Menus & Dialogs

Introduction

- The android platform supports an extensive range of windows and menus.
- The dialogs are asynchronous. This is different from the known synchronous dialogs on windows.
- The dialogs in android are managed. We can reuse them. There is no need to create a new one each time we need a dialog.

The `android.view.Menu` Interface

- Each and every activity in the android platform is associated with a specific object of this type.
- Each object of this type can contain menu items and sub menus.

The `android.view.MenuItem` Interface

- Each menu item is represented by an object of this type.

The `android.view.SubMenu` Interface

- Each sub-menu is represented by an object of this type.

Menu Items Group

- We can group menu items together into one group. Each group is assigned with a group ID. Multiple menu items carrying the same group ID are considered to be part of the same group.

Menu Items Attributes

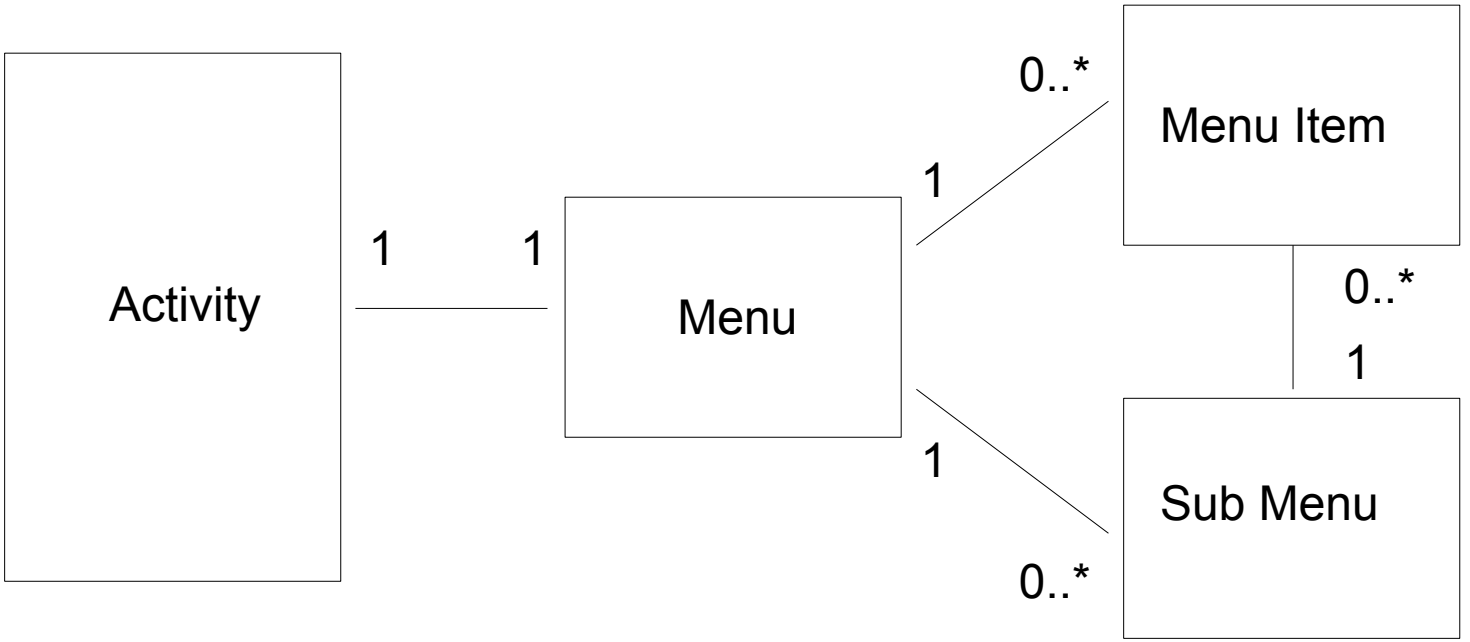
- Each menu item has the following three attributes: name, item ID and order ID.
- The menu items are shown sorted in accordance with their order ID. Those with the smaller order ID are shown on top of the others.

Creating Menu

- Each activity is associated with a specific menu. We don't need to create a menu from scratch. A menu object already exists.
- Overriding the `onCreateOptionMenu(Menu menu)` method we can populate the associated menu with menu items Returning true will make our menu visible.

