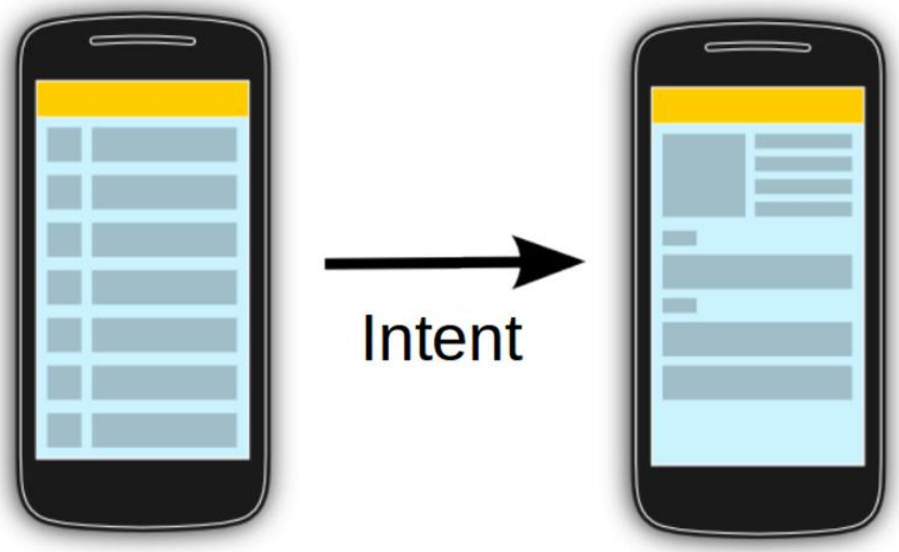
Advanced Programming 2
Recitation 13 – Android Part II

Roi Yehoshua
2017

Activities and Intents

Intent

- An **intent** is an abstract description of an operation to be performed
- It allows application components to request functionality from other apps
- For example, to start an activity you can use the method **startActivity**(intent)



Starting Another Activity

- Example for starting a second activity when clicking a button:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    // Called when user taps the Send button
    public void sendMessage(View view) {
        EditText editText = (EditText)findViewById(R.id.editText);

