Architettura NAOqi e Framework QiSDK

NAOqi è il cuore del sistema operativo di Pepper e fornisce le API per controllare il robot. È strutturato in moduli principali:

- ALTextToSpeech → Controllo della sintesi vocale
- ALMotion → Controllo del movimento
- ALAudio → Gestione del riconoscimento vocale
- ALBehaviorManager → Esecuzione di comportamenti predefiniti
- ALFaceDetection → Riconoscimento dei volti

Codice Realizzato

Ogni classe che deve interagire con pepper deve estendere o implementare in caso di multiereditarietà RobotActivity. Ho aggiunto alcuni import non necessari nel Main in ma che potrebbero essere necessari per altre funzionalità.

```kotlin
import android.content.Context
import android.media.MediaPlayer
import android.os.AsyncTask
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.Toast
import com.aldebaran.qi.Future
import com.aldebaran.qi.sdk.QiContext
import com.aldebaran.qi.sdk.QiSDK
import com.aldebaran.qi.sdk.RobotLifecycleCallbacks
import com.aldebaran.qi.sdk.`object`.actuation.Animate
import com.aldebaran.qi.sdk.`object`.actuation.Animation
import com.aldebaran.qi.sdk.builder.AnimateBuilder
import com.aldebaran.qi.sdk.builder.AnimationBuilder
import com.aldebaran.qi.sdk.builder.SayBuilder
import com.aldebaran.qi.sdk.design.activity.RobotActivity

class MainActivity : RobotActivity(), RobotLifecycleCallbacks {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Register the RobotLifecycleCallbacks to this Activity.
        QiSDK.register( activity: this, callbacks: this)
    }

    override fun onDestroy() {
        // Unregister the RobotLifecycleCallbacks for this Activity.
        QiSDK.unregister( activity: this, callbacks: this)
        super.onDestroy()
    }

    override fun onRobotFocusGained(qiContext: QiContext) {
        // The robot focus is gained.
    }

    override fun onRobotFocusLost() {
        // The robot focus is lost.
    }

    override fun onRobotFocusRefused(reason: String) {
        // The robot focus is refused.
    }
}

```

Riconoscere parole chiavi

```kotlin
fun riconoscereParole(qiContext: QiContext) {
    val phraseSet = PhraseSetBuilder.with(qiContext)
            .withTexts( ...texts: "Si", "No", "Ripeti")
            .build()

    val listen = ListenBuilder.with(qiContext)
            .withPhraseSet(phraseSet)
            .build()

    val result = listen.run()
    val parolaRiconosciuta = result.heardPhrase.text

    println("Pepper ha riconosciuto: $parolaRiconosciuta")
}
```

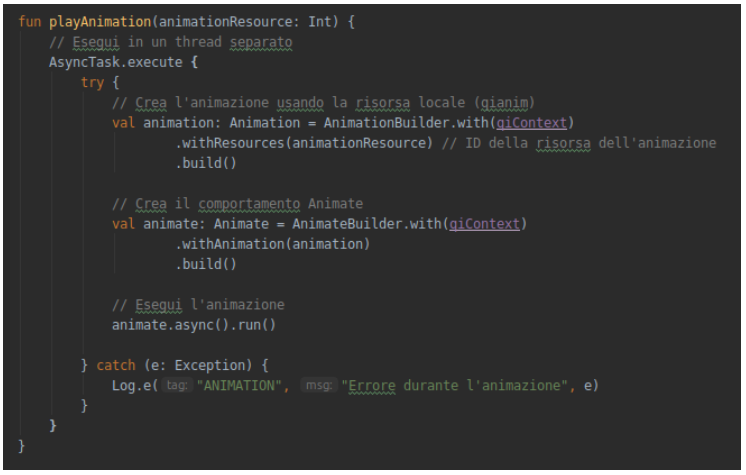
0.1 Far parlare Pepper

```
fun speak(frase: String) {
    val context = qiContext
    if (context != null) {
        // Esegui l'operazione in un thread separato usando AsyncTask
        AsyncTask.execute {
            try {
                // Crea l'oggetto Say
                val say = SayBuilder.with(context)
                        .withText(frase)
                        .build()

                // Esegui l'operazione di parlato in background
                val future: Future<Void> = say.async().run()

                // Gestisci il risultato quando l'operazione è completata
                future.thenConsume { result ->
                    if (result.hasError()) {
                        Log.e( tag: "SPEAK",  msg: "Errore durante il parlato", result.error)
                    } else {
                        Log.d( tag: "SPEAK",  msg: "Pepper ha parlato con successo")
                    }
                }
            } catch (e: Exception) {
                Log.e( tag: "SPEAK",  msg: "Errore", e)
            }
        }
    } else {
        // Mostra un messaggio se QiContext non è disponibile
        Toast.makeText(context,  text: "QiContext non disponibile", Toast.LENGTH_SHORT).show()
    }
}
```

Riprodurre un suono su Pepper

```
fun soundSuccess() {
    mediaPlayer?.let {  it: MediaPlayer
        if (it.isPlaying) {
            it.stop()
            it.release()
            mediaPlayer = null
        }
    }

    mediaPlayer = MediaPlayer.create(context, R.raw.success)
    mediaPlayer?.let {  it: MediaPlayer
        Toast.makeText(context,  text: "Play sound on Robot", Toast.LENGTH_LONG).show()
        it.start()

        it.setOnCompletionListener { mp ->
            mp.release()
            mediaPlayer = null
        }
    }
}
```

eseguire un animazione .qanim su Pepper

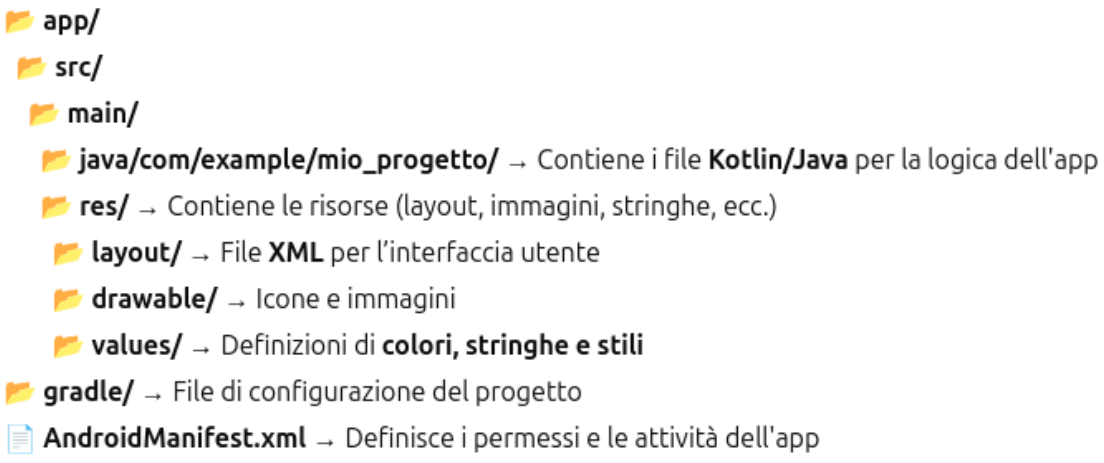


```
fun playAnimation(animationResource: Int) {
    // Esegui in un thread separato
    AsyncTask.execute {
        try {
            // Crea l'animazione usando la risorsa locale (qianim)
            val animation: Animation = AnimationBuilder.with(qiContext)
                    .withResources(animationResource) // ID della risorsa dell'animazione
                    .build()

            // Crea il comportamento Animate
            val animate: Animate = AnimateBuilder.with(qiContext)
                    .withAnimation(animation)
                    .build()

            // Esegui l'animazione
            animate.async().run()

        } catch (e: Exception) {
            Log.e( tag: "ANIMATION",  msg: "Errore durante l'animazione", e)
        }
    }
}
```

