Australian National University

Integrated Development Environments (IDE)

Eric McCreath

Overview

In this lecture we will focus on the use of eclipse. In particular we consider:

- approaches for navigating your source code,
- the debugger,
- javadoc,
- source code formatting and organizing imports,
- using hints (break your code so eclipse can fix it!)
- using refactoring (renaming, extracting methods/variables, creating getters and setters, constructors, creating named constants) and
- what to do when things go astray.

Disclaimer

Note, these slides aim to provide an overview of aspects of ecplise that are worth exploring, rather than a tutorial on how to use them. The ecplise "Help" menu provides a more complete explanation of different aspects of ecplise and how to get them working.

The IDE

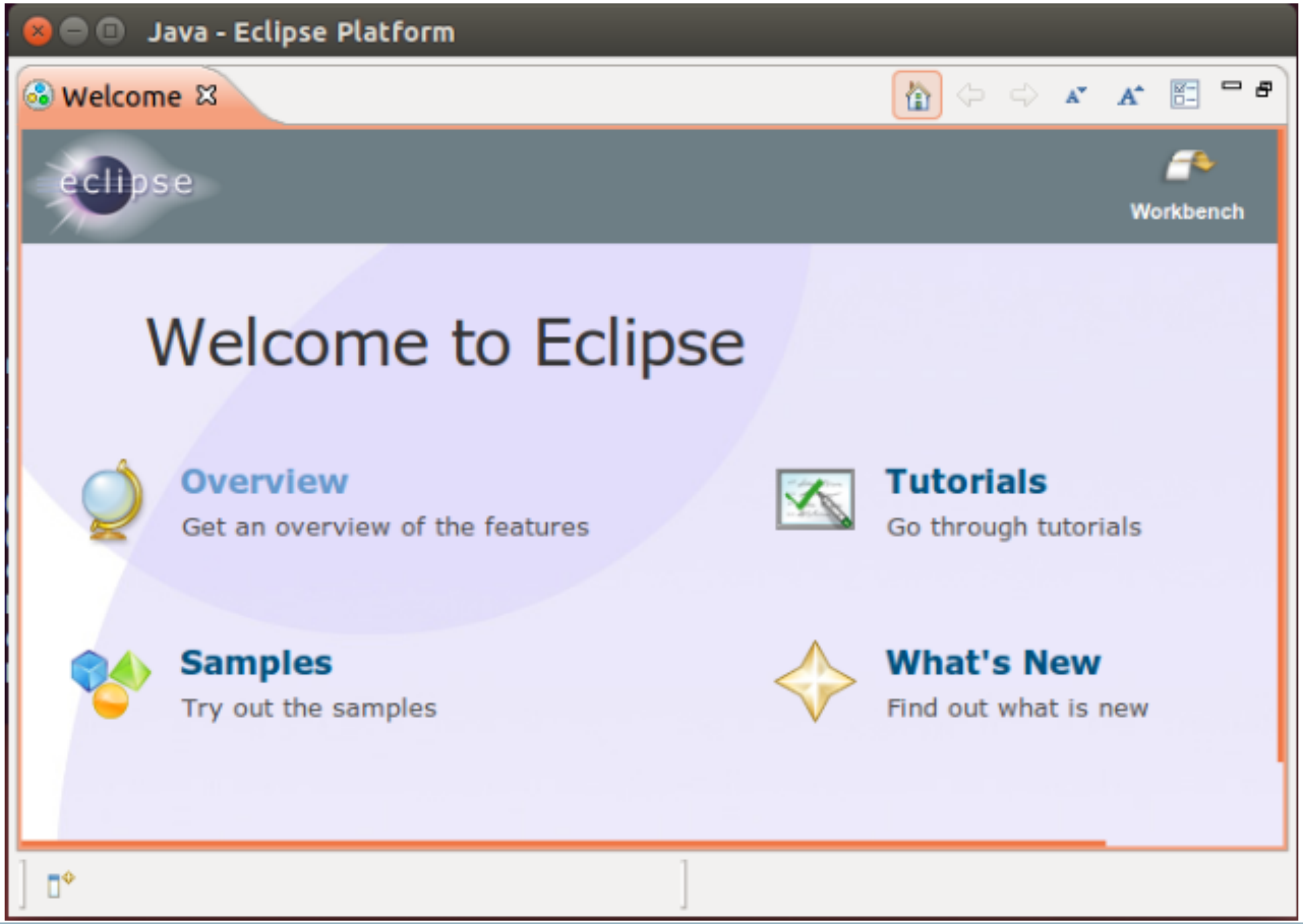
Integrated Development Environments (IDE) are applications that help you develop your code. Normally at a minimum they would help you: create, edit and manage your source, let you compile your code, direct you to the location of errors in your source, and let you run your code. These programs have grown in size and complexity over the years and will often aid you in all sorts of other ways.

There are many IDEs including: IntelliJ IDEA, NetBeans, Microsoft Visual Studio, EMACS, Qt creator, and Embarcadero Delphi.....

Eclipse is one such tool that can greatly improve your programming efficiency, however, eclipse will take some time to master.

The Basics of using eclipse

The first time you run eclipse you should see the "Welcome" Screen (if you don't see this you can use the "Help" menu to bring it up). This provides access to an overview and some tutorials to get you started. See below:



Australian National University

The Basics of using eclipse

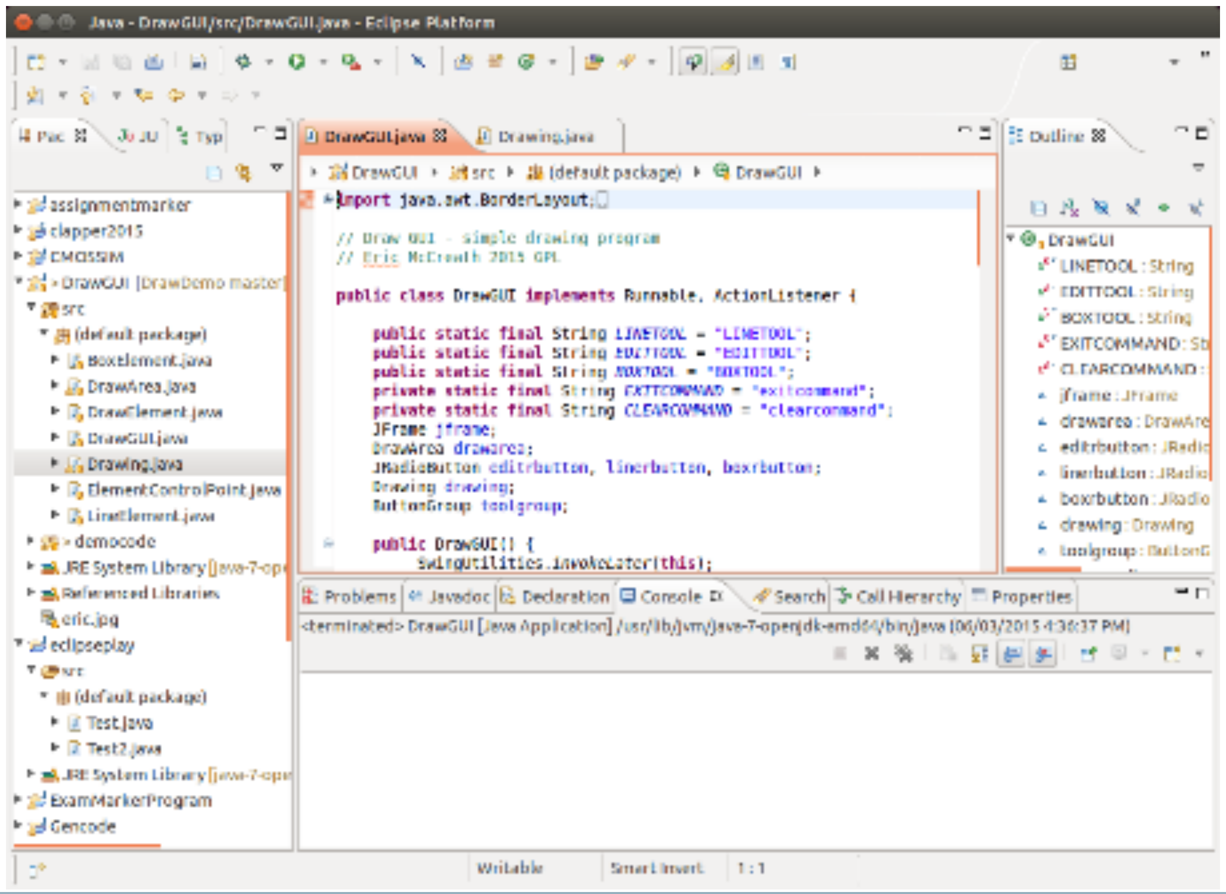
A few points to note:

- You work on a "project" which is stored in a particular directory.
- Start off with the basics of using the eclipse tool (creating and editing classes, fixing errors and running your code).
- Once you get familiar with the basics then explore some of the more advanced features (these can be great time savers).