Creating Menu

- Each activity is associated with a specific menu. We don't need to create a menu from scratch. A menu already exists.



Creating Menu

- Overriding the `onCreateOptionMenu(Menu menu)` method we can populate the associated menu with menu items Returning true will make our menu visible.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    //populate menu items
    ...
    ...
    return true;
}
```

Creating Menu

```
package com.abelski.samples;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class MenuDemoActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
```

Creating Menu

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // calling base class overdding version to ensure that
        // system menu are included
        super.onCreateOptionsMenu(menu);
        menu.add(0,1,0,"Save"); //groupID,itemID,order,title
        menu.add(0,2,1,"Open");
        menu.add(0,3,2,"Close");
        // the return type must be true in order to see the menu
        return true;
    }
}
```

Creating Menu

Menu Items Groups

- The menu ID is the value sent to the callback function when the menu is selected.
- Adding menu items with a different group ID we get separated menu items groups.

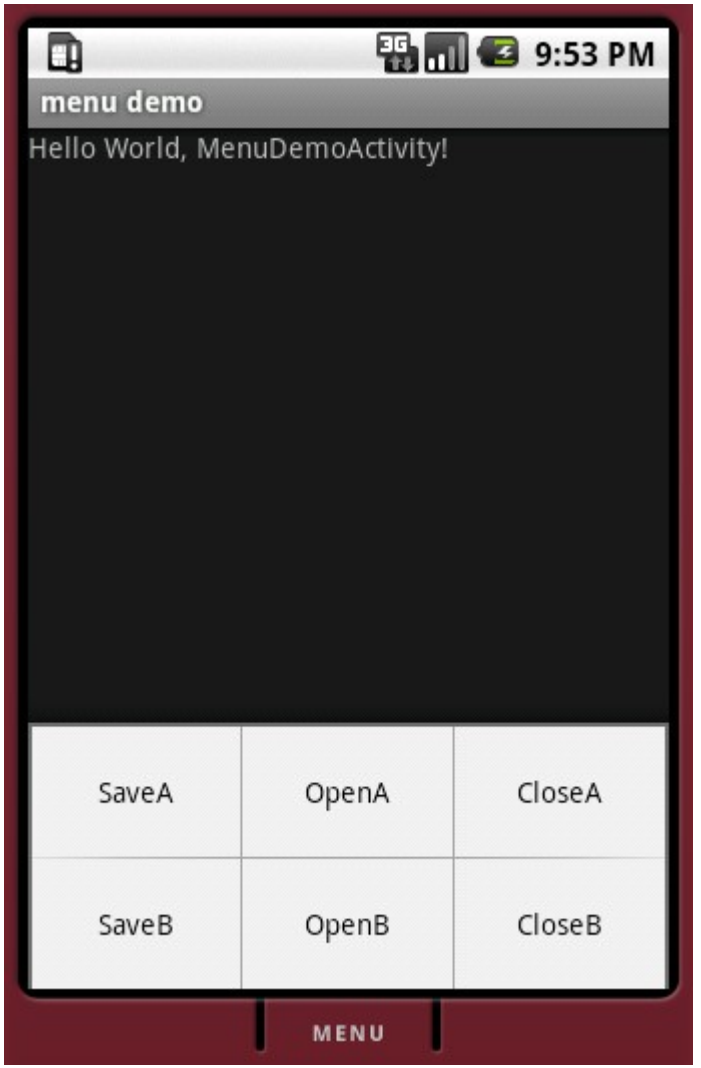
Menu Items Groups

```
package com.abelski.samples;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class MenuDemoActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
```

Menu Items Groups

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        int group_a = 1;
        int group_b = 2;
        super.onCreateOptionsMenu(menu);
        menu.add(group_a,1,0,"SaveA");
        menu.add(group_a,2,1,"OpenA");
        menu.add(group_a,3,2,"CloseA");
        menu.add(group_b,4,3,"SaveB");
        menu.add(group_b,5,4,"OpenB");
        menu.add(group_b,6,5,"CloseB");
        return true;
    }
}
```

Menu Items Groups



Menu Items Groups

- Once using groups we can call the following methods on specific groups.