Tecnologie e Piattaforme

0.2 Struttura del progetto

La struttura del progetto le cartelle o file può variare in base alla versione. Ciò significa che la loro posizione sarà diversa ma non il loro utilizzo!. Anche se usate una versione diversa e la posizione delle cartelle cambia riuscirete ad adattarvi a qualsiasi versione grazie ad un po' di teoria di base.



Configurazione dell'ambiente di sviluppo

Pepper è un robot umanoide sviluppato da SoftBank Robotics, progettato per interagire in modo naturale con gli esseri umani. Per sviluppare applicazioni su Pepper utilizzando Android, SoftBank Robotics ha reso disponibile il Pepper SDK per Android (QiSDK), basato sul framework NAOqi.

Il framework NAOqi è l'architettura software principale che gestisce il comportamento del robot, le sue funzionalità sensoriali e le API per l'interazione con gli sviluppatori.

Qui una panoramica delle tecnologie utilizzate e i passaggi per sviluppare applicazioni per Pepper con Android Studio. Per iniziare lo sviluppo con Pepper su Android, è necessario:

Connessione con un robot Pepper (opzionale per i test reali)

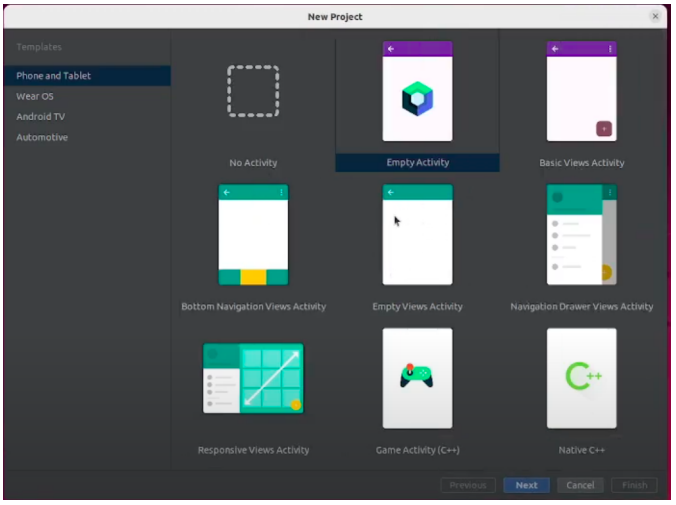
- Android Studio (versione compatibile Ad esempio la 4.1 Minimo 3.4)
- SDK Android (minimo API Level 23 o superiore)
- Pepper SDK per Android (QiSDK).

Installazione di Android Studio

- Scaricare e installare Android Studio dal sito ufficiale: https://developer.android.com
  Se avete linux installate la versione binaria. aprite la cartella android e navigate fino alla directory bin. Dopo di che avviate studio.sh con il comando ./studio.sh. in tal caso abbiate difficolta cercare sempre il file studio.sh.

  In caso abbiate Windows basta scaricare sul sito ufficiale la versione di windows e seguire le istruzione grafiche farà tutto in automatico.

- Aprire Android Studio e creare un nuovo progetto e selezionare con "Empty Activity"



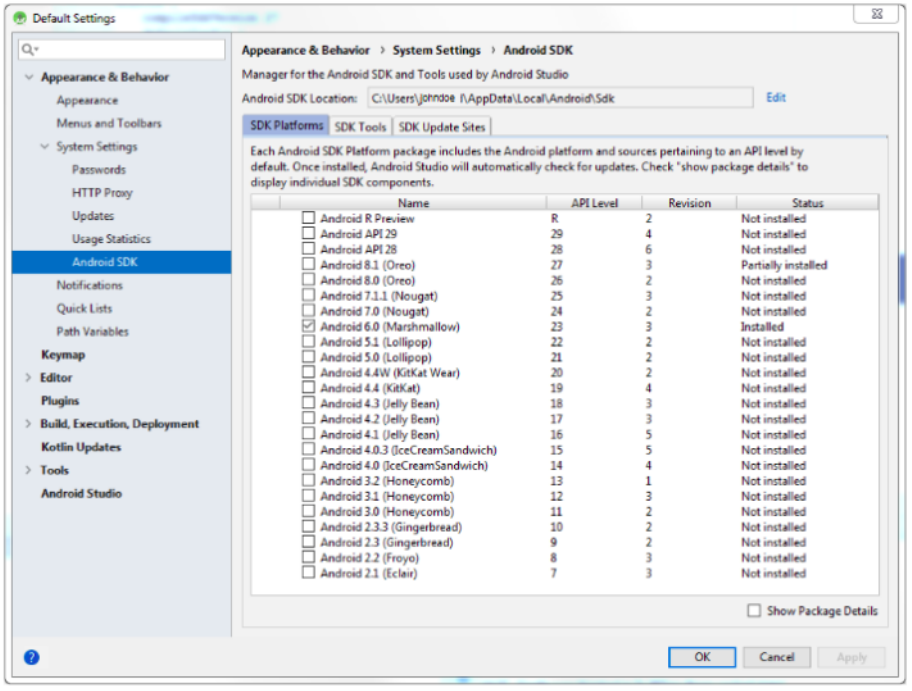
– Configurare il progetto con Java/Kotlin e selezionare API Level 23+

Passi necessari al funzionamento di android e Pepper SDK(QIsdk)

