Diapositiva 1 Presentación.

Agradecer la presencia a los asistentes.
Quién soy y a que me dedico.

Diapo 2:
Apis que vamos a comentar:
Fragment, loadres, actionBar, arrastrar y soltar y animaciones.

Diapositiva 3:

Nueva filosofía para diseñar actividades.
Mas flexibilidad para diseñar pantallas alargadas.

Simplifica la complejidad de la jerarquía de componentes(views), cada fragment tiene su ciclo de vida.
Permite una mayor reusabilidad.

Diapositiva 4:

El ciclo de vida del fragment se sincroniza con el ciclo de vida de la actividad , si la actividad recibe un onPause este se propaga al onPause de cada actividad.

Ademas los fragments añaden nuevos métodos.

onAttach()
Se lanza cuando el fragment se asocia con la actividad, podemos usarlo para implementar callbaks hacia al actividad

onCreateView()
Se llama para crear la jerarquía de views del fragment, debe devolver el parent view del fragment.

onActivityCreated()
Se llama al terminar el método onCreate() de la actividad que contiene el fragment

onDestroyView()
Se llama cuando destruimos un fragment

onDetach()
se llama cuando un fragment es desasociado de la actividad
Cuando la actividad alcanza el estado de resuida, podemos trabajar libremente con cada fragment, sin afectar al ciclo de vida de la actividad. Si sale del estado de resumida el ciclo de vida del fragment vuelve a ser dependiente del de la actividad.

Diapo 5:

Principales clases para implementar Fragment.

Fragment
    Clase base de la que heredamos nuestros fragment personalizados
DialogFragment
    Muestra un dialogo flotante, con la ventaja de poder añadirlo a la pila de fragments
ListFragment
    Muestar un listado de elementos, como un ListActivity
PreferenceFragment
    Muestra una pagina de preferencias, con un PreferenceActivity.

Diapo 6:
Para crear un fragment heredamos de la clase Fragment o alguna de sus subclases e implementamos el método onCreateView para que devuelva la jerarquía de views que contienen el fragment. Similar al concepto de getView de BaseAdapter.

The inflate() method takes three arguments:

The resource ID of the layout you want to inflate.

The ViewGroup to be the parent of the inflated layout. Passing the container is important in order for the system to apply layout parameters to the root view of the inflated layout, specified by the parent view in which it's going.

A boolean indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation. (In this case, this is false because the system is already inserting the inflated layout into the container—passing true would create a redundant view group in the final layout.)

ListFragment es la excepción, al = q ListActivity implementa su propia interfaz de usuario y no necesitamos implementar este método, salvo que queramos una vista personalizada.

Diapo 7:
Para añadir una fragment a una activity desde XML

Creamos la etiqueta fragment con los atributos:
Name: nombre completo del fragment
Id o tag: cadena única para identificar el fragment con findviewbyid o con findFragmentbyTag

Diapo 8:
Para añadir un fragment desde código.
Obtenemos el gestor de fragments
Comenzamos una transacción de fragment
Creamos nuestro fragment
Añadimos el nuevo fragment a la transacción, indicando un ViewGroup donde añadirlo y el fragment que vamos a añadir
Confirmamos la transacción

Diapo 9:

Podemos gestionar nuestros fragment con los métodos add, remove() y replace() para añadir, reemplazar o eliinar fragment de la actividad

Podemos usar addToBackStack() para añadir la operación realizada con el fragment a la pila de fragment.

Podemos recuperar un fragment usando findFragmentbyId o findFragmentbyTag

Diapo 10:

Podemos comunicarnos con el fragment desde la activity usando los métodos findFragmentbyId o findFragmentbyTag

Tambien podemos crear un listener en la actividad que dispararemos desde el fragment, el listener debe ser añadido al fragment en el método onAttach del fragment y debemos recordar eliminarlo en el onDeatach
(Mostrar código de la aplicación)

Sample_fragment_layout: Declaracion fragment en xml

Comunicación Fragment/Activity:
Declarar llamada ContactFragment 40, 46, 109
Remplazar Fragment: SampleFragment 22-39
Remplazar Fragment: DetailsFragment 210-222

Diapo 11:
Las funcionalidades que nos ofrecen los Loaders son:
Cargar datos de forma asíncrona, facilitándonos la codificación de AsyncTask, Threads y Handlers.
Monitorizan el origen de datos y nos avisan de cambios en dichos datos de forma automática.
Se reconecta de forma automática y actualiza las peticiones de datos d eforma automática.

Diapo 12:

LoaderManager: clase que se encarga de gestionar los distintos Loaders que tengamos en una actividd o fragment.

LoaderCallback: Es la interfaz para interactuar nuestra activity/Fragment con el LoaderManager,

Loader: Es la clase base para implementar nuestros propios Loaders, es la encargada de realiar todo el trabajo de carga de datos de forma asíncrona.

