UNIVERSITÀ
DEGLI STUDI
DI PADOVA

SISTEMI EMBEDDED

ANDROID

(Versione 22/06/2018)

Stesura a cura di:
Stefano Ivancich

Questa dispensa è scritta da studenti senza alcuna intenzione di sostituire i materiali universitari. Essa costituisce uno strumento utile allo studio della materia ma non garantisce una preparazione altrettanto esaustiva e completa quanto il materiale consigliato dall'Università.
Lo scopo di questo documento è quello di riassumere i concetti fondamentali degli appunti presi durante la lezione, riscritti, corretti e completati facendo riferimento alle slide per poter essere utilizzato come un manuale "pratico e veloce" da consultare. Non sono presenti esempi e spiegazioni dettagliate, per questi si rimanda ai testi citati e alle slide.

Se trovi errori ti preghiamo di segnalarli qui:
www.stefanoivancich.com
ivancich.stefano.1@gmail.com
Il documento verrà aggiornato al più presto.

INDICE

1.Android Basics

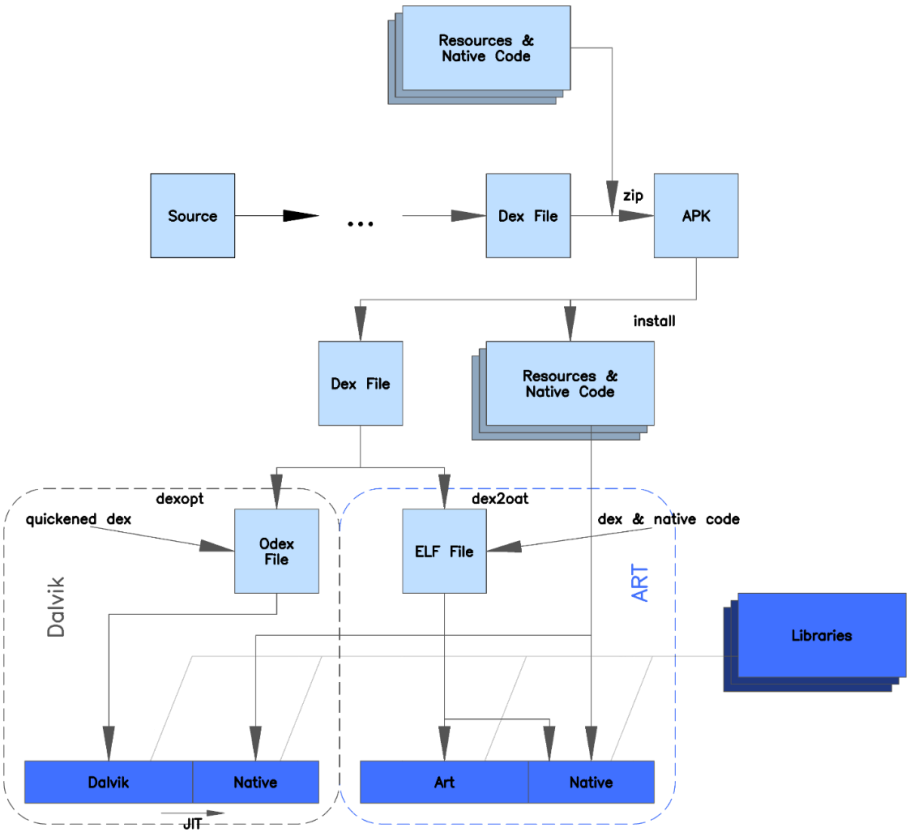
1.1. Sistemi Embedded

Sistema embedded:

- Svolge un insieme ristretto di funzioni (router, bilancia, controllo industriale, …).
- Ha risorse limitate (RAM, CPU, storage, …)
- Ha vincoli sulle tempistiche di risposta.
- Di solito sono programmati con linguaggi di basso livello per avere più controllo su tempo di risposta e utilizzo delle risorse.

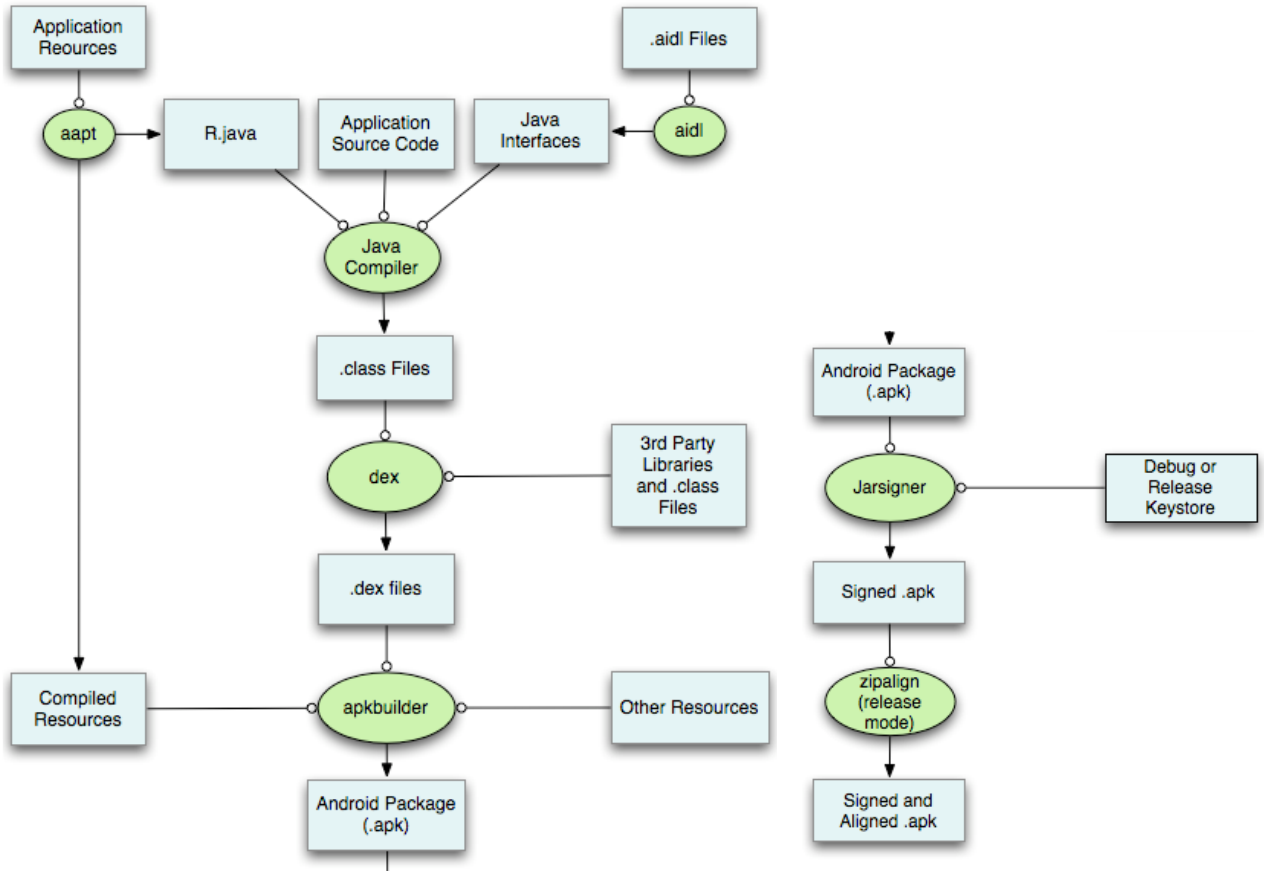
Libreria: collezione di risorse (classi) usate per sviluppare software.
Framework: collezione di librerie organizzate per fornire una specifica funzionalità. Impone un modello di programmazione.
Piattaforma: collezione di framework che permette di eseguire il codice. Richiede un particolare sistema operativo, un insieme di linguaggi di programmazione e librerie run-time. Può includere un architettura hardware.
Ecosistema: piattaforma + community che sviluppa hardware e software per essa (documentazione, manualistica online, …).

Android runtime: Le app vengono sviluppate in Java, poi compilate in bytecode Java, e poi di nuovo in un formato bytecode proprietario (File .DEX) che viene eseguito dalla Virtual Machine di Android (Dalvik oppure ART).



APK: file ZIP con estensione cambiata, contiene cartelle con varie risorse (.DEX, codice nativo, immagini, suoni, stringhe di testo,…). Sono “firmati” in modo da individuare il creatore, la firma delle app sul play store sono certificate da google.

Processo di Build:



Ogni app viene installata in una propria cartella, i cui privilegi appartengono all'app stessa.
Ogni app gira in una copia separata della VM in modo tale che se un app si blocca, non blocca l'intero sistema.
Le app comunicano tra loro tramite un component provider.

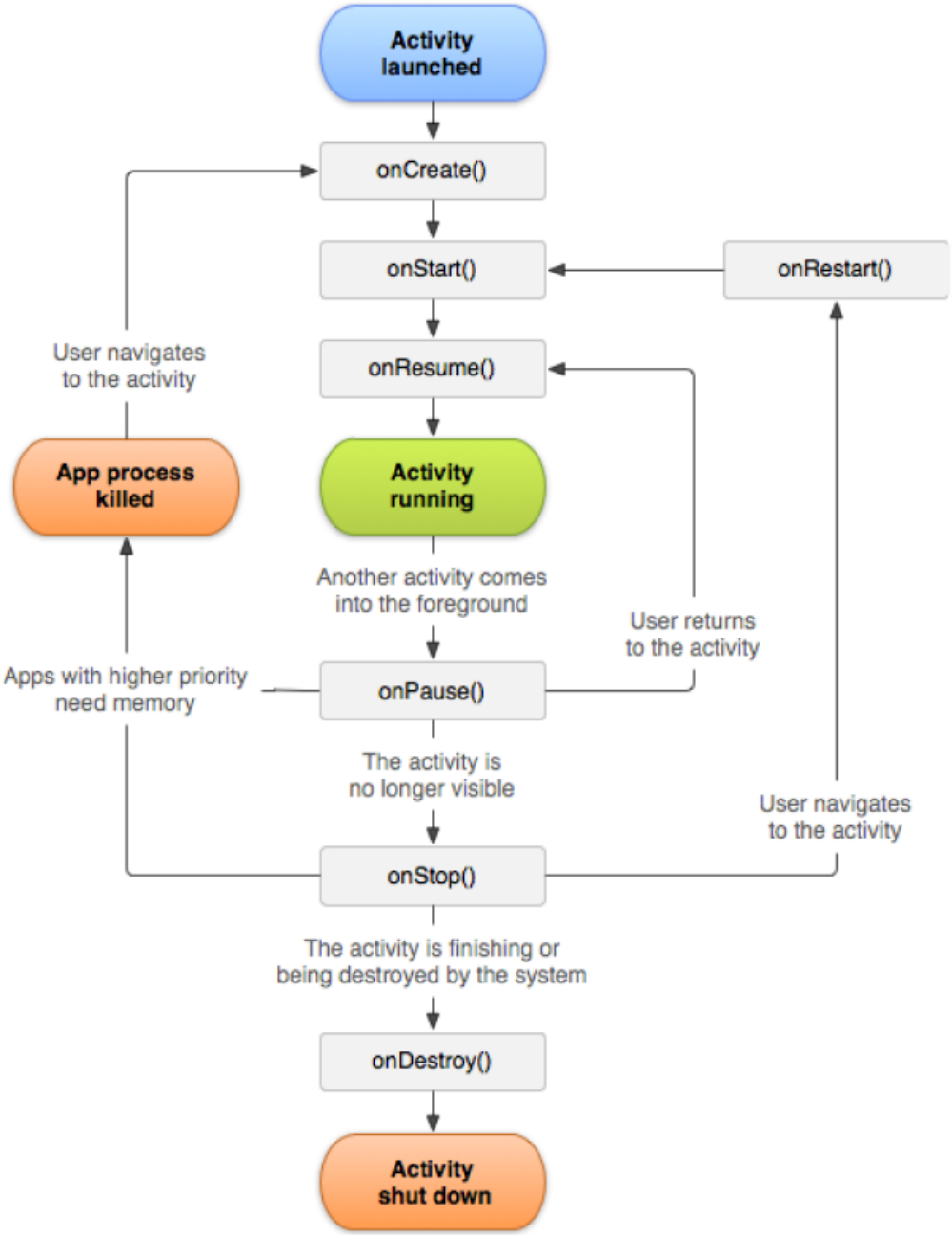
Componenti di un App:

- **Activity**: Singola pagina UI.
- **Service**: svolto in background.
- **Content provider**: incapsula i dati che necessitano di essere condivisi con altre app.
- **Broadcast Receiver**: risponde agli eventi esterni (es batteria scarica).

Ogni componente ha una sua classe.

Intent: messaggio asincrono che richiede un azione. Può richiedere l'utilizzo di uno specifico componente.
Implicito: richiede un servizio in generale, che poi il sistema o l'utente sceglieranno.
Esplicito: richiede uno specifico servizio (app).

Ciclo di vita delle activity:



Le activity vengono gestite in stack.
Task: insieme di activity correlate, vengono messe o tolte dallo stack insieme.

AndroidManifest.xml: dichiara i permessi, requisiti hardware, le componenti dell'app.
Strings.xml: contiene le stringhe.
Build.Gradle: non fa parte dell'apk ma contiene informazioni per compilare, eseguire e la versione di android.

1.2. Android NDK (Native Development Kit)

Serve per compilare in codice macchina anzi che in bytecode, si usa il C/C++.
Contiene dei cross-compilatori, uno per ogni architettura hardware supportata.

Approcci di sviluppo:

- L'applicazione è in gran parte scritta in Java, i pochi metodi scritti In C/C++ sono acceduti tramite la JNI (Java Native Interface).
- Activity native: intere Activity sono implementate in C/C++

Compilare codice C/C++ in Android Studio:

- ndk-build
- CMake (Default di Android Studio)

1.2.1. JNI (Java Native Interface)

Interfaccia java che permette di: chiamare codice C/C++ da Java, chiamare metodi Java da C/C++, mappare i data types.

Per chiamare codice C/C++ da Java:

- Aggiungere la parola "native" prima del nome del metodo.
- Le librerie sono caricate tramite il comando System.loadLibrary("nome");

L'implementazione di un metodo in C/C++:

- Come nome: Java_{package_and_classname}_{function_name}
- Come parametri: (JNIEnv *env, jobject obj, eventuali parametri della funzione)

```
package pkg;

class foo
{
    native double bar(int i, String s);

    static
    {
        System.loadLibrary("my_lib");
    }

    ...
}
```

```
jdouble Java_pkg_foo_bar(JNIEnv *env, // ptr to JNI interface
                         jobject obj, // "this" pointer
                         jint i,      // first "real" parameter
                         jstring s)   // second "real" parameter
{
    ... /* Method implementation */
}
```

Java type	C/C++ Type	Description
boolean	jboolean	8 bit, unsigned
char	jchar	16 bit, unsigned
int	jint	32 bit, signed
String	jstring	Different encodings
...	...	...

1.2.2. NDK-BUILD

Android.mk: contiene la lista dei file da compilare.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)
```

LOCAL_PATH: locazione dei file sorgente.
LOCAL_MODULE: nome del modulo (libreria).
LOCAL_SRC_FILES: file necessari per costruire il modulo.

Application.mk: opzionale. Specifica ulteriori informazioni sulla compilazione, ad esempio l'architettura hardware.

