

Android Intent

Gestione Intent e Intent filter

Lezione 7

Intent

- E' un componente specifico di Android?

App components

- I componenti sono i blocchi/mattoni fondamentali di un'app Android
- Ogni componente è un punto di accesso (entry point) attraverso il quale il sistema o un utente possono accedere all'app
- Ci sono 4 diversi tipi di componenti:
 - *Activities* (le interfacce della nostra app)
 - *Services* (permettono mantenere un'app in esecuzione in background)
 - *Broadcast receivers* (consente al sistema di inviare eventi all'app al di fuori di un normale flusso utente)
 - *Content providers* (gestisce un set condiviso di dati dell'app che è possibile archiviare – tipo esempio, accedere alle info sui contatti)



Intent

- E' un componente specifico di Android?
 - NO!
- Quindi che cos'è?

Intent

- Si usa per attivare 3 dei 4 componenti disponibili, ovvero «**activities, services, e broadcast receivers**», è un messaggio asincrono
 - *che* descrive un'azione da eseguire, inclusi i dati su cui agire, la categoria del componente che dovrebbe eseguire l'azione e altre istruzioni
- Gli intenti (*intent*) associano i singoli componenti tra loro in fase di esecuzione. Si possono considerare come i messaggeri che richiedono un'azione a componenti (indipendentemente dal fatto che il componente appartenga alla tua app o ad un'altra)
- Un Intent viene creato con un oggetto *Intent*, che definisce un messaggio per attivare un componente specifico (explicit intent) o un tipo specifico di componente (implicit intent)

Intent

- Per **activity** e **service**, un *intent* definisce **l'azione da eseguire** (ad esempio, per visualizzare – *view* - o inviare – *send* - qualcosa) e può specificare l'URI dei dati su cui agire, una tra le cose che un component avviato potrebbe avere necessità di conoscere
 - Ad esempio, un intent potrebbe trasmettere una richiesta per un'attività per mostrare un'immagine o per aprire una pagina Web
- In alcuni casi, è possibile avviare un'attività (activity) per ricevere un risultato, nel qual caso l'attività restituisce anche il risultato in un Intent
 - Ad esempio, è possibile emettere l'intenzione di consentire all'utente di scegliere un contatto personale e di restituirlo all'utente. L'intent di ritorno include un URI che punta al contatto scelto
- Per **broadcast receivers**, l'intent definisce semplicemente **l'annuncio da trasmettere**
 - Ad esempio, una trasmissione per indicare che la batteria del dispositivo è scarica include solo una stringa di azione nota che indica che la batteria è scarica

Intent e Content providers

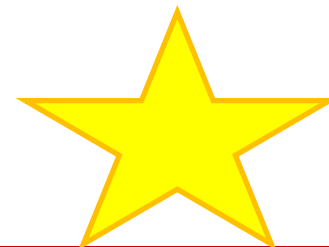
- A differenza delle altre tre componenti, i content provider non vengono attivati dagli intent. Vengono invece attivati quando vengono scelti come target da una richiesta di un *ContentResolver*
- Il *ContentResolver* gestisce tutte le transazioni dirette con il content provider in modo tale che il componente che sta eseguendo transazioni con il fornitore non debba farlo ma debba invece solo chiamare metodi sull'oggetto ContentResolver
 - Ovvero lascia uno **strato di astrazione** tra il fornitore di contenuti e il componente che richiede informazioni (per motivi di sicurezza)

Intent

- Quindi.. Un *intent* è un oggetto di messaggistica che è possibile utilizzare per richiedere un'azione da un altro component di un app
- Sebbene gli Intent facilitino la comunicazione tra i componenti in diversi modi, esistono tre casi d'uso fondamentali:
 1. Avvio di un'activity
 2. Avvio di un service
 3. Consegna broadcast

1. Avvio di un'activity

- Un'Activity rappresenta una singola schermata in un'app
- È possibile avviare una nuova istanza di un'Activity passando un Intent al metodo *startActivity()*
 - L'intent descrive l'attività da avviare e trasporta tutti i dati necessari
- Se si desidera ricevere un risultato dall'activity al termine, si deve utilizzare *startActivityForResult()*
 - La tua activity riceve il risultato come un oggetto *Intent* separato nel callback *onActivityResult()* della tua attività



Warning:
vedi ultime slide!

2. Avvio di un service (1/2)

- Un Service è un componente che esegue operazioni in background senza un'interfaccia utente
 - Vi ricordo che l'Intent service non è sempre la scelta ottimale, alternative:
 - Con androidx (Android Jetpack) è stato introdotto WorkManager (<https://developer.android.com/topic/libraries/architecture/workmanager>)
 - NEW! Con Kotlin, WorkManager fornisce il supporto primario per [coroutines](#)
 - Per maggiori dettagli:
<https://developer.android.com/guide/background/persistent/threading/coroutineworker>

APPROFONDIMENTO: WorkManager

- WorkManager è un'API che semplifica la pianificazione di attività asincrone differibili che dovrebbero essere eseguite anche se l'app viene chiusa o il dispositivo si riavvia
- L'API di WorkManager è un sostituto adatto e consigliato per tutte le precedenti API di pianificazione in background di Android (inclusi FirebaseJobDispatcher, GcmNetworkManager e Job Scheduler)
- WorkManager incorpora le funzionalità dei suoi predecessori in un'API moderna e coerente che funziona dal livello API 14, pur tenendo conto della durata della batteria

APPROFONDIMENTO: WorkManager

- WorkManager è un framework per gestire attività asincrone differite. L'attività viene chiusa quando l'app viene chiusa.
- L'API di WorkManager è parte delle API di Firebase. Vi ricordo che il Service viene eseguito nel thread principale del processo di hosting ed esegue anche quando l'utente non sta interagendo con la tua applicazione, quindi bisogna creare un Service solo se questo è ciò di cui hai bisogno.
- WorkManager è un'API moderna che funziona dal livello API 14, pur tenendo conto della durata della batteria.

Quindi, se per esempio dovete eseguire un pagamento, meglio usare Foreground Service perché l'attività non deve sopravvivere alla morte dell'app e deve essere eseguita immediatamente.

Vi ricordo che il Service viene eseguito nel thread principale del processo di hosting ed esegue anche quando l'utente non sta interagendo con la tua applicazione, quindi bisogna creare un Service solo se questo è ciò di cui hai bisogno.

