Intents

התחלת Activity

Activity מייצגת מסך בודד באפליקציה
ניתן להתחיל מופע חדש של Activity על ידי העברת אובייקט Intent לפעולה startActivity
הIntent מתאר את הActivity ונושא את כל הנתונים הדרושים.

אם ברצוננו לקבל תוצאה מהActivity כשהוא מסתיים, ניתן להשתמש בפעולה registerActivityForResult.
הActivity מקבל את התוצאה כאובייקט Intent בפעולה שמועברת ל registerActivityForResult.

סוגי Intents
ישנם שני סוגים של Intents:

Explicit Intents
מציינות איזו אפליקציה תספק את הכוונה, על ידי אספקת שם החבילה של אפליקציית היעד או שם מחלקה.
בדרך נשתמש בExplicit Intent כדי להפעיל רכיב באפליקציה, מכיוון שיודעים את שם המחלקה של Activity או השירות שרוצים להתחיל.
לדוגמה, נוכל להתחיל Activity חדש בתוך האפליקציה שלך בתגובה לפעולת משתמש, או להתחיל שירות להורדת קובץ ברקע.

Implicit Intents
אינן נותנות שם לרכיב מסוים, אלא מכריזות על פעולה כללית לביצוע, המאפשרת לרכיב מאפליקציה **אחרת** לטפל בה.
לדוגמה, אם רוצים לשלוח מידע מהאפליקציה דרך אפליקציה אחרת,
ניתן להשתמש Implicit Intent כדי לבקש שאפליקציה אחרת בעלת יכולת כזו תקבל את המידע להמשך שליחתו.
לדוגמא ל Whatsapp

Explicit Intent:

```
// Intent without a result expected
val intent = Intent( packageContext: this,SecondActivity::class.java)
startActivity(intent)
```

Explicit Intent with result:

```
// Intent with a result expected
val intent = Intent( packageContext: this,SecondActivity::class.java)
val intentLauncher  = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()) {  it: ActivityResult!
    // here we receive result from the intent
}
// launch the intent with the result launcher
intentLauncher.launch(intent)
```

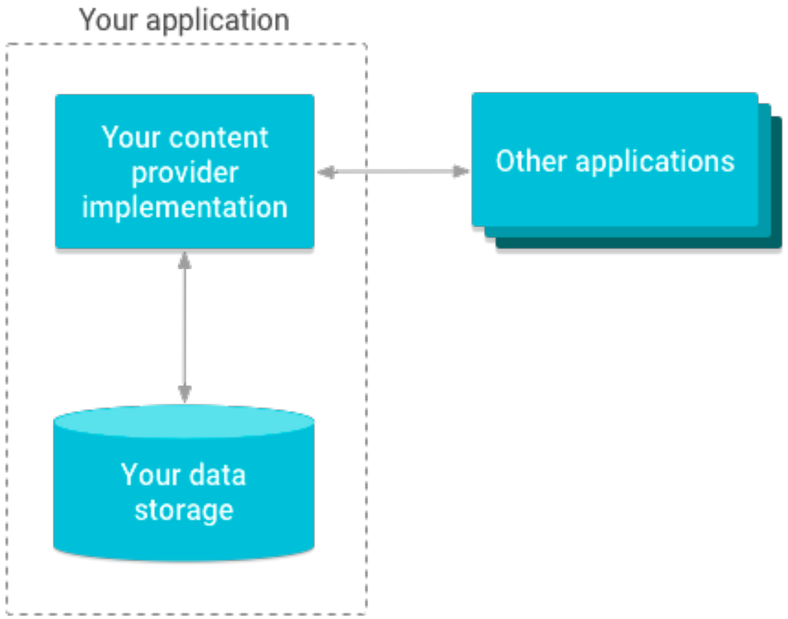
Implicit Intent for sending data through an other app

```
// implicit intent to send data to other app
val sendIntent: Intent = Intent().apply {  this: Intent
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT,  value: "This is my text to send.")
    type = "text/plain"
}

// lets the user choose an app to send the data to (ex: send SMS)
val shareIntent = Intent.createChooser(sendIntent,  title: null)
startActivity(shareIntent)
```

Content Provider

ספקי תוכן יכולים לעזור לאפליקציה לנהל גישה לנתונים המאוחסנים בתוכה,או באפליקציות אחרות,
ולספק דרך לשתף נתונים עם אפליקציות אחרות.
להטמעת Content Provider יש יתרונות רבים, והכי חשוב שניתן להגדיר Content Provider
כדי לאפשר ליישומים אחרים לגשת בצורה מאובטחת ולשנות את נתוני האפליקציה שלנו כפי שמוצג באיור:

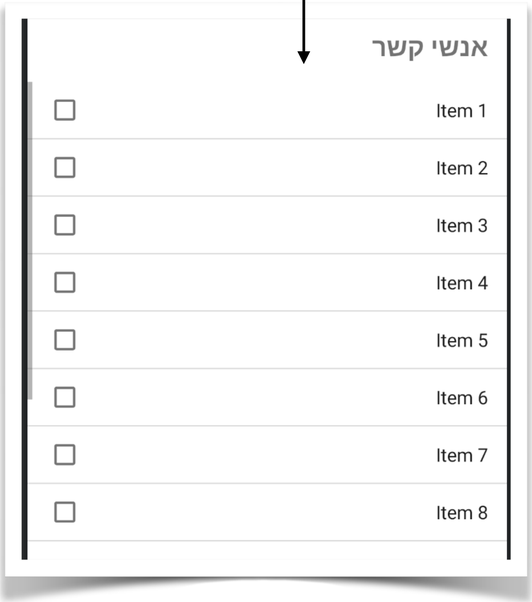


דוגמא למימוש
Content Provider

אנחנו יכולים ליצור Content Provider משלנו על מנת לאפשר לאפליקציה אחרת לגשת לנתונים אצלנו, נלמד על זה בהמשך. כרגע נראה איך ניתן לגשת לContent Provider שאפליקציות אחרות מספקות. בין היתר, אפליקציות של המערכת:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp"
    android:text="@string/contacts"
    android:textAlignment="textStart"
    android:textSize="22sp"
    android:textStyle="bold" />

<ListView
    android:id="@+id/ContactsLv"
    android:layout_width="match_parent"
    tools:listitem="@android:layout/simple_list_item_multiple_choice"
    android:layout_height="400dp"
    android:choiceMode="multipleChoice" />
```



ListView
הנועד להציג את שמות אנשי הקשר
ברשימה

```
private val listView: ListView by lazy { findViewById(R.id.ContactsLv) }

private val mReadContactsPermissionRequestCode = 0

private val mContactRequestData: Array<String> = arrayOf(
    ContactsContract.Contacts.DISPLAY_NAME_PRIMARY,
    ContactsContract.Contacts.CONTACT_STATUS,
    ContactsContract.Contacts.HAS_PHONE_NUMBER
)
```

ב Activity
נבצע בקשת גישה מהמשתמש
לאנשי הקשר, בעת קבלת גישות
נציג את אנשי הקשר ב ListView
ע"י שימוש ב Content Provider
המספק מידע על אנשי הקשר במכשיר

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

<application
    android:allowBackup="true"
```

הכרזה על כך שהאפליקציה
משתמשת בגישות לאנשי קשר

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_fourth)

    // Request READ_CONTACTS permission from user
    ActivityCompat.requestPermissions(
        activity: this,
        arrayOf(Manifest.permission.READ_CONTACTS),
        mReadContactsPermissionRequestCode
    )

}
```

דוגמא למימוש - המשך Content Provider

ב Activity נבצע בקשת גישה מהמשתמש לאנשי הקשר, ובעת קבלת גישות נציג את אנשי הקשר ב ListView
ע"י שימוש ב Content Provider המספק מידע על אנשי הקשר במכשיר.

1

```
private val listView: ListView by lazy { findViewById(R.id.ContactsLv) }

private val mReadContactsPermissionRequestCode = 0

private val mContactRequestData: Array<String> = arrayOf(
    ContactsContract.Contacts.DISPLAY_NAME_PRIMARY,
    ContactsContract.Contacts.CONTACT_STATUS,
    ContactsContract.Contacts.HAS_PHONE_NUMBER
)
```

קוד בקשת גישות - החלטנו לסמן ב 0

מערך של נתונים שרוצים לבקש מהContent Provider

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

<application
```

הכרזה על כך שהאפליקציה משתמשת בגישות לאנשי קשר

2

3

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_fourth)

    // Request READ_CONTACTS permission from user
    ActivityCompat.requestPermissions(
        activity: this,
        arrayOf(Manifest.permission.READ_CONTACTS),
        mReadContactsPermissionRequestCode
    )