Australian National University

Perspectives

When developing code you need to be able to view/explore/modify different aspects of the project. A "perspective" provides you with a particular layout of a collection of "Views". There are a number of standard perspectives such as "Java", "Java Browsing", "Debug", and "Git". You can change between perspectives. They can also be configured. The "Java" perspective is shown below:



```
import java.awt.BorderLayout;

// Draw GUI - simple drawing program
// Eric McCreath 2015 GPL

public class DrawGUI implements Runnable, ActionListener {

    public static final String LINETOOL = "LINETOOL";
    public static final String EDITTOOL = "EDITTOOL";
    public static final String BOXTOOL = "BOXTOOL";
    private static final String EXITCOMMAND = "exitcommand";
    private static final String CLEARCOMMAND = "clearcommand";
    JFrame jframe;
    DrawArea drawarea;
    JRadioButton editrbutton, linerbutton, boxrbutton;
    Drawing drawing;
    ButtonGroup toolgroup;

    public DrawGUI() {
        SwingUtilities.invokeLater(this);
```

```
<terminated> DrawGUI [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (06/03/2015 4:36:37 PM)
```

Navagating your source code

The "Package Explorer" provides a simple way of opening source code for editing.   Once open you can use the tabs to move between different source files.

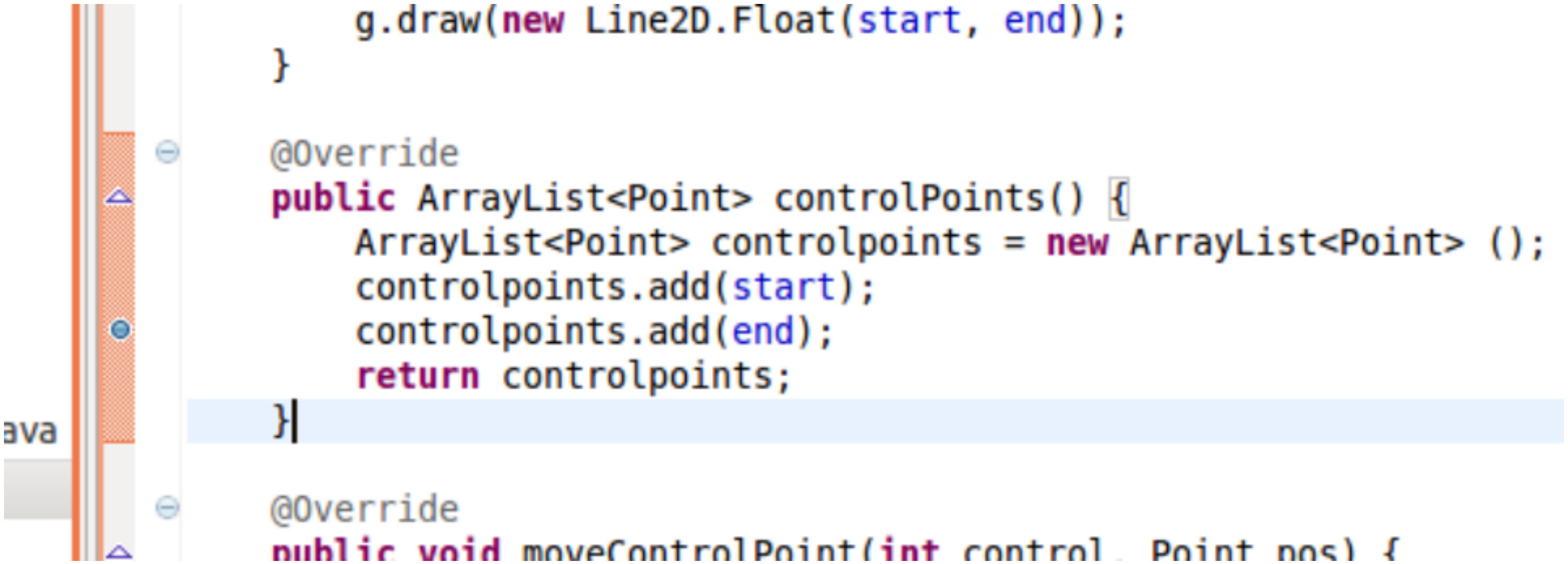
If you are editing code that uses another class that you wish to open up then a quick way of tracking the other class down is using right-click "OpenDeclaration" or F3.

If you wish to find where a class or method is used in your code then you can use right-click "OpenCallHierarchy".

The Search menu provides further options for searching through your code base.

Using the Debugger

You can add "breakpoints" into your code by left-clicking on the left side of text edit view. This creates a blue dot. If you run your code in the debugger then it will stop at the breakpoint.

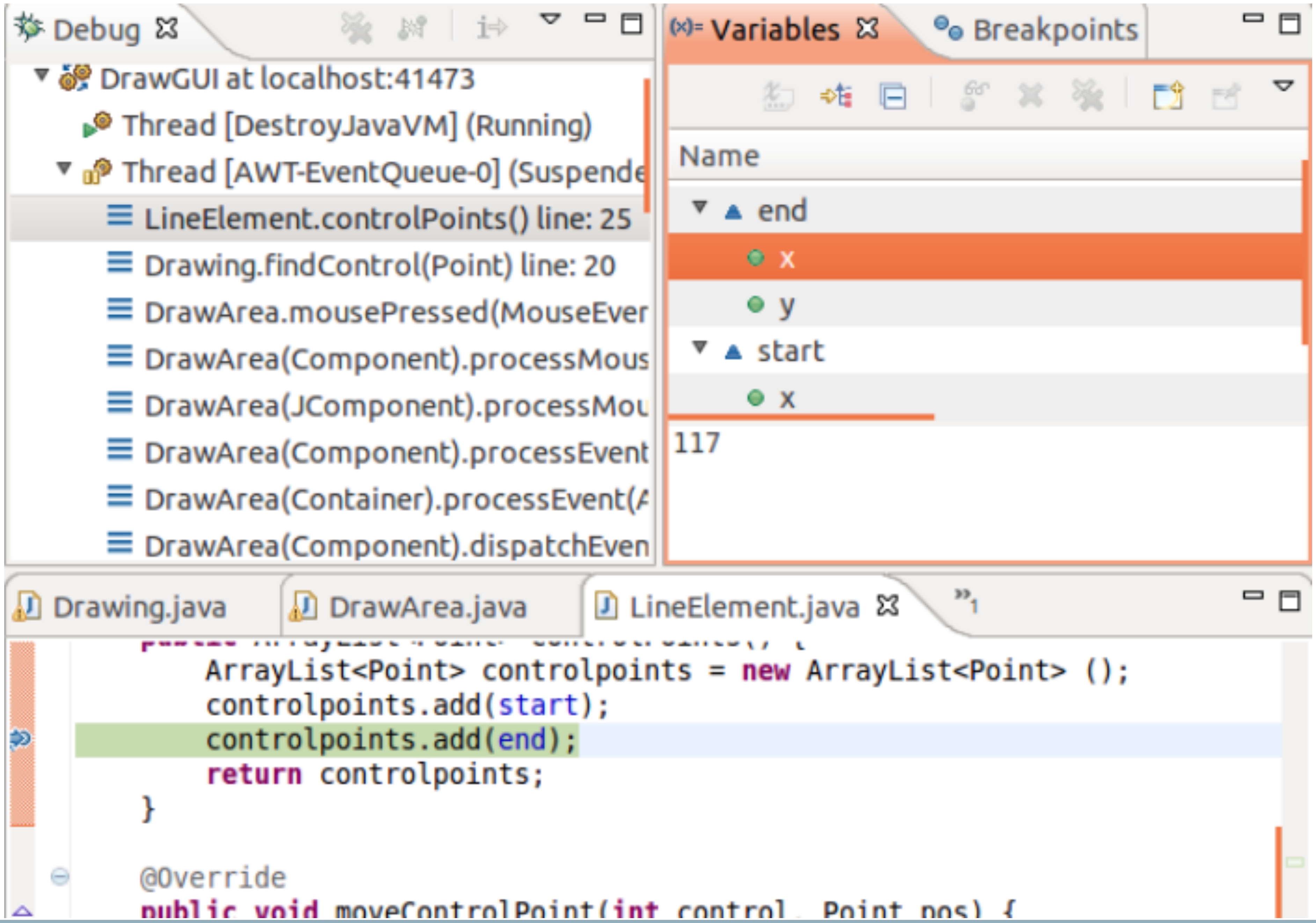


```
        g.draw(new Line2D.Float(start, end));
    }

    @Override
    public ArrayList<Point> controlPoints() {
        ArrayList<Point> controlpoints = new ArrayList<Point> ();
        controlpoints.add(start);
        controlpoints.add(end);
        return controlpoints;
    }

    @Override
    public void moveControlPoint(int control, Point pos) {
```

Australian National University

Using the Debugging

Once in the debugger you can step through your code and inspect the values of different variables. This can be a good way of tracking down problems.



```
DrawGUI at localhost:41473
    Thread [DestroyJavaVM] (Running)
    Thread [AWT-EventQueue-0] (Suspende
        LineElement.controlPoints() line: 25
        Drawing.findControl(Point) line: 20
        DrawArea.mousePressed(MouseEver
        DrawArea(Component).processMous
        DrawArea(JComponent).processMou
        DrawArea(Component).processEvent
        DrawArea(Container).processEvent(A
        DrawArea(Component).dispatchEven
```

```
end
    x
    y
start
    x
117
```

```
        ArrayList<Point> controlpoints = new ArrayList<Point> ();
        controlpoints.add(start);
        controlpoints.add(end);
        return controlpoints;
    }

    @Override
    public void moveControlPoint(int control, Point pos) {
```

Javadoc

Javadoc is a tool for generating documentation for a program from the source code of that program. The documentation generated is based on both the code and the comments present in the code. The comments sit within a / .. / *comment (note the extra).*

So a comment for a Class may be:

```
/**
 * LineElement - holds a simple line
 * @author Eric McCreath
 */
```

eclipse will help in terms of formatting these comments and listing standard descriptive tags.

