

Wstęp do części drugiej artykułu

W części pierwszej artykułu zamieściłem informacje nt. Klasy `Screen` oraz jednej z klas z niej dziedziczących - klasie `Form` oraz opis komponentów, które mogą być jego częścią - elementy klasy `Item`. Poniżej zamieszczam więc opis kolejnych trzech klas dziedziczących z klasy `Screen`: `TextBox`, `Alert` oraz `List`. Zachęcam jednak do przestudiowania pierwszej części tego artykułu, ponieważ obie części są ze sobą bardzo mocno związane.

Pole tekstowe - klasa `TextBox`

`TextBox` jest komponentem umożliwiającym edycję dużej ilości tekstu. Ponieważ dziedziczy z klasy `Screen`, zajmuje cały ekran i można do niego dołączyć stale przewijający się w poprzek ekranu tekst (*ticker*). Abstrahując od tych różnic oraz od faktu, iż nie można do niego dołączyć klasy nasłuchującej zmian wprowadzanych przez użytkownika, jest to element bardzo podobny do dziedziczącego z klasy `Item`, a omówionego wcześniej elementu `TextField`.

Kiedy na przykład - chcąc stworzyć obiekt tej klasy - korzystamy z jedynej jej konstruktora:

```
public TextBox(String title, String text, int maxSize, int constraints),
```

to widzimy, że konstruktor ten ma dokładnie taką samą postać, jak konstruktor klasy `TextField`. I podając czwarty argument konstruktora, korzystamy ze stałych zdefiniowanych również w klasie `TextField` (tam też są one dokładnie omówione).

Również metody pozwalające operować na danych zawartych w ekranie edycji (pobranie, skasowanie czy zmiana zawartości) są takie same jak w przypadku klasy `TextField`.

Przykład zastosowania `TextBox`'a znajdziesz poniżej - w sekcji dotyczącej innej klasy dziedziczącej ze `Screen` - klasy `Alert`.

Wyświetlanie komunikatów - klasa `Alert`

Klasa `Alert` - wywiedziona z klasy `Screen`, jej celem jest ułatwienie programiście wyświetlenie prostych komunikatów na ekranie urządzenia. Aby wyświetlić taki komunikat, korzystamy z jednej z dwóch postaci metody `setCurrent` z klasy `Display`:

```
public void setCurrent(Displayable next);
public void setCurrent(Alert alert, Displayable nextDisplayable);
```

Pierwszą postać już poznaliśmy, druga w przypadku komunikatów okazuje się bardzo przydatna. Jako pierwszy argument, podajemy w niej `Alert` do wyświetlenia, drugim jest `Displayable`, który ma zostać wyświetlony po nim.

Komunikaty mają następujące atrybuty i właściwości:

1. Nagłówek (*title*) - atrybut odziedziczony z klasy `Screen`. Nie wymagany.
2. Treść (*alertText*)

3. Ikona (*alertImage*) - ikona wyświetlana (lub nie - zależnie od urządzenia) z komunikatem. Atrybut nie wymagany.
4. Czas wyświetlania(*timeout*) - atrybut określający jak długo komunikat powinien być wyświetlany. Z tym atrybutem związane są dwie metody:

```
public int getTimeout();
public void setTimeout(int time);
```

Pierwsza służy do sprawdzenia ustawionego czasu, druga może jako argumenty przyjąć dwa rodzaje wartości:

1. wartość dodatnią, określającą (w milisekundach) po jakim czasie komunikat powinien zniknąć.
 2. Wartość określona przez stałą `FOREVER` w klasie `Alert`. Oznacza ona, iż komunikat będzie wyświetlany dopuki nie zwolni go użytkownik.
5. typ komunikatu (*alertType*) - atrybut pomagający urządzeniu w odpowiednim wyświetleniu komunikatu. Być może również zagranie odpowiedniego do typu informacji tonu. Atrybut jest typu `AlertType`, i przyjmuje wartość jednej z poniższych stałych:
 1. `AlertType.ALARM`
 2. `AlertType.CONFIRMATION`
 3. `AlertType.ERROR`
 4. `AlertType.INFO`
 5. `AlertType.WARNING`

Klasa `Alert` ma następujące konstruktory, za pomocą których tworzymy wyświetlane na ekranie komunikaty:

```
public Alert(String title);
public Alert(String title, String alertText,
             Image alertImage, AlertType alertType);
```

W przypadku użycia pierwszego z nich ustawione zostają domyślne wartości. Domyślną wartość atrybutu *timeout* możemy poznać korzystając z metody:

```
public int getDefaultTimeout();
```

Oczywiście komunikatu o czasie wyświetlania ustalonym na inny niż wskazuje na to stała `FOREVER`, użytkownik nie może usunąć przed jego upływem.