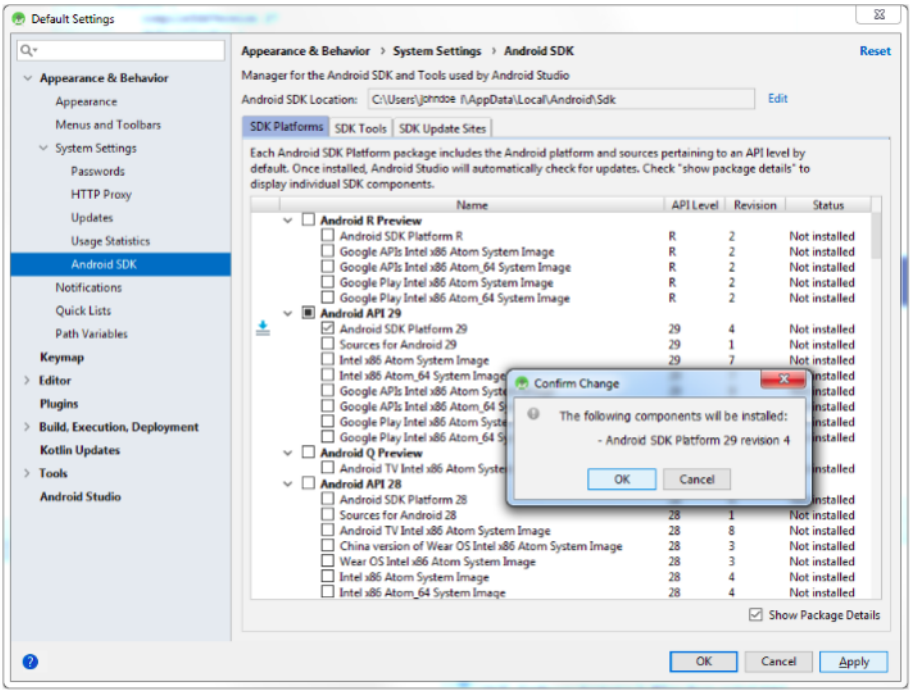
Per integrare Pepper SDK nel progetto, modificare il file build.gradle (Module: app) e aggiungere:

dependencies implementation 'com.aldebaran:qisdk:1.7.5' // Verificare l'ultima versione disponibile Successivamente, sincronizzare il progetto cliccando su "Sync Now" in Android Studio.

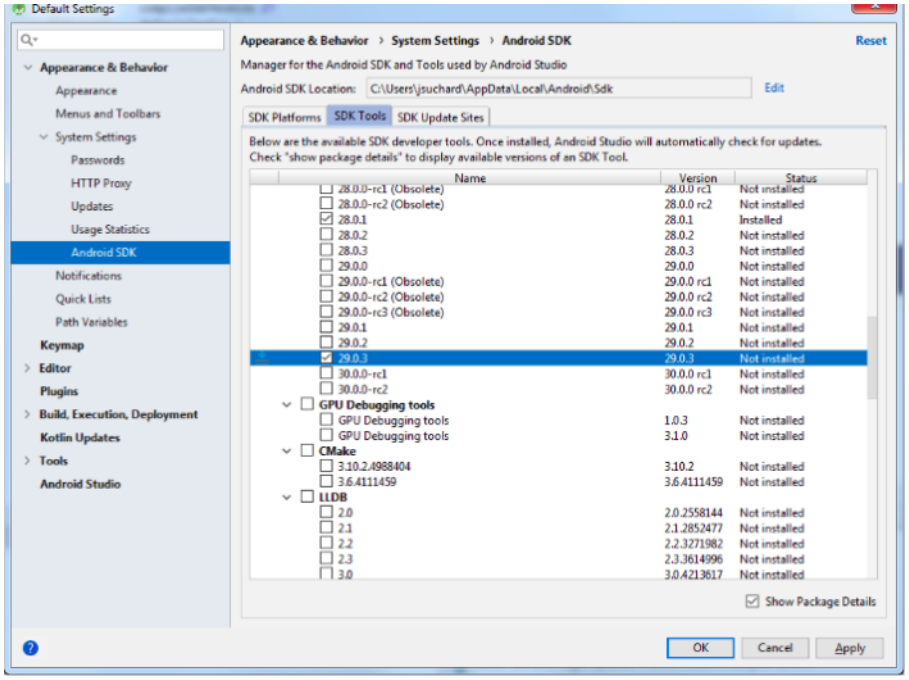
Selezionare File e selezionare setting(impostazioni). Nella barra di ricerca scrivere SDK e successivamente selezionare android SDK. Cliccare la spunta du android .0 marshmallow e cliccare su apply.



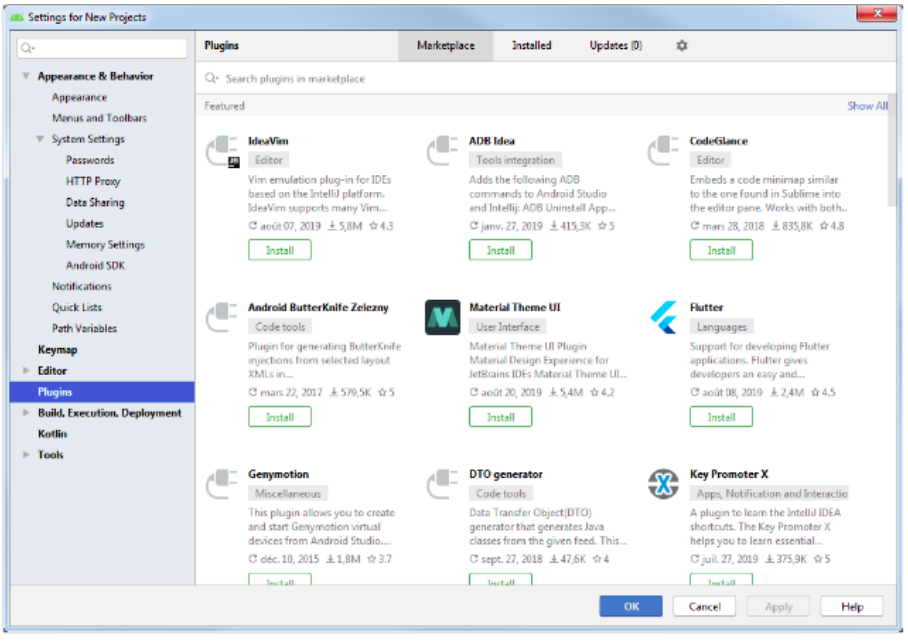
Installato il tutto tornate su android SDK e cliccate su Show Package Details in basso a destra. seleziona android SDK Platform io ho usato la versione 29 ed infine apply.

