App bar (again) ... and more

☐ Support Libraries ... on top of "base android"
  ☐ v4 support libraries - can add a lot of code to your app
    ☐ variety: Accessibility, AsyncTaskLoader, ViewPager, Fragments ...
  ☐ v7 support libraries - Primary is AppCompat -- ActionBar based apps
    ☐ ActionBar -- we have seen minor use of action bar
    ☐ AppCompatActivity -- used is most of our apps and examples
    ☐ AppCompatDialog -- extension of Dialog
    ☐ ShareActionProvider -- standard sharing action
    ☐ CardView -- we briefly looked at this
    ☐ GridLayout -- we briefly looked at this
  ☐ MediaRouter, MediaRouteProvider -- Google Cast
    ☐ RecyclerView -- you have used this
    ☐ preference support -- not really looked at this yet
  ☐ v8 support library
    ☐ RenderScript -- May not have time to talk about this
  ☐ v13 support library
    ☐ FragmentCompat
v14 support -- Preference Support

v17 support

Preference support for TV

Leanaback -- UIs on TV

Vector graphics, both static and animated

Annotations -- meta data

Design package: Nav drawers, floating action buttons, snackbar

Custom tabs

More TV and Wear UI library

Major part of this was the AppCompat -- trying to make all apps look similar

AppBar (aka ActionBar) -- done very little with it

https://developer.android.com/training/appbar

Essentially all we have done is the menu:

onCreateMenuOptionsMenu(Menu menu)

similar in a fragment...

shown as the three vertical dots on the right of the bar
"Best" AppBar/Action Bar is the V7 Toolbar

Action Buttons -- another way to handle menu options

- In XML: add an icon (android:icon)
  - app:showAsAction="ifRoom" or "never",
  - pushes are handled in code similar to "overflow menu"

Up Button

- app manifest: android:parentActivityName attribute
- setup in onCreate()

```java
@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my_child);
    // my_child_toolbar is defined in the layout file
    Toolbar myChildToolbar =
        (Toolbar) findViewById(R.id.my_child_toolbar);
    setSupportActionBar(myChildToolbar);
    // Get a support ActionBar corresponding to this toolbar
    ActionBar ab = getSupportActionBar();
    // Enable the Up button
    ab.setDisplayHomeAsUpEnabled(true);
}
```
Action views and action providers

- action view -- interact on the bar, e.g. search window
- action provider -- customized action, typically appears as a button of some kind
- SearchView -- one form of an action view

```xml
<item android:id="@+id/action_search"
    android:title="@string/action_search"
    android:icon="@drawable/ic_search"
    app:showAsAction="ifRoom|collapseActionView"
    app:actionViewClass="android.support.v7.widget.SearchView"/>
```