Source Code Formatting

Nicely formatted code is easier to read. Code can be quickly formatted in eclipse using either the source->format menu item or ctrl-shift-f key combination.

You can configure the format used. However, the default configuration is generally fine.

This can be a great time saver.

Organizing Imports

Using ctrl-shift-o key combination or source->Organize Imports will remove import directives that are no longer needed and also add in ones that are required for your class.

Once again a great time saver.

hints

Errors in your code are marked with a red dot having a white cross inside. If there is also a yellow light bulb then eclipse has some ways of correcting the problem.   Left click on the yellow light bulb and they will be listed. If one item listed appropriately fixes the problem then select it.

Often adding a feature to your code requires a number of separate changes to the source. By carefully ordering these changes you can often get eclipse hints to do some of the work.

Australian National University

Refactoring Code

Renaming a class will generally involve changing your code at a number of places. Not only do you need to change the class definition and file name, but also where it is used. In a similar way often many changes are required when renaming a method.

Eclipse can do this work for you. Right-click on the element you wish to rename, select refactor->rename and let eclipse do the rest!

It is good programming practice to use named constants. A simple way of doing this in eclipse is to just use the constant directly in your code and then to use Refactor->ExtractConstant to give it a name. This can also extract other constants that have the same value and give them the same name.

Extracting variables and methods

Sometimes you have an expression that you wish to reference via a variable. In such cases, highlight the expression and then right-click Refactor->Extract Local Variable. This will create a local variable, assign the expression to it and use the variable in the place of the original expression.

In a similar way you can extract a method from a sequence of statements.

Getters, Setters, and Constructors

Many programmers will control access to fields by using getter and setter methods. Given the standard nature of this code eclipse will generate it for you.   (Source->Generate Getters and Setters)

Also eclipse can add constructor methods that initialize fields of the class. (Source->Generate Constructor using Fields)

Getters, Setters, and Constructors

The constructor, getters and setters shown below were added to the class with 5 mouse clicks. This savied typing approximately 280 characters.

```
public class Person {
    String name;
    int age;
    public Person(String name, int age) {
        super();
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

Tips when eclipse is being difficult

- Sometimes eclipse will not compile or run your code as expected. A few things to note:
  - this often relates to the setup of Project->Properties->BuildPath, check you have the correct JRE and all the required libraries.
  - sometimes eclipse will have old error information and it will say there is an error in some part of the code which is okay. Modify (just add a space), save and recompile the code and the error generally disappears.
  - attempt to isolate, understand, and replicate problems.

Australian National University

Example Exam Questions

- What are the advantages of using refactoring in eclipse in order to rename a method compared to that of using a simple text editor?
- Create a new project in eclipse and configure the build path so it includes the json-simple-jar library.
- Suppose the directory ~/myprogram contains the source code of a program you are developing. Set up an eclipse project that uses this directory and enables you to use eclipse to work on the project.

Australian National University

Android

Components II

Gaurav Mitra

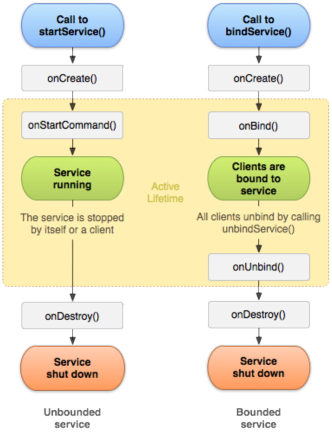
Services[1]

- A *daemon* that can perform long-running operations in the background
- Does not provide a UI
- If *started* by an app component using startService() it keeps running in the background even if user switches to another app.
- It is up to a started service to perform its task and exit and may not return a result to the component that started it
- An app component can *bind* to a service using bindService and interact with it to perform IPC
- A bound service offers a *client-server* interface which can be used by components to issue requests and get results
- A bound service only runs as long as another app component is bound to it. Multiple binds may happen, but when there are no more bound components, the service is destroyed
- Services can also be programmed to be started (and run indefinitely) and also allow binding

[1] http://developer.android.com/guide/components/services.html.

Service Lifecycle[2]

- Create a subclass of the Service class or one of its subclasses
- Override important callback methods as appropriate
  - **onStartCommand()**: This is executed by the system once startService() is issued. If this is implemented, service starts and runs indefinitely. stopSelf() or stopService() must be called to end service
  - **onBind()**: Called when bindService() is issued
  - **onCreate()**: Called to perform one-time setup procedures. Not called if service already running
  - **onDestroy()**: System calls this method when service is destroyed. Used to perform clean up as it is the last call received by a service
- Like any other component, services must be declared in the manifest using the `<service>` tag



[2] http://developer.android.com/guide/components/services.html.

Persistent Data Storage[3]

- Data must persist across user sessions
- Several options provided by Android to save persistent app data
- Possible to have data completely private to the app or accessible to other apps and the user
- Five ways to store data:
  - *Shared Preferences*: Primitive data usually in *key-value* pairs
  - *Internal Storage*: Private data on device memory
  - *External Storage*: Public data on shared external storage
  - *SQLite Database*: Structured data in a private database
  - *Network Connection*: Store data on the web with your own network server
- Private data may also be shared with other apps using *Content Providers*

[3] http://developer.android.com/guide/topics/data/data-storage.html.

Australian National University

Shared Preferences[4]

- Save and retrieve persistent *key-value* pairs of primitive data types such as boolean, float, int, long, string
- Use SharedPreferences objects. To get one of these objects use one of these two methods
  - **getSharedPreferences()**: Use if multiple preferences files present and a filename needs to be specified
  - **getPreferences()**: Use if only one preference file in the Activity and no name needs to be specified
  - Both of these methods requires a *mode* parameter. There are three modes of saving data:
    - *MODE_PRIVATE*: Default mode 0. Created file only accessible to app that created it
    - *MODE_WORLD_READABLE*: Mode 1. Allows all other apps to have read access to created file. Serious security risk. Deprecated in API 17
    - *MODE_WORLD_WRITEABLE*: Mode 2. Allows all other apps to have write access to created file. Serious security risk. Deprecated in API 17

```
SharedPreferences settings = getSharedPreferences("PrefFile", Context.MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
editor.putString("appName", "LectureDemo");
editor.commit();
String appName = settings.getString("appName", "No Name Specified");
```

[4] http://developer.android.com/guide/topics/data/data-storage.html.

Internal Storage[5]

- Files can be directly saved on the device's internal storage which is always available
- Files saved in internal storage are private by default and cannot be accessed by other apps or users
- No extra permissions required to access internal storage
- If app is uninstalled, these files are removed
- Files are created, read, written and deleted using the `File` APIs

```
File file = new File(context.getFilesDir(), filename);
file.delete();
```

- A temporary/cached file can also be created using getCacheDir() instead of getFilesDir()
- A *File* object is suited to reading or writing large amounts of data in start-to-finish order without skipping bytes e.g. text, image files
- Files are saved in encoded bytes. Default encoding is Unicode

[5] http://developer.android.com/guide/topics/data/data-storage.html, http://developer.android.com/training/basics/data-storage/files.html.

Australian National University

Internal Storage[6]

```
/* The following code saves a file called 'myfile' with text 'Hello World'*/
String filename = "myfile";
String string = "Hello world!";
FileOutputStream outputStream; /* Writes to a file in app's internal directory */

try {
  outputStream = openFileOutput(filename, MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}

/* The following code reads the contents of the file named 'myfile'*/
File file = new File(getFilesDir(), filename);
BufferedReader input = new BufferedReader(new InputStreamReader(new FileInputStream(file)
     ));
String line;
StringBuffer buffer = new StringBuffer();
while ((line = input.readLine()) != null) {
    buffer.append(line);
}
/* Print file content to logcat */
Log.d(TAG, buffer.toString());
```

[6] http://developer.android.com/guide/topics/data/data-storage.html, http://developer.android.com/training/basics/data-storage/files.html.

Australian National University

External Storage[7]

- Usually denotes removable storage present on a device such as a micro SD card or other external storage
- Some devices even divide permanent (non-removable) storage space into internal and external partitions
- May not always be available as user may unmount it
- Public access to files stored here
- Use this storage to possibly share files with other apps and users
- Must ask for permission from user to access this space. Add this line to the manifest

```
<uses-permission android:name=‘‘android.permission.WRITE_EXTERNAL_STORAGE’’ />
```

- Must check whether it is available and readable before accessing

[7] http://developer.android.com/guide/topics/data/data-storage.html, http://developer.android.com/training/basics/data-storage/files.html.

Australian National University

External Storage[8]

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

[8] http://developer.android.com/training/basics/data-storage/files.html.

External Storage[9]

```
/* Save public files to external storage which are freely available to other apps
   and are not deleted when your app is uninstalled
*/
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
/* Save private files to external storage which are also available to other apps and
   the user but are not meaningful outside your app; These files are deleted when
   your app is uninstalled
*/
public File getAlbumStorageDir(Context context, String albumName) {
    // Get the directory for the app's private pictures directory.
    File file = new File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

[9] http://developer.android.com/training/basics/data-storage/files.html.