```
removeGroup(id)
```

Calling this method removes all menu items that belong to the group its ID was passed over.

```
setGroupCheckable(id, checkable, exclusive)
```

Calling this method turns all menu items that belong to the group its ID was passed over into checkable. If exclusive is true then only one menu item can be checked at a time.

Menu Items Groups

`setGroupEnabled(id,boolean enabled)`

Calling this method we can enable and disable all menu items that belong to the group its ID was passed over.

`setGroupVisible(id,visible)`

Calling this method we can turn on and off the visibility of all menu items that belong to the group its ID was passed over.

Menu Items Events Handling

- We can handle the menu items events in three ways.
- One option is to override the available callback function. Its name is `onOptionsItemSelected`.
- Another option is to define a listener for our menu. Listener should be an object instantiated from a class that implements the `OnMenuItemClickListener` interface.
- The third option is to use intents.

Overriding Callback Function

- This is a call back function we can override in our class. It will be called each time a menu item is clicked (selected).

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    switch(item.getItemId())
    {
        .....
    }

    ...
    return true;

    ...
    return super.onOptionsItemSelected(item);
}
```

Define Listener

- The listener we define should implement the `OnMenuClickListener` interface.

```
public class MyListener implements OnMenuItemClickListener
{
    ...
    @override
    boolean onMenuItemClick(MenuItem item)
    {
        ...
        return true;
    }
}

MyListener listener = new MyListener(...);
menuItem.setOnMenuItemClickListener(listenr);
...
```

Define Listener

- If the `onMenuItemClick` returns true then no other callback method will be called.
- The listener takes precedence over the call back method.

Using Intents

- Calling the `setIntent` method on a menu item we can associate it with an intent.
- When neither a listener or a callback method respond to a menu item selection, the `startActivity(Intent)` method will be called with the intent associated with the clicked menu item.

Expanded Menu

- When the application has more menu items than what it can display on the main menu then the android platform adds the 'More' menu item. Pressing 'More' we shall get an expanded menu with the additional menu items.
- Expanded menu cannot show icon menus.

Icon Menus

- We can create menu items that include icons instead of the texts or together with them.
- In order to get an icon menu item we just need to call the `setIcon()` method on our menu item object.

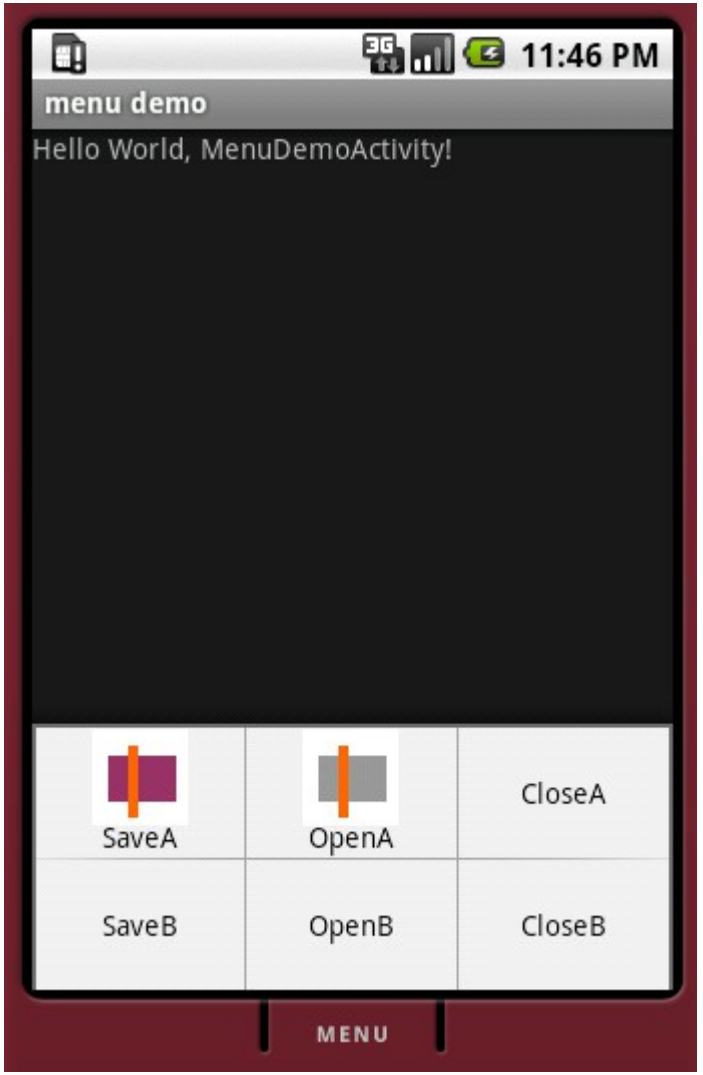
Icon Menus

```
package com.abelski.samples;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MenuDemoActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
```

