04 - ACTIVITIES

Un'applicazione è composta da uno o più componenti dichiarati nel manifest file:

- Activities
- Services
- Broadcast receivers
- Content provider

Automaticamente creati dal S.O. in risposta ad una richiesta contenente l'intento

Android.app.activity

Classe base che **fornisce una GUI** con la quale gli utenti possono interagire per fare qualcosa
Istanziate automaticamente dal sistema che gestisce il ciclo di vita attraverso metodi:

- `onCreate(…)`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`

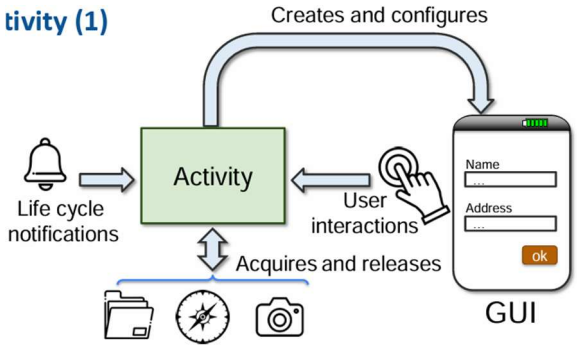
ruoli e doveri/responsabilities:

Le activity hanno molte responsibilities/doveri:

- **Acquisire risorse**
  - Per funzionare, l'applicazione necessita di accedere a CPU, di accedere a dati, memoria, camera, sensori…
- **Creare e configurare GUI**
  - Android pre-alloca una finestra vuota per un'attività → tocca a me riempirla
- **Reagire ad eventi** triggerati da iterazioni con utente
  - Decodificare evento → implementare comportamento desiderato
- Gestire le notifiche riguardanti il **ciclo di vita**
  - salvando dati collezioinati dall'interfaccia e rilasciando le risorse quando lo richiede il s.o

note su activity

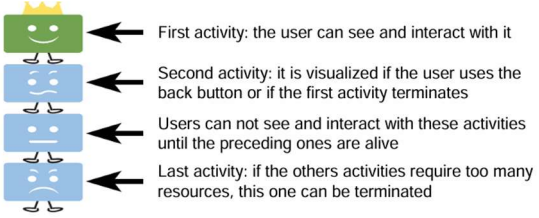
- un'attività tipicamente mostra una sola interfaccia che permente solo alcuni tipi di interazione utente.
- Un'applicazione può contenere tante attività
  - Ognuna per un single task piccolo (creare un nuovo messaggio, leggere un messaggio ricevuto, gestire messaggi)
  - 1 delle attività viene marcata nel manifest file per essere la prima da essere mostrata all'utente quando l'applicazione viene lanciata.
- Un'attività può lanciare un'altra attività, creando un oggetto intent. (di stessa o diversa applicazione)



***NOTA**: Per ogni task iniziato nella home screen del device, il sistema operativo crea un activity stack (ad esempio: gmail-stack, WA-stack) → lo stack è inizializzato dall'attività di default del task corrispondente.*