```
# The ARMv7 is significantly faster
# due to the use of the hardware FPU
APP_ABI := armeabi armeabi-v7a

APP_PLATFORM := android-8
```

APP_ABI: specifica per quali architetture compilare.
APP_PLATFORM: target API level.

Comandi di shell:

- *<ndk>/ndk-build*: genera i moduli.
- *<ndk>/ndk-build NDK_DEBUG=1*: genera i moduli e aggiunge i simboli di debug.
- *<ndk>/ndk-build clean*: cancella tutti i moduli generati.

Come usare ndk-build:

- Mettere i file sorgenti in: *<mod>/jni/…*
- Creare: *<mod>/jni/Android.mk*
- Opzionale: creare *<mod>/jni/Application.mk*
- Compilare lanciando il comando: *ndk-build*

1.2.3. CMake

CMakeLists.txt: lista di comandi

- cmake_minimum_required(…): Versione minima di cmar richiesta.
- add_library(…)
- find_library(…): per le librerie precompilate
- target_link_libraries(…)

Imposta alcune variabili in build.gradle

2. Design Patterns

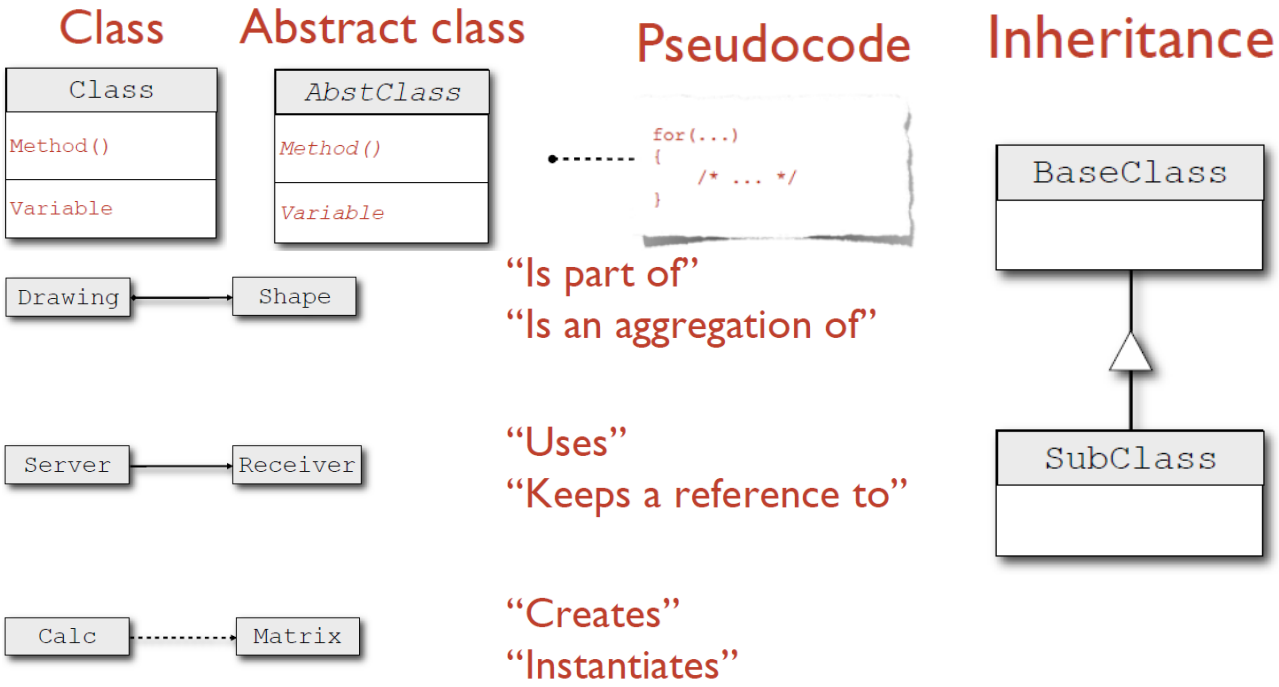
Sono una soluzione ad un problema ricorrente.
Elementi:

- Nome
- Problema
- Soluzione
- Conseguenze

Classificazioni:

- **Comportamentali**: come si scambiano le informazioni tra le entità software.
- **Strutturali**
- **Creazionali**: come creare un entità software

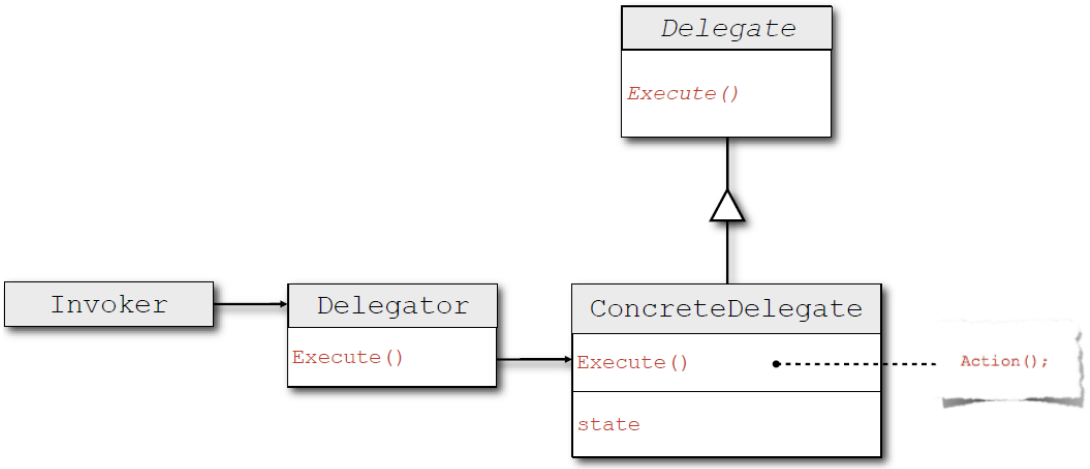
2.1. Notazione



2.2. Pattern Comportamentali

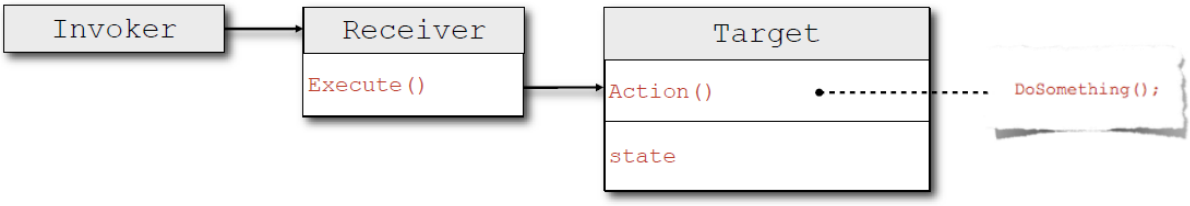
2.2.1. Delegation

Problema: come modificare oggetti complessi senza creare delle sottoclassi.
Soluzione: il codice da aggiungere viene messo in una altra classe (delegata).



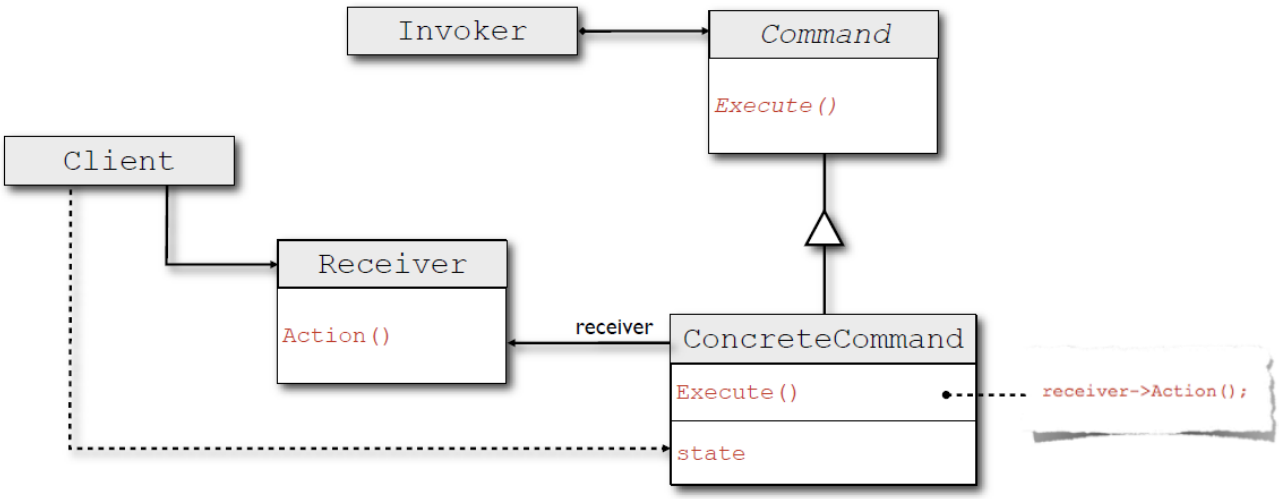
2.2.2. Target-Action

Problema: come notificare un applicazione quando c'è un evento sull'interfaccia grafica.
Soluzione: l'oggetto UI invia un messaggio (l'azione) ad un altro oggetto (target).



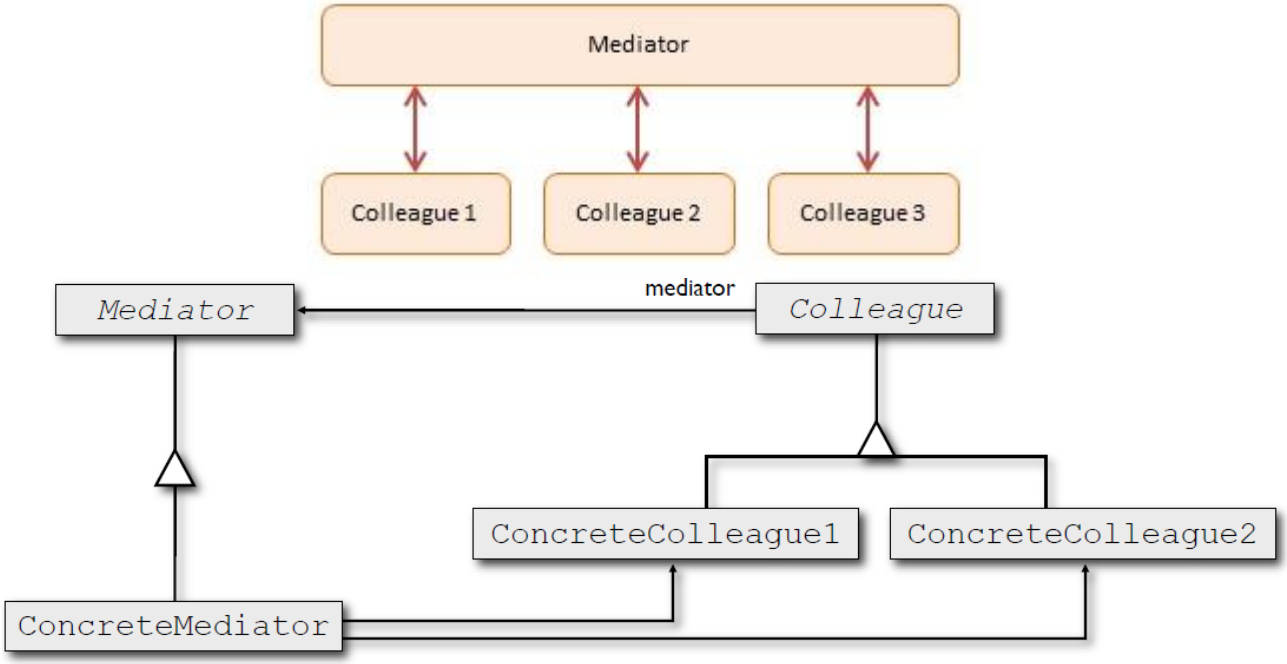
2.2.3. Command

Problema: come inviare richieste agli oggetti senza sapere i dettagli dell'operazione e il destinatario della richiesta.
Soluzione: trasformare la richiesta in un oggetto e una classe astratta Command dichiara l'interfaccia per eseguire le operazioni.



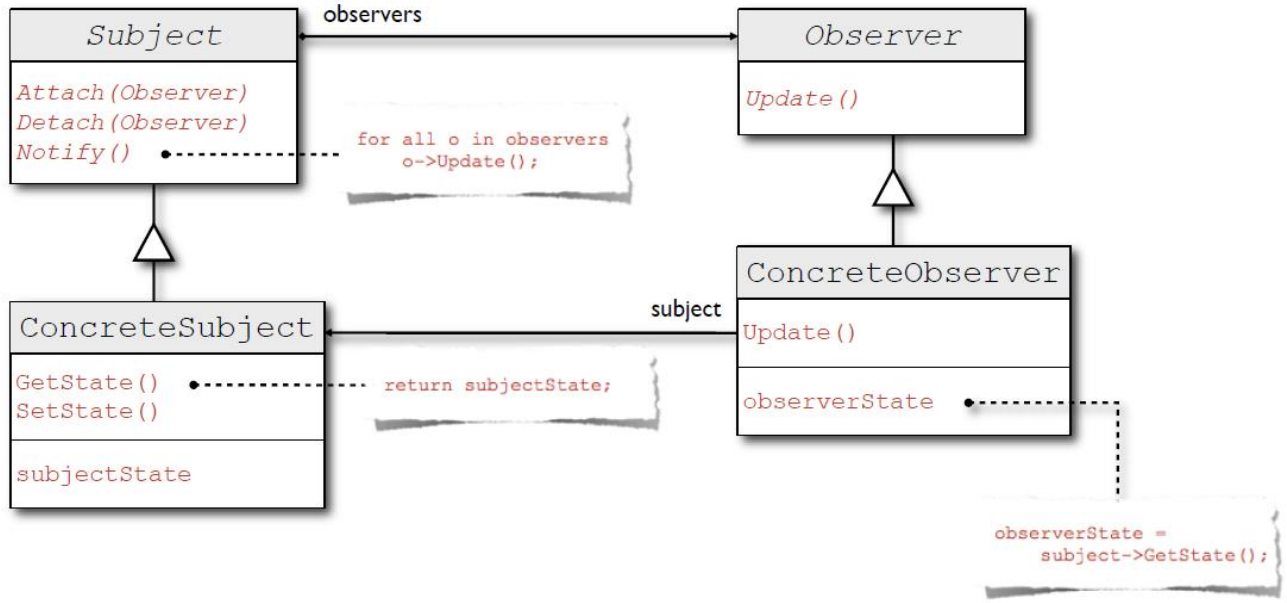
2.2.4. Mediator

Problema: come gestire la comunicazione tra oggetti.
Soluzione: centralizzare le comunicazioni in un oggetto separato (Mediator).
Conseguenze: interazioni molti a molti diventano uno a molti.



2.2.5. Observer

Problema: come mantenere la consistenza tra gli oggetti che hanno dati in comune.
Soluzione: salvare le informazioni in comune in un altro oggetto (subject). Gli oggetti che vogliono salvare le informazioni sono Observer che vengono notificati ad ogni cambiamento.
Conseguenze: difficoltà nel valutare le implicazioni dei cambi di stato.

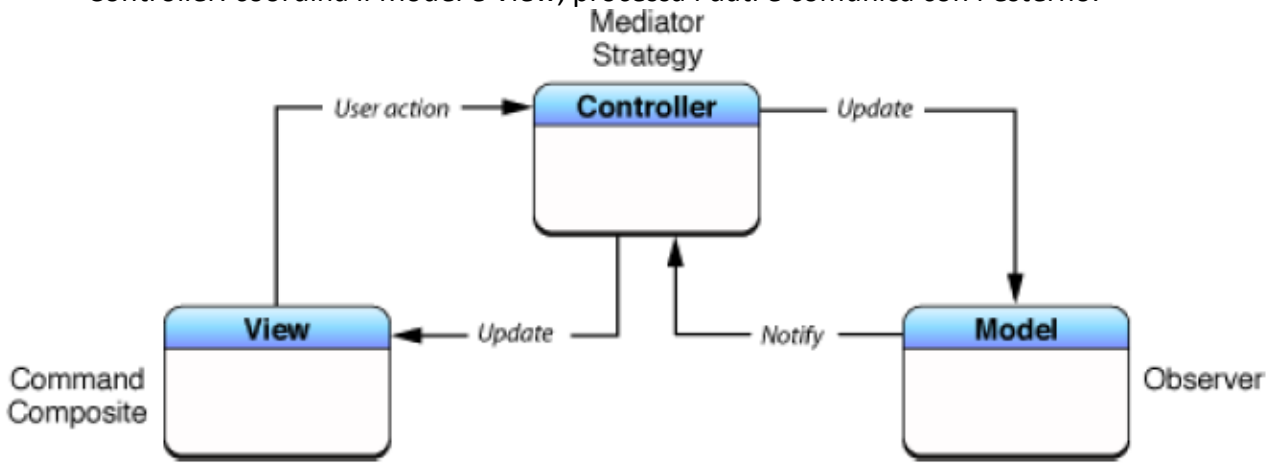


2.3. Pattern Strutturali

2.3.1. Model View Controller

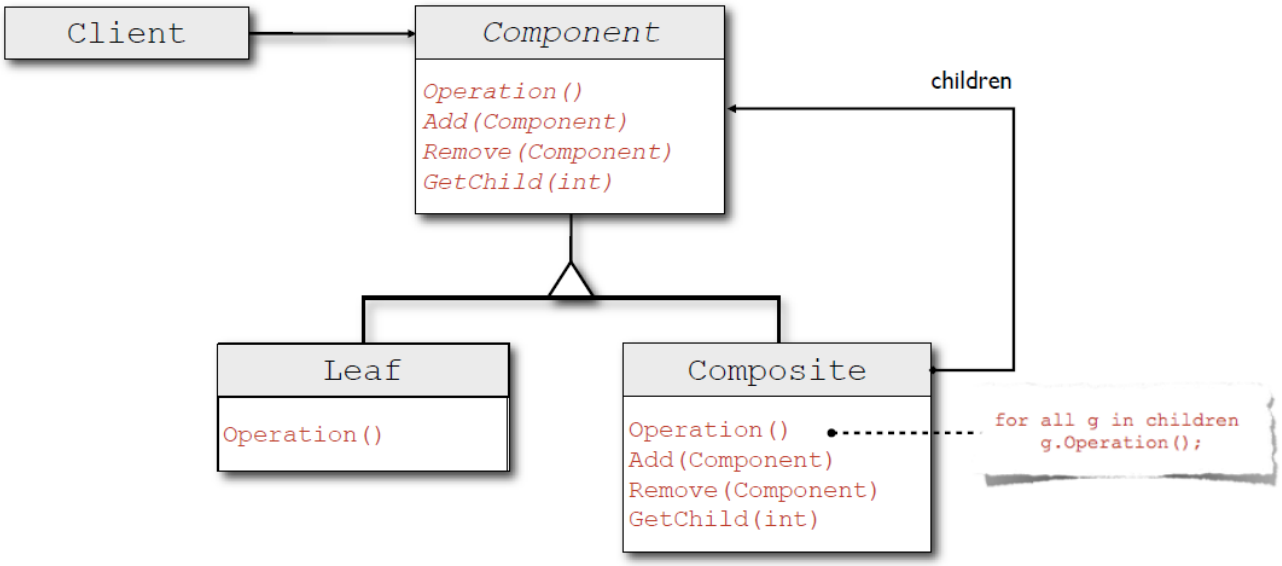
Problema: come sviluppare un applicazione per leggere, modificare, scrivere e visualizzare dati.
Soluzione: dividere l'applicazione in 3 entità:

- Model: contiene i dati
- View: Gestische la UI
- Controller: coordina il Model e View, processa i dati e comunica con l'esterno.



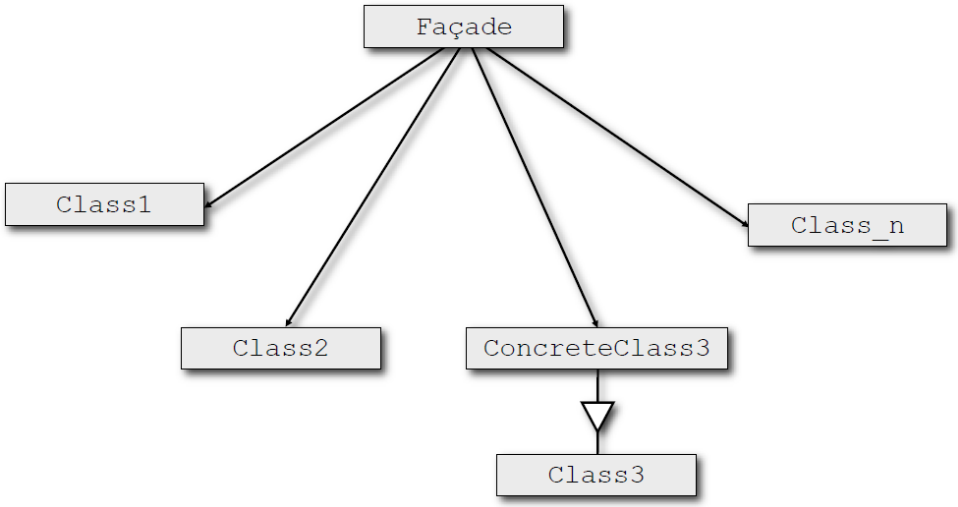
2.3.2. Composite

Problema: come progettare le classi per rappresentare gerarchie.
Soluzione: struttura ad albero.
Conseguenze: la soluzione potrebbe essere troppo generale, ogni figlio può avere solo un padre, problema se si vuole un ordinamento nei figli.



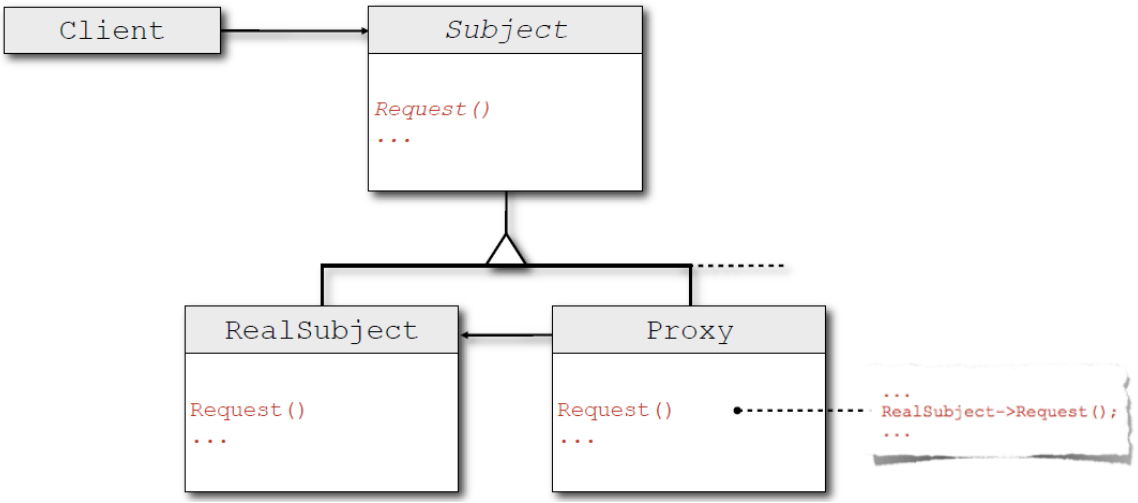
2.3.3. Façade

Problema: come comunicare con tante interfacce esterne.
Soluzione: introdurre un oggetto che si occupa delle comunicazioni.



2.3.4. Proxy

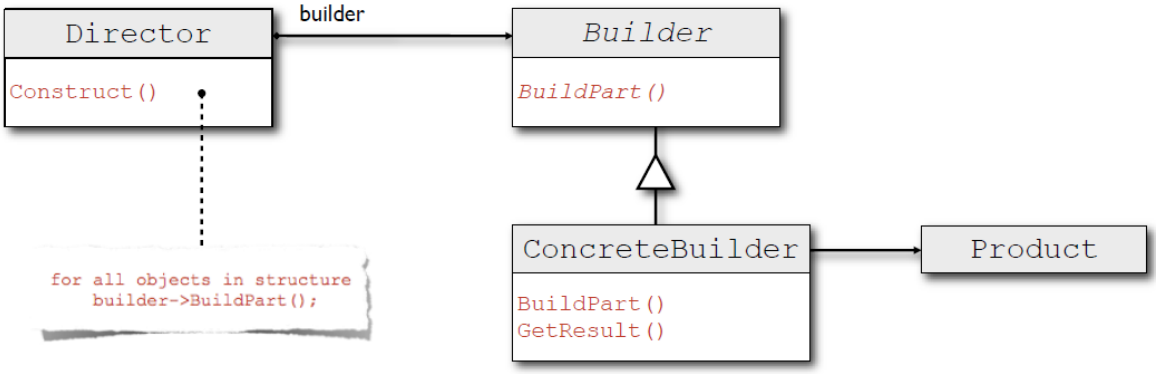
Problema: come far in modo che oggetti complessi che ci mettono tempo per essere istanziati appaiano sempre disponibili.
Soluzione: oggetto proxy che istanzia il vero oggetto solo quando è veramente necessario utilizzarlo.
Conseguenze: rallentamento.



2.4. Pattern creazionali

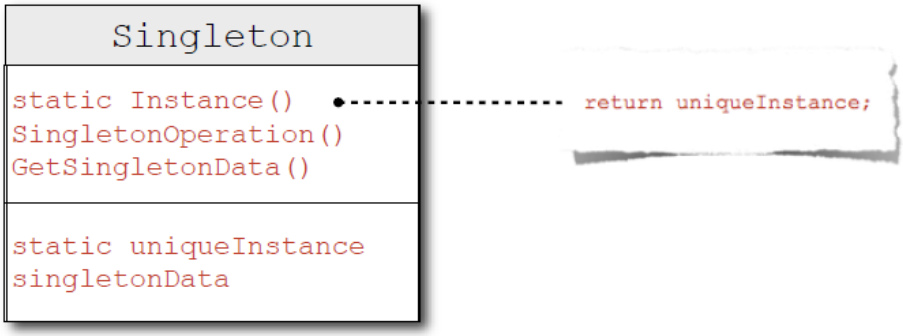
2.4.1. Builder

Problema: come creare molte rappresentazioni diverse di un oggetto.
Soluzione: separare la costruzione dell'oggetto dalla sua rappresentazione. Creare diversi oggetti Builder che opereranno sotto il controllo ei un singolo Director.



2.4.2. Singleton

Problema: come garantire che una classe abbia un numero limitato di istanze.
Soluzione: far in modo che la classe stessa tenga traccia del numero di istanze.



3.Hardware

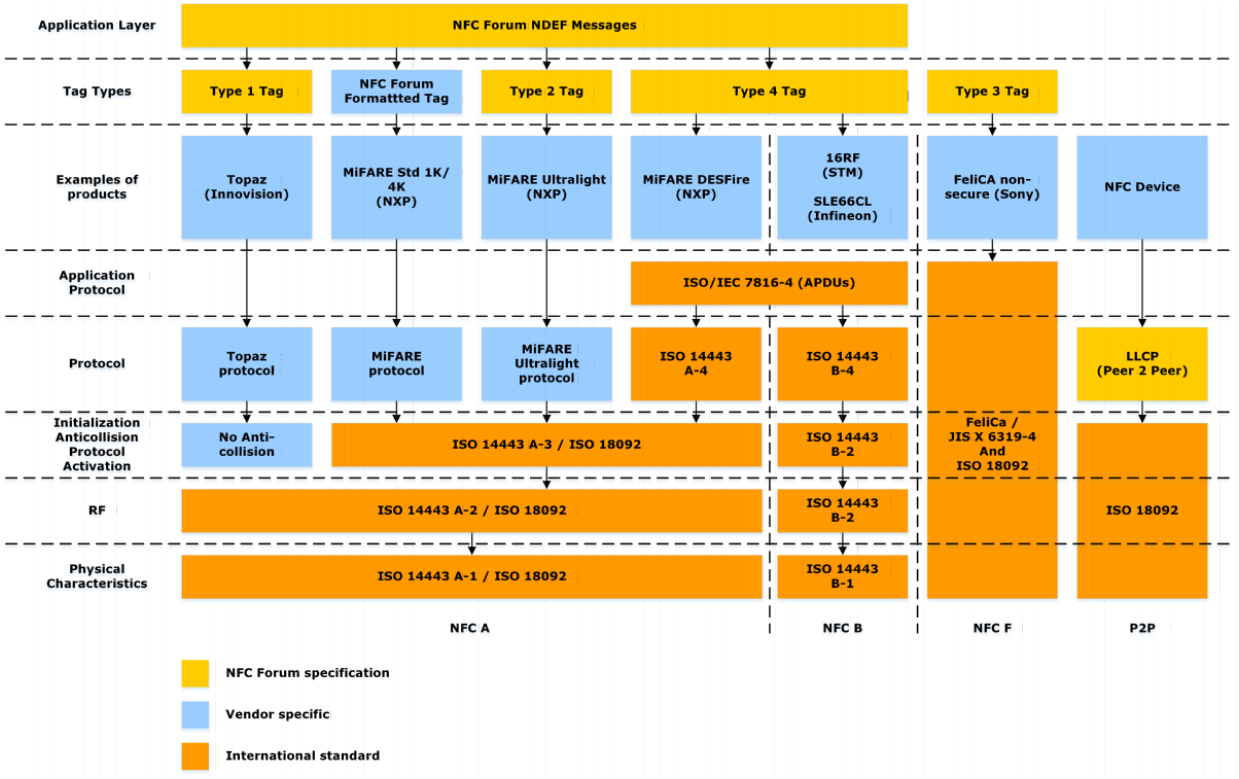
3.1. Caratteristiche fisiche

Accelerometer, Vector magnetometer (compass), Gyroscope, GPS and/or other location facilities, (Front/rear) camera, Microphone, Speaker, Battery.

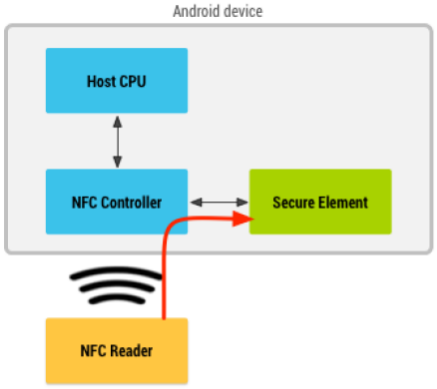
Accelerometro: 3-assi, 100 letture/s, misura 0-10g, usato per orientamento.
Bussola: sensore ad effetto hall 3-assi, 10 letture/s, misura 0-2000uT
Giroscopio: misura la velocità angolare (rad/s), 3-assi, 100 letture/s, misura 0-35 rad/s
GPS: 3 satelliti, ½ letture/s, precisione 10m.
Di solito i sensori sono a 8 bit, la linearità è limitata e consumano molto. Non adatti per scopi industriali seri.
Camera: sensori CMOS, focus fisso e automatico, risolzione 0.3Mpx-41Mpx, video 240p-4K
Microfono: ottimizzato per la voce (<8Khz), può esserci un processore per la voce e microfoni multipli per sopprimere meglio i rumori.
Speaker: ottimizzato per essere il più forte possibile.
Batteria: ricaricabile, ioni di litio perché hanno una buona densità di energia e possono essere di diverse forme, voltaggio tipico 3.7V, capacità 1000-3000mAh, si deteriora se troppo caldo, freddo sovracaricata o caricata poco spesso.
Prossimità: fotoelettrico o infrarossi, spesso ritorna solo vero/falso.
Barometro: piezoresistivo MEMS, usato per l'altitudine.
Termometro, Igrometro(umidità, capacitivo), fingerprint(electrico-ottico o capacitivo).

3.1.1. NFC

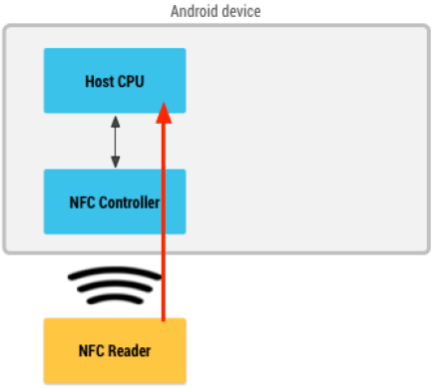
Comunicazione wireless in prossimità, evoluzione dell'RFID, spento quando lo screen è off, 3 moduli: initiator, target (può essere passivo e alimentato dal'initiator dall'induzione magnetica) e peer to peer.



Packages: `android.nfc` and `android.nfc.tech`, `android.nfc.cardemulation` (reader mode and hostbased card emulation)
API basate sullo standard NDEF



With a Secure Element (<4.4)