Australian National University

SQL Databases[10]

- Full support for *SQLite* databases
- Any database created will be accessible to any class in the app but not outside the app
- Create a subclass of SQLiteOpenHelper and override onCreate() to execute SQL commands
- Use SQLiteDatabase query() method to execute a query which will return a Cursor object pointing to all rows resulting from the query

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

[10] http://developer.android.com/guide/topics/data/data-storage.html, http://developer.android.com/training/basics/data-storage/databases.html.

Content Providers[11]

- Manage access to structured sets or repositories of data
- Encapsulate data and provide secure access to it
- Standard interface that connects data in one process with code running in another process
- DO NOT need to create providers if there is no intention of sharing data with other apps
- DO need providers in case you want to copy or paste data or files from your app to other apps
- Android includes providers for audio, video, images, contacts etc

[11] http://developer.android.com/guide/topics/providers/content-providers.html, http://developer.android.com/guide/topics/providers/content-provider-basics.html.

Australian National University

Android

User Interface

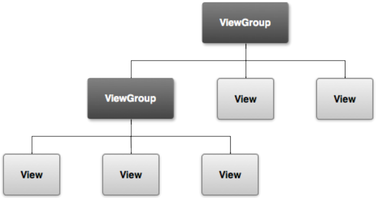
Gaurav Mitra

Australian National University

User Interface[1]

Everything that a user sees and interacts with is classified under UI

- All elements built using *View* and *ViewGroup* objects
  - `View`: An object that draws something on the screen
  - `ViewGroup`: An invisible container that holds other view objects to define a particular layout
- View objects can be instantiated in code or defined in XML layout files
- XML tag names correspond with Java class name e.g.`<TextView>` corresponds with the TextView widget



[1] http://developer.android.com/guide/topics/ui/overview.html.

Layouts[2]

- A visual structure for a UI with two methods for declaration
- *In XML layout files*
  - Well defined XML vocabulary corresponding to Widgets and Layouts
  - Enables better separation of the presentation of the app from code that controls behaviour
  - Allows quick reconfiguration of UI elements without changing control code
  - Easier to visualize structure of UI
  - Possibly easier to debug
  - Can create separate XML layouts for different screen orientations, sizes etc. These layouts could be switched around in a dynamic fashion based on screen conditions
- *Instantiated in Java code at runtime*:
  - Create View and ViewGroup objects
  - Change their properties programmatically.
  - Difficult to separate presentation from control code

[2] http://developer.android.com/guide/topics/ui/declaring-layout.html.

Australian National University

XML Layouts[3]

- Files must be placed in res/layout/*filename*.xml
- Filename used as resource ID i.e. R.layout.filename
- Each layout file must have exactly one root element which is either a View or ViewGroup object

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
...
```

- UI elements have many *attributes*. Some attributes are common to all View objects
  - *ID*: A unique string identifier specified in XML but is referenced as an integer after compilation. XML format for specifying an ID is

```
android:id="@+id/my_widget"
android:id="@android:id/empty"
```

    - @ - XML parser must parse and expand the rest of the ID string and identify it as an ID resource
    - + - It is a new resource name that must be created/compiled and added to R.java
  - *Layout Width*, *Layout Height*
- Declare widgets: Define in layout file with unique ID, then create instance of View object and capture it from layout in Java code e.g. findViewById(R.id.my_widget)

[3] http://developer.android.com/guide/topics/ui/declaring-layout.html.

Australian National University

XML Layout Syntax[4]

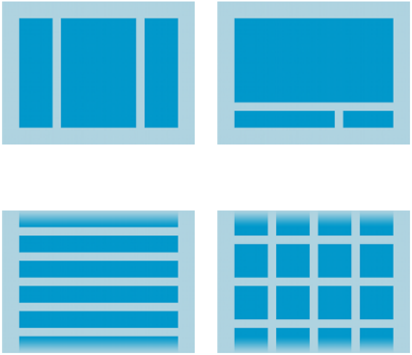
```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
    android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
        android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

- fill_parent: Match dimension to parent
- wrap_content: Set dimension to only size required to fit content
- <include>: Includes another layout file within this file

[4] http://developer.android.com/guide/topics/resources/layout-resource.html.

Layout Types[5]

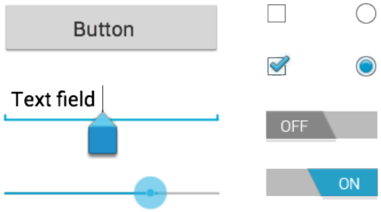
- **Static layouts**: the widgets are usually pre-defined
    - *Linear*: Single horizontal or vertical row. Automatic scrollbar if length of window exceeds length of screen
    - *Relative*: Specify child object locations relative to each other
- **Dynamic layouts**: Widgets pulled from data source such as an array or db query, using an `Adapter` which converts data items to `Views` that can be added to an `AdapterView` layout
    - *List View*: Scrolling single column list
    - *Grid View*: Scrolling grid of columns and rows
- Possible to nest layouts, but should ideally be avoided because of performance issues with nested layouts. Flat hierarchy is better



[5] http://developer.android.com/guide/topics/ui/declaring-layout.html.

Input Controls[6]

- **Button**: Push-button that can be pressed
- **Text Field**: Editable text field. `AutoCompleteTextView` available
- **Checkbox**: An on/off switch that can be toggled. Used to provide options
- **Radio Button**: Like checkbox, but only one option selectable
- **Toggle Button**: On/off button with indicator
- **Spinner**: Drop-down list from which one item can be selected
- **Picker**: Dialog to scroll a set of values and pick one. Used for time, date etc



[6] http://developer.android.com/guide/topics/ui/controls.html.

Input Events[7]

- Capture events from users interaction with individual widgets or View objects
- Multiple *event listeners* (interface in View class which has single *callback* method) provided. Listeners must be registered for individual View objects
  - on**Click()**: User touches item or presses *enter* key when applicable
  - on**LongClick()**: Similar to onClick, but user also holds touch
  - on**FocusChange()**: User navigates onto or away from item
  - on**Key()**: When focused, a hardware key is pressed or released
  - on**Touch()**: Any touch event i.e. press, release, movement gesture etc
  - on**CreateContextMenu()**: When context menu is created on long click

```
Button myButton = (Button) findViewById(R.id.my_button);
myButton.setOnClickListener(
    new View.OnClickListener(){
        public void onClick(View v){
            Log.v(TAG, "Button Pressed");
        }
    }
);
```

[7] http://developer.android.com/guide/topics/ui/ui-events.html.

Menus[8]

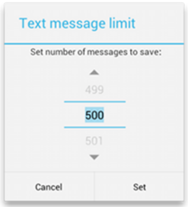
- Standardized menus available from the `Menu` class. Three types of menus:
  - *Options menu and app bar*: Actions relevant to current activity such as *Search*, *Settings* etc. For API 11 and higher, this is in the app bar at the top of each app
  - *Contextual menu*: Offers actions that affect a specific item or context frame in the UI. Most often used for items in `TextView`, `TextView` layouts
    - *Floating Context Menu*: Menu appears as a floating list or dialog
    - *Contextual Action Mode*: Menu appears as an action bar at the top
  - *Popup menu*: Anchored to a `View` in an overflow-style for actions that relate to specific content
- Can be defined in XML or Java code
- Menu click events handled using listeners and callback methods such as onOptionsItemSelected()

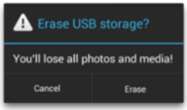
[8] http://developer.android.com/guide/topics/ui/menus.html.

Australian National University

Dialogs[9]

- Small window that prompts user to make a decision or enter additional info
- Should derive from the Dialog base class without instantiating it directly
- Use one of the subclasses: AlertDialog, DatePickerDialog, TimePickerDialog

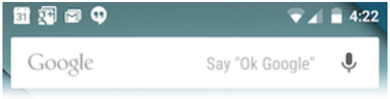




9 http://developer.android.com/guide/topics/ui/dialogs.html.

Notifications[10]

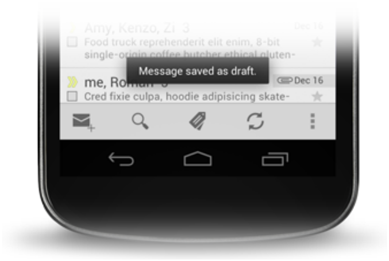
- A message you can display to the user *outside* of apps normal UI
- First appears as an icon in *notification area*. User must open up *notification drawer* to view it
- A `Notification` object must contain
  - *setSmallIcon()*: A small icon
  - *setContentTitle()*: A title
  - *setContentText()*: Detail text
- Notification actions defined by using Intent objects that start an Activity



[10] http://developer.android.com/guide/topics/ui/notifiers/notifications.html.

Toasts[11]

- Provides simple feedback about an operation in a small popup
- Only fills amount of space required for the message
- Current activity remains visible and interactive during Toast display
- Instantiate a Toast object


Amy, Kenzo, Zi 3
Food truck reprehenderit elit enim, 8-bit
single-origin coffee butcher ethical gluten-
Message saved as draft.
me, Roman 3
Dec 16
Cred fixie culpa, hoodie adipisicing skate-

11 http://developer.android.com/guide/topics/ui/notifiers/toasts.html.