        Intent intent = new Intent(this, DisplayMessageActivity.class);
        String msg = editText.getText().toString();
        intent.putExtra("message", msg);
        startActivity(intent);
    }
}
```

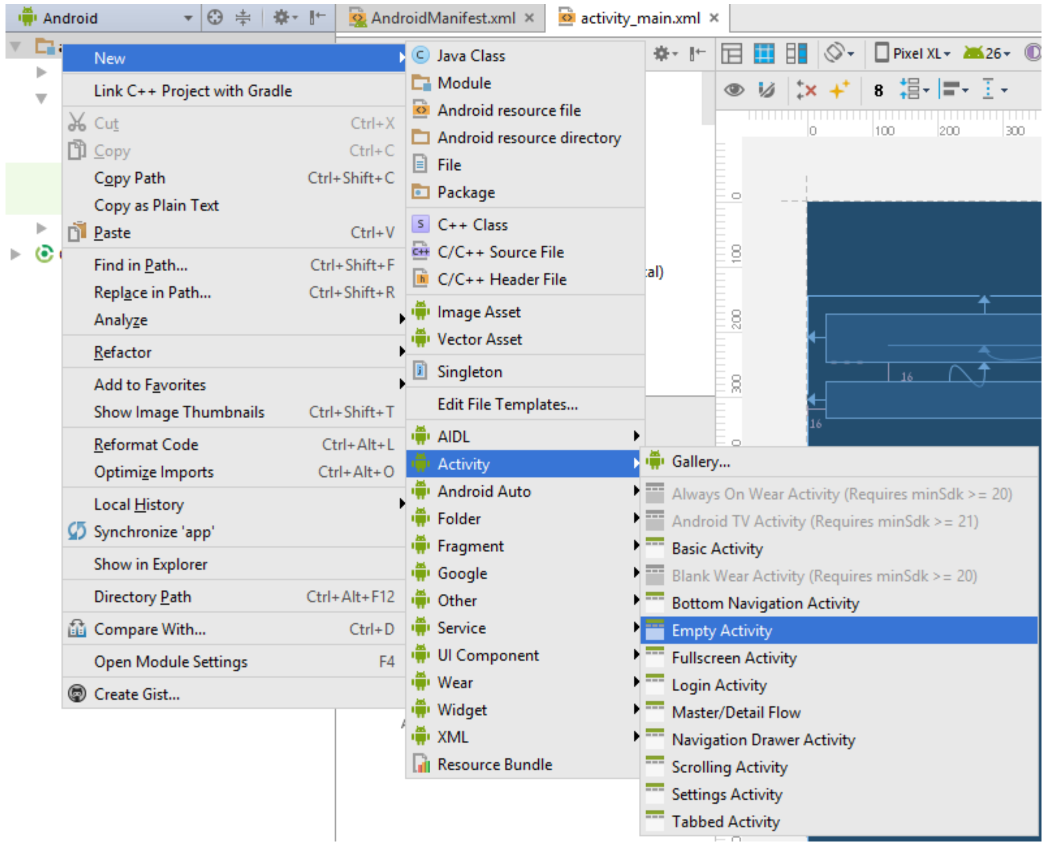
The Intent c'tor takes two parameters: a Context (an Activity is a subclass of Context) and the class of the app component the system should deliver the intent to

The putExtra() method can add data to the intent as key-value pairs

The startActivity() method starts an instance of the activity specified by the Intent

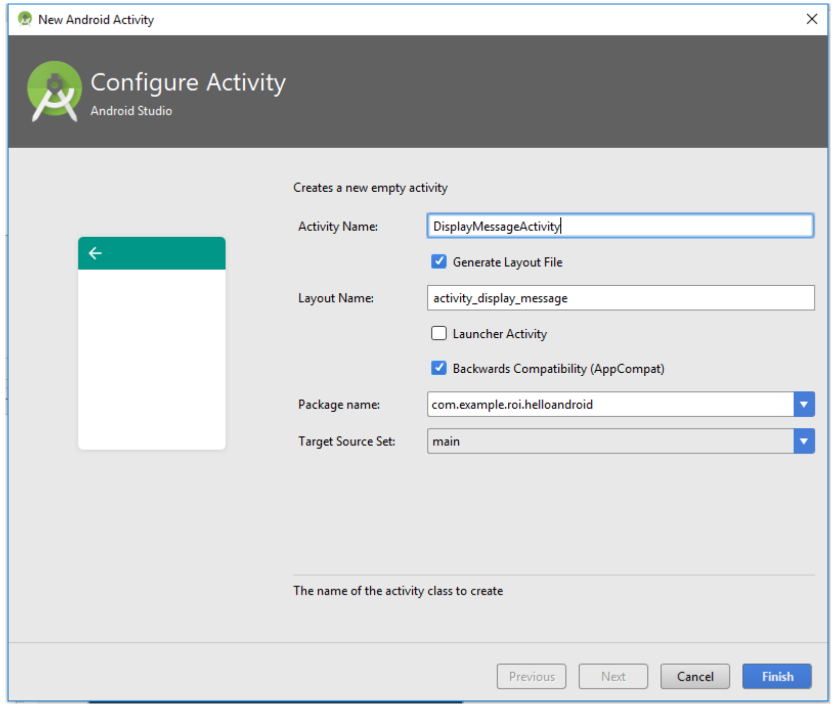
Create the Second Activity

- In the **Project** window, right-click the **app** folder and select **New > Activity > Empty Activity**



Create the Second Activity

- In the **Configure Activity** window, enter "DisplayMessageActivity" for **Activity Name** and click **Finish**
- Android Studio automatically does three things:
  - Creates the DisplayMessageActivity.java file
  - Creates the activity_display_message.xml layout file
  - Adds the required <activity> element in AndroidManifest


New Android Activity

Configure Activity

Android Studio

Creates a new empty activity

Activity Name: DisplayMessageActivity

[✓] Generate Layout File

Layout Name: activity_display_message

[ ] Launcher Activity

[✓] Backwards Compatibility (AppCompat)

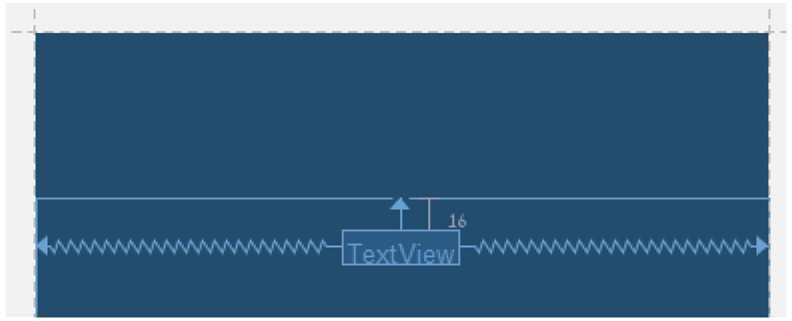
Package name: com.example.roi.helloandroid

Target Source Set: main

The name of the activity class to create

Add a TextView

- The new activity includes a blank layout file
- Add a text view where the message will appear
  - Open the file **app > res > layout > activity_display_message.xml**.
  - Click **Turn On Autoconnect** in the toolbar
  - From the **Pallete** window, drag a **TextView** into the layout and place it near the top of the layout, near the center so it snaps to the vertical line that appear, and then drop it
  - Autoconnect adds constraints to place the view in the horizontal center



Display the Message

- In DisplayMessageActivity.java, add the following code to the onCreate() method:

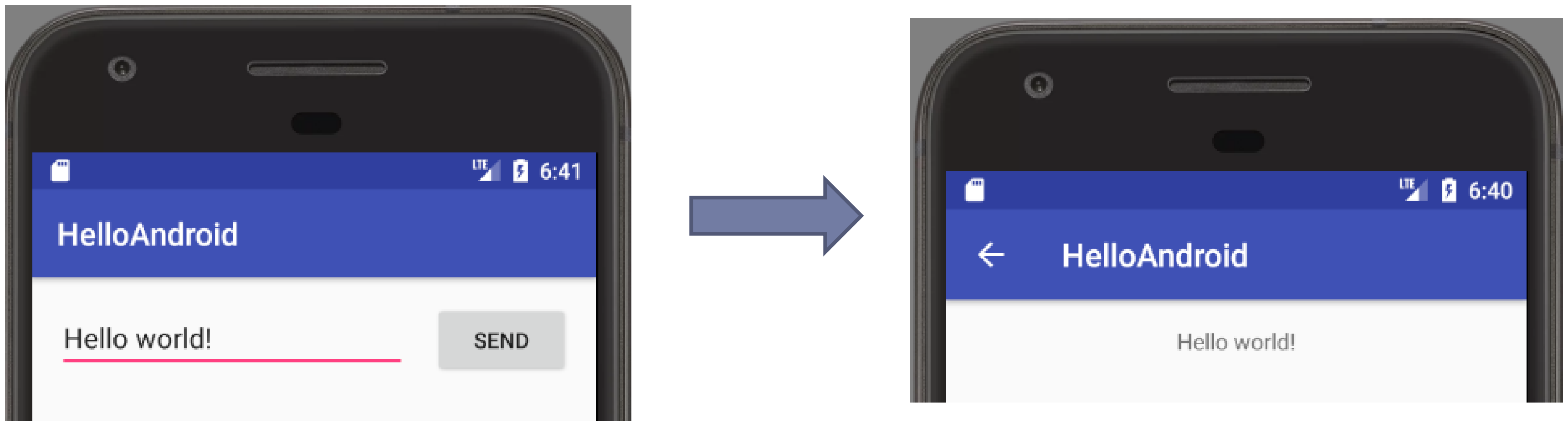
getIntent() returns the intent that started this activity