Icon Menus

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        int group_a = 1;
        int group_b = 2;
        super.onCreateOptionsMenu(menu);
        MenuItem itemA = menu.add(group_a,1,0,"SaveA");
        itemA.setIcon(R.drawable.icon_a);
        MenuItem itemB = menu.add(group_a,2,1,"OpenA");
        itemB.setIcon(R.drawable.icon_b);
        menu.add(group_a,3,2,"CloseA");
        menu.add(group_b,4,3,"SaveB");
        menu.add(group_b,5,4,"OpenB");
        menu.add(group_b,6,5,"CloseB");
        return true;
    }
}
```

Icon Menus



Sub Menus

- We can add sub menus to our menu. We can add menu items to our sub menu. It is impossible to add sub menus to a given sub menu.
- The `SubMenu` interface extends `Menu`.

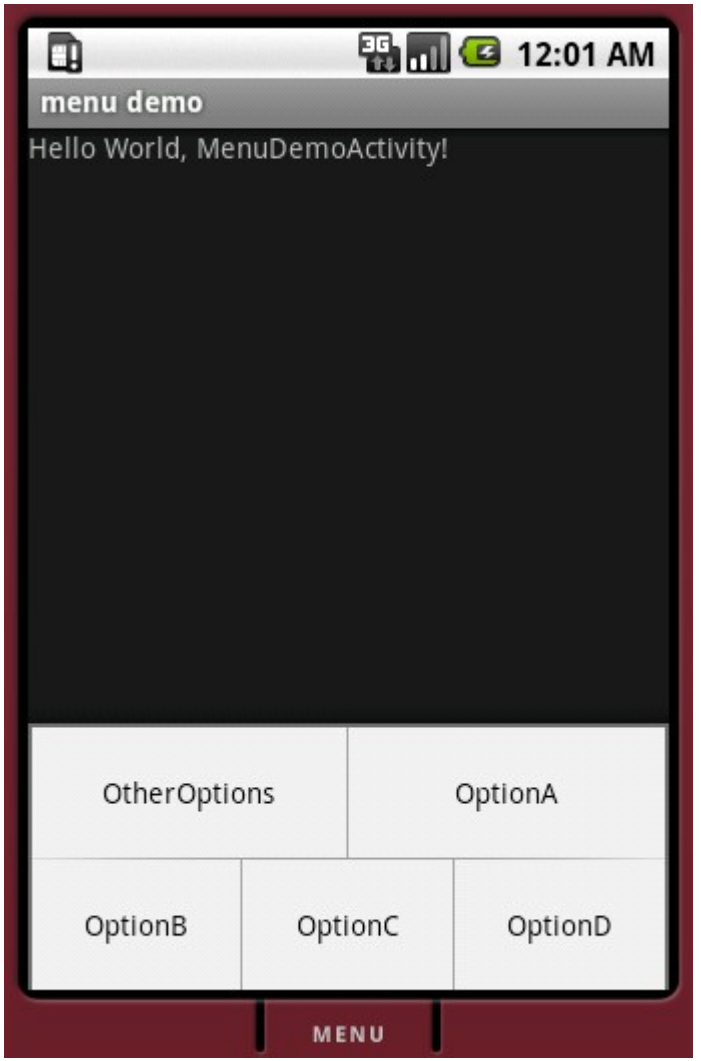
Sub Menus

