
# Session 2 – Multi screen apps

הכנה פרקטית להי-טק

# Session overview

- **Activity**
- **Menus**
- **Intents**
- **Context**
- **Android debugging**

# Activity

## Activity basics –

- From Google documentation: “Activity is a single, focused thing that the user can do”
- Supports interaction with the user
- Supports creating a window with UI:

```
setContentView(View);
```

- Not necessarily full screen

# Activity

## Activity basics –

- Each app has at least one Activity
- It is the entry point of the app:

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

```
public void onCreate(Bundle savedInstanceState) { }
```

# Activity basics –

- Each activity must be declared in the ‘AndroidManifest.xml’ file –

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="il.co.practis.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        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>

        <activity android:name=".Secondary"></activity>
    </application>

</manifest>
```

**Only in the main activity**

**Additional Activity**

# Activity

## Activity basics –

- Each activity can open another one –

```
Intent intent = new Intent(this, SecondaryActivity.class);
startActivity(intent);
```

- Each activity can close itself –

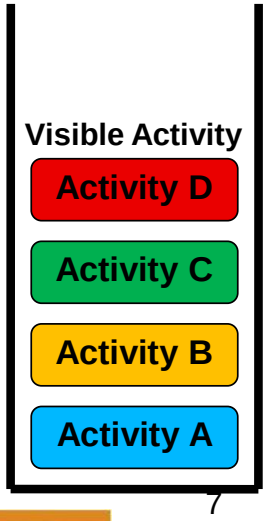
```
finish();
```

# Activity

## Activity basics –

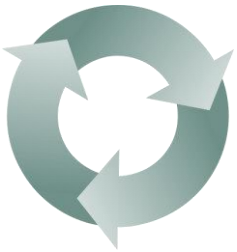
- Activities exist in a stack –

```
public class ActivityD extends Activity {
...

Intent intent = new Intent(this, ActivityE.class);
startActivity(intent);

...
}
```



# Activity

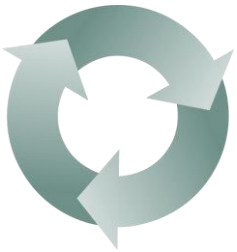
## LifeCycle



- Once created – visible
- Activity might get interrupted (phone call)
- A new activity might become top on stack
- Once not in focus – paused
- When not visible – stopped
- When stopped might get killed for memory

# Activity

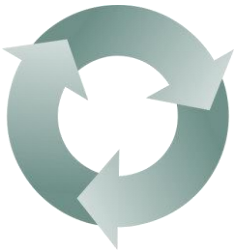
## LifeCycle



- When killed information **might** get lost
- Solution:
  - We are notified once app is paused
  - We have a chance to backup critical information
  - We are notified once app is resumed
  - We can load backed up information
- **Some information is automatically restored**

# Activity

## LifeCycle



- Several techniques to save state data
- OnSaveInstanceState / onRestoreInstanceState
- **SharedPreferences –**
  - Great for basic variable types
  - Persistent even if app was closed
  - Save info in onPause()
  - Load it back in onCreate()

# Activity

## LifeCycle - SharedPreferences - example

```
@Override
protected void onPause()  {
    super.onPause();

    // Create object to store values privately
    SharedPreferences preferences  = getPreferences(MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();

    // Get values from the UI
    EditText txtName = (EditText)findViewById(R.id.txtFirstName);
    String strName = txtName.getText().toString();

    // Store values
    editor.putString("firstName", strName);

    // Commit to storage
    editor.commit();
}
```

# Activity

## LifeCycle - SharedPreferences - example

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


     // Get required storage object
    SharedPreferences preferences = getPreferences(MODE_PRIVATE);

    // Set the values of the UI
    EditText oControl = (EditText)findViewById(R.id.txtFirstName);
    oControl.setText(preferences.getString("firstName", null));
}
```

# Activity

## LifeCycle - SharedPreferences - example

- SharedPreferences is Activity **private**:

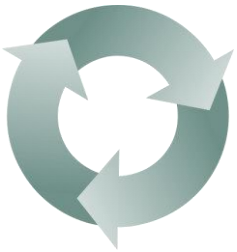
```
SharedPreferences preferences = getPreferences(MODE_PRIVATE);
```