}
```

בקשת גישות לקריאת אנשי הקשר בonCreate של Activity

4

```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    // if the requested permission is READ_CONTACTS
    if (requestCode == mReadContactsPermissionRequestCode) {
        // if the permission was granted -> query contacts from Content Provider
        if (grantResults[0] == PERMISSION_GRANTED) {
            // access Android Contacts Content Provider
            val mCursor = contentResolver.query(
                ContactsContract.Contacts.CONTENT_URI,
                mContactRequestData,
                selection: null,
                selectionArgs: null,
                sortOrder: null
            )
            val list = mutableListOf<String>()
            // read contacts from the cursor object
            if (mCursor != null && mCursor.count > 0) {
                while (mCursor.moveToNext()) {
                    list.add(mCursor.getString(0))
                }
            }
            // apply a simple array adapter with the contact list
            listView.adapter =
                ArrayAdapter( context: this, android.R.layout.simple_list_item_multiple_choice, list)
        }
    }
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
}
```

מימוש פעולה, הנקראת בעת קבלת גישות, בודקים את קוד הבקשה, במידה וזה בקשת גישות לאנשי הקשר,
מוציאים מידע מContent Provider של אנשי הקשר
במערכת על ידי שימוש באובייקט Cursor

5

אנשי קשר

- re0107@netvision net. il
- רמה מוטי
- chuchi180@walla co. il
- בליין 558
- בליין 673
- אופק פטל הגבר
- שלום מורה נהיגה
- בליין 587

תוצאה:
שמות אנשי הקשר מהמכשיר ברשימה

BroadcastReceiver

Broadcasts באנדרואיד הם האירועים במערכת שיכולים להתרחש כאשר המכשיר מופעל,
כאשר מתקבלת הודעה במכשיר או כאשר מתקבלות שיחות נכנסות,
או כאשר המכשיר עובר למצב טיסה וכו'.

BroadcastReceiver משמשים כדי להגיב אירועים כלל-מערכתיים אלה
ומאפשרים לנו להירשם לאירועי המערכת והאפליקציה.

כאשר אירוע זה מתרחש, ה BroadcastReceivers מקבלים הודעה. ישנם שני סוגים עיקריים:

BroadcastReceiver סטטיים: סוגים אלה של BroadcastReceiver מוצהרים בקובץ המניפסט ופועלים גם אם האפליקציה סגורה

BroadcastReceiver דינמיים: פועלים רק אם האפליקציה פעילה או ממוזערת

BroadcastReceiver

מאז API 26, ניתן לטפל ברוב השידורים רק על ידי BroadcastReceiver דינמי, לכן בדוגמאות נשתמש רק בסוג זה.

ישנם כמה שדות סטטיים המוגדרים במחלקה Intent אשר ניתן להשתמש בהם כדי לשדר אירועים שונים.
לקחנו את שינוי מצב טיסה כאירוע שידור, אך ישנם אירועים רבים שעבורם ניתן להשתמש בBroadcastReceiver.

להלן כמה מה Intents החשובים שנוצרים במערכת:

Intent	תיאור האירוע
android.intent.action.BATTERY_LOW :	משודר כאשר סוללה נמוכה
android.intent.action.BOOT_COMPLETED	משודר כאשר המכשיר נדלק
android.intent.action.DATE_CHANGED	משודר כאשר התאריך משתנה
android.intent.action.REBOOT	משודר אחרי ריסטרט למכשיר
android.net.conn.CONNECTIVITY_CHANGE	משודר כאשר יש שינוי במצב האינטרנט במכשיר
android.intent.ACTION_AIRPLANE_MODE_CHANGED	משודר כאשר יש שינוי במצב טיסה במכשיר

BroadcastReceiver

שני הדברים העיקריים שעלינו לעשות כדי להשתמש בBroadcastReceiver:

1

יצירת Broadcast Receiver:

עלינו לכתוב מחלקה היורשת מ *BroadcastReceiver*

יצירת מחלקה היורשת מBroadcastReceiver
וממשת את הפעולה onReceive
אשר מופעלת ברגע שיש שידור

```
class AirplaneModeChangeReceiver: BroadcastReceiver() {

    override fun onReceive(context: Context?, intent: Intent?) {
        // logic of the code needs to be written here
    }

}
```

2

רישום של BroadcastReceiver:

עלינו לרשום את ה*BroadcastReceiver* שיצרנו לשידורים מסוימים.

בדוגמא הנל, לשידורי שינוי במצב טיסה של המכשיר.

בActivity
רישום הBroadcastReceiver שיצרנו לשידורי שינויי
במצב טיסה של המכשיר

```
// define the airplane mode change receiver
private val receiver: AirplaneModeChangeReceiver by lazy {
    AirplaneModeChangeReceiver()
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    // register the receiver when the activity is created
    registerBroadcastReceiver()
}

private fun registerBroadcastReceiver() {
    IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED).also {  it: IntentFilter

