CSCD 370 - GUI Programming with JavaFX

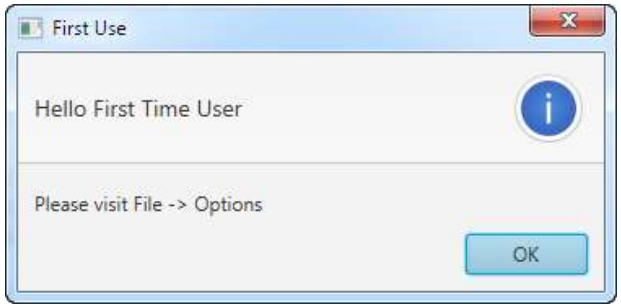
Module 8: Preferences, Clipboard, Drag and Drop

Preferences API

- **Provides persistent storage and retrieval of application settings**
  - two preference trees: system (all users) and user
  - data is stored in an implementation-dependent location
  - which for Windows is in the registry (I know, ick)
  - export/import to/from XML is supported so that preferences can be backed up or ported to other platforms
- **Example use: first run detection**

what do you suppose this is about?

```
Preferences prefs = Preferences.userNodeForPackage(getClass());
if (prefs.getBoolean("FirstRun", true) == true) {
   Alert alert = new Alert(Alert.AlertType.INFORMATION) ;
   alert.setTitle("First Use") ;
   alert.setHeaderText("Hello First Time User");
   alert.setContentText(
         "Please visit File -> Options");
   alert.showAndWait() ;
   prefs.putBoolean("FirstRun", false);
}
```

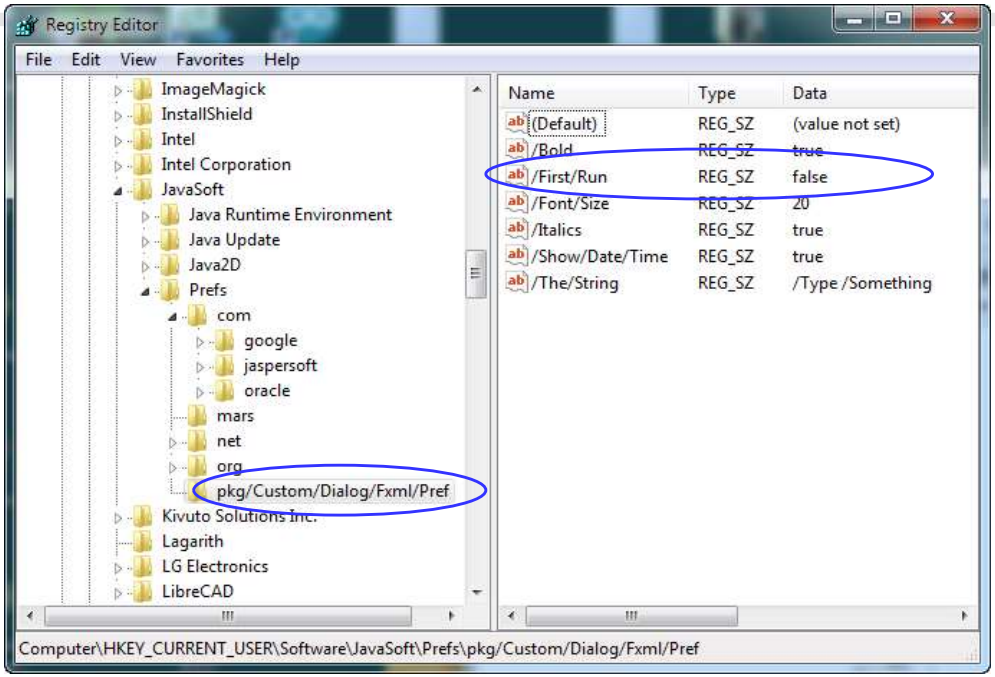


Data Types and Tree Access

- **The usual primitive data types are supported**
  - **String: put(), get()**
  - **putBoolean(), getBoolean()**
  - **putByteArray(), getByteArray()**
  - **putDouble(), getDouble()**
  - **putFloat(), getFloat()**
  - **putInt(), getInt()**
  - **putLong(), getLong()**
- **Also note:**
  - **systemNodeForPackage():** Returns a node in the system preference tree corresponding to the passed object's package
  - **userNodeForPackage():** Returns a node in the user preference tree corresponding to the passed object's package
  - **systemRoot():** Returns the root of the system preferences tree
  - **userRoot():** Returns the root of the user preferences tree

