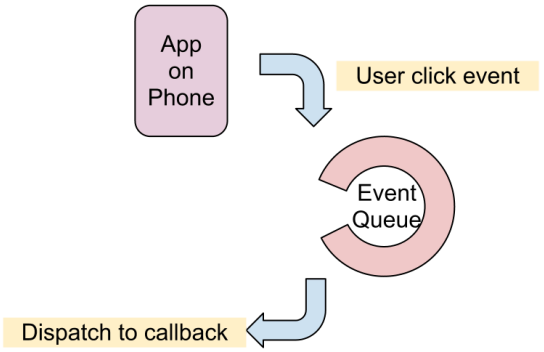
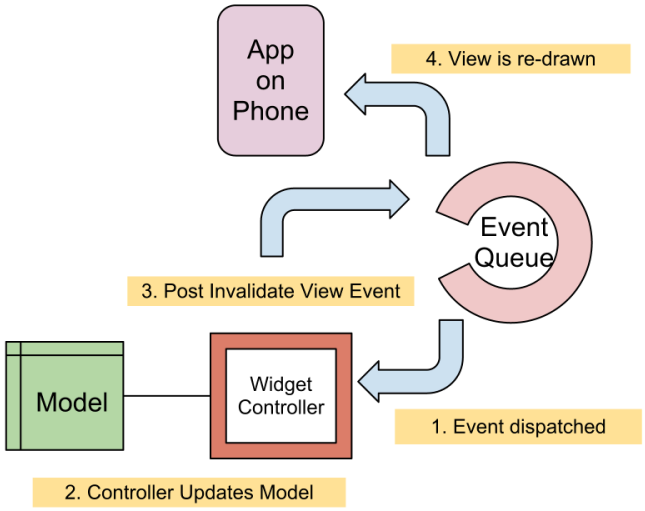
Part 1: Event queue



Single UI thread puts events into the queue and de-queue's them. The processing is slowed if there is too much processing happening in the callback.
Multiple invalidate messages can be combined into 1.



The controller is the class that is implementing the code to handle the dispatched event - often the activity, or can be the view. The model is the data that is shown on the view. For a Text field, the model consists of the textSize, textColor, background, the displayed text - all the properties that can be modified.

Part 2: Callbacks

Image - OnClick event

- Starting with the project from lecture 2, to the MainActivity onCreate method add -

```
ImageView imageView = (ImageView) findViewById(R.id.imageview);
imageView.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
            Toast.makeText(MainActivity.this,
                  "image clicked",
                  Toast.LENGTH_LONG);
      }
});
```

- MainActivity.this is the way to access the this pointer for the outer class. Using only this will access the this pointer for the anonymous inner class for the interface OnClickListener.
- Add the attribute android:id="@+id/imageview" to the image view in activity_main.xml
- Run the application and click on imageview.
- Note that the toast does not show up.
- Put the call to .show() after the makeText for the toast.

```
Toast.makeText(MainActivity.this,
                  "image clicked",
                  Toast.LENGTH_LONG).show();
```

- Now the toast shows whenever the image is clicked.

Button - OnClick event

Button onClick event is handled in a similar way. More on that in next section.

EditText - Text Changed event

- Change the manifest to launch RelativeEdits activity instead of MainActivity.
- In relative_edits.xml - change the number of lines for name to 5.
- Add the following to the onCreate function of RelativeEdits -

```
EditText nameView = (EditText) findViewById(R.id.name);
nameView.addTextChangedListener(new TextWatcher() {
      public void afterTextChanged(Editable arg0) {
            Log.v("Edit", "Text changed to "+arg0.toString());
      }

      public void beforeTextChanged(CharSequence arg0, int arg1,
                  int arg2, int arg3) {}
```

```
            public void onTextChanged(CharSequence arg0, int arg1
                  , int arg2,int arg3) { }
      });
```

- Set up a log filter on the keyword - 'Edit' - as shown in lecture 1.
- Run the application, and type into the name field to see the events being fired.

Spinner - ItemSelected event

- Add the following code to listen for an event from the Spinner

```
Spinner occupationView = (Spinner) findViewById(R.id.occupation);
occupationView.setOnItemSelectedListener(new OnItemSelectedListener() {

      @Override
      public void onItemSelected(AdapterView<?> arg0, View arg1,
                  int arg2, long arg3) {
            Log.v("Edit", "Selected "+ arg3);
      }

      @Override
      public void onNothingSelected(AdapterView<?> arg0) {
            // TODO Auto-generated method stub

      }

});
```

- Run the application and change the value in the spinner to see events on the log.

