisplayMode displayMode;

  private BlockingQueue<String> keyTyped;
  private BlockingQueue<Long> whenKeyTyped;
  private BlockingQueue<Integer> keyPressed;
  private BlockingQueue<Long> whenKeyPressed;
  private BlockingQueue<Integer> keyReleased;
  private BlockingQueue<Long> whenKeyReleased;

  public void keyTyped(KeyEvent ke) {
    keyTyped.offer(String.valueOf(ke.getKeyChar()));
    whenKeyTyped.offer(ke.getWhen());
  }
  public void keyReleased(KeyEvent ke) {

    keyReleased.offer(ke.getKeyCode());
    whenKeyReleased.offer(ke.getWhen());
  }

  public void keyPressed(KeyEvent ke) {

    if (ke.getKeyCode() == KeyEvent.VK_ESCAPE) {
      closeScreen();
      System.exit(0);
    }
    else {
      keyPressed.offer(ke.getKeyCode());
      whenKeyPressed.offer(ke.getWhen());
    }
  }

  public FullScreen3() {
    this(0);
  }

  public FullScreen3(int displayID) {
    super(gDevices[displayID].getDefaultConfiguration());
    gDevice = gDevices[displayID];
    gConfiguration = gDevice.getDefaultConfiguration();
    oldDisplayMode = gDevice.getDisplayMode();
    displayMode = gDevice.getDisplayMode();
    try {
      setUndecorated(true);
      setIgnoreRepaint(true);
      setResizable(false);
      setFont(defaultFont);
```

```
      setBackground(bgColor);
      setForeground(fgColor);
      gDevice.setFullScreenWindow(this);

      Rectangle gcBounds = gConfiguration.getBounds();
      int xoffs = gcBounds.x;
      int yoffs = gcBounds.y;
      int width = gcBounds.width;
      int height = gcBounds.height;
      setBounds(xoffs, yoffs, width, height);

      keyTyped = new LinkedBlockingQueue<String>();
      whenKeyTyped = new LinkedBlockingQueue<Long>();
      keyPressed = new LinkedBlockingQueue<Integer>();
      whenKeyPressed = new LinkedBlockingQueue<Long>();
      keyReleased = new LinkedBlockingQueue<Integer>();
      whenKeyReleased = new LinkedBlockingQueue<Long>();
      addKeyListener(this);

      setNBuffers(nBuffers);
    }
    finally {}
  }
  public void setNBuffers(int n) {
    try {
      createBufferStrategy(n);
      nBuffers = n;
    } catch (IllegalArgumentException e) {
      System.err.println(
          "Exception in FullScreen.setNBuffers(): "
          +"requested number of Buffers is illegal - falling back to default");
      createBufferStrategy(nBuffers);
    }
    try{
      Thread.sleep(200);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
    }
  }
  public int getNBuffers() {
    return nBuffers;
  }
  public void updateScreen() {

    if (getBufferStrategy().contentsLost())
      setNBuffers(nBuffers);
    getBufferStrategy().show();
  }
```

```
public void displayImage(BufferedImage bi) {
  if( bi!=null){
    double x = (getWidth() - bi.getWidth()) / 2;
    double y = (getHeight() - bi.getHeight()) / 2;
    displayImage((int) x, (int) y, bi);
  }
}
public void displayImage(int x, int y, BufferedImage bi) {
  Graphics2D g =  (Graphics2D)getBufferStrategy().getDrawGraphics();
  try {
    if(g!=null && bi!=null)
      g.drawImage(bi, x, y, null);
  }
  finally {
    g.dispose();
  }
}
public void displayText(String text) {
  Graphics2D g = (Graphics2D)getBufferStrategy().getDrawGraphics();
  if( g!=null && text!= null){
    Font font = getFont();
    g.setFont(font);
    FontRenderContext context = g.getFontRenderContext();
    Rectangle2D bounds = font.getStringBounds(text, context);
    double x = (getWidth() - bounds.getWidth()) / 2;
    double y = (getHeight() - bounds.getHeight()) / 2;
    double ascent = -bounds.getY();
    double baseY = y + ascent;
    displayText((int) x, (int) baseY, text);
  }
  g.dispose();
}
public void displayText(int x, int y, String text) {
  Graphics2D g =  (Graphics2D)getBufferStrategy().getDrawGraphics();
  try {
    if(g!=null && text!=null){
      g.setFont(getFont());
      g.setColor(getForeground());
      g.drawString(text, x, y);
    }
  }
  finally {
    g.dispose();
  }
}
public void blankScreen() {
  Graphics2D g = (Graphics2D)getBufferStrategy().getDrawGraphics();
  try {
    if(g!=null){
      g.setColor(getBackground());
      g.fillRect(0, 0, getWidth(), getHeight());
    }
  }
  finally {
    g.dispose();
  }
}