- SharedPreferences is Activity **shared**:
  - Both activities must use same name

```
SharedPreferences pref = getSharedPreferences("someName",
                                              MODE_PRIVATE);
```

# Activity

## LifeCycle



- SharedPreferences supported types:
  - putBoolean(String key, boolean value)
  - putFloat(String key, float value)
  - putInt(String key, int value)
  - putLong(String key, long value)
  - putString(String key, String value)
  - putStringSet(String key, Set<String> values)

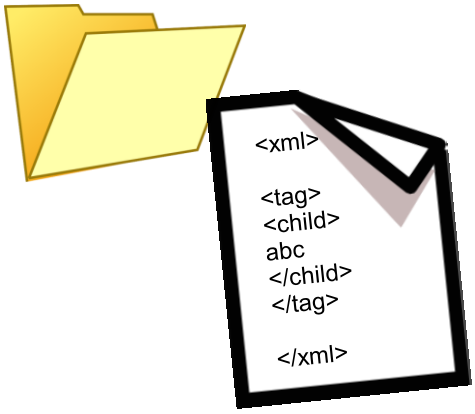
# Menu

## Basics

- List of options
- Displayed when clicking menu button
- Several menu items types exist –
  - Clickable
  - Toggle
  - Checkable

# Menu

## Creating a menu




- Create folder res/menu
- Create menu XML file
- Inflate menu in relevant Activity
- Add handlers for menu items click

# Menu

## Usage – menu XML file –

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android”>
    <item
        android:id="@+id/new_game"
        android:title="@string/New_game"/>
    <item
        android:id="@+id/load_file"
        android:title="@string/Load_File"/>
    <item
        android:id="@+id/quit"
        android:title="@string/Quit"/>
</menu>
```

# Menu

## Usage – Inflating menu –

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.some_menu, menu);

   return true;
}
```

# Menu

## Usage – Handling events –

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_game:
            startNewGame();
            return true;
        case R.id.about:
            showAboutScreen();
            return true;
        case R.id.help:
            showHelpScreen();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

# Menu

## Usage – Advanced menu –

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

    <item
        android:id="@+id/regular_item"
        android:title="@string/regular_item"/>
    <item
        android:id="@+id/pictureItem"
        android:icon="@drawable/ic_launcher"
        android:showAsAction="ifRoom"
        android:title="@string/pictureItem"/>
    <item
        android:id="@+id/checkableitem"
        android:checkable="true"
        android:title="@string/checkableItem"/>
    <item
        android:id="@+id/file"
        android:title="@string/file">

        <!-- "file" submenu -->
        <menu>
            <item
                android:id="@+id/create_new"
                android:title="@string/create_new"/>
            <item
                android:id="@+id/open"
                android:title="@string/open"/>
        </menu>
    </item>

</menu>
```

# Intent

## Intent basics -

- Description of an action to perform
- Mainly used to start Activities:

```
Intent intent = new Intent(this, SecondaryActivity.class);
startActivity(intent);
```

- Also used to send data to secondary activity

# Intent

## Intent basics –

- Send data to secondary Activity Example:
  - Activity1:

```
Intent intent = new Intent(this, SecondaryActivity.class);
intent.putExtra("new_variable_name", "value");
startActivity(intent);
```

  - Activity2:

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
   String value = extras.getString("new_variable_name");
}
```

# Intent

## Intent basics -

- Can also start an activity for result
- Required Steps:
  - Activity1: Run Activity2 for result
  - Activity2: Send result back to Activity1
  - Activity1: Parse result

# Intent

## Intent basics -

- #1 - Activity1: Run Activity2 for result

```
Intent intent = new Intent(this, Activity2.class);
startActivityForResult(intent, 123);
```

