TEHNICI DE PROGRAMARE

DOCUMENTATIE: TEMA 6

-MORNING ALARM-

Stundent: Profir Alexandr

Grupa: 30227

Cuprins

1. Obiectivul temei

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

4. Implementare si testare

5. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

1. Obiectivul temei

*Enunt: De proiectat si realizat o alarma destinata folosirii sub sistemul de operare Android care va permite operatii ca setarea si resetarea alarmei, alegerea zilelor in care va suna, alegerea orei de semnal de alarma si alegerea modului de resetare a ei.*

Aceasta tema are ca obiectiv crearea unei alarme din cite se observa alarma trebuie sa includa functii simple si sa furnizeze o interata comoda pentru utilizator. Imi propun sa realizez o aplicatie care va furniza o interfata grafica de introducere a datelor pentru setarea alarmei, precum va furniza si interfete grafice pentru stingerea ei, pentru inchiderea , plus la toate voi folosi tehnici pentru a da informatii utilizatorului despre starea curenta a alarmelor in genere.

Imi propun sa realizez aceasta aplicatie folosind instrumentele furnizate de interfata de dezvoltare si sa creez o interfata grafica a aplicatiei placuta si simpla, iar controlul si monitorizarea aplicatiei va fi realizat la fel intrun mod simplu.

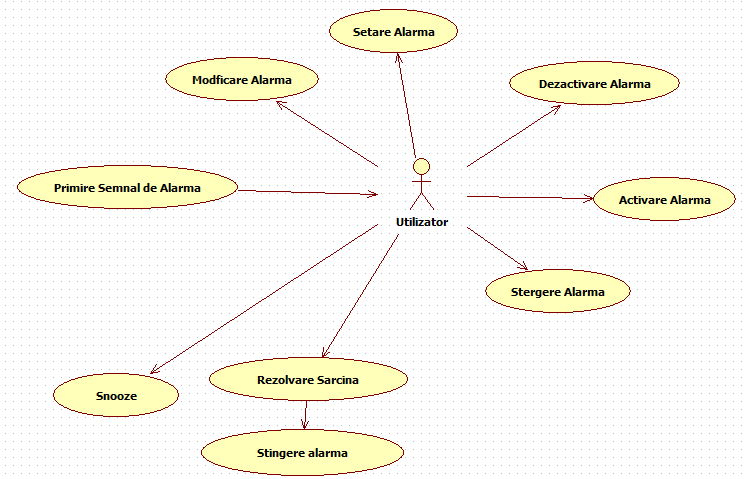
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Analiza problemei o vom face in urmatorii pasi:

1. Avem nevoie de o interfata grafica pentru a putea vedea care alarme sint deja setate si care sunt active sau inactive.
2. Avem nevoie de o interata grafica pentru afisarea ecranului in care se vor seta informatiile despre alarma adica ora la care ea va suna, melodia, modul in care va fi dezativata , in ce zile a saptaminii este activa.
3. Datorita specificului aplicatiei va trebui sa ne facem griji si asupra aspectelor ca activarea telefonului din regimul cu ecran inactiv, activarea ecranului in caz daca este stins si afisarea pe ecran a conditiei de dezactivare a alarmei chiar si in cazul in care telefonul este securizat cu lacat.
4. Deoarece va trebui sa stocam date referitor la mai multe obiecte vom lua in considerare folosirea unei baze de date pentru stocarea informatiilor, despre starea alarmelor si setarile ce lea ales utilizatorul anterior.
5. Va fi nevoie de analizat cazuri particulare de functionare ca restabilirea alarmelor dupa ce telefonul a fost stins, executarea soneriei a unei singure alarme pentru a nu irita utilizatorul, alarma nu trebuie sa impiedice utilizarea de zi cu zi a telefonului si de aceea trebuie privite cu mare grija situatiile de functionare a ei.

Pentru a deduce modelul aplicatiei date avem nevoie de analiza a scenariilor si cazurilor de utilizare.

Cazuri de utilizare :

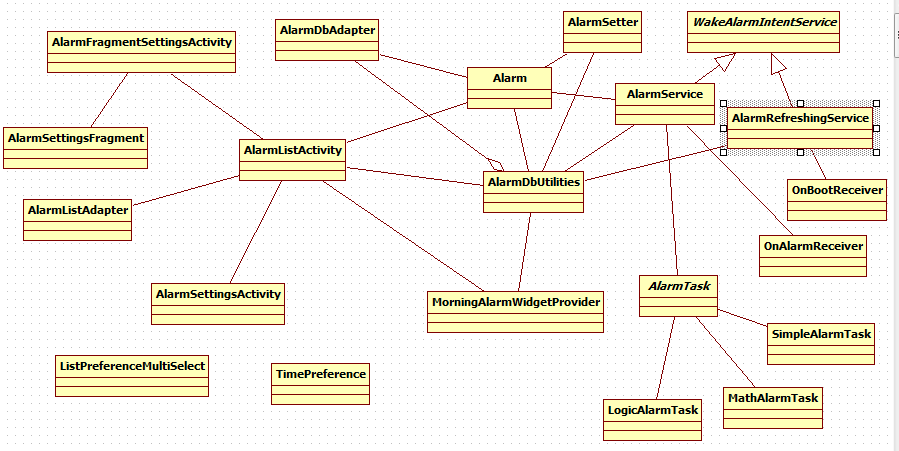


Dupa cum se poate observa din imaginea anterioara pentru a furniza o interfata grafica care corespunde cerintelor ea trebuie sa ia in considerare toate cele 9 cazuri de utilizare. Analizind cazurile de utilizare putem deduce si scenariile dupa care va fi folosita aplicatia.