```java
public class DisplayMessageActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_message);

        // Get the Intent that started this activity and extract the message
        Intent intent = getIntent();
        String msg = intent.getStringExtra("message");

        // Display the message on the textview
        TextView textView = (TextView) findViewById(R.id.textViewMsg);
        textView.setText(msg);
    }
}
```

Add Up Navigation

- Each screen that is not the main entrance to your app should provide navigation so the user can return to its logical parent screen in the app hierarchy
- You declare which activity is the logical parent in the AndroidManifest.xml file:

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity"></activity>
```

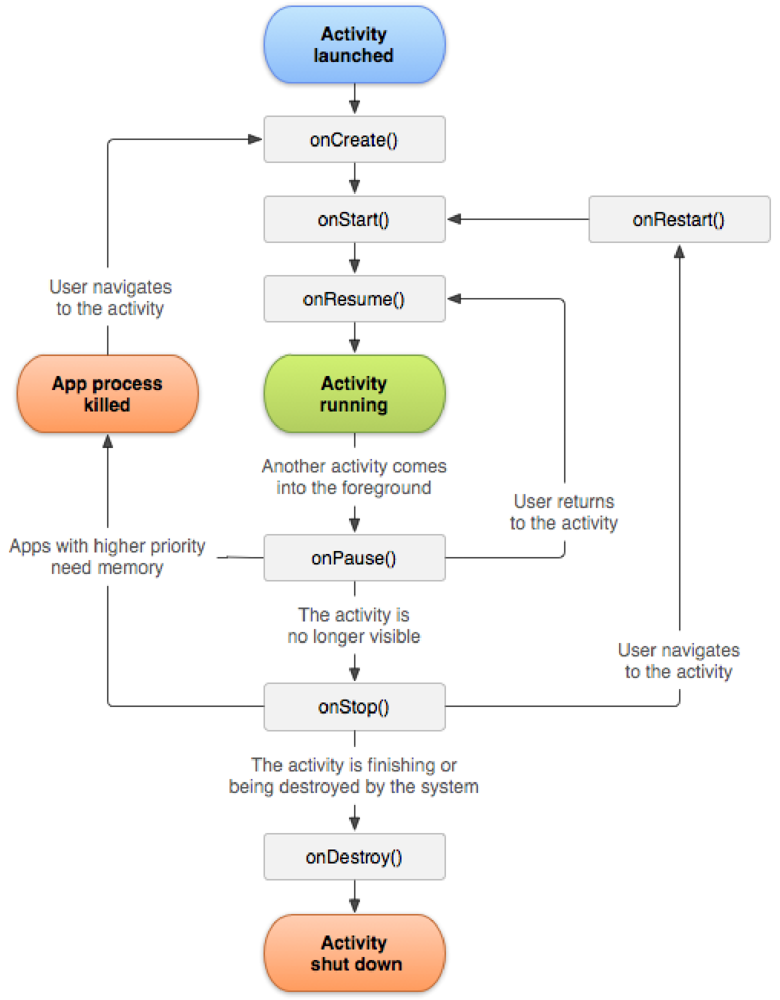


Passing Objects To Another Activity

- In order to pass an object to another activity, your custom class should implement the Serializable interface
- Then you can pass object instances in the intent extra using the putExtra(Serializable..) method
- To retrieve the object in the second activity call getIntent().getSerializableExtra()

```
// To pass an object
intent.putExtra("MyClass", obj);

// To retrieve object in second Activity
getIntent().getSerializableExtra("MyClass");
```

Activity Life Cycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Connecting to the Network

Network Communication

- In order to perform network operations in your application, your manifest must include the INTERNET permission:

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.roi.httpconnection">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

HttpURLConnection

- **HttpURLConnection** object is used to send and receive data via HTTP
- Uses of this class follow a pattern:
  - Define the URL of the request
  - Obtain a new HttpURLConnection by calling **URL.openConnection()** and casting the result
  - Optionally upload a request body
    - Call **setDoOutput(true**) to include a request body
    - Transmit data by writing to the stream returned by **getOutputStream()**
  - Read the response body from the stream returned by **getInputStream()**
  - Disconnect by calling **disconnect()** on the HttpURLConnection

```
public String makeServiceCall(String reqUrl) {
    String response = null;
    try {
        URL url = new URL(reqUrl);
        HttpURLConnection con =
(HttpURLConnection)url.openConnection();
        con.setRequestMethod("GET");
        InputStream in = con.getInputStream();
        response = convertStreamToString(in);
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return response;
}
```

Convert the InputStream to a String

- The **InputStream** represents the text of the response body
- This is how you would convert the InputStream to a string:

```
private String convertStreamToString(InputStream in) {
    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
    StringBuilder sb = new StringBuilder();

    String line;
    try {
        while ((line = reader.readLine()) != null) {
            sb.append(line).append("\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return sb.toString();
}
```

AsyncTask

- To avoid creating an unresponsive UI, Android 3.0 (API level 11) and higher requires you to perform network operations on a thread other than the main UI thread
  - If you don't, a NetworkOnMainThreadException is thrown
- **AsyncTask<Params, Progress, Results>** allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers
- It has 3 generic types:
  - **Params** – the type of the parameters sent to the task upon execution
  - **Progress** – the type of the progress units published during the background computation
  - **Result** – the type of the result of the background computation