Android

Components I

Gaurav Mitra

Australian National University

Android Studio[1]

- The official IDE for Android app development
- Based off the IntelliJ IDEA
- Custom build system for apps called *Gradle*
- Rich layout editor with drag-and-drop theme editing
- Deep integration with version control - git

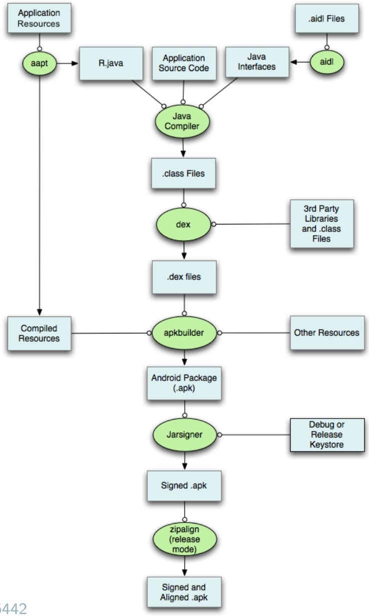
[1] http://developer.android.com/tools/studio/index.html.

Build System

- Build process[a] to produce an *.apk* file
- Complete process is repeated every time a Gradle build task is run for a project
- Flexible and configurable process
- *aapt*: Android Asset Packaging Tool - Compiles app XML resources to produce R.java which allows resources to be referenced from within Java code
- *aidl*: Android Interface Definition Language - Allows definition of programming interfaces that both clients and services can agree upon for IPC[b]
- *dex* tool converts .class files to Dalvik bytecode .dex. All 3rd party libs and .class files also converted to .dex for inclusion in .apk
- Non-compiled resources such as images, compiled resources and dex files are packaged into .apk using the *apkbuilder*
- *Jarsigner* signs the .apk with a debug or release key
- *zipalign* reduces the memory footprint of the .apk
- app/build/outputs/apk/ folder contains output .apk files

[a] http://developer.android.com/sdk/installing/studio-build.htm

[b] http://developer.android.com/guide/components/aidl.html.



Gradle[2]

- A general purpose build automation tool like GNU make
- Supports a diverse range of languages including Java, C and C++
- Uses *plugins* to configure a project in some way typically by adding some pre-configured tasks
- Ships with a number of plugins such as the *Java plugin* which compiles java source code, runs unit tests and creates a .jar file
- Build configuration stored in *build.gradle*
- *Convention based*: Expects certain default values such as where files are located. The Java plugin expects
  - src/main/java: Production source code location
  - src/test/java: Test source code location
  - src/main/resources: Files placed here to be packaged in .jar
  - src/test/resources: Files placed here to be included in classpath used
  - build/libs: Final .jar file location

[2] http://gradle.org/.

Android Manifest File[3]

- Every app must have an app manifest contained in a file named `AndroidManifest.xml`
- The manifest presents essential information about an app to the Android system
- The Android system must have this information before it allows an app's code to be run
- Some features of the manifest include:
  - Specifies Java package name
  - Describes components of an app and names the classes that implement each of the components and publishes their capabilities
  - Declares permissions
  - Lists *Instrumentation* classes that provide profiling and runtime statistics for an app
  - Declares minimum API level
  - Lists the libraries that an app must be linked against

[3] http://developer.android.com/guide/topics/manifest/manifest-intro.html.

App Components

- Set of reusable components that allow creation of app features
- Essential building blocks of an app
- Each component is a different *entry point* to the app
- Not all components are actual entry points for the user i.e. system can communicate with apps using a component
- Some dependencies amongst components
- Each component has a distinct lifecycle that defines how it is created and destroyed
- Four types of app components include:
  - *Activities*: Represents a single screen in an app
  - *Services*: Performs operations in the background without a UI
  - *Content Providers*: Manages a shared set of app data
  - *Broadcast Receivers*: Responds to system-wide broadcast announcements
- Intents and Intent Filters used for communication between components

Australian National University

Intents[4]

- An Intent is an asynchronous messaging object that can be used to request an action from another app component
- Three fundamental use-cases:
  - *Starting an Activity*: Passing an Intent that describes the Activity to start and associated data to startActivity() or startActivityForResult(). Results received are also using Intent objects
  - *Starting a Service*: To perform a one-time operation. Procedure similar to Activity. If Service has client-server interface then Intent can be passed to bindService()
  - *Delivering a broadcast*: One way is to pass an Intent to sendBroadcast()
- Two types of Intents:
  - *Explicit*: Specify the component to start by its fully-qualified class name
  - *Implicit*: No specific component named. A general action to perform is specified. This allows a component from another app to handle it

[4] `http://developer.android.com/guide/components/intents-filters.html`.

Intent Filters[5]

- Used to specify which Implicit Intents an app can receive
- One or more Intent Filters for each app component can be declared using the `<intent-filter>` element in the manifest file
- The types of Intents is based on an Intent's `<action>`, `<data>` and `<category>`
- System delivers an Implicit Intent to an app component only if the intent can pass through one of the Intent Filters specified

[5] http://developer.android.com/guide/components/intents-filters.html.

Australian National University

Activities[6]

- An Activity represents a single screen with a UI
- For example, a note taking app might have an activity to view all notes, another activity to create/compose a note with keyboard input and yet another activity to delete a note
- Although all Activities must work together to form a wholesome app experience, each activity is independent of another
- It is possible for an activity to be triggered by any other app - Multiple entry points
- For example, a browser app may have a feature to save a section of text on a webpage using a notes app. It could then trigger the note creating activity directly to enter the notes app
- An Activity must be declared in the manifest file using an `<activity>` element as a child of the `<application>` element in order for it to be accessible to the system

[6] http://developer.android.com/guide/components/activities.html.

Activities[7]

- Each activity is typically given a full-screen window to draw its UI elements
- The *main* activity is presented to the user when an app is launched for the first time
- Each activity can start other Activities to perform different actions
- Each time a new activity starts, the previous one is stopped. However, it is preserved in the *back stack* which is a LIFO structure
- When a new activity starts, it is pushed to the back stack and takes user focus
- Pressing the back button, pops an activity from the back stack, destroys it, and resumes the previous activity
- When an activity is stopped, resumed or destroyed, it is notified of this change in state through a *callback* method
- The callback provides an opportunity to perform tasks that are appropriate for the change
- To create an Activity:
  - Create a subclass of the Activity class
  - Implement the onCreate() interface and initialize the essential parts of the activity
  - Call setContentView() to define the layout for the activity's UI

[7] http://developer.android.com/guide/components/activities.html.

Views[8]

- An activity's UI is a hierarchy of *Views* i.e. objects derived from the `View` class
- Each View controls a particular rectangular space within an activity window and can respond to user interaction. For e.g. a button
- A number of pre-defined views are provided
- *Widgets* are views that provide visual (and interactive) elements for the screen. For e.g. buttons, text fields, checkboxes, images etc
- *Layouts* are views derived from `ViewGroup` class which provides a unique layout model for its child views such as a linear, grid or relative layout
- Subclasses can be created from the `View` and `ViewGroup` classes to create custom widgets and layouts
- An XML layout file can be used to define a layout
- XML layouts allow maintenance of UI design to be completely independent of Java code that defines activity behaviour
- Pass resource ID of layout to `setContentView()`

[8] http://developer.android.com/guide/components/activities.html.

Fragments[9]

- A Fragment represents a portion of UI in an Activity
- Multiple Fragments can be combined to form a single Activity in the form of a multi-page UI
- Used to create more dynamic and flexible UI designs to support a variety of screen sizes and orientations
- Fragments can be re-used in multiple Activities
- Fragments have their own lifecycle, receive their own input events, can be added or removed while an Activity is running
- Fragments must be embedded in Activities i.e. when Activities have changes in state, fragments have the same changes
- Fragments need not be parts of an Activity i.e. can be an invisible worker
- Creation process similar to that of an Activity

[9] http://developer.android.com/guide/components/fragments.html.

Australian National University

Logging and Debug

- *Android Debug Bridge (adb)*[10]: Tool for communication between dev machines and Android devices
  - *Client*: Runs on dev machine. Can be invoked from cmdline shell using adb command
  - *Server*: Runs on dev machine. Background process which manages communications between client and adb daemon running on emulator or device
  - *Daemon*: Runs on android emulator or device as a background process
- *Logcat*[11]: Collective view of debug, error and warning messages from a particular emulator or device
  - Control verbosity of debug output using multiple log levels
  - Use the *Log*[12] class to print debug messages from an app to logcat

[10] http://developer.android.com/tools/help/adb.html.

[11] http://developer.android.com/tools/help/logcat.html.

[12] http://developer.android.com/reference/android/util/Log.html.

Australian National University

Android

Fundamentals

Gaurav Mitra

Scope

Topics to be covered, and assessed:

- Android Fundamentals
- Android Studio Overview
- App Components (Activity, Service, Content Provider, Broadcast Receiver, Intent etc)
- User Interface or UI (View, Layouts, Input Controls, Input Events, Event Handlers, Menus, Notifications etc)
- Persistent Data Storage

Scope

Topics beyond our scope (NOT to be covered):

- UI (Accessibility, Custom Components, Styles & Themes)
- Animation & Graphics
- Computation with Renderscript
- Media & Camera
- Connectivity (Bluetooth, NFC, Wifi P2P, USB, SIP)
- Synchronization via Cloud resources
- Web Apps

Why Android?