`int requestCode` – Used to identify the origin of each received result

# Intent

## Intent basics -

- #2 - Activity2: Send result back to Activity1

```
Intent returnIntent = new Intent();

// Set result additional parameters
returnIntent.putExtra("result", "blah blah");
returnIntent.putExtra("result-int", 12);
returnIntent.putExtra("result-float", 789.654f);
returnIntent.putExtra("result-boolean", false);

// Set main result value
setResult(RESULT_OK, returnIntent);

// Close Activity2
finish();
```

# Intent

## Intent basics -

- #3 – Activity1: Parse Result

```
protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent data) {
    if (requestCode == 123) {

        if(resultCode == RESULT_OK){
            String result = data.getStringExtra("result");
        }

        if (resultCode == RESULT_CANCELED) {
            // Handle case where Activity2 cancelled
        }
    }
}
```

# Intent

## Intent basics -

- Activity result types (int):
  - -1 RESULT_OK
  - 0  RESULT_CANCELLED
  - 1> User defined result type

```
static final int CUSTOM_RESULT_TYPE = 7;
setResult(CUSTOM_RESULT_TYPE , returnIntent);
```

# Intent

## Intent Advanced -

- So far used to start specific Activity
- Can also start a needed action and not specific activity:
  - Dial a phone number
  - Send SMS with specific content
  - Show contact information
  - Start the Camera

# Intent

## Intent Advanced -

- This usage is based on Action + Data:
- Action – what should be done (>100 exist)
  - ACTION_VIEW
  - ACTION_EDIT
  - ACTION_DIAL
- Data – on what to perform this action
  - Specific contact
  - Provided telephone number

# Intent

## Intent Advanced -

- Android finds best suiting Activity for Action
- Named: Intent resolution
- Based on <intent-filter> descriptions
- Defined in AndroidManifest.xml

# Intent

## Dial phone number Intent –

```
// Get phone number to dial from relevant EditText
EditText phoneNumField = (EditText)findViewById(R.id.txtPhoneNumber);
String phoneNumber = phoneNumField.getText().toString().trim();


// Prepare Intent to open phone dialer
Intent dialIntent = new Intent(Intent.ACTION_DIAL);
dialIntent.setData(Uri.parse("tel:" + phoneNumber));


// Start Intent
startActivity(dialIntent);
```

# Intent

## Uri - Uniform resource identifier

- String of characters
- Used to identify a specific resource
- URL and URN are sub types of URI
- Examples:
  - urn:ISBN:1771-18345-512-1377
  - ftp://domain.com/someFile.txt

# Intent

## Uri – Android Examples:

- file:*///sdcard/file.bin*
- content://contacts/people/1
- http://www.example.com
- tel:123456
- sms:123456
- mms:123456

# Intent

## SMS Intent -

```
// Get phone number to SMS from relevant EditText
EditText phoneNumField = (EditText)findViewById(R.id.txtPhoneNumber);
String phoneNumber = phoneNumField.getText().toString().trim();


// Prepare Intent to open SMS viewer/sender
Uri uri = Uri.parse("smsto:" + phoneNumber);
Intent smsIntent = new Intent(Intent.ACTION_SENDTO, uri);
smsIntent.putExtra("sms_body", "SMS text content here");


// Start Intent
startActivity(smsIntent);
```

# Intent

## View Contact Intent –

```
// Get contact id to display from relevant EditText
EditText contactField = (EditText)findViewById(R.id.txtContactID);
String contactID = contactField.getText().toString().trim();


// Prepare Intent to open Contact viewer
Intent intent = new Intent(Intent.ACTION_VIEW);
Uri uri = Uri.withAppendedPath(
                    ContactsContract.Contacts.CONTENT_URI,
                    contactID);
intent.setData(uri);


// Start Intent
startActivity(intent);
```

# Intent

## Open contact picker for result –

1. Open the picker -

```
public void performClick(View view) {

   // Open the contact picker (after button click)
   Intent intent = new Intent(Intent.ACTION_PICK,
                   ContactsContract.Contacts.CONTENT_URI);

   // Start Intent
   startActivityForResult(intent, 123);
}
```