- Can define your own action view class

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main_activity_actions, ...
    // Configure the search info and add any event listeners...
    return super.onCreateOptionsMenu(menu);
}
```
If the search (action view) option is pressed: expand view
  - onOptionsItemSelected() normally expands view/
  - onActionExpandListener() -- set in onCreateOptionsMenu()
  - onMenuItemActionCollapse() / onMenuItemExpand()
  - (not sure yet how to collapse it via code ... must have a call)

ShareActionProvider -- and example of an action provider

<item android:id="@+id/action_share"
  android:title="@string/share"
  alp:showAsAction="ifRoom"
  app:actionProviderClass="android.support.v7.widget.ShareActionProvider"/>

Ways to configure this share action provider

- Need to specify the title, don’t need an icon.
- Set up an intent with the "sharable content"

MenuItem shareItem = menu.findItem(R.id.action_share);
ShareActionProvider myShareActionProvider =
  (ShareActionProvider) MenuItemCompat.getActionProvider(shareItem);
Intent myShareIntent = new Intent(Intent.ACTION_SEND);
myShareIntent.setType("image/*");
myShareIntent.putExtra(Intent.EXTRA_STREAM, myImageUri);
myShareActionProvider.setShareIntent(myShareIntent);

Can set up a "target rankings" file for each share provider
Custom Action Provider

- **Code setting:**
  
  ```java
  MenuItemCompat.setActionProvider(android.view.MenuItem, ActionProvider)
  ```

- **XML setting:**
  
  ```xml
  <item android:id="@+id/my_menu_item"
       android:title="@string/my_menu_item_title"
       android:icon="@drawable/my_menu_item_icon"
       android:showAsAction="ifRoom"
       android:actionProviderClass="foo.bar.SomeActionProvider"/>
  ```

- **ActionProvider methods**
  
  - **ActionProvider()** -- constructor
  
  - **View onCreateActionView(MenuItem) -- returns the action view**
    
    ```java
    public View onCreateActionView(MenuItem forItem) {
        // Inflate the action provider to be shown on the action bar.
        LayoutInflater layoutInflater = LayoutInflater.from(mContext);
        View providerView = layoutInflater.inflate(R.layout.my_action_provider, null);
        ImageButton button = (ImageButton) providerView.findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) { /* Do something... */ } });
        return providerView;
    }
    ```

  - **onPerformDefaultAction() -- called when menu item is selected**
Example App -- old code but has some useful patterns! TestActionBar in public/csci412

- Not correctly working for some examples, needs upgrade
- Activity classes:
  - DebugActivity -- base activity class for app
    - default base with special constructor
    - implements the IReportBack interface
      - reportBack -> "standard" textView
      - reportTransient -> toast
    - handles default menu option "clear"
    - turns onOptionItemSelected -> onMenuItemSelected call!
  - BaseActionBarActivity -- still not the final activity
    - handles onMenuItemSelected call
  - Other Activities ... based on BaseActionBarActivity
    - Some set up tabs ... tab differences in landscape vs portrait
    - One sets up a list with a "SpinnerAdapter"
    - Listener classes ... with BaseListener ...
- Note: Tab and List navigation via app bar deprecated in support libs
Gradle -- build system for android

- automated build toolkit, can be run from command line
- "has a lot of flexibility"
  - Has a lot of "sensible defaults"
  - Changes needed to add special needs

- Dependencies
  - Modules that need to be added
    - can be "local" or "remote"
      - local -- on build system
      - remote -- using a tool name Maven .. downloads at build time

- Build variants -- done just one variant, but apps can have multiple
  - e.g. Phone version vs Tablet version (can be the same at large app size)

- Manifest entries can be in gradle and auto added to manifest
- Can add signing info into gradle so command line tools can build final .apk
- ProGuard support -- obfuscates Java byte code -- make harder to reverse engineer

- Top level build.gradle (Project: ...)
  - buildscript {...} (repositories and dependencies)
  - allprojects {...} (repositories ...)
  - typically, no changes
Next: build.gradle (Module: app)

Notice, file locations in tree, AndroidStudio shows in one place

Specific app details

- sdk versions, build tool versions, default config
- build types ... various versions here
  - ProGuard enabled here
- Most of the changes go in this file
  - eg. signing configs here

Running gradle from command line

- chmod +x gradlew
- ./gradlew tasks
- ./gradlew assembleDebug
- ./gradlew assembleRelease
Devices have had sensors added to hardware as hardware improves. SensorManager is used to manage sensors. Two kinds of "sensors":

- **Hardware**
- **Software** — really based on multiple hardware sensors

**Motion Sensors:**
- The gravity, linear acceleration, rotation vector, significant motion, step counter, and step detector sensors are either hardware-based or software-based.
- The accelerometer and gyroscope sensors are always hardware-based.

**Position sensors**
- geomagnetic field sensor
- proximity sensor

**Environment sensors**
- ambient temperature, light, pressure, relative humidity, temperature

Google videos on sensors
Using Sensors

```java
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Sensor pressure = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
// pressure may be null if sensor is not available