Modelul aplicatiei este urmatorul: o aplicatie care va permite utilizatorului sa seteze alarme noi , precum si sa modifice, sa stearga sa activeze si sa dezactiveze alarme existente, pe de alta parte aplicatia trebuie sa instiinteze utilizatorul despre faptul ca alarma a ajuns la timpul specificat de acesta si trebuie sa ia decizia daca sa dea comanda Snooze pentru a face o pauza de 5 minute pina la urmatorul semnal sau sa rezolve sarcina pentru a dezactiva alarma. Alarma trebuie sa stocheze toate datele in baza de date, si sa se poata restabili dupa cazuri neobisnuite ca resetarea telefonului sau setarea a mai multor alarme pentru aceeasi ora.

1. Proiectare

a) Diagrama UML



b) Proiectare Clase

Dupa cum se observa din Diagrama UML am proiectat aplicatia folosind 21 de clase, rolul claselor este urmatorul:

AlarmRefreshingService : aceasta clasa este serviciul care este apelat de catre OnBootReceiver atunci cind aplicatiei i se trimite semnalul BootCompleted care este rezultatul restartarii telefonului. Rolul acestei clase este de a seta din nou toate alarmele din deoarece datorita proiectarii sistemului de operare acesta dupa ce face un restart complet a dispozitivului sterge toate alarmele setate anterior, respectiv, pentru ca aplicatia sa fie permanent functionala este necesar restabilirea tuturor alarmelor.

AlarmService : aceasta clasa este serviciul care este apelat de catre OnAlarmReceiver atunci cind aplicatiei i se transmite semnalul de alarma de catre clasa de sistem AlarmManager. Acest serviciu are rolul de a analiza ce fel de alarma este acum si care sunt proprietatile ei si trebuie sa actioneze corespunzator cazului.

AlarmSetter : aceasta clasa este responsabila pentru setarea si resetarea alarmei precum si de activarea sau deactivarea widgetului de pe ecran.

OnAlarmReceiver : este clasa care raspunde de capturarea semnalului de alarma si da startul service-ului AlarmService.

OnBootReceiver : este clasa care raspunde de capturarea semnalului de BOOT\_COMPLETED de la sistemul de operare care semnalizeaza de restartarea dispozitivului si apoi aceasta clasa da startul service-ului AlarmRefreshingService.

WakeAlarmIntentService : clasa abstracta care defineste modul in care serviciile AlarmService si AlarmRefreshingService pastreaza control asupra unui lacat care nu permite telefonului sa se stinga in timp ce se executa.

AlarmDbAdapter : clasa adaptor pentru baza de date, datorita clasei interne DatabaseHelper care extinde clasa SQLiteOpenHelper face adaptarea dintre interfata bazei de date si restul aplicatiei.

AlarmDbUtilities : aceasta clasa furnizeaza un set de functii pentru extragerea, adaugarea, stergerea, actualizarea, interogarea informatiilor din baza de date a aplicatiei.

Alarm : aceasta clasa simbolizeaza insasi obiectul alarmei care pastreaza in cimpurile sale informatii necesare pentru toate operatiile ce sunt executate asupra alarmelor.

AlarmFragmentSettingsActivity : aceasta clasa foloseste clasa AlarmSettingsFragment pentru a prezenta preferintele pentru alarma in framework-ul sistemului de operare cu o versiune mai inalta sau egala cu 3.

AlarmListActivity : este meniul principal in care se pot adauga, sterge, modifica, activa si dezactiva alarmele.

AlarmListAdaper : este clasa Adapter dintre clasa ListView si clasa Alarm pentru ca in activitatea AlarmListViewActivity sa se afiseze continut corespunzator alarmei.

AlarmSettingsActivity : este clasa ce afiseaza preferinte pentru versiuni de sistem de operare Android mai mici sau egale cu 3

MorningAlarmWidgetProvider : este clasa ce controleaza continutul widgetului de pe ecranul principal si permite interactiunea cu aplicatia prin intermediul lui.

AlarmSettingsFragment : clasa ce extinde Fragment si se foloseste la afisarea preferintelor de catre clasa AlarmFragmentsSettingsActivity pentru afisearea preferintelor in framework-ul sistemului de operare cu versiunea mai mare sau egala cu 3

ListPreferenceMultiSelect : se foloseste la setarea preferintelor pentru zilele in care sa functioneze alarma.

TimePreference : aceasta clasa se foloseste la afisarea de TimePicker in preferinte pentru permiterea de selectare a timpului la care va suna alarma.

