**Chương 4: Xử lý sự kiện**

* **Quy trình làm việc với listview:**

Xác định nguồn dữ liệu 🡺 Tạo Adapter cùng kiểu dữ liệu với nguồn – gán nguồn 🡺 Gán Adapter cho list view.

* **Tạo layout Item**: Chứa row cho mỗi item list được custom
* **Tạo model**: Khởi tạo các thuộc tính phục vụ cho việc lấy dữ liệu – Kế thừa Serializable
* Khởi tạo các thuộc tính: get, set, constructor
* **Tạo adapter** kế thừa ArrayAdapter<model>
* Nguồn dữ liệu có kiểu gì thì Adapter có kiểu đó
* Khởi tạo constructor (resource = item)
  + Hiệu chỉnh hàm getView() @Overide
  + Custom lại adapter

public class **CustomAdapter** extends ArrayAdapter<[*Kiểu đối tượng*]>{

public **CustomAdapter**(Context context, int resource, List<[*Kiểu đối tượng*]> objects) {

super(context, resource, objects);

}

}

|  |
| --- |
| Định nghĩa LayoutInflater:  **Instantiates a layout XML file into its corresponding {@link android.view.View} objects.**   * Nó dùng để inflate (bơm, nạp...  ) cái phần XML (Phần giao diện) vào View. * Giả sử bạn muốn tạo một cái view, thì để biết cái view đó đang chịu trách nhiệm phần giao diện nào (XML) thì bạn phải inflate cái XML đó vào view. * Một trường hợp rất thường gặp là khi bạn hiện thực CustomAdapter cho ListView. Trong method getView(), bạn cần phải inflate cái XML (ở đây là layout cho item) vào biến convertView thì ListView mới biết là phần giao diện nào của item cần được thể hiện ra màn hình. * Lớp LayoutInflater này rất hữu ích nếu bạn dùng để tạo giao diện linh hoạt (có thể thêm bớt các view vào root view) chỉ bằng code thuần . |

public **View getView**(int position, View convertView, ViewGroup parent) {

LayoutInflater inflater=**this.**context.getLayoutInflater();*//một lớp dùng để build một layout bt=> một code java android có thể sử dụng được*

View item=inflater.inflate(**this**.resource,null);

**//…. Phai lấy hết tất cả giao diện trong row (item) 🡺 lấy txt, btn**

TextView txtMuaTM = (TextView) row.findViewById(R.id.***txtMuaTM***);