AsyncTaskLoader: Es una implementación de Loaders, que usa un AsyncTask para realizar el trabajo en segundo plano.

CursorLoader: Es una subclase de AsyncTask que realiza una petición a un ContentResolver y nos devuelve un cursor ocn los datos obtenidos.

Diapo 13:
Para usar un loader, primero cargamos el loader con getLoaderManager().initLoader();

Recibe un id único que identifica al loader.
Argumentos adicionales para construir el loader.
La CallBack donde irá informándonos el loader de sus progresos y volcando los datos.

Con el método restartLoader(id, argumentos, callback),
Este método se encarga de reiniciar nuestro loader para que actualice los datos de nuestra petición.

La función de callback de un Lader contiene tres métodos:
onCreateLoader: Instancia y devuelve un nuevo loader para el identificador dado.

onLoadFinished: Nos avisa cuando el loader a termina de obtener los datos. Y nos devuelve los datos obtenidos.

onLoadReset: Se lanza cuando un loader a ser reseteado y sus datos ya no estarán disponibles.
(Mostrar código de la aplicación)
Iniciar Loaders ContactFragment: 65
Implementacion Loaders ContactFragment 123-152
Iniciar Loaders DetailsFragment: 80
Implementacion Loaders DetailsFragment 96-124

Diapo 14:
ActionBar es el widget que sustituye a la tradicional barra de titulo de android. Por defecto incluye el logo de la aplicación y el titulo de la activity que se está mostrando.

Es el sustituto del menú tal y como lo conocemos.
Podemos añadir tabs para navegar entre los distintos fragment.
Provee de listas de selección para facilitar la navegación.
Añade el concepto de ActionView para mejorar el ActionItem

Diapo 15:
Para añadir la actionbar a nuestra aplicación simplemente decimos que el dispositivo destino de nuestra app es un honeycomb.

Para eliminarla desde el manifest seleccionamos el tema Theme.Holo.NoActionBar

Para eliminarla desde código, llamamos a su método hide()

Diapo 16:

Para añadir opciones al actionBar lo hacemos igual que para el menú tradicional, mediante onCreateOptionsMenu(), y cargamos los elementos del menú desde código o desde el elemento <menú><ítem>. La nueva funcionalidad para los ActionItem es poder elegir como se muestran medinate las flags:

public static final int SHOW_AS_ACTION_ALWAYS
Obliga a que se muestre siempre en forma de botón, hay que llevar cuidado de no saturar la action bar, se recomienda no poner mas de dos.

public static final int SHOW_AS_ACTION_IF_ROOM
Muestra el elemento del menú como un botón si dispone de espacio, es lo recomendado

public static final int SHOW_AS_ACTION_NEVER
Nunca muestra el elemento del menú como un botón, lo introduce en la lista desplegable de la derecha del actionbar

public static final int SHOW_AS_ACTION_WITH_TEXT
Siempre muestra el elemento como texto, aunque tenga icono.

Los eventos se gestionan en el onOptionsItemSelected() del fragment que añadió el elemento, y previamente en la actividad que contiene el fragment que añadió el elemento.

Podemos gestionar el tap en el icono de la activity usando el itemId android.R.id.home, Este icono suele tener dos comportamientos, volver al estado inicial de la actividad actual, para este comportamiento se asocia el icono de la aplicación.

El otro comportamiento representado por el icono de la aplicación con la flecha, se usa para navegar por la jerarquía de activities, para activar la flecha en el icono usamos la función setDisplayHomeAsUpEnabled(true),.

La funcionalidad de estos comportamientos debemos implementarla nosotros, respondiendo al itemid android.R.id.home.

Diapo 17:
En el action Bar podemos añadir Views, se hace igual que para añadir actionItems, pero añadiendo la propiedad actionLayout o actionViewClass, desde código usamos la llamada a setActionView()

Diapo 18:
Un action Bar puede mostrar Tabs y podemos usar estos tabs paa navegar por los distintos fragment de una activity, para añadir tabs al action Bar debemos implementar el TabListener, usando sus métodos onTabSelected para añadir un nuevo fragment a la activity, el asociado a la pestaña, el onTabUnselected para eliminar el fragment asociado a la pestaña que se deselecciona y el OnTabReselected para evitar crear una y otra vez el mismo fragment y aprovechar los que ya hemos instanciado.

Debemos decirle al action bar que vamos a usar el modo de navegación por pestañas, llamando a la función setNavigationMode(NAVIGATION_MODE_TABS), creamos todas las pestañas que necesitemos usando el método newTab, le añadimos la etiqueta y el icono a cada pestaña, a cada pestaña debemos asociarle su implementación del TabListener, Añadimos cada pestaña creada usando el método addTab de ActionBar.

