Name : Mujahid Ullah

Subject: Android Development

Submitted To Sir Junaid

Date : 26-Dec-20

Assignment No 3

1.Array adapter:

The Adapter acts as a bridge between the UI Component and the Data Source. It converts data from the data sources into view items that can be displayed into the UI Component. Data Source can be Arrays, HashMap, Database, etc. and UI Components can be ListView, GridView, Spinner, etc. ArrayAdapter is the most commonly used adapter in android. When you have a list of single type items which are stored in an array you can use ArrayAdapter. Likewise, if you have a list of phone numbers, names, or cities. ArrayAdapter has a layout with a single TextView. If you want to have a more complex layout instead of ArrayAdapter use CustomArrayAdapter. The basic syntax for ArrayAdapter is given as:

public ArrayAdapter(Context **context**, int resource, int **textViewResourceId**, T[] **objects**)

context: It is used to pass the reference of the current class. Here 'this' keyword is used to pass the current class reference. Instead of 'this' we could also use the getApplicationContext() method which is used for the Activity and the getApplication() method which is used for Fragments.

Resource :The resource ID for the layout file containing a layout to use when instantiating views.It is used to set the layout file(.xml files) for the list items.

textViewResourceId: The id of the TextView within the layout resource to be populated. It is used to set the TextView where you want to display the text data.

Objects: The objects to represent in the ListView. This value cannot be null. These are the array object which is used to set the array element into the TextView.

String courseList[] = {"C-Programming", "Data Structure", "Database", "Python","Java", "Operating System","Compiler Design", "Android Development"};

Example:

```
    // array objects
String courseList[] = {"C-Programming", "Data Structure", "Database", "Python","Java", "Operating System","Compiler Design", "Android Development"};

//Array Adapter inside Oncreate Method
```

```
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(this,
            R.layout.item_view, R.id.itemTextView, courseList);
      simpleListView.setAdapter(arrayAdapter);
```

2. context,

What is Context in Android?

Android apps are popular for a long time and it is evolving to a greater level as user's expectations are that they need to view the data that they want in an easier smoother view. Hence, the android developers must know the important terminologies before developing the app. In Android Programming we generally come across a word context. So what exactly is this context and why is it so important? To answer this question lets first see what the literal meaning of context is:

The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood.Looking at this definition we come across two things:

- The context tells us about the surrounding information.
- It is very important to understand the environment which we want to understand.

Similarly when we talk about the Android Programming context can be understood as something which gives us the context of the current state of our application. We can break the context and its use into three major points:

- It allows us to access resources.
- It allows us to interact with other Android components by sending messages.
- It gives you information about your app environment.

In the official Android documentation, context is defined as:

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

Understanding Context by a Real World Example

Let's a person visit a hotel. He needs breakfast, lunch, and dinner at a suitable time. Except for these things there are also many other things, he wants to do during the time of stay. So how does he get these things? He will ask the room-service person to bring these things for him. Right? So here the room-service person is the context considering you are the single activity and the hotel to be your app, finally, the breakfast, lunch & dinner have to be the resources.

How Does This Work?

1. It is the context of the current/active state of the application.Usually, the app got multiple screens like display/inquiry/add/delete screens(General requirement of a basic app). So when the user is searching for something, the context is an inquiry screen in this case.

2. It is used to get information about the activity and application.The inquiry screen’s context specifies that the user is in inquiry activity, and he/she can submit queries related to the app

3. It is used to get access to resources, databases, and shared preferences, etc.Via Rest services, API calls can be consumed in android apps. Rest Services usually hold database data and provide the output in JSON format to the android app. The context for the respective screen helps to get hold of database data and the shared data across screens

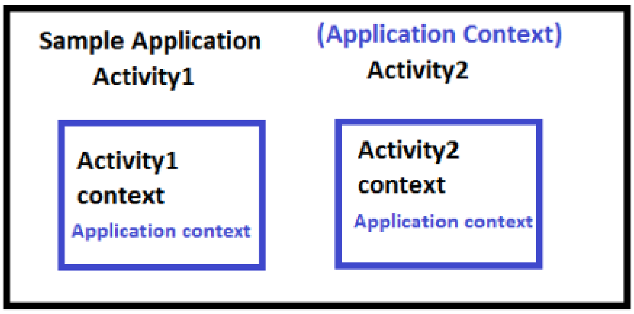
4. Both the Activity and Application classes extend the Context class.