- Smartphones are ubiquitous
- Four major smartphone operating systems - Android, iOS, Windows Phone, Blackberry OS
- Android dominates the smartphone OS market share[1]
  - Q2, 2015: Android (**82.8%**); iOS (13.9%); Windows Phone (2.6%); Blackberry OS (0.3%); Other (0.4%)
  - Q2, 2014: Android (**84.8%**); iOS (11.6%); Windows Phone (2.5%); Blackberry OS (0.5%); Other (0.7%)
- Free and Open source[2]
- Supports multiple instruction-set architectures (ARM, x86 etc) and a variety of hardware platforms ranging from smartphones, tablets, laptops, set-top-boxes to watches, and soon.. cars!
- Built to run on top of the Linux kernel
- Focus on *touch-based* input i.e. UI is a major design priority

[1] http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[2] https://source.android.com/.

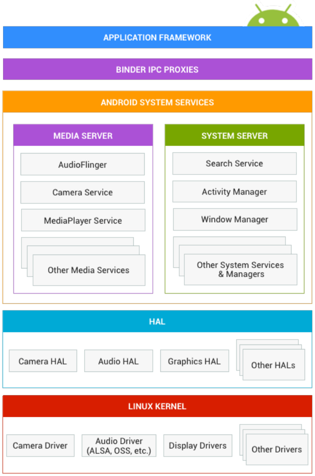
Software Construction Perspective

- Android incorporates the use of multiple software design patterns
- Creating an android app requires knowledge of
  - Java programming
  - Adaptable software design - A single app could target many different devices
  - Responsive UI design
  - Security and permissions
  - Mark-up languages such as XML
  - Persistent storage solutions using databases
  - Inter-process communication - Often between apps
  - Network communication
- Modularity is key in Android's *layered* architecture

Layered System Architecture

Loosely coupled layers built on top of each other[3]

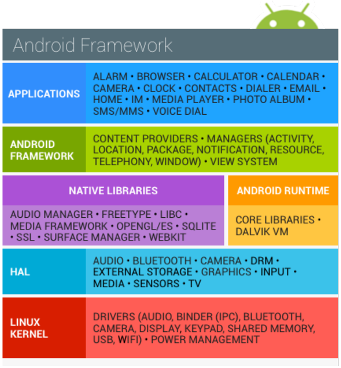
- *Application Framework*: For app developers. Many developer APIs map directly to the underlying HAL interfaces
- *Binder IPC Proxies*: Allows apps to make system calls to the System Server. This communication is hidden from the app developer
- *System Services*: Modular system and media services
- *Hardware Abstraction Layer (HAL)*: Defines standard interface for the hardware vendors to implement. Allows upper android layers to be device agnostic
- *Linux Kernel*: Special build using additions such as *wake locks*



[3] https://source.android.com/devices/.

Layered Software Stack

- *Loosely coupled* layers built on top of each other[a]
- *Android Framework*: For app developers. Our scope is limited to this layer only.
- *Android Runtime*: Dalvik Java virtual machine (VM). Most apps run on this VM
- *Native Libraries*: Some apps are run natively such as core services

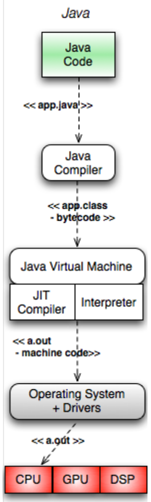

Android Framework
APPLICATIONS
ALARM • BROWSER • CALCULATOR • CALENDAR • CAMERA • CLOCK • CONTACTS • DIALER • EMAIL • HOME • IM • MEDIA PLAYER • PHOTO ALBUM • SMS/MMS • VOICE DIAL
ANDROID FRAMEWORK
CONTENT PROVIDERS • MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW) • VIEW SYSTEM
NATIVE LIBRARIES
AUDIO MANAGER • FREETYPE • LIBC • MEDIA FRAMEWORK • OPENGL/ES • SQLITE • SSL • SURFACE MANAGER • WEBKIT
ANDROID RUNTIME
CORE LIBRARIES • DALVIK VM
HAL
AUDIO • BLUETOOTH • CAMERA • DRM • EXTERNAL STORAGE • GRAPHICS • INPUT • MEDIA • SENSORS • TV
LINUX KERNEL
DRIVERS (AUDIO, BINDER (IPC), BLUETOOTH, CAMERA, DISPLAY, KEYPAD, SHARED MEMORY, USB, WIFI) • POWER MANAGEMENT

[a] https://source.android.com/security/index.html.

Australian National University

Java Virtual Machine

- Java programs are device/processor agnostic
- Source compiled to Java *bytecode* intermediate representation (IR)
- Bytecode is executed by a JVM
- JVM compiles bytecode to machine code *Just-in-time* (JIT) i.e. during the execution of the program
- JIT compilation is a combination of *Ahead-of-time* (AOT) compilation and *interpretation*
- Machine code executed by OS on CPU



Android's Dalvik JVM

Design Requirements:

- Must work on resource constrained hardware - smartphones a few years back had limited amounts of RAM and processing power
- Application Sandbox - to ensure security, performance and reliability
- Lack of swap space
- Battery powered

Android's Dalvik JVM

Dalvik Design Features[4]:

- Every Android app runs in its own process, with its own instance of Dalvik VM
- Multiple Dalvik VM's can run together efficiently
- Optimized intermediate representation: *Dalvik Executable* (.dex) - Minimal memory footprint
- *dex* tool compiles .class files to .dex files
- Since each app runs an instance of Dalvik, both creation and execution of new Dalvik instances must be efficient
- Dalvik relies on the Linux kernel to provide threading and low-level memory management

Dalvik has now been replaced with the improved ART (Android Runtime). However, ART remains compatible with Dalvik and runs Dex bytecode[5]

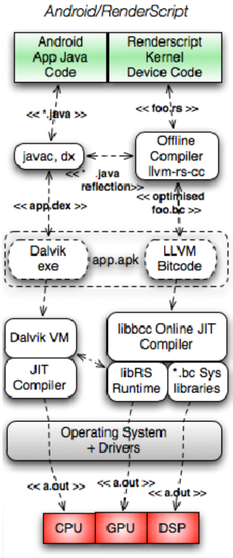
[4] http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf.

[5] https://source.android.com/devices/tech/dalvik/.

Australian National University

Android Code Path

- A typical smartphone has multiple prcessing elements such as a CPU, GPU and DSP
- Java code runs in the Dalvik or ART JVM which runs on the CPU
- How are the GPU and DSP cores used? *OpenGL* and *Renderscript*
- Renderscript code is compiled to *LLVM IR bitcode*
- LLVM Bitcode is JIT compiled and executed on the GPU or DSP



Security Sandbox

No app, by default, has permission to perform any operations that would adversely impact other apps, the OS or the user[6]

- Android is a *privilege-separated* OS
- Each application runs in isolation, in a *process sandbox*, with its own system identity (Linux UID, GID)
- Additional security provided via *permissions*
- Permissions enforce restrictions on specific operations that particular processes can perform
- Apps prevented from reading/writing user's private data, other app's files, unauthorized network access, keeping device awake etc
- The Dalvik VM is not a hard security boundary though - apps may still execute native code using the *Android NDK*

[6] http://developer.android.com/guide/topics/security/permissions.html.

Android Learning Resources

- Google developer docs:
  `http://developer.android.com/guide/index.html`
- API reference:
  `http://developer.android.com/reference/packages.html`
- Video training:
  `http://developer.android.com/training/index.html`
- Multiple books available

Australian National University

Android Version History and API Levels[7]

- *Android Alpha*: First release - November 2007
- *Android 1.0*: First commercial release - API Level 1.0 - September 2008
- Release schedule stabilized to two major releases every year 2010
- . . .
- *Ice Cream Sandwich*: Version 4.0-4.0.4 - API level 15 - October 2011 - 97% of Android devices today have an API level equal or higher than ICS
- . . .
- *Lollipop*: Version 5.0-5.1.1 - API level 22 - November 2014
- *Marshmallow*: Most recent version 6.0-6.0.1 - API level 23 - October 2015

[7] https://en.wikipedia.org/wiki/Android_version_history.

Swing - Part 2

Eric McCreath

Overview

In this lecture we explore some more features of the swing architecture. Including:

- JMenu,
- JTextPane,
- JList,
- JScrollPane,
- JOptionPane,
- JDialog, and the
- JFileChooser

Menus

To set up menus for your application construct a "JMenuBar" which you can add "JMenu"s to. Each "JMenu" you add the "JMenuItem"s to. The "JMenuItem"s works in a similar way to the JButton, so you would normally attach an ActionListener to it and set an ActionCommand for it.

```
    JMenuBar bar =  new JMenuBar();

    JMenu menu = new JMenu("File");
    bar.add(menu);

    JMenuItem menuitem = new JMenuItem("Exit");
    menuitem.addActionListener(this);
    menuitem.setActionCommand(EXITCOMMAND);

    menu.add(menuitem);
```

Once we have created the menu bar we can attach it to the jframe with.

```
    jframe.setJMenuBar(bar);
```

JTextPane

JTextPane provides a way of displaying (and if you wish editing) a block of formatted text. If you only wish to use unformatted text then just use a JTextArea. Noting the way you deal with these two classes is very similar.