Checkbox - Checked Changed

- Add the following to the onCreate function of RelativeEdits

```
CheckBox likeView = (CheckBox) findViewById(R.id.like);
likeView.setOnCheckedChangeListener(new OnCheckedChangeListener() {
      @Override
      public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
            Log.v("Edit", "Like: "+ arg1);
      }

});
```

- Rerun the application and change the value of the like button to see events in the log

The above events can also be used to do immediate validation, and feedback to the user. Touch and keyboard listeners in later lecture.

Part 3: Model View

Each view has a model associated with it. The model is the properties that control how the view looks. For text field it is the textSize, textColor, text, etc. The model can be used to get the value the view contains. We can also define our own screen model to handle the data. So, for the RelativeEdits screen, let's define our model containing the fields name, occupation and like.

- Create package named edu.uw.layouts.data
- Add a class called RelativeEditsModel as the following

```
package edu.uw.layouts.data;

public class RelativeEditsModel {
	public String name;
	public String occupation;
	public boolean like;
}
```

- Note that the fields are public. Make them private with getters/setters if there will be validation in the getters and setters.
- Create an instance of the model in the activity to hold the data.

	RelativeEditsModel model = new RelativeEditsModel();

<u>Reading component data</u>

Let's read the data from the widgets into our model when user clicks OK.

- For the button, add an onClick listener

```
	Button ok = (Button) findViewById(R.id.ok);
	ok.setOnClickListener(new OnClickListener() {
		@Override
		public void onClick(View arg0) {
			updateData();
		}

	});
```

- Add the method to update the model to the class

```
private void updateData() {
	model.name = nameView.getText().toString();
	model.occupation =
		occupationView.getSelectedItem().toString();
	model.like = likeView.isChecked();
	Log.v("Edit",
		String.format("%s, %s, %s", model.name,
			model.occupation, model.like));
}
```

- Make the views class level properties, so, they can be accessed by other functions.

```
EditText nameView;
Spinner occupationView;
CheckBox likeView;
```

Updating view - invalidate

Lets allow only alpha characters & spaces in the name field. If user enters invalid characters, then we will turn the field red when the ok button is pressed.

- Add a function to the model to check if the name is valid

```
public boolean isValidName() {
      for(char ch: name.toCharArray()) {
            if(!Character.isLetter(ch)
                  && !Character.isSpaceChar(ch))
                  return false;
      }
      return true;
}
```

- Add a validate function to update the view if the value is not valid

```
private void validate() {
      if(!model.isValidName()) {
            //set the color on name text field to red
            nameView.
                  setBackgroundColor(
                        getResources().getColor(R.color.red));
      } else {
            nameView.setBackgroundColor(Color.TRANSPARENT);
      }
      nameView.postInvalidate();
}
```

- Invoke the validate function on button click

```
            public void onClick(View arg0) {
                  updateData();
                  validate();
            }
```

- Now if the name is invalid the field turns red on pressing ok button

Note: Try and change this so that the field turns red the moment an invalid character is typed.

Reference_:http://www.packtpub.com/article/android-user-interface-development-validating-handling-input-data

Part 4: Intents

Intents are used in Android to specify what the intent of a given activity is, what category it belongs to, and how to launch it from another activity.

Launching next activity from one activity

To launch an activity on the ok button click from RelativeEdits -

- Write a new function to launch the next activity - MainActivity from our current application in this case.

```
private void launchNext() {
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
}
```

- Invoke the function when ok button is clicked.

```
ok.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View arg0) {
                updateData();
                validate();
                launchNext();
        }
});
```

- Now when the OK button is clicked on the first screen, the next screen is launched.

If you press the back button, the Relative edits screen is still there, it has just been paused. Another activity has been brought in front.

Launching existing well-known activity

Well known activities have intents publicly available. You can invoke those activities s.a. browser, market app, camera app, from your activity using intents

- To RelativeEdits.java add the function

```
private void launchBrowser() {
        Intent i = new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://infinut.wordpress.com"));
        startActivity(i);
}
```

.

