TotalCross Platform

Crud Application

22 /5 /2019
Iaggo Quezado
iaggoquezado@gmail.com

Summary

TotalCross Overview

Supported Platforms

- Android 2.3.3 and above (API level 10).
- iOS 5.0 and above.
- Windows XP and above.
- Linux 32 bits (we only assure that it works on Debian distribution).
- Browser as Java applet (JDK 1.1 and above). Note that in this case you must enable some permission so that you can use files and, consequently, Litebase.
- Java SE
- Windows Phone 8

Installing TotalCross

Totalcross apps are developed in Java, which means we need to configure a few information to get developing

Configuring Environment Variables

To start developing in our favorite IDE, we need to install Java JDK and, to configure it, just follow this series of steps:

Step - User Variables

- Make sure that TOTALCROSS3_HOME is pointing to your SDK (example: C:\Program Files\User\SDK)
- Make sure Path contains this path: %TOTALCROSS3_HOME%\dist\vm\win32\

- **Step - System Variables**
- Create a variable called JAVA_HOME and point to your JDK (example: C:\Program Files\Java\jdk1.8.0_91;)
- Create a variable named CLASSPATH and add these paths: %JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\lib\

dt.jar;%JAVA_HOME%\lib\htmlconverter.jar;%JAVA_HOME%jre\lib;%JAVA_HOME%\jre\lib\rt.jar;

- go to the Path variable and add this path: ;%JAVA_HOME%\bin;

Remember to restart your computer after all changes.

Importing TotalCross in Maven project

Totalcross is also in the Maven repositories and to import it is very simple. Just go to the pom.xml of your Maven project and add the code below:

```
<properties>
<totalcross.activation_key></totalcross.activation_key>
</properties>
<dependencies>
	<dependency>
		<groupId>com.totalcross</groupId>
		<artifactId>totalcross-sdk</artifactId>
		<version>4.1.0</version>
	</dependency>
</dependencies>
<repositories>
	<repository>
		<id>totalcross-repo</id>
		<name>ip-172-31-40-140-releases</name>

	<url>http://maven.totalcross.com/artifactory/repo1</url>
	</repository>
</repositories>
```

Planning

Before begin to write the project, it is necessary to planning what it should be needed, what should be include and how to do it. So, it is better to mold the project before began write it. The application that this document will show how to create a simple application, it is a Create, Read, Update and Delete (CRUD).

First, it is necessary to think which classes will be needed, in this application the CRUD will control a sql database with person's name, number and born date. it will be also necessary two windows, the first one to show the stored data in form of list and select which operation we will use, and the second window to accomplish the selected operation from the first window.

Classes

Person

The first class which will be create is Person, because it will be the model of this CRUD.

```
public class Person {
    private int id;
    private String name;
    private String born;
    private String number;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getBorn() { return born; }
    public void setBorn(String date) { this.born = date; }
    public String getNumber() { return number; }
    public void setNumber(String number) { this.number = number; }
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }


}
```

This class will have id as Index, Name, Date, and cell phone number.

PersonDAO

The next step will be creating PersonDAO in this class is where the SQL database is created and have the methods to control it. Inside of PersonDAO the first step in this class would be create his database with his constructor.

```
public PersonDAO() throws SQLException {
    con = DriverManager.getConnection( url: "jdbc:sqlite:" + Convert.appendPath(Settings.appPath, "crud.db"));

    PreparedStatement pSmt = con.prepareStatement( s: "create table if not exists person(id int primary key," +
            " name varchar, born varchar, number varchar)");
    pSmt.executeUpdate();
}
```

Inside of your constructor, verify if there is already a database created if not, create a table with id as primary key, name, born and number as varchar. The class also need a method to insert, update, delete, return everything and a next id to index the objects.

```
public void insert(Person p) throws SQLException {
    PreparedStatement pSmtInsert = con.prepareStatement( s: "insert into person values(?, ?, ?, ?)");
    pSmtInsert.clearParameters();
    p.setId(proximoId());
    pSmtInsert.setInt( i: 1, p.getId());
    pSmtInsert.setString( i: 2, p.getName());
    pSmtInsert.setString( i: 3, p.getBorn());
    pSmtInsert.setString( i: 4, p.getNumber());
    pSmtInsert.executeUpdate();
}
```