Predecessori in

2. Avvio di un service (2/2)

- Un Service è un componente che esegue operazioni in background senza un'interfaccia utente
- È possibile avviare un Service per eseguire un'operazione una tantum (come il download di un file) passando un Intent a `startService()`
 - L'Intent descrive il servizio da avviare e trasporta tutti i dati necessari.
- Se il servizio è progettato con un'interfaccia client-server, è possibile creare un'associazione al servizio attraverso un altro componente passando un Intent a *`bindService()`*

3. Consegna broadcast

- Un broadcast è un messaggio che qualsiasi app può ricevere
- Il sistema fornisce varie trasmissioni (broadcast) per eventi di sistema, ad esempio quando il sistema si avvia o il dispositivo inizia a caricarsi
- Si può inviare un broadcast ad altre app passando un Intent a *sendBroadcast()* o *sendOrderedBroadcast()*

Tipi di Intent: explicit

- Ci sono due tipi di Intent (esplicito e implicito)
- **Explicit intents:** specifica quale applicazione soddisferà l'intent, fornendo il nome del pacchetto dell'app di destinazione o il nome completo della classe del componente
 - In genere si utilizza un intent esplicito per **avviare un componente nella propria app**, perché si conosce il nome della classe dell'attività o del servizio che si desidera avviare. Ad esempio, potresti avviare una nuova Activity all'interno della tua app in risposta ad un'azione dell'utente o avviare un Service per scaricare un file in background

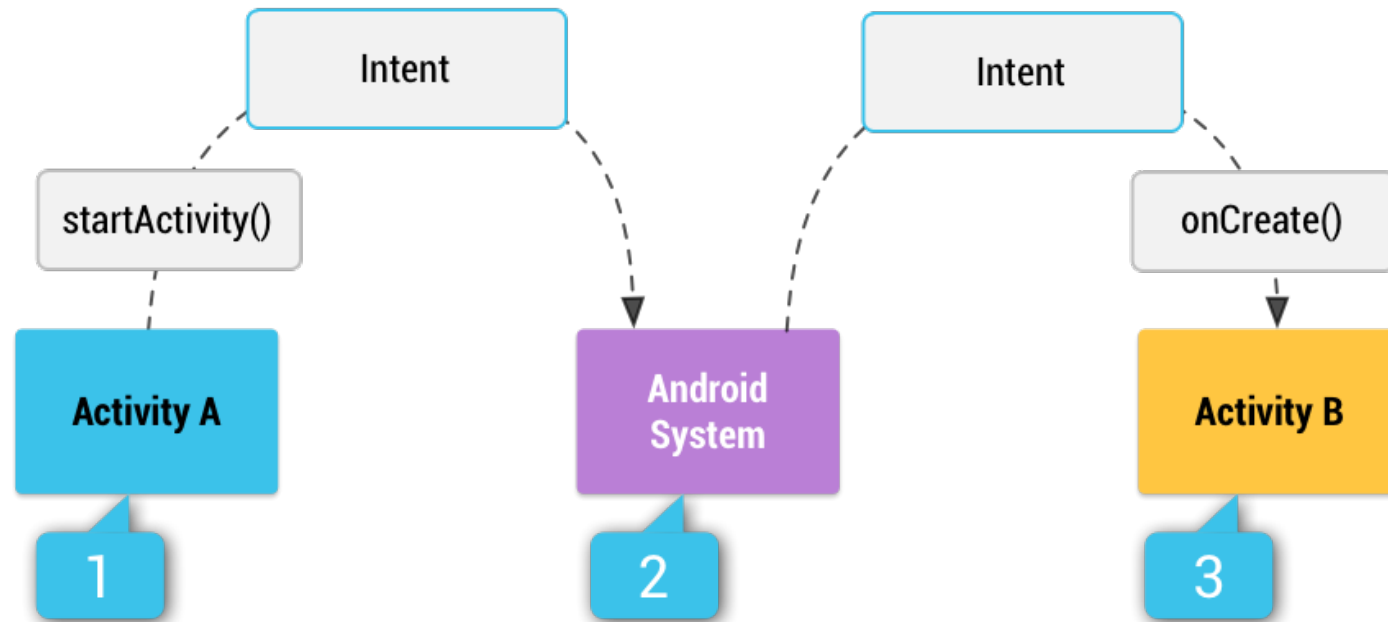
Intent explicit



Tipi di Intent: Implicit

- **Implicit intents:** non si nomina un componente specifico, ma si dichiara invece un'azione generale da eseguire, che consente a un componente di un'altra app di gestirlo
 - Ad esempio, se si desidera mostrare all'utente una posizione su una mappa, è possibile utilizzare un Intent implicito per richiedere che un'altra app mostri la posizione specifica su una mappa
 - Che app? dipende da quali sono installate nel device dell'utente

Implicit intent



- [1] L'activity A crea un Intent con una descrizione dell'azione e lo passa a `startActivity()`
- [2] Il sistema Android cerca in tutte le app un Intent filter che corrisponda all'intent. Quando viene trovata una corrispondenza, [3] il sistema avvia l'attività di corrispondenza (Activity B) invocando il metodo `onCreate()` e passandogli l'intent

Implicit Intent

- Quando si utilizza un Intent implicito (implicit), il sistema Android trova i componenti appropriati da avviare confrontando il contenuto dell'intent con i filtri di intent dichiarati nel file manifest di altre app sul dispositivo
- Se l'intent corrisponde a un intent filter, il sistema avvia quel componente e gli consegna l'oggetto Intent
- Se sono compatibili più filtri di intent che offrono quel componente, il sistema visualizza una finestra di dialogo in modo che l'utente possa scegliere quale app utilizzare
- Un Intent filter è un'espressione nel file manifest di un'app che specifica il tipo di intent che il componente potrebbe ricevere
 - Ad esempio, dichiarando un intent filter per un'activity, è possibile per altre app avviare direttamente l'activity con un determinato tipo di Intent
 - Allo stesso modo, se non si dichiarano filtri di intenti per un'activity, allora l'activity può essere avviata solo con un «Explicit intent»

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

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

            <!--SEND INTENT FILTER-->
            <intent-filter>
                <action android:name="android.intent.action.SEND"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <data android:mimeType="text/plain"/>
            </intent-filter>

