**Wydział Informatyki, Elektroniki i Telekomunikacji**

**Katedra Informatyki**



**Zegar do gier – przypominacz / budzik dla wielu**

**zdarzeń o charakterze cyklicznym**

**Specyfikacja techniczna produktu**

**Damian Kudas, Michał Mrowczyk**

**5 czerwca 2013 r.**

Zawartość

[1. Wstęp 3](#_Toc358831858)

[2. Wykorzystywane elementy z API Androida 3](#_Toc358831859)

[3. Podstawowe pakiety i ich relacje 3](#_Toc358831860)

[4. Pakiet org.clock 6](#_Toc358831861)

[a. Klasa AlarmState 7](#_Toc358831862)

[b. Klasa Listener 7](#_Toc358831863)

[i. Metoda execute() 7](#_Toc358831864)

[c. Klasa Notifier 8](#_Toc358831865)

[d. Klasa MainActivity 8](#_Toc358831866)

[e. Klasa ShowAlarmsActivity 8](#_Toc358831867)

[5. Pakiet org.model 9](#_Toc358831868)

[a. Klasa Alarm 9](#_Toc358831869)

[b. Klasa AlarmList 10](#_Toc358831870)

[i. Metoda synchronizeAlarms 10](#_Toc358831871)

[6. Pakiet org.parser 11](#_Toc358831872)

[a. Klasa AlarmParser 11](#_Toc358831873)

[i. Metoda parseAlarms 11](#_Toc358831874)

[ii. Metoda saveAlarms 11](#_Toc358831875)

[b. Struktura przykładowego XML’a 12](#_Toc358831876)

[7. Pakiet org.timepicker 13](#_Toc358831877)

[a. Klasa Time 14](#_Toc358831878)

[b. Klasa StateMap 14](#_Toc358831879)

[c. Klasa Ball 14](#_Toc358831880)

[d. Klasa MainActivity 14](#_Toc358831881)

[e. Klasa PeriodActivity 15](#_Toc358831882)

# Wstęp

Niniejszy dokument stanowi kompletną specyfikację techniczną produktu. Zawiera szczegóły implementacyjne wraz z opisem zastosowanych rozwiązań oraz użytych algorytmów.

***UWAGA***: Dokument ten należy stosować jako przewodnik po modułach (pakietach) oraz klasach naszej aplikacji wraz z wyszczególnionymi metodami. Czytanie tego dokumentu powinno być wsparte czytaniem dokumentacji wygenerowanej w javadocu oraz kodu źródłowego naszej aplikacji. Javadoc znajduję się w folderze doc i jest spakowany razem ze źródłami projektu.

W dalszej części tego dokumentu omawiamy dokładnie strukturę projektu z podziałem na pakiety, klasy oraz ważniejsze z punktu widzenia programisty metody.

# Wykorzystywane elementy z API Androida

W aplikacji wykorzystujemy obiekt klasy MediaPlayer: (<http://developer.android.com/reference/android/media/MediaPlayer.html>)

Służy on do odgrywania alarmów. Po zakończonej pracy wywoływana jest jego metoda: release() w celu zwolnienia zasobów z nim stowarzyszonych.

Oprócz tego korzystamy z następujących elementów GUI (widgetów):

* TextView - <http://developer.android.com/reference/android/widget/TextView.html>
* Button - <http://developer.android.com/reference/android/widget/Button.html>
* EditText - <http://developer.android.com/reference/android/widget/EditText.html>
* Canvas - <http://developer.android.com/reference/android/graphics/Canvas.html>

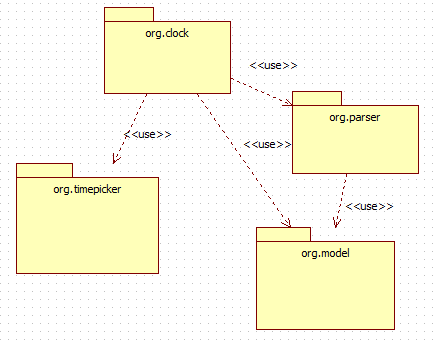
Ponadto korzystamy ze standardowych konstrukcji Androida (jak np. Activities) w celu obsługi osobnych ekranów i akcji na nich wykonywanych przez użytkownika:

<http://developer.android.com/reference/android/app/Activity.html>

# Podstawowe pakiety i ich relacje

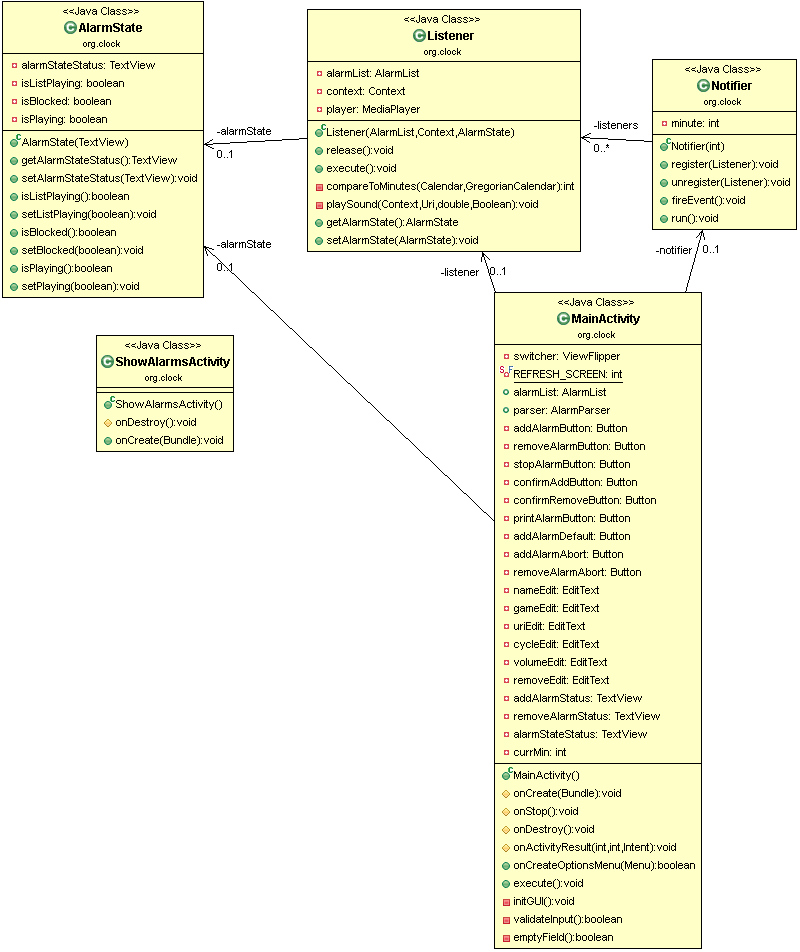
* org.clock – centralny pakiet w aplikacji – znajdują się w nim klasy odpowiedzialne za tworzenie całego GUI (MainActivity) oraz zależność Publisher – Listener (Publisher publikuje zdarzenia związane ze zmianą minuty, a Listener odbiera je od Publisher’a i decyduje czy należy odegrać alarm). Pakiet zawiera też klasę: AlarmState, która działa jako pewien monitor na stan alarmów i ma na celu blokowanie/odblokowywanie pewnych operacji w GUI w zależności np. od tego czy alarm jest w danej chwili odgrywany. Pakiet ten zależy od wszystkich innych pakietów.
* org.model – pakiet stanowiący ogólny model danych dla alarmów i ich kolekcji – niezależny od innych pakietów
* org.parser – pakiet zawierający jedną klasę, która obsługuje parsowanie XML’a – pakiet ten odwołuje się do pakietu org.model w celu inicjalizacji struktur danych
* org.timepicker – pakiet, którego główną rolą jest zapewnienie funkcjonalności zmieniania okresu, godziny i minuty alarmu za pomocą wygodnego GUI. Zawiera także klasy pomocnicze, służące do przechowywania stanu definiowanych ustawień

Poniższy diagram przedstawia pakiety w naszej aplikacji oraz zależności między nimi:



# Pakiet org.clock

Diagram klas dla pakietu org.clock:



## Klasa AlarmState

Klasa ta służy do przechowywania informacji o stanie alarmu. Zawiera trzy interesujące zmienne:

* isListPlaying – ustawiona na true jeśli w danej chwili program odgrywa listę alarmów (alarmy ustawione na tą samą minutę). Zauważ, że program mimo to może nie odgrywać żadnego alarmu, bo aplikacja może być w stanie między alarmami.
* isPlaying – ustawiona na true jeśli program odgrywa w danej chwili alarm. Jeśli program jest ‘między’ alarmami lub isListPlaying == false, to isPlaying też jest false
* isBlocked – ustawiona na true jeśli wchodzimy do trybu dodawania alarmów lub usuwania alarmów. Po wyjściu z tych trybów jest ustawiana na false. Oznacza czy w danym stanie aplikacji alarmy będą odgrywane, czy ignorowane

## Klasa Listener

Klasa, której celem jest wywoływanie metody synchronizującej na rzecz listy alarmów oraz odgrywanie alarmów z wykorzystaniem MediaPlayer. Najważniejszą metodą w tej klasie jest metoda execute, dlatego opisujemy jej działanie.

### Metoda execute()

1. Wywołanie metody synchronizującej na liście alarmów
2. Jeśli zwrócona lista alarmów jest pusta lub aplikacja jest w stanie BLOCKED to koniec, w przeciwnym razie idź do pkt. 3
3. Ustaw zmienną isListPlaying na true i odgrywaj alarmy jeśli ich czas odegrania w minutach jest taki sam jak obecny czas w minutach
4. Po skończeniu odgrywania alarmów ustaw zmienną isListPlaying na false

Powyższy opis stanowi szkic algorytmu działania metody. Po więcej informacji proszę zaglądnąć do kodu i javadoca

## Klasa Notifier

Klasa rejestrująca Listenery – które potencjalnie mogą odgrywać alarmy. Co minutę publikuje ona zdarzenie o zmianie czasu Listener’om tak, aby tamte mogły synchronizować na bieżąco swoje alarmy. Klasa Notifier pełni rolę, którą można by było zastąpić klasą Timer w Javie (wysyła sygnały co określony czas)

## Klasa MainActivity

Główna klasa aplikacji – tworzy widoczne GUI oraz wątki Listener’a i Publisher’a, które służą do obsługi logiki odgrywania alarmów. Klasa ta zawiera także walidację wprowadzanych przez użytkownika aplikacji danych – metoda:

**private boolean** validateInput()

W celu odbierania informacji od ShowAlarmsActivity została zdefiniowana metoda:

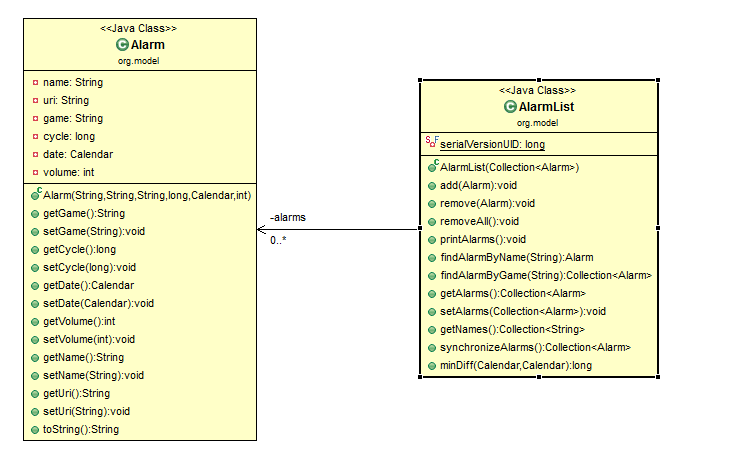
**protected** **void** onActivityResult(**int** requestCode, **int** resultCode, Intent data)

## Klasa ShowAlarmsActivity

Klasa tworząca listę zawierającą informację o nazwach zdefiniowanych alarmów. Klasa ta uruchamia także Activity odpowiedzialne za ustawianie okresu, godziny oraz minuty alarmu. Zwraca ona wynik do MainActivity w postaci stanu zapisanego jako HashMap z kluczem będącym nazwą alarmu, zaś wartością będącą tablicą czteroelementową: {hour, minute, hourPeriod, minutePeriod}. Zobacz javadoca po więcej informacji.