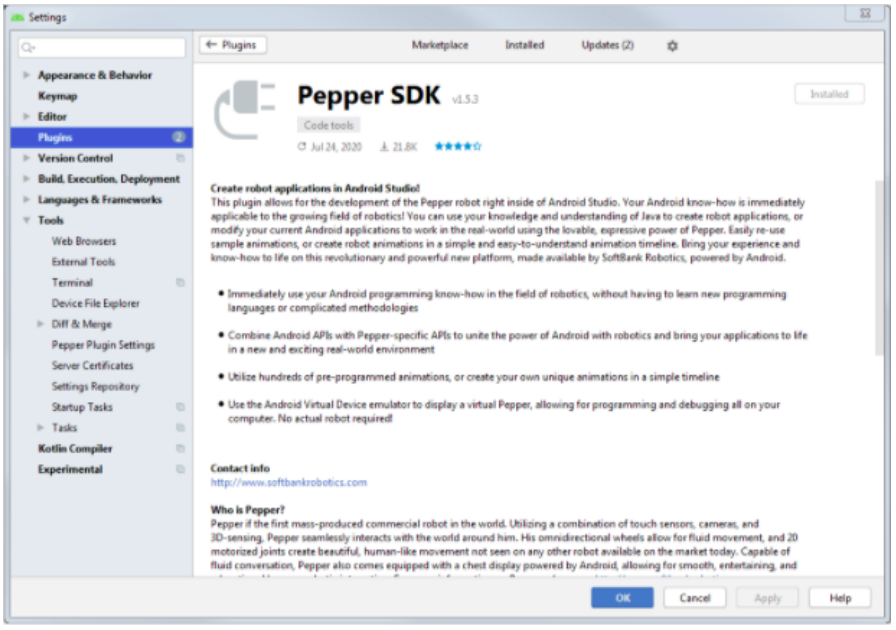


Fare clic sulla scheda Strumenti SDK, selezionare Mostra dettagli pacchetto e selezionare Strumenti di compilazione Android SDK(ndroid SDK Build-Tools) con la stessa versione, quindi fare clic sul pulsante Applica(apply).



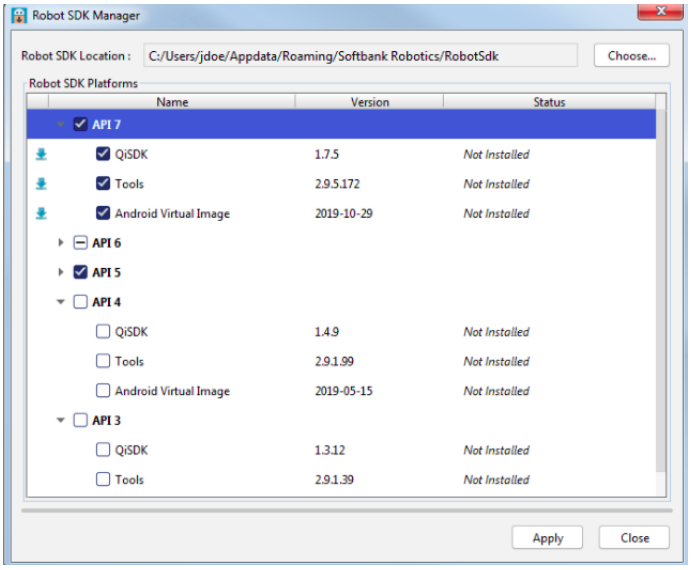
Ora tornate su file selezionate setting(impostazioni) nel menù cercate plugin e con la barra di ricerca trovate ed installate lìestensione Pepper SDk




Settings
Plugins
Marketplace
Installed
Updates (2)
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Web Browsers
External Tools
Terminal
Device File Explorer
Diff & Merge
Pepper Plugin Settings
Server Certificates
Settings Repository
Startup Tasks
Tasks
Kotlin Compiler
Experimental
Pepper SDK
v1.5.3
Installed
Code tools
Jul 24, 2020
21.8K
Create robot applications in Android Studio!
This plugin allows for the development of the Pepper robot right inside of Android Studio. Your Android know-how is immediately applicable to the growing field of robotics! You can use your knowledge and understanding of Java to create robot applications, or modify your current Android applications to work in the real-world using the lovable, expressive power of Pepper. Easily re-use sample animations, or create robot animations in a simple and easy-to-understand animation timeline. Bring your experience and know-how to life on this revolutionary and powerful new platform, made available by SoftBank Robotics, powered by Android.
Immediately use your Android programming know-how in the field of robotics, without having to learn new programming languages or complicated methodologies
Combine Android APIs with Pepper-specific APIs to unite the power of Android with robotics and bring your applications to life in a new and exciting real-world environment
Utilize hundreds of pre-programmed animations, or create your own unique animations in a simple timeline
Use the Android Virtual Device emulator to display a virtual Pepper, allowing for programming and debugging all on your computer. No actual robot required!
Contact info
http://www.softbankrobotics.com
Who is Pepper?
Pepper if the first mass-produced commercial robot in the world. Utilizing a combination of touch sensors, cameras, and 3D-sensing, Pepper seamlessly interacts with the world around him. His omnidirectional wheels allow for fluid movement, and 20 motorized joints create beautiful, human-like movement not seen on any other robot available on the market today. Capable of fluid conversation, Pepper also comes equipped with a chest display powered by Android, allowing for smooth, entertaining, and
OK
Cancel
Apply
Help

Ottieni Robot SDK e tools

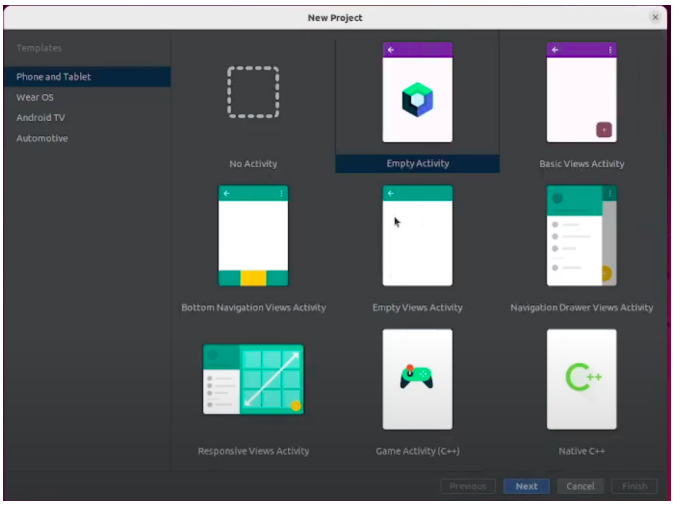
Seleziona Tools > Pepper SDK > Robot SDK Manager. Controlla la versione dell'API di destinazione in cui vuoi sviluppare l'applicazione robot e fai clic sul pulsante Applica.