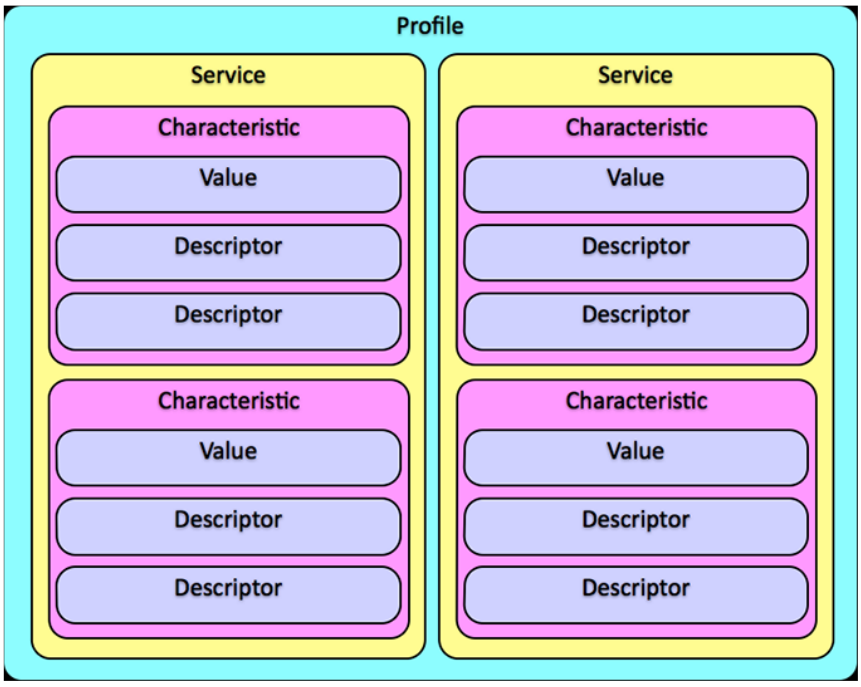


Host-Based Card Emulation (4.4+)

3 modi di operare: Reader/writer mode, P2P mode, Card emulation mode
NFC può essere usato per fare il bootstrap di altre comunicazioni, come bluetooth e wifi.

3.1.2. Bluetooth smart (Bluetooth low energy)

2 dispositivi: central e peripheral
Profili delle applicazioni basati sul Generic Attrubute Profile (GATT) in cui il central è il client e ogni peripheral è un server che fornisce servizi.



Package: `android.bluetooth` (solo central), `android.bluetooth.le` (sia central che peripheral)

3.2. Accesso tramite software

Accesso ai sensori tramite i package `android.hardware` e `android.hardware.camera2`
Accesso ai sensori (no gps) tramite le classe `SensorManager` e `Sensor`

Accesso alla camera: `CameraManager` e `CameraDevice`, richiedono permessi.

Accesso all'audio tramite package `android.media` e classi: `MediaRecorder` (Catturare audio e video, richiede permessi), `MediaPlayer` (play di audio, video e streams), `AudioTrack` (play audio) e `AudioManager` (gestisce sorgenti, output e volume, richiede permessi).

Accesso al GPS tramite il package `android.location` e classe `LocationManager` (richiede permessi, permette alle applicazioni di ottenere aggiornamenti periodici sulla posizione, o di essere notificate quando il dispositivo entra in prossimità di una posizione prestabilita)

Accesso alla batteria tramite i package `android.os` e `java.lang.Object`
I cambiamenti dello stato della batteria sono comunicati tramite intent.
Classi `BatteryManager` (contiene costanti che descrivono gli attributi della batteria), `ApplicationErrorReport.BatteryInfo` (report di un app che consuma troppa energia)

Permessi dichiarati nel manifest tramite `<uses-permission>`

3.2.1. Classe `SensorManager`

Metodi di `SensorManager`:

- **`List<Sensor> getSensorList(int type)`**
  Returns all available sensors of type `type`

- **`Sensor getDefaultSensor(int type)`**
  Returns the default sensor for the type `type`

- **`boolean registerListener(SensorEventListener listener, Sensor sensor, int rate)`**
  Registers a SensorEventListener for the sensor `sensor`. The rate for change notifications, expressed in microseconds, is given by `rate`

- **`void unregisterListener(SensorEventListener listener, Sensor sensor)`**
  Unregisters a listener for the specified sensor

- **`void unregisterListener(SensorEventListener listener)`**
  Unregisters a listener for all sensors

Tipi della classe `Sensor`:

`TYPE_ACCELEROMETER`	Accelerometer
`TYPE_AMBIENT_TEMPERATURE`	Temperature sensor (Android 4.0+)
`TYPE_GRAVITY`	Gravity sensor
`TYPE_GYROSCOPE`	Gyroscope (Android 2.3+)
`TYPE_HEART_RATE`	Heart rate monitor (Android 4.4W+)
`TYPE_LIGHT`	Light sensor
`TYPE_LINEAR_ACCELERATION`	Lin. acceleration sensor (Android 2.3+)
`TYPE_MAGNETIC_FIELD`	Magnetic field sensor
`TYPE_MOTION_DETECT`	Motion detect sensor (Android 7.0+)
`TYPE_PRESSURE`	Pressure sensor
`TYPE_PROXIMITY`	Proximity sensor
`TYPE_RELATIVE_HUMIDITY`	Humidity sensor (Android 4.0+)
`TYPE_ROTATION_VECTOR`	Rotation vector sensor (Android 2.3+)

Metodi di `Sensor`:

- **`int getType()`**
  Returns the type of the sensor
- **`float getMaximumRange()`**
  Returns the maximum range of the sensor
- **`float getResolution()`**
  Returns the resolution of the sensor
- **`int getMinDelay()`**
  Returns the minimum delay allowed between two events in microseconds.
  Returns zero if the sensor only returns a value when a change occurs
- **`float getPower()`**
  Returns the power in mA consumed by the sensor while in use

Metodi dell'interfaccia `SensorEventListener`: (può essere implementata da un activity)

`abstract void onAccuracyChanged (Sensor sensor, int accuracy)`
Called when the accuracy of sensor `sensor` has changed.
The new accuracy is specified in `accuracy`

`abstract void onSensorChanged (SensorEvent event)`
Called when sensed value has changed.
All the information about the event are contained in `event`

Attributi di `SensorEvent`:

`int accuracy`
Accuracy of values reported in the event

`Sensor sensor`
Sensor that generated the event

`long timestamp`
Time (expressed in nanoseconds) when the event happened

`final float[] values`
Values reported by the event.
The length and contents of the array depends on the sensor type. For instance, if the sensor is an accelerometer three values are reported, corresponding to the accelerations in m/s^2 along the three axes

Utilizzare `SensorManager`**:**
Istanziazione: `Context.getSystemService(SENSOR_SERVICE)`
Istanziare i sensori: `getDefaultSensor()` oppure `GetSensorList()`
Ricevere notifiche dei cambiamenti dei sensori: metodo `registerListener()` di `SensorManager`, la classe deve implementare `SensorEventListener`.

3.2.2. Classe `Camera`

Metodi di `Camera`:

- **`static int getNumberOfCameras()`**
  Returns the number of physical cameras available
- **`static void getCameraInfo(int cameraId, Camera.CameraInfo cameraInfo)`**
  Returns information about a particular camera
- **`static Camera open(int cameraId)`**
  Creates a new `Camera` object to access a particular hardware camera
- **`final void release()`**
  Disconnects and releases the `Camera` object resources
- **`Camera.Parameters getParameters()`**
  Returns the current settings for the `Camera` instance
- **`void setParameters(Camera.Parameters params)`**
  Changes the settings for the `Camera` instance
- **`final void setPreviewDisplay(SurfaceHolder holder)`**
  Sets the surface to be used for live preview
- **`final void startPreview()`**
  Starts capturing and drawing preview frames to the screen
- **`final void stopPreview()`**
  Stops capturing and drawing preview frames to the screen
- **`final void takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback postview, Camera.PictureCallback jpeg)`**
  Starts an asynchronous image capture. The camera service will invoke several callbacks to the application at different stages during the capture process.
  If the application does not need a particular callback, a `null` can be passed as the corresponding parameter

Permessi di `Camera`:

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus"
              android:required="false" />
```

