PERSISTENCE DAT

David Bilík

ANDROID DEVELOPER @ ACKEE

#CODECAMPCZ

CO NÁS DNES ČEKÁ

- Ukládání dat pokud aplikace “běží”
- Soubory
- SharedPreferences
- SQLite

KRÁTKODOBÁ DATA - ULOŽENÍ V PAMĚTI

- Uložení dat do Application class
- Jsou smazána při restartování aplikace
- ! I pokud systém zabije aplikaci a vy ji znovu nastarutjete

KRÁTKODOBÁ DATA - AKTIVITY/FRAGMENTY

Viz projekt v minulé hodině

- Aktivity
  - onSaveInstanceState - přežije zabití systémem
  - configurationChanged - nepřežije
- Fragmenty
  - onSaveInstanceState - přežije
  - getArguments().putXXX - přežije
  - setRetainInstance(true) - nepřežije

RETAIN FRAGMENTY

- Fragment, který není zničen při znovuvytvoření aktivity - využije se stejná instance, takže je možné použít uložené členské proměnné

- často se využívají jako data holdery a nemají UI

- nevolají se na nich metody onSaveInstanceState, takže v onCreate(view) je Bundle vždy null

- příklad…

SOUBORY

- Data můžeme ukládat do souborů jako v klasické Java SE
- Aplikace má přístup k její interní paměti, kam nemají přístup jiné aplikace a k externí paměti, kam mají přístup všechny aplikace
- Pro přístup k externí paměti je třeba permission READ_EXTERNAL_STORAGE nebo WRITE_EXTERNAL_STORAGE

SOUBORY

- Context.getFilesDir() - vrací složku pro data v interní paměti
- Environment.getExternalStorageDirectory() - vrací složku pro data v externí paměti
- Environment.getExternalStoragePublicDirectory (String type) - vrací nativní složku pro určitý typ souborů
  - typ nabývá hodnot jako např. MUSIC, DCIM, …

SOUBORY

- příklad…

SHARED PREFERENCES

- Android komponenta, která slouží jako mapa klíč - hodnota
- Ukládá xml soubor s daty do interní paměti aplikace
- Hodí se pro nestrukturovaná jednoduchá data
- Nehodí se pro seznamy dat, složité objekty, vzájemné relace