Nie pomoże tu również dodanie do obiektu typu `Alert` komend (`Command`), gdyż jest to po prostu niemożliwe. Powodem jest przesłonięcie przez klasę `Alert` metod `addCommand()` i `setCommandListener()` z klasy `Displayable`.

Prosty przykład wyświetlenia dwu różnych komunikatów zamieściłem poniżej. Przykład ilustruje również zastosowanie `TextBox`'a.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```

/**
 * MIDlet prezentuje zastosowanie elementów TextBox oraz Alert.
 * @author Konrad Palczewski
 */
public class TextBoxTryout extends MIDlet implements CommandListener {
    private Command exitCommand = new Command("Wyjście", Command.EXIT, 2);
    private Command statsCommand =
        new Command("Statystyka", Command.SCREEN, 2);
    private Display display;
    private TextBox tbox;

    public TextBoxTryout() {
        display = Display.getDisplay(this);

        //Tworzę TextBox z przewijanym tekstem i początkowym wypełnieniem:
        tbox =
            new TextBox("Twój życiorys", "Tu wpisz swój życiorys w +
                telegraficznym skrócie...", 400, TextField.ANY);
        tbox.setTicker(new Ticker("To jest tzw. ticker..."));

        tbox.addCommand(exitCommand);
        tbox.addCommand(statsCommand);
        tbox.setCommandListener(this);
    }

    public void startApp() {
        //Tworzę alert informacyjny, zniknie po potwierdzeniu przez użytkownika
        Alert confAlert =
            new Alert("Potwierdzenie!", "Tę aplikację możesz oglądać tylko +
                jeśli kochasz Javę!", null, AlertType.CONFIRMATION);
        confAlert.setTimeout(Alert.FOREVER);
        display.setCurrent(confAlert, tbox);
    }

    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {}

    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(true);
            notifyDestroyed();
        }
        else if (c == statsCommand) {
            //Zbieram statystykę TextBox'a
            char[] input = new char[tbox.getMaxSize()];
            int size = tbox.getChars(input);

            //Tworzę alert informacyjny, który zniknie po 3 sekundach:
            Alert infoAlert =
                new Alert("Statystyka znaków", "Wpisałeś/aś już "+size+
                    "\nz dozwolonych "+tbox.getMaxSize()+" znaków.",
                    null, AlertType.INFO);
            infoAlert.setTimeout(3000);
            display.setCurrent(infoAlert, tbox);
        }
    }
}

```

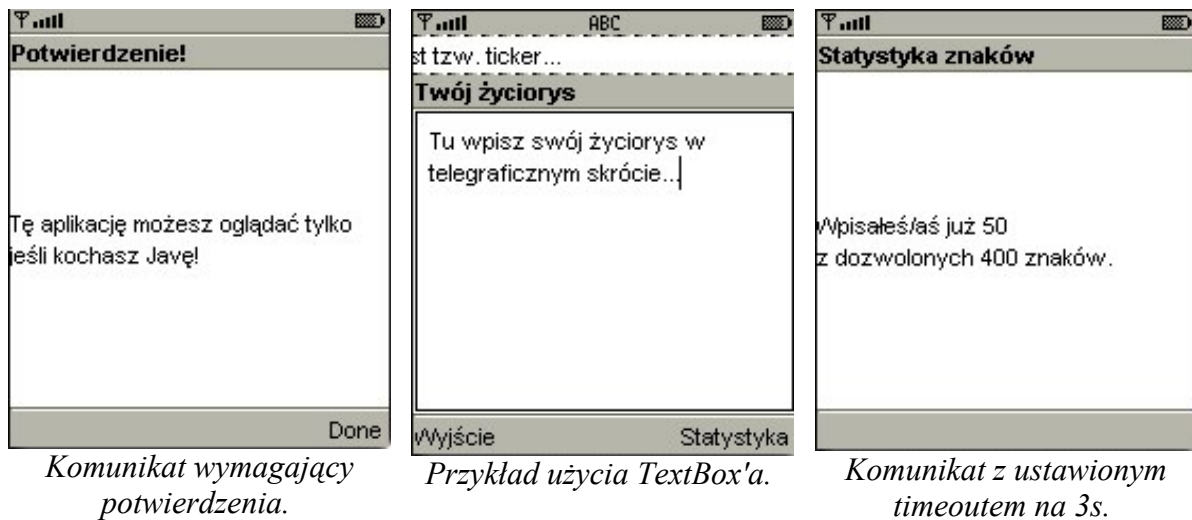
```