in Android 6.0+ devono essere richiesti a runtime:

```
protected void onCreate(Bundle savedInstanceState)
{
    …
    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED)
    {
        tv.setText(R.string.perm_denied);
        requestCameraPermission();
    }
}

private void requestCameraPermission()
{
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
            Manifest.permission.CAMERA))
    {
        // Show dialog to the user explaining why the permission is required
    }

        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.CAMERA},
                REQUEST_CAMERA_PERMISSION);
}
```

```
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    if (requestCode == REQUEST_CAMERA_PERMISSION)
    {
        if (grantResults.length != 1 ||
            grantResults[0] != PackageManager.PERMISSION_GRANTED)
        {
            Log.i(TAG, "CAMERA permission has been DENIED.");

            // Handle lack of permission here
        }
        else
        {
            Log.i(TAG, "CAMERA permission has been GRANTED.");

            // You can now access the camera
        }
    }
    else
    {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

Utilizzo di `Camera`:

- Istanziazione: `Camera.open()`
- Se necessario: prendere i settaggi esistenti con `getParameters()`, modificarlo con `setParameters(Camera.Parameters)`.
- `setPreviewDisplay(SurfaceHolder)`
- `startPreview()`
- Catturare l'immagine: `takePicture(…)`
- Per più foto richiamare `startPreview()`
- Alla fine `release()`

3.2.3. Package `Camera2` (API 21+)

Permette l'accesso a tutte le fasi di acquisizione dell'immagine.

- **`CameraManager`** class
  Allows to enumerate, query, and open available camera devices

- **`CameraDevice`** class
  Instances represent single hardware cameras.
  To acquire an image, configure a `CameraCaptureSession`, then a `CaptureRequest`, and finally process the `TotalCaptureResult`

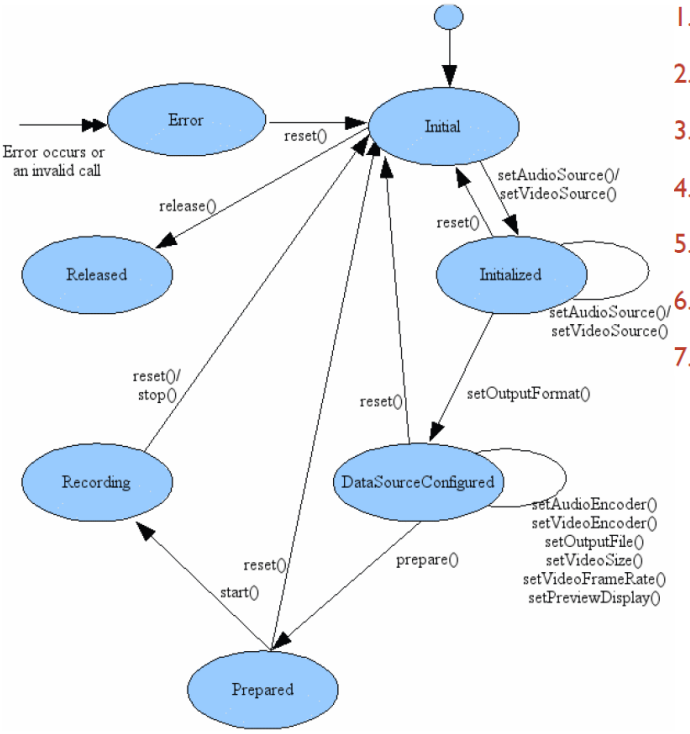
3.2.4. Classe `MediaRecorder`

Può catturare audio e video, codifica i dati e li salva nella memoria.

Metodi:

- `void setAudioSource(int audio_source)`
  Sets the audio/video source to be used for recording.
  See `MediaRecorder.AudioSource` for a list of defined audio sources
- `void setOutputFile(String path)`
  Sets where the output file should be produced
- `void setOutputFormat(int output_format)`
  Sets the format of the output file.
  See `MediaRecorder.OutputFormat` for a list of defined file formats
- `void setAudioEncoder(int audio_encoder)`
  Sets the format audio should be encoded to.
  See `MediaRecorder.AudioEncoder` for a list of defined encoders
- `setVideoSource(...)` and `setVideoEncoder(...)` methods also exist
- `void prepare()`
  Prepares the MediaRecorder for recording.
  Returns when the object is ready
- `void start()`
  Starts recording
- `int getMaxAmplitude()`
  Returns the maximum audio amplitude sampled since the last call to this method, or 0 if it is the first call
- `void stop()`
  Stops recording
- `void release()`
  Releases resources associated with the MediaRecorder

Stati:



Utilizzo:

1. Instantiate a MediaRecorder
2. Set the source(s)
3. Set the output file name and format
4. Set the encoder(s)
5. Call `prepare()`
6. Call `start()` to start capture
7. When you are done, call `stop()` and `release()`

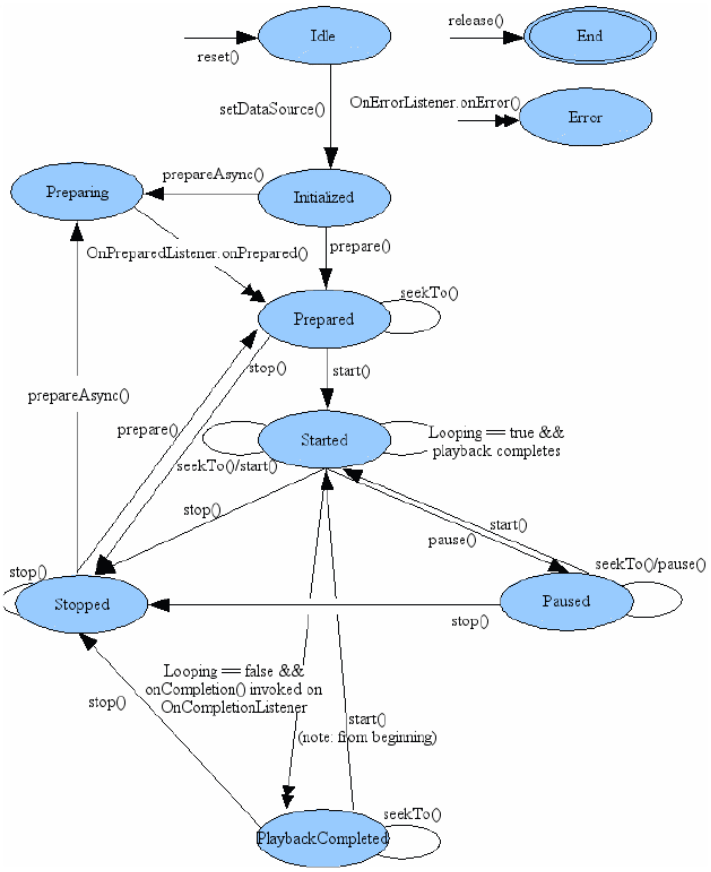
3.2.5. Classe `MediaPlayer`

Prende dati dallo storage interno o dal network, decodifica e riproduce sia l'audio che video.

Metodi:

- **`void setAudioStreamType(int streamtype)`**
  Sets the stream where decoded audio is to be sent
- **`void setDisplay(SurfaceHolder sh)`**
  Sets the surface to be used for video playback
- **`void setDataSource(String path)`**
  `void setDataSource(Context context, Uri uri)`
  Sets the source of media data
- **`prepare()`**
  Prepares the `MediaPlayer` for playback. Returns when the object is ready
- **`start()`, `stop()`**
  Starts (resp., stops) playback
- **`release()`**
  Releases resources associated with the MediaPlayer
- **`boolean isPlaying()`**
  Tells whether media data are being played
- **`int getDuration()`**
  Gets the duration of the media file in milliseconds
- **`int getCurrentPosition()`**
  Gets the current playback position in milliseconds
- **`void seekTo(int msec)`**
  Seeks to specified time position
- **`void pause()`**
  Pauses playback

Stati:



3.2.6. Classe `AudioTrack`

Solo playback, solo PCM, solo dati dal buffer

- **Modo statico:** assicura la minor latenza possibile, l'audio deve essere completamente caricato nel buffer.
- **Modo stream:** i dati possono essere presi mentre è in esecuzione, di solito utilizzato quando l'audio è troppo grande per stare interamente nel buffer.

Metodi:

- `static int getMinBufferSize(int sampleRateInHz, int channelConfig, int audioFormat)`
  Returns the minimum buffer size required for the successful creation of an AudioTrack object.
  Note: in stream mode the minimum size does not guarantee a smooth playback under load
- `AudioTrack(int streamType, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes, int mode)`
  Class constructor.
- `int write(short[] audioData, int offsetInShorts, int sizeInShorts)`
  Writes audio data into the memory buffer. In streaming mode, will block until all data has been written. Returns the number of shorts that were written
- `void flush()`
  Removes all audio data from the memory buffer
- `int getPlayState()`
  Returns the playback state: stopped, paused, or playing
- `void play()`
  Starts playback
- `void pause()`
  Pauses playback
- `void stop()`
  Waits for the memory buffer content to be consumed completely, then stops playback.
  Note: for an immediate stop, use pause(), followed by flush()
- `int getPlaybackHeadPosition()`
  Returns the playback position, expressed in samples. This is a continuously advancing counter
- `int setPlaybackHeadPosition(int positionInFrames)`
  Sets the playback position, expressed in samples. Works only in static mode

3.2.7. Classe `AudioManager`

Per registrare l'audio dal microfono.

Metodi:

- **`void setMode(int mode)`**
  Sets the audio mode (audio routing, including the telephony layer).
  Should be used by applications that need to override the platform-wide management of audio settings

- **`void adjustVolume(int direction, int flags)`**
  Adjusts the volume of the most relevant stream.
  Should be used by applications need to override the platform-wide management of audio settings

- **`boolean isMicrophoneMute()`**
  Checks whether the microphone mute is on or off

- **`boolean isSpeakerphoneOn()`**
  Checks whether the speakerphone is on or off

Permessi:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

3.2.8. Posizione

Classe `LocationManager`

Metodi:

- `List<String> getAllProviders()`
  Returns a list of the names of all known location providers

- `LocationProvider getProvider(String name)`
  Returns the information associated with the location provider `name`

- `void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)`
  Registers the current activity to be notified periodically by the named provider.
  Periodically, the supplied `LocationListener` will be called

- `Location getLastKnownLocation(String provider)`
  Returns a Location indicating the data from the last known location fix obtained from `provider`. This can be done without starting the provider, hence without consuming battery power

- `void removeUpdates(LocationListener listener)`
  Removes any current registration for location updates of the current activity with the given `LocationListener`

Interfaccia `LocationListener`

Metodi:

- `abstract void onLocationChanged(Location location)`
  Called when the location has changed

- `abstract void onProviderDisabled(String provider)`
  Called when the location provider is disabled by the user

- `abstract void onProviderEnabled(String provider)`
  Called when the location provider is enabled by the user

- `abstract void onStatusChanged(String provider, int status, Bundle extras)`
  Called when the provider status changes (example: the provider becomes temporarily unavailable because there is no GPS signal)
  `extras` contain provider-specific status variables

Classe `Location`

Metodi:

- `double getLatitude()`
  `double getLongitude()`
  Return the latitude and longitude contained in the `Location` instance (the "fix")

- `double getAltitude()`
  Returns the altitude of the fix.
  If `hasAltitude()` is `false`, 0.0 is returned

- `float getBearing()`
  Returns the direction of travel in degrees East of true North.
  If `hasBearing()` is `false`, 0.0 is returned

- `float getSpeed()`
  Returns the speed of the device over ground, in m/s.
  If `hasSpeed()` is `false`, 0.0 is returned

- `long getTime()`
  Returns the UTC time of the fix, in milliseconds since January 1, 1970

Permessi:

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

Utilizzo di LocationManager:

1. Obtain an instance of `LocationManager` by calling `Context.getSystemService(LOCATION_SERVICE)`. Do not directly instantiate objects of this class!
2. Implement a `LocationListener` that responds to location updates
3. Register the listener by calling `LocationManager`'s method `requestLocationUpdates(...)`
4. When you are done, call `removeUpdates(...)`

3.2.9. Batteria

Intents:

- **`public static final String ACTION_BATTERY_CHANGED`**
  Sticky intent containing the charging state, level, and other info about the battery
- **`public static final String ACTION_BATTERY_LOW`**
  Indicates the battery is running low. A "Low battery warning" system dialog is also shown to the user
- **`public static final String ACTION_BATTERY_OKAY`**
  Indicates the battery is no longer low
- **`public static final String ACTION_POWER_DISCONNECTED`**
  External power has been removed : the device is now running on batteries
- **`public static final String ACTION_POWER_CONNECTED`**
  External power has been connected: the device is no longer running on batteries

Costanti e stringhe:

- Integer constants for the extended intent datum **`EXTRA_HEALTH`**:
  `BATTERY_HEALTH_COLD` (Android 3.0+), `BATTERY_HEALTH_DEAD`, `BATTERY_HEALTH_GOOD`, `BATTERY_HEALTH_OVERHEAT`, `BATTERY_HEALTH_OVER_VOLTAGE`, `BATTERY_HEALTH_UNKNOWN`, `BATTERY_HEALTH_UNSPECIFIED_FAILURE`
- Integer constants for the extended intent datum **`EXTRA_PLUGGED`**:
  `BATTERY_PLUGGED_AC`, `BATTERY_PLUGGED_USB`
- Integer constants for the extended intent datum **`EXTRA_STATUS`**:
  `BATTERY_STATUS_CHARGING`, `BATTERY_STATUS_DISCHARGING`, `BATTERY_STATUS_FULL`, `BATTERY_STATUS_NOT_CHARGING`, `BATTERY_STATUS_UNKNOWN`
- The extended intent datum **`EXTRA_LEVEL`** is an integer that contains the current battery level (from 0 to `EXTRA_SCALE`)

4.Salvataggio dei dati

Fatto tramite:

- SharedPreferences
- SQLite
- Filesystem
- Network (es google drive)

4.1. SharedPreferences

Quando un app viene chiusa, alla sua riapertura si vuole che sia come se non fosse stata chiusa.
Se un activity viene distrutta e ricreata dal sistema operativo (Es. quando si ruota lo schermo) la nuova activity è una nuova istanza di quella vecchia.
Se un activity viene distrutta per il normale corso (Es. si preme Back) essa è perduta per sempre.
Istance state: informazione associata ad un'istanza di un activity
Persistace state: informazione associata all'applicazione. (Es. preferenze utente) Deve essere preservata in diversi run dell'app.

Si salvano i dati nelle **SharedPreferences**

Leggere:

```
// Get persistent data stored as SharedPreferences
SharedPreferences preferences = getPreferences(MODE_PRIVATE);
String str_et = preferences.getString("editTextValue", null);
boolean bln_cb = preferences.getBoolean("checkBoxValue", false);
int int_sb = preferences.getInt("seekBarValue", 0);
```

Salvare:

```
// Store values between instances here
SharedPreferences preferences = getPreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
// Store status in the preferences
editor.putString("editTextValue", str_et);
editor.putBoolean("checkBoxValue", bln_cb);
editor.putInt("seekBarValue", int_sb);

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

4.2. SQLite

Libreria software che implementa una versione leggera dei database SQL, quasi tutte le funzioni di SQL-92. Non dipende da librerie esterne. Un solo file sorgente binario.

```
create table addressbook
(
    _id   integer primary key,
    name  text,
    phone text
);
```

```
insert into addressbook
values
(
    736,
    'John Doe',
    '555-1212'
);
```

```
update table addressbook
set phone='555-1424'
where  id=736;
```

```
delete from addressbook
where name like "%doe%";
```

```
delete from addressbook
where _id=736;
```

```
select name, phone
from mytable
where
    _id > 100
    and
    name like "%doe%"
order by name;
```

4.2.1. SQLite in C

Metodi in C:

- **int sqlite3_open(char *filename, sqlite3 **ppDb)**
  Opens a connection to the SQLite database identified by `filename`.
  Returns a database connection entity `ppDb`.
  Like all SQLite3 APIs, returns an integer error code

- **int sqlite3_close(sqlite3 *pDB)**
  closes a database connection previously opened by a call to `sqlite3_open()`
- **int sqlite3_prepare_v2(sqlite3 *pDB, char *sqlStatement, int nByte, sqlite3_stmt **ppStmt, char **pzTail)**
  Converts the SQL statement `sqlStatement` into a prepared statement object.
  Returns a pointer `ppStmt` to the prepared object

- **int sqlite3_finalize(sqlite3_stmt *pStmt)**
  Destroys a prepared statement.
  Every prepared statement must be destroyed with this routine in order to avoid memory leaks

- **int sqlite3_step(sqlite3_stmt *pStmt)**
  Evaluates a prepared statement up to the point where the first row of the result is available

- `int sqlite3_column_count(sqlite3_stmt *pStmt)`
  Gives the number of columns in the result set returned by the prepared statement

- `int sqlite3_column_type(sqlite3_stmt *pStmt, int iCol)`
  Returns the datatype code for the initial data type of the result column `iCol`.
  The returned value is one of `SQLITE_INTEGER`, `SQLITE_FLOAT`, `SQLITE_TEXT`, `SQLITE_BLOB`, or `SQLITE_NULL`

- `int sqlite3_column_int(sqlite3_stmt *pStmt, int iCol)`,
  `double sqlite3_column_double(sqlite3_stmt*, int iCol)`,
  `...`
  Family of functions that return information about a single column

- `int sqlite3_exec(sqlite3 *pDB,`
  `const char *sqlString,`
  `int (*callback)(void*,int,char**,char**),`
  `void *, char **errmsg)`
  Convenience wrapper for `sqlite3_prepare_v2()`, `sqlite3_step()`, and `sqlite3_finalize()`.
  Runs the SQL statements contained in `sqlString`.
  If the callback function of the 3rd argument to `sqlite3_exec()` is not `NULL`, then it is invoked for each result row coming out of the evaluated SQL statements

Errori in C:

`SQLITE_OK`	Successful result
`SQLITE_ERROR`	SQL error or missing database
`SQLITE_INTERNAL`	Internal logic error in SQLite
`SQLITE_PERM`	Access permission denied
`SQLITE_ABORT`	Callback routine requested an abort
`SQLITE_BUSY`	The database file is locked
`SQLITE_LOCKED`	A table in the database is locked
`SQLITE_NOMEM`	A `malloc()` failed
`SQLITE_READONLY`	Attempt to write a readonly database
`SQLITE_INTERRUPT`	Operation terminated by `sqlite3_interrupt()`
`SQLITE_IOERR`	Some kind of disk I/O error occurred
`SQLITE_CORRUPT`	The database disk image is malformed
`SQLITE_NOTFOUND`	Unknown opcode in `sqlite3_file_control()`
`SQLITE_FULL`	Insertion failed because database is full
`SQLITE_CANTOPEN`	Unable to open the database file
`SQLITE_PROTOCOL`	Database lock protocol error
`SQLITE_EMPTY`	Database is empty
`SQLITE_SCHEMA`	The database schema changed
`SQLITE_TOOBIG`	String or BLOB exceeds size limit
`SQLITE_CONSTRAINT`	Abort due to constraint violation
`SQLITE_MISMATCH`	Data type mismatch
`SQLITE_MISUSE`	Library used incorrectly
`SQLITE_NOLFS`	Uses OS features not supported on host
`SQLITE_AUTH`	Authorization denied
`SQLITE_FORMAT`	Auxiliary database format error
`SQLITE_RANGE`	2nd parameter to `sqlite3_bind` out of range
`SQLITE_NOTADB`	File opened that is not a database file
`SQLITE_ROW`	`sqlite3_step()` has another row ready
`SQLITE_DONE`	`sqlite3_step()` has finished executing

Esempi in C:

```
char *err;

const char *sqlString =
    "CREATE TABLE IF NOT EXISTS addressbook ("
    "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
    "name TEXT NON NULL, "
    "phone TEXT);";

if (sqlite3_exec(db, sqlString, NULL, NULL, &err) != SQLITE_OK)
{
    sqlite3_close(db);
    LogError(0, @"Error while creating table.");
}
void insertIntoAddressbook(int i, char* name, char* phone)
{
    char *sql = "INSERT INTO addressbook VALUES (?, ?, ?);";
    sqlite3_stmt *stmt;

    if (sqlite3_prepare_v2(db, sql, -1, &stmt, nil) == SQLITE_OK)
    {
        sqlite3_bind_int (stmt, 1, i);
        sqlite3_bind_text(stmt, 2, name, -1, NULL);
        sqlite3_bind_text(stmt, 3, phone, -1, NULL);
    }

    if (sqlite3_step(stmt) != SQLITE_DONE)
        LogError(@"Error while adding row.");

    sqlite3_finalize(stmt);
}
void processContactById(int contactId)
{
    sqlite3_stmt * statement;
    char query_stmt[64];

    snprintf(query_stmt, 64,
             "SELECT name, phone FROM addressbook WHERE _id=%d", contactId);

    if (sqlite3_prepare_v2(db, query_stmt, -1, &statement, NULL) == SQLITE_OK)
    {
        if (sqlite3_step(statement) == SQLITE_ROW)
        {
            // Obtain values with sqlite3_column_text(statement, 0)
            // and sqlite3_column_text(statement, 1),
            // then use them for whatever you like
        }
    }

    sqlite3_finalize(statement);
}
```

4.2.2. SQLite in Java

Java Package: `android.database.sqlite`

Classi:

- `SQLiteDatabase`
- `SQLiteOpenHelper`
- `SQLiteStatement`: incapsula gli statement pre-copmpilati
- `SQLiteQueryBuilder`
- `SQLiteCursor`

Metodi di `SQLiteDatabase`:

- **`SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)`**
  Opens a database according to `flags`
- **`void close()`**
  Closes a database
- **`void execSQL(String sql)`**
  Executes a single SQL statement that is neither a SELECT nor any other SQL statement that returns data
- There are also convenience methods named **`insert`**, **`delete`**, **`replace`**, **`update`**, ... to ease the execution of the corresponding SQL commands
- **`Cursor rawQuery(String sql, String[] selectionArgs)`**
  Runs the provided SQL statement returning data, and returns a `Cursor` over the result set

Metodi di `SQLiteCursor`: accedono al risultato restituito dalle query

- **`int GetCount()`**
  Returns the number of rows in the cursor
- **`boolean moveToFirst(), moveToLast(), moveToNext(), moveToPrevious(), moveToPosition(int position)`**
  Moves the cursor to the specified row
- **`int getType(int columnIndex)`** (Android 3.0+)
  Returns the data type of the given column's value
- **`getString(int columnIndex), getInt(int columnIndex), getFloat(int columnIndex), ...`**
  Returns the value for the given column in the current row

Metodi di `SQLiteOpenHelper`:

- **`abstract void onCreate(SQLiteDatabase db)`**
  Called when the database is created for the first time.
  The implementation should use this method to create tables and relations between tables

- **`abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`**
  Called when the database schema needs to be upgraded (e.g., because a new version of the application has been installed).
  The implementation should use this method to drop/add tables, or do anything else it needs to upgrade to the new schema version

Room: separa l'accesso dati dallo storage. Package `android.arch.persistence.room`, va aggiunfo ad app build.gradle:

```
dependencies {
    …
    // Room
    implementation "android.arch.persistence.room:runtime:1.0.0"
    annotationProcessor "android.arch.persistence.room:compiler:1.0.0"
    …
}
```

Classi: Database, Entity (riga del database), Dao (Data Access Object).

5.Multitasking

5.1. Principi

Parallelismo:

- Livello di bit: aumenta la grandezza delle parole
- Livello di istruzioni: aggiunge unità funzionali che possono operare in parallelo
- Task: aggiunge più processori per eseguire più programmi contemporaneamente.

Processo (task): istanza di un programma in esecuzione, processi diversi non condividono risorse.
Thread: è all'interno di un processo, dotato di stato (running, ready ecc.), ha accesso alla memoria e alle risorse del processo contenente.
Multitasking (multithreading): abilità di gestire più flussi di esecuzione (SE) concorrentemente.
Il numero di flussi può essere maggiore delle unità di esecuzione.
A livello macchina istruzioni in diversi flussi si intervallano in maniera imprevedibile.

Problemi della concorrenza: Thread diversi possono avere accesso a dati comuni, un thread legge dei dati mentre un altro li sta modificando.
Soluzione: locking tramite:

- Mutua esclusione tramite semafori binari sia per leggere che per scrivere.
- Lettura sempre permessa ma scrittura serializzata.
- Semafori numerici.

Altri problemi:

- Starvation: alcuni processi possono rimanere in un attesa indefinita.
- Stallo (deadlock): quando il sistema non può raggiungere il suo stato finale in quanto i processi si bloccano a vicenda.

Thread safe: pezzo di codice che può essere invocato da più thread contemporaneamente. Ovvero non è soggetto a concorrenza e a problemi di consistenza della memoria.

Multitasking cooperativo: ….
Multitasking preemptive: i flussi di esecuzione (SE) sono rimossi forzatamente dall'utilizzo delle risorse. (Metodo utilizzato da android, ios e Wp).
Più flussi possono essere eseguiti contemporaneamente, ma non le app (in generale).
Limiti: le app a schermo sono eseguite senza permessi, quelle in background richiedono permessi.

5.2. Concorrenza in Java

Classe `Thread`.
Sincronizzazione tramite i metodi `wait()`, `notify()` e `join()`.
Muta esclusione di un pezzo di codice tramite `synchronized`
Interfaccia `Runnable` con l'unico metodo `run()` che contiene il codice da eseguire.

Modi per creare un nuovo thread:

- Istanziare una classe derivata da `Thread`

```
public class HelloThread extends Thread
{
    public void run()
    {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[])
    {
        (new HelloThread()).start();
    }
}
```

- Creare un istanza di `Thread`, e passare al costruttore un oggetto che implementa `Runnable`

```
public class HelloRunnable implements Runnable
{
    public void run()
    {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[])
    {
        (new Thread(new HelloRunnable())).start();
    }
}
```

Metodi di `Thread`:

- **`void start()`**
  Causes the thread to begin execution
- **`void setPriority(int newPriority)`**
  Changes the priority of the thread
- **`static void sleep(long millis, int nanos)`**
  Causes the thread to pause execution for the specified number of milliseconds plus the specified number of nanoseconds
- **`public final void wait(long timeout)`** (inherited from `Object`)
  Causes the thread to wait until either another thread invokes the notify() method or a specified amount of time has elapsed
- **`public final void notify()`** (inherited from `Object`)
  Wakes up the thread
- **`void join()`**
  Causes the current thread to pause execution until the thread upon which `join()` has been invoked terminates. Overloads of `join()` allow to specify a waiting period

Se un oggetto è visibile a più thread, le letture e scritture sulle sue variabili possono essere fatte attraverso metodi `synchronized` per evitare problemi di concorrenza. Esempio:

```
public class SynchronizedCounter
{
    private int c = 0;

    // Mutual exclusion: no race conditions
    public synchronized void increment() { c++; }
    public synchronized void decrement() { c--; }

    // Mutual exclusion: no memory consistency errors
    public synchronized int value() { return c; }
}
```

Ogni dichiarazione può essere dichiarata `synchronized` specificando l'oggetto che fornisce il lock.

```
public class MsLunch
{
    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1()
    {
        synchronized(lock1) { c1++; }
    }

    public void inc2()
    {
        synchronized(lock2) { c2++; }
    }
}
```

Altri package per la concorrenza in Java:

- `java.util.concurrent`
- `java.util.concurrent.atomic`
- `java.util.concurrent.locks`

5.3. Concorrenza in Android

Quando il primo componente di un app parte, il sistema operativo crea un nuovo processo con un singolo thred (il **main thread**). Di norma tutti i componenti di un app sono eseguiti in questo thread, le richieste di esecuzione vengono messe in una coda.
Il main thread è anche il UI thread. Quindi per operazioni lunghe è meglio creare thread a parte (**worker threads**) per evitare che la UI si blocchi.

Componenti diversi possono essere eseguiti in processi diversi.
Componenti di app diverse possono essere eseguiti nello stesso processo.

Nel manifest si possono settare appropriatamente gli attributi: `android:process` e `android:multiprocess`.
Non accedere alla UI al di fuori del main thread perché essa non è thread safe, quindi:

- `Activity.runOnUiThread(Runnable)`: mette il Runnable nella event queue del main thread.
- `View.post(Runnable)` e `View.postDelayed(Runnable, long)`: Runnable è messo nella message queue ed eseguito più tardi nel main thread.

Classe `Handler`:

- Riceve messaggi ed esegue codice per gestire i messaggi (metodo `handleMessage()`).
- Una nuova istanza di `Handler` può essere connessa ad un thread esistente o può essere eseguita in un nuovo thread.
- Connettere un'istanza di `Handler` al main thread per gestire in sicurezza i messaggi che riguardano la UI.

Classe `AsyncTask`:

- deve essere sottoclasse
- Override `doInBackground()` per eseguire il codice
- Override `onPostExecute(Result result)` che riceve il risultato dall'esecuzione in background che verrà processato nel main.

```
public void onClick(View v)
{
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap>
{
    // What to do; guaranteed to run in a worker thread
    protected Bitmap doInBackground(String... urls)
    {
        return loadImageFromNetwork(urls[0]);
    }

    // Processing of the result; guaranteed to run in the main thread
    protected void onPostExecute(Bitmap result)
    {
        mImageView.setImageBitmap(result);
    }
}
```

Distruzione dei processi:
Il sistema operativo cerca di mantenere i processi più a lungo possibile, ma può essere costretto a chiuderne alcuni per far spazio alle risorse. Il SO mantiene 5 livelli di importanza basati sui componenti eseguita da un processo. Ad ogni processo è assegnato il livello del più importante componente che esegue. Un livello di un processo più aumentare se ci sono altri processi che dipendono da lui.
Gerarchia:

- **Foreground process** (maximum importance)
  A process that is required for what the user is currently doing
  - It hosts an activity that the user is interacting with
  - It hosts a service that is bound to the activity that the user is interacting with
  - It hosts a service that has called `startForeground()`
  - It hosts a service that is executing one of its lifecycle callbacks
  - It hosts a broadcast receiver that is executing its `onReceive()` method
- **Visible process**
  A process that does not have any foreground components, but still can affect what the user sees on screen
  - It hosts an activity that is not in the foreground, but is still visible to the user (e.g., its `onPause()` method has been called; it started a modal dialog)
  - It hosts a service that is bound to a visible activity
- **Service process**
  A process that is running a service that does not fall into either of the two previous hierarchy levels
- **Background process**
  A process holding an activity that is not currently visible to the user.
  Processes in this hierarchy level are kept in an LRU list, so that the process with the activity that was most recently seen by the user is the last to be killed
- **Empty process** (minimum importance)
  A process that does not hold any active application components

Schedulazione dei processi: Processi in foreground sono schedulati regolarmente, tutti gli altri potrebbero essere fermati o distrutti in qualunque momento.