- Invoke the above function from the cancel button

```
Button cancel = (Button) findViewById(R.id.cancel);
cancel.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View arg0) {
```

```
                launchBrowser();
            }
        });
```

- Now when you press the cancel button on the first screen, the browser is launched.

If you press the back button, the RelativeEdits activity shows again. It was paused, and sent to the background.

Part 5: Sharing data

Simple key value pairs

Key-value pairs can be shared with intents via extra data in the intent. The values can be simple types such as String, boolean, int etc. Serialized objects can also be shared this way, but using java serialization is not recommended.

- Send data from RelativeEdits to MainActivity when it is launched on OK.
- Modify the function launchNext in RelativeEdits to put extra data in the intent.

```
private void launchNext() {
        Intent intent = new Intent(this, MainActivity.class);
        intent.putExtra("name", model.name);
        intent.putExtra("occupation", model.occupation);
        startActivity(intent);
}
```

- In MainActivity, access the data and display it on the textViews in OnCreate function

```
        Bundle bundle = getIntent().getExtras();

        TextView textView = (TextView) findViewById(R.id.textview1);
        textView.setText(bundle.getString("name"));

        TextView textView2 = (TextView) findViewById(R.id.textview2);
        textView2.setText(bundle.getString("occupation"));
```

- Add textview2 as the id for the second textview in activity_main.xml by adding the following property
  android:id="@+id/textview2"
- Now when you run the app and type in a name, select an occupation and click ok, they show up on the next screen

Bundles are also used for intents that are not within the application, and for orientation switches.

Using other published activities - Camera Intent

We can also share data to and from another well known intent. Common example is for taking a picture and then showing it within your view. Sample code is provided for an activity, that has an imageView to display the image, and two buttons - capture to take an image using camera

intent, and select to select an image from the gallery. The activity code shows the following -

- Launching a camera intent to take a picture or gallery intent to select an image
- When the camera or gallery is done, control is returned to our activity, and the onActivityResult function is called.
- We override onActivityResult to get the path of the image file from the returned intent.
- And, we update our image view to show the image.

Sharing data within the application - Singletons

Singletons are Classes that have one and only one static instance that is shared within the application. In a java VM the instance is shared globally within the VM. In Android, it is shared within the application.

- Create a class AppData in the data package

```
public class AppData {
        private static AppData instance = new AppData();
        private AppData() { }
        public static AppData getInstance() {
                return instance;
        }
}
```

- The constructor is made private, so noone else can create another instance.
- There is 1 static instance of the class declared in the class.
- There is a public static function to access the static instance. So, whoever needs to access the class can do so using this function.
- This class can hold instance of our data. Add a member variable of type RelativeEditsModel

```
public class AppData {
        public RelativeEditsModel relativeEditsModel = new RelativeEditsModel();

        private static AppData instance = new AppData();
        ...
}
```

- Remove the instance of RelativeEditsModel from RelativeEdits class. Use the singleton instead by replacing the instance variable model with AppData.getInstance().relativeEditsModel
- Use the same instance to retrieve the name and occupation in MainActivity