AsyncTask

- AsyncTask goes through 4 steps:
  - **onPreExecute** - invoked on the UI thread before the task is executed
    - This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
  - **doInBackground(Params...)** - invoked on the background thread immediately after onPreExecute()
    - This step is used to perform background computation that can take a long time
    - The parameters of the asynchronous task are passed to this step
    - The result of the computation returned by this step will be passed back to the onPostExecute() step
    - This step can also use publishProgress(Progress...) to publish one or more units of progress
  - **onProgressUpdate(Progress...) -** invoked on the UI thread after a call to publishProgress(Progress...)
    - This method is used to display any form of progress in the UI while the background computation is still executing
  - **onPostExecute(Result)** - invoked on the UI thread after the background computation finishes
    - The result of the background computation is passed to this step as a parameter
- AsyncTask must be subclassed to be used
  - The subclass will override at least one method (doInBackground(Params...)), and most often will override a second one (onPostExecute(Result))

GetMazeTask Example

```
public class GetMazeTask extends AsyncTask<String, Void, Maze> {
    private final static String TAG = "GetMazeTask";
    private Activity activity;
    public GetMazeTask(Activity activity) {
        this.activity = activity;
    }
    @Override
    protected Maze doInBackground(String... params) {
        String qs = "name=" + params[0] + "&rows=" + params[1] +
"&cols=" + params[2];
        String apiUrl =
"http://10.0.2.2/MazeWebApp/api/SinglePlayer/GenerateMaze?" + qs;

        HttpHandler handler = new HttpHandler();
        String jsonStr = handler.makeServiceCall(apiUrl);
        Log.d(TAG, "Response from url: " + jsonStr);
        if (jsonStr == null) {
            Log.e(TAG, "Couldn't get json from server.");
            return null;
        }
        Maze maze = getMazeFromJson(jsonStr);
        if (maze == null) {
            Log.e(TAG, "Json parsing error: " + e.getMessage());
        }
        return maze;
    }
```

```
    @Override
    protected void onPostExecute(Maze maze) {
        super.onPostExecute(maze);

        // Start the maze activity with the parsed
maze object
        Intent intent = new Intent(activity,
MazeActivity.class);

        intent.putExtra("maze", maze);
        activity.startActivity(intent);
    }
}
```

The emulator runs behind a virtual router/firewall service that isolates it from your development machine network interfaces

10.0.2.2 is a special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)

Parsing JSON in Android

- Android provides the classes JSONObject and JSONArray to manipulate JSON data
- For parsing a JSON object, you need to create an object of class JSONObject and specify a string containing JSON data to it:

```
String jsonStr;
JSONObject jsonObj = new JSONObject(jsonStr);
```

- The method **getString()** returns the string value of the specified key:

```
String id = jsonObj.getString("id");
```

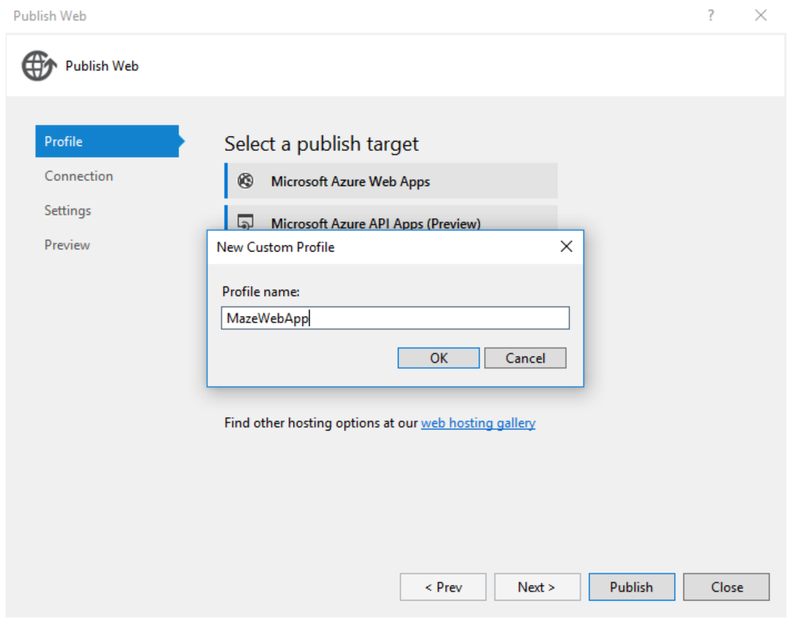
- To read nested JSON objects or arrays you can use the methods **getJSONObject()** and **getJSONArray()** respectively

Parsing Maze JSON Example

```java
private Maze getMazeFromJson(String jsonStr) {
    try {
        JSONObject mazeObj = new JSONObject(jsonStr);
        Maze maze = new Maze();
        int rows = mazeObj.getInt("Rows");
        int cols = mazeObj.getInt("Cols");
        int[][] data = new int[rows][cols];
        String mazeStr = mazeObj.getString("Maze");
        int count = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                char ch = mazeStr.charAt(count++);
                data[i][j] = Character.getNumericValue(ch);
            }
        }
        maze.setData(data);
        JSONObject initialPosObj = mazeObj.getJSONObject("Start");
        Position initialPos = new Position(initialPosObj.getInt("Row"), initialPosObj.getInt("Col"));
        maze.setInitialPos(initialPos);
        JSONObject exitPosObj = mazeObj.getJSONObject("End");
        Position exitPos = new Position(exitPosObj.getInt("Row"), exitPosObj.getInt("Col"));
        maze.setExitPos(exitPos);
        return maze;
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
```

Publish Web API to IIS

- In order to call the web service from your Android app, you first must publish it to an IIS web server
  - The IIS server express in Visual Studio cannot
- Right-click the Web API project and choose Publish
- Create a custom Publish profile