To construct a new one and set the font used:

```
textpane = new JTextPane();
textpane.setFont(new Font("Monospaced", Font.PLAIN, 16));
```

To get the text from the textpane:

```
String text = textpane.getText();
```

You can set the text using "setText" and also add a document listener to be notified of when changes happen. Also you could change the style used.

```
SimpleAttributeSet attributes = new SimpleAttributeSet();
attributes.addAttribute(StyleConstants.CharacterConstants.Italic, true);
textpane.setCharacterAttributes(attributes, true);
```

Australian National University

JList

If you have a list of items you wish to display and select from then you can use JList. You can just create a JList and provide it with an array of data to display. e.g.

```
    String data[] = new String[DATASIZE];
    for (int  i=0;i<DATASIZE;i++) data[i] = "Item: " + i;
    jlist = new JList(data);
```

To see the value selected you can:

```
    jlist.getSelectedValue()
```

You may also listen to changes in the selected value(s) by adding a ListSelectionListener.

If you have the data that makes up a list you wish displayed you can provide JList with an object of a class that implements a "ListModel". A simple way of doing this is extending the "AbstractListModel" class. Note also you can replace the standard cell render with your own one.

JScrollPane

Sometimes the JComponent will not fit within the available window space. e.g. you have a long JList or the JTextArea is too big for the space available. A simple way of solving this problem is to decorate the JComponent within a JScrollPane.

So to embed the JList from the pervious example into a scroll pane one would:

```
    JScrollPane scroll = new JScrollPane(jlist)
```

These can work either horizontally or vertically. By default the scroll bars will only appear as needed. However you can change to a policy such that a scroll bar will: appear as needed, never appear, or always appear. e.g. the below would set the horizontal bar never to appear and the vertical to always appear:

```
    scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

JOptionPane

Sometimes you may wish to display a simple message to the user or ask the user to answer a question. This can be easily done via the JOptionPane. The below will display a message to the user.

```
    JOptionPane.showMessageDialog(jframe, "Message");
```

You can also obtain a "Yes", "No", or "Cancel" response using:

```
    int res = JOptionPane.showConfirmDialog(jframe, "Is this message okay?");
    // res will hold the response value
```

These dialogs are know as modal as they block other window obtaining input until the user responds to the dialog.

JDialog

You can also create your own dialog by extending the JDialog class. This can be set up as modal (other input stops until the dialog is hiden) or modeless (other parts of the GUI continues). With the modeless case you could have multiple dialog, whereas, with modal there would normally only be one.

The JDialog is similar to a JFrame, in that you can add components to the content pane, set up menus, pack, set visible, etc.

One needs to be careful in terms of how data is stored.

File Chooser

If you wish to load or save to a file then a simple way of enabling the user to select the name and director of a file is to use a JFileChooser.

For many application one can construct a single JFileChooser which can be reused whenever the user wishes to choose a file. So in your GUI class you can add as a field:

```
final JFileChooser filechooser = new JFileChooser();
```

Then whenever you would like select a file (based on a button press say) you can call:

```
int res = filechooser.showOpenDialog(parent);
```

If the "res" is JFileChooser.APPROVE_OPTION then filechooser.getSelectedFile() will return the file that was selected. As with many of the other parts of the Swing architecture the JFileChooser can be configured in many ways.

Some Example Exam Questions

- Explain different ways in which you can associate a list of data items with a JList. How is the "Model", "View", and "Control" separated in the JList.
- State some options you have in configuring the JScrollPane. Why would you describe the JScrollPane as decorating another JComponent?
- Modify the HelloWorldGUI code (as stated in the previous swing lecture) so the GUI application has a menu that enables you to, using the JFileChooser, select a file in the current director and display its file size (using a JOptionPane showMessageDialog).

Australian National University

References

Oracle has some good tutorials on these parts of Swing. See:

http://docs.oracle.com/javase/tutorial/search.html

Also the JDK API documentation is a good source of info.

Java Swing - Part 1

Eric McCreath

Overview

In this lecture we will:

- overview the Swing framework for the development of Java GUI applications,
- see how JFrame can be used to make a simple "hello world" window,
- look at different widgets that can be used within Swing (include JButton, JLabel, JTextField),
- explore how JComponent can be extended to create your own widget, and
- consider how one can use simple layout managers.

Swing

- Swing is a commonly used framework for the development of Java GUI applications.
- Swing uses "lightweight components", in that components are implemented purely in Java and don't have native peers. Prior to Swing developers commonly used the Abstract Windowing Toolkit (AWT) in which components where "heavyweight" (the objects of components had native peers). Moving to "lightweight" components made application more platform independant and less bloated (in terms of memory use).
- JavaFX is a new framework which aims to replace Swing.

Swing - Threads and Events

- Swing uses an event driven programming paradigm.
- Swing is not "thread-safe", in that all your Swing method calls should be done within the "event dispatch thread". i.e Don't start other threads that call methods of the swing framework. Note this, is not a big problem as any code that you have attached to a Listener will be executed within the event dispatch thread.
- To keep the application responsive all the code that executes on the event dispatch thread should execute "quickly".
- If code is computationally intensive (or involves considerable latency due to the network or the file system) then the GUI becomes less responsive. You may use a "Worker" thread to do these computationally intensive tasks to keep the GUI responsive.

JFrame

- JFrame is the base class which holds together the entire GUI. The JFrame is like an empty window that you can: add components, add menus, and set preferences.
- There is a few different ways you can set up a JFrame. These include:
  - having your main GUI program extend the JFrame,
  - having the JFrame as a variable within the main method, or
  - having a class which embraces the state of your GUI and having a JFrame as a field within this class.
- Note, as Swing is not thread-safe, one should do all this construction and setting up of the JFrame on the event dispatch thread. This can be done via SwingUtilities invokeLater method.

Australian National University

JFrame

Below is the a simple HelloWorld GUI.

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class HelloGUI implements Runnable {
    JFrame jframe;

    public HelloGUI() {
         SwingUtilities.invokeLater(this);
    }

    public void run() {
        jframe = new JFrame("HelloGUI");
        jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        jframe.getContentPane().add(new JLabel("Hello World"));
        jframe.setVisible(true);
        jframe.pack();
    }

    public static void main(String[] args) {
        new HelloGUI();
    }
}
```

JComponent

- The JComponent is a key abstract class in Swing as it provides the base class for both the standard set of Swing components and new ones an application developer creates.
- The JComponent provides important functionality that other Swing components build on. Including:
  - painting,
  - layouts,
  - drag-n-drop,
  - tool-tips,
  - double-buffering, and
  - key bindings.

JLabel

- JLabel is one of the simplest JComponents.   It provides the ability to display text, an image, or both.

- To make a new JLabel with some text you can just go:

```
    new JLabel("Some Text")
```

Often you don't need a reference for the JLabel so you can add it directly to a JPanel.

- You can also use JLabel to include a simple image:

```
    JLabel jlabel = new JLabel();
    jlabel.setIcon(new ImageIcon("imagefile.jpg"));
```

- You can also use "html" in the text of JLabels (and buttons, menu items, ...). So the below would make the text underlined:

```
    new JLabel("<html><u>Some Text</u></html>")
```

Australian National University

JButton

The JButton is a lot like the JLabel, except the JButton can be pressed. Normally one would attach an ActionListener to a JButton to complete the action that is expected after pressing the button.

```
    JButton jbutton = new JButton("Hello");
    jbutton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            System.out.println("button pressed");
        }
    });
```

We can also direct the action event to a class that implements an ActionListener (has an actionPerformed method). This would normally be your main GUI class. This means you need to now determine which button was pressed (given you often have a number of possible action events). One advantage of this approach is that action events can come from a number of different sources (or even be programactically generated).

Australian National University

JButton

```
public class DrawGUI implements Runnable, ActionListener {
    private static final String HELLO_BUTTON = "HelloButton";
    JFrame jframe;
    public DrawGUI() {
         SwingUtilities.invokeLater(this);
    }
    public static void main(String[] args) {
        new DrawGUI();
    }
    public void run() {
         jframe = new JFrame("DrawGUI");
         jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

         JButton jbutton = new JButton("Hello");
         jbutton.setActionCommand(HELLO_BUTTON);
         jbutton.addActionListener(this);

         jframe.getContentPane().add(jbutton);
         jframe.setVisible(true);
         jframe.pack();
     }
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals(HELLO_BUTTON)) {
            System.out.println("hello button pressed");
        }
    }
}
```

JTextField

A JTextField can be used for text entry. The below code will make a JTextField that is sized for 20 characters (although more can be entered).

```
JTextField jtextfield; // in the fields of the GUI class
jtextfield = new JTextField(20); //  in initialization code
```

To obtain the current text in the text field you can use:

```
jtextfield.getText()
```

One can attach a document listener to the "document" of a text field and observe events when the text changes:

```
    jtextfield.getDocument().addDocumentListener(new DocumentListener() {
             public void changedUpdate(DocumentEvent de) {
                 System.out.println("changed update");
             }
             public void insertUpdate(DocumentEvent de) {
                 System.out.println("insert update");
             }
             public void removeUpdate(DocumentEvent de) {
                 System.out.println("remove update");
             }
         });
```

LayoutManager !@#!

Often one needs to organize a number of components within the window. A layout manager will do this for you.   A LayoutManager will be associated with the main content pane (by default it will use a BoarderLayout).

Also you can use a JPanel which is an empy JComponent that you can add other components to (by default it will use the FlowLayout).