AlarmTask : este o clasa abstracta care defineste comportamentul unei activitati task, adica a unei activitati ce apare pe ecran ca problema pe care trebuie sa o rezolve utilizatorul

LogicAlarmTask : este o clasa care defineste taskul logic pentru inchiderea alarmei.

MathAlarmTask : este o clasa care defineste taskul matematic pentru inchiderea alarmei.

SimpleAlarmTask : este o clasa care defineste cel mai simplu task care contine doar doua butoane snooze si cancel.

c) Design Patternuri

Din diagrama de clase se observa ca am folosit citeva Design – Patternuri:

1. Factory : la clasa WakeAlarmIntentService si clasa AlarmTask
2. Singletone : la clasa AlarmDbAdapter
3. Adapter : la clasa AlarmListAdapter, AlarmDbAdapter

Un design pattern este o solutie generala si reutilizabila a unei probleme comune in design-ul software. Un design pattern nu este un design in forma finala, ceea ce inseamna ca nu poate fi transformat direct in cod. Acesta este o descriere a solutiei sau un template ce poate fi aplicat pentru rezolvarea problemei. In general pattern-urile orientate obiect arata relatiile si interactiunile dintre clase sau obiecte, fara a specifica insa forma finala a claselor sau obiectelor implicate.

Design Pattern-urile fac parte din domeniul modulelor si interconexiunilor. La un nivel mai inalt se gasesc pattern-urile arhitecturale (Architectural Patterns) ce descriu pattern-ul global utilizat al intregului sistem.

Nu toate pattern-urile software sunt design patterns. De exemplu, algoritmii rezolva probleme computationale, nu probleme de design.

Singleton Pattern

Pattern-ul Singleton este utilizat pentru a restrictiona numarul de instantieri ale unei clase la un singur obiect.

La baza pattern-ului Singleton sta o metoda ce permite crearea unei noi instante a clasei daca aceasta nu exista deja. Daca instanta exista deja, atunci intoarce o referinta catre acel obiect. Pentru a asigura o singura instantiere a clasei, constructorul trebuie facut protected (un constructor privat impiedica reutilizarea sa sau accesul unei unitati de testare).

Diferenta dintre o clasa cu atribute si metode statice si un Singleton este aceea ca Singleton-ul permite instantierea lazy, utilizand memoria doar in momentul in care acest lucru este necesar deoarece instanta se creeaza atunci cand se apeleaza getInstance(). Inca un avantaj ar fi faptul ca o clasa Singleton poate fi extinsa si metodele ei suprascrise, insa intr-o clasa cu metode statice acestea nu pot fi suprascrise (overriden).

In aplicatia mea am implementat metoda statica getInstance() a clasei AlarmAdapter care va returna unica referinta catre baza de date atit timp cit nu se schimba referinta catre Context.

Patternul Factory

Patternul Factory face parte din categoria Creational Patterns si ca atare rezolva problema crearii unui obiect fara a specifica exact clasa obiectului ce urmeaza a fi creat. Acest lucru este implementat prin definirea unei metode al carei scop este crearea obiectelor. Metoda va avea specificat ca parametru de intors in antet un obiect de tip parinte, urmand ca, in functie de alegerea programatorului, aceasta sa creeze si sa intoarca obiecte noi de tip copil.

Situatia cea mai intalnita in care se potriveste acest pattern este aceea cand trebuie instantiate multe clase care implementeaza o anumita interfata sau extind o alta clasa (eventual abstracta). Clasa care foloseste aceste subclase nu trebuie sa "stie" tipul lor concret ci doar pe al parintelui.

In aplicatia mea clasele WakeIntentAlarmService si AlarmTask sunt extinse de clase copii care impart functionalitatea clasei parinte cu mici diferente acolo unde metodele sunt suprascrise.

Patternul Adapter

Patternul Adapter este un design pattern care traduce o interfață pentru o clasă într-o interfață compatibila. Un adaptor permite unor claselor să lucreze împreună care în mod normal nu ar putea lucra impreuna din cauza interfeței incompatibile. Adaptorul traduce apelurile interfeței sale în apelurile către interfața originală, și cantitatea de cod necesara pentru a face acest lucru este de obicei mic. Adaptorul este, de asemenea, responsabil pentru transformarea datelor în forma corespunzătoare. De exemplu, în cazul în care mai multe valori booleene sunt stocate ca un singur întreg (de exemplu flags), dar clientul cere valorile individuale booleene, adaptorul ar fi responsabil pentru extragerea valorile corespunzătoare de la valoarea întreagă. Un alt exemplu este transformarea în formatul de date (de exemplu AAAALLZZ la LL / ZZ / AAAA sau ZZ / LL / AAAA).