The insert method which will receive a person and will insert his attributes in the database.

```
public void update(Person p) throws SQLException {
    PreparedStatement pSmtUpdate = con.prepareStatement( s: "UPDATE person SET name = ?, number = ?, born = ? WHERE id = ?");
    pSmtUpdate.clearParameters();
    pSmtUpdate.setString( i: 1, p.getName());
    pSmtUpdate.setString( i: 2, p.getNumber());
    pSmtUpdate.setString( i: 3, p.getBorn());
    pSmtUpdate.setInt( i: 4, p.getId());
    pSmtUpdate.executeUpdate();

}
```

The update method which will receive a person and will update his attributes where his id is locate.

```
public void delete(Person p) throws SQLException {
    PreparedStatement pSmtDelete = con.prepareStatement( s: "DELETE FROM person WHERE id = ?");
    pSmtDelete.clearParameters();
    pSmtDelete.setInt( i: 1, p.getId());

    pSmtDelete.executeUpdate();
}
```

The delete method which will receive a person and will delete from database where his id is locate.

```
public String[][] all() throws SQLException {

    PreparedStatement pSmtSelect = con.prepareStatement( s: "SELECT * FROM person");
    ResultSet rs        =  pSmtSelect.executeQuery();

    int amount = 0;
    while(rs.next()){
        amount += 1;
    }

    String[][] retorno  = new String[amount][4];

    rs  =  pSmtSelect.executeQuery();
    for (int i = 0; i < amount; i++) {
        rs.next();
        for (int j = 0; j < 4; j++) {
```

```
            retorno[i][j] = rs.getString( j+1);
        }

    }
    rs.close();
    return retorno;
}
```

This method all is used to return every person inside the database we will need this to put in the grid which will create latter.

```
private int nextId() throws SQLException{
    int retorno = 1;

    PreparedStatement pSmtId = con.prepareStatement( "SELECT max(id) as  vId from person");
    ResultSet rs = pSmtId.executeQuery();
    while(rs.next()){
        retorno = rs.getInt( "vId") + 1;
    }
    rs.close();
    return retorno;
}
```

This is the last method of PersonDAO this will define the next ID of the person which will be include in the database.

PersonForm

The next step, is to create a new class called PersonForm which will extend to container. Inside of the class it is needed to create a Button id to be able to use in the button in a case, declaring a grid to show the container and also instance a person to use to operate in the program.

```
public class PersonForm extends Container {
    private static final int NEW_BTN_ID     = 100;
    private static final int UPDATE_BTN_ID  = 101;
    private static final int DELETE_BTN_ID      = 102;
    private static PersonDAO dao            = null;
    private Grid pessoaGrid;

    public PersonForm() throws SQLException {
        dao = new PersonDAO();
    }
```

