# Introducció a la programació de dispositius mòbils

Eduard García Sacristán, Joan Climent Balaguer

**Programació multimèdia i dispositius mòbils**

# Índex

## Introducció

Un dels mercats amb major creixement en els darrers anys ha estat el de l'anomenada telefonia intel·ligent. Un dels aspectes fonamentals d'aquesta ha estat l'ampli ventall d'aplicacions disponibles per aquests dispositius, que tenen unes característiques de maquinari avançades. Android és un dels sistemes operatius punters en aquest àmbit i amb el qual és possible crear aplicacions de manera lliure i gratuïta. En aquesta unitat us endinsareu en la programació de dispositius mòbils i multimèdia en Android.

En l'apartat "Entorns de desenvolupament d'aplicacions mòbils" veureu una introducció al món de la programació de dispositius mòbils a l'actualitat, les característiques d'aquest tipus de programació i el sistema Android, i com instal·lar i configurar l'entorn de desenvolupament amb el que treballareu la resta del mòdul. També fareu el vostre primer programa.

En l'apartat "Programació de dispositius mòbils" veureu els conceptes bàsics que s'han de dominar per poder realitzar aplicacions en dispositius mòbils. Estudiareu el cicle de vida de l'aplicació i els components que es fan servir més habitualment en les aplicacions d'Android.

Per seguir els continguts d'aquest mòdul, és convenient anar fent les activitats i els exercicis d'autoavaluació i llegir els annexos (si n'hi ha). Tot i que les unitats formatives tenen un contingut important des del punt de vista conceptual, sempre s'ha procurat donar-los un enfocament pràctic en les activitats proposades.

## Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Aplica tecnologies de desenvolupament per dispositius mòbils avaluant les seves característiques i capacitats.
    - Analitza les limitacions que planteja l'execució d'aplicacions als dispositius mòbils.
    - Descriu diferents tecnologies de desenvolupament d'aplicacions per dispositius mòbils.
    - Instal·la, configura i utilitza entorns de treball pel desenvolupament d'aplicacions per dispositius mòbils.
    - Descriu configuracions que classifiquen els dispositius mòbils segons les seves característiques.
    - Descriu perfils que estableixen la relació entre el dispositiu i l'aplicació.
    - Analitza l'estructura d'aplicacions existents per dispositius mòbils identificant les claus utilitzades.
    - Realitza modificacions sobre aplicacions existents.
    - Utilitza emuladors per comprovar el funcionament de les aplicacions.
2. Desenvolupa aplicacions per dispositius mòbils analitzant i fent servir les tecnologies i llibreries específiques.
    - Genera l'estructura de classes necessària per l'aplicació.
    - Analitza i utilitza les classes que modelen finestres, menús, esdeveniments i controls per al desenvolupament d'aplicacions gràfiques senzilles.
    - Realitza proves d'interacció usuari-aplicació per optimitzar les aplicacions desenvolupades a partir d'emuladors.
    - Documenta els processos necessaris pel desenvolupament de les aplicacions.

# 1. Entorns de desenvolupament d'aplicacions mòbils

Fa uns anys, un directiu d'una gran companyia d'informàtica va profetitzar el començament de l'*època post-PC*, un moment en què els usuaris deixaran de fer servir els ordinadors tal com els coneixem ara (bàsicament ordinadors de taula i ordinadors portàtils) per fer les tasques més habituals del dia a dia: comunicar-se (per correu electrònic, missatgeria o videoconferència), navegar per Internet o consumir continguts.

Aquest moment ja ha arribat, i la varietat de dispositius que tenim actualment ens permet realitzar fàcilment totes les nostres tasques diàries. De fet, la línia que separa uns dispositius d'uns altres cada vegada es fa més difícil de definir: els telèfons tenen cada vegada pantalles més grans, al mateix temps que les tauletes tàctils disposen de diferents mides, de forma que la diferència entre un telèfon gran i una tauleta petita és merament il·lusòria (com la diferència entre portàtils i *netbooks*). Per això, els sistemes operatius i la manera de programar-hi s'ha integrat, de forma que tots aquests dispositius mòbils multimèdia comparteixen les mateixes característiques i han integrat els mateixos sistemes i programari.

## 1.1 El mercat actual d'aplicacions mòbils

Avui en dia la *batalla dels dispositius mòbils* està més d'actualitat que mai. Apple va revolucionar el mercat de la telefonia mòbil amb el seu iPhone, deixant una sèrie de *víctimes* pel camí com van ser Palm, Windows CE i els telèfons de Nokia amb Symbian. Tota la resta del mercat ha estat cercant durant anys l'anomenat *iPhone Killer* (assassí de l'iPhone) o el telèfon que desbancaria el famós telèfon intel·ligent de la companyia de la poma com a telèfon amb més èxit. Finalment, ens hem trobat que no ha existit aquest telèfon, sinó que l'èxit d'altres plataformes (com ara Android, adquirit per Google l'any 2005) ha consistit a establir tot un ecosistema de dispositius compatibles amb què combatre la quota de mercat dels dispositius Apple. Els altres actors d'aquesta competició de mercat (Blackberry de RIM, Firefox OS de Mozilla, Windows Phone de la mà de Microsoft i Ubuntu Touch OS de Canonical, entre d'altres) tenen la difícil tasca de trobar el seu espai en un mercat on cada vegada és més difícil trobar visibilitat.

Tot i no vendre tants telèfons com altres companyies, els beneficis d'Apple superen amb escreix els de les altres companyies.

Actualment, el mercat de fabricants de mòbils està copat amb companyies que obtenen gran part dels seus beneficis de dispositius amb Android. Entre aquestes trobem les companyies que ja desenvolupaven telèfons abans de l'era iPhone com ara Samsung, LG o Sony i companyies que s'han incorporat posteriorment com ara Xiaomi (Mi), Oneplus o Lenovo per exemple.

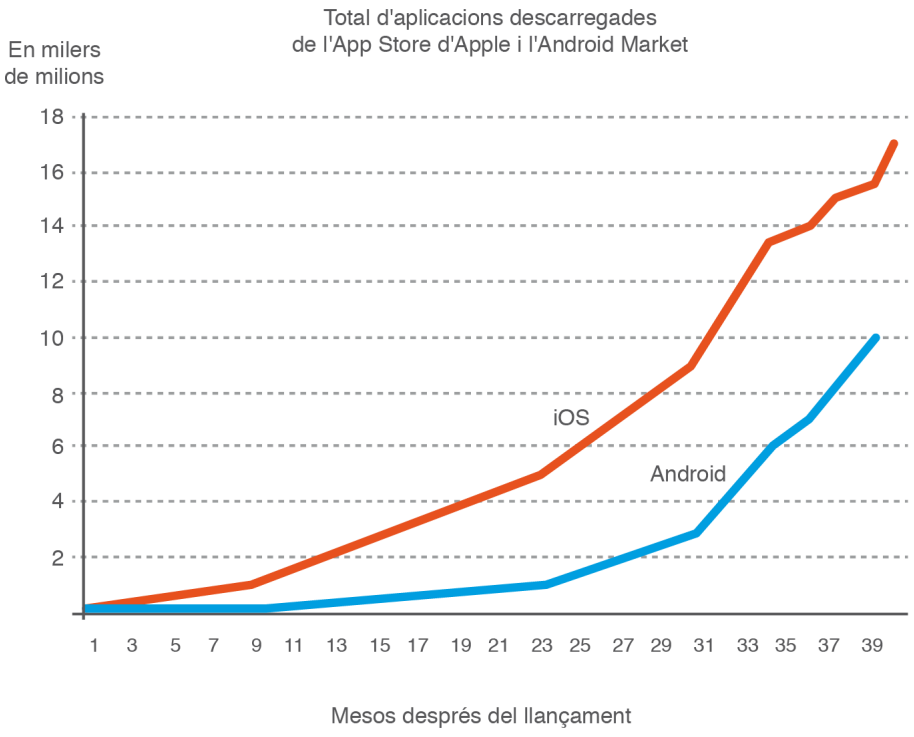
Quant als sistemes operatius dels anomenats telèfons intel·ligents, es pot veure la quota de mercat el desembre de 2014 a la taula 1.1.

**Taula 1.1.** Quota de mercat dels diferents sistemes operatius en 'smartphones' (desembre de 2014)

| Sistema operatiu | Quota de mercat |
|---|---|
| Android | 84,4% |
| iOS | 11,7% |
| Windows Phone | 2,9% |
| Blackberry | 0,5% |
| Altres | 0,6% |

Una de les característiques que més va afiançar Apple amb el seu iPhone va ser la inclusió d'un *mercat d'aplicacions* integrat al propi dispositiu, on els usuaris poden cercar, analitzar i comprar aplicacions d'una manera molt senzilla. Aquest ecosistema d'aplicacions s'ha demostrat com un dels valors afegits més importants en els actuals dispositius mòbils, atès que un aparell amb un gran maquinari té una utilitat molt limitada si no ve acompanyat d'un programari que en faci un servei adequat. En aquest sentit, Google va entendre la importància de les aplicacions i ha facilitat la creació d'aplicacions i la seva publicació en llur mercat d'aplicacions. Així, a pesar d'haver llançat la seva plataforma d'aplicacions més tard, la quantitat d'aplicacions descarregades creix a un ritme molt alt, tal com es pot veure a la figura 1.1.

**Figura 1.1.** Aplicacions descarregades des del llançament de la plataforma



### 1.1.1 Història dels telèfons intel·ligents

El mercat de la telefonia mòbil va deixar de ser fa temps una promesa per establir-se com una de les principals formes de comunicació al mercat de consum actual. Així, si en un principi els telèfons ens permetien realitzar trucades telefòniques i enviar missatges, a poc a poc s'han anat introduint noves característiques i funcionalitats als dispositius mòbils fins arribar al punt actual, on és possible portar a la butxaca un autèntic ordinador personal.

Les tecnologies de transmissió sense fils **GPRS** (*General packet radio service*, servei de ràdio de paquets general) i **WAP** (*Wireless Application Protocol*, protocol d'aplicacions sense fil), un protocol estàndard per accedir a informació a través de xarxes sense fil, van proveïr unes primeres funcionalitats de navegació web als primers telèfons mòbils. La poca velocitat d'aquelles xarxes i, sobretot, el fet que l'experiència d'usuari no fos prou bona degut a la falta de pantalles de qualitat als telèfons, van fer que aquestes tecnologies (i els serveis creats per a elles) no es popularitzessin en excés.

Al final dels anys noranta, la majoria dels telèfons encara tenien unes funcionalitats molt limitades, i aquells usuaris que tenien necessitats més enllà del que podia oferir el seu telèfon també carregaven amb un dispositiu tipus **PDA** (*Personal Digital Assistant*, assistent digital personal), que oferia funcionalitats avançades, com ara: agenda, pantalles tàctils, expansió amb targetes de memòria, connexió sense fils i una sèrie d'aplicacions pròpies de la plataforma. La primera PDA de la història va ser la Psion Organizer desenvolupada l'any 1984 (figura 1.2). Aquestes PDA portaven un sistema operatiu propi, com per exemple Palm OS, BlackBerry OS o Pocket PC. Les darreres versions d'aquests sistemes integraven capacitats de missatgeria i telefonia. Aquests dispositius es podrien anomenar ja *smartphones* (telèfons intel·ligents).

**La primera PDA**

La primera PDA de la història va ser la Psion Organizer desenvolupada l'any 1984. (Psion és una companyia que va desenvolupar programes per al *mític* ordinador ZX Spectrum pels volts dels anys vuitanta). Tot i això, la primera vegada que es va fer servir l'expressió "PDA" va ser l'any 1992 amb la presentació del'Apple Newton.

**FIGURA 1.2.** Psion Organizer, la primera PDA de la història

Un ***smartphone*** (telèfon intel·ligent) és un telèfon mòbil de gamma alta construït sobre una plataforma mòbil amb capacitats i connectivitat superiors a les que té un telèfon convencional.

Encara que el mercat de les PDA estava dominat per l'empresa Palm (fabricant tant del maquinari com del sistema operatiu mòbil), Microsoft va treure al mercat Windows Mobile, basat en la plataforma Pocket PC 2000 que, gràcies a la possibilitat de llicenciar-lo per part d'altres empreses, va ser molt popular i l'any 2007 va obtenir un 42% del mercat.

Justament aquest any 2007, Apple va introduir l'iPhone (després de mesos de rumors i especulacions), que revolucionaria el mercat de telefonia mòbil. Moltes de les seves seves característiques s'han convertit en estàndards de facto a la indústria de telefonia mòbil, com ara:

- Pantalla capacitiva i multitàctil
- GPS
- Teclat en pantalla
- Integració amb alguns serveis de Google, especialment la cerca i Google Maps
- Connexió de dades permanent mitjançant 3G

**Posicionament global**

GPS (Global Positioning System, sistema de posicionament global) és un sistema de navegació basat en satèl·lits, que proporciona localització i hora en qualsevol tipus d'horatge, en qualsevol lloc de la terra, si es té visió directa de quatre o més satèl·lits GPS.

I un parell de característiques que no estaven a l'iPhone original però que es van afegir més endavant, com ara:

- Aplicacions de tercers
- Una botiga d'aplicacions on trobar totes les aplicacions de forma centralitzada (AppStore)

La resta de la indústria va trigar anys a oferir un producte que pogués competir amb l'iPhone (figura 1.3) tant a nivell de *software* com de *hardware*. Es va produir durant molt de temps una lluita per aconseguir restar quota de mercat a Apple en el sector dels *smartphones*.

**Figura 1.3.** L'iPhone original


Però el desenvolupament d'Android va començar, de fet, abans que l'iPhone veiés la llum. Google va comprar la companyia Android Inc. el 2005 i la primera distribució d'Android (el sistema operatiu mòbil) es va anunciar el 2007 per l'**Open Handset Alliance**. Des d'aleshores, la quantitat de fabricants de telèfons que fan servir Android com a sistema operatiu del seu maquinari (telèfons i tauletes tàctils, principalment, encara que existeixen multitud de dispositius que fan servir Android) ha crescut de forma exponencial. El novembre de 2011 existien més de 200 milions de dispositius Android, i Google va anunciar que el desembre de 2011 s'activaven més de 700.000 dispositius Android diaris; aquest índex ja arribava als 2 milions anuals el 2013 (font: www.talkandroid.com).

Una altra fita important en l'evolució dels dispositius mòbils va ser la irrupció de l'iPad, un *tablet* (tauleta tàctil) d'Apple que va sortir al mercat el 2010 i en molt poc temps va popularitzar l'ús d'aquest tipus de dispositius.

**Qui s'encarrega de crear els estàndards?**

L'Open Handset Alliance (que es podria traduir per Aliança d'auriculars oberts) és un consorci de 86 empreses de maquinari, programari i telecomunicacions dedicades a avançar estàndards oberts per a dispositius mòbils.

> Un ***tablet*** és un ordinador portàtil, més gran que un telèfon mòbil o una PDA, integrat dins d'una pantalla tàctil plana i principalment operat mitjançant la seva pantalla tàctil (en lloc d'un teclat físic). Generalment, fa servir un teclat virtual en pantalla o un *stylus* per introduir text. El primer *tablet* que va gaudir d'un gran èxit comercial va ser l'iPad d'Apple.

Actualment existeixen diverses solucions de sistemes operatius per a dispositius portàtils. En aquesta llista anomenem les més importants, encara que el mercat està dominat per les dues primeres:

- Android: desenvolupat per Google i l'Open Handset Alliance. La majoria dels fabricants més importants (Samsung, HTC, Motorola, Sony…) tenen telèfons i tauletes que s'executen sobre Android. Té un 45,86% del mercat (dades de desembre de 2014).

- iOS: desenvolupat per Apple, està present únicament als productes de la marca Apple (iPhone, iPad i iPod); així i tot, té una presència del 43,15% al mercat, gràcies a la bona salut de vendes del sector de tauletes.

- BlackBerry OS: desenvolupat per RIM (*Research In Motion*), i present únicament als seus productes; ha baixat molt la seva quota de mercat, fins a l'1% actual.

- Windows Phone: desenvolupat per Microsoft i present fonamentalment als nous telèfons que Nokia va presentar a finals de 2011; la seva quota de mercat és molt petita actualment tot i que ha anat creixent gràcies a la seva major presència al món de les tauletes.

- Firefox OS: desenvolupat per Mozilla, és un sistema operatiu de codi obert que es basa en HTML5. L'impacte d'aquest sistema operatiu ha estat mínim a causa de la poca diversitat d'aplicacions disponibles.

En aquest materials hem decidit fer servir Android (figura 1.4) com a eina de desenvolupament per les següents característiques:

- És un projecte de codi lliure.

- És gratuït, i les seves eines de desenvolupament també ho són.

- Actualment és (juntament amb Apple iOS) la plataforma de dispositius mòbils més popular del mercat, amb una previsió de creixement encara més gran (és probable que en el futur la quota de mercat d'Android sigui encara superior a l'actual).

- El seu desenvolupament es realitza en Java, llenguatge molt popular i de fàcil aprenentatge.

**Figura 1.4.** Logo d'Android

Per tot això, pensem que Android és la plataforma ideal per introduir-se en el desenvolupament de la programació en dispositius mòbils i multimèdia.

### 1.1.2 Característiques d'Android

Android té una llicència de codi lliure i està disponible de forma gratuïta per als fabricants per a la seva adaptació. No existeixen configuracions fixes de maquinari i programari, encara que Android suporta les següents característiques:

- Missatgeria: suporta SMS (*Short Message Service*, servei de missatges curts) i MMS (*Multimedia Messaging Service*, servei de missatgeria multimèdia).
- Suport de maquinari: acceleròmetres, càmera, brúixola, sensor de proximitat i GPS.
- Emmagatzematge: fa servir SQLite, una base de dades relacional ultralleugera, per a l'emmagatzematge de dades.
- Formats: suporta multitud de formats multimèdia (imatges, so, vídeo...).
- Connectivitat: suporta xarxes GSM/EDGE, HSDPA, LTE, CDMA, UMTS, Bluetooh, Wi-fi i WiMAX.