```
package com.abelski.samples;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;

public class MenuDemoActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
```

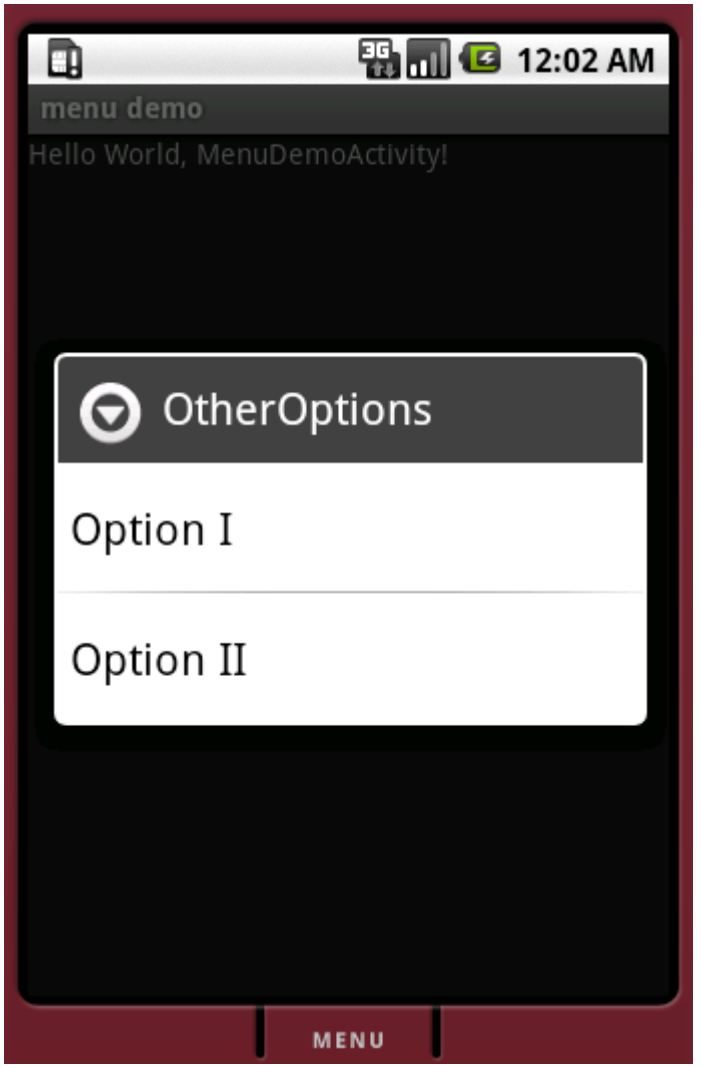
Sub Menus

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        super.onCreateOptionsMenu(menu);
        menu.add(0,1,1,"OptionA");
        menu.add(0,2,2,"OptionB");
        menu.add(0,3,3,"OptionC");
        menu.add(0,4,4,"OptionD");
        SubMenu sub = menu.addSubMenu("OtherOptions");
        sub.add(0,5,5,"Option I");
        sub.add(0,6,6,"Option II");
        return true;
    }
}
```

Sub Menus



Sub Menus



System Menus

- Calling the overridden version of `onCreateOptionsMenu` method we allow the parent class to add the system menus.

Context Menus

- Context menus are the small menus we get when right click our mouse.
- Getting a context menu when using the android might happen in various ways. It depends on the specific handset model. In most cases, a long press will get us the context menu.
- The `ContextMenu` class describes a context menu.

Context Menus

- Whereas a context menu is owned by a specific view, a common options menu is owned by a specific activity. Therefore, an activity can have one options menu only and unlimited number of context menus. Each context menu is associated with a specific view.
- Populating a context menu is done within the scope of the activity class. Overriding the `onCreateContextMenu()` we can add the menu items to our context menu.

Context Menus

- Whereas the `onCreateOptionsMenu()` method is automatically called the `onCreateContextMenu()` is not.
- Not every view should have a context menu. Just the ones we want them to have should have one.
- When we want a specific view to have a context menu we should register that view with its activity specifically for the purpose of having a context menu.