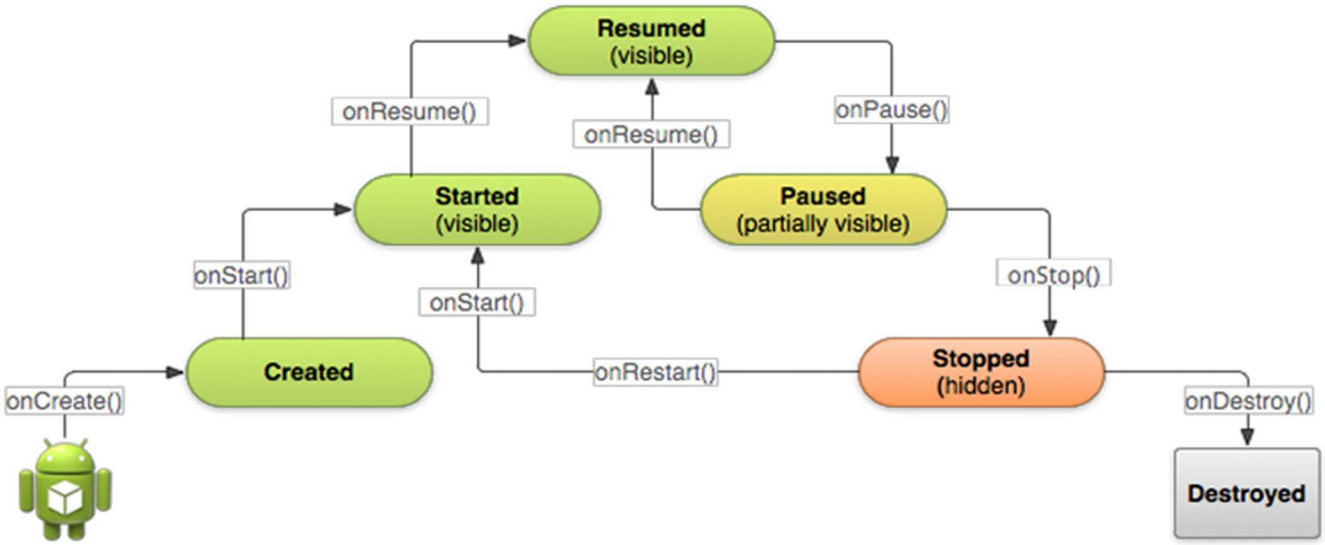
Durante il suo ciclo di vita, un'attività può fare richiesta al sistema di iniziare una **nuova attività:**

- Questa viene creata e inserita **al top dello stack** e diventa visibile e interattibile
- L'attività precedente viene shiftata di una posizione e rimane in background finché è viva
- Se la nuova attività termina o l'utente preme BACK, la precedente attività raggiunge il top dello stack e diventa nuovamente visibile e interattibile



Note su **ciclo di vita** delle attività:

- Ci possono essere molti stack di attività contemporaneamente
- Un'attività può essere interrotta o messa in pausa quando determinati eventi sono triggerati
    - Chiamata ricevuta
- Android invia diverse **notifiche di tracciamento** dello status di un'applicazione e la sua evoluzione in base all'interazione utente, eventi a livello di sistema e disponibilità di risorse
    - Bisogna svolgere azione necessarie a garantire la gestione corretta delle risorse dell'applicazione
    - Rilasciare e riacquisire se necessario (es: youtube quando lo faccio diventare pop-up)
- La gestione del ciclo di vita include situazioni meno ovvie:
    - Rotazione schermo → solitamente attività distrutta e ricreata con i nuovi parametri
    - Cambio lingua → attività distrutta e ricreata



- **Created state**: attività esiste ma non visibile
- **Started state**: attività popolata, aggiunta GUI, ora è anche visibile → user vede sullo screen ma l'utente non può ancora interagire con l'applicazione
- **Resumed state**: l'attività è in primo piano e l'utente può interagire con essa → fully interactive
- **Paused state**: attività non più interactive, è visible/partially visible (utente ha premuto back, switch -off-, chiamata in arrivo)
- **Stopped state**: attività stoppata
    - Se faccio onRestart → applicazione restartata dal background
    - Se faccio onDestroy → chiusa definitivamente

NOTA: non salvare riferimenti ad attività altrimenti si va ad inibire le funzionalità di Android nel rilasciare attività e risorse.

Gestione del ciclo di vita dell'attività

Nella maggior parte dei casi non bisogna far nulla, nel caso in cui i metodi base siano overriden e si voglia accedere a quelli originali, si deve usare super.funzione (Sto accedendo alla superclass)

```
class MainActivity : AppCompatActivity() {

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // other actions
  }
```

onCreate è l'unico metodo che arriva con i parametri.
Usato in due scenari:

- **Acceso dell'utente da zero** → b=null
- Android aveva chiusa applicazione e la **sta riaprendo**, b preso da disco e ricaricato
    - Contiene informazioni di stato che passiamo al super

```
class ExampleActivity: AppCompatActivity() {

  override fun onCreate(b: Bundle?) {

    super.onCreate(b)

    //Programmatically create and initialize a view
    val tv = TextView(this)
    tv.text = "Hello, Android!"
    tv.gravity = Gravity.CENTER

    //show the view content
    setContentView(tv)
  }
}
```

onStart(): metodo chiamato quando l'applicazione diventa visibile all'utente

- Se l'attività era stata visibile precedentemente, prima di lui viene chiamato restart

onResume(): metodo chiamato quando l'attività raggiunte il top dello stack e e diventa interattiva