En aquests moments, Android es fa servir en multitud de dispositius, com ara telèfons intel·ligents, *tablets*, lectors de llibres electrònics, *netbooks*, reproductors d'MP3 i MP4, televisors, rellotges...

L'Android OS (*sistema operatiu d'Android*) està dividit en cinc seccions. De la capa més baixa a la més alta, són les següents:

- **El kernel de Linux**: sobre el qual està basat Android. Aquesta capa conté tots els *drivers* de dispositius (a baix nivell) per als components de maquinari del dispositiu Android.
- **Llibreries (biblioteques de funcions**): conté el codi que proporciona les característiques principals de l'Android OS. Per exemple, les biblioteques de funcions de Webkit proporcionen funcionalitats de navegació web, i les d'SQLite d'accés a bases de dades.
- **Android Runtime:** està al mateix nivell que les llibreries i proporciona una sèrie de llibreries fonamentals que permeten als desenvolupadors escriure aplicacions d'Android fent servir Java. El *runtime* d'Android també inclou la màquina virtual Dalvik, que permet a cada aplicació Android executar-se en el seu propi procés (que és una instància d'aquesta màquina virtual). Les aplicacions Android estan compilades en executables Dalvik. A partir de la versió 5.0 (Lollipop) s'ha canviat la màquina virtual de Dalvik a ART, incrementant el rendiment general de les aplicacions.
- **Framework d'aplicacions:**proporciona als desenvolupadors les diferents característiques de l'Android OS per tal que les puguin fer servir en les seves aplicacions.

**ART vs Dalvik**

Mentre que Dalvik compila *just-in-time (JIT)* ART ho fa *ahead-of-time (AOT)*, és a dir, Dalvik compila les aplicacions al moment d'executar-les i ART ho fa en el moment de la instal·lació.

- **Aplicacions:** a la capa més alta, hi trobem les aplicacions que es distribueixen amb el dispositiu Android (com telèfon, contactes, navegador, etc.), així com altres aplicacions que es descarreguen i s'instal·len des de la botiga de Google, *Google Play Store*. Les aplicacions que escriurem estaran situades en aquesta capa.

### 1.1.3 Google Play

Un dels factors fonamentals que determina l'èxit d'una plataforma mòbil és la quantitat d'aplicacions que ofereix. Això va quedar demostrat amb l'iPhone d'Apple i el seu ecosistema d'aplicacions. De la mateixa manera, el fet que sigui fàcil per als desenvolupadors publicar les seves aplicacions i fàcil per als usuaris comprar-les determinarà la facilitat amb què creixerà el mercat d'aplicacions.

L'agost del 2008, Google va anunciar l'Android Market, el seu propi mercat d'aplicacions, que va ser rebatejat com a Google Play el març de 2012 (https://play.google.com/store). Els usuaris poden descarregar aplicacions de tercers directament des del seu dispositiu (a través de l'aplicació Play Store, preinstal·lada a la majoria de dispositius Android). Google Play disposa d'aplicacions de pagament, així com d'altres completament gratuïtes o gratuïtes amb micropagaments integrats.

El cicle de vida d'una aplicació d'Android es podria definir de la següent manera:

1. Descobriment de l'aplicació, tant des de Google Play (el més habitual) com d'una altra forma.
2. Descàrrega i instal·lació de l'aplicació.
3. Execució de l'aplicació.
4. Actualització de l'aplicació si existeixen versions més modernes (opcional).
5. Desinstal·lació de l'aplicació.

## 1.2 Dispositius mòbils

En primer lloc, hem de determinar què és programació per a dispositius mòbils i, per tant, què considerem com a dispositiu mòbil. En general, considerem que un dispositiu és *mòbil* si té les següents característiques:

- Té una mida reduïda i un pes lleuger. Per tant, la mobilitat és una de les seves principals característiques. Idealment, és un dispositiu que es pot portar a la butxaca o a una bossa petita i al que es pot accedir ràpidament.

- Té una connexió sense fils contínua en forma de connexió WiFi o connexió a Internet permanent fent ús de les xarxes de telefonia.
- Proveeix una alta capacitat d'interacció a l'usuari mitjançant una pantalla tàctil o una pantalla i un teclat.
- Té capacitat de processament i memòria interna.

Així, considerem dispositius mòbils els telèfons mòbils, els telèfons intel·ligents (*smartphones*), les PDA i els *tablets*. Descartem els ordinadors portàtils com a dispositius mòbils, atès que donats el seu pes i la seva mida són dispositius que no es poden portar a la butxaca.

Actualment existeixen també els dispositius *wearables*, que tenen menors dimensions que els dispositius mòbils i es sincronitzen amb els *smartphones* mitjançant la tecnologia *bluetooth* per afegir funcionalitats al sistema. En aquest sector trobem dispositius com els *smartwatch* o les polseres intel·ligents.

Google ha creat un sistema operatiu basat en Android per a aquest tipus de dispositius, *Android Wear*, que va ser presentat el 2014.

### 1.2.1 Limitacions en la programació de dispositius mòbils

Els dispositius mòbils tenen unes característiques particulars quant a les seves capacitats de processament i memòria. El fet que aquests dispositius siguin principalment mòbils i, per tant, d'unes mides i pes reduïts, determina les seves característiques de maquinari. Principalment, les seves limitacions, comparant-los amb els dispositius de taula, són les següents:

- La seva capacitat de processament és reduïda.
- La quantitat de memòria és reduïda (RAM i d'emmagatzematge).
- Les dimensions del dispositiu són reduïdes, i per tant el maquinari d'interacció amb l'usuari (pantalla i teclat) també ho és.
- Solen tenir una connexió permanent a Internet, i per tant s'ha de donar especial importància a la seguretat.
- La connexió a Internet, si és mòbil, té una velocitat limitada i es perd amb freqüència.

Totes aquestes característiques es van millorant a poc a poc, i els nous dispositius cada vegada tenen més capacitat de processament, més memòria, pantalles més grans i millors connexions a Internet. Així i tot, sempre aniran per darrere dels ordinadors de taula, i per tant s'han de tenir en compte les seves característiques quan programem sobre aquests dispositius.

## 1.3 Entorns de desenvolupament

El primer pas per començar a desenvolupar programes per a dispositius mòbils és instal·lar l'**Android SDK** (*Software Development Kit*, equip de desenvolupament de programari) i preparar l'entorn de desenvolupament per primera vegada.

Aquests passos poden canviar amb el temps, i estan especificats a la web d'Android (developer.android.com/sdk/installing.html); per tant, es recomana revisar que el procediment segueix sent vàlid.

Es poden desenvolupar aplicacions per a dispositius mòbils amb Android des de qualsevol sistema operatiu (GNU/Linux, MacOS, MS Windows) i amb diversos entorns de desenvolupament; l'oficial i més popular actualment és l'**Android Studio** tot i que també es poden desenvolupar aplicacions en Eclipse i en Netbeans. Escollirem per als materials desenvolupar amb Ubuntu 14.04 LTS 64 bits (*Long Term Support*, les versions LTS són suportades durant quatre anys per Ubuntu) i Android Studio.

Es recomana la utilització d'un sistema operatiu de 64 bits, Google ens informa que els arxius binaris de 32 bits desapareixeran en un futur. Si us és impossible fer servir una màquina de 64 bits podeu afegir l'opció `-force-32-bit` a *Run/Edit Configurations* a la pestanya de l'emulador o bé fer un `export ANDROID_EMULATOR_FORCE_32BIT=true` des del terminal; si voleu fer persistent aquest darrer pas, podeu crear un fitxer `.profile` al vostre directori *home* i afegir aquí l'`export`.

PhoneGap és un *framework open source* per desenvolupar aplicacions multiplataforma amb *HTML5*, *CSS* i *JavaScript*.

Cal fer menció a més, de l'existència de *frameworks* (entorns de treball) que ens permetran crear aplicacions per Android fent servir altres llenguatges de programació com ara: *C#*, *python* i fins i tot amb una combinació d'*HTML5* + *CSS* + *JavaScript*. Alguns exemples podrien ser: *Xamarin*, *Phonegap*, *dot42*, *Kivy*, etc.

**Android Studio**

Android Studio és un IDE (*Integrated Development Environment*, entorn de desenvolupament integrat) basat en l'IntelliJ IDEA que ha substituït oficialment a l'Eclipse com a eina per desenvolupar aplicacions.

### 1.3.1 Preparant l'ordinador

En primer lloc, heu de confirmar que el vostre ordinador compleix els requeriments del sistema per desenvolupar en Android. Hem de dir que les exigències sobre les característiques de la màquina són elevades (especialment per a l'execució del simulador). Els requeriments del sistema poden canviar al llarg del temps. Podeu trobar els requeriments actualitzats al web oficial d'Android: developer.android.com/sdk/requirements.html.

Els requeriments mínims han de ser:

Comuns:

- 2 GB de RAM com a mínim, 4 recomanats
- 2 GB lliures al disc dur
- 1280x800 de resolució de pantalla
- Oracle Java Development Kit (JDK) 7

Linux:

- Escriptori GNOME o KDE
- Llibreria CNU C (glibc) 2.11 o posterior

Windows:

- Microsoft Windows 8/7/Vista/2003 (32 o 64 bits)
- Opcional per a tenir acceleració per *hardware*: processador Intel amb Intel VT-x, Intel EM64T i Execute Disable (XD) bit

Mac OS X:

- Mac OS X 10.8.5 o superior fins a 10.9
- Java Runtime Environment (JRE) 6
- Opcional per a tenir acceleració per *hardware*: processador Intel amb Intel VT-x, Intel EM64T i Execute Disable (XD) bit

Entorn de desenvolupament suportat, Android Studio:

- Android Studio 1.0 o superior
- JDK 7 (Java Development Kit, equip de desenvolupament de Java) com a mínim. Amb el JRE (Java Runtime Environment, entorn d'execució de Java) no és suficient

La forma d'instal·lar l'entorn de desenvolupament també pot canviar amb les noves versions, per tant es recomana fer una ullada a la pàgina oficial d'Android per veure les instruccions d'instal·lació: developer.android.com/sdk/installing.html.

Per instal·lar el JDK a Ubuntu, de nou podeu revisar la darrera informació continguda a la documentació oficial: https://help.ubuntu.com/community/Java

Per defecte, la màquina de Java d'Oracle ja no està distribuïda amb Ubuntu per problemes de llicències. Les versions actuals d'Ubuntu porten instal·lada la implementació OpenJDK.

L'objectiu principal del projecte OpenJDK és produir una implementació de codi obert de la plataforma Java Standard Edition.

Podeu descarregar el JDK des de la pàgina oficial d'Oracle en format .tar.gz o .rpm però per facilitar la instal·lació ho farem des del repositori de webupd8 tal com ens recomanen en la documentació oficial d'Ubuntu.

Així, afegim el repositori ppa amb la següent comanda:

En cas de no disposar de la comanda **add-apt-repository** instal·leu python-software-properties: `sudo apt-get install python-software-properties`

```
sudo add-apt-repository ppa:webupd8team/java
```

Després actualitzarem la llista de repositoris i farem la instal·lació d'*oracle-java8-installer*:

```
sudo apt-get update
  sudo apt-get install oracle-java8-installer
```

Una vegada instal·lat, podrem alternar entre les versions de Java amb la comanda `sudo update-alternatives -config java` on podrem escollir entre l'OpenJDK i el JDK d'Oracle. Podeu comprovar quina versió de Java teniu instal·lada amb la comanda `java -version`.

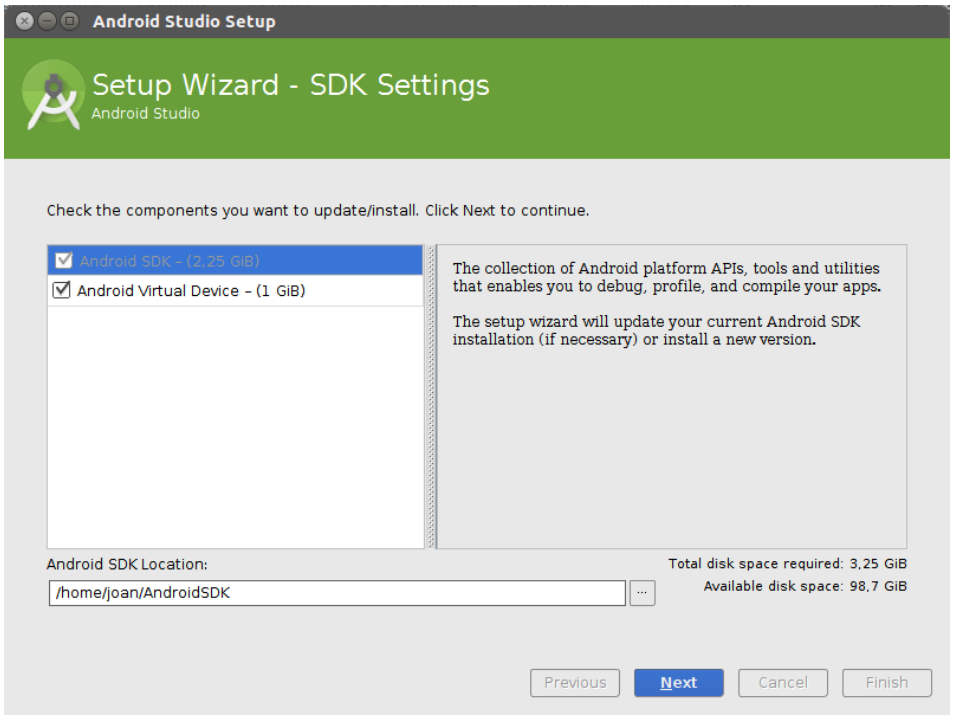
### 1.3.2 Instal·lant l'Android Studio i l'SDK

Podeu descarregar l'Android Studio des de la següent adreça: http://developer.android.com/sdk/index.html

Obrim un terminal, anem fins al directori de descàrrega i descomprimim el fitxer. Per tal d'executar l'Android Studio hem de canviar el directori a `android-studio/bin` i allà executar `./studio.sh`.

```
usuari@ubuntu:~/$ cd Descargas
  usuari@ubuntu:~/Descargas$ unzip android-studio-ide-*.zip
  usuari@ubuntu:~/Descargas$ cd android-studio/bin
  usuari@ubuntu:~/Descargas/android-studio/bin$ ./studio.sh
```

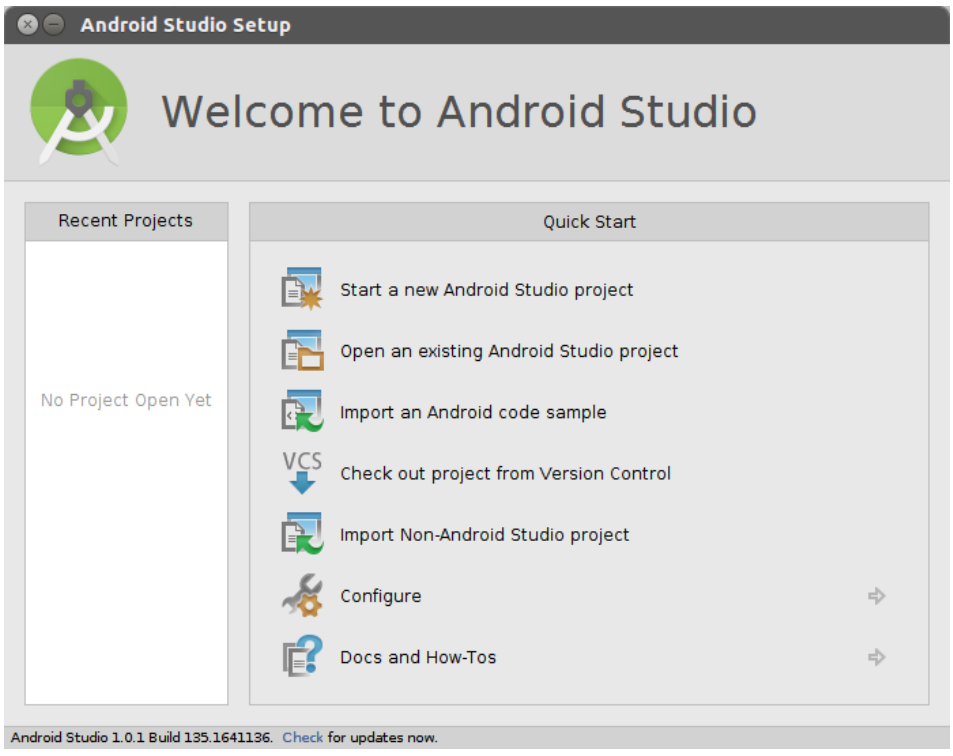
**Figura 1.5.** Android Studio la primera vegada que s'executa. Instal·lació 'Custom'




Seguint la configuració inicial instal·larem l'Android SDK, la darrera API d'Android i un emulador (figura 1.5), és a dir, tot allò que necessitem per desenvolupar aplicacions. Si ho volem, també disposem d'una versió *stand-alone* de l'SDK per descarregar des de http://developer.android.com/sdk/index.html#Other

Una vegada finalitzat l'assistent hauríem de veure la pantalla de la figura 1.6, a partir d'aquest moment ja podrem crear el nostre primer projecte.

**Figura 1.6.** Pantalla principal de l'Android Studio



Per tal de millorar el rendiment de l'emulador, si el nostre microprocessador és compatible, seria interessant instal·lar l'*Intel® Hardware Accelerated Execution Manager* seguint les instruccions del següent enllaç: Intel HAXM.

### 1.3.3 Gestió de l'SDK

L'últim pas en la configuració de l'SDK d'Android és la descàrrega dels components essencials per al nostre entorn de desenvolupament, que farem mitjançant una eina inclosa en el paquet, l'AVD Manager.

L'SDK fa servir una estructura modular que en separa les principals parts, els afegits, les eines, els exemples i la documentació en components instal·lables de forma separada. L'SDK Starter Package (paquet de començament d'SDK) que heu instal·lat inclou únicament la darrera versió de les eines de l'SDK. Podeu descarregar altres components, com les darreres versions de la plataforma d'Android quan es publiquen.

