Session 4

Collections API And ListView

Trình bày : **Đỗ Phú Huy**
BM Tin học
Cao đẳng Công Nghệ

Google Android

Modules

- **Introduction to a collections API**
- **Introduction to a ListViews**

Module 7

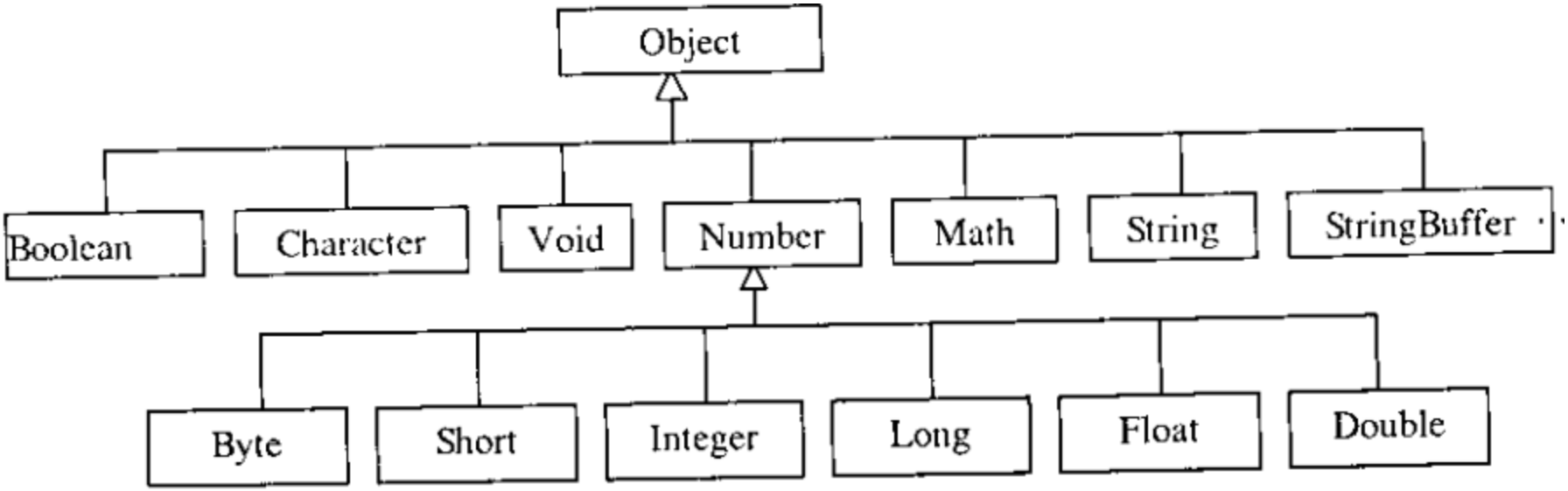
Introduction to a collections API

Google Android

Topics

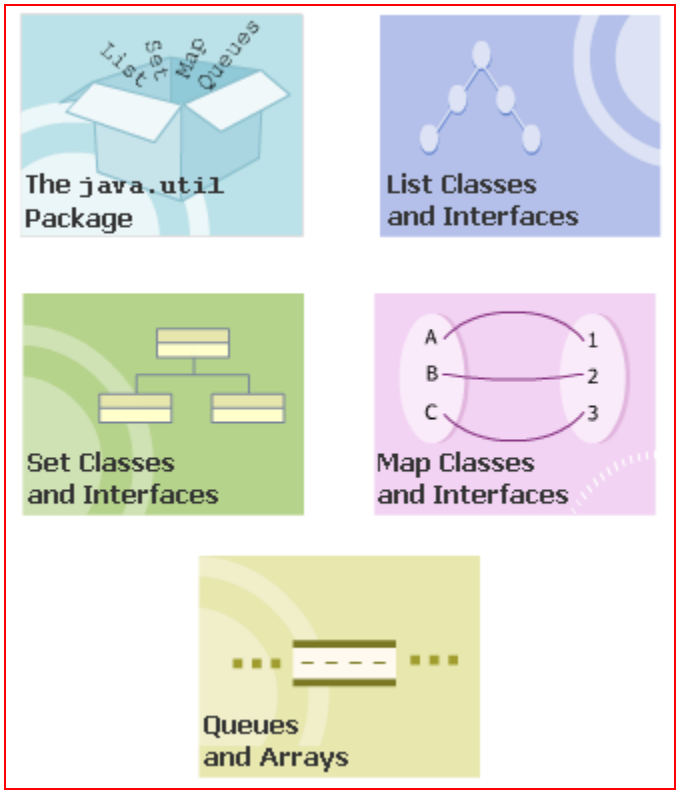
- Introduction to a List
- Introduction to a Set
- Introduction to a Map
- Introduction to a Queue
- Introduction to a Array