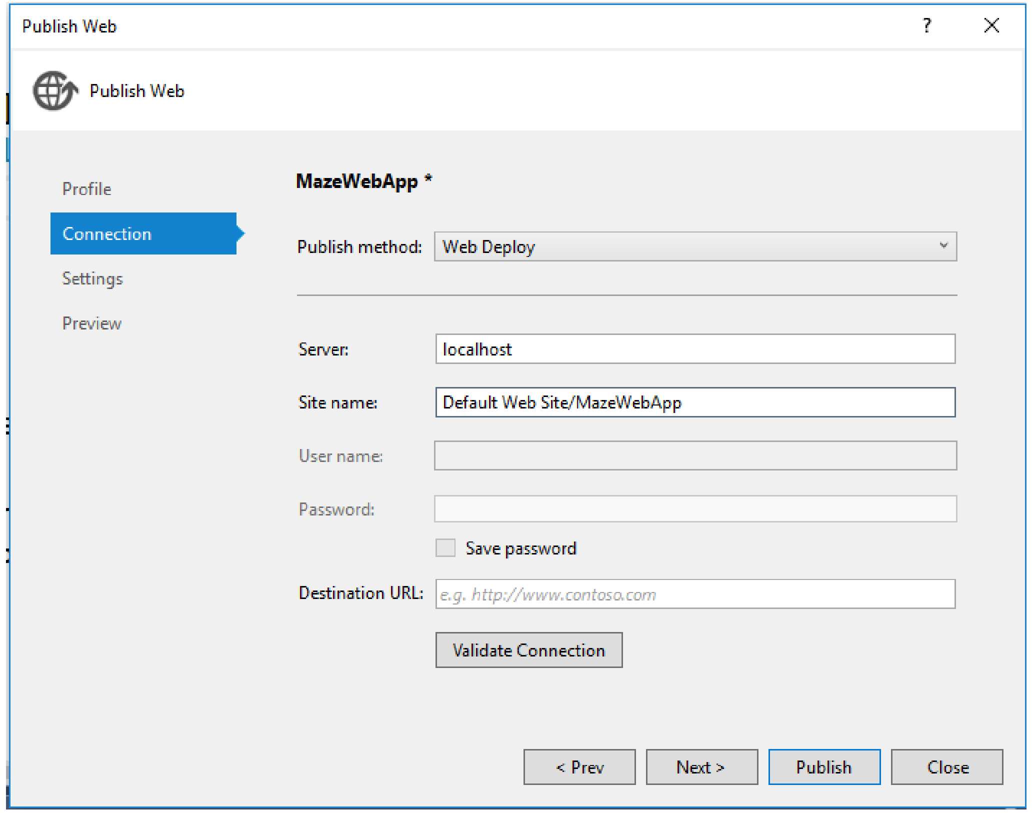


Profile name: MazeWebApp

Find other hosting options at our web hosting gallery

Publish Web API to IIS

- Specify the web server address and the site/folder to which you want to publish
- Click Publish


Publish Web

Publish Web

MazeWebApp *

Publish method: Web Deploy

Server: localhost

Site name: Default Web Site/MazeWebApp

User name:

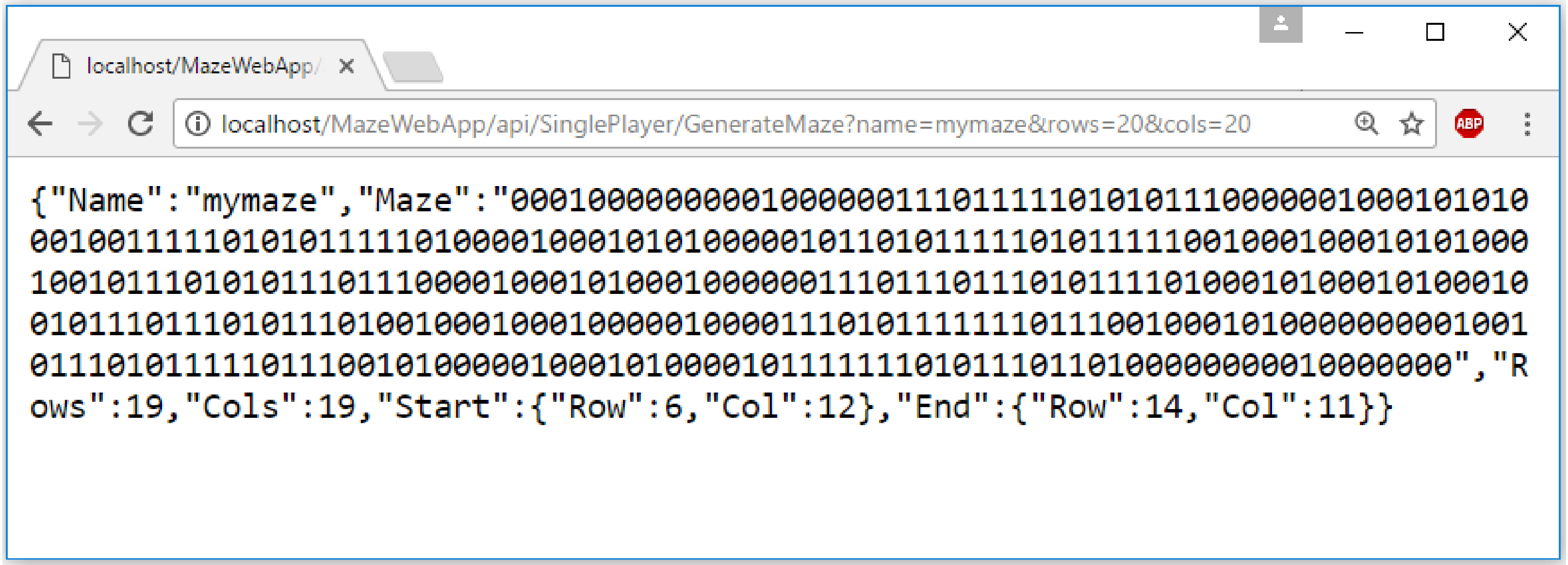
Password:

Save password

Destination URL: e.g. http://www.contoso.com

Publish Web API to IIS

- Verify that you can access the service from its new address



localhost/MazeWebApp/api/SinglePlayer/GenerateMaze?name=mymaze&rows=20&cols=20

```
{"Name":"mymaze","Maze":"0001000000000100000011101111101010111000000100010101010
0010011111010101111101000010001010100000101101011111010111110010001000101010000
1001011101010111011100001000101000100000011101110111010111101000101000101000010
0101110111010111010010001000100000100001110101111111011100100010100000000001001
0111010111110111001010000010001010000101111111010111011010000000000010000000","R
ows":19,"Cols":19,"Start":{"Row":6,"Col":12},"End":{"Row":14,"Col":11}}
```

Canvas

Drawing Objects

- To draw something, you need two basic components:
  - a **Canvas** that defines the shapes that you want to draw on the screen
  - a **Paint** to describe the colors and styles for the drawing
- Draw primitive shapes using **drawRect()**, **drawOval()**, and **drawArc()**
  - Change whether the shapes are filled, outlined, or both by calling setStyle()
- Draw more complex shapes using the **Path** class
  - Define a shape by adding lines and curves to a Path object, then draw the shape using drawPath()
- To draw text use **drawText()**
  - Specify the typeface by calling **setTypeface()**, and the text color by calling **setColor()**
- To draw bitmaps use **drawBitmap()**

Drawing a Maze Example

- To get a predefined canvas object, you need to create custom View component and draw with a Canvas in the onDraw() method
- Create the Paint objects in the constructor of this class
  - Creating objects ahead of time is an important optimization since views are redrawn very frequently, and many drawing objects require expensive initialization

```java
public class MazeBoard extends View {
    private Paint freeCellPaint;
    private Paint wallPaint;

    public MazeBoard(Context context) {
        super(context);

        freeCellPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        freeCellPaint.setColor(Color.WHITE);
        freeCellPaint.setStyle(Paint.Style.FILL);
        wallPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        wallPaint.setColor(Color.BLACK);
        wallPaint.setStyle(Paint.Style.FILL);
    }
}
```

Handling Layout Events

- In order to properly draw your custom view, you need to know what size it is
- For that purpose, you need to override **onSizeChanged()**
  - This method is called when the view is first assigned a size, and when its size changes
  - Calculate positions, dimensions, and any other values related to your view's size

```
public class MazeBoard extends View {
    private int mazeWidth, mazeHeight;
    private int cellWidth, cellHeight;
    …
    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        super.onSizeChanged(w, h, oldw, oldh);
        // Account for padding
        int xpad = getPaddingLeft() + getPaddingRight();
        int ypad = getPaddingTop() + getPaddingBottom();

        mazeWidth = w - xpad;
        mazeHeight = h - ypad;
        cellWidth = mazeWidth / maze[0].length;
        cellHeight = mazeHeight / maze.length;
    }
}
```

Drawing the Maze

- Override the onDraw() method

The parameters for Rect c'tor are (left, top, right, bottom, Paint)

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    for (int i = 0; i < maze.length; i++) {
        for (int j = 0; j < maze[i].length; j++) {
            Paint paint;
            if (maze[i][j] == 1)
                paint = wallPaint;
            else
                paint = freeCellPaint;

            Rect rect = new Rect(j * cellWidth, i * cellHeight, (j + 1) *
cellWidth, (i + 1) * cellHeight);
            canvas.drawRect(rect, paint);
        }
    }
}
```

Drawable Resources

- A drawable resource is a graphic that can be drawn to the screen and which you can:
  - Retrieve with APIs such as getDrawable(int)
  - Apply to another XML resource with attributes such as android:drawable and android:icon
- There are several different types of drawables, e.g. a bitmap file, a nine-patch file, etc.
- For example, for an image saved at res/drawable/myimage.png:
  - This layout XML applies the image to a View:

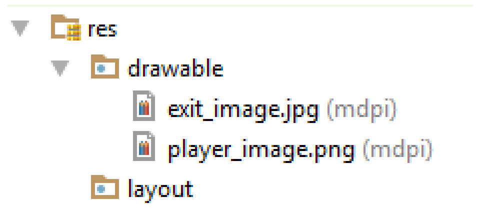
```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

  - The following application code retrieves the image as a Bitmap:

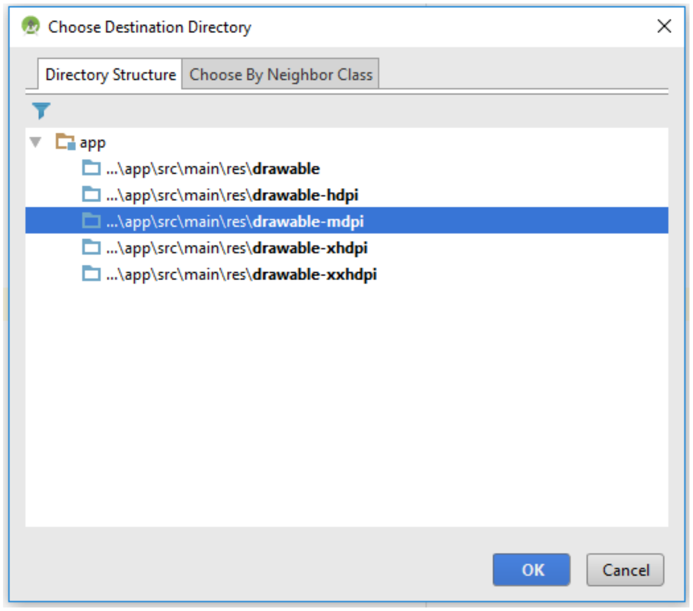
```
Resources res = getResources();
Bitmap bitmap = BitmapFactory.decodeResource(res, R.drawable.player_image);
```

Copy Images to Android Studio

- Add your images to the res/drawable folder:
  - Go to your image in windows, then press Ctrl+C or right-click and Copy
  - Go to the res/drawable folder and press Ctrl+V or right click it and Paste
  - Choose in which drawable subfolder to place the image (hdpi, mdpi, etc.)