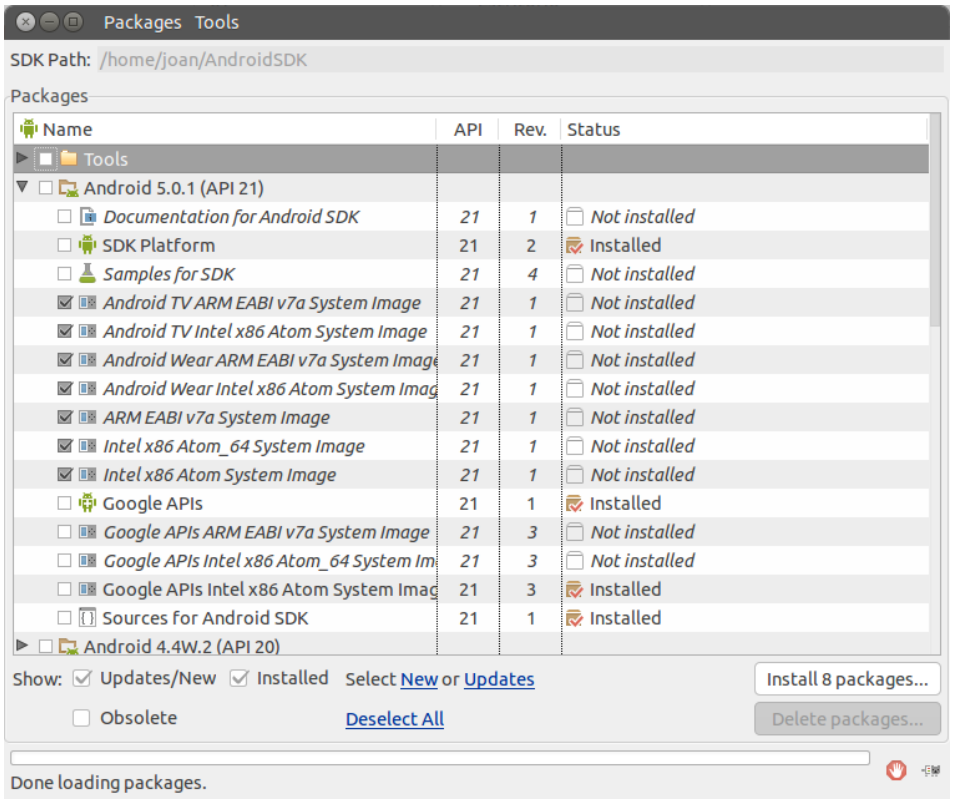
Podeu executar l'Android SDK de diverses maneres:

**Figura 1.7.** SDK Manager

- Des de la pantalla principal de l'Android Studio, seleccioneu *Configure/Android SDK Manager*.
- Des de qualsevol projecte, seleccioneu *Tools/Android/SDK Manager* o feu clic a la icona de la figura 1.7.
- Des de la consola, entreu dins del directori */tools* del directori de l'SDK d'Android i executeu `./android`.

Veureu un diàleg com el de la figura 1.8.

**FIGURA 1.8.** Paquets a instal·lar




Des d'aquí podeu seleccionar els components que voleu instal·lar o actualitzar.

Aquesta és una petita descripció dels components que hi podeu trobar:

- **SDK Tools**: conté les eines per comprovar i depurar les aplicacions Android. Aquestes eines estan instal·lades juntament a l'SDK d'Android i reben actualitzacions periòdiques. Es troben a la carpeta */tools* dins de la carpeta d'instal·lació de l'SDK.

- **Android SDK Platform-tools** (*eines de plataforma de l'SDK d'Android*): conté eines per desenvolupar i depurar l'aplicació dependents de la plataforma. Suporten les últimes característiques de la plataforma Android i, en general, s'actualitzen únicament quan hi ha una nova plataforma disponible. Es troben a la carpeta */platform-tools* dins de la carpeta d'instal·lació de l'SDK.

- **Plataformes Android:** hi ha plataformes de l'SDK disponibles per cada versió disponible d'Android. Cada plataforma Android conté una llibreria completa d'Android, imatges del sistema, codi d'exemple i *skins* de l'emulador (les diferents *pells* que simulen l'aparença externa dels dispositius, quan se simulen).

- **Samples:** conté el codi d'exemple i les aplicacions disponibles per a cada plataforma de desenvolupament d'Android. Si esteu començant a desenvolupar amb Android, assegureu-vos de descarregar els *samples* (exemples).

- **Documentació:** conté una còpia local de la darrera documentació de l'API

(*Applicacion Programming Interface*, interfície de programació d'aplicacions) d'Android.

Els afegits de tercers (*Third party Add-ons*) proporcionen components que permeten la creació d'un entorn de desenvolupament fent servir una llibreria externa d'Android específica (com, per exemple, la llibreria de Google Maps) o una imatge de sistema d'Android feta a mida (en anglès, *customized*).

### 1.3.4 Components recomanats

Google recomana la instal·lació de certs components per començar amb el desenvolupament d'Android.

En un **entorn bàsic** de desenvolupament hem de tenir, com a mínim:

- **SDK Tools**: amb la instal·lació realitzada des de l'assistent ja disposem d'aquest component.
- **SDK Platform-tools**: amb la instal·lació realitzada des de l'assistent ja disposem d'aquest component.
- **SDK Platform**: la instal·lació bàsica ens haurà descarregat la darrera versió de l'SDK Platform i una imatge del sistema d'un dispositiu. Amb això ja podreu compilar l'aplicació i executar un *Android Virtual Device* (*AVD*, dispositiu virtual Android). Per començar descarregueu, a banda de la darrera versió de la plataforma, la versió 2.3.3 (la versió 2.3.3 és compatible amb el 99,3% dels dispositius Android). Podeu descarregar més endavant altres versions per comprovar que el vostre programa és compatible amb aquestes.

**Quina versió d'Android cal fer servir?**

Quan desenvolupeu per a Android us pot sorgir el dubte de quina versió de la plataforma feu servir. Si feu servir la darrera versió disponible, és possible que el vostre programa estigui disponible únicament per a un percentatge reduït dels dispositius que hi ha al mercat. Per aquest motiu, és interessant en alguns casos fer servir versions anteriors perquè la vostra aplicació sigui compatible amb la major quantitat de dispositius possibles.

A banda d'aquests tres components que acabem d'especificar, la **instal·lació recomanada** és la següent:

- **Documentation**: la documentació dels components és útil perquè us permet treballar *offline* (sense connexió) i mirar la informació de referència de l'API des del mateix IDE.
- **Samples**: els exemples de components proporcionen codi que podeu fer servir per aprendre Android, carregar-lo com a projecte i executar-lo, o, fins i tot, reaprofitar-lo en els nostres projectes. Existeixen *samples* (exemples) per a cada versió de la plataforma, així doncs caldrà que escolliu els que corresponguin.

- **USB Driver**: aquest component únicament és necessari si es desenvolupa des de Windows i tenim un dispositiu Android sobre el qual volem instal·lar l'aplicació per la seva depuració i pel testeig. Per a plataformes GNU/Linux i Mac OSX no cal cap *driver* específic.

Per a una **instal·lació completa** es pot instal·lar també:

- **Google API**: dóna a la vostra aplicació accés a les llibreries externes de mapes, cosa que farà més senzill mostrar i manipular mapes a la vostra aplicació.
- **Plataformes addicionals de l'SDK**: si voleu publicar la vostra aplicació, haureu de descarregar les versions de la plataforma Android on voleu que la vostra aplicació s'executi. La recomanació és compilar l'aplicació sobre la versió més baixa d'Android que voleu que estigui suportada (per assegurar la compatibilitat), però comprovar-la (fer els tests) sobre la versió més alta on voleu que s'executi. Podeu comprovar les aplicacions en diferents plataformes executant-les en un AVD a l'emulador d'Android.

Una vegada instal·lada com a mínim la configuració bàsica dels components de l'SDK, podeu començar a desenvolupar aplicacions per a Android.

#### PATH del sistema

Podeu afegir opcionalment l'adreça de les carpetes /tools i /platform-tools dins de la carpeta de l'SDK al PATH del sistema. D'aquesta manera, tindreu un accés més fàcil a les eines. Per afegir les carpetes al PATH heu de consultar la documentació del vostre sistema operatiu. En sistemes basats en UNIX (com GNU/Linux i Mac OSX) heu de modificar el contingut del fitxer .bashrc que es troba a la vostra carpeta d'usuari i afegir les carpetes a la variable PATH; per exemple:

```
1 PATH=${PATH}:/home/usuari/android-sdk/tools:/home/usuari/android-
      sdk/platform-tools
```

# 2. Programació de dispositius mòbils

La programació de dispositius mòbils en Android té unes característiques particulars que la diferencien de la programació tradicional en equips de taula. La majoria d'aquestes característiques vénen donades pel fet que els dispositius mòbils tenen unes prestacions inferiors als equips de taula, i per tant el sistema operatiu ha hagut d'implementar unes solucions enginyoses per resoldre-les. D'altra banda, el sistema operatiu i l'API que es fan servir per a les aplicacions del sistema són únics. Aquestes aplicacions (a més a més) tendeixen a estar molt relacionades entre si. Per aquest motiu, és important veure en detall la forma de realitzar una aplicació i entendre els conceptes bàsics de la programació de dispositius mòbils.

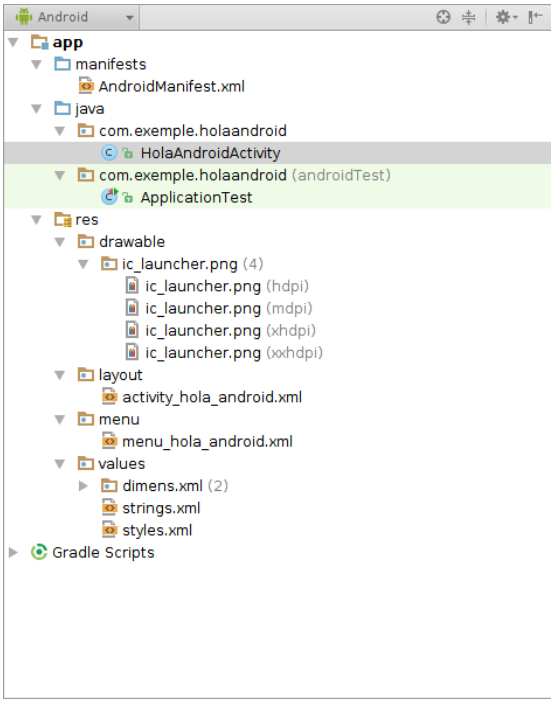
## 2.1 Conceptes previs

Existeixen tota una sèrie de conceptes que és interessant veure abans de començar directament amb la programació del dispositiu i l'explicació de les diferents classes que constitueixen l'API d'Android.

### 2.1.1 Parts d'una aplicació Android

Abans de res, si ja heu fet una aplicació bàsica amb Android ("Hola món!"), podeu mirar en el seu interior i examinar-ne cadascuna de les parts. En cas contrari, obriu el projecte "01 HolaAndroid" que trobareu als *Annexos* de la unitat. Quan obriu el projecte amb l'Android Studio veureu a la pestanya *Project* una sèrie de fitxers i carpetes com els que es mostren a la figura 2.1.

FIGURA 2.1. Vista Android del projecte. Podem canviar-la a Project i a Packages




La carpeta *Java* conté els fitxers Java amb el codi del projecte, en aquest cas "HolaAndroidActivity". Aquest fitxer conté el codi de l'activitat, i és aquí on escriureu el codi de la vostra aplicació.

El directori */res* conté tots els recursos que es fan servir a l'aplicació. Dins hi ha altres subdirectoris (diferents carpetes *drawable* per a diferents resolucions, *layout*, *values*). Les carpetes ***drawable***, en aquest moment només contenen les icones de l'aplicació en diferents resolucions. La carpeta *layout* conté el fitxer **activity_hola_android.xml**. Si l'obriu amb l'editor de text pla obtindreu un codi com el següent:

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools" android:layout_width="
           match_parent"
3      android:layout_height="match_parent" android:paddingLeft="@dimen/
           activity_horizontal_margin"
4      android:paddingRight="@dimen/activity_horizontal_margin"
5      android:paddingTop="@dimen/activity_vertical_margin"
6      android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".
           HolaAndroidActivity">
7
8      <TextView android:text="@string/hello_world" android:layout_width="
           wrap_content"
9          android:layout_height="wrap_content" />
10 </RelativeLayout>
```

Fixeu-vos en el camp "**android:text="@string/hello_world"**". La variable "**@string/hello_world**" fa referència al camp "hello_world" que es troba al fitxer **strings.xml** dins de la carpeta *res/values*. El fitxer **strings.xml** conté les diferents cadenes de text que es fan servir al vostre projecte. Si l'obriu amb l'editor de text trobareu un codi semblant a aquest:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Hola Android</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

Tot i que podeu ficar els camps de text com cadenes de caràcters constants, és recomanable que deseu totes les constants de text (els textos que apareixeran al vostre programa) dins del fitxer string.xml i que feu referència a aquests textos a partir de l'identificador "@string". D'aquesta manera, si heu de traduir la vostra aplicació a un altre idioma, tot el que necessitareu és traduir els textos del fitxer strings.xml amb l'idioma desitjat i tornar a compilar el projecte.

El directori */assets* conté altres recursos que es fan servir en l'aplicació, com ara fitxers de text, bases de dades... En aquest cas no el tenim disponible i l'haurem de crear manualment a la carpeta */res* en cas de voler fer-lo servir.

El fitxer **AndroidManifest.xml** del directori *manifests* és el fitxer de manifest de l'aplicació Android. Aquí hi ha especificats els permisos i altres característiques de l'aplicació.

## 2.2 Components bàsics d'una aplicació Android

Els blocs principals per construir les vostres aplicacions són els components que fareu servir per programar les aplicacions Android. Són elements conceptuals que ficareu junts per construir una aplicació en conjunt:

- Activitats
- *Fragments*
- Serveis
- Proveïdors de continguts
- Receptors de *broadcast*

### 2.2.1 Activitats

Podríem entendre l'***activity*** (activitat) de manera aproximada com cada una de les "pantalles" d'una aplicació. Una definició més precisa establiria que les activitats són cada un dels components de l'aplicació que proveeixen a l'usuari una pantalla amb la que aquest pugui interactuar.

Per exemple, una aplicació de missatgeria pot tenir una activitat que mostri la llista de contactes, una altra per enviar missatges i una altra per llegir missatges rebuts. Tot i que aquestes activitats treballen juntes dins de l'aplicació de missatgeria, cadascuna és independent de les altres.

A més a més, en Android, les aplicacions poden activar activitats (pantalles) d'altres aplicacions. Per exemple, l'aplicació de missatgeria a la que ens referíem podria obrir una activitat de l'aplicació de fotografia per enviar una foto juntament amb un missatge. De la mateixa manera, l'aplicació de fotografia podria obrir l'activitat d'una aplicació de correu electrònic per enviar la foto com a adjunt d'un correu.

Una activitat està implementada com una subclasse de la classe "Activity".

Podeu trobar més informació sobre la classe `Activity` a la *Guia del desenvolupador* d'Android:

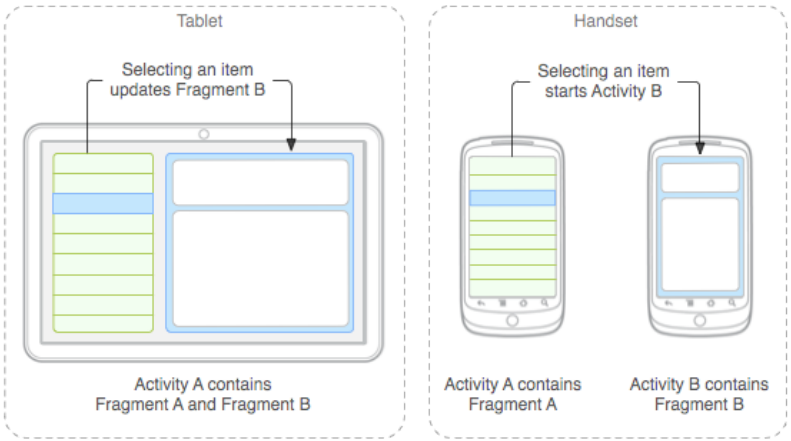
developer.android.com/guide/topics/fundamentals/activities.html.

### 2.2.2 'Fragments'

A la versió d'Android 3.0 (*Honeycomb*, una versió d'Android exclusiva per a tauletes) Google va incorporar el concepte de ***fragments***. Les majors dimensions de les pantalles en les tauletes permeten mostrar més informació que en un telèfon mòbil i per tant s'havia de trobar la manera de poder combinar diverses pantalles d'informació en una sola.

Un ***fragment*** és una porció d'una activitat. Podrem tenir diversos *fragments* i modificar-los durant l'execució de l'aplicació per tal de combinar-los segons les necessitats.

**Figura 2.2.** Exemple de "fragments" a Android



A la figura 2.2 podem veure com a la vista del telèfon mòbil una activitat disposa d'un sol *fragment* mentre que a la tauleta podem afegir dos *fragments* en cada activitat. Aquesta manera de treballar ens facilita la feina als desenvolupadors ja que, al tractar els *fragments* de manera modular, podem afegir i treure elements de l'activitat segons la quantitat d'espai lliure de què disposem i simplificar la programació per a dispositius amb diferents mides de pantalla.

### 2.2.3 Serveis

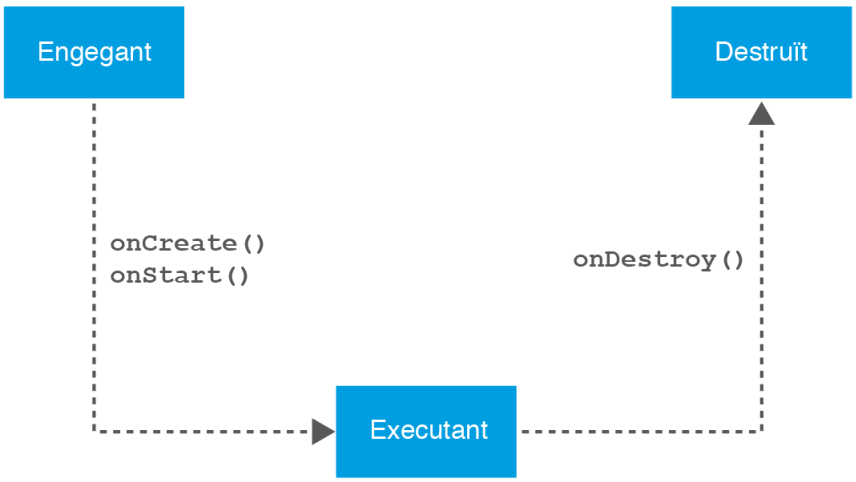
Un "***Service***" (servei) és un component que s'executa en el *background* (en el fons, que no es veu) per realitzar tasques per a treballs remots.

Els serveis s'executen, com dèiem, en el *background* i no tenen components d'interfície d'usuari. Poden fer les mateixes coses que una activitat, però no tenen interfície. Són útils per realitzar tasques durant un temps determinat, independentment de la pantalla. Per exemple, podeu voler agafar dades del receptor GPS del telèfon o reproduir música mentre esteu canviant a altres aplicacions. Aquests serveis poden haver estat iniciats per un altre component, com per exemple una activitat, i podran continuar en execució independentment del component que hagi fet la crida.

Els serveis d'Android no són el mateix que els serveis de Linux (també anomenats *daemons*), que representen un component del sistema operatiu de més baix nivell.

Els serveis tenen un cicle de vida molt més senzill que les activitats, com podreu veure si compareu la figura 2.3 amb la figura 2.5 que es mostra més endavant.

**Figura 2.3.** Cicle de vida d'un servei




Els serveis, o s'engeguen o s'aturen. A més a més, en general, el seu cicle de vida està més controlat pel programador, i no tant pel sistema.

Els serveis s'implementen com a classes derivades de la classe "Service". Podeu trobar més informació a la *Guia del desenvolupador* d'Android sobre aquesta classe, a developer.android.com/guide/topics/fundamentals/services.html.

### 2.2.4 Proveïdors de continguts

**'Intent'**

Un *intent* (en el sistema Android) és un mecanisme que permet als usuaris coordinar funcions de diferents aplicacions per tal de realitzar una tasca. Podeu trobar-ne més informació en l'enllaç http://developer.android.com/reference/android/content/Intent.html.

Els ***content providers*** (proveïdors de continguts) són interfícies destinades a compartir dades entre les aplicacions. Per defecte, Android executa les aplicacions de forma independent, de manera que les dades de l'aplicació estan aïllades de la resta d'aplicacions del sistema. Encara que es poden traspassar petites quantitats d'informació entre les aplicacions a través dels *intents*, els proveïdors de continguts són més adequats per compartir conjunts de dades entre aplicacions.

La informació es pot inserir, actualitzar, esborrar i consultar a un proveïdor de continguts. Android fa servir aquest mecanisme tota l'estona. Per exemple, els proveïdors de continguts de contactes (*contacts provider*) proporcionen la informació dels contactes del telèfon a diferents aplicacions. Així, una aplicació amb els permisos adequats pot consultar el *content provider* (`ContactsContract.Data`) per llegir i escriure informació sobre una persona en concret. Un altre exemple: el *Media Store* (Magatzem de mitjans), emmagatzema i serveix diferents tipus de dades per compartir, com fotos i música, entre les diferents aplicacions.

Els *content providers* també són útils per llegir i escriure dades que són privades a la vostra aplicació i no estan compartides. Per exemple, una aplicació de bloc de notes pot fer servir un *content provider* per desar notes.

La separació entre el magatzem de dades i la interfície d'usuari proporciona molta flexibilitat a Android. Per exemple, es podria instal·lar una aplicació alternativa per visualitzar els contactes, o una aplicació podria accedir des de l'escriptori a algunes de les dades de configuració del sistema (que també estan emmagatzemades en un proveïdor de continguts) per modificar-ne algunes en concret, com la d'activar i desactivar el Wi-Fi.

Els proveïdors de continguts tenen interfícies relativament senzilles amb mètodes estàndard per inserir, actualitzar, esborrar i consultar la informació, de manera molt semblant als mètodes d'accés a bases de dades.

Un *content provider* està implementat com una subclasse de "ContentProvider" i ha d'implementar una sèrie d'APIs que permeten a altres aplicacions fer transaccions. Per a més informació podeu consultar la secció de *content providers* de la *Guia del desenvolupador* d'Android: developer.android.com/guide/topics/providers/content-providers.html.

### 2.2.5 Receptors de 'broadcast'

**'Broadcasting'**

El terme *broadcasting*, popularitzat en els dècades de 1920 i 1930 per la implantació de la ràdio i la televisió, es fa servir en l'actualitat per referir-se a qualsevol emissió massiva de continguts d'un únic emissor a múltiples receptors.

Els ***broadcast Rreceivers*** (receptors de *broadcast*) són un tipus de components que responen a anuncis de sistema dirigits a totes les aplicacions del dispositiu (*broadcast*). En certa manera, és una implementació d'Android d'un sistema de subscripció d'esdeveniments. El receptor simplement es queda en estat latent esperant a activar-se quan s'esdevingui un esdeveniment al qual està subscrit.

El sistema està enviant missatges de *broadcast* tota l'estona. Per exemple, quan es rep una trucada, s'apaga la pantalla o la bateria té un nivell de càrrega baix, aquests esdeveniments són anunciats pel sistema. Un altre exemple: si voleu que un servei quedi engegat una vegada el sistema ha arrancat, podeu subscriure'l al *broadcast* que diu que el sistema ha acabat d'engegar-se.

Les aplicacions també poden enviar missatges de *broadcast*. Per exemple, una aplicació pot anunciar que s'han acabat de descarregar algunes dades i que aquestes ja estan disponibles per usar-se.

Els receptors de *broadcast* no tenen una representació visual, però quan es llancen executen una part de codi (com engegar una activitat o servei) i poden crear una notificació a la barra d'estat per alertar l'usuari quan ha ocorregut un esdeveniment de *broadcast*.

Un *broadcast receiver* està implementat com una subclasse de "BroadcastReceiver" i cada missatge de *broadcast* s'envia com un objecte `Intent`. Podeu trobar més informació a la *Guia del desenvolupador* d'Android a l'apartat de BroadcastReceiver, a developer.android.com/reference/android/content/BroadcastReceiver.html.

### 2.2.6 Activant els components

Tres dels cinc tipus de components (activitats, serveis i receptors de *broadcast*) s'activen per missatges asíncrons anomenats ***intents*** (intencions). Els *intents* lliguen els components individuals entre si durant l'execució, tant si els components pertanyen a la mateixa aplicació com si no. Podeu pensar en els *intents* com missatges que s'envien entre si els diferents components de les aplicacions (i que, de fet, poden servir per activar-ne alguns d'ells).

**URI**

Un URI (*Universal Resource Identifier*, identificador de recursos universal) és una cadena de caràcters que serveix per referenciar recursos en el sistema. Aquesta identificació permet la interacció amb representacions del recurs en el sistema. El mètode `Uri.parse(String)` serveix per analitzar i codificar un String en forma d'URI.

Un *intent* es crea amb un objecte de la classe `Intent`, que defineix un missatge per activar un component específic (*intent* explícit) o un tipus de component (*intent* implícit).

Per activitats i serveis, un *intent* defineix una acció que s'ha de realitzar (per exemple, "veure" o "enviar" alguna cosa) i pot especificar l'URI de les dades sobre les quals s'ha d'actuar. Per exemple, un *intent* pot enviar una petició d'obrir una pàgina web i especificar l'adreça de la web, per tal que l'activitat corresponent pugui obrir la pàgina correctament.

En altres casos es pot començar una activitat i esperar rebre un resultat. Per exemple, es podria iniciar una activitat per seleccionar una persona de l'agenda de contactes, i l'activitat retornaria el resultat en un *intent* (aquest inclouria l'URI apuntant al contacte seleccionat).

Per als Receptors de *broadcast*, un *intent* simplement defineix l'anunci que s'està transmetent; per exemple, un missatge a tot el dispositiu per indicar que el nivell bateria del dispositiu és baix inclouria únicament un String indicant aquest anunci.

Els proveïdors de continguts no s'activen per *intents*, sinó amb peticions d'un ***ContentResolver*** (Resolutor de continguts). El Resolutor de continguts tracta totes les transaccions directes amb el proveïdor de continguts, per tal que no ho hagi de fer directament el component que el fa servir. Això proporciona una capa d'abstracció entre el proveïdor de continguts i el component que sol·licita la informació (per seguretat).

Existeixen mètodes diferents per activar cada tipus de component:

- Les activitats es poden invocar creant un objecte de la classe `Intent` i passant-lo als mètodes `startActivity(Intent i)` o a `startActivityForResult (Intent i, int requestCode)` si voleu que l'activitat us retorni un resultat.
- Es pot iniciar un servei passant un *intent* a `startService(Intent service)`. Podeu enllaçar a un servei passant un *intent* a `bindService (Intent service, ServiceConnection conn, int flags)`.
- És possible iniciar un missatge de *broadcast* passant un *intent* als mètodes `sendBroadcast (Intent i)`, `sendOrderedBroadcast (Intent intent, String receiverPermission)` o `sendStickyBroadcast (Intent i)`.
- Es pot realitzar una consulta a un proveïdor de continguts amb una crida a `query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)` d'un ContentResolver.

Podeu consultar informació sobre tots aquests mètodes a la *Guia del desenvolupador* d'Android: developer.android.com/reference/packages.html.

### 2.2.7 El fitxer AndroidManifest.xml

Abans que el sistema Android iniciï un component de l'aplicació, el sistema ha de saber que aquest component existeix. Per anunciar al sistema els components que

farà servir l'aplicació existeix el fitxer **AndroidManifest.xml** (el fitxer de *manifest* de l'aplicació). L'aplicació ha de declarar tots els components en aquest fitxer, que ha d'estar a l'arrel del directori del projecte.

A més de declarar els components de l'aplicació, el fitxer de *manifest* fa les següents funcions, entre d'altres:

- Identifica els permisos que necessita l'aplicació (com ara permisos per accedir a Internet).
- Declara les característiques de maquinari i programari necessàries per executar correctament l'aplicació, com poden ser, entre d'altres:
  - La càmera.
  - La resolució mínima de la pantalla per poder funcionar.
  - Els dispositius d'entrada necessaris per executar l'aplicació (com el *pad* de quatre direccions).
  - Bluetooth.
  - Pantalla multitàctil.
- Declara les llibreries d'aplicació, a banda de les API d'Android, que s'han d'enllaçar (*link*) a l'aplicació, per exemple com la llibreria de Google Maps.

El fitxer XML del projecte d'*Hola Món* és el següent:

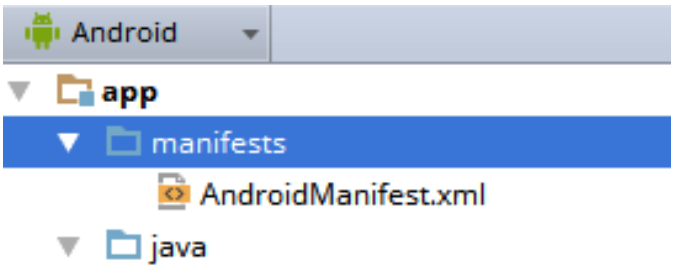
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exemple.holaandroid" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".HolaAndroidActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

El fitxer *manifest* el podeu trobar a la carpeta *manifests* de qualsevol projecte, tal i com podeu veure a la figura 2.4.

**FIGURA 2.4.** Carpeta manifests d'un projecte




En l'element `<application>` hi podeu trobar altres atributs específics de l'aplicació, per exemple `android:icon` apunta al recurs de la icona que identifica l'aplicació i `android:label` apunta al recurs (dins del fitxer **string.xml**) amb el nom de l'aplicació.

L'element `<activity>` serveix per declarar una activitat. Així, l'atribut `android:name` especifica el nom de la subclasse (que hereta de la classe `Activity`) que tindrà el codi de l'activitat. Aquesta classe ha d'estar implementada i aparèixer al projecte. Els components de l'aplicació han d'estar declarats fent servir les etiquetes: `<activity>`, `<service>`, `<receiver>` i `<provider>`.

Aquells components que programeu i incloeu al projecte però no estiguin declarats al fitxer AndroidManifest.xml no es podran executar, atès que no són visibles pel sistema. Els receptors de *broadcast*, però, es poden declarar al *manifest* o bé crear-se dinàmicament amb codi i registrar-se al sistema tot cridant a `registerReceiver()`.

**Gradle**

*Gradle* és una eina per automatitzar la construcció de projectes. Podrem configurar fàcilment distintes versions de la nostra aplicació, per exemple: generar una versió *demo* o *lite* de la nostra aplicació, la versió "completa", per fer "debug", etc. A més, disposem d'una gran quantitat de *plugins* i la gestió de dependències millora respecte *ant*, l'antic sistema de construcció utilitzat a Android. Per més informació podeu consultar el següent enllaç: https://developer.android.com/tools/building/configuring-gradle.html.

La tasca principal del *manifest* és informar el sistema sobre els components de l'aplicació. Actualment, algunes de les característiques que abans trobàvem al fitxer de *manifest* han passat a formar part del fitxer `build.gradle (Module: app)`, que trobem dins de `Gradle Scripts`, els quals pengen a la vegada de l'arrel del projecte; és el cas de la declaració de la versió de la plataforma, el nivell mínim d'API o la versió de la nostra aplicació.

Aquest és el fitxer *build.gradle* d'*HolaAndroid*:

```
1  apply plugin: 'com.android.application'
2
3  android {
4      compileSdkVersion 21
5      buildToolsVersion "21.1.2"
6
7      defaultConfig {
8          applicationId "com.exemple.holaandroid"
9          minSdkVersion 10
10         targetSdkVersion 21
11         versionCode 1
12         versionName "1.0"
13     }
14     buildTypes {
15         release {
16             minifyEnabled false
17             proguardFiles getDefaultProguardFile('proguard-android.txt'), '
                   proguard-rules.pro'
18         }
19     }
20 }
```