Basic Data Type

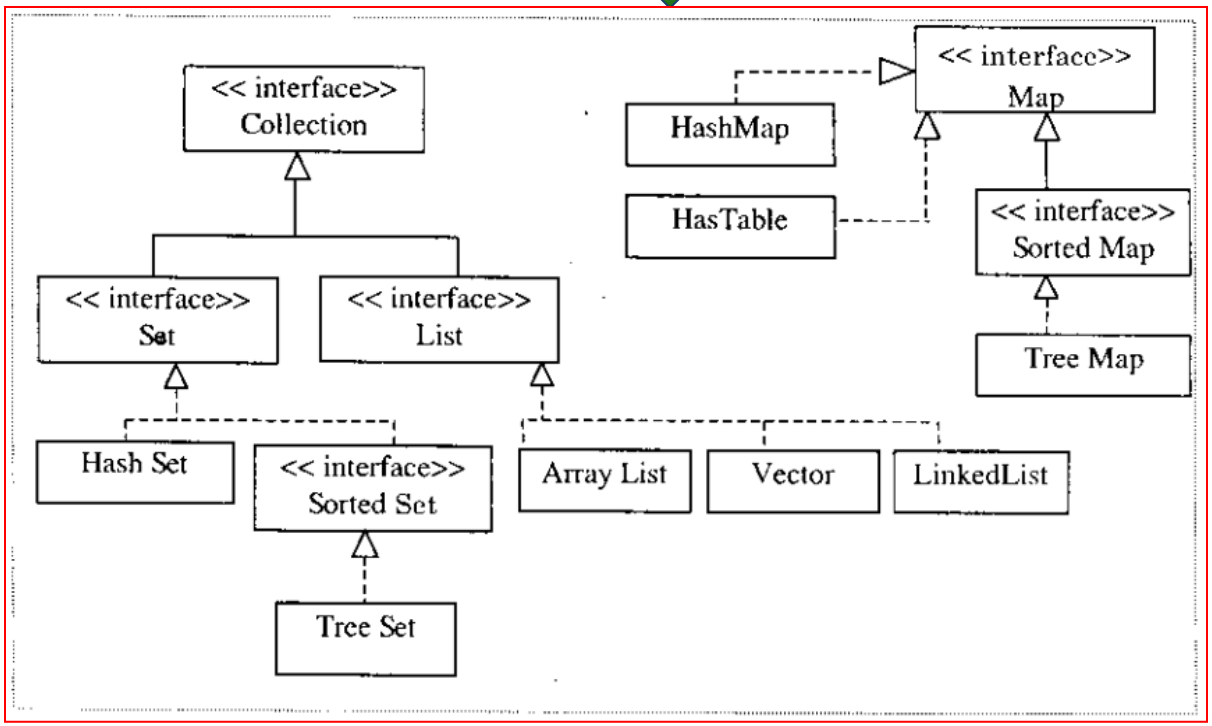


Collections API

Collections API (Application Programming Interface) is a unified architecture for representing and manipulating collections.



Collections API (con't)



android

Collections Framework

A collection is a container that helps to group multiple elements into a single unit. Collections help to store, retrieve and manipulate data.

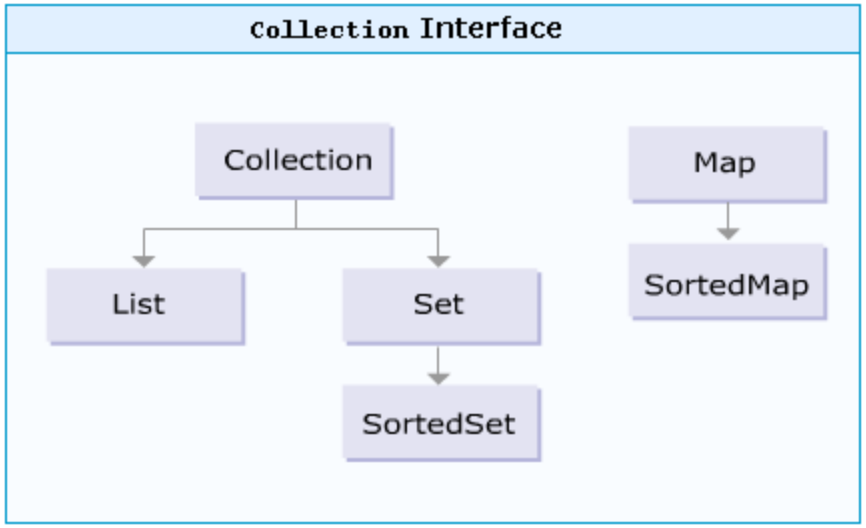
In Java, the Collections Framework provides a set of interfaces and classes for storing and manipulating groups of objects in a standardized way. The Collections Framework was developed with the following objectives in mind:

- It should be of high-performance.
- The different types of collections should work similar to each other and should have interoperability.
- It should be easy to extend a collection.

Collections Interface

Collections Framework consists of interfaces and classes for working with group of objects. At the top of the hierarchy lies the `Collection` interface. The `Collection` interface is extended by the following sub interfaces:

- `Set`
- `List`
- `Queue`



Collections Classes

Collection classes are standard classes that implement the `Collection` interfaces. Some of the classes provide full implementations of the interfaces whereas some of the classes provide skeletal implementations and are abstract. Some of the `Collection` classes are:

- `HashSet`
- `LinkedHashSet`
- `TreeSet`

Methods of "Collection" Interface

A `Collection` interface helps to pass collection of objects with maximum generality. A collection class holds a number of items in the data structure and provides various operations to manipulate the contents of the collection, such as add item, remove item, and find the number of elements.