In android, context is the main important concept and the wrong usage of it leads to memory leakage. Activity refers to an individual screen and Application refers to the whole app and both extend the context class.

There are mainly two types of context are available in Android.

- Application Context and
- Activity Context

The Overall view of the App hierarchy looks like the following:



It can be seen in the above image that in “Sample Application”, nearest Context is Application Context. In “Activity1” and “Activity2”, both Activity Context (Here it is Activity1 Context for Activity1 and Activity2 Context for Activity2) and Application Context.The nearest Context to both is their Activity Context only.

Application Context: This context is tied to the life cycle of an application. Mainly it is an instance that is a singleton and can be accessed via getApplicationContext(). Some use cases of Application Context are:

- If it is necessary to create a singleton object
- During the necessity of a library in an activity

getApplicationContext():

It is used to return the context which is linked to the Application which holds all activities running inside it. When we call a method or a constructor, we often have to pass a context and often we use “this” to pass the activity context or “getApplicationContext” to pass the application context. This method is generally used for the application level and can be used to refer to all the activities. For example, if we want to access a variable throughout the android app, one has to use it via getApplicationContext().

```
Example
// Activity 1
public class <your activity1> extends Activity {
        ........
        ........
private <yourapplicationname> globarVar;
        ........
@Override
public void onCreate(Bundle savedInstanceState) {
        .......
final GlobalExampleClass globalExampleVariable = (GlobalExampleClass)
getApplicationContext();

        // In this activity set name and email and can reuse in other activities
        globalExampleVariable.setName("getApplicationContext example");
        globalExampleVariable.setEmail("xxxxxx@gmail.com");

        .......
        }

// Activity 2
public class <your activity2> extends Activity {
        ........
        ........
private <yourapplicationname> globarVar;
        .......
@Override
public void onCreate(Bundle savedInstanceState) {
        .......
final GlobalExampleClass globalExampleVariable = (GlobalExampleClass)
getApplicationContext();

// As in activity1, name and email is set, we can retrieve it here
final String globalName = globalExampleVariable.getName();
final String globalEmail = globalExampleVariable.getEmail();

        .......
        }
```

this:
"this" argument is of a type "Context". To explain this context let's take an example to show a Toast Message using "this.

3) What is file in java:

The File class is Java's representation of a file or directory path name. Because file and directory names have different formats on different platforms, a simple string is not adequate to name them. The File class contains several methods for working with the path name, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.

- It is an abstract representation of file and directory pathnames.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

Example:

```
import java.io.*;
public class FileExample {
    public static void main(String[] args) {
        File dir=new File("/Users/sonoojaiswal/Documents");
        File files[]=dir.listFiles();
        for(File file:files){
            System.out.println(file.getName()+" Can Write: "+file.canWrite()+"
                    Is Hidden: "+file.isHidden()+" Length: "+file.length()+"
bytes");
        }
    }
}
```

Example 2:

```
import java.io.*;
public class FileExample {
    public static void main(String[] args) {
        File f=new File("/Users/sonoojaiswal/Documents");
        String filenames[]=f.list();
        for(String filename:filenames){
            System.out.println(filename);
        }
    }
}
```

4) Super Keyword in Java

The super keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword:

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.

- super() can be used to invoke immediate parent class constructor.