    }
}

```

A oto efekt wykonania powyższego kodu:

TABELA1:



Na powyższym przykładzie widzimy więc, jak u góry drugiego screenu ekranu przewija się ticker. Możemy również zauważyć (na ostatnim screenie), iż kiedy timeout ustawiony jest na konkretną wartość, wówczas użytkownik nie może zwolnić komunikatu.

Widać również, że mimo, iż rozróżniłem oba zaprezentowane komunikaty pod względem ich typów, to emulator, na którym uruchomiony został MIDlet, nie rozróżnił ich zupełnie. Jest to więc zależne od urządzenia.

Lista elementów - klasa List

Lista jest bardzo podobna do komponentu formularza - `ChoiceGroup`. Większość wsólnego interfejsu została zgrupowana w interfejsie `Choice`, który obie te Klasy implementują. Dlatego działanie na elementach listy będzie takie samo, jak działanie na elementach komponentu `ChoiceGroup`. Dodawanie, modyfikowanie i usuwanie pól grupy zostało opisane w artykule "Tworzenie GUI wysokiego poziomu - cz.1". Tutaj opiszę jedynie różnice pomiędzy klasą `List`, a klasą `ChoiceGroup`.

`List` wywiedziony jest z klasy `Screen`, a co za tym idzie zajmuje cały ekran, można do niego dołączać własne komendy, oraz *nie* można dołączyć do niego `ItemStateListener` (bo nie wywodzi się z klasy `Item`).

Tworzenie elementu `List` jest bardzo podobne do tworzenia elementu `ChoiceGroup`. Dwa konstruktory mają taką samą formę w obu klasach:

```

public List(String title, int type);
public List(String title, int type, String[] strings, Image[] images);

```

Jedyna różnica polega na tym, iż pierwszy argument jest tutaj interpretowany jako nagłówek listy, a nie jako etykieta komponentu. Dwa poznane wcześniej typy grupy (`EXCLUSIVE` i `MULTIPLE`) - i tutaj mają zastosowanie. Jednak lista dodaje jeszcze jeden typ, mianowicie typ `IMPLICIT`.

Wcześniej poznane dwa typy zachowują się tak samo jak w przypadku ich odpowiedników w klasie `ChoiceGroup`. Trzeci - `IMPLICIT` - jest bardzo podobny do typu `EXCLUSIVE`. Podobieństwo polega na tym, iż w obu przypadkach można wybrać tylko jeden element ze wszystkich dołączonych do komponentu. Różnica skolei polega na wyglądzie elementów w obu typach listy. Bo, podczas gdy lista typu `EXCLUSIVE` prezentowana jest jako zbiór *radio buttonów*, to lista typu `IMPLICIT` prezentowana jest jako zwykła lista elementów. W tym drugim jednak przypadku automatycznie dodawana jest komenda typu *"select"*, i wybór konkretnego elementu następuje przez jego aktywację, a za wybrany element przyjmuje się ten, na którym skupiona była uwaga na ekranie w momencie aktywacji komendy.

Ten element klasy `Command` zadeklarowany jest w klasie `List` jako stała `SELECT_COMMAND`. I te właśnie wielkości (referencja do obiektu `List` i do aktywowanej komendy) przekazywane są jako argumenty dołączonej do listy `CommandListener`'a.

Poniższy przykład prezentuje sposób korzystania z listy typu `IMPLICIT` i kilka formularzy z elementami `ChoiceGroup` różnych typów.

Oto kod MIDletu:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * MIDlet ma zaprezentować obiekt List i formularz z
 * różnymi typami ChoiceGroup.
 * @author Konrad Palczewski
 */
public class ListTryout
extends MIDlet implements CommandListener, ItemStateListener {

    private Command exitCommand= new Command("Wyjście", Command.EXIT, 2);
    private Command backCommand = new Command("Wróć", Command.BACK, 1);
    private Display display;
    private List list = new List("Menu", List.IMPLICIT);
    private Form form1 = new Form("EXCLUSIVE ChoiceGroup");
    private Form form2 = new Form ("MULTIPLE ChoiceGroup");
    private ChoiceGroup chGroup1 =
        new ChoiceGroup("Wybierz opcję:", ChoiceGroup.EXCLUSIVE);
    private ChoiceGroup chGroup2 =
        new ChoiceGroup("Wybierz opcję:", ChoiceGroup.MULTIPLE);

    public ListTryout() {
        display = Display.getDisplay(this);

        //Definiuję wygląd listy (głównego menu)
        list.setTicker(new Ticker("Wybierz rodzaj ChoiceGroup..."));
        list.append("EXCLUSIVE", null);
        list.append("MULTIPLE", null);
    }
}
```

```

list.addCommand(exitCommand);
list.setCommandListener(this);

//Definiuję wygląd pierwszego formularza z ChoiceGroup typu EXCLUSIVE
chGroup1.append("opcja 1",null);
chGroup1.append("opcja 2",null);
chGroup1.append("opcja 3",null);
chGroup1.append("opcja 4",null);
form1.append(chGroup1);
form1.addCommand(backCommand);
form1.setCommandListener(this);
form1.setItemStateListener(this);

//Definiuję wygląd pierwszego formularza z ChoiceGroup typu EXCLUSIVE
chGroup2.append("opcja 1",null);
chGroup2.append("opcja 2",null);
chGroup2.append("opcja 3",null);
chGroup2.append("opcja 4",null);
form2.append(chGroup2);
form2.addCommand(backCommand);
form2.setCommandListener(this);
form2.setItemStateListener(this);
}

public void startApp() {
    display.setCurrent(list);
}

public void pauseApp() {}

public void destroyApp(boolean unconditional) {}

/** Obserwator pozwalający reagować na akcje związane z komendami.
 */
public void commandAction(Command c, Displayable s) {
    if (c == List.SELECT_COMMAND) {
        if (list.getString(list.getSelectedIndex()) == "EXCLUSIVE") {
            display.setCurrent(form1);
        }
        else if (list.getString(list.getSelectedIndex()) == "MULTIPLE") {
            display.setCurrent(form2);
        }
    }
    else if (c == backCommand) {
        display.setCurrent(list);
    }
    else if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    }
}

/** Obserwator zmian w stanie obiektów elementu Form.
 * Możemy tutaj reagować na zmiany ich stanu.
 */