In aplicatia mea am folosit acest pattern pentru a adapta aplicatia mea la interfata bazei de date si la interfata clasei ListView.

d) Pachete

In aceasta aplicatie am folosit patru pachete:

1. app.alarmmanager: AlarmRefreshingService, AlarmService, AlarmSetter, OnAlarmReceiver, OnBoot Receiver, WakeAlarmIntentService
2. app.database: AlarmDbAdapter, AlarmDbUtilities
3. app.morningalarm : Alarm, AlarmFragmentsSettingsActivity , AlarmListActivity,AlarmSettingsActivity, MorningAlarmWidgetProvider
4. app.morningalarm.preferences : AlarmSettingsFragment, ListPreferenceMultiSelect, TimePreference
5. app.tasks : AlarmTask, LogicAlarmTask, MathAlarmTask, SimpleAlarmTask.

d) Structuri de Date

Structurile de date folosite pentru aplicatia data au fost ArrayList-urile standarte din Java ce s-au folosit pentru pastrarea datelor despre Alarme

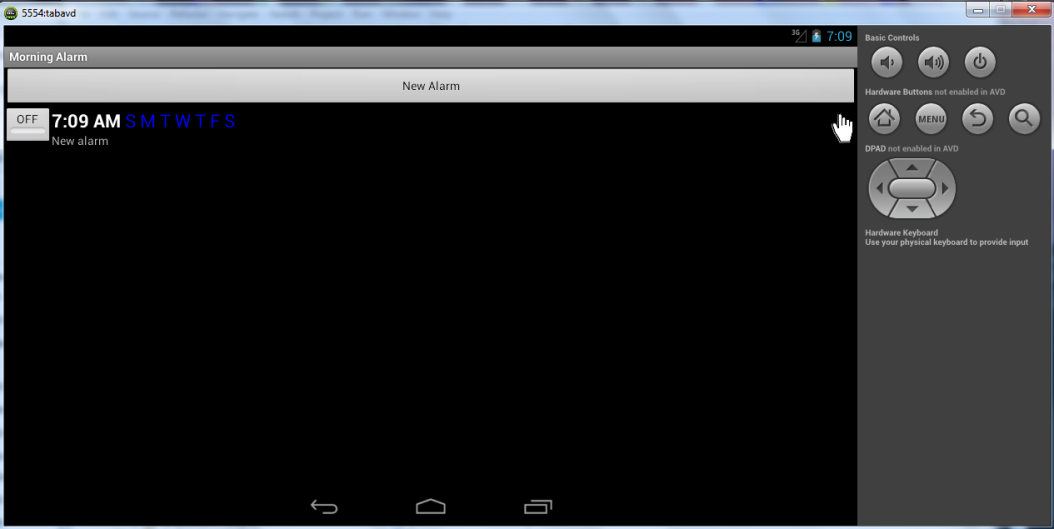
f) Interfete

In aceasta tema am creat interfete noi, cele folosite fiind cele propuse de sistem.

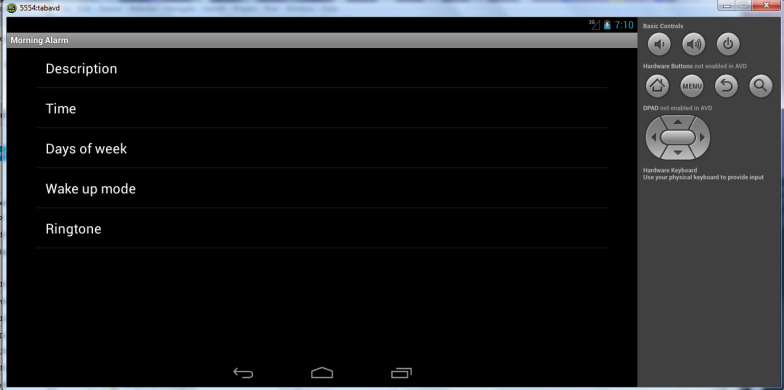
g) Interfata utilizator

Interfata utilizator a fost realizato in felul urmator:

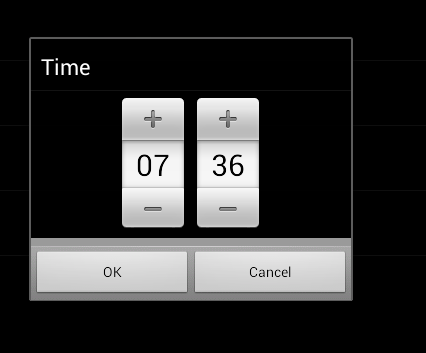
pe aceasta imagine se observa fereastra principala a aplicatiei



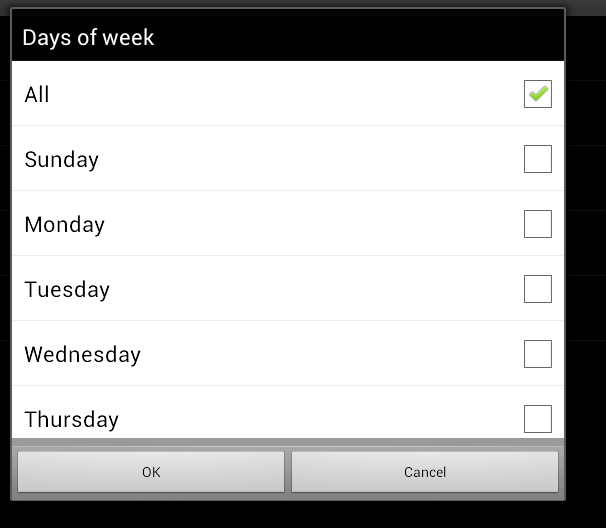
daca tastam alegem un element din lista de alarme atunci ni se deschide fereastra urmatoare in care apar preferintele aplicatiei in care putem seta preferinte noi sau sa modificam pe cele existente ca ele sa convina intereselor celui care le foloseste, preferintele au cinci elemente si fiecare din ele are o anumita sarcina:



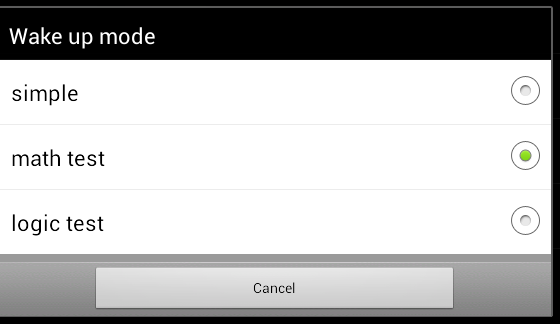
1. Description : seteaza descrierea alarmei, aceasta preferinta foloseste ca un un mic „tip” pentru utilizator ca acesta si peste mult timp dupa ce a setat alarma sa isi dee seama de ce a facut asta .
2. Time : este preferinta cea mai importanta deoarece anume prin ea se stabileste la ce ora anume va suna alarma, preferinta are o fereastra de dialog care contine un timepicker prin care utilizatorul poate seta ora si minuta in care va suna alarma .



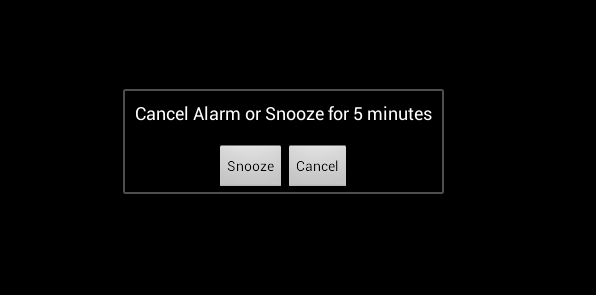
1. Days of Week : aceasta preferinta apare ca o fereastra de dialog care contine zilele saptaminii precum si optiunea all. Aceasta preferinta permite alegerea si setarea zilelor saptaminii in care va suna alarma facind astfel alarma inactiva in zilele neselectate de utilizatorul aplicatiei.



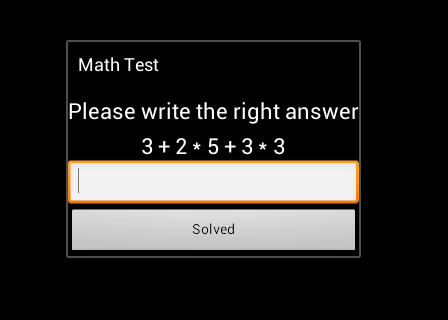
1. Wake up mode : este una dintre cele mai interesante preferinte ale acestei aplicatii care la prima vedere apare simpla dar poate schimba in mod cardinal functionalitatea ei, aceasta preferinta apre sub forma unei liste de radio butoane in care sint trei optiuni pentru alegerea testului dimineata pentru a face alarma inactiva, aceste optiuni sint :



1. Simple Test : test simplu care apare pe ecranintro fereastra de dialog in care apar doua butoane cancel pentru a dezactiva alarma si snooze pentru a o face sa revina peste 5 minute.



1. Math Test : test matematic care apare pe ecran intro fereastra de dialog ce contine o simpla ecuatie matematica care trebuie rezolvata pentru a dezactiva alarma.



1. Logic Test : test logic ce apare sub forma unei ferestre de dialog ce contine un simplu test logic care trebuie rezolvat pentru a dezactiva alarma

In asa fel utilizatorul poate alege modul in care va fi trezit in functiile de necesitatile sale.

1. Ringtone : este o alta optiune foarte importanta care permite utilizatorului sa aleaga tonul care va cinta atunci cind va suna alarma, tonurile sint selectate din tonurile de sistem al sistemului de operare Android.

4. Implementare si testare

4.1 Implementare

Clasa AlarmRefreshingService :

* public void doAlarmWork(Intent intent) : seteaza din nou toate alarmele

Clasa AlarmService :

* synchronized void doAlarmWork(Intent intent) : este apelat cind a fost primit un semnal de alarma , metoda primeste prin extras parametrul ID dupa care cauta alarma in baza de date si afla daca trebuie sa fie apelata activitatea de afisare sau trebuie resetata alarma pe ziua urmatoare.