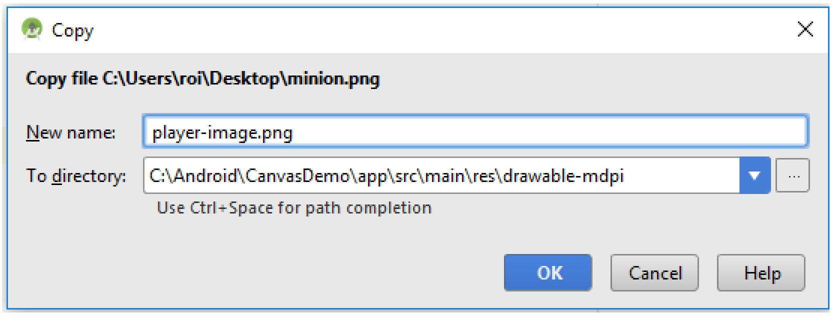


```
res
  drawable
    exit_image.jpg (mdpi)
    player_image.png (mdpi)
  layout
```



Choose Destination Directory

```
app
  ...\app\src\main\res\drawable
  ...\app\src\main\res\drawable-hdpi
  ...\app\src\main\res\drawable-mdpi
  ...\app\src\main\res\drawable-xhdpi
  ...\app\src\main\res\drawable-xxhdpi
```



Copy

Copy file C:\Users\roi\Desktop\minion.png

New name: player-image.png

To directory: C:\Android\CanvasDemo\app\src\main\res\drawable-mdpi

Use Ctrl+Space for path completion

Draw Images on the Canvas

Canvas.drawBitmap() gets the following parameters:

- **bitmap** - the bitmap to be drawn
- **src** - May be null. The subset of the bitmap to be drawn
- **dst** - The rectangle that the bitmap will be scaled/translated to fit into
- **paint** - May be null. The paint used to draw the bitmap

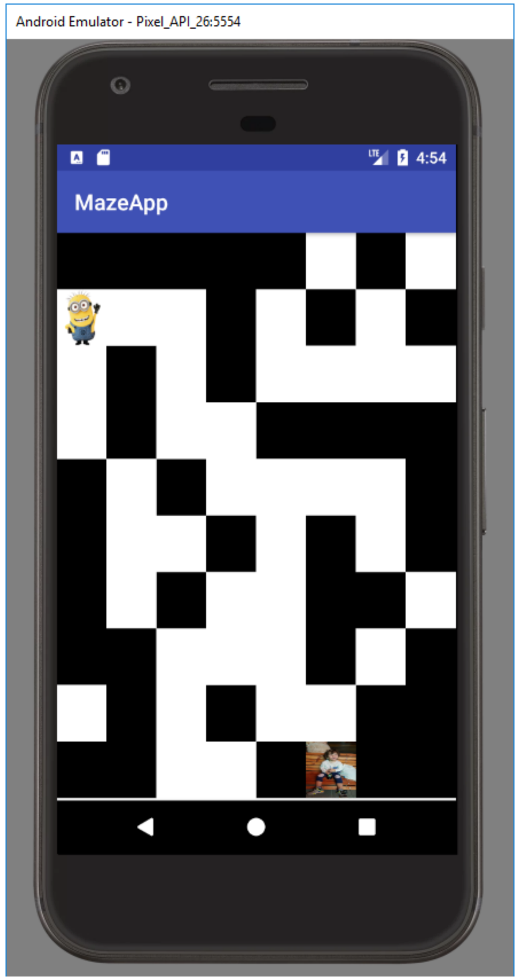
```
public class MazeBoard extends View {
    private Bitmap playerImage;
    private Bitmap exitImage;
    private Point playerPos;
    private Point exitPos;

    public MazeBoard(Context context, int[][] maze, Point playerPos, Point exitPos) {
        …
        Resources res = getResources();
        playerImage = BitmapFactory.decodeResource(res, R.drawable.player_image);
        exitImage = BitmapFactory.decodeResource(res, R.drawable.exit_image);
    }
    @Override
    protected void onDraw(Canvas canvas) {
        …
        for (int i = 0; i < maze.length; i++) {
            for (int j = 0; j < maze[i].length; j++) {
                Rect rect = new Rect(j * cellWidth, i * cellHeight, (j + 1) * cellWidth,
(i + 1) * cellHeight);
                if (i == playerPos.y && j == playerPos.x)
                    canvas.drawBitmap(playerImage, null, rect, null);
                else if (i == exitPos.y && j == exitPos.x)
                    canvas.drawBitmap(exitImage, null, rect, null);
                else
                    canvas.drawRect(rect, paint);
            }
        }
    }
}
```