            <!--VIEW INTENT FILTER-->
            <intent-filter>
                <action android:name="android.intent.action.VIEW"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <category android:name="android.intent.category.BROWSABLE"/>
                <data android:scheme="http"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Explicit intent e Services

- Attenzione
 - Per garantire la sicurezza dell'app, utilizzare sempre un Explicit Intent all'avvio di un Service e non dichiarare gli intent filter per i propri servizi
 - L'uso di un Implicit Intent per avviare un servizio è un rischio per la sicurezza perché non si può essere certi di quale servizio risponderà all'intent e l'utente non può vedere quale servizio viene avviato
 - A partire da Android 5.0 (livello API 21), il sistema genera un'eccezione se si chiama `bindService ()` con un Implicit Intent

Building un intent

- Un oggetto Intent contiene informazioni che il sistema Android utilizza per determinare quale componente avviare (come il nome esatto del componente o la categoria del componente che dovrebbe ricevere l'intent), oltre alle informazioni utilizzate dal componente destinatario per eseguire correttamente l'azione (come l'azione da intraprendere e i dati su cui agire)
- Le principali informazioni contenute in un Intent sono:
 - Component name
 - Action
 - Data
 - Category
 - Extras
 - Flags

Component name

- Il nome del componente da avviare
- È opzionale, ma è quella informazione critica che rende un intent un Explicit Intent, ciò significa che l'intent deve essere consegnato solo al componente dell'app definito dal nome del componente
- Senza il nome del componente, l'intent è Implicit e il sistema decide quale componente deve ricevere l'intent in base alle altre informazioni sull'intent (come l'azione, i dati e la categoria– a breve li vedremo in dettaglio)
- Se devi avviare un componente specifico nella tua app, devi specificare il nome del componente
 - Con i Services, specificare sempre il nome del componente! Altrimenti, non puoi essere sicuro di quale servizio risponderà all'intent e l'utente non può vedere quale servizio viene avviato
- Si usa l'oggetto *ComponentName*, specificando il nome di classe completo del componente di destinazione, incluso il nome del pacchetto dell'app, ad esempio *com.example.ExampleActivity*. È possibile impostare il nome del componente con *setComponent ()*, *setClass ()*, *setClassName ()* o con il costruttore *Intent*

Action

- Una stringa che definisce l'azione generica da eseguire (come *view* o *pick*)
- L'action determina in gran parte la struttura del resto dell'intent, in particolare le informazioni contenute nei dati e negli extra
- È possibile specificare le proprie action per l'uso da parte degli intent nella propria app (o per l'utilizzo da parte di altre app per invocare componenti nella propria app), ma di solito si specificano le costanti di azione definite dalla classe Intent o da altre classi del framework
- È possibile specificare l'action per un intent con *setAction()* o attraverso il costruttore di Intent

Action

- Esempi comuni per iniziare un'attività:
 - ACTION_VIEW -> utilizza questa azione con un intent con startActivity() quando si hanno informazioni (dati) che un'activity può mostrare all'utente
 - es: una foto da visualizzare in un'app della galleria o un indirizzo da visualizzare in un'app della mappa
 - ACTION_SEND -> conosciuto anche come share intent, è necessario utilizzarlo come Intent con startActivity() quando si dispone di alcuni dati che l'utente può condividere tramite un'altra app, come un'app di posta elettronica o un'app di condivisione social
 - Per altri esempi, guardare qui:
<https://developer.android.com/reference/android/content/Intent>

Action

- Se definisci le tue action, assicurati di includere il nome del pacchetto dell'app come prefisso, ovvero:

```
const val ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL"
```

Data

- L'URI (oggetto *Uri*) che fa riferimento ai dati su cui agire e/o al tipo MIME di tali dati
- Il tipo di dati forniti è generalmente dettato dall'azione dell'intent
 - Ad esempio, se l'azione è ACTION_EDIT, i dati devono contenere l'URI del documento da modificare
- Quando si crea un intent, è spesso importante specificare il tipo di dati (il suo tipo MIME) oltre al suo URI
 - Ad esempio, un'activity in grado di visualizzare immagini probabilmente non sarà in grado di riprodurre un file audio, anche se i formati URI potrebbero essere simili
 - Specificare il tipo MIME dei tuoi dati aiuta il sistema Android a trovare il componente migliore per ricevere il tuo Intent. Tuttavia, il tipo MIME può talvolta essere dedotto dall'URI, in particolare quando i dati sono un «*content:*» URI. Un *content:* URI indica che i dati si trovano sul dispositivo e controllati da un ContentProvider, che rende il tipo MIME di dati visibile al sistema
- Per impostare solo l'URI dei dati, chiamare *setData()*. Per impostare solo il tipo MIME, chiama *setType()*. Se necessario, è possibile impostare entrambi in modo esplicito con *setDataAndType()* – NB: farlo solo attraverso questo metodo!

Category

- Una stringa contenente informazioni aggiuntive sul tipo di componente che dovrebbe gestire l'intent
- Qualsiasi numero di descrizioni di categoria può essere inserito in un intent, ma la maggior parte degli intent non richiede una categoria
- Alcune categorie comuni:
 - CATEGORY_BROWSABLE
 - L'attività di destinazione consente di essere avviata da un browser Web per visualizzare i dati a cui fa riferimento un collegamento, ad esempio un'immagine o un messaggio di posta elettronica
 - CATEGORY_LAUNCHER
 - L'activity è l'activity iniziale di un task, ed è elencata nel programma di avvio dell'applicazione del sistema
- È possibile specificare una categoria con `addCategory ()`

Intent proprietà

- Le 4 proprietà appena descritte (component name, action, data e category) rappresentano le caratteristiche che definiscono un intent
- Leggendo queste proprietà, il sistema Android è in grado di risolvere quale componente dell'app dovrebbe avviare
- Tuttavia, un intent può contenere informazioni aggiuntive che non influiscono sul modo in cui viene risolto in un componente dell'app, ovvero:
 - **Extras**
 - **Flags**