public Color getBackground(){

  return bgColor;
}

public void setBackground(Color bg){

  bgColor = bg;
}

public void hideCursor() {
  Cursor noCursor = null;
  Toolkit tk = Toolkit.getDefaultToolkit();
  Dimension d = tk.getBestCursorSize(1,1);
  if((d.width|d.height)!=0)
    noCursor = tk.createCustomCursor(
        gConfiguration.createCompatibleImage(d.width, d.height),
        new Point(0, 0), "noCursor");
  setCursor(noCursor);
}

public void showCursor() {
  setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
}

public void closeScreen() {
  if (gDevice.isDisplayChangeSupported())
    setDisplayMode(oldDisplayMode);
  gDevice.setFullScreenWindow(null);
  dispose();
}

public String getKeyTyped(long ms){

  String c = null;
  try {
    if(ms < 0)
```

```
      c = keyTyped.take();
    else
      c = keyTyped.poll(ms, TimeUnit.MILLISECONDS);
  } catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    e.printStackTrace();
  }
  return c;
}

public String getKeyTyped(){

  return keyTyped.poll();
}

public Long getWhenKeyTyped(){

  return whenKeyTyped.poll();
}

public void flushKeyTyped(){

  keyTyped.clear();
  whenKeyTyped.clear();
}

public Integer getKeyPressed(long ms){

  Integer c = null;
  try {
    if(ms < 0)
      c = keyPressed.take();
    else
      c = keyPressed.poll(ms, TimeUnit.MILLISECONDS);
  } catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    e.printStackTrace();
  }
  return c;
}

public Integer getKeyPressed(){

  return keyPressed.poll();
}

public Long getWhenKeyPressed(){
  return whenKeyPressed.poll();
}

public void flushKeyPressed(){

  keyPressed.clear();
  whenKeyPressed.clear();
}

public Integer getKeyReleased(long ms){

  Integer c = null;
  try {
    if(ms < 0 )
      c = keyReleased.take();
    else
      c = keyReleased.poll(ms, TimeUnit.MILLISECONDS);
  } catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    e.printStackTrace();
  }
  return c;
}

