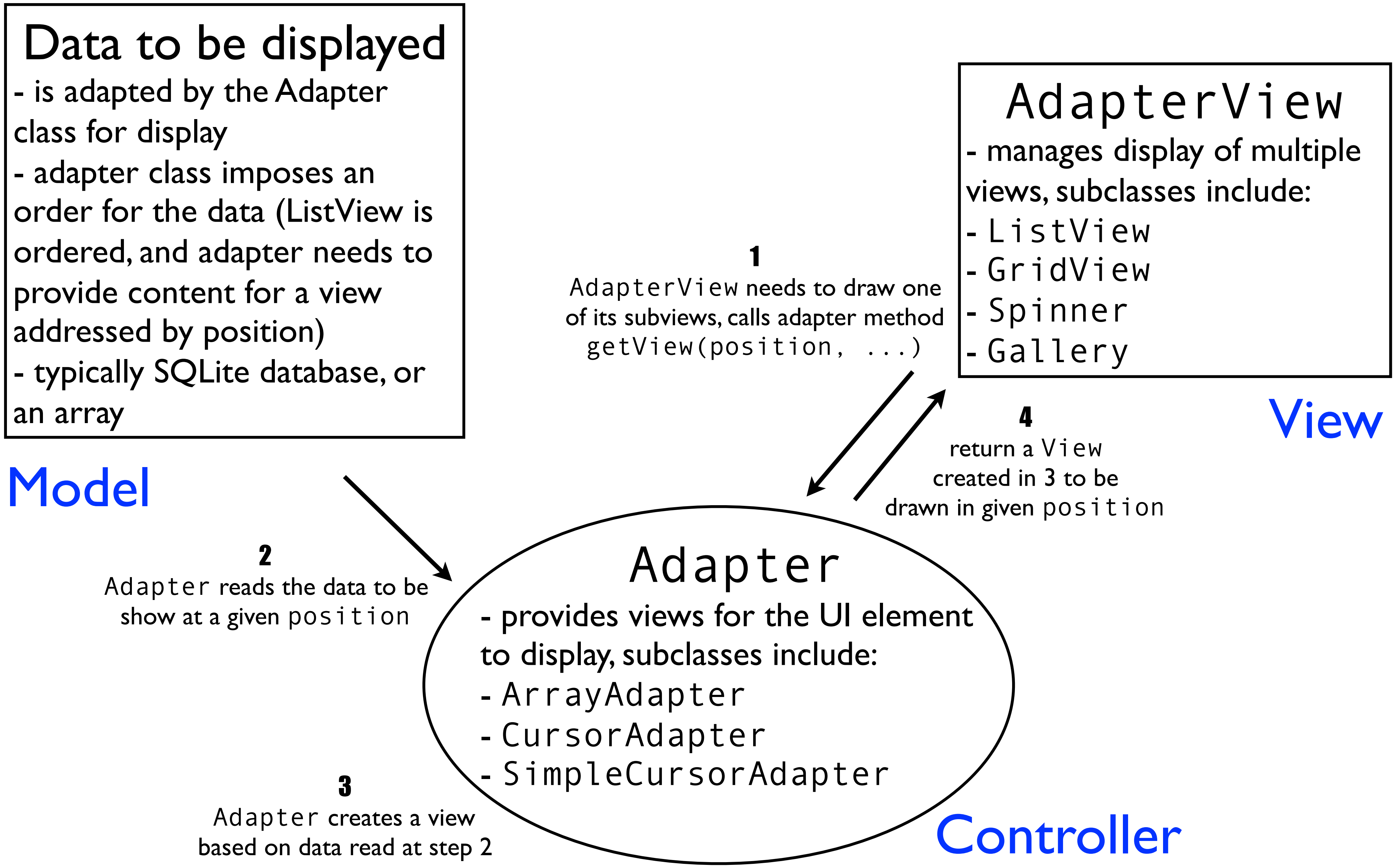
Composite UI Views

- Often a UI element is expected to show together a number of related views
  - address book, photo album, browser history, course list
- Composite UI element may add some functionality that makes it easier to browse the data
  - scrollbars, flipping pages etc.
- Composite UI element also makes it possible to select an item to be acted on (and select the action)
  - show/play etc the item, edit the item, delete the item

Adapter architecture



Data to be displayed

- is adapted by the Adapter class for display
- adapter class imposes an order for the data (ListView is ordered, and adapter needs to provide content for a view addressed by position)
- typically SQLite database, or an array

Model

`AdapterView`

- manages display of multiple views, subclasses include:
- `ListView`
- `GridView`
- `Spinner`
- `Gallery`

View

1
`AdapterView` needs to draw one of its subviews, calls adapter method `getView(position, ...)`

2
`Adapter` reads the data to be show at a given `position`

3
`Adapter` creates a view based on data read at step 2

4
return a `View` created in 3 to be drawn in given `position`

`Adapter`

- provides views for the UI element to display, subclasses include:
- `ArrayAdapter`
- `CursorAdapter`
- `SimpleCursorAdapter`

