Existen varias maneras de obtener persistencia en los datos de la aplicación en distintas ejecuciones:

1. Mecanismo ligero: Preferencias compartidas / Shared preferences. Permiten guardar pequeñas cantidades de datos.
2. Sistema de archivos tradicional.
3. Base de datos relacional SQLite.

La mejor de las tres es la de preferencias compartidas; Android incorpora el objeto SharedPreferences para guardar y leer los datos persistentes en forma clave-valor. Esta clase permite almacenar cualquier tipo de dato primitivo. Estos datos persistirán incluso después de cerrar la aplicación, sobre todo en archivos XML.

……………………………………………………………………………………………………………

Para inicializar un objeto de esta clase se usa el método getSharedPreferences(“ArchivoPreferencia”, MODE\_PRIVATE);

\*En caso de usar un único archivo de preferencias se usa el mismo método sin parámetros.\*

Para escribir valores en el archivo de preferencias se llama al método edit(); de este se obtiene un objeto SharedPreferences.Editor al que se le añaden valores mediante métodos (putBoolean(); o putString();). Todos estos métodos usan en sus parámetros una clave-valor.

Para leer valores de preferencias se usan los métodos análogos getBoolean();, getString(); y getInt(); de la clase. Al igual que los métodos de escritura, estos tienen dos parámetros: El primero es el nombre de la clave del archivo de preferencias y el segundo es el valor predeterminado que se usa en caso de no encontrar la primera clave.

El sitio más adecuado para cargar y guardar las preferencias de la aplicación es en los métodos onCreate(); y onStop();

……………………………………………………………………………………………………………

Android proporciona un sistema de base de datos relacional basado en SQLite. Cuando la cantidad de datos es considerable debe estar estructurada en la base de datos para asegurar la integridad de los datos y especificar relaciones entre ellos. Cada base de datos que se cree únicamente pertenecerá a su respectiva aplicación. La forma de usar las bases de datos es mediante una clase que permite encapsular los accesos, simplificando código.

La creación de la base de datos debe encapsularse para que no sea visible ni accesible para el usuario. La base de datos debe crearse una sola vez, por lo que el código no debe ir en ningún método onCreate(); Esto podría resetear la base de datos en cada ejecución.

En la creación y manipulación de la base de datos se usan clases auxiliares como SQLiteOpenHelper. Esta permite montar la estructura de la DB además de su modificación y actualización.

Esta clase incluye métodos abstractos que se deben implementar:

1. onCreate(); : Se lanza al activar un objeto de la clase. Genera las tablas durante su primera ejecución o las actualiza si se lanza con la DB ya creada.
2. onUpgrade(); : Sirve para actualizar la DB. Dependiendo del contenido puede que sea necesario implementar mecanismos de copia de seguridad.

Las operaciones CRUD (Create, Read, Update y Delete) se pueden realizar con cualquier base de datos. Para lanzar estas instrucciones SQL es necesario usar la clase SQLiteDatabase que implementa todos los métodos para realizar las operaciones. Esta recoge los valores de cada campo en la tabla correspondiente haciendo uso del método que hayamos diseñado en la anterior clase auxiliar.

Para ***lanzar instrucciones SQL*** a la DB se usa la clase ***SQLiteDatabase,*** esta contiene los ***métodos para ello:***

La ***operación de lectura*** que una consulta simple se realiza mediante uno de estos métodos, a diferencia de la consulta múltiple, para la que hace falta un objeto de la clase Cursor además de una vista dinámica que almacene los resultados y permita su visualización.

La ***vista dinámica*** que se emplea es RecyclerView, este permite cargar otras vistas o grupos de estas de forma dinámica. Su programación es compleja, para ello se necesita:

1. Tener una clase que represente el objeto que se va a recuperar, con sus correspondientes atributos además de métodos getter y setter respectivos.
2. Tener un método en la clase DBNotas para recuperar los datos mediante una consulta que devuelva una colección de elementos.
3. Elaborar un adaptador que se asocie a un layout.xml y lo replique de forma dinámica en el RecyclerView por cada elemento.
4. Establecer desde el MainActivity la correspondencia entre un objeto del RecyclerView, la vista en el layout.xml y el adaptador que genera el listado.