Extras

- Coppie chiave-valore che contengono informazioni aggiuntive necessarie per eseguire l'azione richiesta
- Si possono aggiungere ulteriori dati con vari metodi *putExtra()*, ognuno dei quali accetta due parametri: il nome della chiave e il valore
- Ad esempio, quando si crea un intent per inviare un'e-mail con ACTION_SEND, è possibile specificare il destinatario con la chiave EXTRA_EMAIL e specificare l'oggetto con la chiave EXTRA_SUBJECT
- La classe Intent specifica molte costanti EXTRA_* per tipi di dati standardizzati
 - Se devi dichiarare le tue chiavi extra (per Intent che l'app riceve), assicurati di includere il nome del pacchetto dell'app come prefisso, come mostrato nell'esempio seguente:

```
const val EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS"
```

Flags

- I flag sono definiti nella classe Intent e funzionano come metadati per l'intent
- I flag possono indicare al sistema Android come avviare un'attività (ad esempio, a quale task deve appartenere l'attività) e come trattarla dopo il lancio (ad esempio, se appartiene all'elenco delle attività recenti)

Per ulteriori informazioni, consultare il metodo `setFlags()` -

[https://developer.android.com/reference/android/content/Intent#setFlags\(int\)](https://developer.android.com/reference/android/content/Intent#setFlags(int))

Esempio di Explicit intent

- Un intent explicit è quello che si usa per avviare un **componente specifico della propria app**, come una particolare attività o servizio nella tua app
- Per creare un intento esplicito, definire il nome del componente per l'oggetto Intent — tutte le altre proprietà di intent sono facoltative
- Il costruttore *Intent(Context, Class)* fornisce all'app (Context) e al componente un oggetto Class

Esempio di Explicit intent

- Ad esempio, se hai creato un Service nella tua app, denominato DownloadService, progettato per scaricare un file dal Web, puoi avviarlo con il codice seguente:

```
// Executed in an Activity, so 'this' is the Context  
// The fileUrl is a string URL, such as "http://www.example.com/image.png"  
val downloadIntent = Intent(this, DownloadService::class.java).apply {  
    data = Uri.parse(fileUrl)  
}  
startService(downloadIntent)
```

- Questo Intent avvia esplicitamente la classe DownloadService nell'app

Esempio di un Implicit Intent

- Un intent implicito specifica un'azione che può invocare qualsiasi app sul dispositivo in grado di eseguire l'azione
- L'uso di un intent implicito è utile quando l'app non è in grado di eseguire l'azione, ma altre app probabilmente possono e desideri che l'utente scelga quale app utilizzare
 - Ad esempio, se si dispone di contenuti che l'utente desidera condividere con altre persone, crea un intent con l'azione ACTION_SEND e aggiungi extra che specificano il contenuto da condividere. Quando chiami startActivity() con quell'intent, l'utente può scegliere un'app attraverso la quale condividere il contenuto

Implicit Intent

```
// Create the text message with a string.
```

```
val sendIntent = Intent().apply {  
    action = Intent.ACTION_SEND  
    putExtra(Intent.EXTRA_TEXT, textMessage)  
    type = "text/plain"  
}
```

```
// Try to invoke the intent.
```

```
try {  
    startActivity(sendIntent)  
} catch (e: ActivityNotFoundException) {  
    // Define what your app should do if no activity can handle the intent.  
}
```

resolveActivity()

- Nota: È possibile che un utente non disponga di app in grado di gestire l'intent implicito avviato con `startActivity()`
 - un'app potrebbe essere inaccessibile anche a causa delle restrizioni del profilo o delle impostazioni messe in atto da un amministratore.
- In tal caso, la chiamata ha esito negativo e l'app si arresta in modo anomalo.
- Per verificare che un'attività riceverà l'intent, usa `resolveActivity()` sull'oggetto Intent.
 - Se il risultato è non nullo, esiste almeno un'app in grado di gestire l'intent ed è sicuro chiamare `startActivity()`.
 - Se il risultato è nullo, non utilizzare l'intent e, se possibile, disabilitare la funzione che emette l'intent

Esempio: resolveActivity()

- Esempio: verificare che l'intent si risolva in un'attività. Questo esempio non utilizza un URI, ma viene dichiarato il tipo di dati dell'intent per specificare il contenuto trasportato dagli extra

```
val sendIntent = Intent(Intent.ACTION_SEND)
```

```
...
```

```
// Always use string resources for UI text.
```

```
// This says something like "Share this photo with"
```

```
val title: String = resources.getString(R.string.chooser_title)
```

```
// Create intent to show the chooser dialog
```

```
val chooser: Intent = Intent.createChooser(sendIntent, title)
```

```
// Verify the original intent will resolve to at least one activity
```

```
if (sendIntent.resolveActivity(packageManager) != null) {
```

```
    startActivity(chooser)
```

```
}
```

startActivity() - funzionamento

- Quando viene chiamato startActivity(), il sistema esamina tutte le app installate per determinare quali sono in grado di gestire questo tipo di intent (un intent con l'azione ACTION_SEND e che trasporta i dati "text / plain")
- Se esiste solo un'app in grado di gestirla, quell'app si apre immediatamente con lo specifico l'intent. Se più attività accettano l'intento, il sistema visualizza una finestra di dialogo, in modo che l'utente possa scegliere quale app utilizzare



Scelta dell'app

- Quando esiste più di un'app che risponde all'Implicit Intent, l'utente può selezionare quale app utilizzare e rendere quell'app la scelta predefinita per l'azione
 - La possibilità di selezionare un valore predefinito è utile quando si esegue un'azione per la quale l'utente probabilmente desidera utilizzare sempre la stessa app, ad esempio quando si apre una pagina Web (gli utenti spesso preferiscono un solo browser Web)
- Tuttavia, se più app sono in grado di rispondere all'intent e l'utente potrebbe voler utilizzare un'app diversa ogni volta, è necessario mostrare esplicitamente una finestra di dialogo di scelta
 - La finestra di dialogo di selezione chiede all'utente di selezionare quale app utilizzare per l'azione (l'utente non può selezionare un'app predefinita per l'azione). Ad esempio, quando la tua app esegue «share» con l'azione ACTION_SEND, gli utenti potrebbero voler condividere utilizzando un'app diversa a seconda della situazione attuale, quindi dovresti sempre utilizzare la finestra di dialogo di selezione
- Per mostrare il selettore, crea un Intent usando ***createChooser()*** e passalo a ***startActivity()***