Clasa AlarmSetter :

* public void refreshAllAlarms() : face refresh la toate alarmele, le seteaza din nou
* public void setAlarm(Alarm alarm) : seteaza alarma alarm
* private void getAlarmUpToDate(Alarm alarm) : verifica daca alarma este trecuta de ora curenta atuni o muta pe ziua viitoare
* public void removeAlarm(String alarmId) : sterge alarma cu id-ul alarmId din sistem

Clasa OnAlarmReceiver:

* public void onReceive(Context context, Intent intent) : este apelat atunci cind un semnal de alarma a fost primit, primeste id-ul alarmei si il transmite mai departe la service-ul AlarmService ca sa faca actiunea necesara

Clasa OnBootReceiver:

* public void onReceive(Context context, Intent intent) : metoda este apelata atunc cind telefonul a fost resetat la primirea semnalului apeleaza service-ul alarmRefreshingService pentru a seta si actualiza toate alarmele

Clasa WakeAlarmIntentService:

* public static void acquireStaticLock(Context context) : pune lacat la context
* synchronized private static PowerManager.WakeLock getLock(Context context): primeste lacat ca aplicatia sa fie in regim wake
* final protected void onHandleIntent(Intent intent) : executa metoda doAlarmWork si elibereaza lacatul

Clasa AlarmDbAdapter:

* public static AlarmDbAdapter getInstance(Context context) : metoda singleton
* public AlarmDbAdapter open() throws SQLException : deschide DataBaseHelper si primeste acces la baza de date
* public void close() : inchide database helper
* public long createAlarm() : introduce alarma in baza de date
* public long deleteAlarm(Alarm alarm) : sterge alarma din baza de date
* public long deletAll() : sterge toate alarmele din baza de date
* public Cursor fetchAllAlarms() : returneaza cursor ce contine toate alarmele
* public Cursor fetchAlarm(String rowId) throws SQLException : returneaza cursor ce contine alarma cu id rowId
* public Cursor fetchNewAlarm() throws SQLException : returneaza alarma cu descrierea de alarma noua
* public Cursor fetchEnabledAlarms() throws SQLException : returneaza alarma cu descrierea de alarmele activate
* public long updateAlarm(Alarm alarm) : face update la alarma

Clasa AlarmDbUtilities:

* public static final ArrayList<Alarm> fetchCursor(Cursor c) : returneaza arrayList cu toate alarmele din cursor
* public static final ArrayList<Alarm> fetchAllAlarms(Context context) : returneaza toate alarmele din baza de date
* public static final Alarm fetchAlarm(Context context, String alarmId) : returneaza Alarma cu id-ul alarmId din baza de date
* public static final Alarm fetchNewAlarm(Context context) : returneaza din baza de date alarma nou creata
* public static final void deleteAlarm(Context context, Alarm alarm) : sterge din baza de date alarma
* public static final void deleteAll(Context context) : sterge toate alarmele din baza de date
* public static final void updateAlarm(Context context, Alarm alarm) : face update la alarma in baza de date
* public static final ArrayList<Alarm> fetchEnabledAlarms(Context context) : returneaza ArrayList cu toate alarmele care sunt activate

Clasa Alarm:

* public long getTime() : returneaza timpul la care va suna alarma
* public void setTime(long time) : seteaza timpul la care va suna alarma
* public String getWakeUpMode() : returneaza tipul de task
* public void setWakeUpMode(String wakeUpMode) : seteaza modul de trezire
* public String getRingtone() : returneaza ringtonul
* public void setRingtone(String ringtone) : seteaza ringtonul
* public String getDaysOfWeek() : returneaza zilele in care va functiona alarma
* public void setDaysOfWeek(String daysOfWeek) : seteaza zilele in care va functiona alarma
* public int isEnabled() : returneaza daca este activata
* public void setEnabled(int enabled) : seteaza daca alarma este activata
* public String getId() : returneaza id-ul alarmei
* public void setId(String id) : seteaza id-ul alarmei
* public String getDescription() : returneaza descrierea alarmei
* public void setDescription(String description) : seteaza descrierea alarmei

Clasa AlarmFragmentSettingsActivity:

* public void onCreate(Bundle SavedInstanceState) : metode oncreate pentru startul activitatii care va afisa preferintele

Clasa AlarmListActivity:

* public void onCreate(Bundle SavedInstanceState) : metode oncreate pentru startul activitatii care va afisa lista cu alrme
* private void emptyTextViewVisibility() : metoda ce determina daca trebuie afisat sau nu textview cu textul ca nu sint alarme
* public boolean onCreateOptionsMenu(Menu menu) : metoda ce creeaza meniu cu optiuni
* public boolean onOptionsItemSelected(MenuItem item) : se apeleaza la alegerea unui element din meniul cu optiuni si sterge toate alarmele
* public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) : se apeleaza la crearea de meniu context
* public boolean onContextItemSelected(MenuItem item) : se apeleaza la alegerea unui element din meniul context si sterge elementul ales
* protected void onActivityResult(int requestCode, int resultCode, Intent data) : se apeleaza la revenirea din preferinte si seteaza alarma sau actualizeaza pe una existenta

Clasa AlarmSettingsActivity:

* public void onCreate(Bundle savedInstanceState) : clasa ce creaza ecranul cu preferinte pentru versiuni <=3

Clasa MorningAlarmWidgetProvider:

* public void onReceive(Context context, Intent intent) : metoda creaza activitatea AlarmListActivity atuni cind widgetul de pe ecran a fost tastat
* public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) : metoda se apeleaza atunci cind widgetul este creat
* public static void updateWidget(Context context, boolean enabled) : metoda este apelata pentru a seta starea imaginii din widget

Clasa AlarmSettingsFragment:

* public void onCreate(Bundle SavedInstanceState) : metoda ce creeaza fragmentul ce afiseaza preferintele pentru versiuni >=3