- Partono Animazioni, video, suoni, acquisizione temporanea delle risorse

onPause() → metodo chiamato per spostare attività nella seconda posizione dello stack delle attività

- Vengono fermate animazioni, video e suoni
- Importante rilasciare tutte le risorse non necessarie per il background (Stoppo il timer, etc)
    - Commit dei dati + togliere registrazione come listener di eventi

onStop() : metodo chimaato quando l'applicazione non è poiù visibile agli utenti

onDestroy(): attività terminata, viene rimossa da memoria. Può essere invocato:

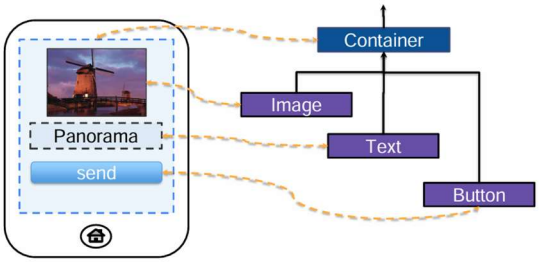
- su esplicita richiesta dell'applicazione (chiamato metodo finish(...))
- dal s.o. per richiedere delle risorse

Nota: per capire se finish è stato invocato su un'app o dal s.o. si può usare il metodo isFinishing()

Preparazione della GUI (onCreate):

Setup dell'interfaccia utente e renderla visibile:

- Vista è un albero di oggetti, connessi in relazione padre-figlio
    - Figli sono sotto riquadri dei padri
    - Insieme di metodi comuni
        - **onMeasure()** → quanto grande oggetto
            - quanto spazio necessita
        - **onDraw()** → ho fornito una porzione dello spazio su schermo, per favore ridimensionati



Interfaccia utente è dinamica ed evolve, **event driven**.

- Virtual tree con componenti
    - Con una libreria grafica molto potente
    - Apparentemente sto definendo funzioni per scrivere su schermo → funzioni senza return
        - Le funzioni sono con notazione Composable
            - Compiler le manipola e fa di tutto con esse
        - Solo setContent può invocare altre composable function
            - Le altre composable function non possono invocare altre composable function

Nota: Inizialmente si pensava di usare un codice procedurale con object oriented, successivamente si è passati ad un what do u want to see approach ma servivano troppi oggetti. Successivamente nuove tecniche:react con object oriented → class extended react object.
Infine, si è pensato al virtual tree con componenti.

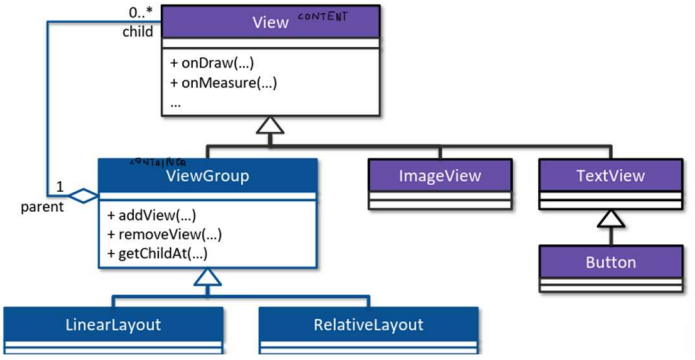
A noi compare come un albero con un insieme di widgets
onCreate() crea il tree e fornisce la posizione di questo
majorArea è coperta dalla route (Container) ed è splittata in 3 sotto-aree.
insieme di oggetti devono essere derivati da un insieme di classi fornite da Android per modellare widgets della GUI.

Gerarchia:

La classe android.view.View rappresenta ogni widget che può essere mostrato a schermo.
La sottoclasse android.view.ViewGroup rappresenta un sottoinsieme di view che può comportarsi come un visualContainer con figli.

Composite pattern → classi combinate al fine di riuscire ad ottenere un determinato scopo.

Content ha come figlii dei containter che possono avere vari figli → container



Creare gerarchia

setContentView(Container) è usata per istanziare un albero, questo ha un problema perché il nr di proprietà che devo settare può essere troppo elevato.