Sensor.TYPE_X, AMBIENT_TEMPERATURE, GRAVITY, LIGHT, MAGNETIC_FIELD,
PRESSURE, TEMPERATURE (device), PROXIMITY, RELATIVE_HUMIDITY,
ACCELEROMETER (3 axis), GYROSCOPE (3 axis), LINEAR_ACCELERATION (3 axis),
ROTATION_VECTOR (orientation of device software),
GEOMAGNETIC_ROTATION_VECTOR (not as accurate as ROTATION, less
power), MOTION_DETECT, STATIONARY_DETECT, SIGNIFICANT_MOTION (wake-up
sensor)

Body and exercise sensors

HEART_BEAT, HEART_RATE, LOW_LATENCY_OFFBODY_DETECT, STEP_COUNTER,
STEP_DETECTOR

List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
List<Sensor> allSensors = sensorManager.getSensorList(Sensor.TYPE_GYROSCOPE);

Hardware sensors are earlier in the list, software ones later in the list
Wake-up sensor vs non-wake up sensor (during lock mode)
(API 24, 7) added dynamic Sensors (Andriod Things Platform)
Ways to find dynamic sensors ...
```
Methods a sensor has ... help determines information about it

- getName()
- getPower() -- depends on sensor what it means
- getMaximumRange(), getResolution, getMinDelay
- getVersion, getVendor
- getMaxDelay -- (API 21) slowest a sensor can report repeated events
- getReportingMode() -> REPORTING_MODE_X:
  - CONTINUOUS, ON_CHANGE, ONE_SHOT, SPECIAL_TRIGGER

Monitoring a Sensor

- Depends on the reporting mode ... All but one shot

  - SensorEventListener -- as you might expect!
    - onSensorChanged(SensorEvent)
    - onAccuracyChanged(Sensor, accuracy)

- SensorEvent:
  - sensor, accuracy, values (float array), timestamp in nanoseconds
  - accuracy: SENSOR_STATUS_X: ACCURACY_LOW, ACCURACY_MEDIUM, ACCURACY_HIGH, UNRELIABLE, NO_CONTACT
One shot sensors:

- `TriggerEventListener` and `onTrigger(TriggerEvent)` should be used

Example code: (simple version)

```java
Sensor sensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
mySensorEventListener = new SensorEventListener() {
    public void onSensorChanged(SensorEvent evt) {
        ... 
    }
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        ... 
    }
} 

sensorManager.registerListener(mySensorEventListener, sensor, SensorManager.SENSOR_DELAY_NORMAL);
```

- Overloaded `registerListener` takes 4th argument of max delay in microseconds
- Note: you can also have your activity or fragment implement `SensorEventListener` interface

Event values:

- Depends on sensor, may have 1 or more values. Many return 3.
- E.g. `RELATIVE_HUMIDITY` returns 1, value is a percent
- 1 value for: `LIGHT`, `PRESSURE`, `PROXIMITY`, `*TEMPATURE`, `STATIONARY_DETECT`, `MOTION_DETECT`, `HEART_BEAT`, `LOW_LATENCY_OFF_BODY_DETECT`, `SIGNIFICANT_MOTION`, `HEART_RATE`, `STEP_COUNTER`, `STEP_DETECTOR`
Emulator can emulate a number of sensors

Best practices:

- Always verify sensors exist
- Provide alternatives to sensor input ... for devices that don’t have that sensor
- Don’t use deprecated Sensor types
- Be conservative when selecting Sensor reporting frequencies
  - e.g. pressure changes very slow ... except in certain situations
- onSensorChanged handler may be called a lot ... minimum of work done here
- When done using them, make sure to unregister the listeners
- When you choose to work with a particular sensor, you have a lot of work to understand and use well

Professional Android book talks a lot more about:

- device movement and orientation (useful with VR or AR)
- Has a section of creating a compass and artificial horizon (pilots might like this!)
- Also a "weather station" app.
- Google has an Activity Recognition API that tries to figure out what you are doing!
  - IN_VEHICLE, ON_BICYCLE, ON_FOOT, STILL, WALKING, RUNNING, UNKNOWN, TILTING ...
<< not done >>