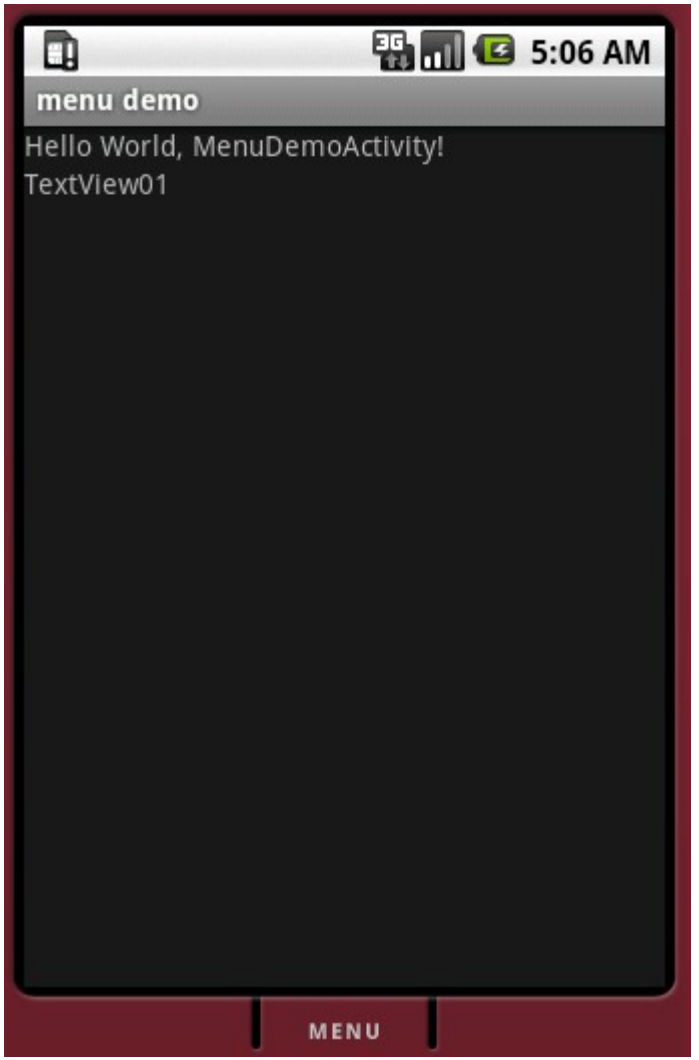
Context Menus

- We register a view with an activity for that purpose by calling the `activity.registerForContextMenu()` method.
- In order to add a context menu associated with a specific view we first need to register the view by calling the `registerForContextMenu()` method. Once that method was called the `onCreateContextMenu()` callback function will be called. Within this callback method the context menu will be populated with its menu items.

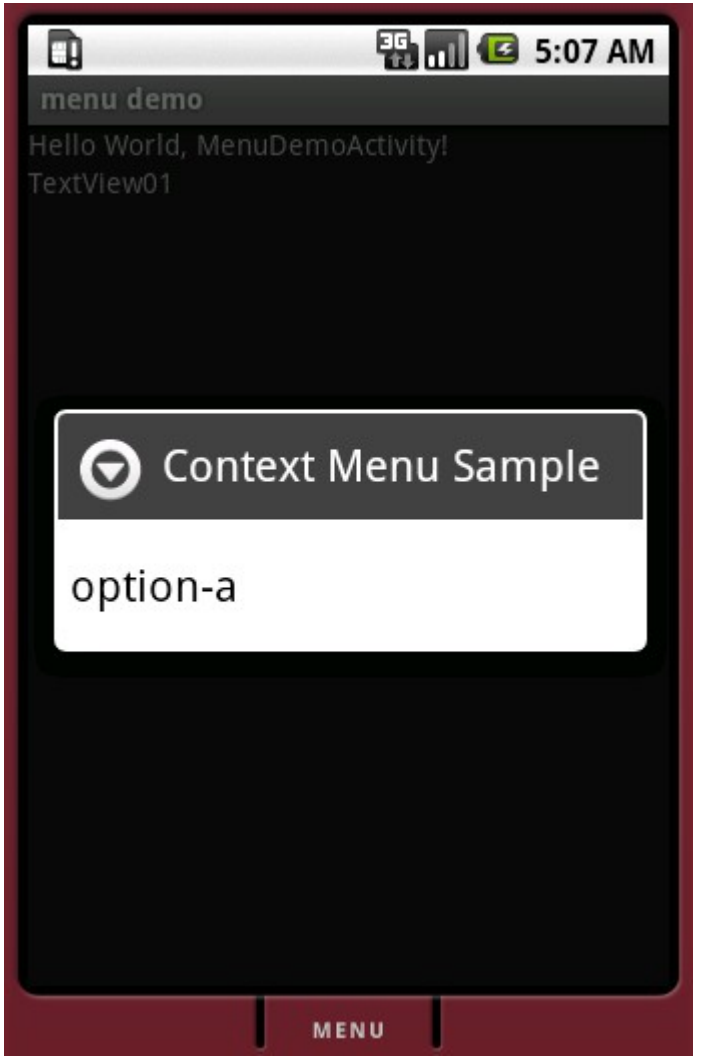
Context Menus

```
package com.abelski.samples;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.view.ContextMenu;
import android.view.View;

public class MenuDemoActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = (TextView) this.findViewById(R.id.TextView01);
        registerForContextMenu(tv);
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo)
    {
        menu.setHeaderTitle("Context Menu Sample");
        menu.add(2, 4, 1, "option-a");
    }
}
```