Dotato di grande **flessibilità** in quanto si può inserire/configurare ogni componente in base ai dati processati da applicazione; e grande **controllo** in quanto il programmatore decide cosa creare.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
// create and configure the main container
    val ll = LinearLayout(this)
    ll.orientation = LinearLayout.VERTICAL
// create and configure the parts it contains
    val iv = ImageView(this)
    iv.setImageResource(R.mipmap.ic_launcher)
    val tv = TextView(this)
    tv.text = "Panorama"
    tv.gravity = Gravity.CENTER_HORIZONTAL
    val b = Button(this)
    b.text = "Send"
    ll.addView(iv)
    ll.addView(tv)
    ll.addView(b)
// show the hierarchy
    setContentView(ll)
}
```

Creare vista gerarchica tramite XML: "res/layout" descrive la gerarchia visiva

- L'ambiente di sviluppo associa automaticamente una costante intera nella classe R.layout ad ogni file xml
- The name of the constant is the name of the file, the value is automatically selected in order to make it unique.

Esiste una seconda versione di setContentView che accetta un intero → costante predefinita legata ad uno screen attraverso reference ad un file.

```
override fun onCreate(b: Bundle?) {
   super.onCreate(b)
   setContentView(R.layout.activity_main)
}
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
            "http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
  <ImageView
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:src="@drawable/panorama"/>
  <TextView
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="@string/picture_text"/>
  <Button android:id="@+id/button1"
     android:text="@string/button_label"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"/>
</LinearLayout>
```

Mentre la gerarchia è descritta su XML, customizzazione può essere fatta via codice.
Ogni elemento ha un attributo ID, a cui può essere assegnato un valore:
`android: id = "@+id/element_name"`
Il compilatore crea quindi una costante chiamata **R.id.element_name** e le associa un valore unico.
Dopo aver caricato il file XML, si può fare riferimento ad ogni widget usando il metodo generico: `findViewById<T: View>(id: Int): T?`

```
override fun onCreate(savedInstanceState: Bundle?) {

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

   val button : Button = findViewById(R.id.button1)

   button.setOnClickListener {
      // … do something when button is pressed
```

Jetback compose library

Creare un compose → formato da funzione **@composable** il cui body invoca altre funzioni composable.

Es: Funzione emit column

Column capisce che è un composable che mette i propri figli insieme:

- *Image*
- *Text*
- *Button*

```
@Composable
fun Panorama(name: String) {
    Column(
        Modifier.padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
    ) {
        Image(
                painter = painterResource(R.drawable.ic_launcher_background),
          contentDescription = null,
          modifier = Modifier.fillMaxSize(0.8f)
        )
        Text("$name!")
        Button(onClick = { /* TODO */ }) {
            Text("Send")
        }
    }
}
```

Fully declarative, compatible, designed for and with material design, Pochi tools richiesti.

NOTA SUI PROCESSI ANDROID:

il processo che ospita un'applicazione viene creato e distrutto quando lo decide il sistema, non l'utente.

- Chiudendo un'applicazione non chiudo il processo
- Processi possono terminare anche con applicazione ancora aperta

Un'attività in **stopped state** può esser uccisa in ogni momento da Android, se quest'ultimo necessità di risorse.

Quindi, informa l'attività che sarà killata invocando **onSaveInstanceState()** → devi storare i tuoi dati e passarmeli nel bundle in modo tale che sia serializzabile e che io possa metterlo su disco.

Quando il s..o. vuole **ricreare il processo, userà onCreate() e onRestoreIstanceState().**

Le attività possono chiedere ad altre attività di partecipare usando un **intent**: (startActivity(), startService(), sendBroadcast())

- **Implicit intent** → dico cosa mi serve ma non dico chi deve farlo → necessito che sia mandata una e-mail
  - **Action**: stringa univoca presa da una lista predefinita o specificata dal programmatore
  - **URI**: identificatore di risorsa il cui schema può essere usato per identificare possibili recipienti
  - **Category** → stringa che provvede dettagli addizionali all'azione richiesta

```
val action:String = Intent.ACTION_DIAL
val uri:URI = Uri.parse("tel:+390110905555")
val dialer = Intent(action, uri)

startActivity(dialer)
```

- **Explicit intent** → indica il componente specifiche che deve essere attivato.

Quando invoco start_activity una transazione inizia