Simple Call Blocker

Egyszerű hívásblokkoló Androidos telefonokra

Tartalomjegyzék

Frontend

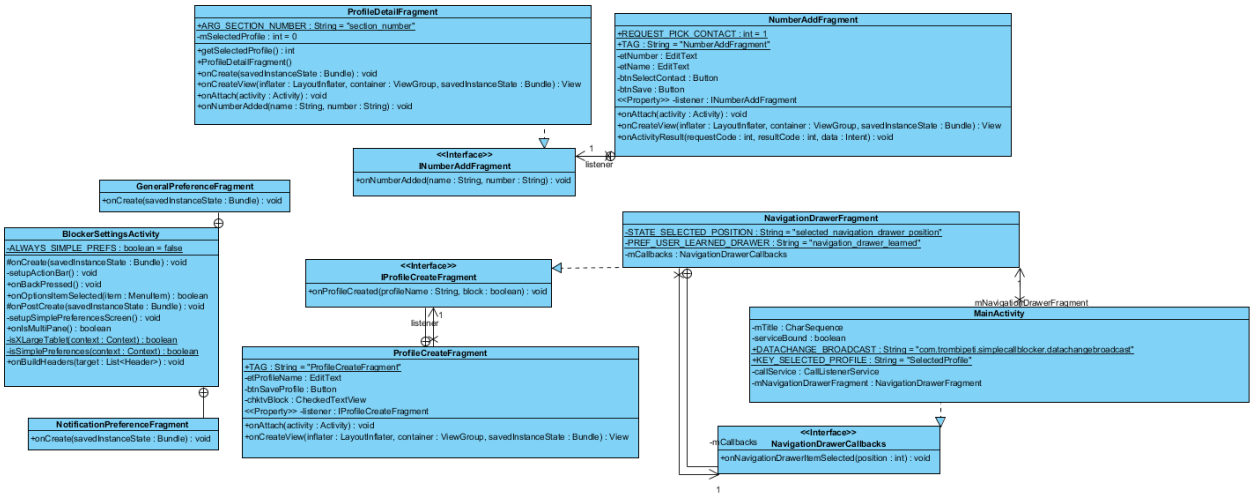
Az alkalmazás alapja az Android Studio által generált Navigation Drawer típusú alkalmazás. A fő Activity (`MainActivity`) egy Navigation Drawer Fragment komponenst használ a navigálásra. Ezt a MainActivityben lévő fragment (`ProfileDetailFragment`) cserélgetésével éri el. Szintén innen érhető el az alkalmazás Beállítások (`BlockerSettingsActivity`) ablaka is, amely egy külön Activity.

A profilok és a tiltandó szám hozzáadását az ActionBarban lévő gombbal, illetve az "Add number" feliratú gommbal (ez a `ProfileDetailFragment`ben van) indíthatjuk el. Az `OnClick` eseményre megjelenik egy-egy `DialogFragment` komponens (`ProfileCreateFragment`, illetve `NumberAddFragment`). Ezeknek definiálnak egy-egy interfészt (`IProfileCreateFragment`, `INumberAddFragment`), és ezt a típust megvalósító objektum fel tud iratkozni a változásokra. Megvalósítás szintjén a "Profil létrehozás" eseményre (`IProfileCreateFragment.onProfileCreated(String profileName, boolean block)`) a `NavigationDrawerFragment` van feliratkozva, és az esemény létrehozására eltárol egy új profilt (`BlockProfile`) az adapterében (`NavigationAdapter`). A "Szám hozzáadás" eseményre (`INumberAddFragment.onNumberAdded(String name, String number)`) pedig a profilt megjelenítő fragment, a `ProfileDetailFragment` van feliratkozva. Ezen fragment layoutjában van egy `ListView`, amelynek az adaptere egy `ContactAdapter.` Ehhez fogja hozzáadni az újonnan létrehozott `Contact` objektumot.

A profilok törlésére az `ActionBarban` található gomb szolgál. Ennek a kezelése a NavigationDrawerFragment dolga, ugyanis ennek az adaptere kezeli a profilokat. Ebből következően a gomb `OnClick` eseményére felugró AlertDialog eseménykezelője a `NavigationDrawerFragment` adapterének szól, hogy törölje az aktuális profilt. Törlés után pedig az eggyel feljebb lévő fragmentet jeleníti meg.

A profilon belül a blokkolandó számok törlését a jobb oldalukon lévő gombbal valósíthatjuk meg. Ez a ListView elemeinek layoutját leíró .xml fájlban definiáltam (`contactview.xml`). Ez megnyomásra szintén feldob egy `AlertDialog`ot, és pozitív válasz esetén a `ContactAdapter`ben példányosított eseménykezelő kitörli a tiltandó számot a listából.