Diapo 19:
Para añadir una lista de navegación debemos especificar al action Bar que vamos a usar este componente, llamando a la función setNavigationMode(NavigationModeList) en el onCreate de la activity, creamos un SpinerAdapter con los valores que contendrá la lista de selección e implementamos un listener del tipo OnNavigationListener e implementamos su método onNavigationItemSelected, que será el encargado de cambiar el fragment que debe mostrarse. Pasamos el spinerAdapter y el listener al método setListNavigationCallbacks.

(Mostrar código de la aplicación)
Configurar uso del icono SampleFragment: 18
Responder a las acciones de ActionBar SampleFragment 46-76
Declarar Actionitems menu > menú_principal.xml
Añadir elementos a la action Bar ContactsFragment 59
Añadir ActionView a la Action Bar: 161-171

Añadir elementos a la action Bar DetailsFragment 62
Añadir ActionItems a la Action Bar DetailsFragment 137 - 161

Diapo 20:
Honeycomb permite realizar Drag&Drop de forma muy senclla através de una actividad, moviendo elementos dentro de un fragment o entre fragments, para codificar esta funcionalidad debemos tener encuenta los siguientes procesos:

Con este framework podemos mover datos de un componente(view) a otro. dentro de una misma activity, incluso entre distintos fragments.

Hay cuatro pasos básicos a la hora de usar este framework.

started: Con la llamada a startDrag() indicamos al sistema que comenzamos con el proceso de arrastrar, el sistema responde pintando la sombra

para nuestro arrastre e informando a todso los componentes que implemente el evento drag con la acción ACTION_DRAG_STARTED, si queremos que nuestro componente
siga recibiendo estos eventos durante el proceso de arrastre debemos responder con un valor de true a la acción citada. Esto registra al componente para seguir recibiendo eventos.
Devolviendo false decimos al sistema que no estamos interesados en recibir estos eventos.

Continuing: Este proceso se alarga durante el proceso de arrastar el componente, si el usuario introduce el componente que está arrastrando dentro de un componente registrado para recibir
eventos de drag, este recibirá al accion ACTION_DRAG_ENTERED

Dropped: cunado el usaurio levante el dedo se generará la acción ACTION_DROP sobre el componente sobre el que se ha soltado el componente arrastrado.

Ended: El sistema lanza la acción ACTION_DRAG_ENDED a todos los componentes registrados en el evento DRAG, para indicar que ha terminado el proceso de arrastrar.

Diapo 21:

Los eventos DRAG.
Para habilitar un componente para recibir eventos drag, tenemos dos metodos, sobreescribir su metodo onDragListener o asignando una implementacion de evento Drag
con el metodo setOnDragListener()

Durante el proceso de arrastre el sistema va propagando eventos Drag con una serie de datos y accións asociados al evento.

ACTION_DRAG_STARTED: Esta acción se recibe en los componentes que estan a la escucha de eventos Drag y se genera cuando la aplicación hace la llamada a startDrag()
ACTION_DRAG_ENTERED: Este evento se recibe cuando el usuario ha entrado en la zona de soltar del componente en cuestion

ACTION_DRAG_EXITED: Es el contrario a ENTERED, se recibe al salir
ACTION_DRAG_LOATION: Se recibe despues del ENTERED y miestras la sombra esté dentro de la region del componente donde soltamos.
ACTION_DROP: Lo recibe el componente sobre el que se suelta la sombra, siempre que devolviese true al STARTED
ACTION_DRAG_ENDED: Se recibe cuando el proceso de arrastrar soltar ha terminado.

Diapo 22:
La sombra del arrastrar.

Durante el proceso de arrastrar y soltar, el sistema muestra una imagen bajo el dedo del usuario, esta imagen representa los datos que estamos moviendo.

La imagen se llama Drag Shadow. Para construir esta imagen disponemos de un metodo sobrecargado:
DragShadowBuilder(View), este metodo recibe el componente a mostrar durante el arrastre, durante la recepcion de los eventos podemos acceder a ella para
modificarla en función de la acción recibida.

DragShadowBuilder(), podemos realizar una implementacion libre de lo que queremos q se muestre durante el arrastre como sombra.

onProvideShadowMetrics(), Usamos este metodo para decidir las dimensiones de la imagen a mostrar y el punto donde vamos a dibujarlas. Dimension contiene la anchura X y la altura
de la sombra. touch_point sirve para indicar la posición donde dibujaremos la sombra.

onDrawShadow(), es el metodo encargado de dibujar la sombra

Para mejorar la fluidez con la que se arrastra la sombra es aconsejable usar tamaños pequesnños para esta.