Clasa ListPreferenceMultiSelect:

* public void setEntries(CharSequence[] entries) : Intializeaza elementele vectorului din care se vor alege checkboxurile.
* protected void onPrepareDialogBuilder(Builder builder) : metoda ce pregateste fereastra de dialog care va afisa optiunile.
* private boolean isCheckAllValue( int which ) : metoda returneaza true daca a fost selectat elementul all in caz contrar returneaza valoarea elementului selectat.
* private void checkAll( DialogInterface dialog, boolean val ) : metoda ce seteaza valoarea tuturor elementelor din lista.
* public String[] parseStoredValue(CharSequence val) : metoda ce face parsing la valoarea preferintei si returneaza vetorul de stringuri.
* private void restoreCheckedEntries() : metoda ce seteaza inapoi preferintele dupa ce sa iesit din fereastra de dialog.
* protected void onDialogClosed(boolean positiveResult) : se apeleaza la inchiderea preferintei
* protected static String join( Iterable< ? extends Object > pColl, String separator ) : metoda ce uneste valorile elementelor intrun singru string folosind un separator al clasei
* public static boolean contains( String straw, String haystack, String separator ) : metoda ce afla daca in preferinte se contine un element

Clasa TimePreference:

* public void initialize() : initializeaza preferinta
* protected View onCreateDialogView() : creaza fereastra de dialog cu timepicker
* public void onDialogClosed(boolean positiveResult) : salveaza starea la inchiderea ferestrei de dialog

Clasa AlarmTask:

* public void onCreate(Bundle savedInstanceState) : metoda ce se apeleaza cint se creaza activitatea
* protected void setSignals() : metoda ce seteaza semnalele sonore si de vibrare
* protected abstract void solveCondition() : metoda abstracta ce seteaza valoarea variabilei finishAlarm cu true daca sa rezolvat sarcina sau da snooze
* public static Alarm getAlarm() : metoda ce returneaza ultima alarma din clasa
* public static void setAlarm(Alarm newAlarm) : metoda ce seteaza ultima alarma din clasa
* synchronized public static void setActive() : metoda ce seteaza valoarea variabile active cu true
* synchronized public static void setInActive() : metoda ce seteaza valoarea variabile active cu false
* public static boolean isActive() : metoda ce returneaza valoarea variabile active
* public Dialog getDialog() : metoda ce returneaza fereastra de dialog cu sarcina

4.2 Testare

4.2.1 Fundamente

Framework-ul de testare Android, o parte integrata in mediului de dezvoltare, oferă o arhitectură și instrumente puternice care ajută de testat fiecare aspect al aplicației la fiecare nivel de unitate.

Framework-ul de testare are urmatoarele caracteristici cheie:

Suite de testare Android sunt bazate pe JUnit. Putem utiliza simplu JUnit pentru a testa o clasă care nu apeleaza Android API, sau extensii Android JUnit pentru a testa componentele Android. Pentru utilizatorii noi la testarea Android, se poate începe cu testele de uz general, cum ar fi AndroidTestCase și apoi se poate trece la clase mai sofisticate.

Extensiile Android JUnit oferă componente specifice pentru diferite cazuri de testare. Aceste clase oferă metode ajutatoare pentru a crea obiecte false și metode care ajută la controlul ciclului de viață al unei componente.

Suitele de testare sunt incluse în pachetele de testare, care sunt similare cu pachete principale de aplicații, astfel încât nu apara nevoia pentru a învăța un nou set de instrumente sau tehnici pentru proiectarea si construirea de teste.

Instrumentele de SDK pentru construcție și teste sunt disponibile în Eclipse cu ADT, și, de asemenea, sub formă de linie de comandă pentru a fi utilizate cu alte IDE-uri. Aceste instrumente obțin informații de la proiectul de aplicare testat și foloseste aceste informații pentru a crea în mod automat fișierele de build , fișier manifest, și structura de directoare pentru pachetul de testare.

SDK-ul oferă, de asemenea monkeyrunner, un API pentru testarea dispozitivelor folosind programe Python, și UI/Application Exerciser Monkey, un instrument de linie de comandă pentru UI stress-testing prin trimiterea de evenimente pseudo-aleatoare pentru un dispozitiv.

4.2.2 Structura testelor

Componentele android de build si test asuma ca proiectele de test sun organizate intro structura standard de teste, cazuri de testare, pachete de testare.

Testarea in android este bazata pe JUnit. Testarea Android este bazata pe JUnit. În general, un test JUnit este o metodă ale cărei testeaza o parte a aplicatiei testate. Metodele de testare trebuie organizate în clase numite cazuri de testare (sau suite de testare). Fiecare test este un test izolat al unui modul individual în aplicația supusa testarii. Fiecare clasă este un container pentru metodele de testare aferente, cu toate că creează adesea si metode helper.

În JUnit, se face build la unul sau mai multe fișiere sursă de test într-un fișier de clasă. Similar, în Android se folosesc instrumente de build din SDK pentru a face build la unul sau mai multe fișiere sursă de testare în fișiere de clasă într-un pachet de teste Android. În JUnit, se un test-runner pentru a executa clase de test. În Android, se utilizeaza instrumente de testare pentru a încărca pachetul de testare și aplicare în cadrul testului, iar instrumentele apoi executa un runner de testare specifi pentru Android.