```
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     compile 'com.android.support:appcompat-v7:21.0.3'
25 }
```

Per a més informació sobre el contingut i l'estructura del fitxer AndroidManifest.xml, podeu consultar la documentació de la *Guia del desenvolupador* d'Android: developer.android.com/guide/topics/manifest/manifest-intro.html.

#### Declarant les capacitats del component

Es pot fer servir un *intent* per activar activitats, serveis i receptors de *broadcast*. Per fer-ho es pot especificar a l'*intent* el nom del component que es vol activar (fent servir el nom de la classe de l'activitat que voleu activar, per exemple). Però existeix una altra forma d'activar activitats, a través del que es coneix com a ***intent actions* (*intents* d'acció)**.

Un *intent* d'acció descriu el tipus d'acció que es vol realitzar (i, opcionalment, les dades sobre les quals es vol realitzar) i delega al sistema la tasca de trobar un component en el dispositiu que pugui realitzar aquesta acció, i activar-lo. Per exemple, es pot llançar un *intent* on s'especifica que es vol visitar una pàgina web o fer una foto, i deixar que el sistema trobi l'activitat (de l'aplicació que sigui) que pugui realitzar aquesta acció. En el cas que hi hagi múltiples components que puguin realitzar l'acció, es demana a l'usuari quina d'elles vol fer servir.

Com pot saber el sistema quins components són apropiats per fer la tasca que es demana des de l'*intent*? Els components poden definir uns **filtres d'*intents*** (***intent filters***) al fitxer de *manifest* que seran comparats pel sistema amb l'acció que s'ha demanat fer. Podeu, opcionalment, definir aquests filtres d'*intents* al *manifest* de la vostra aplicació per tal que el vostre component pugui respondre a *intents* d'altres aplicacions. Per fer-ho, heu d'afegir un element `<intent-filter>` com un fill de l'element de la declaració de component al fitxer XML.

Podeu trobar més informació sobre els filtres d'*intents* a la *Guia del desenvolupador* d'Android: developer.android.com/guide/topics/intents/intents-filters.html.

### 2.3 Activitats

A Android, una ***Activity*** (activitat) és un component d'aplicació que proporciona una finestra amb una interfície d'usuari de la vostra aplicació. Una aplicació també pot no tenir cap activitat.

Generalment, les aplicacions tenen més d'una activitat i el seu principal propòsit és el d'interactuar amb l'usuari. Des del moment que una activitat es mostra en pantalla fins al moment que es tanca, l'activitat passa per una sèrie de fases, conegudes com el ***cicle de vida*** de l'activitat. És molt important que entengueu el cicle de vida de l'activitat per tal que la vostra aplicació funcioni correctament.

Una ***Activity*** (activitat) és un component que proporciona una interfície gràfica amb la qual interactua l'usuari.

Podeu trobar la darrera informació oficial de Google sobre activitats a developer.android.com/guide/topics/fundamentals/activities.html.

Una aplicació consisteix en múltiples activitats que estan relacionades entre si. En general, existeix una activitat considerada l'**activitat principal** (o *main*) de l'aplicació, que és la que es mostra quan es llança l'aplicació per primera vegada. Aquesta activitat principal pot iniciar altres activitats per realitzar diferents accions. Cada vegada que es llança una activitat, l'activitat prèvia s'atura, però el sistema guarda l'activitat a la pila (anomenada *back stack*). Quan s'inicia una activitat, aquesta concentra el focus de l'aplicació.

Quan una activitat s'atura perquè una altra comença, la primera queda notificada a través dels **mètodes de *callback***, que són mètodes de l'activitat que són cridats quan aquesta canvia d'estat (si es crea, s'atura, es destrueix...) i que li permeten realitzar tasques en relació a aquest canvi d'estat. Per exemple, quan l'activitat s'atura aquesta hauria d'alliberar els recursos reservats i tancar les comunicacions de xarxa. Quan una activitat es llança, podria activar aquestes mateixes connexions de xarxa.

**Callback**

Una funció de *callback* (retrotrucada o devolució de trucada) és una referència a una porció de codi executable ("funció A") que es passa com a argument en la crida a un altre codi ("funció B"). D'aquesta manera, el codi (la "funció B") pot cridar la funció que se li passa com a argument ("funció A") quan ho necessiti. Per exemple, imagineu que voleu cridar una funció que realitza una cerca a una llista ("B"), i que quan trobi un element en concret voleu que cridi una funció vostra per mostrar-ho per pantalla ("A"). En aquest cas, es passaria a la crida de la funció B una referència a A. Quan B trobi l'element, farà una crida a A.

### 2.3.1 Treballant amb activitats

Per crear una activitat heu de crear una classe derivada de la classe `Activity`(o una subclasse d'aquesta). En la classe que heu creat heu d'implementar els mètodes de *callback* que el sistema cridarà en les transicions dels diferents estats del cicle de vida de l'activitat (quan es crea, es destrueix, etc.). Els dos mètodes més importants són:

- `onCreate()`: el sistema crida aquest mètode quan es crea l'activitat. Dins, heu d'inicialitzar els components inicials de la vostra aplicació. Aquí s'ha de cridar `setContentView()` per definir la distribució de la interfície d'usuari de l'activitat (coneguda com a *layout*).
- `onPause()`: el sistema crida aquest mètode, en primer lloc, per indicar que l'usuari està sortint de la vostra activitat. Encara que això no signifiqui necessàriament que l'activitat es destrueixi, cal que deseu tots els canvis

que voleu conservar de l'Aplicació, ja que és possible que l'usuari no torni a l'activitat.

La interfície d'usuari d'una activitat està formada per una jerarquia de **vistes** (objectes derivats de la classe `View`). Cada vista (*view*) controla un espai rectangular dins de l'espai de l'activitat i pot respondre a la interacció de l'usuari. Per exemple, una vista podria ser un botó que inicia una acció quan és premut per l'usuari.

A Android disposeu d'una sèrie de vistes que ja estan fetes que podeu fer servir per dissenyar la vostra activitat. Els ***widgets*** són vistes que proporcionen elements visuals i interactius en pantalla com un botó, un camp de text, un *checkbox* o una imatge. Els ***layouts*** (distribucions) també són vistes que deriven de la classe `ViewGroup`. Contenen altres vistes i proporcionen un tipus de distribució específica per a elles. Per exemple, un *layout linear* pot contenir altres botons (vistes) que es mostraran en forma de vista en la pantalla. Podeu crear subclasses derivades de `View` i `ViewGroup` (o les seves subclasses) per crear els vostres *widgets* i *layouts* i aplicar-los a les vostres activitats.

Els *layouts* de les diferents activitats de l'aplicació estan definits a una sèrie de fitxers XML que es troben a la carpeta */res/layout* del projecte. Existeix un fitxer XML per cada *layout*, sent el fitxer **main.xml** el que es crea per defecte per a la primera activitat de l'aplicació. Podeu carregar els *layouts* com la interfície gràfica de la vostra activitat amb una crida a la funció `setContentView (int layoutResID)` que té un únic argument, l'identificador del *layout* que es vol carregar.

A més de les activitats, els ***intents*** són un altre concepte únic d'Android. Aquests, bàsicament, permeten a diferents activitats de diferents aplicacions treballar conjuntament com si totes pertanyessin a la mateixa aplicació.

### 2.3.2 Cicle de vida d'una activitat

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Cicle de vida" de la secció "Annexos".

Per crear una activitat primer heu de crear una classe de Java que sigui descendent de la classe base `Activity`. Per seguir l'explicació, creeu un nou projecte.

Fixeu-vos en la definició de la classe que trobeu quan creeu un nou projecte:

```
1 public class ActivitatPrincipal extends ActionBarActivity {
2
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
6         setContentView(R.layout.activity_main);
7     }
8 }
```

La classe `ActivitatPrincipal` hereta d'`ActionBarActivity` que a la vegada és un descendent d'`Activity`. La classe `Activity` defineix una sèrie d'*events* que defineixen el cicle de vida de l'aplicació:

- `onCreate()`: es crida quan l'activitat s'ha creat per primera vegada.
- `onStart()`: es crida quan l'activitat és visible per a l'usuari.
- `onResume()`: es crida quan l'activitat comença a interaccionar amb l'usuari.
- `onPause()`: es crida quan l'activitat actual es pausa i l'activitat anterior és represa.
- `onStop()`: es crida quan l'activitat ja no és visible per a l'usuari.
- `onDestroy()`: es crida abans que l'activitat sigui destruïda pel sistema (manualment o pel sistema per estalviar memòria).
- `onRestart()`: es crida quan l'activitat s'ha aturat i s'ha tornat a iniciar.

Un ***handler*** (controlador) és un mètode o funció que s'executa per tractar un esdeveniment ocorregut al sistema.

Per defecte, l'activitat que es crea al nou projecte conté l'*event* `onCreate()`. Dins d'aquest mètode (és un *handler* de l'*event*) hi ha el codi que serveix per mostrar els elements de la **UI** (*user interface*, interfície d'usuari) en pantalla. El cicle de vida d'una activitat i les diferents fases per les quals passa són les següents:

- Estat inicial
- Estat en execució
- Estat pausat
- Estat aturat (*stop*)
- Estat destruïda

#### Estat inicial

Quan una activitat s'està engegant passa a través de tota una sèrie de crides a mètodes de *callback* on podeu introduïr codi. Tot seguit, passarà a l'estat d'execució. Aquest procés de creació de l'activitat (la transició entre l'estat inicial i l'estat en execució) és una de les operacions més costoses en temps de computació i, per tant, afecta la vida de la bateria del dispositiu. Aquest és el motiu pel qual les activitats no es destrueixen automàticament quan es deixen de mostrar, ja que l'usuari pot voler tornar-hi aviat.

#### Estat en execució

Una activitat en execució és la que està actualment en pantalla i oferint interacció a l'usuari. També diem que l'activitat està *en focus*, és a dir, que totes les interaccions de l'usuari (com tocar la pantalla, escriure al teclat, pressionar els diferents botons del telèfon) són tractades per aquesta activitat. Per aquest motiu, hi ha una única activitat en estat d'execució en cada moment. L'activitat en execució és la que té prioritat en termes de memòria i recursos, per tal d'oferir a l'usuari un temps de resposta reduït.

### Estat pausat

Quan una activitat no té el *focus* (l'usuari no hi està interactuant) però encara està visible en pantalla, diem que està en estat de pausa. No és una situació freqüent, ja que la pantalla del dispositiu sol ser petita i l'activitat, generalment, o bé la fa servir tota o bé no la fa servir en absolut. Però de vegades es mostra un diàleg en pantalla,fent que l'activitat que estava en execució quedi en pausa. A més, totes les activitats que es tancaran (estat *stop*) passen abans per l'estat de pausa. Les activitats en estat de pausa encara tenen una prioritat alta en termes de memòria i altres recursos, perquè romanen visibles i no es poden amagar.

### Estat aturat ('stop')

Una activitat està en estat aturat quan no és visible però encara està en memòria. Una activitat aturada es pot tornar a mostrar (i, per tant, tornaria a ser una activitat en execució) o es pot destruir definitivament alliberant l'espai que ocupava a la memòria. El sistema desa les activitats en estat aturat, ja que és probable que l'usuari les vulgui tornar a executar aviat, i reiniciar una activitat aturada és més ràpid que engegar-la des de zero. Això és així perquè els objectes ja estan carregats en memòria i és més fàcil portar-los al capdavant de la pantalla. Les activitats aturades es poden treure de memòria en qualsevol moment (ja que el sistema pot necessitar la memòria que estan ocupant).

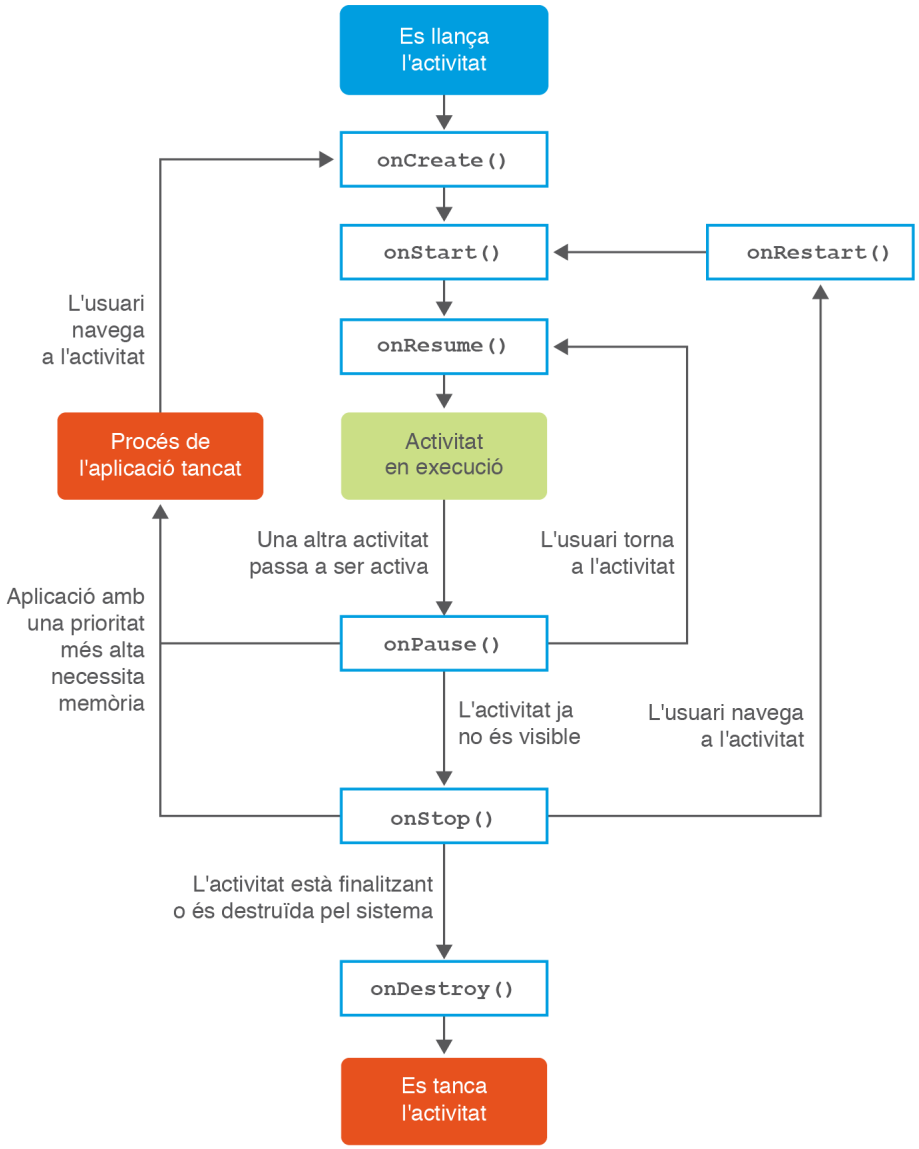
### Estat destruïda

Una activitat destruïda és aquella que ja no està en memòria. L'*activity manager* ha decidit que aquesta activitat ja no és necessària i ha alliberat l'espai de memòria que estava ocupant. Abans que es destrueixi es poden fer certes accions, com salvar informació important de l'activitat.

### Cicle de vida de l'activitat

La figura 2.5 mostra les etapes del cicle de vida de l'activitat.

**Figura 2.5.** Cicle de vida d'una activitat






**En execució i aturada**

El fet que una activitat estigui en execució no vol dir necessàriament que estigui fent moltes coses. Podria estar senzillament esperant interacció per part de l'usuari. Igualment, una activitat en estat aturat no vol dir que necessàriament no estigui fent res. Els noms dels estats fan referència a com d'activa està l'activitat respecte a l'entrada de l'usuari. En altres paraules, si l'activitat està visible, si té el focus o si no és visible en absolut.

Què fa l'activitat en cada estat? Per veure i entendre les diferents fases del cicle de vida d'una activitat, el millor és crear un projecte i interactuar-hi, i veure quan ocorren els diferents esdeveniments.

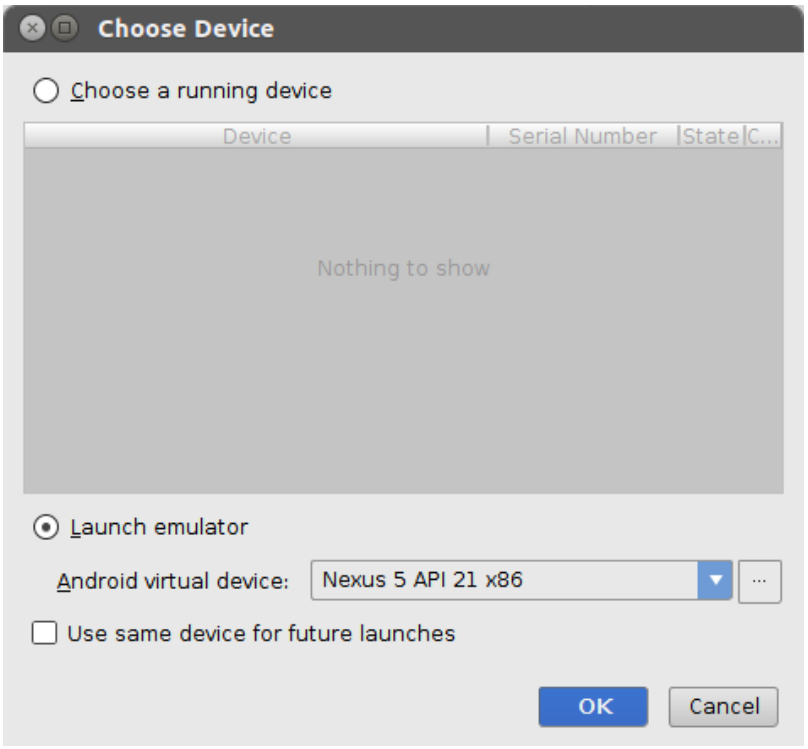
Una vegada creat el nou projecte, el fitxer **ActivitatPrincipal** quedarà:

```
1  package cat.xtec.ioc.cicledevida;
2
3  import android.support.v7.app.ActionBarActivity;
4  import android.os.Bundle;
5  import android.util.Log;
6
7
8  public class ActivitatPrincipal extends ActionBarActivity {
9
10     String tag = "Events";
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
```

```
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_main);
16
17          Log.d(tag, "A l'event onCreate()");
18      }
19
20      @Override
21      public void onStart() {
22          super.onStart();
23          Log.d(tag, "A l'event onStart");
24      }
25
26      public void onRestart() {
27          super.onRestart();
28          Log.d(tag, "A l'event onRestart()");
29      }
30
31      public void onResume() {
32          super.onResume();
33          Log.d(tag, "A l'event onResume()");
34      }
35
36      public void onPause() {
37          super.onPause();
38          Log.d(tag, "A l'event onPause()");
39      }
40
41      public void onStop() {
42          super.onStop();
43          Log.d(tag, "A l'event onStop()");
44      }
45
46      public void onDestroy() {
47          super.onDestroy();
48          Log.d(tag, "A l'event onDestroy()");
49      }
50
51  }
```

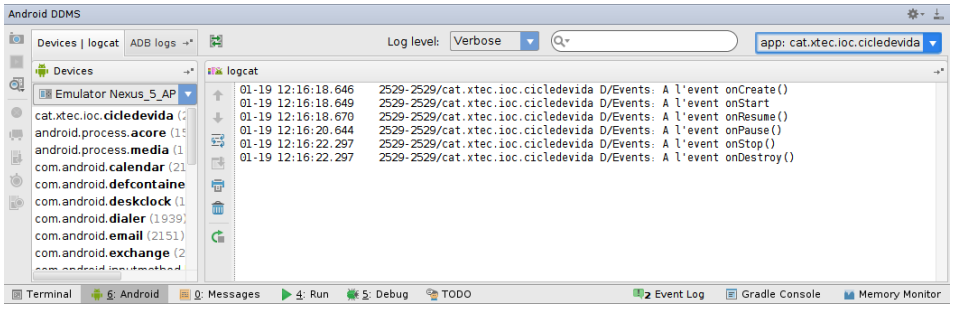
Abans d'executar el projecte feu clic al menú *Run/Edit Configurations* i a l'opció *Target device* assegureu-vos que està marcada l'opció *Show chooser dialog*. Això farà que cada vegada que executeu el programa se us demani on s'executarà d'entre les diferents opcions possibles del sistema (màquines virtuals o dispositius físics connectats a l'ordinador). Feu clic a *Run* per executar el projecte. Veureu una pantalla com la de la figura 2.6. En aquest cas en concret, no disposem de cap dispositiu ni emulador en funcionament, i se'ns proposa executar una màquina virtual d'un Nexus 5.

**FIGURA 2.6.** Selecció de dispositiu on s'executarà l'aplicació




Una vegada fet això, podeu executar el projecte per depurar amb el botó dret el projecte al *Package Explorer* i l'opció *Debug As/Android Application*. Seguiu aquests *events* amb el gràfic de la figura 2.5 ("Cicle de vida d'una activitat"). Podeu veure també els esdeveniments de *log* a la finestra *LogCat* de l'Android Studio, tal com podeu veure a la figura 2.7.

**FIGURA 2.7.** Log d"events'




Podeu provar d'afegir filtres fent clic al desplegable de la part superior dreta i escollint *Edit filter configuration*.

Així, quan es carrega l'aplicació per primera vegada es mostrarà al *logcat* el següent:

```
1 D/Events: A l'event onCreate()
2 D/Events: A l'event onStart()
3 D/Events: A l'event onResume()
```

Les inicialitzacions de les variables de la vostra aplicació les podeu fer a l'*event* `onCreate`, que com vèiem al codi anterior és el primer en executar-se. Quan

premeu el botó de tornar enrere al simulador, els *events* que es mostren al *log* seràn aquests:

```
D/Events: A l'event onPause()
D/Events: A l'event onStop()
D/Events: A l'event onDestroy()
```

Podeu observar com quan premem la tecla *enrere* l'aplicació es destrueix (crida el mètode `onDestoy()`) de manera que si tornem a executar-la, ja sigui des del menú o des de la tecla *recents* (prement durant una estona el botó *home* en aquells telèfons on no existeixi aquesta tecla), es tornaran a produir els següents esdeveniments:

```
D/Events: A l'event onCreate()
D/Events: A l'event onStart()
D/Events: A l'event onResume()
```

Si mentre estem a l'aplicació premem el botó *home* i tornem a executar-la, la seqüència d'esdeveniments serà la següent:

```
D/Events: A l'event onPause()
D/Events: A l'event onStop()
D/Events: a l'event onRestart()
D/Events: A l'event onStart()
D/Events: A l'event onResume()
```

En aquest cas no es fa cap crida a `onDestroy()` i per tant, al tornar a tenir l'aplicació en primer pla no s'executa el mètode `onCreate()`. Quan una aplicació està pausada o aturada i tornem a ella, executarà el mètode `onRestart()`, seguits d'`onStart()` i `onResume()`.

Hem vist com l'activitat es destrueix quan es prem el botó de tornar enrere (*Back button*). És molt important aquest punt, ja que l'estat de l'activitat es perd. Per tant, heu d'afegir codi addicional per preservar l'estat de l'activitat abans no es destrueixi (en qualsevol dels *events* que s'executen abans de destruir l'aplicació) i tornar a restituir l'estat quan es torni a obrir. Per exemple, les dades que voleu desar quan l'aplicació deixa d'estar en execució les podríeu desar a `onPause()` i tornar a carregar-les des d'`onResume()`. </newcontent>

L'*event* `onPause()` és cridat als dos escenaris, quan l'activitat s'envia al fons (*background*) i quan és eliminada amb el botó *Back*. Quan l'activitat es torna a engegar sempre es crida `onStart()` i `onResume()`, independentment de si ha tornat del fons (*background*) o si s'ha creat de nou.

## 2.4 'Intents'

Els ***intents*** són missatges enviats entre els diferents elements que conformen les aplicacions d'Android. Les activitats, serveis i *broadcast receivers* (receptors de *broadcast*) són activats a través d'aquests missatges. Els *intents* li indiquen a una activitat que s'iniciï, o a un servei que comenci o s'aturi, o són simplement

missatges de *broadcast* (dirigits a totes les aplicacions del dispositiu). En definitiva, serveixen per comunicar components de la mateixa aplicació o d'altres.

> Els ***intents*** serveixen per comunicar diferents components de la mateixa aplicació o d'aplicacions diferents. Generalment fan referència a una acció que s'ha de realitzar i van acompanyats de dos elements: l'acció que s'executarà i la informació que aquesta necessita.

L'*intent* és un objecte de la classe `Intent`, i conté la informació de l'operació que es vol realitzar o, en el cas dels *broadcasts*, la descripció d'algun esdeveniment que ha tingut lloc en el sistema. Existeixen diferents maneres d'enviar els *intents* en funció del tipus de component al qual van dirigits.

Per adreçar un *intent* a una activitat, podeu fer servir els mètodes `Context.startActivity (Intent intent)` o `Activity.startActivityForResult (Intent intent, int requestCode)`, depenent de si espereu que l'activitat us retorni alguna informació o no.

Per iniciar un servei o enviar informació a un servei podeu fer servir el mètode `Context.startService (Intent service)`. Es pot fer servir `Context.bindService (...)` per establir una connexió entre el servei i el component que l'està cridant.

Per adreçar un *intent* a tots els receptors de *broadcast* podeu fer servir `Context.sendBroadcast (Intent intent)`, `Context.sendOrderedBroadcast (...)` o `Context.sendStickBroadcast (Intent intent)`.

En cada cas, el sistema Android troba l'activitat, servei o receptor de *broadcast* que respondrà a l'*intent*, instanciant-lo si cal.

### 2.4.1 Objecte "Intent"

Un `Intent` és un objecte que conté informació d'interès per al component que el rep (com l'acció que ha de realitzar i les dades sobre les quals l'ha de realitzar). També conté informació per al sistema Android (com la categoria del component que ha de tractar l'*intent* i les instruccions sobre com llançar el component). Conté la següent informació:

- Nom del component
- Acció
- Dades
- Categoria

- Extres
- *Flags*

### Nom del component

El nom del component que ha de tractar l'*intent* és un objecte `ComponentName`. El nom del component és opcional. Si s'especifica, l'*intent* és enviat a la instància de la classe especificada. Si no s'especifica, el sistema trobarà un component per tractar l'*intent* (en base a una altra informació especificada a l'objecte `Intent`).

El component s'especifica amb `setComponent()`, `setClass()` o `setClassName()`.

### Acció

L'acció és un *string* que especifica l'acció a realitzar o, en el cas dels *intents broadcast*, una acció que ha tingut lloc i que s'està anunciant. La classe `Intent` defineix alguns tipus d'accions estàndards, com les que es poden veure a la taula 2.1. Podeu veure tots els tipus de constants d'acció a la *Guia del desenvolupador* d'Android, a la pàgina de la classe `Intent`.

**Taula 2.1.** Diferents tipus d'accions

| Constant | Component al qual va dirigit l'*intent* | Acció |
|---|---|---|
| `ACTION_CALL` | Activitat | Realitza una trucada. |
| `ACTION_EDIT` | Activitat | Mostra dades a l'usuari per a edició. |
| `ACTION_MAIN` | Activitat | Inicia com a activitat principal d'una aplicació. No s'espera valor de retorn. |
| `ACTION_SYNC` | Activitat | Sincronitza dades des del dispositiu al servidor. |
| `ACTION_VIEW` | Activitat | Mostra dades a l'usuari. |
| `ACTION_BATTERY_LOW` | Receptor de *broadcast* | Advertència de que la bateria del dispositiu està baixa. |
| `ACTION_HEADSET_PLUG` | Receptor de *broadcast* | S'han connectat o desconnectat els auriculars del dispositiu. |
| `ACTION_SCREEN_ON` | Receptor de *broadcast* | S'ha encès la pantalla. |
| `ACTION_AIRPLANE_MODE_CHANGED` | Receptor de *broadcast* | S'ha modificat el mode avió del dispositiu. |

Podeu definir i llegir l'acció d'un objecte `Intent` amb els mètodes `setAction()` i `getAction()`.

### Dades

Es tracta de la informació relativa a les dades sobre les quals s'ha de realitzar l'acció (especificant l'URI) i el tipus MIME d'aquestes dades.

**MIME**

MIME (*Multipurpose Internet Mail Extensions*, extensions de correu d'Internet multipropòsit) és un estàndard d'Internet que es va dissenyar originalment per especificar el tipus de les dades adjuntes a un correu electrònic, però que actualment serveix per especificar el tipus de dades en general.

Depenent del tipus d'acció, l'URI especificarà unes dades o altres. Per exemple, quan l'acció és `ACTION_CALL` el camp de dades serà de tipus "tel:" i anirà acompanyat d'un URI amb el número de telèfon al que s'ha de trucar. O si l'acció és `ACTION_VIEW`, el camp de dades pot ser del tipus "http:" i s'aportarà l'adreça del lloc web que es vol visitar.

Moltes vegades, el tipus de dades es pot deduir de l'URI, per exemple mirant si és la URL d'un web o una imatge del dispositiu. Però es pot definir específicament el tipus de les dades a l'objecte *intent*.

El mètode `setData (Uri data)` serveix per especificar l'URI de les dades, i el mètode `setType (String type)` serveix per especificar el tipus MIME de les dades. També podeu fer servir el mètode `setDataAndType (Uri data, String type)` per especificar ambdós. Per llegir l'URI i el tipus MIME, podeu fer servir `getData()` i `getType()`, respectivament.

### Categoria

És una cadena que conté informació addicional sobre el tipus de component que ha de tractar l'*intent*. La classe `Intent` defineix algunes categories, entre les quals es troben les de la taula 2.2.

**TAULA 2.2.** Alguns dels tipus de categories definides a l"intent'

| Constant | Significat |
|---|---|
| `CATEGORY_PREFERENCE` | L'activitat és un panell de preferències. |
| `CATEGORY_APP_MUSIC` | L'activitat ha de ser capaç de reproduir o manipular música. |
| `CATEGORY_APP_MESSAGING` | L'aplicació ha de ser capaç d'enviar i rebre missatges. |
| `CATEGORY_APP_BROWSER` | L'activitat ha de ser capaç de navegar per Internet. |

Podeu comprovar la llista completa de categories de la definició de la classe `Intent` a la *Guia del desenvolupador* d'Android.

Podeu fer servir els mètodes `addCategory(String category)`, `removeCategory(String category)` i `getCategories()` per afegir, esborrar i obtenir una llista de les categories de l'objecte *intent*, respectivament.

### Extres

Els extres són parells clau-valor que aporten informació addicional que s'ha d'enviar al component perquè tracti l'*intent*. Així, diferents URI tenen associats alguns extres en particular. Per exemple, l'acció `ACTION_HEADSET_PLUG`, que correspon a una canvi en l'estat de la connexió dels auriculars del dispositiu, té un camp extra indicant si els auriculars s'han endollat o no. Un altre exemple, l'acció `ACTION_TIMEZONE_CHANGED`, que correspon al fet que s'ha canviat l'ús horari, té un extra *time-zone* per especificar l'ús horari que s'ha definit.

La classe `Intent` té una sèrie de mètodes que es fan servir per afegir extres, els podeu consultar a la *Guia del desenvolupador* d'Android. També es pot fer servir un objecte del tipus `Bundle` per afegir extres. Un objecte de la classe `Bundle` (literalment de l'anglès, 'paquet' o 'feix') permet emmagatzemar una sèrie de dades (que poden ser de diferent tipus). Les variables es poden afegir al `Bundle` simplement donant un nom i la variable que es vol guardar. En certa manera, recorda els *arrays* amb què es passen variables entre els formularis en PHP: un *array* de *tuples* [nom, valor]. Podeu fer servir els mètodes `putExtras(Bundle extras)` i `getExtras()` per afegir i obtenir extres de l'*intent*.

#### 'Flags' (banderes)

Es poden especificar *flags* de diferents tipus que indiquin al sistema com ha de llançar l'activitat i com tractar-la després que s'hagi engegat. Tots aquests *flags* estan definits a la classe `Intent`.

#### Enviament de 'broadcasts'

Les aplicacions fan servir els objectes `Intent` per enviar *broadcasts* i activar components del sistema. Per exemple, a la taula 2.3 teniu alguns exemples d'*intents* que criden a serveis estàndard del sistema, amb el seu significat.

**Taula 2.3.** Diferents 'intents'

| Aplicació objectiu | URI de l'*intent* | Acció de l'*intent* | Resultat |
|---|---|---|---|
| Navegador | http://ioc.xtec.cat | `ACTION_VIEW` | Obre en un navegador l'adreça especificada. |
| Navegador | "text" | `ACTION_WEB_SEARCH` | Es realitza una cerca a Google de la cadena especificada. |
| Marcador | tel:*número de telèfon* | `ACTION_CALL` | Truca al número de telèfon especificat. |
| Marcador | tel:*número de telèfon* | `ACTION_DIAL` | Marca el número de telèfon especificat (però no truca). |
| GoogleMaps | geo:*latitud*,*longitud*<br>geo:*latitud*,*longitud*?z=*zoom*<br>geo:0,0?q=*adreça*<br>geo:0,0?q=*pizzeria* | `ACTION_VIEW` | Obre GoogleMaps a la localització especificada o fent la cerca demanada. |

### 2.4.2 Filtres d"intents'

Hi ha dos tipus d'*intents* diferents:

- Explícits: on s'especifica el component explícitament pel seu nom (el camp *nom del component* de l'`Intent` té un valor introduït). Generalment, no podem saber el nom específic dels components d'altres aplicacions, per això

els *intents* explícits generalment es fan servir per cridar components de la nostra pròpia aplicació.

- Implícits: aquests no tenen definit el nom del component al qual van dirigits. Generalment, es fan servir per activar components d'altres aplicacions.

Els *intents* explícits s'envien a través del sistema a una instància de la classe definida, únicament es fa servir el nom del component (definit al `ComponentName` de l'objecte `Intent`) per determinar quin és el component al qual va dirigit l'*intent*.

Com els *intents* implícits no especifiquen quin és el component al qual va dirigit l'*intent*, el sistema ha de trobar el millor component per tractar l'*intent*. Per fer-ho, compara els continguts definits a l'objecte `Intent` amb unes estructures definides als components que serveixen per *anunciar* al sistema les capacitats del component. Aquestes estructures serveixen per dir al sistema quin tipus d'*intents* és capaç de tractar el component i s'anomenen ***intent filters*** (filtres d'*intents*).

> Els ***intent filters*** serveixen per dir al sistema les capacitats del component. El sistema sabrà a quin tipus d'*intent* pot respondre el component comparant les característiques de l'*intent* amb el filtre d'*intent*.

Per exemple, podeu crear una activitat que serveixi per veure pàgines web. Podeu *anunciar* al sistema que la vostra activitat està oberta a rebre *intents* per visitar llocs web. La propera vegada que es faci un *intent* per visitar un web des del vostre dispositiu, la vostra activitat serà una de les candidates per obrir el web (en el cas que hi hagi diversos components per atendre l'`Intent`, el sistema demanarà a l'usuari quin vol fer servir).

Els filtres permeten que els components puguin rebre *intents* implícits. Si un component no té *intent filters*, únicament pot rebre *intents* explícits (amb el seu nom). Un component que defineix filtres d'*intents* pot rebre *intents* implícits i explícits.

Un component té un filtre diferent per cada treball que pot fer. Per exemple, una aplicació multimèdia pot tenir diferents tipus de filtres per anunciar que pot obrir diferents tipus de fitxers (fotos, àudio, vídeo, etc.).

Un filtre és una instància de la classe `IntentFilter`. El sistema comprova els components per veure quin pot rebre l'*intent* implícit, per aquest motiu ha de conèixer les capacitats del component abans d'engegar-lo. Per això els filtres s'han de definir al fitxer **AndroidManifest.xml** com a elements `<intent-filter>`. L'única excepció són els filtres per als receptors de *broadcast* que es registren dinàmicament cridant el mètode `Context.registerReceiver()`.

Un filtre té camps similars als de l'objecte `Intent`: acció, dades i categoria. Els *intents* implícits es comparen amb els camps del filtre. Perquè l'*intent* sigui enviat al component, la comparació amb els tres camps del filtre ha de ser vàlida. Es realitza un test amb cada camp per separat:

- Acció

- Categoria
- Dades (l'URI i el tipus de dades)

Els extres i els *flags* no es fan servir per determinar quin component rep l'`Intent`.

### Acció

L'element `<intent-filter>` en el fitxer de *manifest* mostra les accions del filtre com a subelements `<action>`. Per exemple:

```
<intent_filter>
  <action android:name="android.settings.SOUND_SETTINGS"/>
  <action android:name="android.settings.WIFI_SETTINGS"/>
  <action android:name="android.intent.action.CALL"/>
</intent_filter>
```

Encara que els objectes `Intent` únicament defineixen una acció, els filtres poden definir-ne més d'una. La llista no pot estar buida, ha de tenir almenys un element `<action>` o bloquejarà tots els *intents*. Per tal que l'objecte `Intent` passi el test del filtre d'acció ha de coincidir una de les accions definides al filtre. Si l'objecte `Intent` no especifica una acció, passarà el test d'acció automàticament i es comprovaran la resta de camps.

### Categoria

L'element `<intent-filter>` també defineix les categories com a subelements, per exemple:

```
<intent_filter>
  <categoryandroid:name="android.intent.category.DEFAULT"/>
  <categoryandroid:name="android.intent.category.BROWSABLE"/>
</intent_filter>
```

Fixeu-vos que es fan servir les cadenes completes per definir les accions i les categories (a diferència del que hem vist per a la creació de l'objecte `Intent`). Així, quan a l'objecte `Intent` definiu la categoria `CATEGORY_BROWSABLE`, al filtre d'*intent* s'ha de definir `android.intent.category.BROWSABLE`.

Perquè l'*intent* passi la comprovació de categoria, totes les categories definides a l'objecte `Intent` han d'estar presents al filtre d'*intents*.

Existeix una petita característica que cal aclarir. Tots els *intents* implícits que s'envien a `startActivity()` es tracten pel sistema com si tinguessin la categoria `CATEGORY_DEFAULT` definida. Per tant, les activitats que volen rebre *intents* implícits han d'incloure `android.intent.category.DEFAULT` en el seu filtre d'*intents*.

Els filtres `android.intent.action.MAIN` i `android.intent.category.LAUNCHER` són excepcions. Defineixen activitats que comencen noves tasques i que estan representades a la pantalla llançadora

d'aplicacions (la pantalla on l'usuari pot escollir entre totes les aplicacions que vol llançar). Poden incloure `android.intent.category.DEFAULT` en la llista de categories, encara que no ho necessiten.

#### Dades

La definició de dades està especificada com un subelement del filtre d'*intents*.