public Integer getKeyReleased(){

  return keyReleased.poll();
}

public Long getWhenKeyReleased(){

  return whenKeyReleased.poll();
}

public void flushKeyReleased(){

  keyReleased.clear();
  whenKeyReleased.clear();
}
public boolean isFullScreenSupported() {
  return gDevice.isFullScreenSupported();
}
public void setDisplayMode(DisplayMode dm) {
  if(displayMode.equals(dm))
    return;
  else if (!gDevice.isDisplayChangeSupported() || dm == null) {
    System.err.println("Exception in FullScreen.setDisplayMode(): "
        + "Display Change not Supported or DisplayMode is null");
      closeScreen();
      System.exit(0);
    }
    else if (!isDisplayModeAvailable(dm)) {
      System.err.println("Exception in FullScreen.setDisplayMode(): "
          + "DisplayMode not available");
      System.err.println(" ");
      System.err.println("Supported DisplayModes are:");
      String[] dms = reportDisplayModes();
      for (int i = 0; i < dms.length; i++) {
        System.err.print(dms[i]);
        if ((i + 1) % 4 == 0)
          System.err.println();
      }
      closeScreen();
      System.exit(0);
    }
    else {
      try {
        gDevice.setDisplayMode(dm);
        displayMode = dm;
        setBounds(0, 0, dm.getWidth(), dm.getHeight());
        Thread.sleep(200);
      } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
      } catch (RuntimeException e) {
        System.err.println("Exception in FullScreen.setDisplayMode(): "
            + "DisplayMode not available or " + "Display Change not Supported");
        System.err.println("");
        System.err.println("Supported DisplayModes are:");
        String[] dms = reportDisplayModes();
        for (int i = 0; i < dms.length; i++) {
          System.err.println(dms[i]);
          if ((i + 1) % 4 == 0)
            System.err.println();
        }
        closeScreen();
        System.exit(0);
      }
    }
  }
  private boolean isDisplayModeAvailable(DisplayMode dm) {
    DisplayMode[] mds = gDevice.getDisplayModes();
    for (int i = 0; i < mds.length; i++) {
      if (mds[i].getWidth() == dm.getWidth()
          && mds[i].getHeight() == dm.getHeight()
          && mds[i].getBitDepth() == dm.getBitDepth()
          && mds[i].getRefreshRate() == dm.getRefreshRate())
        return true;
      }
      return false;
    }
    public String reportDisplayMode() {
      StringBuilder message = new StringBuilder();
      if (displayMode.getBitDepth() == DisplayMode.BIT_DEPTH_MULTI)
        message.append("  Bit Depth = -1 (MULTIPLE) \n");
      else
        message.append("  Bit Depth = " + displayMode.getBitDepth() + "\n");
      message.append("  Width = " + displayMode.getWidth() + "\n");
      message.append("  Height = " + displayMode.getHeight() + "\n");
      if (displayMode.getRefreshRate() == DisplayMode.REFRESH_RATE_UNKNOWN)
        message.append("  Refresh Rate = 0 (unknown/unmodifiable) \n");
      else
        message.append("  Refresh Rate = " + displayMode.getRefreshRate() + "\n");
      return message.toString();
    }
    public DisplayMode getDisplayMode() {
      return displayMode;
    }
    public boolean isDisplayChangeSupported() {
      return gDevice.isDisplayChangeSupported();
    }
    public String[] reportDisplayModes() {
      DisplayMode[] dms = getDisplayModes();
      String[] message = new String[dms.length];
      for (int i = 0; i < dms.length; i++) {
        StringBuilder m = new StringBuilder("(" + dms[i].getWidth() + ","
            + dms[i].getHeight() + "," + dms[i].getBitDepth() + ","
            + dms[i].getRefreshRate() + ") ");
        message[i] = m.toString();
      }
      return message;
    }
    public DisplayMode[] getDisplayModes() {
      return gDevice.getDisplayModes();
    }
  }
```

9.4. Examples

Here is a sample test program, which allows user set the DisplayMode safely

```
/*
 * chapter 3: DisplayTest.java
 *
 * Tests FSEM and DisplayMode change support, if supported lets user set the
 * DisplayMode
 *
 */
import java.awt.DisplayMode;
import java.awt.GraphicsEnvironment;
import java.util.StringTokenizer;
import javax.swing.JOptionPane;

public class DisplayTest {

  public static int dialog(String message, String[] options) {
    int selectedValue = JOptionPane.showOptionDialog(null, message,
        "Information", JOptionPane.DEFAULT_OPTION,
        JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);
    return selectedValue;
  }

  public static void dialog(String message) {

    JOptionPane.showMessageDialog(null, message);
  }

  public static String inputDialog(String message, String[] options) {

    String s = (String) JOptionPane.showInputDialog(null, message,
        "Query Dialog", JOptionPane.PLAIN_MESSAGE, null, options, options[0]);
    return s;
  }