# Intent

## Open contact picker for result –

2. Parse picker result –

```
public void onActivityResult(int reqCode, int resultCode, Intent data) {
super.onActivityResult(reqCode, resultCode, data);

switch (reqCode) {
    case (123) :
    if (resultCode == Activity.RESULT_OK) {
        Uri contactData = data.getData();
        Cursor cursor = getContentResolver().query(contactData, null,
                                                    null, null, null);
        if (cursor.moveToFirst()) {
            String name = cursor.getString(cursor.getColumnIndex(
                           ContactsContract.Contacts.DISPLAY_NAME));

            EditText nameField =
                                (EditText)findViewById(R.id.txtContactName);
            nameField.setText(name);
} } } }
```

# Intent

## Open webpage Intent -

```
public void performClick(View view) {

    // Prepare Intent to open URL
    String url = "http://www.practis.co.il";
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse(url));


    // Start Intent
    startActivity(intent);
}
```

# Intent

## Camera Intent -

1. Open the Camera -

```
public void performClick(View view) {

// Prepare Camera Intent
Intent cameraIntent = new
                    Intent(MediaStore.ACTION_IMAGE_CAPTURE);


// Start Intent
startActivityForResult(cameraIntent , 123);

}
```

# Intent

## Camera Intent -

2. Obtain picture taken –

```
protected void onActivityResult(int requestCode, int
                                resultCode, Intent data) {
// Parse result
if (requestCode == 123 && resultCode == RESULT_OK) {
   Bundle extras = data.getExtras();
   Bitmap imageBitmap = (Bitmap) extras.get("data");

   // Set ImageView to contain picture taken
   ImageView mImageView =
                  (ImageView)findViewById(R.id.mImageView);
   mImageView.setImageBitmap(imageBitmap);
   }
}
```

# Context

## What is this Context?

- An object (Base class for Activity)
- Represents environment data
- Provides access to various resources
- Simple explanation: Hotel example

# Context

## What is it used for?

- Loading a resource
- Starting a new Activity
- Dynamically adding View
- Obtaining system services

# Android debugging

## Logging -

- We can but don’t use the console
- Use **logcat** instead:
  - Tool to view and filter information from the Android logging system
  - Android Studio has built-in dedicated window

# Android debugging

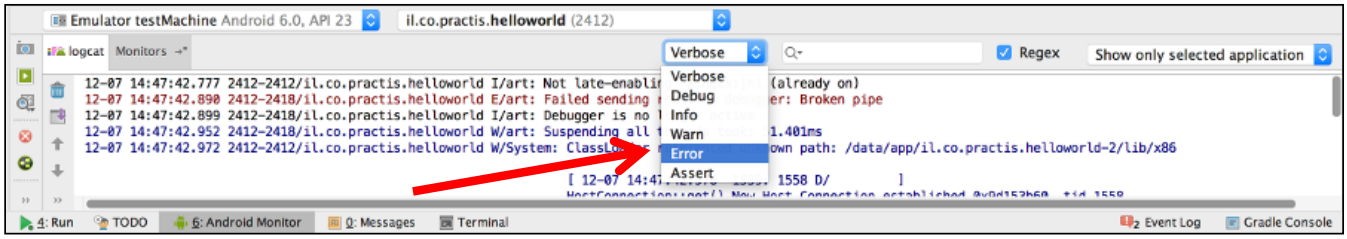
## Logging –

- This window opens automatically
- Just need to run the app
- Will open at the bottom of the screen

# Android debugging

## Logging –

- Can set severity of events to watch:



# Android debugging

## Logging – Usage

- We can add messages to the log
- There are several Severity levels:
  - Verbose
  - Debug
  - Information
  - Warning
  - Error

# Android debugging

## Logging – Usage

```
Log.v("MyTag", "This is an example of Verbose message");

Log.d("MyTag", "This is an example of Debug message");

Log.i("MyTag", "This is an example of Information message");

Log.w("MyTag", "This is an example of Warning message");

Log.e("MyTag", "This is an example of Error message");
```

# Android debugging

## Regular Debugging -

- Slower than regular execution
- Uses different VM settings
- **Not exactly the same as a regular run**
- Might encounter different behaviors!


**For Questions:**
**yoav@practis.co.il**

Practis

הכנה פרקטית להי-טק