Finishing writing that down, as written in the planning section this class will be the first window. It will be needed three buttons which will be used to create, to update and to delete and also a grid to show all the data.

```
public void initUI(){

    Button newButton   = new Button( text: "New");
    Button updateButton = new Button( text: "Update");
    Button deleteButton   = new Button( text: "Delete");
    // Create new buttons
    newButton.appId  = NEW_BTN_ID;
    updateButton.appId = UPDATE_BTN_ID;
    deleteButton.appId   = DELETE_BTN_ID;
    // Give the buttons ID
    add(newButton, CENTER,  y: TOP + 2+ UnitsConverter.toPixels( value: DP + 8),  w: PARENTSIZE + 95, PREFERRED);
    add(updateButton, CENTER,  y: AFTER + UnitsConverter.toPixels( value: DP + 8),  w: PARENTSIZE + 95, PREFERRED);
    add(deleteButton, CENTER,  y: AFTER + UnitsConverter.toPixels( value: DP + 8),  w: PARENTSIZE + 95, PREFERRED);
    // Declare where the buttons will be locate
    newButton.setBackForeColors(Color.getRGB( rrggbb: "18d25a"), Color.WHITE);
    updateButton.setBackForeColors(Color.getRGB( rrggbb: "1878d1"), Color.WHITE);
    deleteButton.setBackForeColors(Color.getRGB( rrggbb: "d34419"), Color.WHITE);
    // Set Buttons colors
    pessoaGrid = new Grid(new String[]{"Id", "Name","Date","Number"}, new int[]{-4, -30,-25,-40}, new int[]
            {CENTER, LEFT,LEFT,LEFT},  checkEnabled: false);
    // Create a grid with lacuna names, location, how the information will be centralized, disable check box
    add(pessoaGrid, LEFT,  y: AFTER + 2,  w: FILL - 2,  h: FILL - 2);
    try {
        pessoaGrid.setItems(dao.all());
        // this will fill the grid
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Now it is needed a method to control events from the button being pressed where the new button and the update button will open a new window and the delete button will delete the select person in the grid.

```
public void onEvent(Event event){
    // Event method
    switch (event.type) {
        case ControlEvent.PRESSED:
            // what will happen when the button pressed
            switch (((Control) event.target).appId) {
                // what will happen with which button pressed using his id
                case NEW_BTN_ID:
                    new PersonEditForm(dao, new Person(),  editMode: 1).popupNonBlocking();
                    // Edit mode 1 insert a new person
                    break;
                case UPDATE_BTN_ID:{
                    if(SelectPerson()){
                        new PersonEditForm(dao, getSelectPerson(),  editMode: 2).popupNonBlocking();
                        // Edit mode 2 update a new person
                    }
                    break;
                }
```

```
            case DELETE_BTN_ID:{
                if(SelectPerson()){
                    try {
                        dao.delete(getSelectPerson());
                        // Delete Button case will delete the selected person
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    try {
                        pessoaGrid.setItems(dao.all());
                        // This will refresh the grid to show the updated grid
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    pessoaGrid.setSelectedIndex(-1);
                }
                break;
            }
        }
        break;
```

It is also necessary a method to return to this window when the second window would be closed.

```
        case ControlEvent.WINDOW_CLOSED:{

            if(event.target instanceof PersonEditForm){
                // Event Window closed this is when the second window is closed
                try {
                    pessoaGrid.setItems(dao.all());
                    // refresh the grid
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                pessoaGrid.setSelectedIndex(-1);
            }
        }
        break;
    }
}
```

In the end of this class it needed to write the method to select a person and to get a selected person.

```
private boolean SelectPerson(){
    boolean SelectPerson = pessoaGrid.getSelectedIndex() >= 0;

    if(!SelectPerson){
        new MessageBox( title: "Atenttion",  msg: "Nobody was select").popup();;
        // this will show if pressed update button but no one was select
    }
    return SelectPerson;
}
```

```
private Person getSelectPerson(){
    Person p = new Person();
    String[] colunas = pessoaGrid.getSelectedItem();
    // this will return the select person from the grid
    try{
        p.setId(Convert.toInt(colunas[0]));
        p.setName(colunas[1]);
        p.setNumber(colunas[3]);
        p.setBorn(colunas[2]);
        // this will set the EDIT with value from the select person
    }catch(InvalidNumberException e){
        e.printStackTrace();
    }
    return p;
}
```

PersonEditForm

Now this class will be the second window where the user will be editing person attributes. It needs to have it extend to window and also need to declare variable such as button id, edits to be able to write and a PersonDAO.