Scelta dell'app

```
val sendIntent = Intent(Intent.ACTION_SEND)
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
val title: String = resources.getString(R.string.chooser_title)
// Create intent to show the chooser dialog
val chooser: Intent = Intent.createChooser(sendIntent, title)

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(chooser)
}
```

In questo esempio viene visualizzata una finestra di dialogo con un elenco di app che rispondono all'intent passato al metodo `createChooser()` e utilizza il testo fornito come titolo della finestra di dialogo

Ricevere un Intent implicito

- Per pubblicizzare le Intent implicite che l'app può ricevere, occorre dichiarare uno o più filtri di intent per ciascuno dei componenti dell'app con un elemento **<intent-filter>** nel file **manifest**
- Ogni intent-filter specifica il tipo di intenti che accetta in base all'azione, ai dati e alla categoria dell'intento
- Il sistema fornisce un Implicit Intent al componente dell'app solo se l'intent può passare attraverso uno dei filtri dell'intento

Intent filter

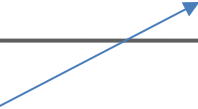
- Un componente dell'app deve dichiarare filtri separati per ogni processo univoco che può svolgere
 - Ad esempio, un'attività in un'app della galleria di immagini può avere due filtri: un filtro per visualizzare un'immagine e un altro filtro per modificare un'immagine. Quando l'attività inizia, ispeziona l'intent e decide come comportarsi in base alle informazioni nell'intent (come mostrare o meno i controlli dell'editor)
- Ogni filtro Intent è definito da un elemento <intent-filter> nel file manifest dell'app, nidificato nel componente app corrispondente (come un elemento <activity>)

<intent-filter>

- All'interno di <intent-filter>, puoi specificare il tipo di intenti da accettare usando uno o più di questi tre elementi:
 - <action>
 - Dichiarare l'azione Intent accettata, nell'attributo name. Il valore deve essere il valore di stringa letterale di un'azione, non la costante di classe
 - <data>
 - Dichiarare il tipo di dati accettati, utilizzando uno o più attributi che specificano vari aspetti dell'URI dei dati (schema, host, porta, percorso) e tipo MIME
 - <category>
 - NB: per ricevere Implicit Intent, è necessario includere la categoria CATEGORY_DEFAULT nel filtro intent. I metodi startActivity() e startActivityForResult() trattano tutti gli intent come se dichiarassero la categoria CATEGORY_DEFAULT. Se non dichiari questa categoria nel tuo filtro di intenti, nessun intent implicito si risolverà nella tua attività

category.DEFAULT

l'attività può essere lanciata solo da componenti della stessa applicazione



```
<activity android:name="ShareActivity" android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

una dichiarazione di attività con un filtro intent per ricevere un intent ACTION_SEND quando il tipo di dati è testo

NOTA! Android 12 e **exported**

- **Safer component exporting**
- Con Android 12 (o superiori), devi dichiarare esplicitamente l'attributo **android:exported** per questi componenti dell'app
- Se il componente app include la categoria LAUNCHER, imposta `android:exported` su `true`.
- Nella maggior parte degli altri casi, imposta `android:exported` su `false`
- <https://developer.android.com/about/versions/12/behavior-changes-12#exported>

Altri dettagli

android:exported

- Whether the activity can be launched by components of other applications:
- If "true", the activity is accessible to any app, and is launchable by its exact class name.
- If "false", the activity can be launched only by components of the same application, applications with the same user ID, or privileged system components. This is the default value when there are no intent filters.
- If an activity in your app includes intent filters, set this element to "true" to let other apps start it. For example, if the activity is the main activity of the app and includes the [category android.intent.category.LAUNCHER](#).
- If this element is set to "false" and an app tries to start the activity, the system throws an [ActivityNotFoundException](#).
- This attribute isn't the only way to limit an activity's exposure to other applications. Permissions are also used to limit the external entities that can invoke the activity. See the [permission](#) attribute.

android:exported

- Se `exported="false"`, l'activity può essere avviata solo da componenti della stessa applicazione o componenti di sistema privilegiati
 - Questo è il valore predefinito quando non sono presenti filtri di intento

Esempio

```
<activity android:name="MainActivity" android:exported="true">
  <!-- This activity is the main entry, should appear in app launcher -->
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>

<activity android:name="ShareActivity" android:exported="true">
  <!-- This activity handles "SEND" actions with text data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
  <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="application/vnd.google.panorama360+jpg"/>
    <data android:mimeType="image/*"/>
    <data android:mimeType="video/*"/>
  </intent-filter>
</activity>
```

- Manifest file di un social-sharing app
- La seconda activity (ShareActivity) permette di rispondere a diverse azioni Intent

MainActivity

```
<activity android:name="MainActivity" android:exported="true">
  <!-- This activity is the main entry, should appear in app launcher -->
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

La prima Activity, MainActivity, è l'entry point principale – l'activity che si apre quando l'utente esegue l'app attraverso l'icona

- L'**action_main** è l'azione che indica che è l'entry point principale e che non si aspetta intent data
- **Category_launcher** è la categoria che indica che activity icon dovrebbe essere posizionata nel sytem app launcher
- Se l'elemento <activity> non specifica un'icona con `icon`, il sistema utilizza l'icona dall'elemento <applicazione>
- Entrambi devono esserci per far sì che l'activity appaia nell'app launcher

ShareActivity

```
<activity android:name="ShareActivity" android:exported="true">
  <!-- This activity handles "SEND" actions with text data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
  <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="application/vnd.google.panorama360+jpg"/>
    <data android:mimeType="image/*"/>
    <data android:mimeType="video/*"/>
  </intent-filter>
</activity>
```

- Questa activity permette di facilitare la condivisione di testo e di media content
- Si può accedere dalla MainActivity, ma si può anche accedere a ShareActivity direttamente tramite la richiesta di altre app, tramite Android

Pending intent

- Un PendingIntent Object permette di creare una sorta di contenitore per gli Intent
- Lo scopo è di permettere ad una applicazione esterna di eseguire l'intent come se la richiesta avvenisse dalla stessa app
- Esempi di utilizzo:
 - Dichiarazione di un intent da eseguire quando l'utente esegue un'azione con la notifica (il NotificationManager del sistema Android esegue l'intent)
 - Dichiarazione di un intent da eseguire quando l'utente esegue un'azione con il widget dell'app (l'home screen app esegue l'intento)
 - Dichiarazione di un intent da eseguire in un determinato momento futuro (l'AlarmManager del sistema Android esegue l'intento)