…

**final** TiGia db = **this**.**objects**.get(position);*//tra ve ti gia hien tai*

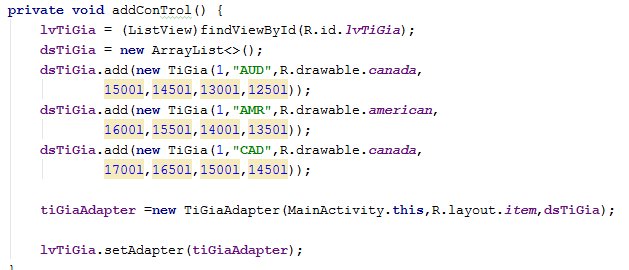
*muon ve*

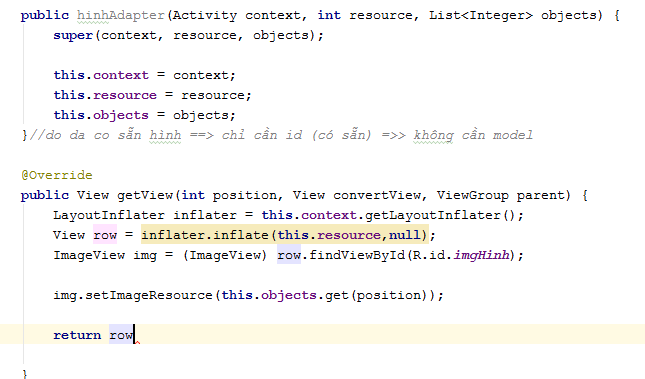
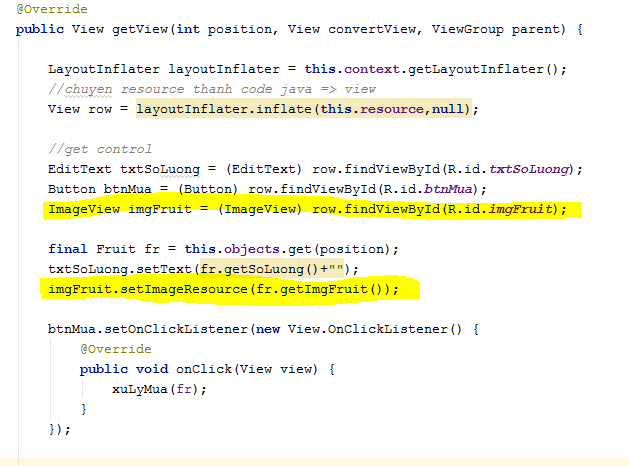
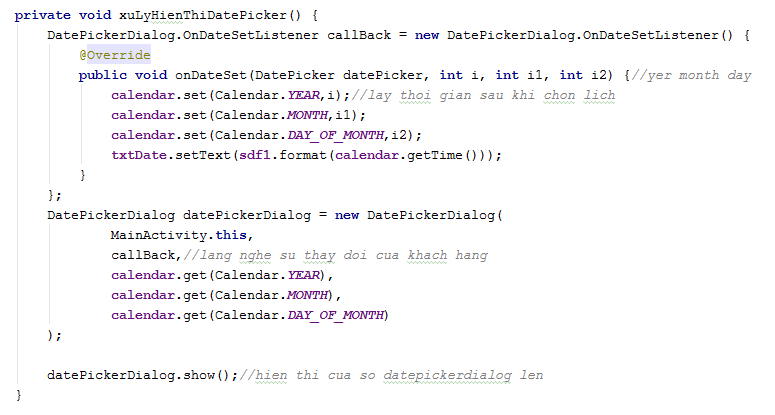
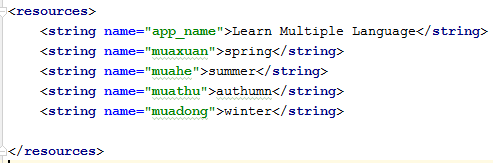
*…*txtNuoc.setText(db.getNuoc().toString());

**return** item;//row 🡺 quy định giao diện

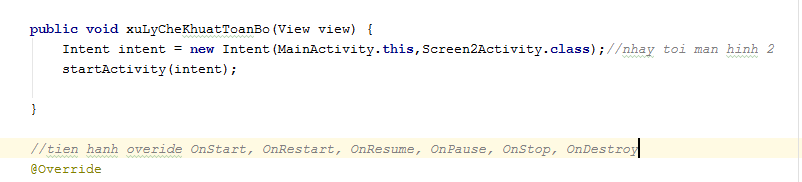
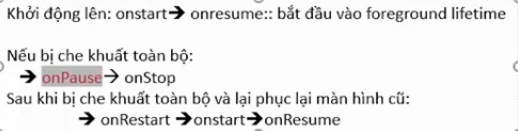
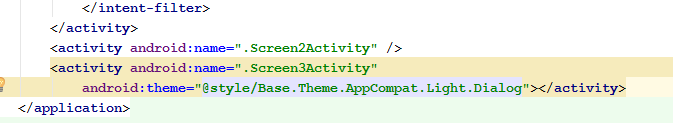
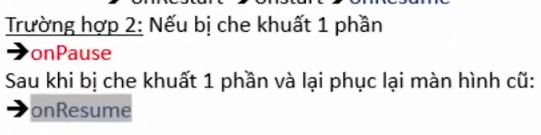
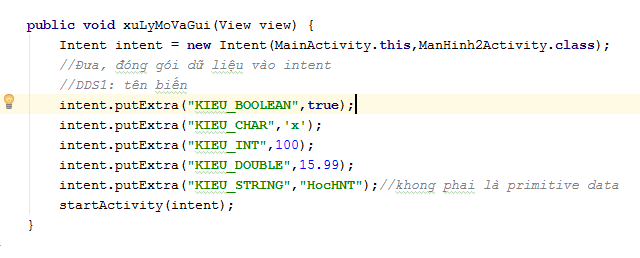
}

* Khai báo bộ ba listview 🡺 ArrayList(data) 🡺 Adapter 🡺 tiến hành thêm control và event



* **Gridview**
* Tương tự ListView (Hiển thị dạng ống) 🡺 tạo layout riêng 🡺 lấy hình(mảng dữ liệu kiểu int), dữ liệu…
* Activity.xml🡺 thêm thuộc tính numColumn
* Custom + Overide : 
* Xử lý nhân ảnh dựa trên ID Drawable 🡺 truyền id qua thuộc tính được khai báo ở model:
  + Lưu ý với EditText\*(number): khi setText phải chuyển về kiểu String + “”
* 
* **Làm việc với DateTimePicker**
* Sử dụng variable As Listener (lắng nghe, thay đổi sự kiện)
* Khai báo Date/TimePickerDialog 🡺 new đối số (màn hình sử dụng, Variable As Listener (onDateSet,,,,), year,mon,day)
* Set Giá trị vào Ontime\dateSet:
* **Làm việc với Tabselector:**
* Sử dụng FrameLayout
* 3 phần: Tab Host, TabWidgets, FrameLayout
* Quy trình: Layout 🡺 findId Tabhost (qua setup()), TabSpec “tabSpec” 🡺 tạo tab: “tabSpec”.newTabspec(“name”)
* Activity.xml: Framelayout 🡺 Linearlayout 🡺<**include layout="@layout/screen1"**/>
* Sửa id của từng linearLayout để setContent
* TabHost tabHost = (TabHost) findViewById(R.id.***tabHost***);*//ở version mới có thể bắt sửa @android:id*
* Lấy drawable (search drawble for indicator android)
* **Multiple language:**
* Cấu hình giá trị trong file string.xml(mặc định là tiếng anh): 
* Tạo resource theo Locate ngôn ngữ muốn sử dụng (trùng **name)🡺 new value 🡺 resource**