# Pakiet org.model

Diagram klas dla pakietu org.model:



## Klasa Alarm

Proste Java’ove POJO (Plain Old Java Object) – zawiera pola, gettery oraz settery reprezentujące alarm. Opis znaczenia tych pól:

* name – nazwa alarmu
* uri – ścieżka do dźwięku alarmu
* game – nazwa gry
* cycle – cykl w minutach alarmu
* date – czas odegrania alarmu z dokładnością do minut
* volume – głośność alarmu w zakresie od 0 do 100

Klasa ta w naturalny sposób przekłada się na reprezentację XML’ową, którą opiszemy przy okazji opis pakietu parsera (rozdział 5)

## Klasa AlarmList

Zawiera kolekcję alarmów oraz proste metody dostępowe do alarmów (np. wyszukiwanie alarmów po nazwie gry). Najważniejszą funkcjonalnością tej klasy jest metoda umożliwiająca synchronizację alarmów.

### Metoda synchronizeAlarms

Kod tej metody zamieszczamy w dokumentacji, gdyż ma ona kluczowe znaczenie w działaniu aplikacji:

**public** Collection<Alarm> synchronizeAlarms() {

Collection<Alarm> result = **new** Vector<Alarm>();

Calendar now = **new** GregorianCalendar();

**for** (Alarm alarm : alarms) {

**long** diff = minDiff(now, alarm.getDate());

**if** (diff == 0) {

result.add(alarm);

} **else** **if** (diff > 0) {

// Need for synchronization, because alarm time is outdated

alarm.getDate().add(Calendar.*MINUTE*, (**int**) (alarm.getCycle() \* (diff / alarm.getCycle() + 1)));

**if** (diff % alarm.getCycle() == 0) {

alarm.getDate().add(Calendar.*MINUTE*, (**int**) (-diff));

result.add(alarm);

}

}

}

**for** (Alarm alarm : result) {

alarm.getDate().add(Calendar.*MINUTE*, (**int**) alarm.getCycle());

}

**return** result;

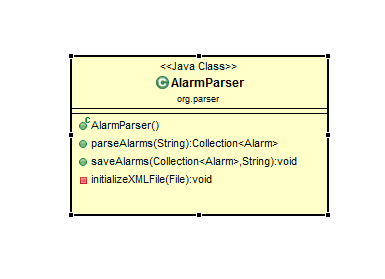
}

W skrócie metoda ta:

* Tworzy pustą kolekcję alarmów oraz obiekt Calendar zawierający obecny czas
* Dla każdego zdefiniowanego alarmu sprawdza, czy jego czas odgrywania jest taki sam (z dokładnością do minut) jak obecny czas. Jeśli tak to dodaje go do listy.
* W przeciwnym wypadku jeśli czas odgrywania alarmu jest starszy niż obecny czas, to metoda przesuwa czas odgrywania tego alarmu o wielokrotność okresu do przodu, w taki sposób, aby czas wynikowy był co najmniej równy obecnemu (lub większy, ale nie więcej niż o wielokrotność okresu)
* W tym celu metoda ta korzysta z operacji modulo: %

# Pakiet org.parser

Diagram klas dla pakietu org.parser:



Jak widać pakiet ten zawiera tylko jedną klasę: AlarmParser.

## Klasa AlarmParser

Klasa AlarmParser służy do parsowania XML’a. Do tego celu wykorzystywany jest parser DOM (Document Object Model). Parser ten tworzy strukturę dokumentu XML w pamięci w postaci drzewa, umożliwiając przy okazji manipulowanie tym drzewem (dodawanie i usuwanie elementów oraz atrybutów). Klasa ta zawiera dwie proste metody:

## Metoda parseAlarms

**public** Collection<Alarm> parseAlarms(String path)

Umożliwia parsowanie alarmów zdefiniowanych w pliku XML znajdującym się w systemie plików w ścieżce – path

## Metoda saveAlarms

**public** **void** saveAlarms(Collection<Alarm> alarms, String path)

Metoda umożliwia zapisywanie kolekcji alarmów – alarms do pliku pod ścieżką – path

***UWAGA***: Jeśli w pliku znajdowały się wcześniej jakieś alarmy, to są one najpierw usuwane, a następnie zapisywana jest tam cała kolekcja alarmów.