4.2.3 Proiecte de testare

Proiectele de testare ca si proiectele de aplicatii Android sunt organizate sub forma de proiecte.

Un proiect de test este un director sau proiect Eclipse în care se creaza cod sursă, fișierul manifest, și alte fișiere pentru un pachet de teste . Android SDK conține instrumente pentru Eclipse cu ADT și pentru linia de comandă care creează și actualizează proiecte de testare automat. Instrumentele creaza directoare care se utilizeaza pentru codul sursă și a resurselor și fișierului manifest pentru pachetul de testare. Instrumentele de linie de comandă creează, de asemenea, fișierele Ant de care este nevoie.

Se poate crea un proiect de testare oriunde în sistemul de fișiere, dar cea mai bună abordare este de adaugat proiect de test astfel încât directorul de teste root / este la același nivel ca și directoriul de proiect src / a aplicatiei generale. Acest lucru va ajuta la gasirea testelor asociate cu o aplicație.

4.2.4 Rulare de teste

Cazuri de testare sunt conduse de o clasa de test runner care încarcă clasa de test, set up-uri, se execută, și sfirseste fiecare test. Un runner de teste Android trebuie să fie, de asemenea, instrumentat, astfel încât sistemul de instrumente pentru startul aplicației sa poata controla modul de incarcare a pacheteleor de teste cu cazuri de testare. Runnerul care va rula testele pentru platforma Android se specifica in în fișierul de manifest a pachetului de test.

InstrumentationTestRunner este clasa primara de testare Android. Ea extinde frameworkul runner de testare JUnit și este de asemenea instrumentata. Poate rula oricare dintre categoriile de test case oferite de Android si suporta toate tipurile posibile de testare.

Trebuie de specificat InstrumentationTestRunner sau o subclasă în fișierul manifest a pachetului de testare, în elementul <instrumentation>. De asemenea, codul InstrumentationTestRunner rezidă în android.test.runner bibliotecă partajată, care nu este în mod normal, legat de cod Android. Pentru a o include, trebuie să se precizeze într-un element de <uses-library>. Nu este nevoie să se înființeze aceste elemente de catre utilizator. Eclipse cu ADT, si instrumentul de linie de comandă Android le construiesc în mod automat și să le adăuga la fișierul manifest al pachetului de test.

Notă: Dacă se utilizeaza un runner diferit de InstrumentationTestRunner, trebuie de modificat elementul <instrumentation> pentru a indica clasa pe care se doreste să fie utilizata.

Pentru a rula InstrumentationTestRunner, se folosesc clase interne sistem apelate de instrumente de Android. Când se executa un test în Eclipse cu ADT, clasele sunt numite în mod automat. Când se executa un test de la linia de comandă, se ruleaza clasele de test cu Android Debug Bridge (ADB).

Clasele de sistem de încarca și ruleaza pachet de teste, inchide toate procesele care execută o instanță a aplicatiei de testat, și apoi încarca o nouă instanță a aplicatiei de testat.Testele trec apoi controlul la InstrumentationTestRunner, care ruleaza fiecare clasă de test din pachetul de testare. Se poate controla, de asemenea, care cazuri de testare și metode sunt exploatate folosind setările în Eclipse cu ADT, sau folosind flaguri cu instrumentele de linie de comandă.

Nici clasele sistemului, nici InstrumentationTestRunner nu ruleaza aplicația testata acest lucru este facut de cazurile de testare în mod direct. Ele apeleaza metode din alicatie în cadrul testului, sau apeleaza propriile metode care declanșeaza evenimente a ciclului de viață în aplicatia testata. Aplicatia este sub controlul complet al cazului de testare, care îi permite să configureze mediul de testare înainte de a rula un test.

Pentru aplicatia mea am construit proiectul de MorningAlarmTest in care rulez toate testele pentru clasele din aplicatie.

5. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

Tema data mi sa parut foarte interesanta si folositoare pentru dezvoltarea mea personala, datorita ei am invatat foarte multe lucruri noi care neaparat imi vor folosi in cariera mea ulterioara de specialist. Datorita acestei teme am invatat cum sa folosesc si sa implementez design patternurile Factory, Singletone, Adapter, la fel am invatat cum sa proiectez o aplicatie eficienta care va rula pe dispozitive cu sistemul de operare Android.

Rezolvind sarcinile puse in tema curenta am invatat cum trebuie de lucrat in mod eficient, ordinea si modul in care trebuie analizata problema si cautata solutia.

Dezvoltari ulterioare:

1. Se mai pot adauga si alte moduri de trezire, adica setare de noi taskuri.
2. Se pot adauga mai multe melodii pentru trezire.
3. Se poate implementa un sistem prin care utilizatorul singur alege melodia care va suna cind se va activa alarma.

Bibliografie

1. <http://developer.android.com/>
2. <http://en.wikipedia.org/wiki/Adapter_pattern>
3. <http://cursuri.cs.pub.ro/~poo/wiki/index.php/Design_Patterns_Basics>

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ........................................