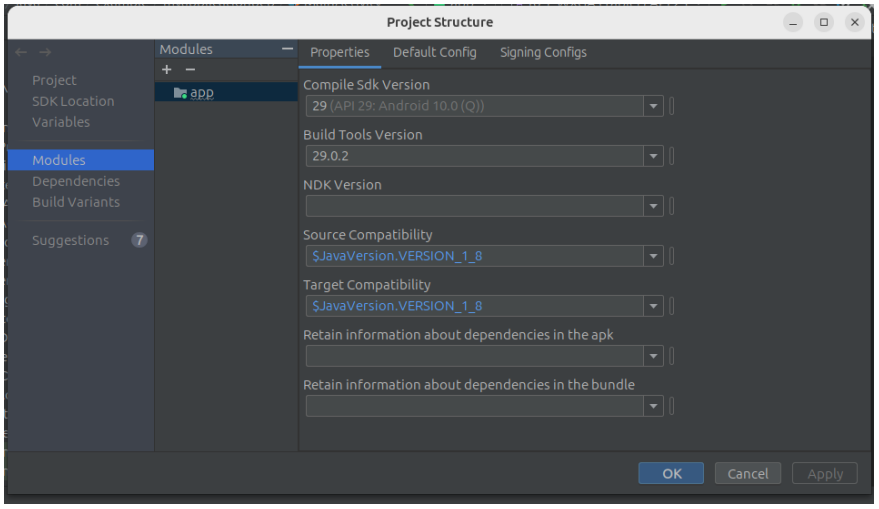
Applicazione Realizzata

Crea un Applicazione robot

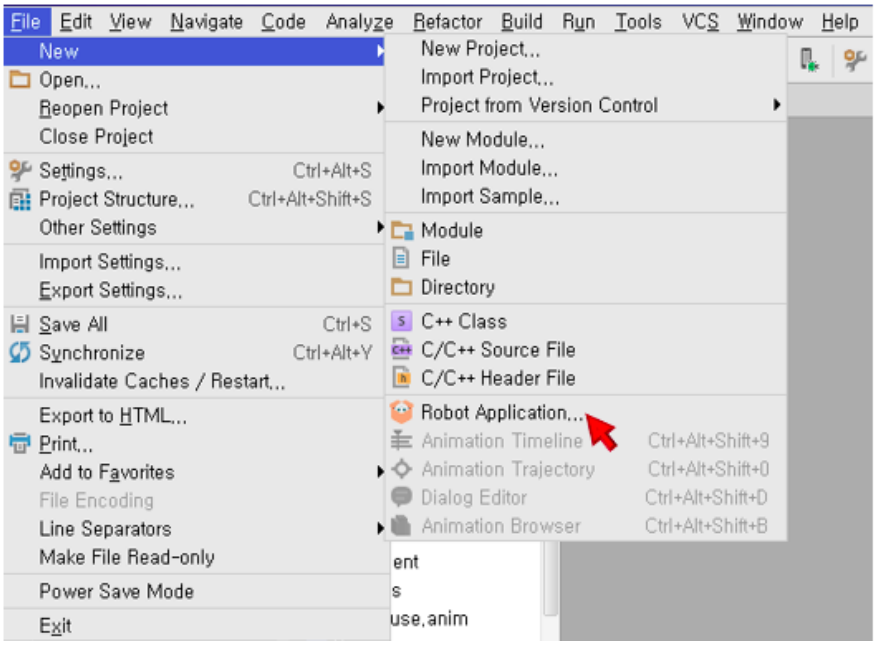
Crea progetto: Seleziona empty activity e crea un nuovo progetto o us quello già esistente.



aperto il progetto seleziona File > Project Structure... > Module e seleziona le API compatibili.

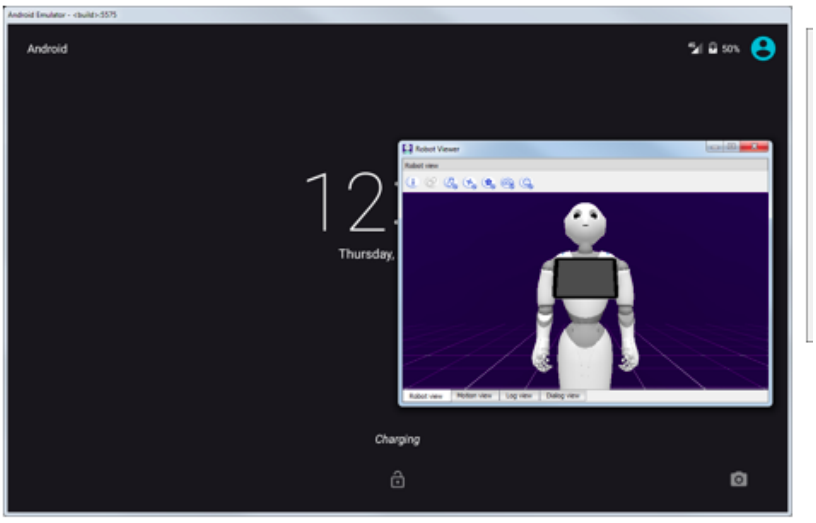


selezione File > New > Robot Application.



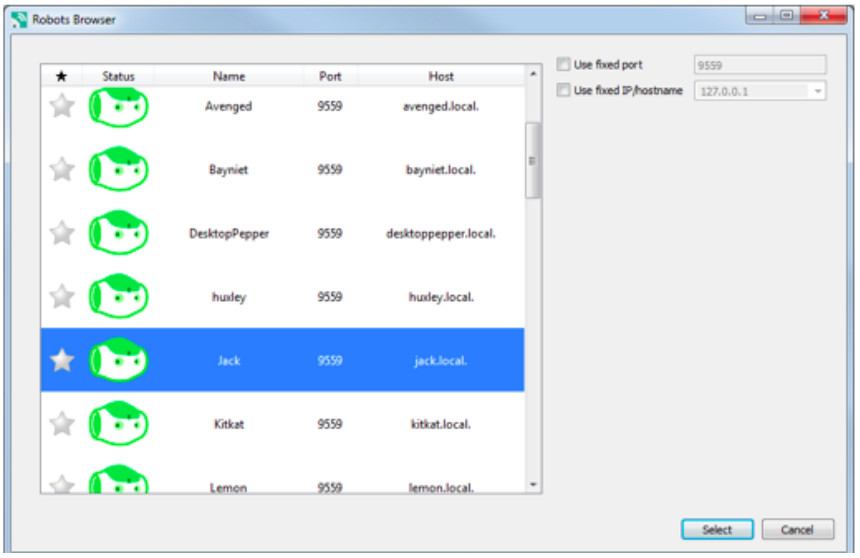
Seleziona la versione minima di Robot SDK e il modulo da robotizzare. Quindi fare clic su OK. Il tuo progetto è ora un'applicazione robotica! Di conseguenza, dovrebbero essere visualizzate le modifiche seguenti: Il file robotsdk.xml è stato aggiunto alla directory assets/robot. Contiene la versione minima di Robot SDK per il tuo progetto (quella che hai selezionato). Nuovi strumenti sono disponibili in Tools > Pepper SDK. Per lanciare l'emulatore del robot Tools > Pepper SDK > Emulator Usa l'emulatore per

avviare un nuovo emulatore di robot e connettiti a un vero robot. La dipendenza QiSDK è stata aggiunta al file build.gradle del modulo. Ti fornisce tutte le funzionalità per interagire con Pepper. Al tuo AndroidManifest.xml è stato aggiunto un tag uses-feature. Indica che l'applicazione utilizza Pepper e sarà disponibile solo per il tablet robot.