        // receiver is the broadcast receiver that we have registered

        // and it is the intent filter that we have created

        registerReceiver(receiver, it)
    }
}

// unregister the receiver when the activity is stopped
override fun onStop() {
    super.onStop()
    unregisterReceiver(receiver)
}
```

הגדרת
הBroadcastReceiver

רישום שלו בעת תחילת
הActivity

הפעולה אשר מבצעת
רישום של BroadcastReceiver
עבור שידורי שינוי במצב טיסה

מחיקת רישום בעת
הפסקת הActivity

BroadcastReceiver

בשקופית קודמת ראינו איך יוצרים BroadcastReceiver
וביצענו הרשמה שלו עבור שידורי שינוי במצב טיסה.

כעת נטפל בשידורים אשר מתקבלים, בפעולה onReceive
שהשארנו ריקה:

```
class AirplaneModeChangeReceiver : BroadcastReceiver() {

    // this function will be executed when the user changes his
    // airplane mode
    override fun onReceive(context: Context?, intent: Intent?) {

        // intent contains the information about the broadcast
        // in our case broadcast is change of airplane mode

        // if getBooleanExtra contains null value,it will directly return back
        val isAirplaneModeEnabled = intent?.getBooleanExtra( name: "state",  defaultValue: false) ?: return

        // checking whether airplane mode is enabled or not
        if (isAirplaneModeEnabled) {
            // showing the toast message if airplane mode is enabled
            Toast.makeText(context,  text: "Airplane Mode Enabled", Toast.LENGTH_LONG).show()
        } else {
            // showing the toast message if airplane mode is disabled
            Toast.makeText(context,  text: "Airplane Mode Disabled", Toast.LENGTH_LONG).show()
        }
    }
}
```

טיפול בשידורים אשר מתקבלים ב BroadcastReceiver:

המצב טיסה הנוכחי של המכשיר
נמצא כתכונה "state"
באובייקט Intent אשר מתקבל בפעולה onReceive
שנקראת כשיש שינויים במצב טיסה

אם מצב טיסה הנוכחי נמצא כפעיל,
נציג הודעה על המצב בהתאם

אם מצב טיסה הנוכחי נמצא כלא פעיל,
נציג הודעה על המצב בהתאם

Service

Service הוא רכיב יישומי שיכול לבצע פעולות ארוכות טווח ברקע אשר אינו מספק ממשק משתמש.
לאחר ההפעלה, Service עשוי להמשיך לפעול במשך זמן מה, גם לאחר שהמשתמש יעבור לאפליקציה אחרת.
בנוסף, רכיב מסוים באפליקציה (למשל Activity) יכול להיקשר לService כדי ליצור איתו אינטראקציה ואפילו לבצע תקשורת בין-תהליכית (IPC).
לדוגמה, Service יכול לטפל בעסקאות רשת, לנגן מוזיקה, לבצע קלט/פלט של קבצים או ליצור אינטראקציה עם Content Provider, הכל ברקע.

זהירות! Service פועל בMain Thread של תהליך האירוח שלו

Service אינו יוצר Thread משלו ואינו פועל בתהליך נפרד אלא אם כן מציינים אחרת.
לכן עלינו להפעיל את כל פעולות הכבדות על Thread נפרד בתוך הService כדי למנוע שגיאות של יישום לא מגיב.

נדבר על זה בהרחבה כשנעשה דוגמא

Service

סוגי Services
ישנם שלושה סוגי Service:

Forground Service
Forground Service מבצע פעולה כלשהי אשר **מורגשת ישירות** על ידי המשתמש.
לדוגמה, אפליקציית שמע תשתמש בForground Service כדי לנגן שירים ברקע.
Forground Services חייבים להציג הודעה למשתמש כל עוד הם פועלים.
Services מסוג זה ממשיכים לפעול גם כאשר המשתמש אינו מקיים אינטראקציה עם האפליקציה.

כאשר משתמשים בForground Service, עלינו להציג הודעה כדי שהמשתמשים יהיו מודעים באופן פעיל לכך שהService פועל.
לא ניתן לבטל הודעה זו אלא אם הService יופסק או יוסר מהליך ה Forground.

Background Service
Background Service מבצע פעולה **שאינה מורגשת ישירות** על ידי המשתמש.
לדוגמה, אם אפליקציה השתמשה בService כדי לבצע הורדה של קובץ גדול, זה בדרך כלל יהיה שירות רקע.

Bound Service
Bound Service נקשר כאשר רכיב יישום נקשר אליו על ידי קריאה לפעולה ()bindService.
שירות מסוג זה מציע ממשק לקוח-שרת המאפשר לרכיבים ליצור אינטראקציה עם הService,
לשלוח בקשות, לקבל תוצאות ואפילו לעשות זאת על פני תהליכים עם תקשורת בין תהליכים (IPC).
Bound Service פועל רק כל עוד רכיב יישום אחר קשור אליו.
מספר רכיבים יכולים להיקשר לשירות בבת אחת, אך כאשר כולם מתנתקים, השירות נהרס.

Service

כדי ליצור Service, עליך ליצור מחלקה אשר יורשת מהמחלקה
Service
או להשתמש באחת ממחלקות המשנה הקיימות שלו.
באפליקציה עצמה, עלינו לדרוס כמה פעולות המטפלות בהיבטים
מרכזיים של מחזור חיי הService
ומספקות מנגנון המאפשר לרכיבים להיקשר לService, במידת
הצורך.
אלו הן הפעולות החשובות ביותר שעלינו לדרוס:

onStartCommand()

המערכת מפעילה שיטה זו על ידי קריאה ל-startService() כאשר רכיב אחר (כגון Activity) מבקש להפעיל את הService.
כאשר פעולה זו מופעלת, הService מופעל ויכול לפעול ברקע ללא הגבלת זמן.
אם אנחנו מיישמים פעולה זו זאת, באחריותינו להפסיק את הService כאשר עבודתו תושלם על ידי קריאה ל-stopSelf() או stopService().

onBind()

המערכת מפעילה פעולה זו כאשר רכיב מסוים באפליקציה כגון Activity מבצע פעולת Bounding על השירות, לרוב לא נשתמש בפעולה זו,
אבל אנו מחויבים לממש אותה בעת ירושה מService ונחזיר null

onCreate()

המערכת מפעילה פעולה זו כדי לבצע הליכי הגדרה חד-פעמיים כאשר הService נוצר לראשונה (לפני שהוא קורא ל-onStartCommand().
אם הService כבר פועל, פעולה זו אינה נקראת.

onDestroy()

המערכת מפעילה פעולה זו כאשר הService אינו בשימוש עוד והוא מושמד.
הService צריך ליישם פעולה זו כדי לנקות משאבים וזיכרון שלא בשימוש. זוהי הפעולה האחרונה שהשירות מקבל.

Service

דוגמא לשימוש בBackground service