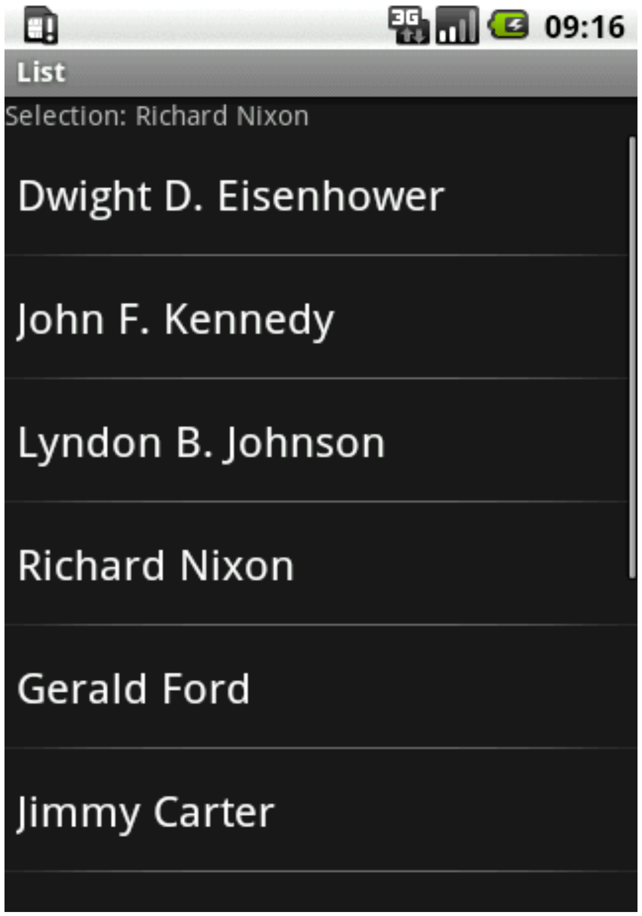
Controller

ListView and adapter

- ListView can well be used in part of a normal layout, related to normal activity.
  - `ListActivity` hosts `ListView` objects that can be bound to different data sources
  - `ListActivity` has a default layout – customized layout can be defined by adding a `ListView` object with the id “`@android:id/list"`
- Data binding will be done by implementing some adapter and then binding that to `ListView` with `setListAdapter()` method.
- Multiple adapter types exist, each with different levels of functionality. One often used in simple cases is generic `ArrayAdapter<T>` for arrays of objects of class T

ArrayAdapter example

- Create String Array
  - `String[] presidents = {…}`
- Android provides some standard layout resources in the `R.layout` class, e.g. `simple_list_item_1`
- Instantiate `ArrayAdapter`
  - `ArrayAdapter<String> myAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, presidents);`
- Bind view and adapter
  - `setListAdapter(myAdapter);`



Selection: Richard Nixon

Dwight D. Eisenhower

John F. Kennedy

Lyndon B. Johnson

Richard Nixon

Gerald Ford

Jimmy Carter

onCreate and onListItemClick

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ArrayAdapter<String> myAdapter = new ArrayAdapter<String>(this,R.layout.list,
                              android.R.layout.simple_list_item_1, presidents);
  ListView list = (ListView)getListView();
  list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
      Log.d("TAG", "onItemClick() @ " + position);
     }
  });

  setListAdapter(myAdapter);
}
```

Customized adapter for list view

- Inherit from the `BaseAdapter` and implement `ListAdapter` interface.
- `Implement` `getView()` method to return a `View` (whose layout is, for example, `LinearLayout`) for list view cell
- If content is changed, remember to notify adapter with `notifyDataSetChanged()`
- This is probably the way to implement adapter except in the very simplest cases
  - No hard-to-find restrictions that some of the ready-made adapters have
  - Full control on the mapping
  - (Note: in *observer* methods make sure to call superclass method)

adapter methods

```
@Override
public int getCount() {
    return MyModel.getInstance().getTeam().getPlayerCount();
}

@Override
public Object getItem(int position) {
    return this.currentList.get(position);
}

@Override
public long getItemId(int position) {
    return position;
}

@Override
public boolean hasStableIds() {
    return false;
}

@Override
public int getItemViewType(int position) {
    return 0; // all views are similar…
}

@Override
public int getViewTypeCount() {
    return 1; // … so, one type only
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
  …
}
```

```
@Override
public boolean areAllItemsEnabled() {
    return true;
}

@Override
public boolean isEnabled(int position) {
    return true; // unless you use separators, this is good
}

@Override
public void registerDataSetObserver(DataSetObserver observer) {
    super.registerDataSetObserver(observer);
}

@Override
public void unregisterDataSetObserver(DataSetObserver observer) {
    super.unregisterDataSetObserver(observer);
}


@Override
public boolean isEmpty() {
  return (getCount() == 0);
 }
```

remember to call super methods here

this is important, more on next slide

getView() example

```java
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View targetView = convertView;
    if(targetView == null) {
        targetView = this.layoutInflater.inflate(R.layout.playerlistitem, null);
    }

    Player p = this.currentList.get(position);
    if(p != null) {
        TextView tv = (TextView)targetView.findViewById(R.id.name);
        tv.setText(p.getName());
        tv = (TextView)targetView.findViewById(R.id.number);
        tv.setText("" + p.getNumber());
        tv = (TextView)targetView.findViewById(R.id.salary);
        tv.setText("" + p.getSalary());
    }
    return targetView;
}
```

convertView contains (possibly) a view that can be filled with new data

inflater creates the view from XML description

fill the view just like any layout

Reading list

- http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews
- http://developer.android.com/guide/topics/ui/binding.html