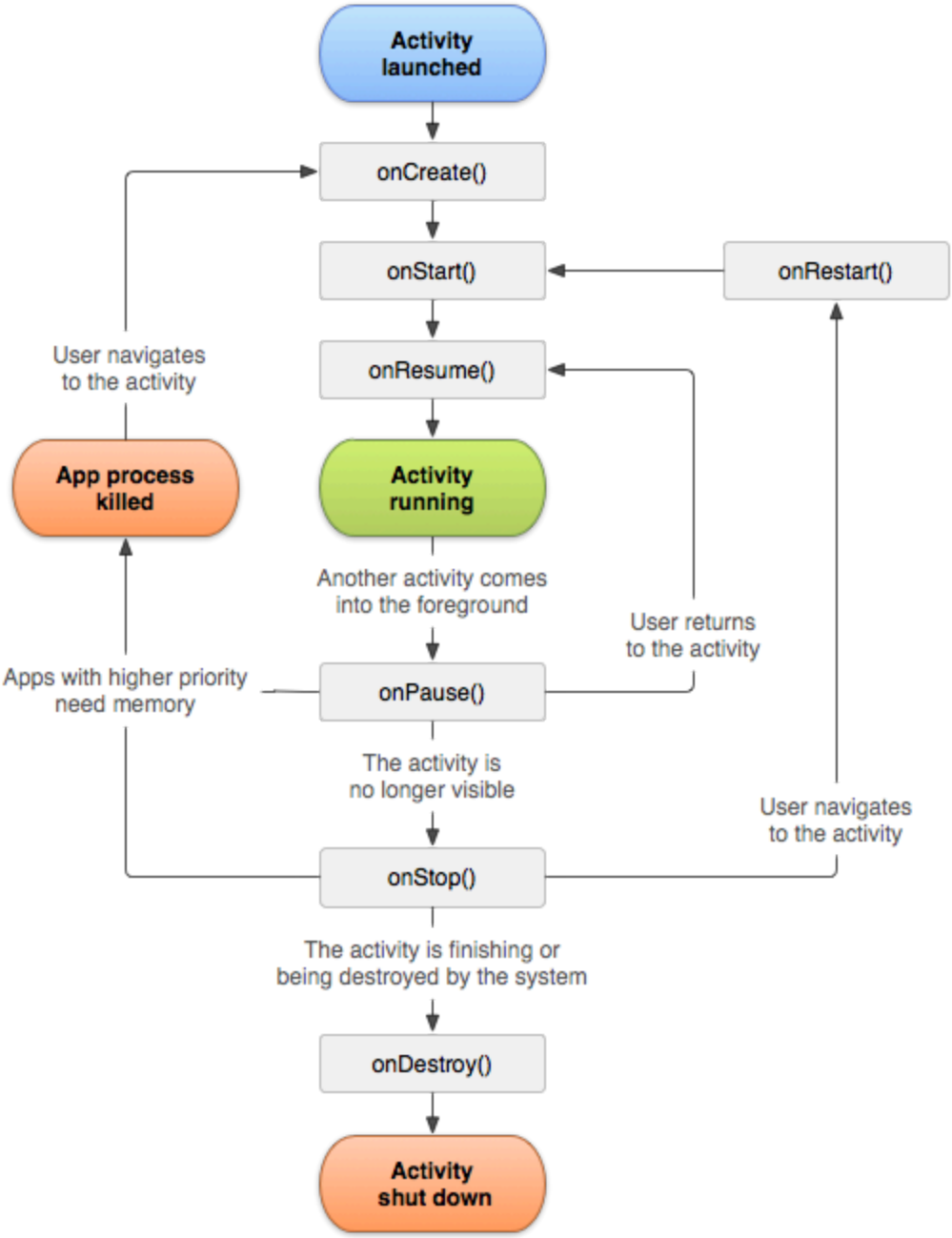
```
TextView textView = (TextView) findViewById(R.id.textview1);
textView.setText(AppData.getInstance().relativeEditsModel.name);

TextView textView2 = (TextView) findViewById(R.id.textview2);
textView2.setText(AppData.getInstance().relativeEditsModel.occupation);
```

Now we can share artibitrarily complex data between activities within the same application. This

is also how you can share the data between activities for your project.

External shares in next course - sql-lite database on phone, web apis.
References - http://developer.android.com/guide/faq/framework.html#3

Part 6: Activity Lifecycle



onPause and onResume do not destroy your Activity. But onDestroy() does. You don't know

when onDestroy might be called. It could be immediately after onPause, or it could be sometime later.
The execution of onResume happens after onCreate as well. So, code that goes in onResume does not need to be repeated onCreate

Show a toast onPause

- Add the following to RelativeEdits class

```
@Override
protected void onPause() {
        super.onPause();
        Toast.makeText(this, "Pausing edit activity", Toast.LENGTH_LONG).show();
}
```

onPause can be used to popup a dialog - perhaps to ask the user to give the app a review (not recommended)

Saving data for orientation switch
Orientation switch causes the activity to be destroyed and recreated. onSaveInstanceState is called before the activity is destroyed to save any state and make it available during recreate. Android will attempt to keep the state for common widgets.

Use bundles or save state to your singleton, and restore it.

- To use bundles to save state, add the following to RelativeEdits class

```
@Override
protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putString("name", nameView.getText().toString());
}
```

- To reset the state on-recreate after orientation change,

```
        if(savedInstanceState!=null) {
                nameView.setText(savedInstanceState.getString("name"));
                Toast.makeText(
                        this, "name loaded from saved state",
                        Toast.LENGTH_LONG).show();
        }
```

References -
http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

Homework 2

Given the screen, when I click ok, go to the next screen and display the data captured in the first screen as text fields.