Distruzione (Killing) dei Thread: quando la configurazione runtime cambia (es rotazione dello schermo) i worker threads vengono distrutti anche se appartengono ad un processo di elevata priorità.
Soluzioni:

- Spegnere e riavviare i thread propriamente.
- Chiedere al SO di non distruggere i thread

Esecuzione in background:

- **Services**: permettono eseguire le app in background per molto tempo (minuti). Dopo di che il SO ferma i servizi dell'app.
- **Broadcast receivers**: permetto di eseguire le app in background per un breve periodo di tempo (max 10s).
- **Content provider:** incapsulano dati strutturati e forniscono l'accesso ad essi. Usati di solito per condividere dati tra processi diversi. L'accesso ad essi richiede tempo, il che può rallentare la UI se eseguiti nel main thread.
- **Classe** `JobScheduler`: permette di definire un'unità di lavoro (job) per essere schedulata asincronamente in background.

5.4. Concorrenza in C++

Ogni programma ha almeno un Thread e ne può creare di più.
Sincronizzazione tramite `join()` e `detach()`.
Mutua esclusione tramite tipi atomici e classi mutex.
Un thread inizia immediatamente quando un oggetto viene istanziato dalla classe `Thread`.
Il codice viene passato all'interno di una funzione come parametro al costruttore di `Thread`.

```
#include <iostream>
#include <thread>

void f(int i)
{
    std::cout << "Hello, here is an int: "
              << i << std::endl;
}

int main()
{
    std::thread t1(f, 27);

    // If you omit this call, the result is undefined
    t1.join();

    return 0;
}
```

Metodi di `Thread`:

- **`bool joinable()`**
  Returns `true` if the thread object is *joinable*, i.e., it actually represents a thread of execution, and `false` otherwise
- **`id get_id()`**
  If the thread object is joinable, returns a value that uniquely identifies the thread
- **`void join()`**
  Causes the current thread to pause execution until the thread upon which `join()` has been invoked terminates
- **`void detach()`**
  Causes the current thread to be detached from the thread upon which `detach()` has been invoked

Tipi atomici: sono tipi in cui è garantito l'accesso esclusivo

Atomic type	Contains
atomic_bool	bool
atomic_char	char
atomic_int	int
atomic_uint	unsigned int

Mutex:

- **`mutex` class**
  Implements a binary semaphore. Does not support recursion (i.e., a thread shall not invoke the `lock()` or `try_lock()` methods on a mutex it already owns): use the `recursive_mutex` class for that

- **`timed_mutex` class**
  A mutex that additionally supports timed "try to acquire lock" requests (`try_lock_for(...)` method)

- **`void lock(...)` function**
  Locks all the objects passed as arguments, blocking the calling thread until all locks have been acquired

- **`bool try_lock(...)` function**
  Nonblocking variant of `lock(...)`. Returns `true` if the locks have been successfully acquired, `false` otherwise

6. Componenti delle App

Activity: Singola pagina UI.
Service: svolto in background.
Broadcast Receiver: risponde agli eventi esterni (es batteria scarica).
Content provider: incapsula i dati che necessitano di essere condivisi con altre app.

6.1. Services

Eseguiti in background per un tempo potenzialmente indefinito per svolgere operazioni di lunga durata o eseguire processi del client.
Gestiti tramite le classi astratte `Service` e `IntentService`, sono dichiarati nel `AndroidManifest.xml`
Possono essere privati all'applicazione che gli ha definiti oppure utilizzabili da altre app.
Di default un'istanza di `Service` è eseguita nel main thread, è comunque possibile creare dei worker thread. Un'istanza di `IntentService` lo fa in automatico.
È possibile eseguire un servizio in un processo separato specificandolo nel manifest.

Principali attributi del manifest:

- **`android:enabled`** (boolean)
  Whether or not the service can be instantiated by the system

- **`android:exported`** (boolean)
  Whether or not components of other apps can invoke the service or interact with it

- **`android:permission`** (string)
  Name of a permission (e.g., a `signature` permission) that an entity must have in order to launch the service or bind to it

- **`android:process`** (string)
  Name of the process where the service is to run

Come usare un service:

- Implementarlo come sottoclasse di `Service` o `IntentService`
- Dichiararlo nel manifest
- Avviarlo invocando `startService()` o `bindService()`
- Per comunicare con esso si usano intent, `Binder` (solo se il client e il servizio sono nella stessa app e processo), `Messanger` (anche quando sono in app e processi diversi) o AIDL

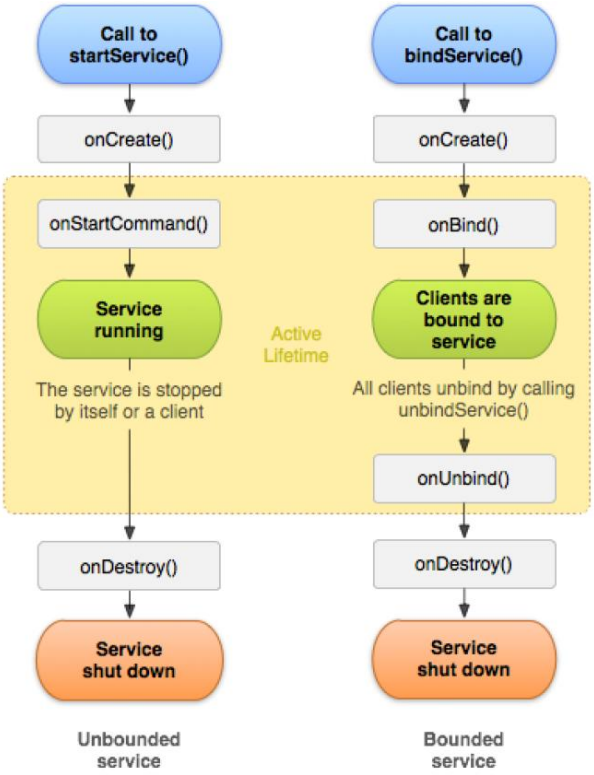
6.1.1. Classe `Service`

Classe di base per tutti i servizi, no UI, si usano le notifiche per interagire con l'utente. Di default no thread.

Stati di un servizio e ciclo di vita

- **Started service**
  Created by invoking the `Context.startService(Intent service)` method. The service performs the job specified by the intent and does not return a result to the caller. When the job is over, the service may 1) terminate itself or 2) wait for further jobs

- **Bound service**
  Created by invoking the `Context.bindService(Intent service, ServiceConnection conn, int flags)` method. Runs as long as at least one app component is bound to it. Offers a client-server, bidirectional communication interface that allows components to interact with the service, even across processes (IPC)



Metodi principali:

- `void onCreate()`
  Called by the system when the service is first created

- `int onStartCommand(Intent intent, int flags, int startId)`
  Called after another component has started the service by invoking `Context.startService(…)`. The integer `startId` is a unique token representing the start request. Must return a value that describes how the OS should continue the service after a kill (more about it later)

- **START_NOT_STICKY**
  The OS does not recreate the service, unless there are pending intents to deliver. Any unfinished or pending jobs are lost: the app must manually restart them
- **START_STICKY**
  The OS recreates the service and calls `onStartCommand(...)` with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. Unfinished jobs are lost, pending jobs are not
- **START_REDELIVER_INTENT**
  The OS recreates the service and calls `onStartCommand(...)` with the last intent that was delivered to the service. Pending intents are delivered in turn

- **IBinder onBind(Intent intent)**
  Called after another component has requested to bind with the service by invoking `Context.bindService(...)`. Must return an interface that the component can use to communicate with the service
- **void startForeground(int id, Notification n)**
  Notifies the OS that killing the service would be disruptive to the user. Supplies a `Notification` to be shown to the user while in this state
- **void stopForeground(boolean removeNotificat)**
  Removes the service from foreground state, allowing the OS to kill it more freely
- **void stopSelf()**
  Stops the service. The effect is the same as when a component invokes `Context.stopService(...)`
- **boolean onUnbind(Intent intent)**
  Called when all clients have disconnected from a particular interface published by the service. The default implementation does nothing
- **void onDestroy()**
  Called by the OS to notify the service that it is being removed. The service should clean up any resources it holds

Un componente può fermare un servizio tramite `Context.stopService(Intent service)`
Un servizio non viene distrutto finché ci sono componenti collegati a `BIND_AUTO_CREATE`
Un componente può scollegarsi da un servizio tramite `Context.unBindService(ServiceConnection conn)`

Limiti dalle API 26: quando l'app va in background ha un tempo limitato per avviare e usare i servizi, dopo di che questi vengono fermati.

6.1.2. Classe `IntentService`

Classe ereditata da Service. È più facile da utilizzare. Quando una richiesta (`Intent`) arriva, la gestisce in un worker thread, e la ferma in automatico quando il lavoro è svolto.
Si deve solo implementare `onHandleIntent(Intent intent)`.
Le richieste vengono messe in coda e processate una alla volta.
Esempio:

```
public class HelloIntentService extends IntentService
{
    // A constructor is required,
    // and must call the super IntentService(String) constructor
    // with a name for the worker thread
    public HelloIntentService()
    {
        super("HelloIntentService");
    }

    // This method is called from within the worker thread
    // with the intent that started the service.
    // When this method returns, IntentService stops the service,
    // as appropriate
    @Override
    protected void onHandleIntent(Intent intent)
    {
        // Do some work

        /*...*/
    }
}
```

Il servizio si invoca tramite:

```
Intent i = new Intent(this, HelloIntentService.class);
startService(i);
```

6.1.3. Bound service (servizio collegato)

Creare una connessione di lunga durata con uno o più componenti di app.
Un componente si connette al servizio invocando `Context.bindService(Intent service, ServiceConnection conn, int flags)`, che ritorna TRUE se la connessione è stabilita.
Tramite l'oggetto `conn` il servizio notifica il componente quando inizia o si ferma e fornisce un'interfaccia che specifica come il componente può comunicare con il servizio.
Quando non ci sono componenti legati al servizio il sistema lo distrugge.

- **`void onServiceConnected(ComponentName name, IBinder servicebinder)`**
  Called when a connection to the service has been established, with the **`servicebinder`** object describing the communication interface with the service
- **`void onServiceDisconnected(ComponentName name)`**
  Called when a connection to the service has been lost

Several ways to provide an object implementing the `IBinder` interface

- Extend the Android **`Binder`** class
- Use a **Messenger**
  The service defines a `Handler` that is the basis for a `Messenger` that can then share an `IBinder` with the client, allowing the client to send commands to the service using `Message` objects. The client can define an additional `Messenger` of its own so the service can send messages back
- Use **AIDL**
  Create an .aidl file that defines the programming interface. The Android SDK tools use this file to generate an abstract class that implements the interface and handles IPC, which you can then extend within the service

The `onBind()` method of the service returns an instance of a `Binder`-derived class that either:

1. contains public methods that the client can call
2. returns the current instance of the class implementing the service, which has public methods the client can call
3. returns an instance of another class hosted by the service with public methods the client can call

6.2. Broadcast receiver

Risponde agli eventi esterni (es batteria scarica).
Gestito tramite la classe astratta `BroadcastReceiver` e `Intent` per inviare/ricevere broadcasts.
Può essere registrato in modo:

- **Statico**: nel manifest tramite il tag `<receiver>`
- **Dinamico**: invocando nel codice `registerReceiver(BroadcastReceiver receiver, IntentFilter filter)`

I broadcast possono essere generati dal sistema che li manda a tutte la app o dalle app che possono inviarli a sé stesse o a tutti.
Non possono mostrare una UI ma possono creare notifiche.
Limiti da API 26+: le app non possono usare il manifest per registrare broadcat impliciti.

Esempi di broadcast:

- **`Intent.ACTION_AIRPLANE_MODE_CHANGED`**
  The user has switched the phone into or out of "airplane mode"
- **`Intent.ACTION_CONFIGURATION_CHANGED`**
  Device configuration (orientation, locale, etc) has changed
- **`Intent.ACTION_DATE_CHANGED`**, **`Intent.ACTION_TIME_CHANGED`**
  The date/time has changed
- **`Intent.ACTION_INPUT_METHOD_CHANGED`**
  An input method has been changed
- **`Intent.ACTION_LOCALE_CHANGED`**
  The current device's locale has changed
- **`Intent.ACTION_PACKAGE_CHANGED`**
  An existing application package has been changed
  (e.g. a component has been enabled or disabled)
- **`Intent.ACTION_BOOT_COMPLETED`**
  Broadcast once after the system has finished booting
- **`Intent.ACTION_CAMERA_BUTTON`**
  The camera button was pressed
- **`Intent.ACTION_DEVICE_STORAGE_LOW`**
  `Intent.ACTION_DEVICE_STORAGE_OK`
  Indicates low memory condition on the device begins / no longer exists
- **`Intent.ACTION_SCREEN_OFF`**
  `Intent.ACTION_SCREEN_ON`
  The device has gone to / exits from non-interactive mode
- **`Camera.ACTION_NEW_PICTURE`**
  `Camera.ACTION_NEW_VIDEO`
  A new picture/video has been taken by the camera,
  and it has been added to the media store
- **`AudioManager.ACTION_AUDIO_BECOMING_NOISY`**
  Audio is about to become "noisy" due to a change in audio outputs (e.g., a wired headset has been unplugged)
- **`ConnectivityManager.CONNECTIVITY_ACTION`**
  A change in network connectivity has occurred:
  a default connection has either been established or lost

Utilizzo:

- Implementarlo come sottoclasse di `BroadcastReceiver`
- Registrarlo
- Metodo `onReceive(Context context, Intent intent)` viene invocato quando si riceve il broadcast. A cui vengono dati 10 secondi per completare l'operazione.

```
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction();
        Log.i(TAG, "Received broadcast action: " + action);

        // Perform some useful work here
        // (after having further examined the intent, if necessary)

        …

    }
}
```

Inviare broadcast: `Context.sendBroadcast(Intent intent)`

Classe `LocalBroadcastManager`: classe di aiuto agli intent solo sugli oggetti locali con il processo. È più efficiente che mandare un broadcast globale. In questo modo i dati non escono dalla app, per evitare che vengano rubati dati sensibili.

6.3. Content provider

Incapsula dati che hanno bisogno di essere condivisi con altre applicazioni.
Modello: i dati sono esposti in tabelle di cui ognuna ha una colonna `_ID`.

Classi:

- `ContentProvider` classe astratta che incapsula i dati
- `ContentResolver` classe astratta che fornisce l'accesso ai dati.
- `Uri` classe che fornisce un modo per identificare le tabelle e le righe. Sintassi: `content://nomeContentProvider/NomeTabella/id`

Devono essere dichiarati nel manifest.

```
<provider android:name="MyAddressBook"
          android:authorities="it.unipd.dei.esp1112.AddressBookProvider"
          android:exported="true">
  <grant-uri-permission android:pathPattern=".*" />
</provider>
```

Metodi di `ContentProvider`:

- **`boolean onCreate()`**
  Initializes the content provider.
  Advice: lengthy initializations, such as opening an `SQLiteDatabase`, should be deferred until the content provider is actually used

- **`String getType(Uri uri)`**
  Returns the MIME type of data in the content provider at the given URI
- **`Uri insert(Uri uri, ContentValues values)`**
  Inserts a new row in the table at the URI `uri`.
  Returns the URI for the newly inserted item

- **`int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)`**
  Updates all rows matching the `selection` filter in the table (or record) identified by `uri`. New values are contained in `values`, which is a mapping from column names to new column values.
  Returns the number of affected rows

- **`int delete(Uri uri, String selection, String[] selectionArgs)`**
  Deletes all rows matching the `selection` filter in the table (or record) identified by `uri`.
  Returns the number of affected rows
- **`Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`**
  Performs a query among all rows matching the `selection` filter in the table (or record) identified by `uri`.
  `projection` contains a list of columns to put into the result.
  The result of the query is returned as a `Cursor` object

Metodi di `ContentResolver`:

- `ContentResolver`s are not instantiated, but obtained by invoking **`getContentResolver()`** from within an activity or other application component
- Same methods of a `ContentProvider`: `insert()` `update()`, `delete()`, `query()`, ...