Risoluzione di un Intent

- Quando il sistema riceve un'intent implicito di avviare un'activity, cerca l'activity migliore per l'intent confrontandola con i filtri di intent in base a tre aspetti:
 - action
 - data (sia URI che tipo di dati)
 - category

Test di *action*

- Un Intent filter può dichiarare zero o più elementi azione (<action>)
- Per superare il filtro, l'azione specificata nell'oggetto Intent deve fare match con almeno una azione
- MA, se l'oggetto Intent non specifica un'azione, passa il test se almeno un action è definita nel filtro

```
<intent-filter>
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.VIEW" />
    ...
</intent-filter>
```

Test di *category*

- Un intent filter può specificare zero o più elementi <category>
- Perché un intent superi il test di categoria, ogni categoria nell'intent (oggetto) deve corrispondere a una categoria nel filtro
- Non è necessario il contrario: il filtro Intent può dichiarare più categorie di quelle specificate nell'Intent (oggetto) e l'Intent ancora passa
 - Pertanto, un intent senza categorie supera sempre questo test, indipendentemente dalle categorie dichiarate nel filtro
- Nota: android applica la categoria CATEGORY_DEFAULT a tutti gli implicit intent passati a startActivity() e startActivityForResult()

android.intent.category.DEFAULT

<intent-filter>

<category android:name="android.intent.category.DEFAULT" />

<category android:name="android.intent.category.BROWSABLE" />

...

</intent-filter>

Si deve mettere

«**android.intent.category.DEFAULT**»

se si vuole che l'activity riceva
implicit filter

Test di *data*

- Per specificare i data di intent accettati, un filtro di intent può dichiarare zero o più elementi <data>
- Ogni elemento <data> può specificare una struttura URI e un tipo di dati (MIME media type)
- Ogni parte dell'URI è un attributo separato: schema, host, port e path (ognuno dei quali opzionale)

<scheme> : // <host> : <port> / <path>

Esempio:

content://com.example.project:200/folder/subfolder/etc

schema

host

port

path

Data

```
<intent-filter>  
    <data android:mimeType="video/mpeg"  
android:scheme="http" ... />  
    <data android:mimeType="audio/mpeg"  
android:scheme="http" ... />  
    ...  
</intent-filter>
```

Test di *data*: regole

- Un intent che non contiene né un URI né un tipo MIME supera il test solo se il filtro non specifica alcun URI o tipo MIME
- Un intent che contiene un URI ma nessun tipo MIME (né esplicito né inferibile dall'URI) supera il test solo se il suo URI corrisponde al formato URI del filtro e allo stesso modo il filtro non specifica un tipo MIME
- Un intent che contiene un tipo MIME ma non un URI supera il test solo se il filtro elenca lo stesso tipo MIME e non specifica un formato URI
- Un intent che contiene sia un URI che un tipo MIME (esplicito o inferibile dall'URI) passa la parte del tipo MIME del test solo se quel tipo corrisponde a un tipo elencato nel filtro. Passa la parte URI del test se il suo URI corrisponde a un URI nel filtro o se ha un *content:* o *file:* URI e il filtro non specifica un URI

Esempio: image/*

```
<intent-filter>
```

```
  <data android:mimeType="image/*" />
```

```
  ...
```

```
</intent-filter>
```

ES: L'element <data> dice ad Android che il componente può ricevere image data da un content provider e visualizzarlo

Esempio: video/*

<intent-filter>

<data android:scheme="http" android:mimeType="video/*" />

...

</intent-filter>

Es. L'elemento data così definito dice ad Android che il componente può recuperare i dati video dalla rete per eseguire l'azione

PackageManager

- La classe PackageManager fornisce dei metodi per determinare che componenti sono in grado di rispondere ad un intent
- Esempio
 - `queryIntentActivities()` restituisce una lista con tutte le attività che possono eseguire l'intent passato come argomento
 - `queryIntentServices()` fa la stessa cosa per i services

Implicit Intent

- Ci sono moltissime azione che si possono svolgere attraverso implicit intent relativamente a:
 - Alarm Clock ([ACTION_SET_ALARM](#), [ACTION_SET_TIMER](#))
 - Calendar
 - Camera
 - Contacts/People app
 - Email
 - File storage
 - Maps
 - Music and Video
 - Setting
 - Web Browsers
 - Search
 - ...
 - <https://developer.android.com/guide/components/intents-common>

Camera

- Per aprire l'app della fotocamera e ricevere la foto o il video risultante, utilizzare l'azione ACTION_IMAGE_CAPTURE o ACTION_VIDEO_CAPTURE
- Occorre anche specificare la posizione URI in cui si desidera che la fotocamera salvi la foto o il video, in EXTRA_OUTPUT extra
- Quando l'app della fotocamera riporta il focus sulla tua attività, (l'app riceve il callback onActivityResult()), si può accedere alla foto o al video nell'URI specificato con il valore EXTRA_OUTPUT

Esempio: uso della camera (fino alla 10)

```
const val REQUEST_IMAGE_CAPTURE = 1  
val locationForPhotos: Uri = ...
```

```
fun capturePhoto(targetFilename: String) {  
    val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE).apply {  
        putExtra(MediaStore.EXTRA_OUTPUT, Uri.withAppendedPath(locationForPhotos, targetFilename))  
    }  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivityForResult(intent, REQUEST_IMAGE_CAPTURE)  
    }  
}
```

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent) {  
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Activity.RESULT_OK) {  
        val thumbnail: Bitmap = data.getParcelableExtra("data")  
        // Do other work with full size photo saved in locationForPhotos  
        ...  
    }  
}
```

Con Android
fino alla
versione 10!!



Warning:
vedi ultime slide!

Esempio: uso della camera (Android 11+)

```
val REQUEST_IMAGE_CAPTURE = 1
```

```
private fun dispatchTakePictureIntent() {  
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)  
    try {  
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)  
    } catch (e: ActivityNotFoundException) {  
        // display error state to the user  
    }  
}
```

Con Android
versione 11 e
superiori!!



Warning:
vedi ultime slide!

Per esporre l'intent camera

Se la vostra app vuole esporre la funzionalità camera, allora dovete inserire l'intent-filter nel vostro file manifest.xml