```
public class PersonEditForm extends Window {

    private static final int SALVAR_BTN_ID      = 100;
    private static final int CANCELAR_BTN_ID    = 101;
    // buttons ids
    private final PersonDAO dao;
    private Edit nameEdit;
    private Edit numberEdit;
    private Edit dateEdit;
    // instancing Edits
```

Now the constructor needs to receive the PersonDAO, Person and the editMode to fulfill the operation. It needs also a method to call this new window from the first window.

```
PersonEditForm(PersonDAO dao, Person person, int editMode){
    this.dao = dao;
    this.appObj = person;
    this.appId = editMode;

}
protected void postPopup(){
    super.postPopup();
    initUI();
    // method to call this window
}
```

After that, the next step will be the second window interface, there will be two buttons, labels, edits.

```
public void initUI(){
    super.initUI();
    Label lb4 = new Label( text: "Person's Data",CENTER);
    // Window label
    Button saveButton   = new Button( text: "Save");
    Button cancelButton = new Button( text: "Cancel");
    // Button text
    Edit   idEdit         = new Edit( mask: "9999");
    nameEdit              = new Edit();
    numberEdit           = new Edit( mask: "(99)9999-99999");
    // Edit mask to make the input his format
    numberEdit.setMode(Edit.NORMAL,  maskedEdit: true);
    numberEdit.setValidChars(Edit.numbersSet);
    //Edit set to only make the number mask  editable
    numberEdit.setKeyboard(Edit.KBD_NUMERIC);
    // Edit keyboard set to only make input numbers
    dateEdit            = new Edit( mask: "99/99/99");
    dateEdit.setMode(Edit.NORMAL,  maskedEdit: true);
    dateEdit.setValidChars(Edit.numbersSet);
    dateEdit.setKeyboard(Edit.KBD_NUMERIC);
    lb4.setBackForeColors( back: 0x2979ff,  fore: 0xEEEEEE);
    // Change label window color
    saveButton.appId = SALVAR_BTN_ID;
    cancelButton.appId = CANCELAR_BTN_ID;
    // Button receiving his ID
    idEdit.setEditable(false);
    // Set idEdit to not receive input
    idEdit.setText(Convert.toString(((Person) appObj).getId()));
    // Edit only works with String so it need to transform
    nameEdit.setText(((Person) appObj).getName());
    numberEdit.setText(((Person) appObj).getNumber());
    dateEdit.setText(((Person) appObj).getBorn());
```

Now it is necessary to set where these components will be located in the window.

```
    add(cancelButton,  x: RIGHT - 2,  y: TOP + 2,  w: PREFERRED + 2, PREFERRED);
    add(saveButton,  x: BEFORE - 2, SAME, SAME, SAME);
    add(new Label( text: "Name:"),  x: LEFT+145,  y: AFTER + 7);
    add(nameEdit, CENTER, AFTER ,  w: SCREENSIZE + 85, PREFERRED);
    add(new Label( text: "Number:"),  x: LEFT+145,  y: AFTER + 5);
    add(numberEdit, CENTER, AFTER ,  w: SCREENSIZE + 85, PREFERRED);
    add(new Label( text: "Date:"),  x: LEFT+145,  y: AFTER + 5);
    add(dateEdit, CENTER, AFTER ,  w: SCREENSIZE + 85, PREFERRED);
    add(lb4,CENTER, TOP,  w: SCREENSIZE + 100, PREFERRED);
    add(cancelButton, CENTER,  y: BOTTOM - 10,  w: PARENTSIZE + 95, PREFERRED);
    add(saveButton, CENTER,  y: BOTTOM - 230,  w: PARENTSIZE + 95, PREFERRED);
    // Setting buttons, Labels and Edits locations size, Weight and etc.
    saveButton.setBackForeColors(Color.getRGB( rrggbb: "1878d1"), Color.WHITE);
    cancelButton.setBackForeColors(Color.getRGB( rrggbb: "d34419"), Color.WHITE);
    // Settings buttons colors

}
```

The class is almost finished, the next method will be events what will happen when the buttons are pressed.