The important methods in the `Collection` interface are:

- `add(E obj)`
- `contains(Object obj)`
- `isEmpty()`
- `size()`

"List" Interface

The `List` interface is an extension of the `Collection` interface. It defines an ordered collection of data and allows duplicate objects to be added to a list. Its advantage is that it adds position-oriented operations, enabling programmers to work with a part of the list.

The `List` interface uses an index for ordering the elements while storing them in a list. `List` has methods that allow access to elements based on their position, search for a specific element and return their position, in addition to performing arbitrary range operations.

Methods of "List" Interface

The methods supported by the `List` interface are:

- `add(int index, Object o)`
- `addAll(int index, Collection c)`
- `get (int index)`
- `set (int index, Object o)`
- `remove(int index)`
- `subList(int start, int end)`

"ArrayList" Class

`ArrayList` class is an implementation of the `List` interface in the Collections Framework.

The `ArrayList` class creates a variable-length array of object references. The list cannot store primitive values such as double. In Java, standard arrays are of fixed length and they cannot grow or reduce its size dynamically. Array lists are created with an initial size. Later as elements are added, the size increases and the array grow as needed.

`ArrayList` class is best suited for random access without inserting or removing elements from any place other than the end.

"ArrayList" Class (con't)

An instance of `ArrayList` can be created using any one of the following constructors:

- `ArrayList()`
- `ArrayList(Collection <? extends E> c)`
- `ArrayList(int initialCapacity)`

```
List<String> listObj = new ArrayList<String> ();
System.out.println("The size is : " + listObj.size());
for (int ctr=1; ctr <= 10; ctr++) {
listObj.add("Value is : " + new Integer(ctr));
}
…
```

In the given code snippet an `ArrayList` object is created and the first ten numbers are added to the list as objects. The primitive int data type is wrapped by the respective wrapper class.

Method of "ArrayList" Class

The `ArrayList` class inherits all the methods of the `List` interface.
The important methods in the `ArrayList` class are:

- `add(E obj)`
- `trimToSize()`
- `ensureCapacity(int minCap)`
- `clear()`
- `contains()`
- `size()`

Method of "ArrayList" Class

The ArrayList class inherits all the methods of the List interface.
The important methods in the ArrayList class are:

- add(E obj)
- trimToSize()
- ensureCapacity(int minCap)
- clear()
- contains()
- size()

```
List<String> listObj = new ArrayList<String> ();
System.out.println("The size is : " + listObj.size());
for (int ctr=1; ctr <= 10; ctr++) {
listObj.add("Value is : " + new Integer(ctr));
}
listObj.set(5, "Hello World");
System.out.println("Value is: " +(String)listObj.get
(5));…
```

"Vector" Class

The `Vector` class is similar to an `ArrayList` as it also implements dynamic array. `Vector` class stores an array of objects and the size of the array can increase or decrease. The elements in the `Vector` can be accessed using an integer index.

The difference between `Vector` and `ArrayList` class is that the methods of `Vector` are synchronised and are thread-safe. This increases the overhead cost of calling these methods. Unlike `ArrayList`, `Vector` class also contains legacy methods which are not part of collections framework.

"Vector" Class (con't)

The constructors of this class are:

- `Vector()`
- `Vector(Collection<? extends E> c)`
- `Vector(int initCapacity)`
- `Vector(int initCapacity, int capIncrement)`

Method of "Vector" Class

Vector class defines three protected members. They are:

- int capacityIncrement which stores the increment value.
- int elementCount which stores the number of valid components in the Vector.
- Object [] elementData which is the array buffer into which the components of the vector are stored.

```
Vector<Object> vecObj = new Vector<Object>();
vecObj.addElement(new Integer(5));
vecObj.addElement(new Integer(7));
vecObj.addElement(new Integer(45));
vecObj.addElement(new Float(9.95));
vecObj.addElement(new Float(6.085));
System.out.println("The value is: " +(Object)
vecObj.elementAt(3));
```

The important methods in the Vector class are:

- addElement(E obj)
- capacity()
- toArray()
- elementAt(int pos)
- removeElement(Object obj)
- clear()

Method of "Set" Interface

The `Set` interface creates a list of unordered objects. It creates non-duplicate list of object references. The `Set` interface inherits all the methods from the `Collection` interface except those methods that allow duplicate elements.

`Set` interface is best suited for carrying out bulk operations. The bulk operation methods supported by the `Set` interface are:

- `containsAll(Collection<?> obj)`
- `addAll(Collection<? extends E> obj)`
- `retainAll(Collection<?> obj)`
- `removeAll(Collection<?> obj)`

"Map" Interface

A `Map` object stores data in the form of relationships between keys and values. Each key map to at least a single value. If key information is known, its value can be retrieved from the Map object. Keys should be unique but values can be duplicated.

The Collections API provides two general-purpose Map implementation:

- `HashMap`
- `TreeMap`

`HashMap` is used for inserting, deleting and locating elements in a `Map`. `TreeMap` is used to arrange the keys in a sorted order.