```
<intent-filter>
  <data android:mimeType="image/*"/>
  <data android:scheme="http" android:type="video/*"/>
</intent-filter>
```

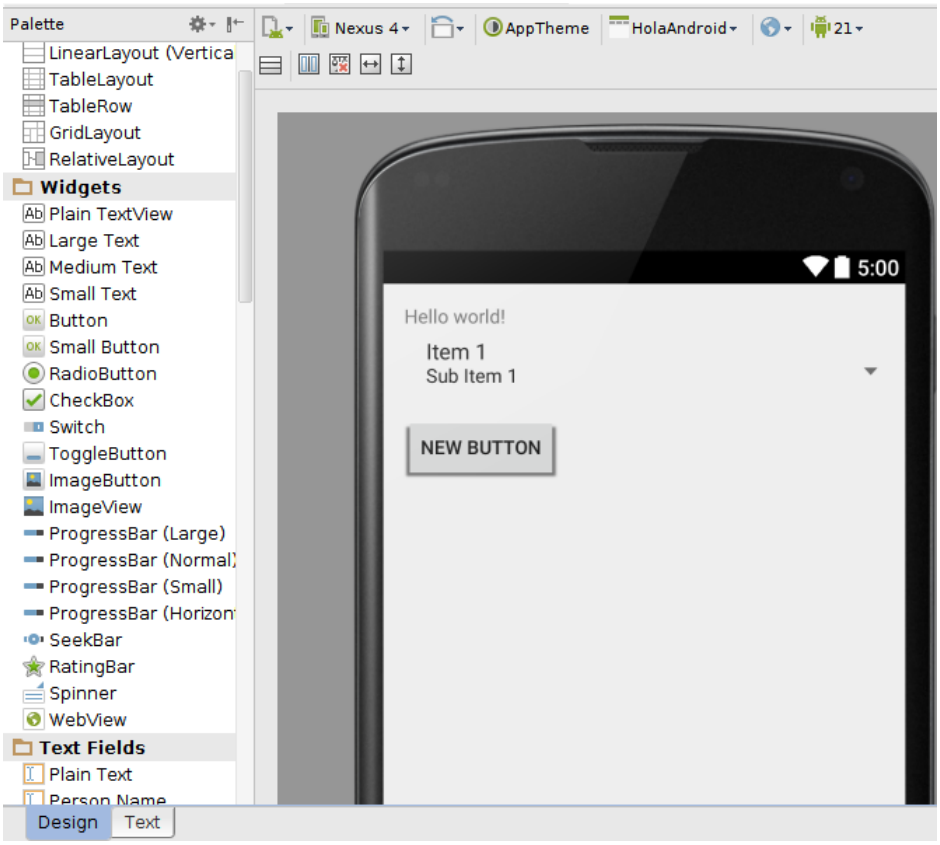
Cada element `<data>` pot especificar un URI i un tipus de dades (MIME). L'atribut `type` de l'element `<data>` especifica un tipus MIME de les dades. En l'exemple, el primer element `<data>` diu al sistema que el component pot obtenir una imatge i mostrar-la. El segon element és un filtre que defineix un esquema i un tipus de dades, que en aquest cas li diu al sistema Android que el component pot obtenir dades de vídeo de la xarxa i mostrar-les.

Com que la majoria de les dades es proporcionen amb proveïdors de continguts, els filtres que especifiquen tipus de dades no especifiquen URI. Tant l'objecte `Intent` com el filtre poden usar un comodí "*" per especificar un subtipus. Per exemple, ''`text/*`'' o ''`audio/*`'' indica que se seleccionarà qualsevol subtipus de `text` o d'`audio` respectivament.

## 2.5 Interfície d'usuari d'Android

Les activitats serveixen per interactuar amb l'usuari i mostren la interfície d'usuari (o **UI**, *User Interface*) de la vostra aplicació, que pot tenir elements gràfics (*widgets*, en anglès) com botons, caixes de text, etiquetes... La UI està definida al fitxer **activity_*.xml**, a la carpeta *res/layout* del vostre projecte. Si l'obriu a l'editor de l'Android Studio, veureu dues pestanyes: *Design* i *Text*, que mostren el contingut del fitxer XML i la interpretació gràfica del mateix, com es pot veure a la figura 2.8.

**FIGURA 2.8.** Edició de la UI des d'Android Studio




Quan s'executa l'aplicació es carrega el fitxer XML amb la UI al controlador d'*event* `onCreate()` de la vostra classe d'activitat, fent servir el mètode `setContentView()`.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hola_android);
    }
```

En la fase de compilació, cada element del fitxer XML és compilat en una classe d'UI d'Android equivalent amb els seus mètodes i atributs.

**Crear la UI**

En general, el més senzill és crear la UI mitjançant un fitxer XML, encara que és possible crear la UI mitjançant programació. En alguns casos, com els videojocs, aquesta darrera opció és preferible.

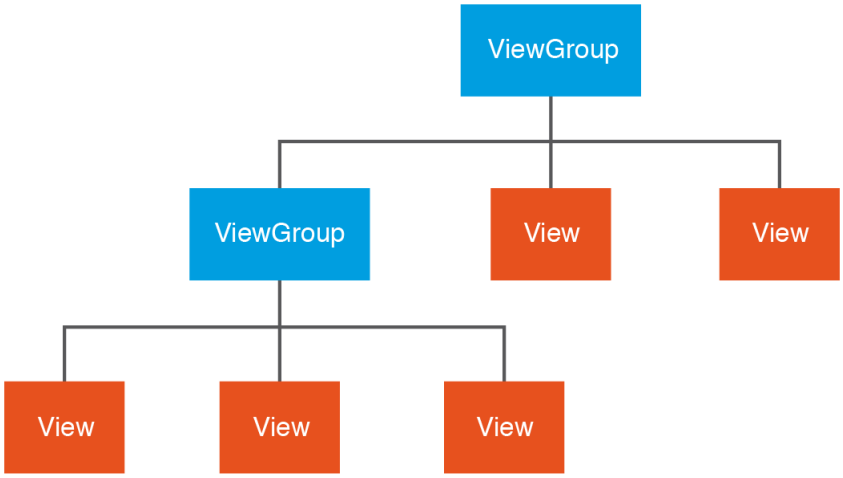
La UI a Android es construeix fent servir objectes `View` (vistes) i `ViewGroup` (grups de vistes). Existeixen moltes *Views* i *ViewGroups*, tots derivats de la classe `View`.

Els objectes `View` són la unitat bàsica de la UI de la plataforma Android. La classe `View`(android.view.View) serveix de classe base per a totes les subclasses anomenades ***widgets***, que ofereixen objectes UI totalment implementats com a camps de text i botons.

La classe `ViewGroup`(android.view.ViewGroup) serveix de classe base per a les subclasses anomenades ***layouts***, que proporcionen diferents tipus de disposicions dels elements. Una o més *Views* es poden agrupar juntes dins d'un *ViewGroup*. El *ViewGroup* (que en si mateix és un tipus de *View*, ja que hereta d'aquesta classe) proporciona una disposició en la qual es mostrarà la seqüència de *Views*.

A Android, es defineix la UI d'una activitat fent servir una jerarquia de *Views* i *ViewGroups*, tal com es pot veure a la figura 2.9.

**FIGURA 2.9.** Jerarquia que defineix una UI d'una activitat




Per enllaçar l'arbre jeràrquic de vista a la pantalla, l'activitat ha de cridar `setContentView()` i passar-li com a referència l'objecte al node arrel de la jerarquia. El node arrel de la jerarquia va cridant els seus nodes fills (els altres *Views* i *ViewGroups*) per tal que, finalment, tots es dibuixin en pantalla.

Android té els diferents tipus de *ViewGroups*:

- LinearLayout
- GridLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView

El més habitual és combinar els diferents tipus de *layout* per crear la UI que volem (tal com es mostra a la figura 2.9, inserint-ne uns dins dels altres).

### 2.5.1 'Layout'

La forma més comuna de definir el *layout* de l'activitat i expressar la jerarquia de *Views* és a través d'uns fitxers XML de *layout*, que proporcionen una estructura del *layout* que resulta fàcil de seguir pel programador. Cada element XML és un objecte `View` o `ViewGroup`. Els objectes `View` serien les fulles de l'arbre i

els objectes `ViewGroup` les branques de l'arbre (fixeu-vos en la figura 2.9). Per exemple, el fitxer XML d'un *layout* lineal vertical amb quatre botons seria el següent:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

  <Button
    android:id="@+id/btnMapa"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Mapa"/>

  <Button
    android:id="@+id/btnWeb"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Web"/>

  <Button
    android:id="@+id/btnContactes"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Contactes"/>

  <Button
    android:id="@+id/btnTrucar"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Trucar"/>

</LinearLayout>
```

Fixeu-vos com el `LinearLayout` conté els elements Button. Es podria niuar un altre `LinearLayout` (o un altre tipus de *ViewGroup*) dins seu per afegir complexitat al *layout*. Si voleu saber més sobre els fitxers de *layout* XML, podeu consultar la *Guia del desenvolupador* d'Android a developer.android.com/guide/topics/ui/declaring-layout.html.

Si voleu veure les diferents característiques dels diferents *layouts*, les podeu trobar a developer.android.com/guide/topics/ui/layout-objects.html.

### 2.5.2 Creació del 'layout'

Per mostrar els diferents tipus de *layout* que existeixen es poden crear UI mitjançant l'edició d'un fitxer XML o mitjançant l'editor gràfic que proporciona Android Studio. Per obrir-lo, obriu el fitxer **.xml** que es troba a la carpeta *res/layout* del projecte.

Disposem de dues maneres fer de servir els *layout* existents:

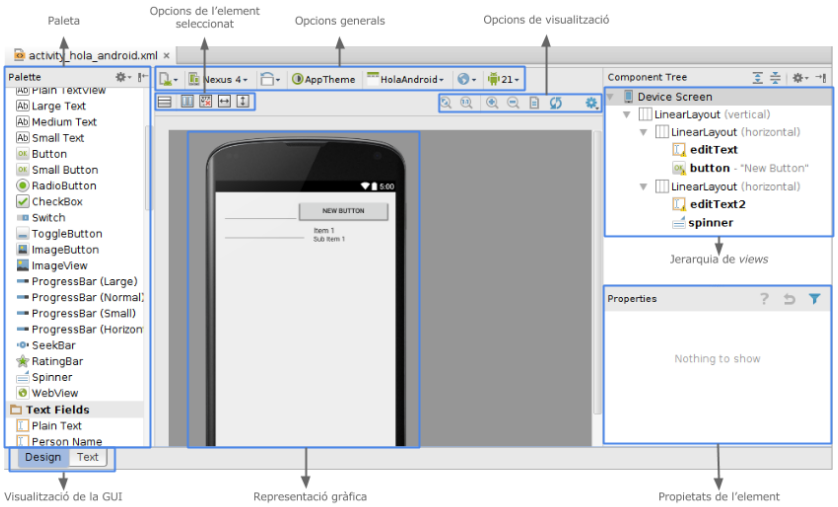
- Mitjançant l'edició d'un fitxer XML

- Mitjançant l'editor gràfic que proporciona Android Studio

Per fer servir l'editor gràfic d'Android Studio cal que obriu el fitxer **XML** que es troba a la carpeta *res/layout* del projecte.

A la figura 2.10 podeu veure les diferents parts de l'editor.

**FIGURA 2.10.** Editor de GUI a Android Studio




Les diferents parts són les següents:

- **Visualització de GUI** (*graphic user interface*, interfície gràfica d'usuari): amb aquestes pestanyes podeu escollir si voleu veure la GUI com un fitxer de text de tipus XML (pestanya "Text") o bé la seva representació gràfica (pestanya "Design").
- **Opcions generals**: inclou algunes opcions per visualitzar el *layout*, segons:
  - El tipus de dispositiu: mida de pantalla i resolució.
  - El seu format (emmarcat o apaïsat).
  - El seu idioma.
  - La versió de l'SDK.
  - El tema disponible.
- **Jerarquia de *Views***: a la dreta podeu veure la jerarquia de *Views* i *ViewGroups* que formen part de la GUI. Podeu fer clic a cadascuna d'elles per seleccionar-les i així modificar les seves propietats al requadre corresponent. Feu servir aquesta representació per tenir clar com està estructurada la vostra GUI.
- **Representació gràfica**: aquí podeu veure com es mostrarà la vostra interfície gràfica. Podeu seleccionar els *Views* i *ViewGroups* i desplaçar-los per les diferents parts de la GUI. Igualment, fent clic sobre cada element podreu modificar les seves propietats.

- **Paleta**: aquí teniu els diferents *Views* i *ViewGroups* que podeu fer servir a la vostra GUI. Per afegir qualsevol d'aquests *Views*, els podeu arrossegar amb el ratolí a la representació gràfica del GUI o dins de la jerarquia de *Views*. De vegades és més senzill introduir els *Views* a la jerarquia a través de la paleta. Feu clic a les diferents carpetes (*Widgets*, *Text Fields*, *Layouts*...) per fer una ullada als diferents elements gràfics amb els quals podeu treballar.
- **Opcions de visualització**: us permeten modificar la forma en què es visualitza la representació gràfica de la GUI.
- **Opcions de l'element seleccionat**: ofereix diferents opcions que es poden aplicar a l'element seleccionat. Les opcions varien entre els diferents *Views*; per tant, quan seleccioneu un *View* cal que us fixeu en quines són les seves opcions.

Android Studio dóna ajuda contextual sobre el cursor del ratolí. Si deixeu el cursor sobre els diferents botons de l'editor, us apareixerà una petita etiqueta explicant què fa aquest botó.

Els *Views* i *ViewGroups* tenen una sèrie d'atributs comuns que podeu veure a la taula 2.4.

**Taula 2.4.** Atributs comuns de View i ViewGroup

| Atribut | Descripció. |
|---|---|
| `layout_width` | Amplada del View. |
| `layout_height` | Alçada del View. |
| `layout_marginTop` | Especifica l'espai extra a dalt del View. |
| `layout_marginBottom` | Especifica l'espai extra a baix del View. |
| `layout_marginLeft` | Especifica l'espai extra a l'esquerra del View. |
| `layout_marginRight` | Especifica l'espai extra a la dreta del View. |
| `layout_gravity` | Especifica com els Views fills es posicionen. |
| `layout_weight` | Especifica quant de l'espai extra del *layout* s'hauria de reservar per al View. |
| `layout_x` | Especifica la coordenada *x* del View. |
| `layout_y` | Especifica la coordenada *y* del View. |

**Atributs del View**

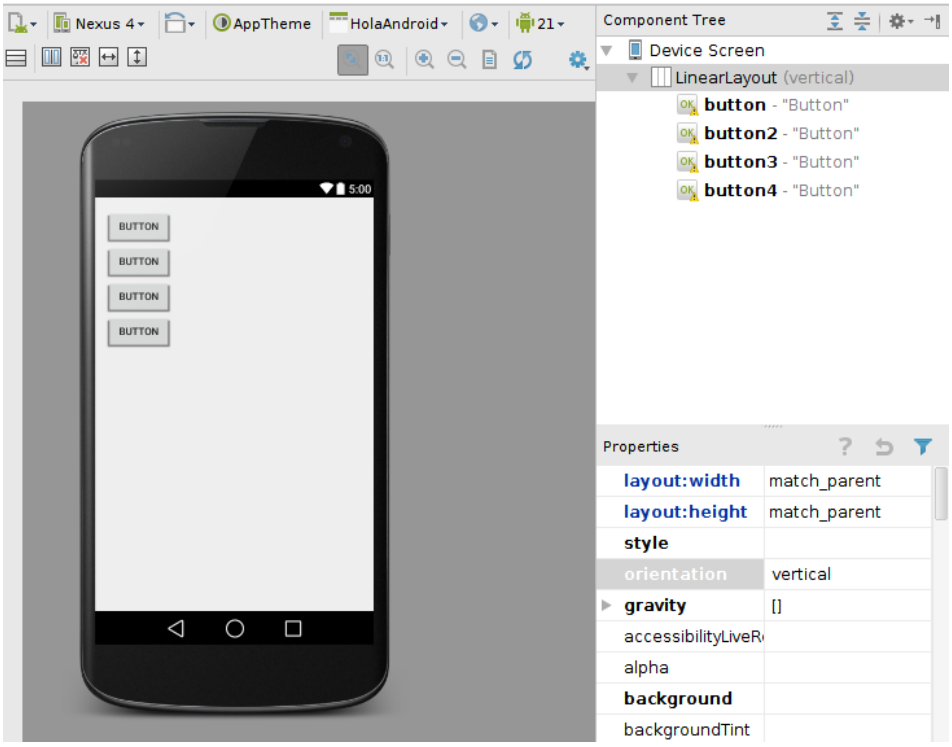
Alguns dels atributs del View únicament són aplicables quan el View està dins d'un ViewGroup. Per exemple, els atributs `layout_weight` i `layout_gravity`únicament s'apliquen quan el View està dins d'un `LinearLayout` o `TableLayout`.

Podeu veure els diferents tipus de *layouts* disponibles a la *Guia del desenvolupador* d'Android: developer.android.com/resources/tutorials/views/index.html.

Un `LinearLayout` mostra els *Views* en una única fila o columna, vertical o horitzontalment. Per exemple, a la figura 2.11 podeu veure un `LinearLayout` on hem introduït quatre botons.

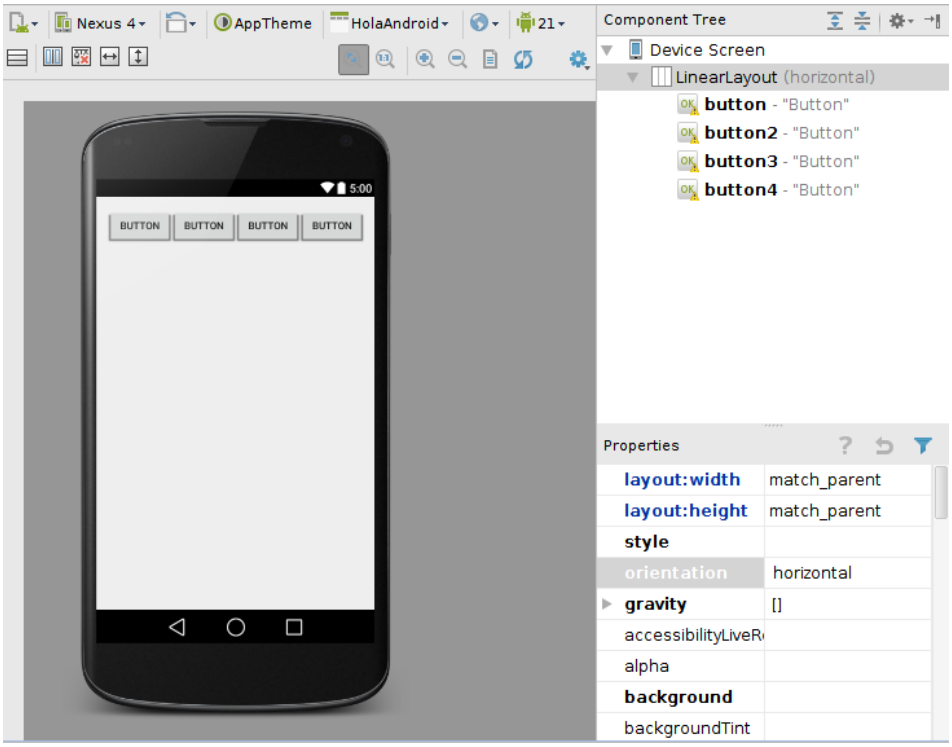
Tot seguit veurem el funcionament de `LinearLayout`; hem triat aquest *layout* perquè és el més senzill de fer servir. Si feu una ullada a la part de la documentació oficial que parla dels diferents *layouts* podreu veure altres estils gràfics disponibles.

**FIGURA 2.11.** Amb un 'Linear Layout' els components es distribueixen linealment



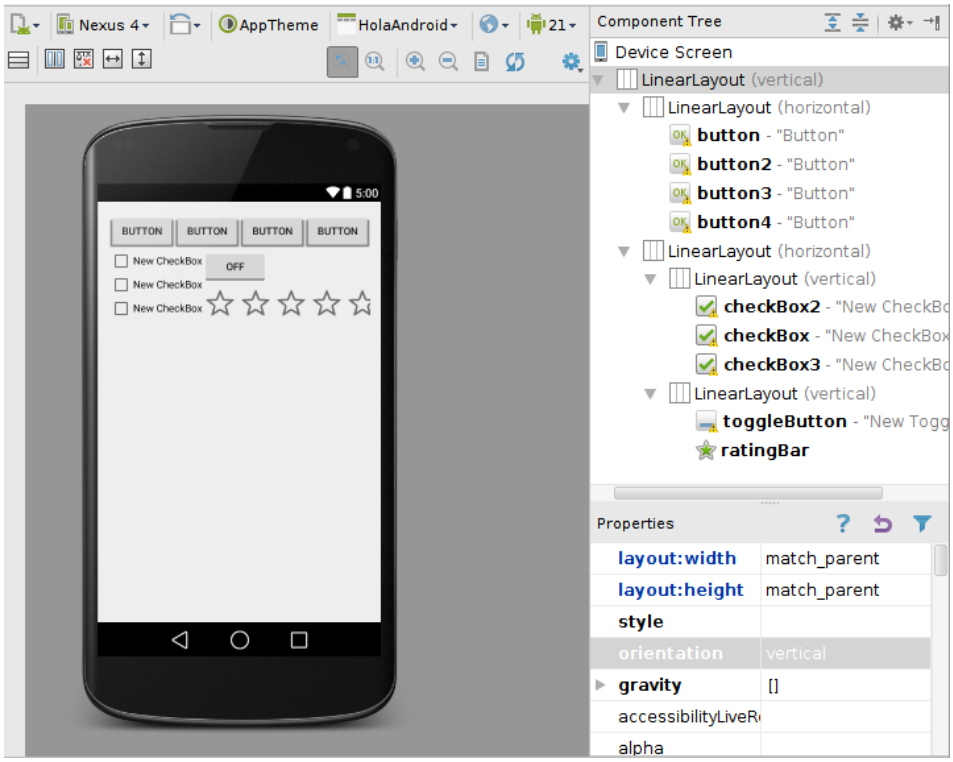

Clicant el botó dret del ratolí sobre `LinearLayout` podeu veure i modificar a la jerarquia de *Views* algunes de les seves propietats. Per exemple, si canvieu l'orientació a *Orientation/Horizontal* veureu com els botons es disposen de manera horitzontal, tal com es pot veure a la figura 2.12.

**FIGURA 2.12.** 'Linear Layout' horitzontal



Podeu niuar els *layouts* els uns dins dels altres per crear disposicions més complexes, tal com es pot veure a la figura 2.14.

**Figura 2.13.** 'Linear Layout' niuats




Podeu modificar les propietats de cada `View` seleccionant-les en el requadre *Properties*. Per exemple, podeu canviar el valor de l'ID que identifica el `View`, o les seves dimensions d'amplada i alçada. En la figura 2.14 podeu veure tres botons amb amplades diferents. El primer té l'amplada que ve per defecte, al segon li hem assignat el valor `Fill Parent` (omplir pare) a la propietat `Layout Width`(amplada del *layout*), i al tercer li hem assignat per aquesta mateixa propietat el valor de 100 dp (100 punts de pantalla).

**Figura 2.14.** Diferents valors de 'Layout Width'

Finalment, a partir del *layout* que creeu mitjançant les eines gràfiques es generarà un fitxer XML amb el qual el sistema Android sabrà com crear la interfície gràfica en temps d'execució. Per exemple, el *layout* de la figura 2.14 correspon al següent fitxer XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"/>

    <Button
        android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button"/>

    <Button
        android:id="@+id/button3"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="Button"/>

</LinearLayout>
```