```
public void onEvent(Event event){
    if (event.type == ControlEvent.PRESSED) {
        // event type controller button pressed
        switch (((Control) event.target).appId) {
            case SALVAR_BTN_ID: {
                // save button id pressed
                if (NameExist()) {
                    // check if name exist
                    if (NumberCorrect()) {
                        // check if any number is missing
                        if (DateCorrect()) {
                            // check if any number date is missing
                            switch (appId) {
                                case 1: {
                                    // Edit mode 1 New
                                    try {
                                        dao.insert((Person) appObj);
                                        // save new person
                                    } catch (SQLException e) {
                                        e.printStackTrace();
                                    }
                                    break;
                                }
                                case 2: {
                                    // Edit mode 2 update
                                    try {
                                        dao.update((Person) appObj);
                                        // Update new person
                                    } catch (SQLException e) {
                                        e.printStackTrace();
                                    }
                                    break;
                                }
                            }
                            unpop();
                            // Back to first window
                        }
                    }
                }
                break;
            }
            case CANCELAR_BTN_ID: {
                // Button cancel pressed
                unpop();
                // Back to first window
                break;
            }
        }
    }
```

To end this class, it needs to written method to check if the name exists, if the number is complete and if the data is not missing any number.

```
private boolean NameExist(){
    // check if name exist
    if(!(this.nameEdit.getLength() > 0)){
        new MessageBox( title: "Error",  msg: "Name invalid").popup();
        // Make a pop up message telling name invalid
        return false;
        // Dont make the program continue and return to input the data
    }

    return true;
}
private boolean NumberCorrect(){
    if(!(this.numberEdit.getLength()> 10)){
        // Check if there is any number missing
        new MessageBox( title: "Error",  msg: "Number invalid").popup();
        // Number invalid pop up a message telling
        return false;
        // Dont make the program continue and return to input the data
    }
     return true;
}
private boolean DateCorrect(){
    if(!(this.dateEdit.getLength()> 5)){
        // Check if there is any number missing
        new MessageBox( title: "Error",  msg: "Date invalid").popup();
        return false;
        // Dont make the program continue and return to input the data
    }
    // Input Edit Text in the person to make a new or update.
    ((Person) appObj).setName(this.nameEdit.getText());
    ((Person) appObj).setNumber(this.numberEdit.getText());
    ((Person) appObj).setBorn(this.dateEdit.getText());
    return true;
}
```

Main and MainApplication

Now all what is left is Main class and the main application class where the early one will be the instance of our application and the latter one will be where our application will run. In some case, like this one it is better to set uistyle material for better design.

```
public class Main extends MainWindow {

    public Main() throws SQLException {
        super( title: "Registry", VERTICAL_GRADIENT);
        gradientTitleStartColor = 0x2979ff;
        gradientTitleEndColor = 0x2979ff;
        setUIStyle(Settings.Material);
        // UI Style
        Settings.uiAdjustmentsBasedOnFontHeight = true;
        setBackColor(Color.brighter(Color.BRIGHT, Color.LESS_STEP));
        // Program Background color
        swap(new PersonForm());
    }
}
```

```java
public class MainApplication {
    public static void main(String[] args) {

        TotalCrossApplication.run(Main.class);
        // Make the application run
    }
}
```

Now its possible to run the project from the MainApplication class.

GitHub Project

The project it is available to download from GitHub from my depository:

https://github.com/laggoq/TotalcrossApp

Useful Links

Information which was crucial to create this project:

https://totalcross.com/documentation/en/api/en/components/TotalcrossOverview.php

https://totalcross.gitbook.io/playbook/components/edit

https://totalcross.gitbook.io/playbook/components/label

https://totalcross.gitbook.io/playbook/components/button

https://totalcross.gitbook.io/playbook/apis/main-window

https://totalcross.gitbook.io/playbook/apis/window

https://totalcross.gitbook.io/playbook/apis/container

https://totalcross.com/documentation/en/api/en/components/Events.php

https://totalcross.com/documentation/en/api/en/components/UIStyles.php