Esempio:

```
// Use the Uri class to build the URI
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

// Obtain a content resolver
ContentResolver cr = getContentResolver();

// Query for the specific record
Cursor cur = cr.query(myPerson, null, null, null, null);
// Form an array specifying which columns to return.
// The names of the columns are available as constants in the People class
String[] projection = new String[] {People._ID, People._COUNT,
                                    People.NAME, People.NUMBER};

// Get the base URI for the People table in the Contacts content provider
Uri contacts =  People.CONTENT_URI;

// Make the query
Cursor Cur = getContentResolver().query
                  (contacts,
                   projection, // Which columns to return
                   null,       // Which rows to return (all rows)
                   null,       // Selection arguments (none)
                   People.NAME + " ASC"); // Put results in ascending order by name
```

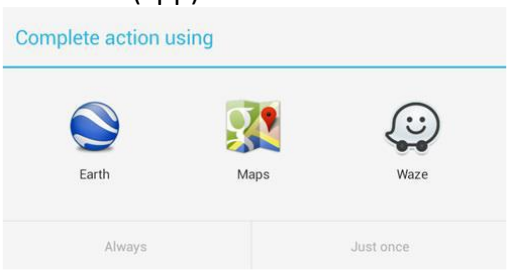
User Interface

Approcci:

- Programmatico: tramite codice Java. Flessibile, UI può essere costruita a run time, non chiaro quando modificare, difficolta nel supporto multilingua.
- Dichiarativo: file esterno XML che contiene una struttura dati. Miglior design, non serve ricompilare l'app, facile per il supporto multilingua e dimensioni dello schermo.
- Misto

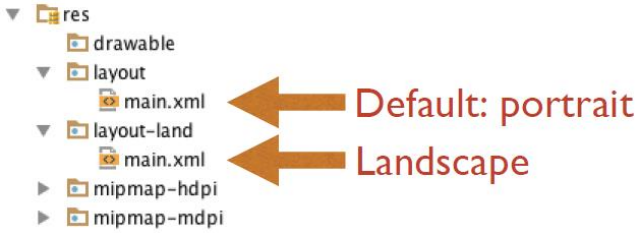
Cambiare User Interface

Siccome ogni interfaccia ha una classe distinta si usa un bottone per richiamare un'altra UI.
Intent: messaggio asincrono che richiede un azione. Può richiedere l'utilizzo di uno specifico componente.
Implicito: richiede un servizio in generale, che poi il sistema o l'utente sceglieranno.
Esplicito: richiede uno specifico servizio (app).



Gestione della Screen Orientation

Si possono creare diversi file XML per una stessa UI mettendoli in cartelle diverse con nomi diversi in base ai **qualifiers**.



7.User Interface

7.1. Linee guida

Informazioni non necessarie non vanno mostrate. L'applicazione dovrebbe rispondere istantaneamente agli input.
L'applicazione deve avviarsi in fretta senza richiedere configurazioni, caricamento di componenti …
Un app dovrebbe servire ad un solo scopo, cercare di evitare di mettere tante funzionalità in un app sola.
Salvataggi: auto-salvare, implementare una funzione di UNDO
Principi:

- **User control:** è un errore togliere all'utente la possibilità di decidere. L'utente si concentra sul contenuto quindi non bisogna oscurarlo.
- **Integrità estetica:** un app deve trasmettere un messaggio riguardo al suo scopo e la sua identità. Cercando di integrare bene l'apparenza con le funzioni.
- **Consistenza:** permette alle persone di fare leva sulle conoscenze e abilità pregresse.
- **Metafora**: gli oggetti, le icone, ecc devono essere familiari col mondo reale.
- **Manipolazione diretta:** l'utente tocca sullo schermo gli oggetti, ci sono delle animazioni (es scrolling), molti-touch gestures.
- **Feedback:** l'utente si aspetta una risposta immediata dalle sue azioni.

7.2. UI in Android

Gestures:

- `boolean onTouch(View v, MotionEvent event)`
  Part of the `View.OnTouchListener` interface.
  The user has performed an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

- `void onClick(View v)`
  Part of the `View.OnClickListener` interface.
  The user has touched the item

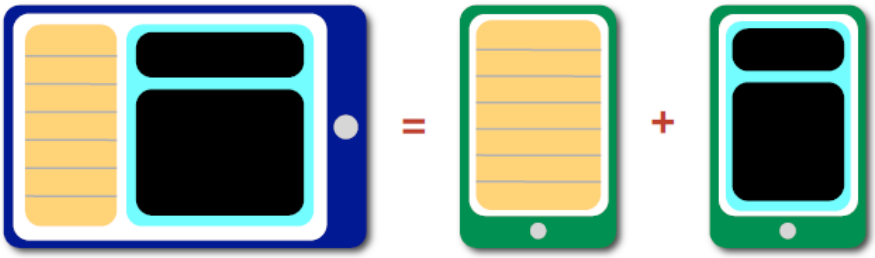
- `boolean onLongClick(View v)`
  Part of the `View.OnLongClickListener` interface.
  The user has touched and holds the item

I widget della UI standard rispondono da soli a delle gestures (es ListView risponde al flik), per i custom widget si può implementare il metodo `onTouchEvent(MotionEvent event)`.

Grandezza dello schermo: di solito richiedono differenti UI, cercare di mettere elementi resizable, e di introdurne di specifici per i tablet.
Grandezze: small (`-small`), normal (`-normal`), large (`-large`), extra large (`-xlarge`)
Orientazioni: portrait (`-port`), landscape (`-land`)
Densità: 120 DPI (`-ldpi`), 160 DPI (`-mdpi`), 240 DPI (`-hdpi`), 320 DPI (`-xhdpi`), 480 DPI (`-xxhdpi`), 640 DPI (`-xxxhdpi`)
Esempi:

- Directory per default layout: "`res/layout`"
- Directory per screen large e landscape: "`res/layout-large-land`"
- Directory per gli artwork: "`res/drawable`"
- Directory per artwork US-English e landscape: "`res/drawable-en-rUS-land`"

Multi-pane layout: modo di creare UI per tablet e telefoni

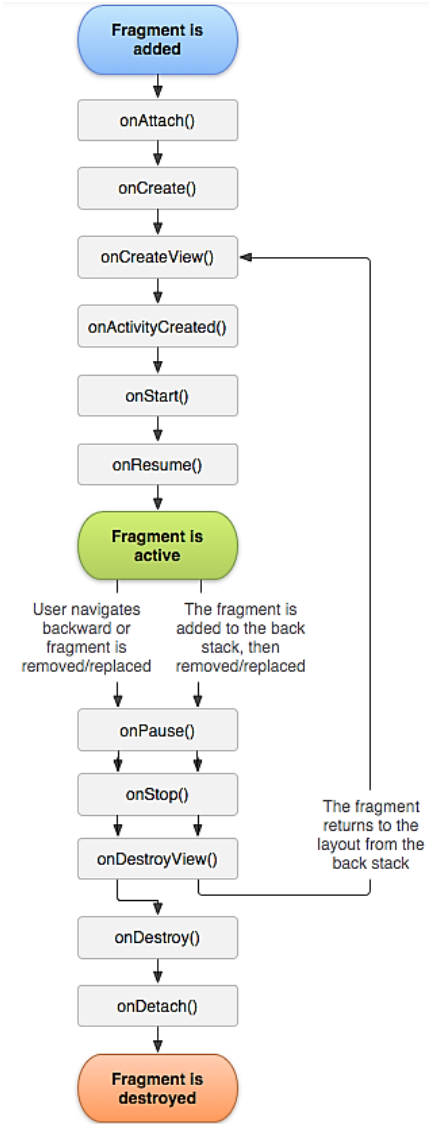


7.2.1. Fragment

Classe `Fragment`

Rappresentano una porzione di UI, sono ospitati da un activity, hanno un loro layout e un loro lifecycle.
I metodi di callback devono essere chiamati dall'activity che li ospita.
Modi di ospitare:

- **Dichiarativo**: aggiungere il fragment al file layout dell'activity che ospita.
- **Programmatico**: aggiungere il fragment nel codice Java dell'activity, istanziare la US nel metodo `onCreateView()` del fragment.



Metodi:

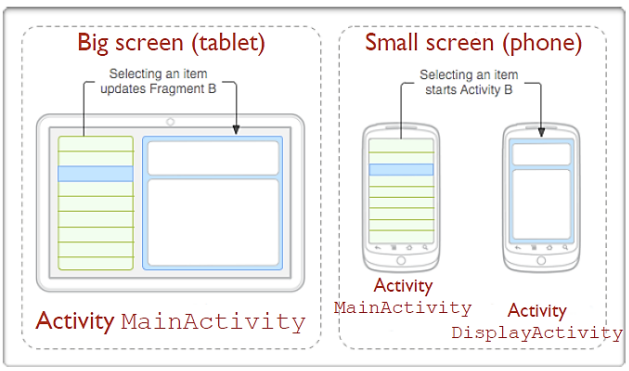
- `View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)`
  Instantiates the UI for a fragment and attaches it to `container`
- An implementation for `onCreateView()` must be provided by the programmer
- If the UI is defined in an XML file, the system-provided `LayoutInflater` can be used to instantiate ("inflate") it
- `void onDestroyView()`
  Destroys a previously-created user interface

Classe `FragmentManager`

Permette l'interazione con i fragment (Es aggiungerne o rimuoverne).

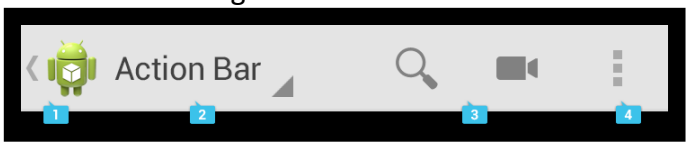
- `Fragment findFragmentById(int id)`
  Returns the fragment which is identified by the given `id` (as specified, e.g., in the XML layout file)
- `FragmentTransaction beginTransaction()`
  Start editing the Fragments associated with the `FragmentManager`. The transaction is ended by invoking the `commit()` method of `FragmentTransaction`

Esempio:



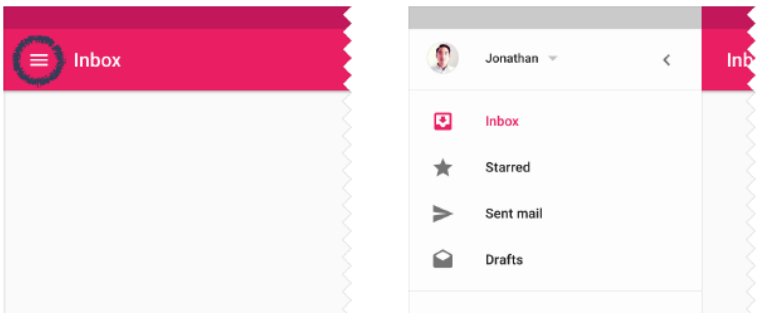
- `DisplayActivity` is started only if the screen is small. It hosts fragment `DetailsFragment`
- `MainActivity` always manages fragment `TitlesFragment` and, depending on the screen size, hosts `DetailsFragment` as well or starts `DisplayActivity`

Action Bar: Componente che contiene l'icona dell'applicazione, un view control (tabs o spinner), alcuni action items, il action overflow menu button. Può anche contenere un navigation drawer. Può essere popolata nel metodo `onCreateOptionsMenu()` che viene chiamano all'avvio dell'activity.
Gli action items e overflow items sono gestiti come menù.
Il metodo `onOptionsItemSelected()` viene chiamato quando un elemento viene cliccato.
Se c'è poco spazio alcuni action item vengono messi nel overflow menu.



App bar: sostituisce la action bar da Android 5.0+. La nav icon può essere una freccia per tornare indietro o un controllo per aprire il navigation drawer.

Navigation Drawer: mostra le opzioni di navigazione.



Overflow menu: gruppo di action items che non sono importanti da mostrare nella action bar.

Ha la stessa funzione del bottone hardware.

Multiple windows: da Android 7.0+ (API 24+) split-screen, Freeform mode. Da API26+ anche Picture in picture mode, cioè mettere un activity in un angolo.
Aggiungere nel manifest gli attributi: `android:resizeableActivity`, `android:supportsPictureInPicture`, `android:minHeight`, `android:minWidth`
Metodi:

- `void onMultiWindowModeChanged (boolean isInMultiWindowMode, Configuration newConfig)`
  Called by the system when the component changes from fullscreen mode to multi-window mode and vice versa

- `void onPictureInPictureModeChanged (boolean isInPIPMode, Configuration newConfig)`
  Called by the system when the component changes to/from picture-in-picture mode

- `void enterPictureInPictureMode()`
  Enters picture-in-picture mode

Ciclo di vita: rimane lo stesso, ma solo un activity è attiva, l'altra è in pausa. Ogni resize viene visto come un cambiamento, cioè l'activity viene killata e ricreata.
Tra activity che condividono lo schermo si possono spostare oggetti.

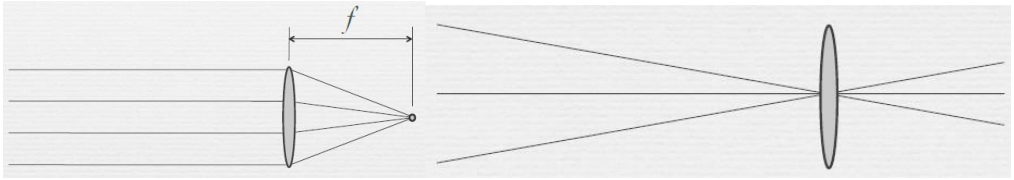
Support library: servono per poter utilizzare librerie non supportate nelle versioni più vecchie di Android.

8. Computational photography

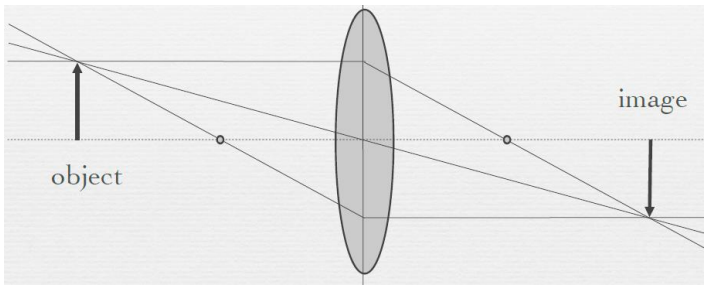
8.1. Formazione dell'immagine

Una macchina fotografica produce la stessa proiezione geometrica planare 2D di una camera oscura, una lente sostituisce il foro stenopeico e il film o un sensore digitale diventa il piano dell'immagine, ruotando la fotocamera (e l'obiettivo) attorno al centro dell'obiettivo si aggiungono o rimuovono i punti di fuga.

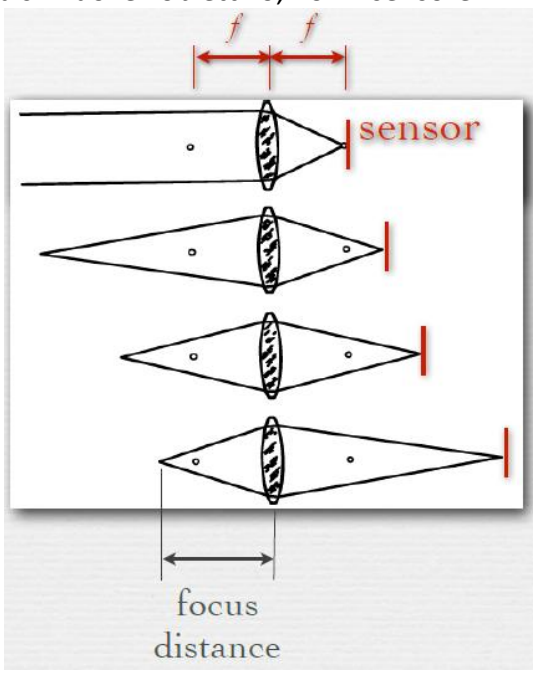
I raggi paralleli convergono in un punto situato alla lunghezza focale f dalla lente.
I raggi che attraversano il centro della lente non sono deviati.



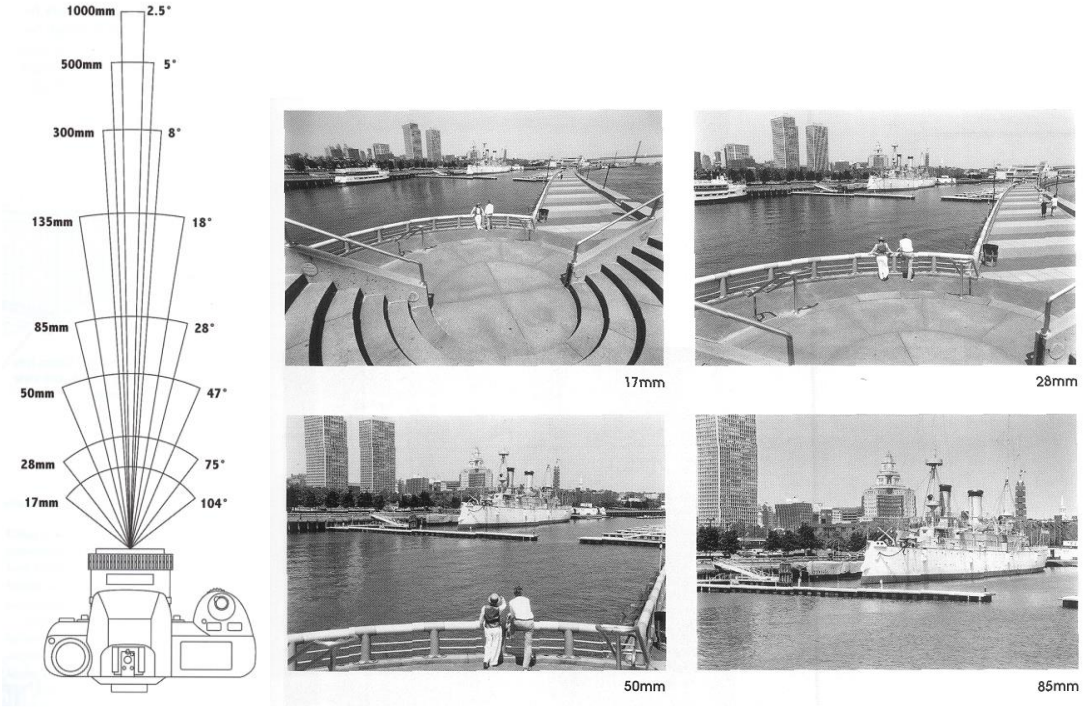
I raggi provenienti da punti su un piano parallelo all'obiettivo sono focalizzati su punti in un altro piano parallelo all'obiettivo.