Connettersi ad un robot reale

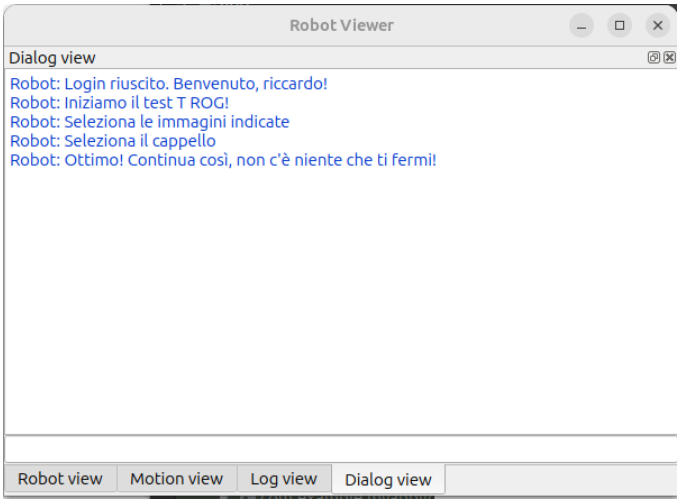
Choose Tools > Pepper SDK > Connect ed il Robots Browser apparirà. Seleziona uno dei robot nell'elenco. Se il robot non viene visualizzato, puoi anche selezionare e completare i campi Usa porta di correzione e Usa IP/nome host fisso. In questo caso, utilizzare 9559 come porta fissa.


Robots Browser
Status
Name
Port
Host
Avenged
9559
avenged.local.
Bayniet
9559
bayniet.local.
DesktopPepper
9559
desktoppepper.local.
huxley
9559
huxley.local.
Jack
9559
jack.local.
Kitkat
9559
kitkat.local.
Lemon
9559
lemon.local.
Use fixed port
9559
Use fixed IP/hostname
127.0.0.1
Select
Cancel

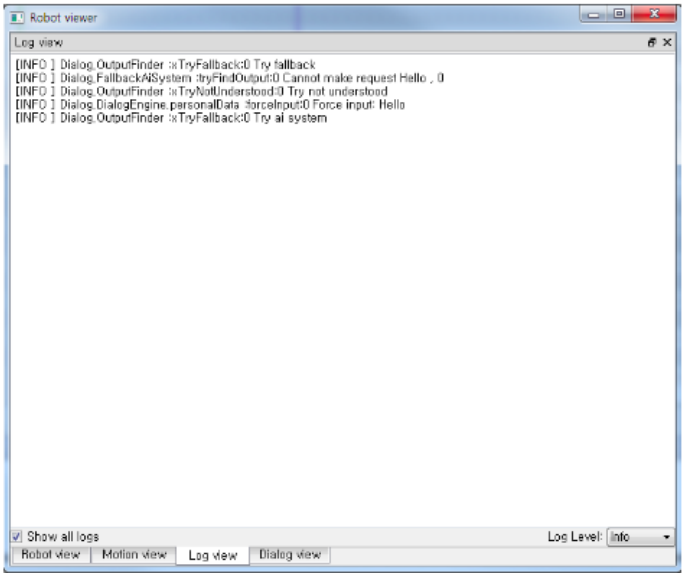
Robot Viewer

Il visualizzatore Robot raccoglie diversi strumenti che consentono di monitorare il robot a cui è collegato il plugin: La vista del robot mostra una vista 3D del robot, Motion View consente di controllare i movimenti del robot, La vista dialogo consente di guardare gli output dei dialoghi e di inserire gli input dei dialoghi.

Dialog view

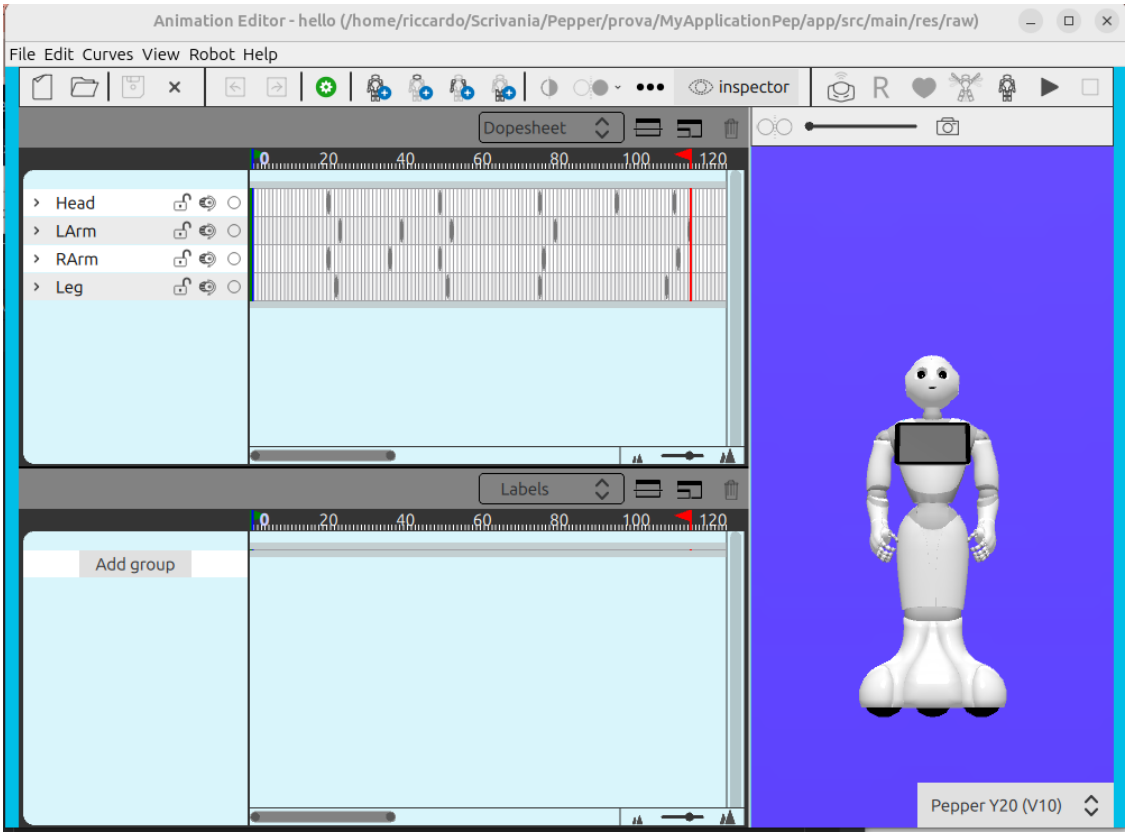


Log view


Robot viewer
Log view
[INFO] Dialog.OutputFinder :tryFallback:0 Try fallback
[INFO] Dialog.FallbackAiSystem :tryFindOutput:0 Cannot make request Hello , 0
[INFO] Dialog.OutputFinder :tryNotUnderstood:0 Try not understood
[INFO] Dialog.DialogEngine.personalData :forceInput:0 Force input: Hello
[INFO] Dialog.OutputFinder :tryFallback:0 Try ai system
Show all logs
Log Level:
Info
Robot view
Motion view
Log view
Dialog view