```
<activity ...>
  <intent-filter>
    <action android:name="android.media.action.IMAGE_CAPTURE"
  />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

E-mail

```
fun composeEmail(addresses: Array<String>, subject: String, attachment: Uri) {  
    val intent = Intent(Intent.ACTION_SEND).apply {  
        type = "*/*"  
        putExtra(Intent.EXTRA_EMAIL, addresses)  
        putExtra(Intent.EXTRA_SUBJECT, subject)  
        putExtra(Intent.EXTRA_STREAM, attachment)  
    }  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivity(intent)  
    }  
}
```

ACTION_SENDTO (no allegati) oppure
ACTION_SEND (per un allegato) or
ACTION_SEND_MULTIPLE (per vari allegati)

E-mail

- Per essere sicuri che l'azione venga eseguita con un app per le email e non altre app di messaggistica, usare *mailto*:

```
fun composeEmail(addresses: Array<String>, subject: String) {  
    val intent = Intent(Intent.ACTION_SENDTO).apply {  
        data = Uri.parse("mailto:") // only email apps should handle this  
        putExtra(Intent.EXTRA_EMAIL, addresses)  
        putExtra(Intent.EXTRA_SUBJECT, subject)  
    }  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivity(intent)  
    }  
}
```

Per esporre e-mail action

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.SEND" />
    <data android:type="*/*" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.SENDTO" />
    <data android:scheme="mailto" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

Mappa

- Per aprire una mappa: **ACTION_VIEW** e specificare la localizzazione tramite URI (es. **geo:latitude,longitude**)

```
fun showMap(geoLocation: Uri) {  
    val intent = Intent(Intent.ACTION_VIEW).apply {  
        data = geoLocation  
    }  
    if (intent.resolveActivity(packageManager) != null) {  
        startActivity(intent)  
    }  
}
```

Mappa: intent filter

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <data android:scheme="geo" />
    <category android:name="android.intent.category.DEFAULT"
  />
  </intent-filter>
</activity>
```

- Per maggiori dettagli sulle altre *action*, controllate la documentazione al seguente link
- <https://developer.android.com/guide/components/intents-common>

IMPORTANTE

- Abbiamo detto che per richiamare una specifica azione con un intent implicito, basta richiamare il metodo «resolveActivity()» che poi a sua volta utilizza «getPackageManager()» per ottenere tutte le app che offrono quell'azione
- ```
if (intent.resolveActivity(packageManager) != null) {
 startActivity(intent)
}
```

# IMPORTANTE: Android 11 e package visibility

---

- Benissimo! Ma solo fino alla versione Android 10
- Con Android 11 (API level 30) si sono introdotte nuove restrizioni relative a «package visibility» (<https://developer.android.com/about/versions/11/privacy/package-visibility>)
- E non è possibile interagire direttamente con la maggior parte di package esterni

# IMPORTANTE: Android 11 e package visibility

- Due possibili soluzioni:
  - Dare il permesso generico QUERY\_ALL\_PACKAGES
  - Oppure, consigliato, utilizzare un element <queries> nel manifest.xml

Esempio:

...

```
<queries>
 <intent>
 <action android:name="android.intent.action.VIEW" />
 <category android:name="android.intent.category.BROWSABLE"/>
 <data android:scheme="https" />
 </intent>
</queries>
```

# IMPORTANTE: Android 11 e package visibility

---

- L'elemento `<queries>` permette quindi di filtrare i risultati dei metodi di `PackageManager` che restituiscono risultati relative ad altre app

# Esempio più completo: nel Manifest.xml

```
....
<queries>
 <!-- Browser -->
 <intent>
 <action android:name="android.intent.action.VIEW"/>
 <data android:scheme="http" />
 </intent>
 <!-- Camera -->
 <intent>
 <action android:name="android.media.action.IMAGE_CAPTURE" />
 </intent>
 <!-- Gallery -->
 <intent>
 <action android:name="android.intent.action.GET_CONTENT" />
 </intent>
</queries>
...
```

Potete aggiungere  
l'elemento <queries>  
sopra all'elemento  
<application> ...  
</application>

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.progetto">

 <queries>
 <intent>
 <action android:name="android.intent.action.SEND"/>
 <data android:mimeType="text/plain"/>
 </intent>
 </queries>

 <application
 android:allowBackup="true"
 android:icon="@mipmap/ic_launcher"
 android:label="Travels"
 android:roundIcon="@mipmap/ic_launcher_round"
 android:supportsRtl="true"
 android:theme="@style/Theme.Progetto">
 <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=".SettingsActivity" />
 </application>

</manifest>
```

# Super importante..!

- Non è assolutamente chiaro dalla documentazione ma se si usa «ComponentActivitiy» allora «**startActivityForResult**» è **deprecato**!
- Negli altri casi (con Activity), non è deprecato ma comunque la guida di Android dice:
- «While the underlying [startActivityResult\(\)](#) and [onActivityResult\(\)](#) APIs are available on the Activity class on all API levels, Google strongly recommends using the Activity Result APIs introduced in AndroidX [Activity](#) and [Fragment](#) classes. The Activity Result APIs provide components for registering for a result, launching the result, and handling the result once it is dispatched by the system.»

```
java.lang.Object
↳ android.content.Context
↳ android.content.ContextWrapper
↳ android.view.ContextThemeWrapper
↳ android.app.Activity
↳ androidx.activity.ComponentActivity
```

startActivityResult

```
public void startActivityResult(@NonNull Intent intent, int requestCode)
```

! This method is deprecated.

This method has been deprecated in favor of using the Activity Result API which brings increased type safety via an [ActivityResultContract](#) and the prebuilt contracts for common intents available in [androidx.activity.result.contract.ActivityResultContracts](#), provides hooks for testing, and allow receiving results in separate, testable classes independent from your activity. Use [registerForActivityResult](#) passing in a [StartActivityResult](#) object for the [ActivityResultContract](#).