Context Menus



Context Menus



Context Menus

- Handling the content menu related events is done by overwriting `onContextItemSelected()` call back method.

```
@Override
public boolean onContextItemSelected(MenuItem item)
{
    if (item.itemId() = _____)
    {
        ...
        return true;
    }
    ...
}
```

The `onPrepareOptionsMenu()` Method

- This method is get called each time the menu is invoked.
- We can use this method for changing our menu dynamically, such as disabling parts of our menu or removing some of the menu items.

Menus Through XML

- We can interact with our menus via an XML document we should place within the '`res/menu`' sub folder.
- The XML document that describes our menus isn't created automatically. It is up to us to choose whether to have one.
- Each one of the menus and the menu items will get a resource ID number we can use. As with the user interface controls we will get those Ids as fields in `R.java`.

Alert Dialog

- The first step in order to get an alert dialog is to construct an `AlertDialog.Builder` object.
- Once we have an `AlertBuilder` object we can customize it by calling various methods on it (e.g. `setTitle()`, `setPositiveButton()` etc.).
- After customizing the `AlertDialog.Builder` object we can call the `create()` method and get an `AlertDialog` object.
- Calling `show()` on the `AlertDialog` object will display it.

Alert Dialog

- Using this builder class we can construct an alert dialog window through which the user will be able to perform any of the following:
  1. Read a textual message and respond with 'yes' or 'no'.
  2. Select an item from a list.
  3. Select multiple items from a list.
  4. View the progress of our application.
  5. Choose an option from a set of options.