Osztálydiagram



1. Ábra: A frontend osztálydiagramja

Backend

Service

A hívások blokkolását egy háttérben futó service, a `CallListenerService` végzi. Ez folyamatosan fut, akkor is, amikor az alkalmazás grafikus felülete nem. Ez regisztrál indításkor egy belső osztályban megvalósított, PhoneStateListener osztályból származtatott objektumot a TELEPHONY_SERVICE-en, LISTEN_CALL_STATE paraméterrel, vagyis a hívás állapotainak változásáról fog értesülni. Az esemény bekövetkezésekor megnézi, hogy kell-e tiltani az aktuális számot, és ha igen, a következő bekezdésben részletezett módon megteszi azt.

Az GUI indításakor a `MainActivity` bindol a service-hez, és a beállítások ablakban történő változtatások hatására elindítja/leállítja azt.

A profilok/beállítások változásakor az adott komponens egy Broadcast üzenettel tudatja a service-el, hogy változás történt. Ezt egy `BroadcastReceiver`, a `DataUpdateReceiver` kezeli, és frissíti az adatokat.

A hívások tiltása

Mivel a hivatalos Android API nem nyújt módot a folyamatban lévő hívások lerakására, kis trükkhöz kellett folyamodnom. Ehhez a Java nyelv Reflection nevű elemét használtam, amely pongyolán fogalmazva az egyes osztályok és azok adattagjainak, függvények név szerinti elérését, illetve módosítását teszi lehetővé. Mivel az Android nyílt forrású, ennek segítségével a nem publikált, privát API is elérhetővé válik.

Nem volt tehát más dolgom, mint az `ITelephony` interfész kódját az internetről letöltve, a com.android.internal.telephony csomagba bemásoltam, majd a `TelephonyReflection` osztályban

név szerint elértem az `endCall()` függvényt. Mivel ez alapból privát, egy egyszerű `setAccessible(true)` hívással elérhetővé kellett tennem. Ezután már hívható volt, és valójában működik is.

Model

A profilok tárolását egy `Singleton pattern` tervezési mintával megvalósított, `BlockProfilesSingleton` nevű osztályban oldottam meg. Ez tartalmaz egy tömböt a tárolt `BlockProfile` objektumokból, valamint külön az alapértelmezett profil objektumát is.

A `BlockProfile` objektum tárolja a tiltandó számokat (`Contact` objektumok), valamint a beállításait is. Ezen kívűl biztosít egy egyszerű interfészt a Contactok eléréséhez is.

A `Contact` objektum nem túl nagy bonyolultságú, tartalmazza nevet és a számot, valamint megvalósítja a `Comparable` interfészt is, hogy a belőle készített tárolókon végig lehessen iterálni.