# Super importante..!

---

- Cosa occorre fare ora?
- Si deve usare:
  - **registerForActivityResult()**
- Quando si avvia un'Activity per ottenere un risultato, è possibile (e, in caso di operazioni ad uso intensivo di memoria, quasi certo) che il processo e l'activity vengano distrutti a causa della memoria insufficiente
- Per questo motivo, l'Activity Result APIs disaccoppiano il callback del risultato dal punto del codice in cui si avvia l'altra Activity. Poiché la richiamata dei risultati deve essere disponibile quando il processo e l'attività vengono ricreati, la richiamata deve essere registrata incondizionatamente ogni volta che viene creata l'Activity, anche se la logica di avvio dell'altra Activity avviene solo in base all'input dell'utente o ad altra business logic



# registerForActivityResult()

- **registerForActivityResult()** prende un [ActivityResultContract](#) e un [ActivityResultCallback](#) e restituisce un [ActivityResultLauncher](#), che sarà usato per launch l'altra Activity
- Un ActivityResultContract definisce il tipo di input necessario per produrre un risultato insieme al tipo di output del risultato
- Le API forniscono contratti predefiniti per azioni di base come scattare una foto, richiedere autorizzazioni e così via. Puoi anche creare un contratto personalizzato.
- ActivityResultCallback è un'interfaccia a metodo singolo con un metodo onActivityResult() che accetta un oggetto del tipo di output definito in ActivityResultContract

```
val getContent = registerForActivityResult(GetContent()) { uri: Uri? ->
 // Handle the returned Uri
}
```

# registerForActivityResult()

---

- RegisterForActivityResult() è sicuro da chiamare prima che venga creato l'activity, consentendone l'utilizzo direttamente durante la dichiarazione delle variabili membro per le istanze ActivityResultLauncher restituite
- È necessario chiamare RegisterForActivityResult() prima che il l'activity venga creata, ma non è possibile avviare ActivityResultLauncher finché il ciclo di vita dell'activity non ha raggiunto CREATED

```
val getContent = registerForActivityResult(GetContent()) { uri: Uri? ->
 // Handle the returned Uri
}

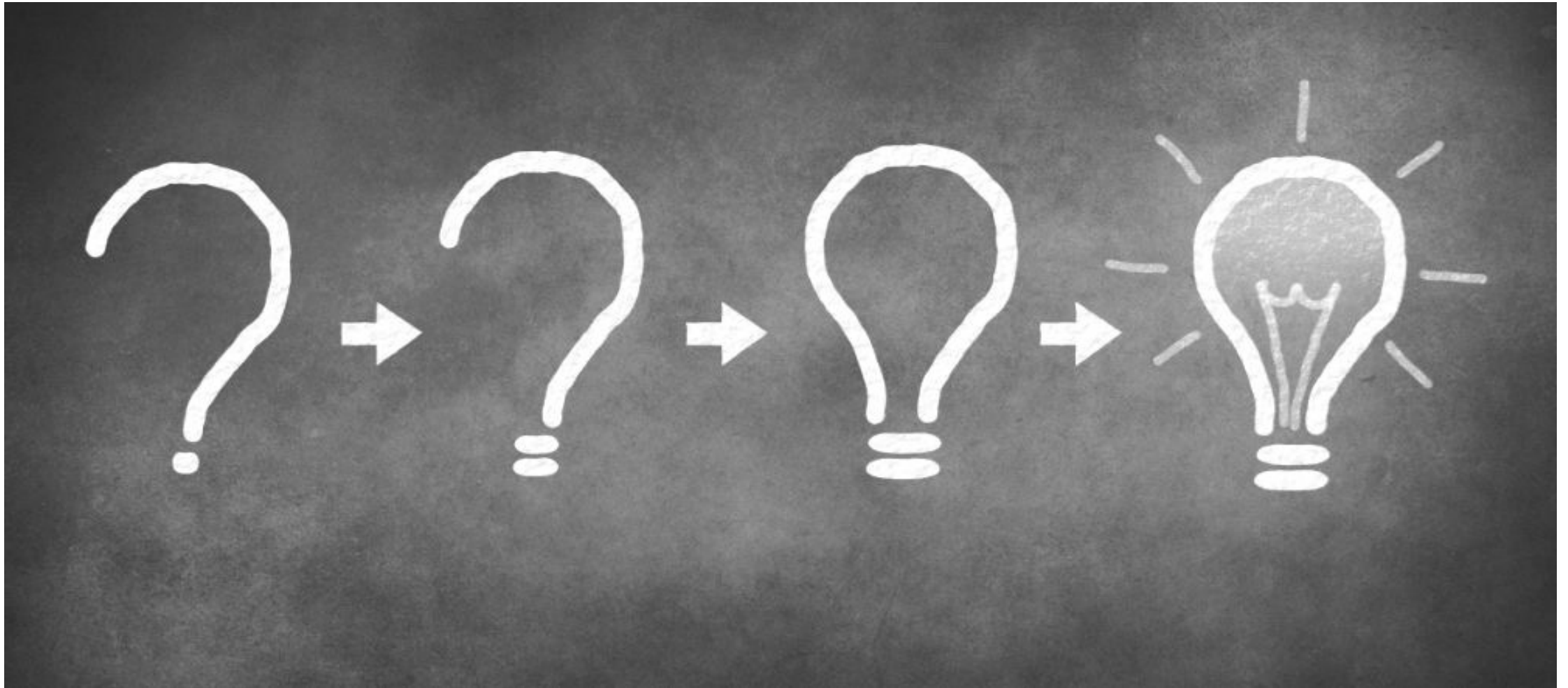
override fun onCreate(savedInstanceState: Bundle?) {
 // ...

 val selectButton = findViewById<Button>(R.id.select_button)

 selectButton.setOnClickListener {
 // Pass in the mime type you want to let the user select
 // as the input
 getContent.launch("image/*")
 }
}
```

In  
Lab!

# Domande?



# Riferimenti

---

- <https://developer.android.com/guide/components/intents-filters>
- <https://developer.android.com/reference/android/content/Intent>
- <https://developer.android.com/guide/components/intents-common>
- <https://developer.android.com/training/sharing/send>
- <https://www.tutlane.com/tutorial/android/android-intent-filters-with-examples>
- <https://www.tutlane.com/tutorial/android/android-camera-app-with-examples>
- <https://developer.android.com/training/sharing/send>
- <https://developer.android.com/guide/topics/manifest/queries-element>
- <https://developer.android.com/training/basics/intents/result>