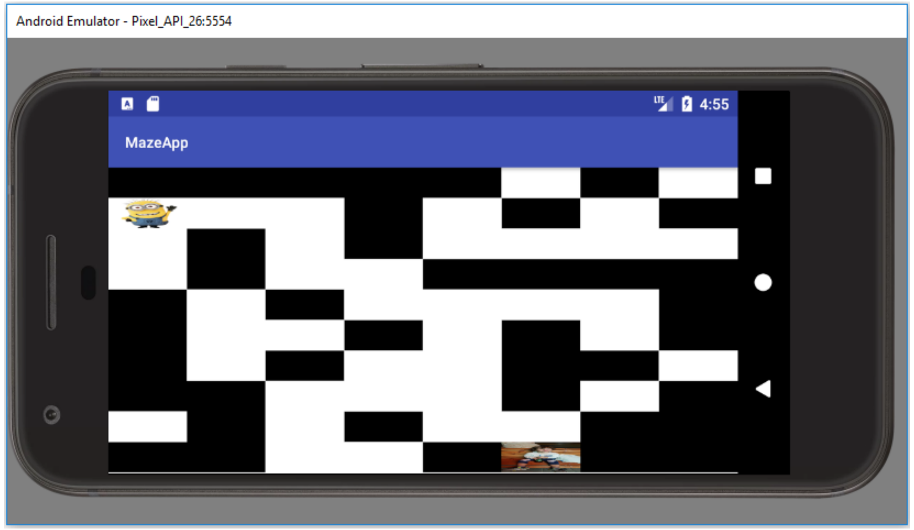
Maze Activity

- Add MazeActivity without generating a layout file
- Copy the following code to the onCreate() method:

```
public class MazeActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        int[][] maze = {
            {1, 1, 1, 1, 1, 0, 1, 0},
            …
            {1, 1, 0, 0, 1, 0, 1, 1}
        };

        Point playerPos = new Point(0, 1);
        Point exitPos = new Point(5, 9);
        MazeBoard mazeBoard = new MazeBoard(this, maze, playerPos, exitPos);
        setContentView(mazeBoard);
    }
}
```

Run the App

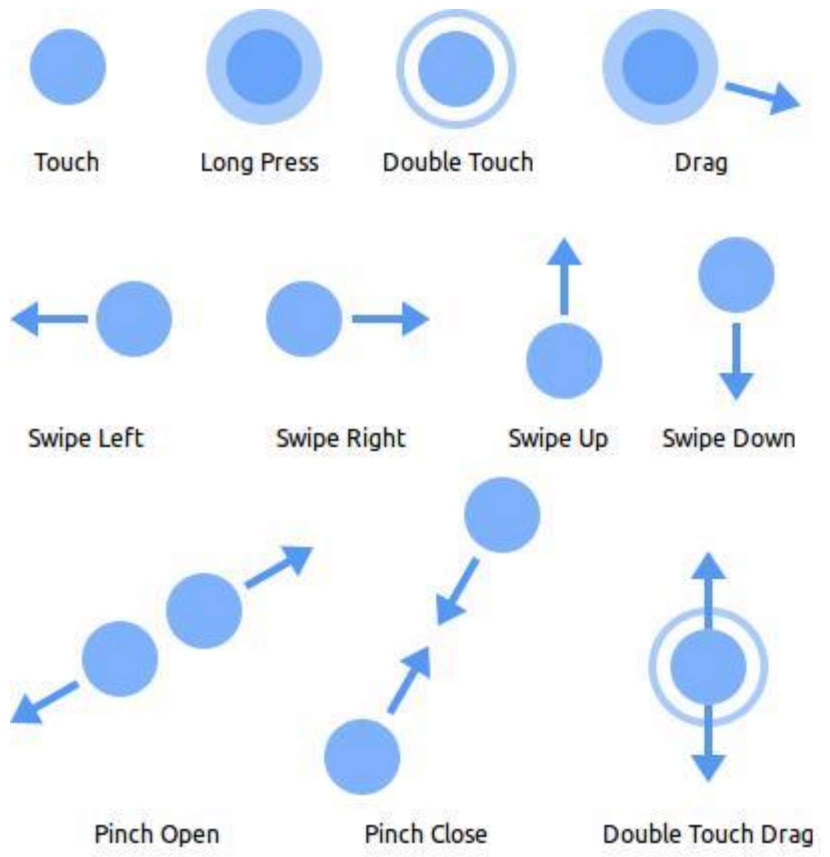

Android Emulator - Pixel_API_26:5554
4:54
MazeApp


Android Emulator - Pixel_API_26:5554
4:55
MazeApp

Gestures Detection

Detecting Common Gestures

- Gestures are motions that trigger interactions between the touch screen and the user
- They last from the first touch on the screen to the point when the last finger leaves the surface



Responding to Touch Events

- In order to make your view respond to touch events, you must implement the onTouchEvent() method
- This method receives a **MotionEvent** object
- Motion events describe movements in terms of an action code and a set of axis values
- For example, when the user first touches the screen, the system delivers a touch event with the action code ACTION_DOWN, the X and Y coordinates of the touch and information about the pressure, size and orientation of the contact area
- The method **MotionEvent.getActionMasked()** returns the current action being performed (e.g., ACTION_DOWN, ACTION_UP, etc.)

Dragging an Object

- A common operation for a touch gesture is to use it to drag an object across the screen
- To implement a drag operation you need to track the following motion events:
  - **ACTION_DOWN** - A pressed gesture has started, the motion contains the initial starting location
  - **ACTION_MOVE** - A change has happened during a press gesture (between ACTION_DOWN and ACTION_UP). The motion contains the most recent point, as well as any intermediate points since the last down or move event.
  - **ACTION_UP** - A pressed gesture has finished, the motion contains the final release location as well as any intermediate points since the last down or move event.
  - **ACTION_CANCEL** - The current gesture has been aborted. You will not receive any more points in it. You should treat this as an up event, but not perform any action that you normally would.

Dragging an Object Example

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    int action = MotionEventCompat.getActionMasked(event);

    switch (action) {
        case MotionEvent.ACTION_DOWN: {
            int pointerIndex = MotionEventCompat.getActionIndex(event);
            float x = event.getX();
            float y = event.getY();
            Point pos = convertTouchPosToMazeCell(x, y);
            if (pos.equals(playerPos))
                isPlayerMoving = true;
            break;
        }
        case MotionEvent.ACTION_MOVE: {
            if (!isPlayerMoving)
                return true;
            float x = event.getX();
            float y = event.getY();
            Point pos = convertTouchPosToMazeCell(x, y);

            if (pos.x >= maze[0].length || pos.y >= maze.length) {
                isPlayerMoving = false;
            }
```

```
            // Check if this is a free cell
            else if (maze[pos.y][pos.x] == 0) {
                playerPos = pos;
                invalidate();
            } else {
                isPlayerMoving = false;
            }
            break;
        }
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            isPlayerMoving = false;
            break;
    }
    return true;
}
```