Adapter

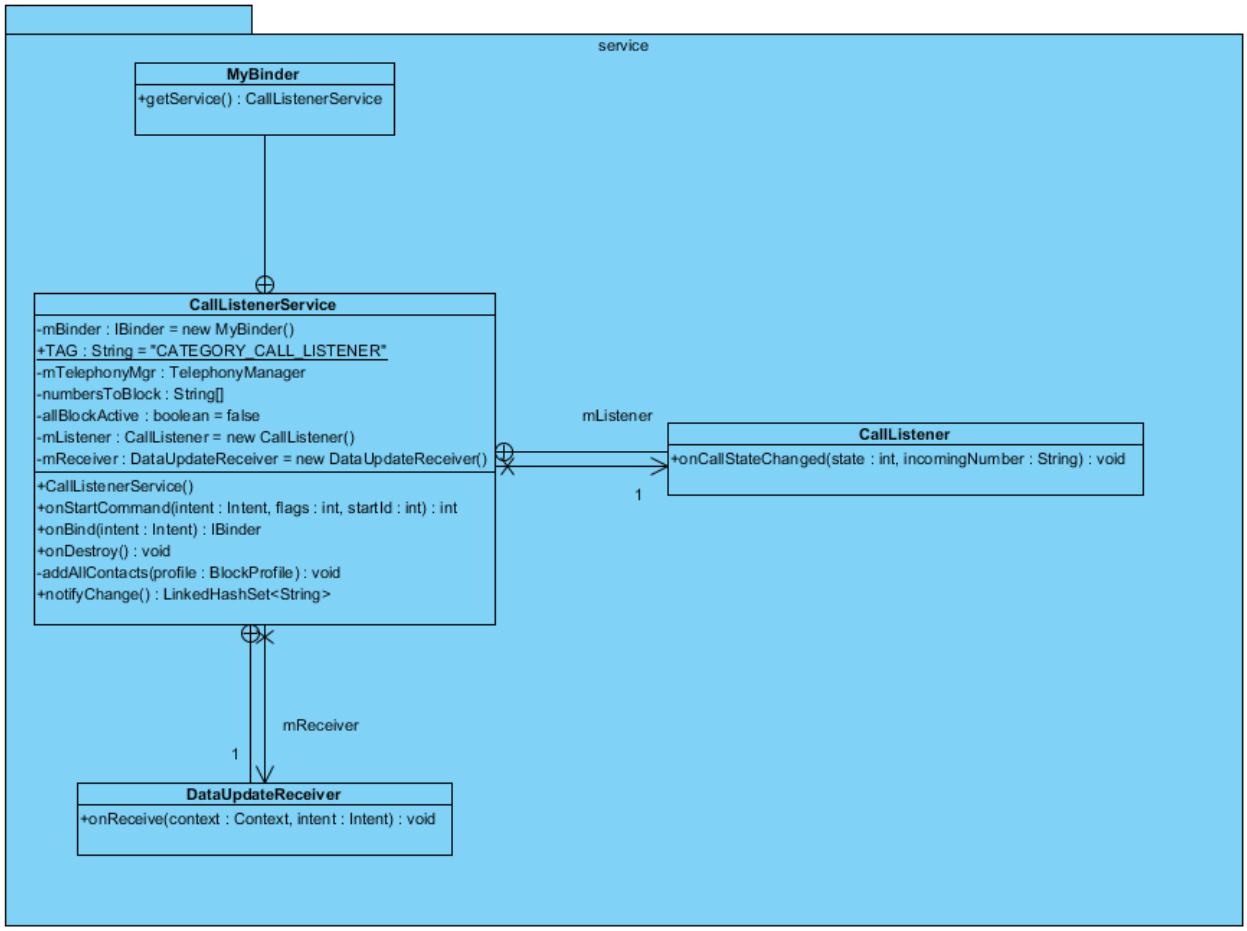
A modelben lévő adatok megjelenítését az `adapter` csomagban lévő osztályok szolgáltatják (`ContactAdapter` és `NavigationAdapter`). Ezek a BaseAdapterből származnak.

A `ContactAdapter` a konstruktorában meg kell kapja az aktív profil indexét. Egyébként a `BlockProfilesSingleton` példányából kéri le a megjelenítendő adatokat.

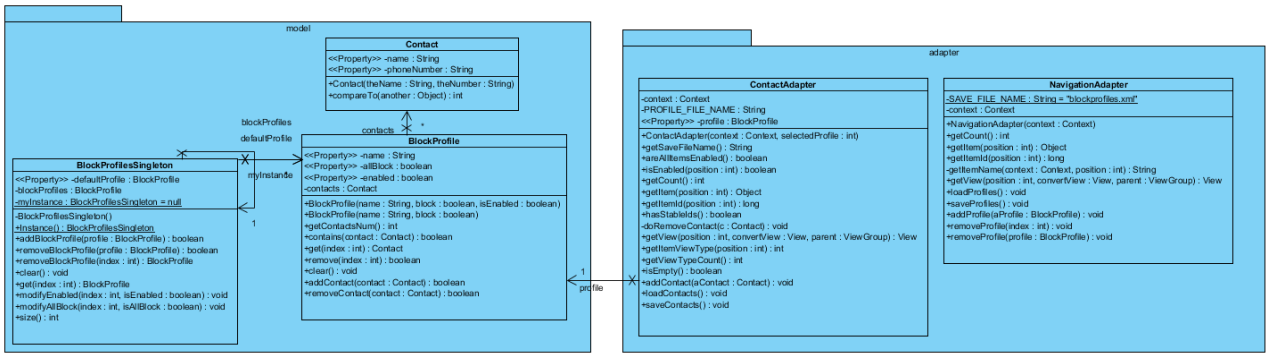
A `NavigationAdapter` hasonlóan működik, a `BlockProfilesSingleton` példányához ad hozzá/töröl profilokat, valamint kéri le tőle a profilok nevét, beállításait.

Ahhoz, hogy a profilok perzisztensen megmaradjanak, mindkét adapter lehetőséget ad a fájlba való mentésre, illetve az onnan való visszaolvasásra. Az adatokat XML formátumban tárolják, a szintaktika kezelésére pedig a Java beépített `XmlPullParser` osztályát használják. Az adatok módosításakor automatikus mentés történik, létrehozáskor pedig beolvassák a fájlok tartalmát.

Osztálydiagramok



2. Ábra: A service package



3. Ábra: A model és az adapter package-ek