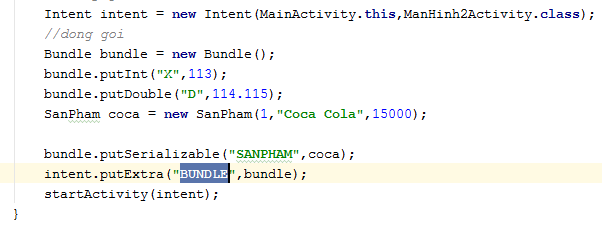
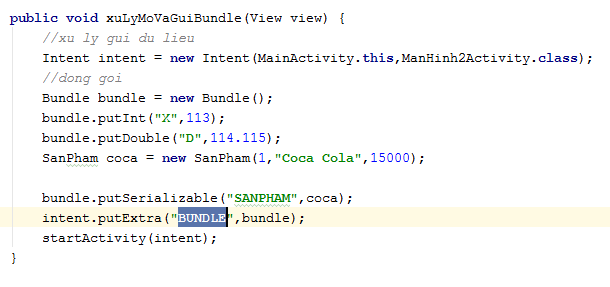
**Chương 5: Activity**

* **Activity và vòng đời**
* Có nhiều trạng thái khác nhau **(**[**Tham khảo slide 32**](document_32-activity.pptx)**)**
  + Running : người sử dụng đang tương tác
  + Pause: tạm dừng, ngưng sử dụng
  + Stop: ngưng sử dụng hoàn toàn (Không phải destroy)
* Phân biệt **Visible lifetime** và **Foreground lifetime (**[**Tham khảo slide 32**](document_32-activity.pptx)**)**
  + **Visible Lifetime:**Bắt đầu hoạt động 🡺 chuẩn bị ngắt, tắt chương trình
  + **Foreground Lifetime:** là 1 phần trong Visible, là vòng đời cho phép người sử dụng tương tác với phần mềm ⬄ người sử dụng chỉ có thể tương tác trong Foregound lifetime
* Quy trình:
  + New Activity🡺emply 🡺 xử lý
  + 
  + Sự kiện theo thứ tự xử lý (che khuất toàn bộ 🡺 trở về): 
  + Xử lý che khuất 1 phần: AndroidManifest: 
  + Sự kiện theo thứ tự xử lý (che khuất một phần🡺 trở về):
* **Intent** [**(Tham khảo slide 33 – Intent)**](document_33-intent.pptx)
* Công dụng:
  + Khởi động Activity
  + Khởi động Service
  + Chuyển thông điệp tới Broadcasst Receiver
* Loại:
  + Implicit : built sẵn trong android (không tường minh)
  + Explicit: do android hoặc do ta xây dựng (Chỉ rõ thành phần tham gia)
* Gọi activity do System quản lý
* Mở:
  + Đơn thuần.
  + Gửi dữ liệu.
  + Gửi dữ liệu và nhận kết quả trả về.
* **Truyền và nhận dữ liệu giữa các Activity**
* Các trường hợp sử dụng
  + Mở 1 màn hình
  + Mở một màn hình và gửi dữ liệu (primitive,object data)
    - Gửi trực tiếp vào **intent**
    - Đóng gói vào **Bundle** trước
  + Mở màn hình gửi dữ liệu, nhận kết quả trả về
* Chuyển dữ liệu Activity:
  + Đóng gói dữ liệu vào intent: 
  + Hứng dữ liệu từ màn hình trước:



* + Hứng kiểu đối tượng:

Intent.getSerializableExtra(“Tên Biến”);(bắt buộc ép kiểu)

* Chuyển gửi dữ liệu đóng gói qua bundle(container – Đóng gói)
  + Gửi dữ liệu  
    
  + Hứng dữ liệu  
    
* **Truyền dữ liệu và tự động lắng nghe kết quả trả về giữa các Activity**
* **Xây dựng Intent gọi đến các ứng dụng HĐH**
* **Project**