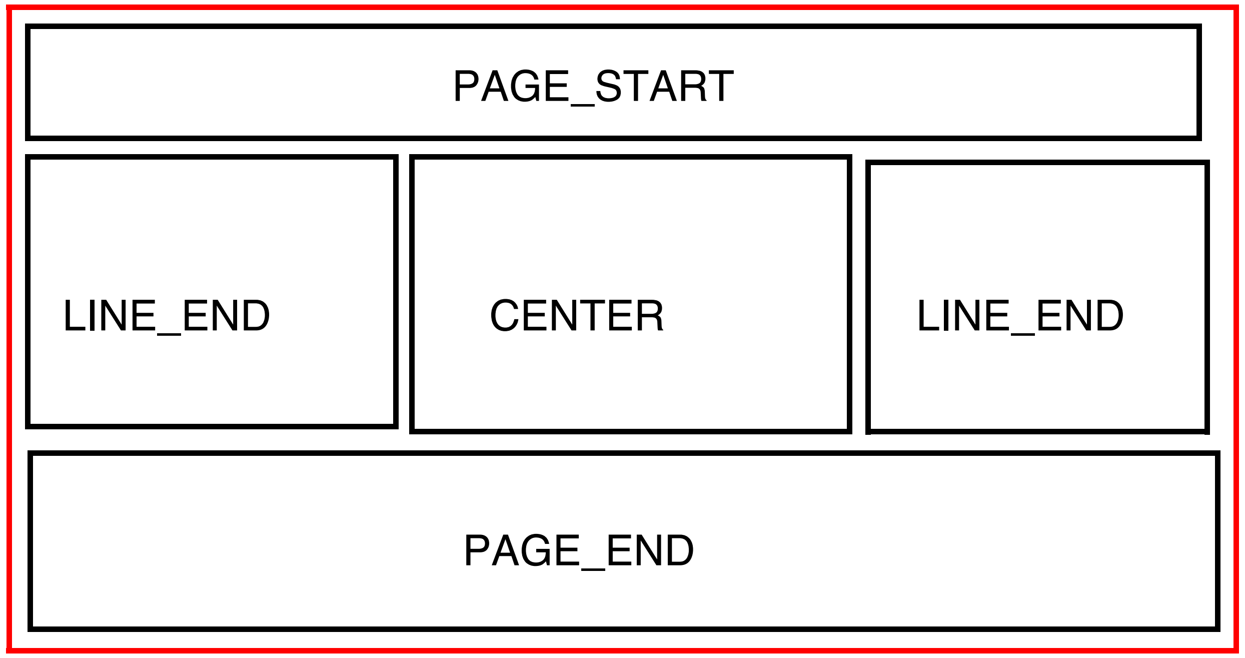
If you set the layout manager to "null" then you can manually layout a content pane or JPanel. This can be done using setBounds on all the children and setSize on the parent.

Often one can do a good layout by combining a number of simple layout managers with a few JPanels. For more complex situations it may be worth implementing your own layout manager.

LayoutManager !@#!

The LayoutManagers I find most useful are:

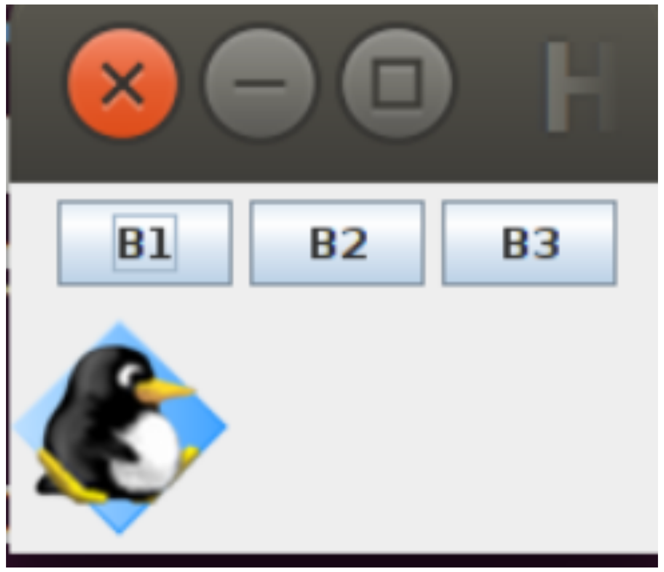
- FlowLayout - the JComponents are placed in a row based on their prefered size, if they don't fit then they "flow over" into multiple rows.
- BorderLayout - in the BorderLayout components are placed in one of 5 areas, so when you add a component you would normally also state the area it should be placed.



BorderLayout

Code below creates a JPanel that has an image in the center and some buttons at the top. Note it uses a BorderLayout for the main layout and the buttons sit within a subpanel of the main panel using a flow layout by default.

```
JPanel mainPanel = new JPanel();
mainPanel.setLayout(new BorderLayout());
JPanel buttonPanel = new JPanel(); // uses flowlayout by default
buttonPanel.add(new JButton("B1"));
buttonPanel.add(new JButton("B2"));
buttonPanel.add(new JButton("B3"));
mainPanel.add(buttonPanel, BorderLayout.PAGE_START);
JLabel imagelabel = new JLabel();
imagelabel.setIcon(new ImageIcon("/usr/share/app-install/icons/supertux.png"));
mainPanel.add(imagelabel, BorderLayout.CENTER);
jframe.getContentPane().add(mainPanel);
```



Australian National University

LayoutManager !@#!

Other handy LayoutManagers are:

- GridLayout - this uses a grid of equally sized cells. When you construct this manager you state the number of rows and columns.
- BoxLayout - this enable you to layout component either horizontally or vertically and also give some control over the alignment of the boxes.

Oracle has a usefull tutorial on LayoutManagers see:

http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html

Making Your Own Components

Note if standard JComponent(s) almost does what you want you can often extend them and include your desired modification by overwriting existing methods.

However, if you need to make something from the ground up then you can simply extend the JComponent class.   This will normally involve overwriting the paintComponent method. The below component will fill the area with white and draw a diagonal black line accross the component.

```
public class DrawArea extends JComponent  {
    public DrawArea() {
        this.setPreferredSize(new Dimension(200,300));
    }
    @Override
    protected void paintComponent(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(Color.black);
        g.drawLine(10, 10, getWidth()-10, getHeight() - 10);
    }
}
```

Example Exam Questions

- Explain what the "event dispatch thread" does in the Swing GUI framework.
- In Swing what is the relationship between a JFrame, a JComponent, and a LayoutManager?
- Implement a simple Swing GUI that has 5 buttons, with labels "B1", "B2", ..., layed out using the FlowLayout manager.

References

Oracle provides some useful tutorials:

`https://docs.oracle.com/javase/tutorial/uiswing/components/index.html`

There is also the jdk documentation which is on the student system which is helpfull if you know the names of the classes you are interested in. See:

`/usr/share/doc/openjdk-7-jre/api/index.html`

This is also available on the internet.

Australian National University

Graphical User Interfaces (GUI)

Eric McCreath

Overview

In this lecture we will:

- discuss the purpose of GUI and GUI toolkits.
- look at the common event driven approaches taken by GUIs.
- look at some example code for different GUI implementations.

Australian National University

What is a GUI?

"The graphical user interface (GUI), is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLIs), which require commands to be typed on a computer keyboard." - **https://en.wikipedia.org/wiki/Graphical_user_interface**

Australian National University

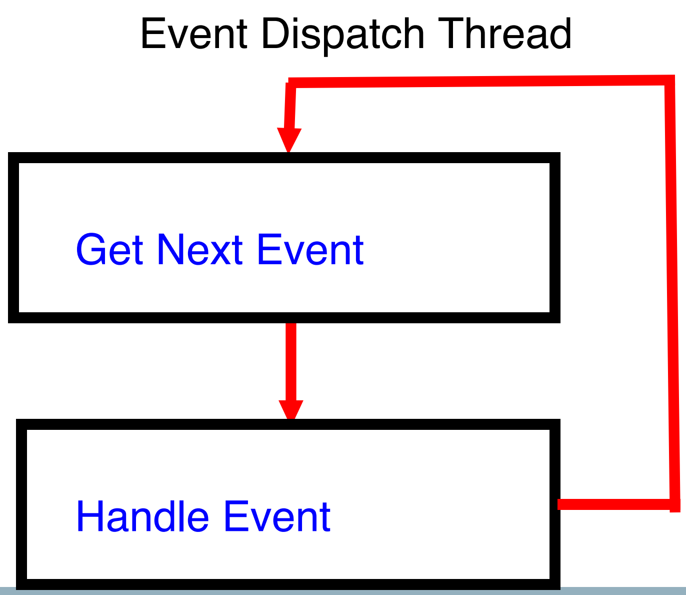
Why use a standard GUI library?

Two key advantages of using existing standard toolkits or libraries for your GUI are:

- You can save an enormous amount of development time as even a simple GUI library is a large and complex item of software that involved considerable amount of coding and testing time.
- The standard set of widgets provided directs you to create a much more standard GUI. Such a standard form of interaction makes it simpler for users to interact with your application.

Events

- Often a GUI toolkit will use an event driven programming paradigm. This is done using a single event handling loop. The loop gets the next event to handle, handles it, and then loops around waiting for the next event.
- Often this event loop code is hidden from the developer implementing an application.
- There would normally only be 1 thread of execution handling all the GUI application code, thus, the developer does not need to worry about race conditions.



Layout and execution

The GUI toolkit will generally provide a convenient way of adding and configuring the components that make up the GUI. Also the toolkit will provide a way of helping the application execution (rendering and gathering user input).

Adding and laying out GUI components programatically can be tedious, hence, there are a number of GUI applicaitons which will help create a GUI.   These will either generate code directly which you then augment, or they create a description of the GUI using a markup language, often XML. Your GUI application can then read these files to create the GUI.