Location on Windows

- **Registry:**
  - HKEY_CURRENT_USER/ Software/JavaSoft/Prefs/ (your package name)
  - note how camelCase values appear



- **For system-level Preferences:**

```
Preferences prefs = Preferences.systemNodeForPackage(getClass());
```

  - will fail (not crash) when running from a user account
  - requires root/admin privileges, and there is no "run as administrator" option for jar files
  - you can run a command prompt as admin, and then: java –jar xxx.jar
  - or package it as an .exe (with e.g., launch4j), and then "run as administrator"

Location On Other Platforms

- On Linux (~ is the user's root diretory):

  ~/.java/.userPrefs/<pkgName>/prefs.xml

- On Mac (~ is the user's root directory):

  ~/Library/Preferences/com.apple.java.util.prefs/prefs.plist

  – which is an XML file with java preferences for each app:

```
<key> MyLab7/ </key>
<dict>
    <key>FirstRun</key>
    <string>false</key>
    ...
</dict>
```

- On Commodore 64:

  – yes, Java HAS been ported to the Commodore 64 (see b2fJ)

  – it doesn't appear to have Preferences though

Preference Change Listener

- Fires from in-app changes only
  - i.e., not external registry edits
  - in my experience this event handler is not executed on the UI thread, so I have had to use a runLater() call to do any UI manipulation from here

```
prefs.addPreferenceChangeListener(
            prefChangeEvent -> onPrefChanged(prefChangeEvent));
...
// this handler does NOT run on the application (UI) thread,
// so execute a runLater if you need to manipulate the UI
private void onPrefChanged(PreferenceChangeEvent pce) {
   Platform.runLater(new Runnable() {
      @Override public void run() {
          // TODO: modify UI in response to preferences changes
      }
   });
}
```

Preference Change Listener

- **If you need to know *which* preference was changed**
  - **the PreferenceChangeEvent only gives strings for the new value**
  - **but you can also use get methods on the Preferences object**

```
Preferences prefs = Preferences.userNodeForPackage(getClass());
switch (pce.getKey()) {
    // TODO: react to key-specific changes...
    case "SomeKey":
        int newValue1 = Integer.parseInt(pce.getNewValue()) ;
        int newValue2 = prefs.getInt("SomeKey") ;
        ...
}
```

Passed into the change listener

- **The Preferences API does not handle user interaction**
  - **you can build a Dialog for that, either a customized Alert or a custom extension of Dialog**
  - **in which case the dialog launcher no longer needs to obtain a Result from the Dialog**
  - **instead, you can have the Dialog change the preferences, and have the launching class install a Preference change listener (previous slide)**
  - **Then "fire and forget" the Dialog**

Export / Import

- Storage of preferences is platform-dependent
  - you can export to and import from XML (See Lab7DialogPrefModelessSingle)
  - prefs.exportNode(OutputStream): export the current Node (all preferences)
  - exportSubtree(OutputStream): export this Node and any descendents
  - importPreferences(InputStream): import all preferences represented by the XML document in the stream
- This is intended to support the movement of preferences between platforms
  - You cannot tell Java to use an XML file on Windows - it will always use the registry for Preferences
  - I suppose you could export to XML on app start, manipulate preferences in your own XML code, and import the XML to the registry on app close (ugh?)