Animation Editor

Seleziona File > New > Animation Timeline. Dai un nome all'animazione e scegli clicca su create. Lo schermo mostrerà l'Animation Editor. Le animazione Timeline avranno come estensione .qianim file.



La vista Dopesheet mostra le chiavi come punti.

Una chiave assegna un valore a un attuatore. Un tasto Riepilogo raggruppa tutti i tasti definiti per un arto.

La vista Curve mostra le chiavi come punti su una curva.

Una curva rappresenta la variazione del valore assegnato al relativo attuatore.

Mostra l'interpolazione tra due chiavi.

I fotogrammi sono numerati sull'asse della sequenza temporale.

La bandiera verde indica l'inizio dell'animazione, La bandiera rossa indica la fine La linea blu identifica il Frame selezionato

li attuatori sono elencati sul lato sinistro delle viste.

Sono raggruppati in base agli arti: testa , braccio sinistro , braccio destro , gamba .

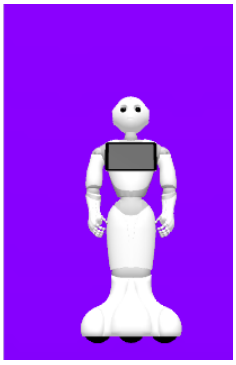
Per entrambe le visualizzazioni:

Gli arti possono essere espansi o compressi. Lock curve impedisce modifiche indesiderate su una curva. Quando attivato, il colore della curva cambia in grigio e le chiavi non possono essere modificate. La curva muta disattiva una curva quando l'animazione viene riprodotta. L'opzione AutoKey consente di creare chiavi modificando gli attuatori nella vista 3D del robot . Solo per la vista Curva :

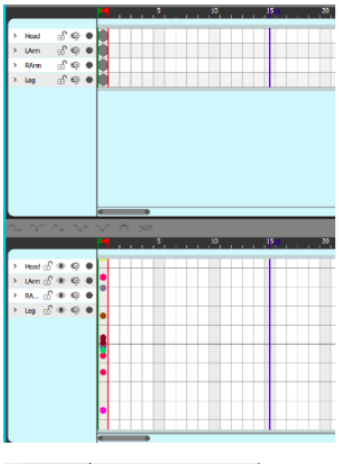
Il colore di ogni curva è visualizzato accanto al nome dell'attuatore. Cliccaci sopra per scegliere un altro colore. Nascondi curva consente di semplificare la visualizzazione della curva per concentrarsi su un attuatore specifico.

Ciao mondo

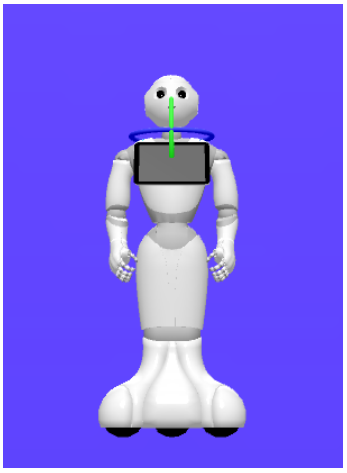
Selezionare Robot > Posture > Stand . La postura corretta è questa



Selezionare Edit > Create whole body key(s). Seleziona un altro Frame, ad esempio il Frame 15 . La linea blu mostra il Frame selezionato.



Nella vista Robot 3D , seleziona la testa. Compaiono cursori e rotelle che consentono di muovere la testa.



Trascinare la rotella verde o il cursore HeadPitch per abbassare la testa. La chiave principale viene creata automaticamente nel Frame 15.

Seleziona un altro Frame, ad esempio il Frame 30 . Trascinare la rotella verde o il cursore HeadPitch per sollevare la testa.

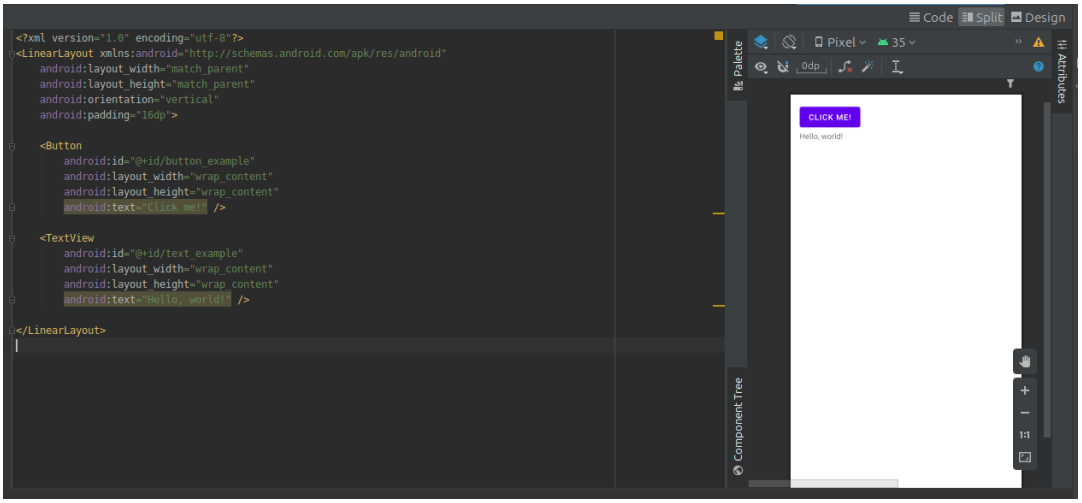
La chiave di testa viene creata automaticamente nel fotogramma 30.

Sull'asse Frame , seleziona le ultime 2 Chiavi. Trascinare la barra blu per spostare la selezione verso destra. questo permetterà di rendere il movimento meno brusco.

Animazione è pronta.