Method of "Map" Interface

The important methods of a `Map` interface are:

- `put(K key, V value)`
- `get(Object key)`
- `containsKey(Object Key)`
- `containsValue(Object Value)`
- `size()`

"HashMap" Class

"HashMap" Class

The `HashMap` class implements the `Map` interface and inherits all its methods. An instance of `HashMap` has two parameters: initial capacity and load factor. Initial capacity determines the number of objects that can be added to the `HashMap` at the time of the hash table creation. The load factor determines how full the hash table can get before its capacity is automatically increased.

The constructors of this class are:

- `HashMap()`
- `HashMap(int initialCapacity)`
- `HashMap(int initialCapacity, float loadFactor)`
- `HashMap(Map<? extends K, ? extends V> m)`

```
staffObj.put("102", new EmployeeData("Harry Hacker"));
      staffObj.put("103", new EmployeeData("Joby
Martin"));
      System.out.println(staffObj);
      staffObj.remove("103");

     staffObj.put("106", new EmployeeData("Joby
Martin"));
     System.out.println(staffObj.get("106"));
    System.out.println(staffObj);
     …
}
```

"HashTable" Class

The `Hashtable` class implements the `Map` interface but stores elements as a key/value pairs in the hash table. While using a hashtable, a key is specified to which a value is linked. The key is than hashed and then the hash code is used as an index at which the value is stored. The class inherits all the methods of the `Map` interface.

The constructors of this class are:

- `Hashtable()`
- `Hashtable(int initCap)`
- `Hashtable(int initCap, float fillRatio)`
- `Hashtable(Map<? extends K, ? extends V> m)`


```
Hashtable<String, String> bookHash = new Hashtable
<String, String>();
bookHash.put("115-355N", "A Guide to Advanced Java");
bookHash.put("116-455A", "Learn Java by Example");
bookHash.put("116-466B", "Introduction to Solaris");
String str = (String) bookHash.get("116-455A");
System.out.println("Detail of a book " + str);
System.out.println("Is table empty " +
bookHash.isEmpty());
System.out.println("Does table contains key? " +
bookHash.containsKey("116-466B"));
Enumeration name = bookHash.keys();
while (name.hasMoreElements()) {
    String bkCode = (String)name.nextElement();
    System.out.println( bkCode +": " + (String)
bookHash.get(bkCode));
}
```

"Queue" Interface

A `Queue` is a collection for holding elements that needs to be processed. In `Queue`, the elements are normally ordered in First In First Out (FIFO) order. The priority queue orders the element according to their values. A queue can be arranged in other orders too. Queues support the standard collection methods and also provide additional methods to add, remove and review queue elements.

Some of the important methods supported by this class are:

- `poll()`
- `peek()`
- `remove()`
- `offer(E obj)`
- `element()`

"PriorityQueue" Class

Priority queues are similar to queues but the elements are not arranged in FIFO structure. They are arranged in a user-defined manner. The elements are ordered either by natural ordering or according to a comparator. A priority queue does not allow adding of non-comparable objects nor does it allow null elements. A priority queue is unbound and allows the queue to grow in capacity.

The constructors of this class are:

- `PriorityQueue()`
- `PriorityQueue(Collection<? extends E> c)`
- `PriorityQueue(int initialCapacity)`
- `PriorityQueue(int initialCapacity, Comparator<? super E> comparator)`
- `PriorityQueue(PriorityQueue<? extends E> c)`
- `PriorityQueue(SortedSet<? extends E> c)`

Method of "PriorityQueue" Class

The `PriorityQueue` class inherits the method of the `Queue` class. The other methods supported by the `PriorityQueue` class are:

- `add(E obj)`
- `clear()`
- `comparator()`

```
PriorityQueue<String> queue = new PriorityQueue
<String>();
queue.offer("New York");
queue.offer("Kansas");
queue.offer("California");
queue.offer("Alabama");

System.out.println("1. " + queue.poll()); // removes
System.out.println("2. " + queue.poll()); // removes
System.out.println("3. " + queue.peek());
System.out.println("4. " + queue.peek());
System.out.println("5. " + queue.remove()); // removes
System.out.println("6. " + queue.remove()); // removes
System.out.println("7. " + queue.peek());
System.out.println("8. " + queue.element());// Throws
Exception
```

"Array" Class

`Arrays` class provide a number of methods for working with arrays such as searching, sorting, and comparing arrays. The class has a static factory method that allows the array to be viewed as lists. The methods of this class throw an exception if the array reference is null.

Each of the methods that are listed has an overloaded version that differs according to the type of the array or array arguments. Some of the important methods of this class are:

- `equals(<type>[] arrObj1, <type>[] arrObj2)`
- `fill(<type>[] array, <type> value>)`
- `fill (<type>[] array, int fromIndex, int toIndex, type value)`
- `sort(<type> [] array)`
- `sort(<type> [] array, int startindex, int endIndex)`
- `toString()`