1) **super is used to refer immediate parent class instance variable.**

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

Example:

```
class Animal{
    String color="white";
}
class Dog extends Animal{
    String color="black";
    void printColor(){
        System.out.println(color);//prints color of Dog class
        System.out.println(super.color);//prints color of Animal class
    }
}
class TestSuper1{
    public static void main(String args[]){
        Dog d=new Dog();
        d.printColor();
    }}
```

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

2) super can be used to invoke parent class method : The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
    void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
    void eat(){System.out.println("eating bread...");}
    void bark(){System.out.println("barking...");}
    void work(){
        super.eat();
        bark();
    }
}
class TestSuper2{
    public static void main(String args[]){
        Dog d=new Dog();
        d.work();
    }}
```

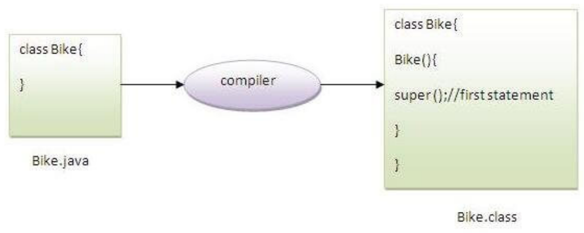
In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.To call the parent class method, we need to use super keyword.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```
class Animal{
    Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
    Dog(){
        super();
        System.out.println("dog is created");
    }
}
class TestSuper3{
    public static void main(String args[]){
        Dog d=new Dog();
    }}
```

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().


```
class Bike{
}
```

```
class Bike{
Bike(){
super();//first statement
}
}
```

As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

Another example of super keyword where super() is provided by the compiler implicitly.

super example: real use

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

```
class Person{
    int id;
    String name;
    Person(int id,String name){
        this.id=id;
        this.name=name;
    }
}
class Emp extends Person{
    float salary;
    Emp(int id,String name,float salary){
        super(id,name);//reusing parent constructor
        this.salary=salary;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}
class TestSuper5{
    public static void main(String[] args){
        Emp e1=new Emp(1,"ankit",45000f);
        e1.display();
    }}
```

5) Call Back

Implementations of this interface are passed to a CallbackHandler, allowing underlying security services the ability to interact with a calling application to retrieve specific authentication data such as usernames and passwords, or to display certain information, such as error and warning messages.

Callback implementations do not retrieve or display the information requested by underlying security services. Callback implementations simply provide the means to pass such requests to applications, and for applications, if appropriate, to return requested information back to the underlying security services.

CallbackHandlers: CallbackHandlers are implemented in an application-dependent fashion. For example, implementations for an application with a graphical user interface (GUI) may pop up windows to prompt for requested information or to display error messages. An implementation may also choose to obtain requested information from an alternate source without asking the end user.

Underlying security services make requests for different types of information by passing individual Callbacks to the CallbackHandler. The CallbackHandler implementation decides how to retrieve and display information depending on the Callbacks passed to it. For example, if the underlying service needs a username and password to authenticate a user, it uses a NameCallback and PasswordCallback. The CallbackHandler can then choose to prompt for a username and password serially, or to prompt for both in a single window. A default CallbackHandler class implementation may be specified by setting the value of the auth.login.defaultCallbackHandler security property.

If the security property is set to the fully qualified name of a CallbackHandler implementation class, then a LoginContext will load the specified CallbackHandler and pass it to the underlying LoginModules. The LoginContext only loads the default handler if it was not provided one.

All default handler implementations must provide a public zero-argument constructor.

Sending events back to an activity with callback interface

Example

//If you need to send events from fragment to activity, one of the possible solutions is to define callback interface and require that the host activity implement it.

Send callback to an activity, when fragment's button clicked

//First of all, define callback interface:

```
public interface SampleCallback {
    void onButtonClicked();
}
```

Next step is to assign this callback in fragment:

```
public final class SampleFragment extends Fragment {

    private SampleCallback callback;

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof SampleCallback) {
            callback = (SampleCallback) context;
        } else {
            throw new RuntimeException(context.toString()
                    + " must implement SampleCallback");
        }
    }

    @Override
    public void onDetach() {
        super.onDetach();
        callback = null;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        final View view = inflater.inflate(R.layout.sample, container, false);
        // Add button's click listener
        view.findViewById(R.id.actionButton).setOnClickListener(new
View.OnClickListener() {
            public void onClick(View v) {
                callback.onButtonClicked(); // Invoke callback here
            }
        });
        return view;
    }
}
And finally, implement callback in activity:

public final class SampleActivity extends Activity implements SampleCallback {
    // ... Skipped code with settings content view and presenting the fragment

    @Override
    public void onButtonClicked() {
        // Invoked when fragment's button has been clicked
    }
}
```