La operación de actualización se producirá sobre una sola fila cada vez, para ello se necesita una actividad que permita visualizar los detalles de una fila en concreto y permitir su edición mediante un formulario:

1. Diseñar una actividad para visualizar una sola entrada y enlazar cada una con esta. Se hace en el ViewHolder de la clase Adapter.
2. Recuperar la información de la entrada que se va a visualizar. Se hace mediante un método de la clase DBNotas.
3. A través de botones permitir guardar los cambios realizados en el formulario.
4. Los datos del formulario deben incrustarse en una sentencia UPDATE, contenida en un método de la clase DBNotas.

La operación de borrado se plantea al final del proceso porque se beneficia del resto de código que ya se ha programado. Esta debe añadirse en la vista de un elemento concreto, para ello:

1. Se añade un botón en dicha vista que solicite la confirmación de borrado, esto puede hacerse mediante la clase AlertDialog.Builder.
2. Esta confirmación debe lanzar el método de la clase DBNotas que ejecute la sentencia DELETE para el elemento en concreto.
3. Si se ha ejecutado correctamente debe volver de nuevo a la lista dinámica.

……………………………………………………………………………………………………………

La conectividad depende de la localización y características de un dispositivo. Esto se debe tener en cuenta a la hora de desarrollar aplicaciones que requieren conectividad a la red.

Al iniciar una aplicación, Android pone en marcha un proceso con un solo hilo en ejecución. Normalmente todos los componentes de la aplicación se ejecutan sobre este hilo, pero a veces conviene ejecutar diferentes tareas de la aplicación en diferentes hilos.

Cuando una aplicación accede a Internet u otros recursos externos se produce un retraso de duración variable desde que la aplicación pide los datos hasta que los recibe.

Si la aplicación funciona con un único hilo, este puede quedar detenido en la llamada a un método, en ese caso la ejecución se verá afectada hasta la recepción de los datos. Si esto ocurre, lo más probable es que el usuario cierre la aplicación. Para evitarlo es necesario que la aplicación continúe respondiendo a la interacción del usuario además de mostrar algún tipo de animación como una barra de progreso.

Para conseguir un hilo compatible con Android en segundo plano se necesita crear una instancia de la clase Handler en cada actividad. Hay dos opciones para comunicar a un hilo con esta clase:

1. Mensajes: Para enviar un mensaje a un Handler se necesita el método obtainMessage() para obtenerlo mediante un objeto Message. Cuanto más complejo sea el procesamiento del Handler, más probable es que sea necesario poner datos en Message para ayudar al Handler a distinguir los eventos. Después habrá que enviar el Message al Handler a través de su cola de mensajes usando los métodos de la clase Message. Para procesar estos mensajes, el Handler necesita handleMessage() que será invocado con cada mensaje que haya en la cola.
2. Objetos Runnable: El Handler ejecuta los objetos Runnable en el hilo de cada actividad. Dado que Handler soporta los métodos post() y postDelayed() para añadir objetos Runnable a la cola, también se pueden usar en View.

Algunas veces no sabemos sobre qué hilo se está ejecutando la aplicación, para ello, las Activity ofrecen el método runOnUiThread() que trabaja como los métodos post() en Handler y View, haciendo que el objeto Runnable se ejecute en el hilo de la interfaz de usuario si no estamos ejecutando la app sobre él. En caso contrario invoca inmediatamente el Runnable.

Otro factor a tener en cuenta es la conectividad, que no solo depende de las características de la app sino también de las de red disponible y del dispositivo móvil. Por ello se deben proporcionar alternativas para las conexiones inestables, el SDK proporciona herramientas y clases para asegurar un buen funcionamiento en la app.

Para transferir datos hacia y desde la red se usa el protocolo HTTP. Este encapsula cualquier tipo de datos y los asegura mediante SSL / Capa de Conexión Segura.