Module 8

Introduction to a ListViews

Google Android

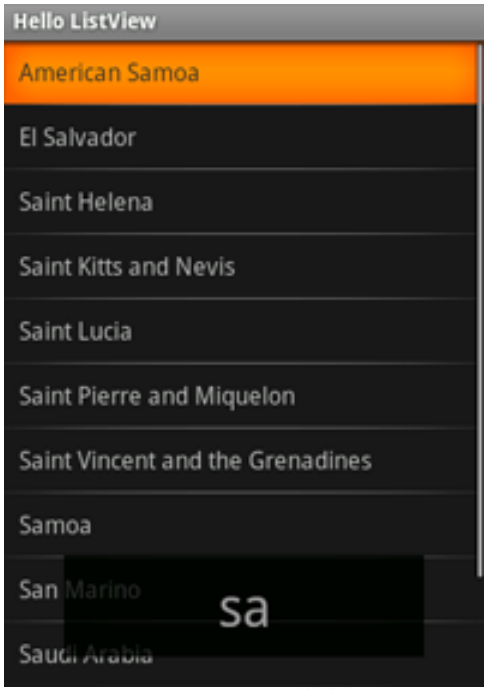
Topics

- What's ListView ?
- Elements of ListView
- Types of ListView
- Step By Step Create Normal ListView
- Step By Step Create Custom ArrayAdapter

What's ListView ?

Được dùng để hiển thị một danh sách thông tin trong từng dòng. Mỗi dòng thường được tải lên từ một file XML được cố định sẵn lượng thông tin và loại thông tin đó.

http://developer.android.com/reference/android/widget/ListView.html



Elements of ListView ?

Để hiển thị danh sách trên màn hình, ta cần 3 yếu tố chính sau:

- **Data Source** :Data Source có thể là ArrayList, HashMap, …
- **Adapter**: Adapter là một lớp trung gian được dùng để lưu dữ liệu tương ứng ở Data Source vào đúng vị trí của nó trên ListView.
- **Adapter view**: Đối tượng hiển thị trực quan thông tin trong Data Source để người dùng có thể tương tác và xử lý.

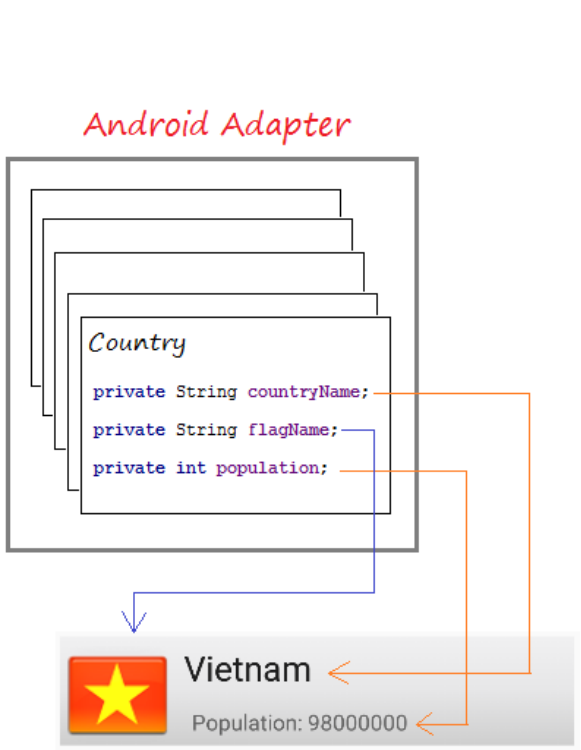
Elements of ListView ?

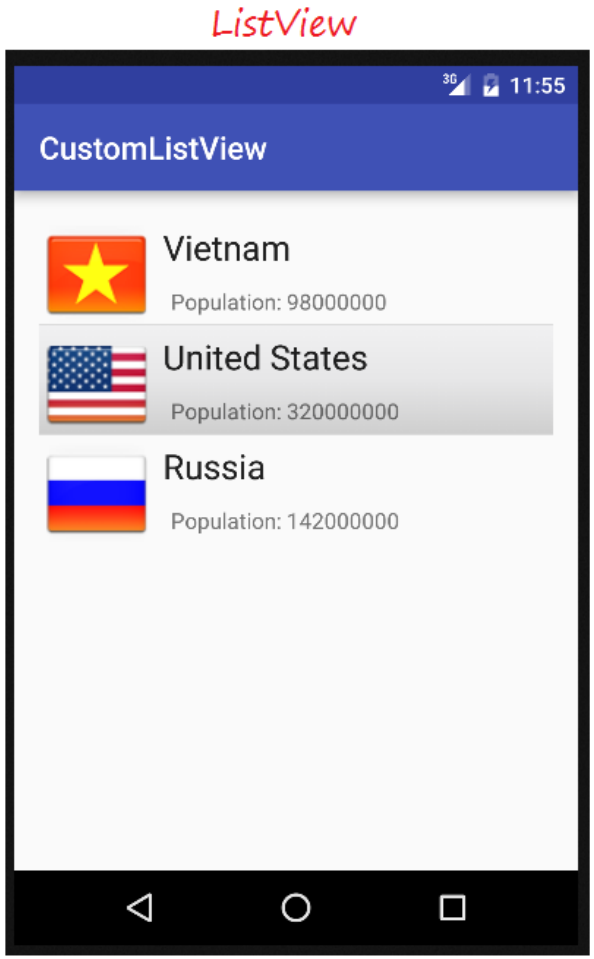
Example:

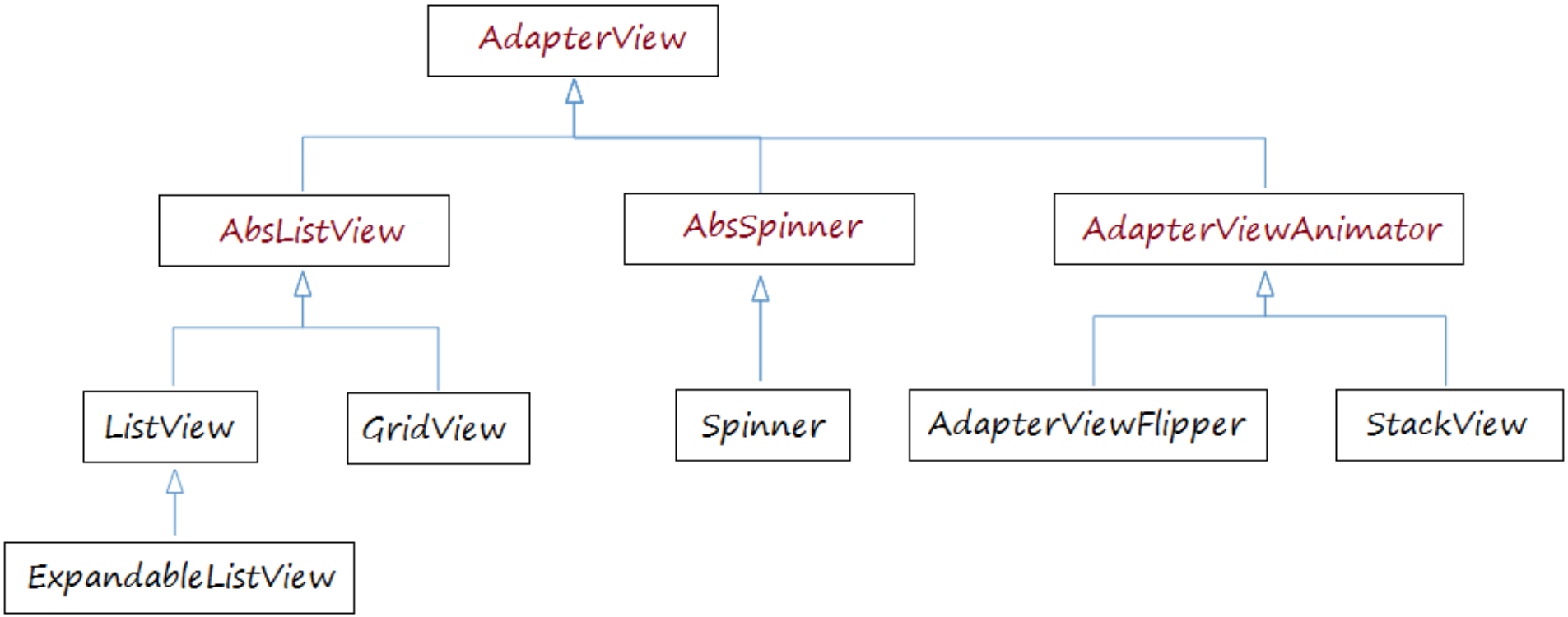
Array, Cursor,..

Example:

GridView, ListView, Spinner, StackView, ...

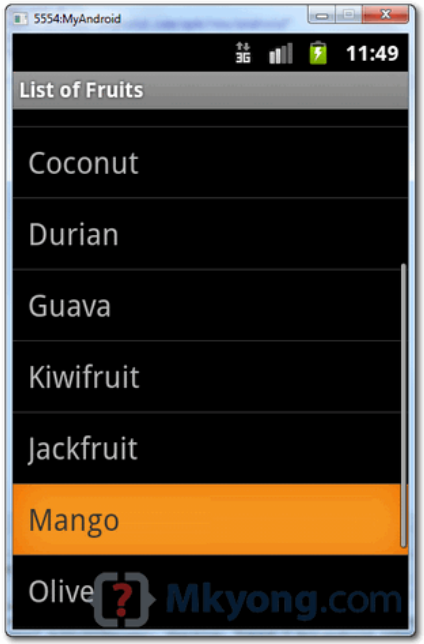

Android Adapter
Country
private String countryName;
private String flagName;
private int population;
Vietnam
Population: 98000000


ListView
11:55
CustomListView
Vietnam
Population: 98000000
United States
Population: 320000000
Russia
Population: 142000000

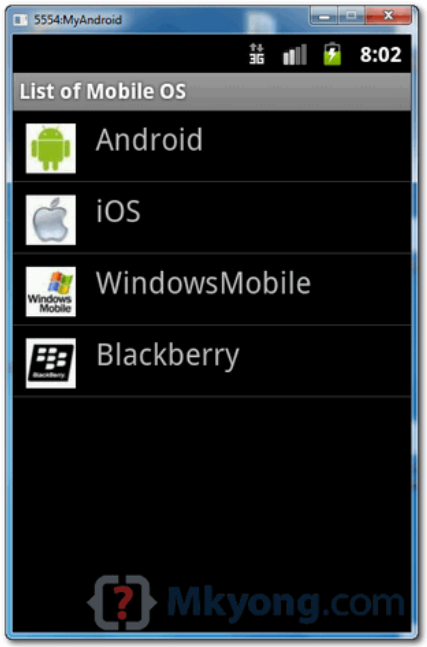

AdapterView
AbsListView
AbsSpinner
AdapterViewAnimator
ListView
GridView
Spinner
AdapterViewFlipper
StackView
ExpandableListView


android

Types of ListView ?