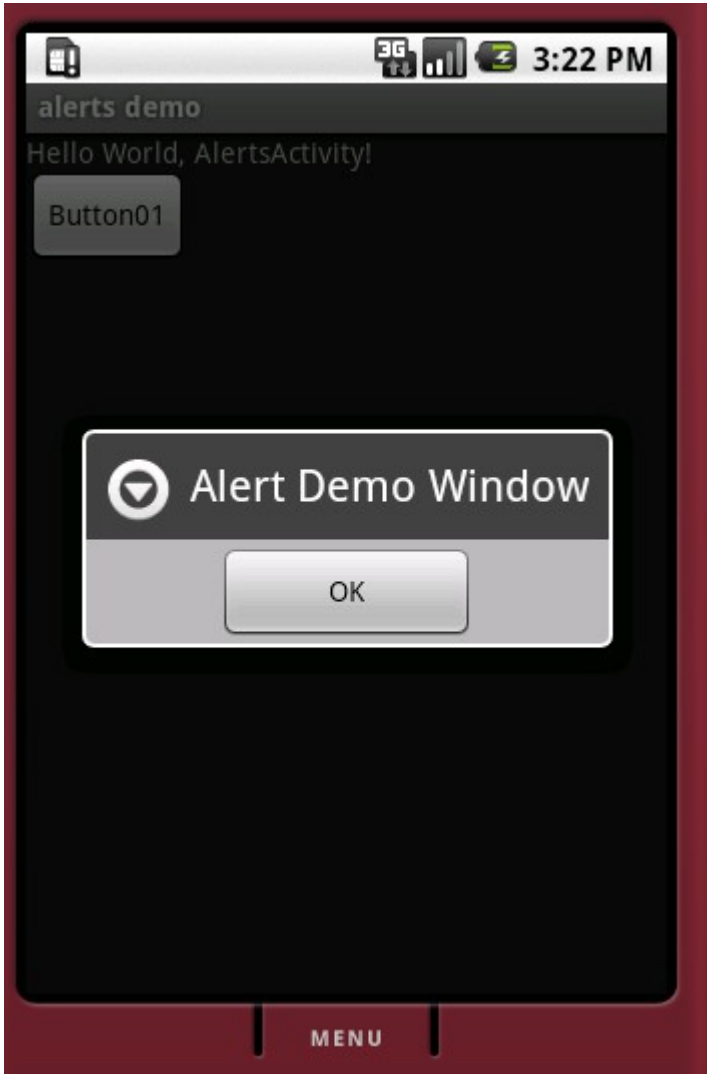
Alert Dialog

```
package com.abelski;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class AlertsActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final Button bt = (Button)this.findViewById(R.id.Button01);
```

Alert Dialog

```
        bt.setOnClickListener(new OnClickListener()
        {
            public void onClick(View view)
            {
                if(view==bt)
                {
                    AlertDialog.Builder builder =
                        new AlertDialog.Builder(AlertsActivity.this);
                    builder.setTitle("Alert Demo Window");

                    DialogInterface.OnClickListener listener =
                        new DialogInterface.OnClickListener()
                                {public void onClick(DialogInterface dialog,
                                int which) {  }};
                    builder.setPositiveButton("OK", listener);
                    AlertDialog ad = builder.create();
                    ad.show();
                }

            }
        }
    }
}
```

Alert Dialog



Prompt Dialog

- Based on the same technique described for creating an alert dialog we can a prompt dialog, a dialog that returns back to the program a string. That string can be one of those entered by the user or another meaningful string.

Prompt Dialog

- The first step would be selecting the layout view we want our prompt dialog to use.
- Once we create the XML document that describes the user interface layout we want to have we can move forward and get a View object based on that XML.
- Setting the View object in the `Builder` we use, setting the other user interface controls we want our prompt dialog to have and we are set.

Prompt Dialog

- We can now call the `show()` method in order to display our prompt dialog.

Prompt Dialog

```
package com.abelski;


import android.app.Activity;
import android.app.AlertDialog;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.EditText;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;

public class PromptDialogProjectActivity extends Activity
{
    private String dialogReplay;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LayoutInflater li = LayoutInflater.from(this);
        final View view = li.inflate(R.layout.promptdialoglayout, null);
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

Prompt Dialog

```
        builder.setTitle("Prompt Dialog");
        builder.setView(view);
        OnClickListener listener = new OnClickListener()
        {
            public void onClick(DialogInterface di, int id)
            {
                if (id == DialogInterface.BUTTON_POSITIVE)
                {   //ok button was pressed
                    EditText et = (EditText)
                        view.findViewById(R.id.prompt_for_text);
                    dialogReplay = et.getText().toString();
                }
                else
                {   //cancel button was pressed
                    dialogReplay = null;
                }
            }
        };
        builder.setPositiveButton("OK", listener);
        builder.setNegativeButton("Cancel", listener);
        AlertDialog ad = builder.create();
        ad.show();
    }
}
```

Prompt Dialog

Prompt Dialog

- The displayed dialog is an asynchronous process. Once the dialog is shown the main thread returns and continues in its execution.
- Nevertheless, the dialog is modal. Mouse clicks apply to the dialog only.
- We cannot have a simple dialog where we ask for response and wait for it before moving forward with the execution.

Prompt Dialog

- One of the ways to overcome this behavior is to have the activity implement a callback method that will be called when the prompt dialog is closed.

  The last code sample shows a possible way for implementing a callback method that will be called when the prompt dialog is closed.