Al començament és normal que feu servir l'editor gràfic per crear els vostres *layouts*, però quan aneu agafant experiència veureu que editar directament el fitxer XML fa molt més ràpida l'edició del *layout*.

### Unitats de mesura

Per especificar la mida d'un element a la interfície gràfica d'Android es poden fer servir les següents unitats de mesura:

- **dp** (*density-independent pixel*, píxel independent de la densitat). És la recomanada per especificar la mida dels *Views* en el vostre *layout*.
- **sp** (*scale-independent pixel*, píxel independent de l'escala). Similar a la dp i recomanada per especificar mides en general.
- **pt** (*point*, punt). Un punt està especificat com 1/72 d'una polsada; està basat en la mida física de la pantalla.
- **px** (pixel). Correspon als píxels reals en pantalla. Aquesta mida no es recomana, atès que la vostra interfície gràfica es pot mostrar de formes diferents en els diferents dispositius.

### 2.5.3 'Layouts' XML

El *layout* defineix la forma en què els elements gràfics de l'activitat es mostren a l'usuari. Es pot declarar el *layout* de dues maneres diferents:

- **Declarant els elements gràfics a un fitxer XML**. Aquest fitxer contindrà les *Views* i les seves subclasses que definiran l'UI. L'avantatge d'aquest sistema és que permet separar la presentació de l'aplicació del codi que la controla. Així, podeu modificar la UI (o adaptar-la a un altre dispositiu) sense haver de recompilar el projecte. Per exemple, podeu crear diferents *layouts* XML per a diferents mides de pantalla, diferents orientacions de la pantalla o diferents llenguatges.
- **Instanciant els elements del *layout* en temps d'execució**. Es poden crear els elements gràfics des del mateix programa. Això és útil quan voleu crear una UI en funció d'alguna cosa que no sabeu en el moment de dissenyar l'aplicació (per exemple, en funció d'unes dades que rebeu per Internet).

Podeu fer servir qualsevol dels dos mètodes (tot i que generalment fareu servir el primer), i fins i tot ambdós a la vegada. Podeu dissenyar la UI de l'aplicació i després, en temps d'execució, modificar l'estat dels elements que estan en pantalla.

En general, el vocabulari XML per declarar interfícies d'usuari té una estructura similar als noms corresponents a les classes i mètodes que representen. Així, és fàcil saber a quines classes i atributs corresponen els elements de l'XML (encara que no sempre són idèntiques).

Es poden crear *layouts* d'UI i els elements que contenen ràpidament, fent servir un fitxer XML de la mateixa manera que es creen les pàgines web amb HTML, amb una sèrie d'element niuats. Cada fitxer *layout* ha de tenir un element arrel, que ha de ser un objecte `View` o `ViewGroup`. Una vegada definit l'element arrel, podeu afegir *widgets* o objectes de *layout* com a elements fills de l'arrel per crear a poc a poc la jerarquia de *Views* que definirà el vostre *layout*. Per exemple, podeu veure tot seguit el fitxer XML d'un *layout* que usa un `LinearLayout` i que inclou a dins un `TextView` i un `Button`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

<TextView
  android:id="@+id/text"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hola, sóc un TextView"/>

<Button
  android:id="@+id/button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Hola, sóc un Button"/>
</LinearLayout>
```

### Carregar el recurs XML

Podeu trobar els fitxers d'android.R a dins del directori *app/build/generated/source/r/debug*, accessibles des de la vista *Project* de l'explorador del projecte.

Els fitxers XML de *layout* són compilats a la vostra aplicació dins d'un recurs *View*. Això es produeix quan, en iniciar-se l'aplicació, és cridat el mètode de *callback* `onCreate()` de la vostra activitat, mètode a través del qual es carrega el recurs del *layout* des del codi. Per dur a terme aquest procés, crideu el mètode `setContentView (int layoutResID)`, que té com a argument un identificador de recurs de *layout*. Aquest ID, el podeu trobar a la classe que conté els recursos `android.R`. Concretament, els identificadors de *layouts* es troben dins de `R.layout.`*`nom_del_fitxer_layout`*. Així, si el nostre *layout* està contingut al fitxer **principal.xml**, trobareu l'identificador a `R.layout.principal`. La forma de carregar aquest recurs seria la següent:

```
1 public void onCreate(Bundle savedInstanceState)
2 {
3   super.onCreate(savedInstanceState);
4   setContentView(R.layout.principal);
5 }
```

### Atributs

Els objectes de les classes `View` i `ViewGroup` tenen els seus propis atributs. Alguns d'ells són comuns a tots els *Views*, com l'ID, mentre que d'altres són específics de cada tipus de *View*; per exemple els `TextView` tenen l'atribut `textSize`, que no està present en tots els *Views*.

Tots els objectes `View` tenen un número d'ID al qual estan associats, i que identifiquen el *View* de forma unívoca. Aquest ID està present en forma de *string* al fitxer XML, encara que en compilar l'aplicació és referenciat mitjançant un *Integer*. Aquest atribut és present a tots els objectes `View` i derivats d'ell. La sintaxi per definir l'ID al fitxer XML és la següent:

```
1 android:id="@+id/el_meu_boto"
```

El símbol + diu al compilador que es tracta d'un nou recurs i que s'ha de crear i afegir als recursos del projecte (que es troben al fitxer "R.java"). Quan es fa referència a un ID de recurs d'Android, s'ha de fer així:

```
1 android:id="@android:id/empty"
```

Introduint el *namespace* del paquet, esteu referenciant l'ID de la classe de recursos `android.R`.

La forma habitual de crear Views i referenciar-los des de l'aplicació és la següent:

En primer lloc, cal definir el `View` en el fitxer XML de *layout* i assignar-li un ID únic:

```
1 <Button android:id="@+id/botoAcceptar"
2   android:layout_width="wrap_content"
3   android:layout_height="wrap_content"
4   android:text="@string/acceptar"/>
```

Tot seguit creeu la instància de l'objecte `View` (o classe derivada d'ell) que correspongui i referencieu-lo al del *layout* (generalment es fa servir la funció de *callback* `onCreate()`):

```
1 Button Boto =(Button) findViewById(R.id.botoAcceptar);
```

A partir d'aquest moment ja podeu començar a treballar amb el botó des del codi.

### 2.5.4 'Widget' i events de teclat

Un ***widget*** és un objecte `View` que serveix d'interfície d'interacció amb l'usuari. Existeixen multitud de *widgets* ja implementats a Android, com botons, *checkboxes*, caixes de text, etc. El programador pot crear els seus propis *widgets* definint una classe que sigui descendent de `View` o d'un *widget* ja existent (per exemple, pot heretar les propietats d'una caixa de text).

Una vegada afegits els *widgets*, caldrà programar la interacció entre el programa i l'usuari. Cal que captureu els *events* de l'objecte `View` amb què l'usuari està interactuant i fer que succeeixi quelcom (executar una part de codi).

La classe `View` (de la qual els *widgets* hereten les propietats) té una sèrie de **mètodes de*callback*** que serveixen per tractar els *events* de la interfície gràfica. Aquests mètodes són cridats pel sistema quan l'acció corresponent succeeix en aquest objecte. Per exemple, quan es toca un botó, el sistema crida el mètode de *callback* `onTouchEvent()`.

La classe `View` té una sèrie d'**interfícies** anomenades `On<quelcom>Listener`, cadascuna amb un mètode de *callback* anomenat `on<quelcom>()`. Per exemple, `OnClickListener` (per tractar els clics a un *View*) defineix el mètode `onClick()`.

Aquestes interfícies, anomenades ***event listeners*** (*escoltadors* d'esdeveniments), us permeten capturar la interacció de l'usuari amb la UI. Si voleu que el vostre `View` sigui notificat en ser clicat, heu d'implementar l'interfície `OnClickListener`, definir el mètode `onClick()`, que establirà l'acció que s'ha de realitzar quan es faci clic, i registrar-lo al View amb `setOnClickListener()`.

> Un ***event listener*** és una interfície de la classe `View` que conté un únic mètode de *callback*. Aquests mètodes seran cridats pel sistema Android quan el `View` (al qual està registrat l'*event listener*) s'activa per la interacció de l'usuari.

Per exemple, podeu definir un *event listener* amb un mètode de *callback* i registrar-lo a un `View` de tipus botó. Quan l'usuari premi el botó, el sistema cridarà el mètode de *callback* definit i podreu executar el codi que vulgueu.

Per això heu de definir un *event listener* i registrar-lo al View. Un *event listener* és un mètode que es cridarà pel sistema quan es produeixi un esdeveniment en concret, per exemple quan es premi un botó.

Per registrar un event listener, en primer lloc heu d'obtenir una referència al View sobre el que voleu definir la funció de callback. Per obtenir la referència podeu fer servir el mètode `findViewById(int id)` de l'objecte de l'activitat. L'argument del mètode (l'identificador) és el que està definit al fitxer AndroidManifest.xml i el podeu obtenir a partir de l'objecte `R.id`.

Per exemple, per obtenir una referència a un botó anomenat "buttonSi" faríeu el següent:

```
Button btnSi;

//Listener delbotoSi
btnSi = (Button) findViewById(R.id.buttonSi);
```

Podeu trobar tots els identificadors de *widgets* de la vostra activitat a la variable `R.id`. Una vegada tingueu la referència al botó, fareu una crida al mètode `setOnClickListener (View.OnClickListener l)`. Aquest mètode registra el *callback* que serà invocat quan aquest View sigui clicat. Com a únic argument, se li passa la funció de *callback* que s'executarà.

Ho podeu fer de dues maneres diferents. A la primera, senzillament, definiu la funció de *callback* a la mateixa crida a `setOnClickListener()`. Per exemple:

```
btnSi.setOnClickListener(

  new OnClickListener()
  {
    @Override
    publicvoid onClick(View v)
    {
      //Codi que s'executarà quan es faci clic
      Toast.makeText(c, "Boton si", Toast.LENGTH_LONG).show();
    }
  }
);
```

En aquest codi es crida`setOnClickListener()` i com a argument es fa una implementació de la interfície `OnClickListener()` que té un únic mètode per definir, onClick(). Dins d'aquest mètode estarà el codi que s'executarà quan es premi aquest botó. En aquest cas, el codi correspon a mostrar un missatge per pantalla amb `Toast`.

Això s'hauria de fer per cada View que existeixi al *layout* i amb què es vulgui interactuar.

La segona forma consisteix a implementar `OnClickListener` com a part de l'activitat. Això evitarà haver de crear totes les implementacions de `OnClickListener`, simplement diem que la nostra activitat implementa la interfície `OnClickListener` afegint-ho a la seva definició amb:

```
public class ProvabotonsActivity extends Activity implements OnClickListener {
```

Ara, l'activitat ja implementa la *interface*, i únicament s'ha de sobreescriure el mètode `OnClick(View v)`. En aquest cas es cridarà aquest mètode independentment de quin View hagi estat clicat, per tant el primer que s'haurà de fer és comprovar quin ha sigut l'objecte clicat, com es pot veure al següent codi:

```java
public class ProvabotonsActivity extends Activity implements OnClickListener {

Button btnSi, btnNo;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  //Definim els listeners
  btnSi = (Button)findViewById(R.id.buttonSi);
  btnSi.setOnClickListener(this);
  btnNo = (Button)findViewById(R.id.buttonNo);
  btnNo.setOnClickListener(this);
}

public void onClick(View v)
{
  // Fer alguna cosa quan s'ha polsat un View.

    // Comprovem quin ha estat el View clicat.
    if(v== btnSi)
    {
      //Executem el codi
     Toast.makeText(this, "Botó si", Toast.LENGTH_LONG).show();
    }
    else if(v == btnNo)
    {
      Toast.makeText(this, "Botó no", Toast.LENGTH_LONG).show();
    }
  }
}
```

A les *interfaces* d'*event listeners* existeixen el següents mètodes de *callback*:

- `onClick()`: de `OnClickListener`. Es crida quan l'usuari fa clic en l'ítem.
- `onLongClick()`: de `OnLongClickListener`. Es crida quan l'usuari fa una pulsació llarga damunt de l'ítem.
- `onFocusChange()`: de `OnFocusChangeListener`. Es crida quan l'usuari navega en o fora de l'ítem fent servir els cursors o el *trackball*.
- `onKey()`: de `OnKeyListener`. Es crida quan l'usuari té el focus a l'ítem i prem o solta una tecla del dispositiu.
- `onTouch()`: de `OnTouchListener`. Es crida quan l'usuari fa una acció de *tocar* l'ítem (inclou prémer, soltar o un gest sobre l'ítem).
- `onCreateContextMenu()`: de `OnCreateContextMenuListener`. Es crida quan un menú contextual es crea sobre l'ítem (resultat d'una pulsació llarga o de la pulsació de la tecla de menú).