Normal ListView



Custom ArrayAdapter

ANDROID

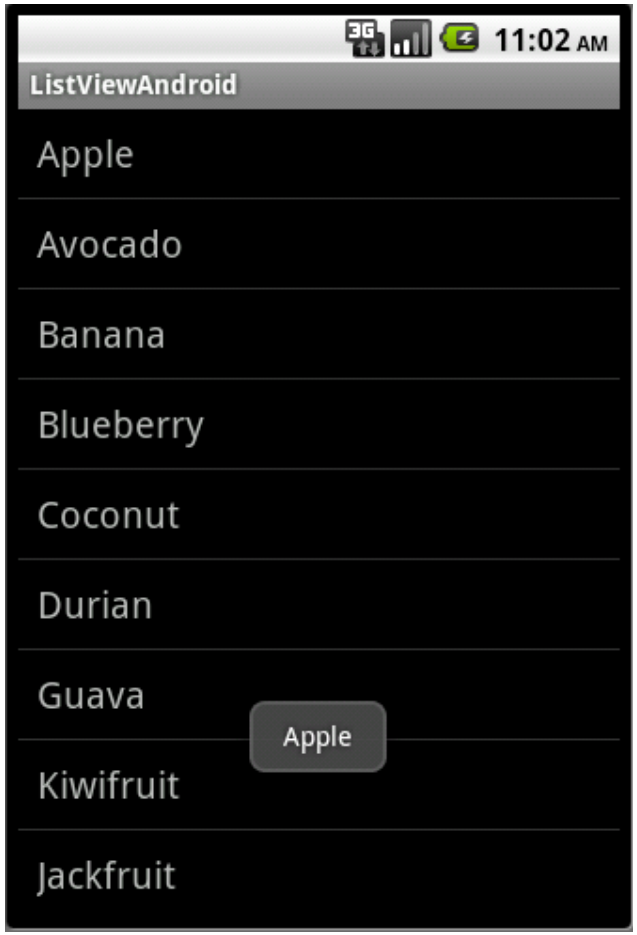
Step By Step Create Normal ListView

```
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
```

Step By Step Create Normal ListView

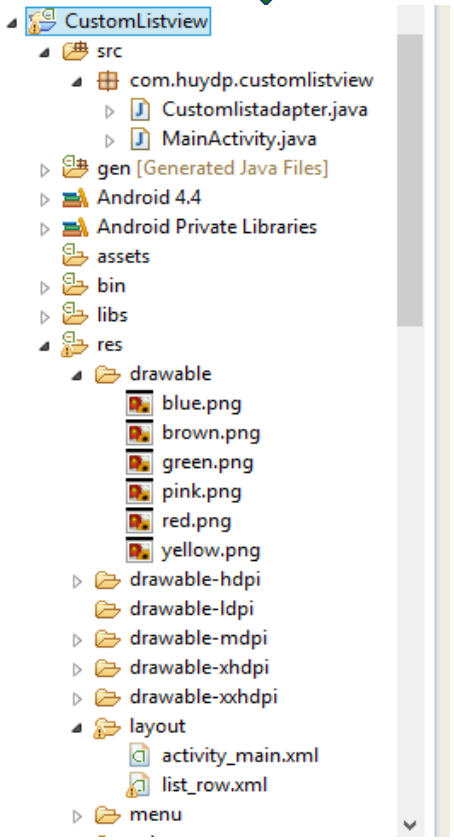
```
public class NormalListViewActivity extends ListActivity {
    /** Called when the activity is first created. */
    static final String[] FRUITS = new String[] { "Apple", "Avocado", "Banana",
            "Blueberry", "Coconut", "Durian", "Guava", "Kiwifruit",
            "Jackfruit", "Mango", "Olive", "Pear", "Sugar-apple" };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setListAdapter(new ArrayAdapter<String>(this,
                softech.aptech.bonn.listview.R.layout.layout_normal, FRUITS));

        ListView listView = getListView();
        listView.setTextFilterEnabled(true);

        listView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                // TODO Auto-generated method stub
                // When clicked, show a toast with the TextView text
                Toast.makeText(getApplicationContext(),
                        ((TextView) view).getText(), Toast.LENGTH_SHORT).show();
            }
        });

    }
}
```

Step By Step Create Normal ListView



Step By Step Create Custom listview

Step By Step Create Custom listview



Step By Step Create Custom listview

Để thực thi Listview với Image ta cần tạo 2 file layout xml như sau

- File thứ nhất chứa listview.
- File thứ hai chứa các mục trên từng dòng của listview, có thể chứa imageview và một textview.