  public static void main(String[] args) {

    if (dialog("Now we will start diagnosing the Display(s) \n"
        + "Are you ready?", new String[] { "YES", "NO/EXIT" }) == 1)
      System.exit(0);
    int numOfDisplays = GraphicsEnvironment.getLocalGraphicsEnvironment()
        .getScreenDevices().length;
    dialog("Found " + numOfDisplays + " display(s)");
    FullScreen2[] fs = new FullScreen2[numOfDisplays];
    for (int j=0; j< numOfDisplays; j++) {
      fs[j] = new FullScreen2(j);
      try {
        fs[j].displayText("Display: " + j);
        Thread.sleep(2000);
        fs[j].blankScreen();
        fs[j].displayText("Bounds: " + fs[j].getBounds().x + " "
            + fs[j].getBounds().y + " " + fs[j].getBounds().width + " "
            + fs[j].getBounds().height );
        Thread.sleep(2000);
        boolean supported = fs[j].isDisplayChangeSupported();
        fs[j].closeScreen();
        if (!fs[j].isFullScreenSupported())
          dialog("Full Screen Exclusive Mode is not supported for Display "
              + j + "\n"
              + "Java uses alternative methods to imitate Full Screen Mode");
        else
          dialog("Full Screen Exclusive Mode is supported for Display " + j);
        dialog("Current DisplayMode is: \n" + fs[j].reportDisplayMode());
        if (supported) {
          String[] dms = fs[j].reportDisplayModes();
          StringBuffer message = new StringBuffer();
          for (int i = 0; i < dms.length; i++) {
            message.append(dms[i]);
            if ((i + 1) % 4 == 0)
              message.append("\n");
          }
          dialog("DisplayMode change is supported for Display " + j +"\n"
              + "Available Modes are:\n" + message);
          String srep = inputDialog("Choose DisplayMode or click on Cancel",
              dms);
          if (srep != null) {
            StringTokenizer rep = new StringTokenizer(srep, "(,)");
            int[] dp = new int[4];
            for (int k = 0; k < 4; k++)
              dp[k] = Integer.parseInt(rep.nextToken());
            DisplayMode dm = new DisplayMode(dp[0], dp[1], dp[2], dp[3]);
            fs[j] = new FullScreen2();
            fs[j].setDisplayMode(dm);
            fs[j].blankScreen();
            fs[j].displayText("Display: " + j);
            fs[j].updateScreen();
            Thread.sleep(2000);
            fs[j].blankScreen();
            fs[j].displayText("Bounds: " + fs[j].getBounds().x + " "
                + fs[j].getBounds().y + " " + fs[j].getBounds().width + " "
                + fs[j].getBounds().height );
            fs[j].updateScreen();
            Thread.sleep(2000);
          }
        }
        else
          dialog("DisplayMode change is not supported for Display " + j);
      } catch (InterruptedException e) {}
      finally {
        fs[j].closeScreen();
      }
```

```
            }
        }
    }
```

In this program I use some interesting features, which allow interaction with the user. They are implemented in dialog() and inputDialog() methods. Java's Swing API provides simple tools to interact with users through the JOPtionPane class. (See "How to make dialogs" chapter in "Trail: Creating a GUI with JFC/Swing" at `http://java.sun.com/docs/books/tutorial/uiswing/components/dialog.html`.)

We first determine whether FSEM and DisplayMode change are supported by invoking isFullScreenSupported() and isDisplayChangeSupported() methods. If DisplayMode change is supported we show a list of available modes in a dialog window. Then user can choose one of the available modes and the screen switches to that mode with a call to the setDisplayMode() method.

Another useful object in this example is StringTokenizer(String text, String delim) method. It breaks the given String text into smaller tokens stopping at any one of the given delimeters. Here our delimeters are "(", ")" and ",". This way we first get user's preferred DisplayMode as a String and by stripping it out of the delimeters we construct a new DisplayMode object.

9.5. Summary

- If you have a multiple display system you can choose the one to use to display your stimulus in full screen exclusive mode. Use the constructor FullScreen(int displayID) to choose the screen to display your stimulus. (Use DisplayTest.java to figure out display id's in a multi-display system.)
- For display characteristics use
  - isFullScreenSupported() method to learn whether Full Screen Exclusive Mode (FSEM) is supported,
  - reportDisplayMode() (or getDisplayMode()) method to learn your current display mode (width, height, bit depth, refresh rate),
  - isDisplayChangeSupported() method to learn whether you can change your display's characteristics,
  - reportDisplayModes() (or getDisplayModes()) method to see all available DisplayModes.
- Use setDisplayMode() method to change the DisplayMode in FSEM.