(Mostrar código de la aplicación)
Iniciar el D&D ContactFragment 200-211
Preparar view para recibir Eventos D&D DetailsFragment: 73
Procesar eventos recibidos DetailsFragment: 169-202

Diapo 23:

Animacion basada en propiedades.

Comportamiento.
Estas animaciones estan basadas en el cambio de los valores de las propiedades de los componentes que queremos animar.
Este framework nos permite controlar las siguientes caracteristicas de una animacion:

Duracion, por defecto 300ms
Tipo de interpolación: como calcular los valores que irá tomando la propiedad
Numero de repeticiones y comportamiento.
Agrupar animaciones

Retardo de refresco de cada frame: Por defecto es de 10 ms pero depende la carga del sistema.

Funcionamiento:
ValueAnimator es la clase encargada de controlar el tiempo que lleva la animación, el valor actual de la propiedad que se está animando,
el valor de inicio y de fin para dicha propiedad. También encapsula el objeto TimeInterpolator que define la interpolación de la
animación y un TypeEvaluator que especifica como se calculan los valores de la propiedad que va a ser animada

Para iniciar una animación creamos un ValueAnimator o una de sus implementaciones y establecemos sus valores iniciales. La llamada al metodo
start() inicia la animacion, durante la animacion el objeto ValueAnimator hace uso de TimeInterpolator para calcular el tiempo que es válido

cada valor que se le asignará a la propiedad y TypeEvaluator para calcular el valor de la propiedad.

Diapo 24
clases principales.

Animadores:
ValueAnimator: Encargado de proporcionar la secuencia de frames que tendrán la animación y la duración de cada uno
ObjectAnimator: Implementación de ValueAnimator especifica para animar propiedades de un objeto
AnimatorSet: Mecanismo para sincronizar nuestras animaciones.

Evaluadores:
IntEvaluator Genera valores del tipo entero
floatEvaluator del tipo float
ArgbEvaluator colores basados en ARGB

TypeEvaluator Interfaz para implementar nuestro propio tipo de evaluador.

Interpoladores:
AccelerateDecelerateInterpolator: Decelerado al comienzo y al fina lde al animación.
AccelerateInterpolator: Se acelera hacia el final de la animacion
DecelerateInterpolator An interpolator whose rate of change starts out quickly and and then decelerates.
LinearInterpolator An interpolator whose rate of change is constant.
OvershootInterpolator An interpolator whose change flings forward and overshoots the last value then comes back.
TimeInterpolator An interface that allows you to implement your own interpolator.

Diapo 25:
Usando ValueAnimator.
Nos permite generar una secuencia de valores desde un valor definido hasta otro, durante un intervalo de tiempo especificado.

Usando ObjectAnimator.

ObjectAnimator es una clase hija de ValueAnimator y nos permite animar una propiedad de un objeto de forma mu sencilla. Los requisitos para poder utilizar esta animacion
sobre cualquier objeto son:

Tener los setter en formato camel, setXXX
Si solo proporcionamos un valor, este será asumido como valor de finalizacion de la animación, se usará getXXX para obtener el valor inicial.

Dependiendo del objeto que estemos llamando puede ser necesario llamar a su metodo invalidate()

Diapo 26:
Coreografía de animaciones.
Con la clase AnimationSet podemos enlazar una animación con otra o incluso ejecutar animaciones de forma simultanea, para elo usamos los metodos
.before(), .with(), .after()

Ejemplo:
Ejecuta bounceAnim.
Plays squashAnim1, squashAnim2, stretchAnim1, and stretchAnim2 at the same time.
Plays bounceBackAnim.
Plays fadeAnim.

Diapo 27:

Eventos de las animaciones.
Animator.AnimatorListener
onAnimationStart() - Called when the animation starts.
onAnimationEnd() - Called when the animation ends.
onAnimationRepeat() - Called when the animation repeats itself.
onAnimationCancel() - Called when the animation is canceled. A cancelled animation also calls onAnimationEnd(), regardless of how they were ended.
(Mostrar código de la aplicación)
Ejemplo de Animacion secuencial DetailsFragment 232-273

Diapo 28:

This SDK component contains static libraries providing access to newer APIs on older platforms. To use those libraries, simply copy them as static libraries into your project.

"v4" provides support for using new APIs on Android API 4 (1.6 - Donut) and above.

v4/android-support-v4.jar contains:
- Fragment API. New in API 11 (3.0 - Honeycomb).
http://developer.android.com/reference/android/app/Fragment.html
- Loader API. New in API 11 (3.0 - Honeycomb).
- MenuCompat allows calling MenuItem.setShowAsAction which only exists on API

Diapo 29
Usar el Compability pack solo es necesario importar el jar

Diapo 30:
Copiar el jar a nuestro proyecto

Diapo31:
Importar el jar al java buildpath

Diapo 32:
Despedida.