Per mettere a fuoco oggetti a distanze diverse, spostare il sensore rispetto all'obiettivo. In una fotocamera palmare, in realtà si muove l'obiettivo, non il sensore.

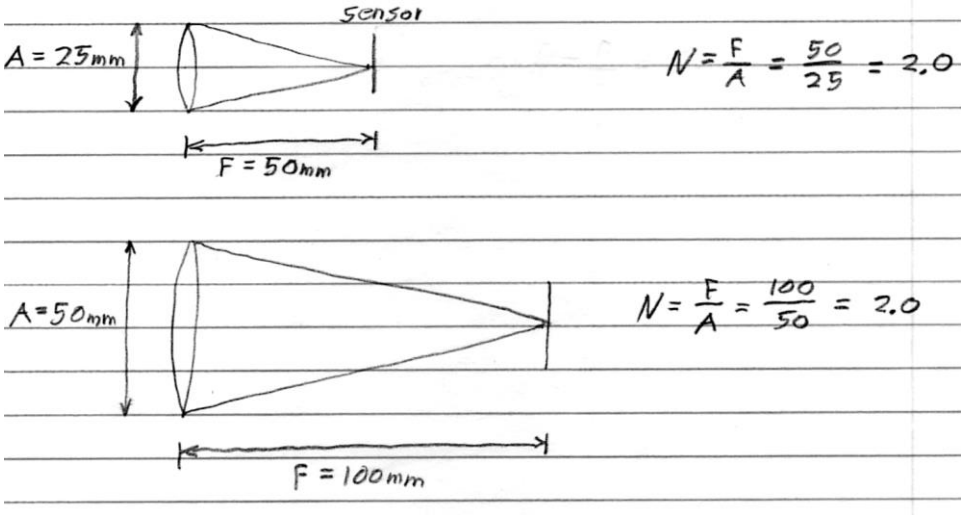


Modificare la lunghezza focale: gli obiettivi più deboli hanno lunghezze focali maggiori, per mantenere lo stesso oggetto a fuoco, spostare il sensore più indietro.
Se la dimensione del sensore è costante, il campo visivo diventa più piccolo.

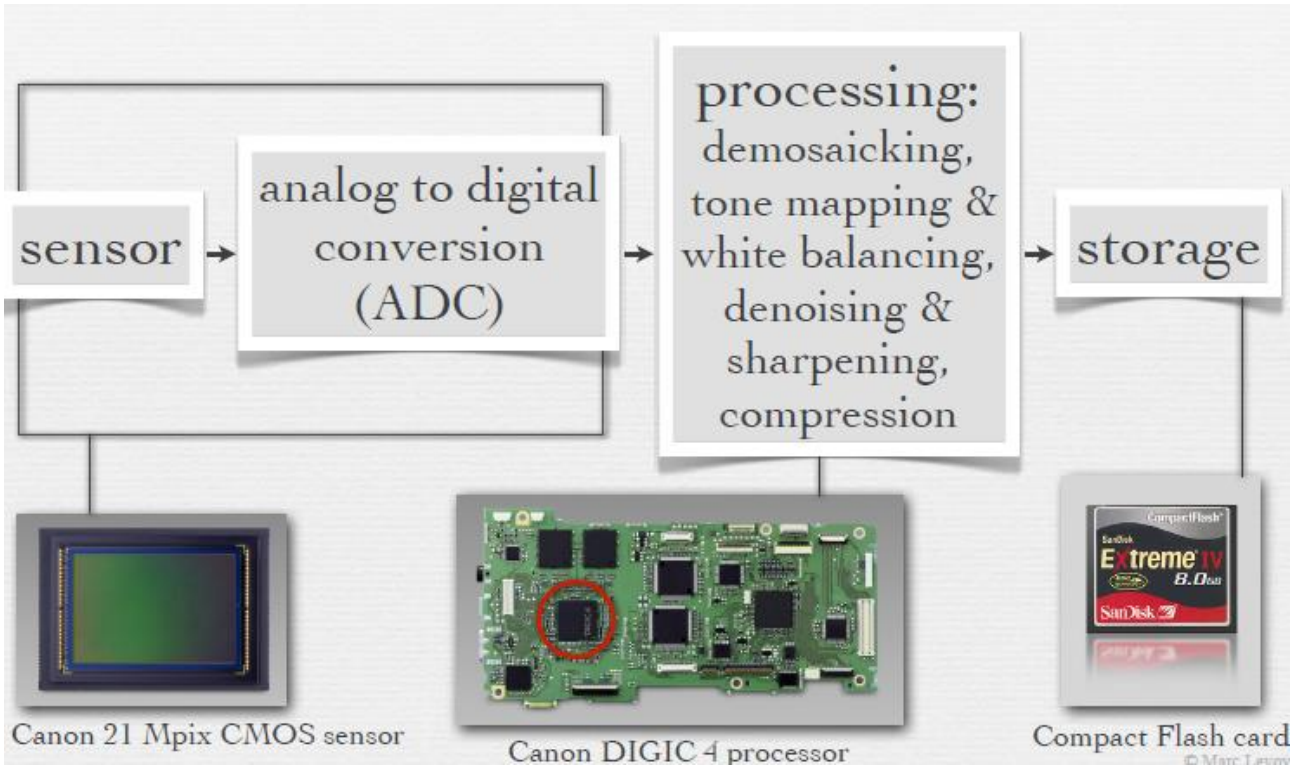


Modificare la dimensione del sensore: se la dimensione del sensore è piccola, anche il campo visivo è più piccolo. I sensori più piccoli hanno meno pixel o pixel più piccoli, che sono più rumorosi.

esposizione = irradianza × tempo
irradianza (E): quantità di luce che cade su un'area unitaria di sensore al secondo, controllata dall'apertura.
tempo di esposizione (T): in secondi, controllato da otturatore (shutter).
Apertura: N=f/A (distanza focale/diametro)

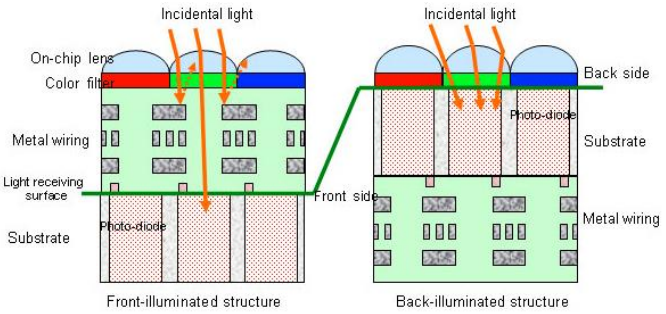


8.2. Catturare Immagini

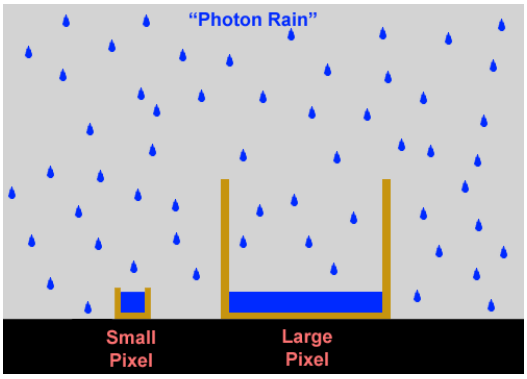


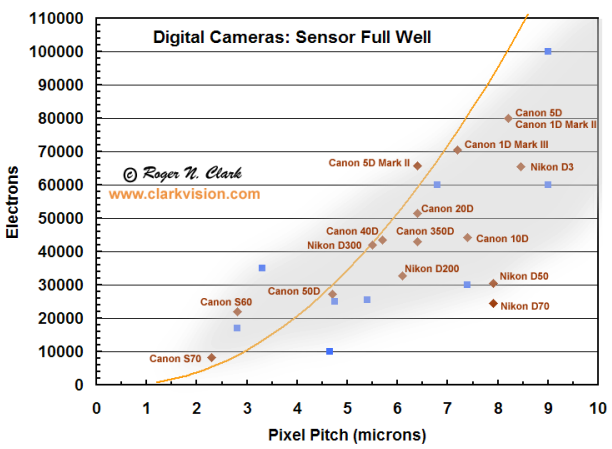
8.2.1. Conversione dei fotoni in carica elettrica

In base al tipo di materiale il sensore è +o- sensibile, si cerca di catturare più fotoni possibili.
Ci sono dei filtri RGB per rilevare solo certe frequenze.



La grandezza dei pixel può variare



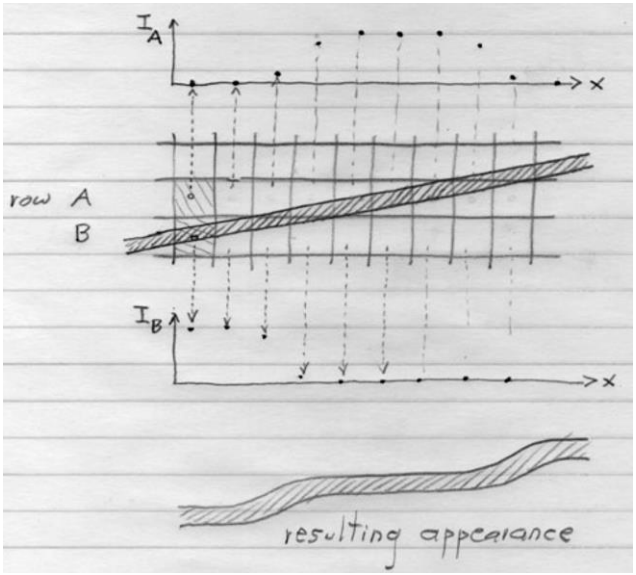


Tecnologie di sensori: CMOS(+utilizzate) e CCD
Effetto blooming: carice forti vicine, es sole.
Effetto smearing: succede nei CCD, si vedono delle righe di luce nelle foto.

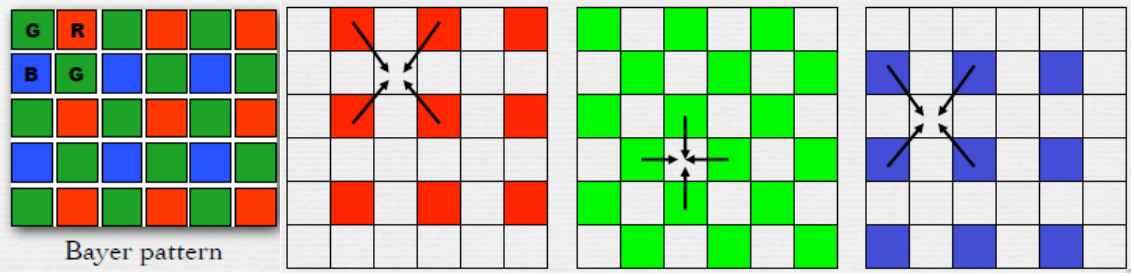
8.2.2. Conversione da carica elettrica a digitale

Servono almeno 10 bit in uscita dall'ADC, le foto vengono salvate nei file RAW.
I file jpeg includono un tone mapping.

Aliasing: alte frequenze la linea di luce è più sottile del pixel, quindi si usa un filtro passa basso, ma l'immagine risulta più sfocata.



Tecnologie del rilevamento del colore: 3-chip oppure Sensori impilati verticalmente: uno sotto l'altro oppure Array di filtri: diversi pattern in cui 2/3 dei colori sono ricavati.



8.2.3. Rumore

La fonte principale è il Photon Shot, ovvero quando il fotone colpisce il sensore.
Dark current: elettroni dislocati da attività termica casuale
Hot pixels: difetti di fabbricazione
Read noise: rumore elettronico durante la lettura dei pixel.
Soluzioni: raffreddare il sensore, sottrarre dark frame o image processing
ISO è l'amplificazione del segnale prima della conversione in digitale

8.3. Pipeline di elaborazione delle immagini

Demosaicizzazione: permette di ricostruire la rappresentazione a colori di un'immagine partendo dai dati grezzi ottenuti dal sensore utilizza un color filter array (CFA). Tramite interpolazione, filtro passa basso.
Rimozione della sfocatura (deblurring): algoritmo ADMM
Correzione del contrasto (tone mapping): output=input^{gamma}
Bilanciamento del bianco:

- scegli un oggetto nella foto che ritieni neutrale (da qualche parte tra bianco e nero) nel mondo reale
- calcolare i fattori di scala (SR, SG, SB) che mappano gli oggetti da (R, G, B) a neutro (R = G = B)
- applica lo stesso ridimensionamento a tutti i pixel dell'immagine rilevata

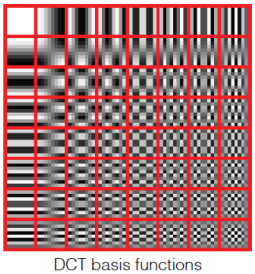
Riduzione del rumore (Denoising):

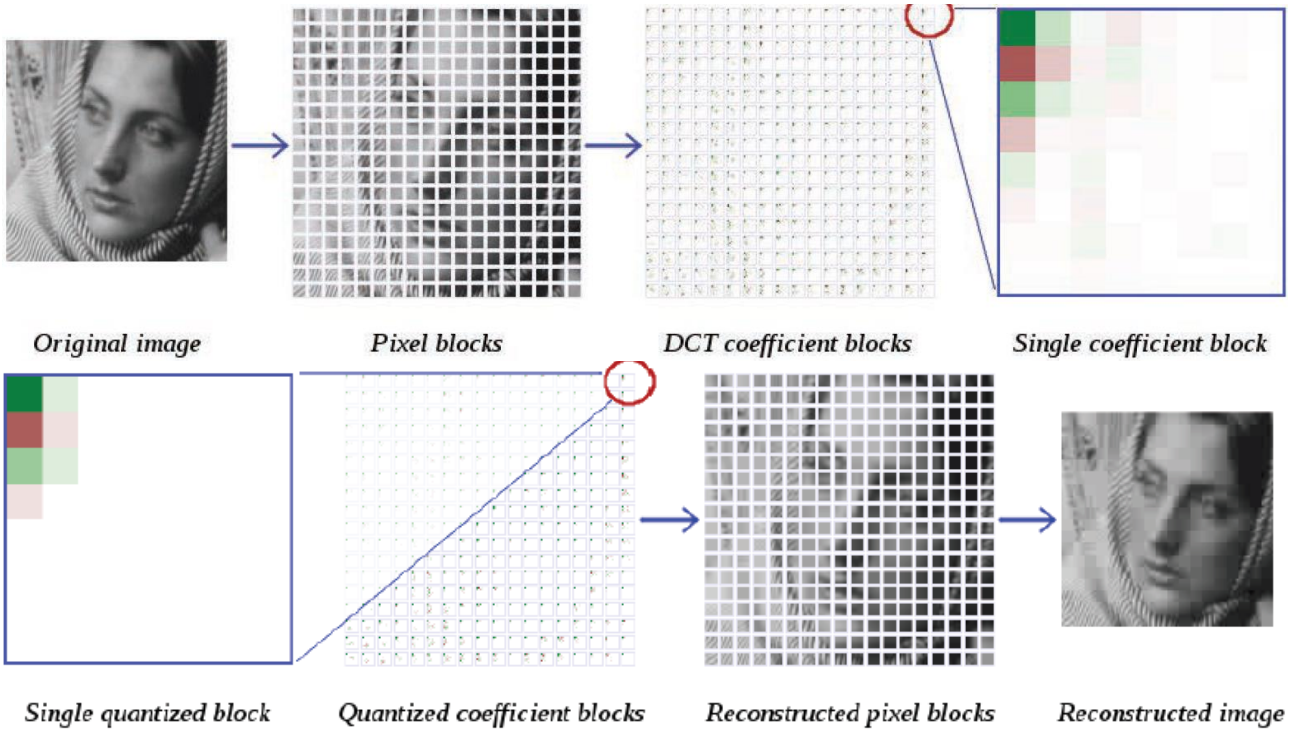
- Idea generale: media un numero di pixel simili per ridurre il rumore
- Local, Linear Smoothing: media dei vicini locali, es utilizzando un filtro passa-basso gaussiano
- Local, Nonlinear Filtering: usa il filtro Mediano(median) nel vicinato locale
- Bilateral Filtering: media nel vicinato locale, ma solo medie intensità simili
- Non-local Means: sfruttare l'auto-somiglianza nell'immagine; pixel medi con un vicinato simile, ma non è necessario che siano vicini

Gamma lineare da 10/12 bit a 8 bit: High dynamic range (HDR), la scena ha un rapporto >100.00:1 dynamic range, JPEG ha 255:1. Scalare i pixel non va bene quindi la compressione gamma viene applicata indipendentemente al rosso, verde e blu, il tone mapping ha gamma=0.5
Sharpening: significa rendere le immagini più nitide. Miscela tra versione originale e contrasto elevato, controllata da una maschera che rappresenta i bordi della scena

Compressione:

- Trasformare nella famiglia di colori YCbCr (Y:luminanza, Cb e Cr: crominanza)
- Sottocampionamento (downsample) delle componenti Cb e Cr
- Dividere in blocchi di 8x8 pixel
- Trasformata coseno discreta (DCT) di ciascun blocco e componente
- Quantizzare i coefficienti
- Codifica entropica


DCT basis functions


Original image
Pixel blocks
DCT coefficient blocks
Single coefficient block
Single quantized block
Quantized coefficient blocks
Reconstructed pixel blocks
Reconstructed image