## Struktura przykładowego XML’a

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<alarms>

<alarm>

<name>alarm1</name>

<uri>/data/data/org.clock/files/adams.mp3</uri>

<game>WOW</game>

<cycle>140</cycle>

<volume>80</volume>

<date>

<year>2013</year>

<month>4</month>

<day>4</day>

<hour>15</hour>

<minute>30</minute>

</date>

</alarm>

<alarm>

<name>alarm2</name>

<uri>/data/data/org.clock/files/circus.mp3</uri>

<game>TitanQuest</game>

<cycle>120</cycle>

<volume>70</volume>

<date>

<year>2013</year>

<month>4</month>

<day>4</day>

<hour>16</hour>

<minute>30</minute>

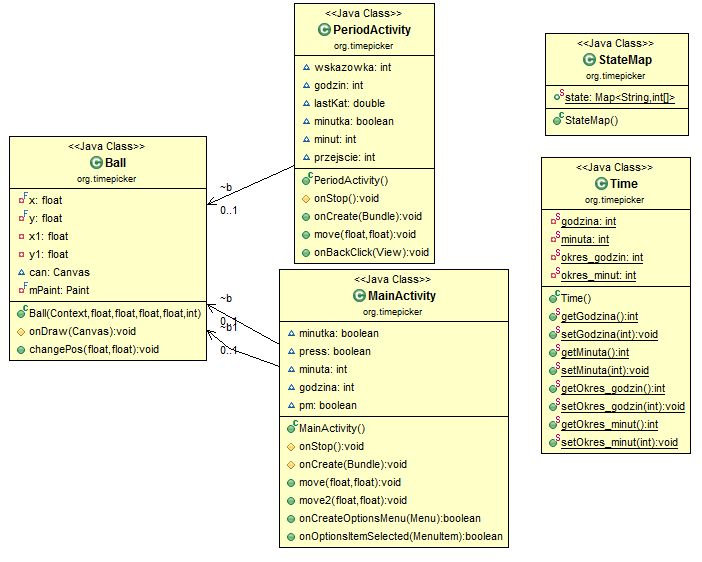
</date>

</alarm>

</alarms>

# Pakiet org.timepicker

Diagram klas dla pakietu org.timepicker:



## Klasa Time

Prosta klasa zawierająca pola statyczne:

* godzina – jak sama nazwa wskazuje jest to ostatnio ustawiona godzina
* minuta - jest to ostatnio ustawiona przez użytkownika minuta
* okres\_godzin - jest to ostatnio ustawiony okres w godzinach (może być również większy niż 24h)
* okres\_minut – jest to ostatnio ustawiony okres w minutach

Zawartość tej klasy przydaje się później w Activities, które są wyżej w hierarchii do wyciągania informacji o ustawieniach użytkownika przy aktualizacji alarmu

## Klasa StateMap

Zawiera statyczną HashMapę java’ovą służącą do wyciągania odwzorowania między nazwą alarmu, a czwórką: {godzina, minuta, okres\_godzin, okres\_minut}. Umożliwa łatwą aktualizację ustawień alarmów na bazie zmian wprowadzonych przez użytkownika

## Klasa Ball

Służy do rysowania wskazówki na Canvas’ie (czyt: płótnie)

## Klasa MainActivity

Służy do ustawień godziny i minuty dla alarmu. Zegar jest 12 godzinny, więc aby obsłużyć ustawienia dla 24h doby wprowadzono zmienną pm (z angielskiego: post meridian). Z technicznego punktu widzenia klasa wykorzystuje kilka funkcji matematycznych (np. funkcje acos(x) oraz asin(x)) do wyznaczania kątów. Wszystko to przeliczane jest w odpowiedni sposób, aby wyłuskać godzinę i minutę. Klasa ta zwraca potrzebne informacje do wywołującej ją Activity: ShowAlarmsActivity

Więcej informacji na temat tej klasy można znaleźć w javadocu oraz kodzie źródłowym projektu.

## e. Klasa PeriodActivity

Odpowiednik klasy MainActivity dla ustawień okresu alarmu. Klasa ta zapamiętuje ile przejść – cykli przy obracaniu wskazówki zostało wykonanych, co umożliwia zapamiętanie okresów większych od 24h. Klasa ta zwraca do wywołującej ją Activity (MainActivity) informacje o okresie w godzinach i minutach. Szczegółowe informacje na temat klasy w załączonym javadocu oraz kodzie źródłowym.