SHARED PREFERENCES

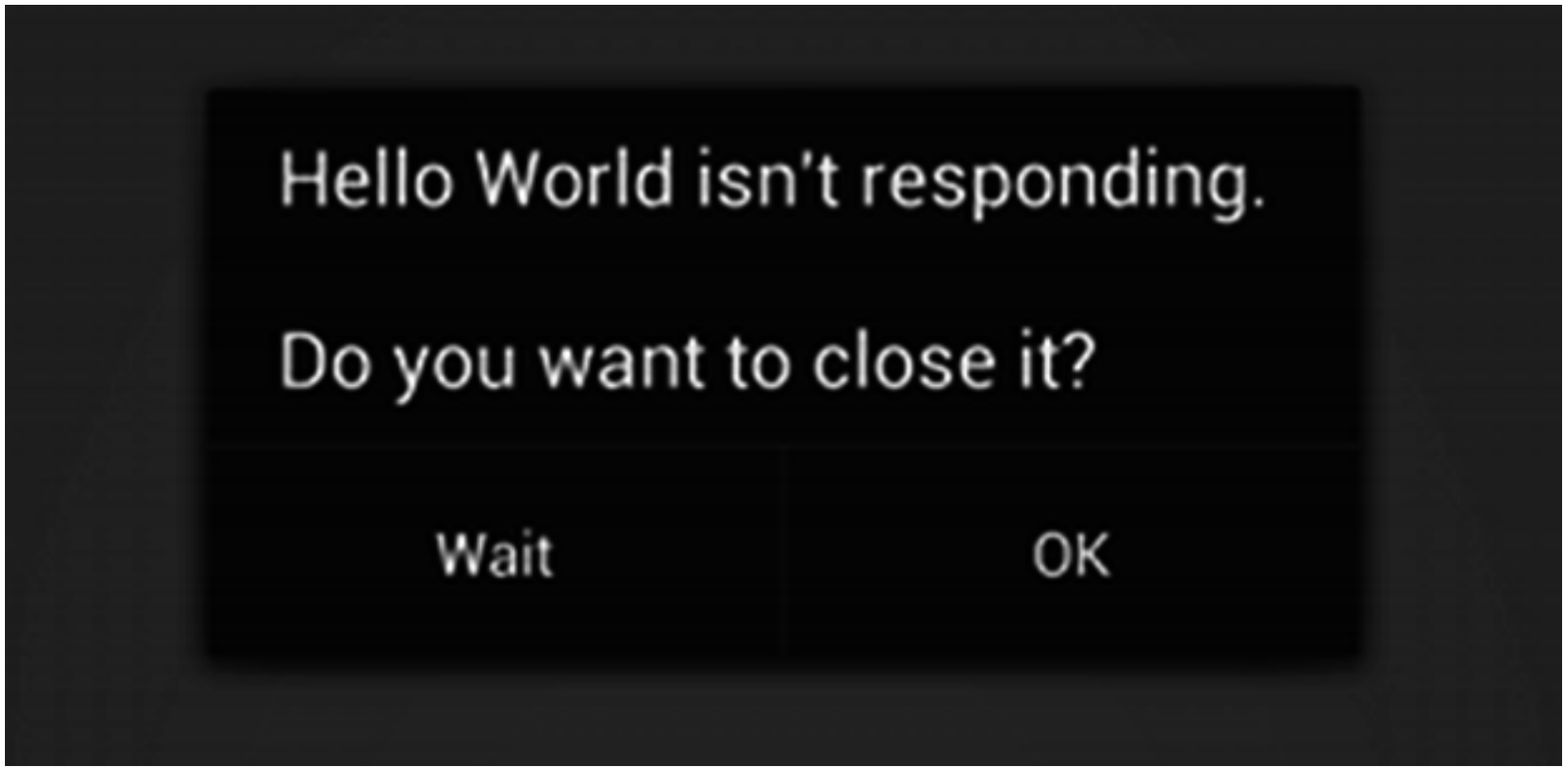
- Získání SharedPreferences - Context.getSharedPreferences(name, mode)
  - name je název souboru, do kterého jsou data ukládána
  - mode udává v jaké viditelnosti je soubor vytvořen - privátní (pouze pro aplikaci) a nebo může být čten/ zapisován jinými aplikacemi
- Zápis hodnot
  - je třeba získat SharedPreferences.Editor metodou edit()
  - vkládat data metodami .putXXX(key, value)
  - na editoru zavolat commit() (uloží synchronně) nebo apply() (uloží asynchronně)

SHARED PREFERENCES

- Získání hodnot
  - getXXX(key, defaultValue)
- Další možností jak získat preferences je v Aktivitě zavolat getPreferences(mode)
- Vrátí prefs pro danou aktivitu - nejsou sdíleny naskrz aplikací
- příklad…

THREADING

- UI aplikace je vykreslováno na tzv. Main/UI vlákně a nemůže být vykreslováno z žádného jiného vlákna
- Veškeré časově náročné operace by měly být prováděny na některém vedlejším vlákně
- Pokud je UI vlákno zastaveno na více jak 5 vteřin (může se lišit v různých verzích Androidu), je vyhozen Application Not Responding (ANR) dialog



THREADING

- Je třeba tedy náročné operace (networking, zápis/čtení souboru, přístupy k databázi, složitý výpočet,…) delegovat na jiné vlákno. Jak?

THREADING

- Thread
  - Klasická komponenta jako v Javě SE. Složité provázání s UI vláknem.
- AsyncTask
  - Androidí komponenta, která zjednodušuje provádění kodu na hlavním vlákně. Programátor musí sám řešit připojování k aktivitě/fragmentu aby předešel memory leakům.
- Loader
  - Vytuněný AsyncTask, který se sám stará o připojování a odpojování od aktivity/fragmentu. Programátor však i tak musí napsat spoustu boilerplate kódu. Využívá se nejčastěji jeho podtřída CursorLoader pro výsledky query do databáze

ASYNCTASK

- *doInBackground(Params …)* - jediná metoda, kterou je třeba přepsat. Běží na vedlejším vlákně
- *onPreExecute/onPostExecute(Result)* - metody, které jsou zavolány před/po "výpočtu" a jsou spuštěny na hlavním vlákně
- *onProgressUpdate(Progress)* - spuštěno na hlavním vlákně, pokud se v doInBackground zavolá *publishProgress()*
- *AsyncTask<Params, Progress,Result>*
  - Params - datový typ vstupních parametrů
  - Progress - datový typ progressu
  - Result - datový typ návratové hodnoty z vedlejšího vlákna