Các bước tiến hành:

1. Tạo layout cho listview
2. Tạo layout cho dòng của listview
3. Áp dụng adapter tùy chỉnh để chứa 1 imageview và 1 textview
4. Viết file main activity. Tạo listview và thêm adapter cho nó tại đây.
5. Chạy chương trình để kiểm tra.

Step By Step Create Custom ArrayAdapter

Bước 1:

activity_main.xml

Đây là layout chính chứa listview.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:paddingBottom="@dimen/activity_vertical_margin"
 android:paddingLeft="@dimen/activity_horizontal_margin"
 android:paddingRight="@dimen/activity_horizontal_margin"
 android:paddingTop="@dimen/activity_vertical_margin"
 tools:context=".MainActivity" >
 <ListView
 android:id="@+id/listView"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentLeft="true"
 android:layout_alignParentTop="true" >
 </ListView>
</RelativeLayout>
```

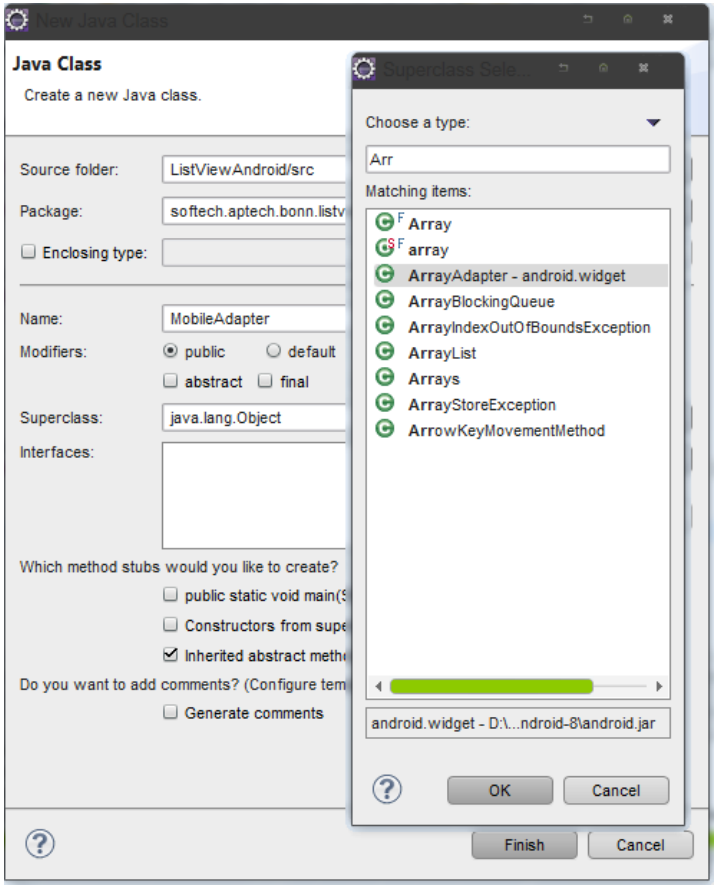
Bước 2:

list_row.xml

Sau khi tạo layout cho listview ta tạo ra dòng cho nó -> tạo layout chứa 1 ảnh và 1 text field.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal" >

<ImageView
android:id="@+id/imageView"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />

<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="40dp"
android:layout_gravity="center_vertical"
android:text="TextView" />

</LinearLayout>
```

Step By Step Create Custom ArrayAdapter



Bước 3

Customlistadapter.java

Tạo adapter cho listview, sử dụng arrayadapter. Vì mỗi dòng chứa một ảnh và một text nên cần hai mảng để chứa ‘id của tất cả ảnh’ và ‘dữ liệu của text’.

```
public class Customlistadapter extends ArrayAdapter<String>{
	String[] color_names;
	Integer[] image_id;
	Context context;

	public Customlistadapter(Activity context,Integer[] image_id,
String[] text){
	super(context, R.layout.list_row, text);
		// TODO Auto-generated constructor stub
		this.color_names = text;
		this.image_id = image_id;
		this.context = context;
	}

	@Override
	public View getView(int position, View convertView, ViewGroup
parent) {
	// TODO Auto-generated method stub
		LayoutInflater inflater = (LayoutInflater) context
		.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
		View single_row = inflater.inflate(R.layout.list_row,
null,
		true);
		TextView textView = (TextView)
single_row.findViewById(R.id.textView);
		ImageView imageView = (ImageView)
single_row.findViewById(R.id.imageView);
		textView.setText(color_names[position]);
		imageView.setImageResource(image_id[position]);
		return single_row;
	}
}
```

Bước 4:

MainActivity.java

Sử dụng adapter cho listview trong file main, tại đây ta gán các ảnh là các id kiểu integer và dữ liệu text là tên các màu hiển thị tương ứng.

```
public class MainActivity extends Activity {

	String color_names[] = {"red", "green", "blue", "yellow", "pink",
"brown"};
	Integer image_id[] = {R.drawable.red, R.drawable.green,
R.drawable.blue, R.drawable.yellow, R.drawable.pink, R.drawable.brown};
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		Customlistadapter adapter = new Customlistadapter(this, image_id,
color_names);
		ListView lv = (ListView) findViewById(R.id.listView);
		lv.setAdapter(adapter);

	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is
present.
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

}
```

Q&A

Thanks you every one ☺

Google Android