Creare un'Interfaccia con XML

In Android, l'interfaccia utente (UI) viene generalmente definita utilizzando XML (eXtensible Markup Language). L'XML è un linguaggio di markup che consente di strutturare i dati in modo leggibile sia per le macchine che per gli esseri umani. In Android, viene utilizzato per definire il layout dell'applicazione, ovvero come gli elementi visivi (come pulsanti, etichette, campi di testo, ecc.) devono essere disposti nella schermata dell'app. Android Studio, l'IDE ufficiale per lo sviluppo di applicazioni Android, fornisce un potente strumento di Design View che consente di creare e visualizzare l'interfaccia utente in modo intuitivo. All'interno di Android Studio, possiamo passare tra la Code View (dove scriviamo il codice XML) e la Design View (dove vediamo un'anteprima dell'interfaccia utente). La Design View di Android Studio offre anche un'interfaccia drag-and-drop che permette di posizionare elementi UI direttamente sulla schermata, come bottoni, etichette, caselle di testo, senza la necessità di scrivere manualmente il codice XML. Quando si spostano gli elementi nell'editor visivo, Android Studio aggiorna automaticamente il codice XML, il che rende il processo molto più rapido e user-friendly.



Nel codice sopra, un LinearLayout è il contenitore principale che contiene due componenti: un Button e un TextView. Ogni componente ha un id che può essere utilizzato per fare riferimento ad esso nel codice Kotlin/java e modificarne le proprietà.

Gestire gli Eventi in Kotlin/java

In Android, un evento è una reazione a un'azione compiuta dall'utente, come un tocco su un pulsante, uno swipe o un inserimento di testo. Gli eventi più comuni in Android sono quelli legati ai click (ad esempio, il clic su un bottone) o all'input (come la scrittura in una casella di testo).Un evento si verifica quando un utente interagisce con un elemento dell'interfaccia utente (UI). Ad esempio, un utente può cliccare un bottone, e questo è un "evento di click". Gli eventi vengono gestiti attraverso listener (ascoltatori) che monitorano e reagiscono all'azione dell'utente.

Android fornisce diversi tipi di listener per diversi tipi di eventi ad esempio:

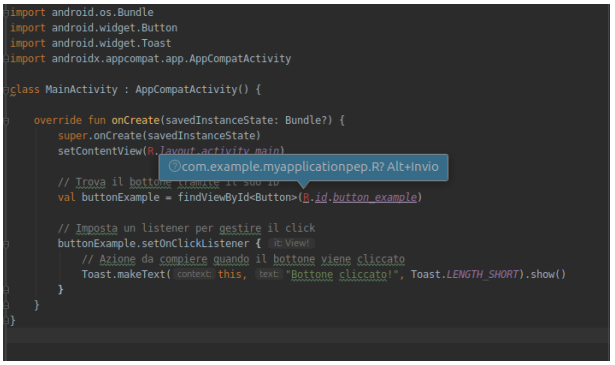
- OnClickListener: gestisce gli eventi di clic su un bottone.
- OnTouchListener: gestisce gli eventi di tocco (come swipe o tap).
- OnFocusChangeListener: gestisce quando un elemento perde o guadagna il focus.

Per gestire gli eventi in Android, associamo un listener agli elementi dell'interfaccia. Ad esempio, se vogliamo gestire un evento di click su un bottone, possiamo farlo nel seguente modo:

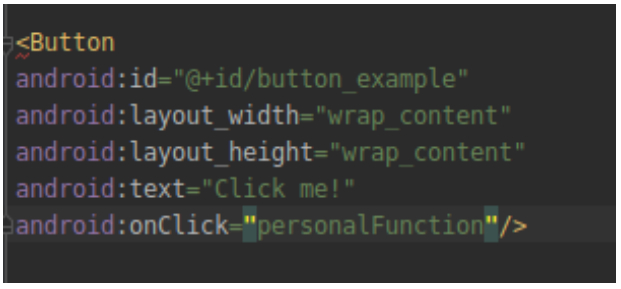
Codice XML per il layout (come visto prima):

```
<Button
android:id="@+id/button_example"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Click me!" />
```

Nel file Kotlin associato all'attività (Activity) o al fragment, possiamo usare il seguente codice per gestire l'evento di click del bottone. Si noti che in caso non è stato importata la libreria android fornisce in automatico l'inclusione.



Un altro modo semplice è aggiungere nell xml <Android android:onClick="Function"> e definire la funzione nell'activity responsabile.



AndroidManifest.xml

Il file AndroidManifest.xml è un componente essenziale di qualsiasi applicazione Android. Si trova nella cartella principale del progetto e svolge il ruolo di "dichiarazione" dell'app, specificando informazioni fondamentali per il suo funzionamento. Il file AndroidManifest.xml è necessario per:

* Dichiarare le attività (Activity): Ogni interfaccia utente dell'app è associata a una Activity, e queste devono essere registrate nel Manifest affinché possano essere riconosciute dal sistema operativo.
* Specificare i permessi: Se l'app ha bisogno di accedere a risorse particolari come Internet, la fotocamera o il GPS, deve richiedere l'autorizzazione dell'utente attraverso il Manifest.
* Definire intent-filter: Indicano come l'app può essere avviata da altre applicazioni o dal sistema.
* Impostare l'icona e il nome dell'app: Il Manifest contiene riferimenti all'icona dell'app e al suo nome visualizzato.
* Specificare la versione dell'app e la compatibilità: Il file definisce la versione minima di Android supportata e la versione target per cui l'app è ottimizzata.

Emulatore Android

L'emulatore Android è uno strumento integrato in Android Studio che simula dispositivi Android, permettendo agli sviluppatori di testare le loro applicazioni su vari dispositivi e configurazioni senza la necessità di utilizzare dispositivi fisici. Grazie alla sua capacità di emulare diversi tipi di dispositivi, versioni del sistema operativo e risoluzioni dello schermo. L'emulatore di Android Studio offre una serie di funzionalità che lo rendono un potente strumento di test:

- Simulazione di diversi dispositivi: È possibile emulare diversi modelli di dispositivi Android con specifiche caratteristiche hardware, come schermi di dimensioni variabili, risoluzioni, e versioni del sistema operativo.
- Simulazione di sensori: L'emulatore permette di simulare anche vari sensori, come GPS, accelerometro e sensori di orientamento, che possono essere utili per testare applicazioni che dipendono da questi.
- Test delle prestazioni: L'emulatore fornisce una piattaforma per monitorare il comportamento delle app, tra cui il consumo della memoria e delle risorse di sistema, essenziale per ottimizzare la performance dell'app.
- Debugging in tempo reale: Durante il processo di sviluppo, l'emulatore permette di eseguire il debug dell'applicazione, visualizzare i log in tempo reale e testare il comportamento delle funzionalità implementate, senza la necessità di dispositivi fisici.