```
class RandomNumberService : Service() {

    // object that can execute actions on other threads with delay
    private lateinit var timer: Timer

    override fun onBind(intent: Intent): IBinder? {
        // we do not user binding in this example
        return null
    }

    private fun toast(toast: String) {
        // toasting logic goes here
        Toast.makeText(applicationContext, toast, Toast.LENGTH_LONG).show()
    }

    override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {
        // Send a notification that service is started
        toast( toast: "Service started.")
        // Do a periodic task
        timer = Timer()
        timer.schedule( delay: 500,  period: 5000) {  this: TimerTask
            showRandomNumber()
        }
        return START_STICKY
    }

    override fun onDestroy() {
        super.onDestroy()
        toast( toast: "Service destroyed.")
        timer.cancel()
    }

    // Custom method to do a task
    private fun showRandomNumber() {
        val rand = Random()
        val number = rand.nextInt( bound: 100)
        toast( toast: "Random Number : $number")
    }
```

הגדרת אובייקט
מסוג Timer
על מנת להפעיל פעולה בחזרות
כל כמה זמן

יישום ריק של onBind
לא משתמשים בדוגמא זו

פעולה המקפיצה Toast
על הContext של כל
האפליקציה

יישום של onStartCommand
מפעיל את הTimer
עם הפעולה showRandomNumber
כל 5 שניות

יישום של onDestory
מבטל את הTimer
כאשר הService מושמד

פעולה אשר מגרילה מספר
בין 0 ל100 ומציגה אותו בToast

```
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

<activity android:name=".SecondActivity"/>
<activity android:name=".ThirdActivity"/>
<service android:name=".RandomNumberService"/>
</application>
```

הכרזה על הService בקובץ הManifest
כמו שמכריזים על Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.third_activity)
    // declare an intent to start our service
    val intent  = Intent( packageContext: this,
        RandomNumberService::class.java)
    // starts the service
    startService(intent)
}
```

הגדרת Intent עבור ה Service שיצרנו
והפעלתו בonCreate של Activity כלשהו