DATABÁZE

- Databáze se využívá pro ukládání strukturovaných dat, které mezi sebou mohou mít vazby.
- Na Androidu je built-in databáze SQLite
- Velice rychlá a kompaktní databáze, která je využívána i na iOS, Chrome, …
- Datové typy - NULL, INTEGER, REAL, TEXT, BLOB
- Má dynamické typování – datový typ je určen hodnotou a ne kontejnerem

VYTVÁŘENÍ DATABÁZE

- Pro vytváření databáze se nejčastěji využívá *SQLiteOpenHelper(Context, databaseName, CursorFactory, databaseVersion)*
- *onCreate(SQLiteDatabase)* - zavoláno, pokud databázový soubor neexistuje. Využívá se pro inicializaci tabulek
- *onUpgrade*(*SQLiteDatabase, oldVersion, newVersion*) - zavoláno při změně verze databáze
- Měla by existovat jedna instance *OpenHelperu* v aplikaci. Při více přístupech do databáze může nastat chyba, že se snaží aplikace přistupovat k databázi, která je zamčena někým jiným (jinou částí naší aplikace)

VYTVOŘENÍ DATABÁZE

- Příklad

ZÍSKÁNÍ DAT

- OpenHelper má metody *getWritableDatabase()* a *getReadableDatabase()* pro získání databáze, která má povolen jen zápis/čtení. Vrací SQLiteDatabase objekt
- pro SELECT nad tabulkou se používá metoda query na SQLiteDatabase objektu
- *db.query(tableName, projection, selection, selectionArgs, groupBy, having, orderBy)*
  - SELECT *projection* FROM *tableName* WHERE *selection* GROUP BY *groupBy* HAVING *having* ORDER BY *orderBy*
  - *selectionArgs* slouží pro nahrazení parametrů v *selection* stringu. Používá se kvuli SQL injection
  - Pokud nějaká část dotazu není potřeba, pošle se null

ZÍSKÁNÍ DAT

- *query()* vrací Cursor objekt, který obsahuje načtená data z tabulky. Přes tento cursor se pak získávají jednotlivé sloupce v řádku
- *cursor.getInt(columnIndex)*
  - vrací integer sloupce, který je na pozici *columnIndex* z projekce
  - pro získání tohoto indexu je na cursoru metoda *cursor.getColumnIndex(columnName)*
- Cursor má metody *.moveToNext(), moveToPrevious(), moveToFirst()*…pro pohyb mezi řádky výsledku. Pro získání všech záznamů z cursoru se přes něj musí proiterovat
- Cursor má neustále spojení do databáze a je třeba ho zavírat metodou *close()*
- příklad…

ZÍSKÁNÍ DAT

- *query()* metoda má mnoho různých variant s možnostma nad rámec těchto základních
- pokud je však třeba napsat nějaký složitější SELECT, např. s JOIN atd, existuje metoda *db.rawQuery(String sql, String [] args)*, která přijímá String s SELECT dotazem

ZÁPIS DAT

- *db.insert(String tableName, String nullColumnHack ContentValues)*
  - *tableName* - jméno tabulky do které se vkládá
  - *nullColumnHack* - jméno sloupce, do kterého se má vložit nějaká hodnota pokud jsou ContentValues empty. často se používá “null” a ignoruje se
  - *ContentValues* - mapa klíč-hodnota, do které se jako klíče vkládají názvy sloupců
  - vrací id nově vytvořeného záznamu nebo -1 při chybě

UPDATE DAT

- db.update(String tableName, ContentValues, selection, selectionArgs)
  - tableName - jméno tabulky
  - ContentValues - mapa klíč-hodnota, do které se jako klíče vkládají názvy sloupců. Obsahuje jen změněné sloupce
  - selection - podmínka pro updatované řádky
  - vrací počet upravených záznamů

SMAZÁNÍ DAT

- db.delete(String tableName, selection, selectionArgs)
  - tableName - jméno tabulky
  - selection - podmínka pro mazané řádky
  - vrací počet smazaných záznamů

ZÁVĚREM

- Příklad
- Domácí úkol
- Dotazy

DĚKUJI ZA POZORNOST!

OTÁZKY?