```

```

public void itemStateChanged(Item item) {
    if ((ChoiceGroup)item == chGroup1)
        System.out.println("Zmieniono ChoiceGroup typu EXCLUSIVE.");
    if ((ChoiceGroup)item == chGroup2)
        System.out.println("Zmieniono ChoiceGroup typu MULTIPLE.");
    System.out.println("Stan elementów:");
    for (int i=0;i<((ChoiceGroup)item).size();i++)
        System.out.println("opcja " + (i+1) + ": "
            + ((ChoiceGroup)item).isSelected(i));
}
}

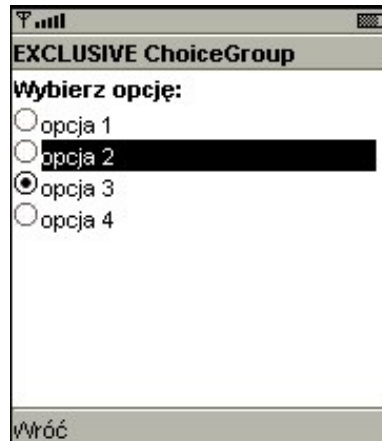
```

I efekty:

TABELA1:



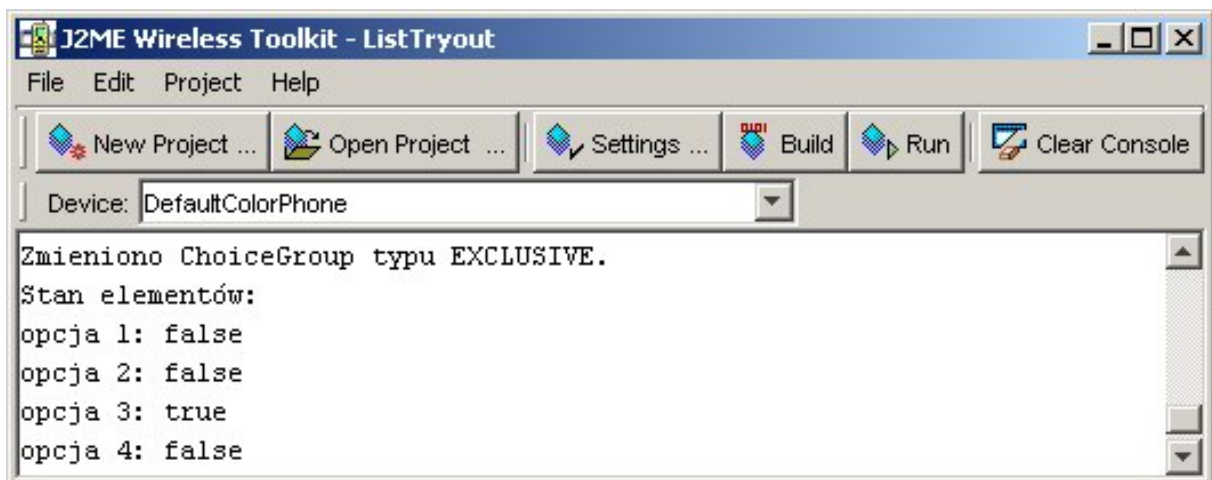
*Menu główne, czyli lista typu
IMPLICIT.*



*ChoiceGroup typu
EXCLUSIVE.*



*ChoiceGroup typu
MULTIPLE.*



Raport utworzony przy zmianie w ChoiceGroup typu EXCLUSIVE.