System Clipboard

- **Clipboard transfers and drag and drop are substantially simplified since Swing**
- **The clipboard is not just for text**
  - **it supports several standard (MIME) data types, including text, images, audio, and video**
  - **more importantly, it allows custom (app-specific) data types**
  - **custom data types are placed on the clipboard with an associated descriptor called a DataFormat**
  - **a custom data type is a Serializable class that encapsulates the data you want to place or retrieve with a single put or get**

Some example standard MIME types: application/octet-stream, text/plain, text/css, text/html, text/javascript, image/bmp, image/gif, image/png, image/jpeg, image/tiff, audio/wav, audio/ogg, audio/mpeg, video/mp4, video/ogg,

Clipboard, Dragboard Classes

- **Clipboard**
  - represents the operating system clipboard

```
Clipboard clipboard = Clipboard.getSystemClipboard();
```

- **Dragboard**
  - a clipboard that is specific to drag and drop
  - returned by Node.startDragAndDrop()
  - supports drag and drop both within app and between apps
- **ClipboardContent**
  - this class defines content for a Clipboard or Dragboard
  - it supports File Lists, HTML, Images, RTF, plain Strings, and URLs
  - you can also define your own DataFormat class as long as it is based on a Serializable class

Putting Data on the Clipboard

- **Example of putting Standard Content**

```
Clipboard clipboard = Clipboard.getSystemClipboard();
ClipboardContent content = new ClipboardContent() ;
content.putString("some text") ;
content.putHtml("<i>Some</i> text");
clipboard.setContent(content) ;
```

- **Example of putting Custom Content**

```
Clipboard clipboard = Clipboard.getSystemClipboard();
ClipboardContent content = new ClipboardContent() ;
DataFormat df = new DataFormat("cscd370/mydata") ;
content.put(df, myData) ;
content.putString("in addition to this string, there is
                              also custom content") ;
clipboard.setContent(content) ;
...
Class MyData implements Serializable
{
   // TODO: class must be Serializable
}
```

this is intended to be a string unique to your app

Getting Data from the Clipboard

- **Example of getting Standard Content**

```
Clipboard clipboard = Clipboard.getSystemClipboard();
if (clipboard.hasString()) {
   String aString = clipboard.getString() ;
   TODO: do something with it
}
else // TODO: maybe check for other content type as well
```

- **Example of getting Custom Content**

```
DataFormat df = new DataFormat("cscd370/mydata") ;
if (clipboard.hasContent(df)) {
   MyData myData = (MyData)clipboard.getContent(df) ;
   TODO: do something with it
}
else // TODO: maybe check for other content type as well
...
Class MyData implements Serializable
{
   // TODO: class must be Serializable
}
```

Drag and Drop

- **Any Node can participate in Drag and Drop**
- **To allow a Node to be a Drag source:**
  - **register an OnDragDetected listener on the Node - that listener receives a MouseEvent**
  - ***you can use the same listener for multiple drag sources* (e.g., game pieces) - the source of the Drag (which game piece is being dragged) can be obtained from mouseEvent.getSource()**
  - **obtain a Dragboard by calling startDragAndDrop(…) on the source, passing it a TransferMode (.COPY, .MOVE, or .LINK)**
  - **put desired data content onto the Dragboard**
- **The default Drag shadow is a special cursor**
  - **to change that, call: dragBoard.setDragView(Image, ...)**
  - **this is typically the source Node image or a snapshot of it**
  - **you may want to specify an offset to the drag view**

Drop Node

- **To allow a Node to be a Dropped On**
  - **you must register two handlers on it (as follows)**
  - **both receive a DragEvent**
- **OnDragOver handler *(may service several nodes)***
  - **you can determine the particular target being dragged over (e.g., which cell in a game grid) from event.getTarget()**
  - **determine whether this target Node is available for a drop (e.g., a cell in a game may not be available if already played on)**
  - **if it is available, determine whether the DragBoard (obtained from the event) contains an acceptable data type**
  - ***the preceding is important because the data source might be from outside the application***
  - **if both are OK, call event.acceptTransferModes(), passing the mode that the Node will accept (COPY or MOVE), which makes the next handler available on that node ...**
  - **failure to call that provides visible feedback to the user in the form of a universal NO sign (cannot drop here)**

Drop Node

- **OnDragDropped handler**
  - get the DragBoard from the event
  - it should already be known to have acceptable data from the DragOver handler (or we wouldn't receive this event)
  - so grab the data and do something appropriate, which would typically change the state of the drop Node
  - when done, call: event.setDropCompleted(true)