1->Notification:

1. 通过getSystemService()得到NotificationManager
2. 通过构造方法创建Notification.Builder对象，并设置属性
3. 调用Notification.Builder对象的builder()方法得到Notification对象
4. 通过NotificationManager的notify()发送Notification
5. 通过NotificationManager的cancel()取消Notification

2->Handler:

1. 在**接收线程**中调用**Looper**.prepare()得到本线程的Looper
2. 在**接收线程**中创建Handler对象，重写handleMessage()方法
3. 在**接收线程**中调用**Looper**.loop()启动Looper
4. 在**发送线程**中调用接收线程的Handler对象发送消息

3->获取LayoutInflater:

1. 调用Context对象的getLayoutInflater()

LayoutInflater inflater = context.getLayoutInflater();

1. 调用getSystemService()

LayoutInflater inflater = getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

1. 调用LayoutInflater的from()静态方法

LayoutInflater inflater = LayoutInflater.from(context);

4->AlertDialog:

1. 通过构造方法创建AlertDialog.Builder对象
2. 调用AlertDialog.Builder对象的setXxx()方法设置属性
3. 调用AlertDialog.Builder的**create()**方法创建AlertDialog对象
4. 调用AlertDialog对象的show()方法显示Dialog

【设置单选列表项对话框： 调用AlertDialog.Builder的setSingleChoiceItems()】

【设置多选列表项对话框： 调用AlertDialog.Builder的setMultiChoiceItems()】

【自定义列表项对话框： 调用AlertDialog.Builder的setAdapter()】

【自定义View对话框： 调用AlertDialog.Builder的setView()】

5->BroadcastReceiver

启动：

1. 创建Intent
2. 调用Context的sendBroadcast(Intent)系列方法

实现：

1. 创建BroadcastReceiver的子类，重写onReceive()方法
2. 注册该BroadcastReceiver

注册：

1. 代码中动态注册

**registerReceiver(receiver,intentFilter)**/**unregisterReceiver(receiver)**

1. AndroidMainfest.xml文件中静态注册

6->反射机制获取类的3种方式

1. **调用Class.forName()静态方法**

Class c = Class.forName(“className”);

1. **通过类的class属性**

Class c = java.lang.Long.class;

1. **通过对象调用getClass()方法**

Long l = 99l;

Class c = l.getClass();

7->反射机制

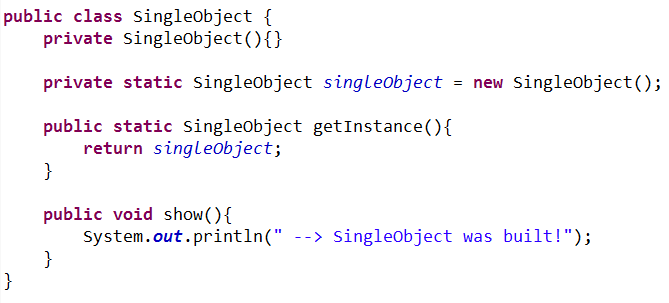
1. 通过的方式获取Class类的对象，其代表正在运行的类或接口
2. 调用Class类的一系列方法获取各种信息

**8->工厂模式**

1. **定义一个接口，抽象出工厂实体类的方法**
2. **定义工厂实体类**
3. **定义工厂类，能根据传入的信息创建不同的对象**
4. **创建工厂类的对象，调用其创建方法，传入不同信息，创建不同对象**

9->单例模式

1. 单例类中构造函数私有化
2. 单例类中创建本类的静态对象
3. 单例类中定义一个静态方法将本类对象向外界暴露



10->单例模式的几种实现方式

1. 线程不安全懒汉式
2. 线程安全懒汉式
3. 饿汉式
4. 双检锁 [ 对线程不安全懒汉式的改进 ]
5. 静态内部类 [ 对饿汉式的改进 ]

11->观察者模式

1. 观察者实现：

(1)、定义构造方法，构造方法中需要传入被观察者对象，通过该对象将自己添加到被观察者的通知队列中

(2)、实现自己的被通知方法

1. 被观察者实现：

(1)、定义观察者集合

(2)、定义添加/删除观察者的方法

(3)、定义被观察的数据并在其被改变时通知观察者

1. 测试类实现

(1)、创建被观察者对象

(2)、在构造方法中传入被观察者对象创建观察者对象

(3)、被观察者对象改变状态

12-> 策略模式

1. 定义所有不同策略类需要实现的接口
2. 定义不同的策略类
3. 定义使用策略模式的行为类，该类的行为方法中调用策略对象的方法执行操作
4. 在测试类中查看行为类对象在策略改变时的行为变化

13-> 用SQLiteDatabase进行数据库查询

1. 调用静态方法SQLiteDatabase.openDatabase()或openOrCreateDatabase()得到SQLiteDatabase对象
2. 调用SQLiteDatabase对象的execSQL()方法创建数据表
3. 调用SQLiteDatabase对象的query()方法查询，得到Cursor对象（或其他操作）
4. 调用Cursor对象的moveXxx()系列方法移动指针
5. 调用Cursor对象的getXxx()方法获取指定行的指定列的数据

14->使用SQLite3管理数据库

启动数据库： sqlite3 db路径

查看当前数据库： .databases

查看数据库里的数据表： .tables

查看sqlite3支持的命令： .help

当应用提示某条SQL语句有错误时，可以利用sqlite3这个工具测试这条语句，以保证语法正确。

15->使用SQLiteOpenHelper工具类：

1. 创建SQLiteOpenHelper的子类，重写onCreate()和onUpgrade()方法
2. 创建SQLiteOpenHelper的子类对象，调用getWritableDatabase()方法得到SQLiteDatabase对象
3. 最后调用SQLiteOpenHelper对象的close()方法关闭数据库

16-> SharedPreferences接口：

1. 调用context.getSharedPreferences()方法得到SharedPreferences对象
2. 调用SharedPreferences对象的edit()方法得到SharedPreferences.Editor对象
3. 调用SharedPreferences.Editor对象的putXxx()方法存入数据
4. 调用SharedPreferences.Editor对象的commit()方法提交修改
5. 调用SharedPreferences对象的registerOnSharedPreferenceChangeListener()和unregisterOnSharedPreferenceChangeListener()方法监听SharedPreferences

17-> 解析XML

DOM、SAX、PULL 3种方法前3步都是：

1. 得到解析器工厂对象
2. 得到解析器对象
3. 传入XML文件路径（得到Document对象或开始解析）

18-> Fragment

**静态Fragment:**

将Fragment作为普通控件，直接写在Activity的布局文件中，通过android:name 属性让布局文件与fragment.java文件关联。此时Fragment不能动态切换，也不需要 使用到FragmentManager、FragmentTransaction类，与普通控件无异。（作为Fragment， 需要写其布局文件和.java文件）

**动态Fragment:**

将需要切换Fragment的部分定义为FrameLayout，通过FragmentTransaction的 replace()和commit()交替显示不同的Fragment.

<include

android:id="@+id/layout1">

layout="@layout/layout2"/>

MainActivity.java中对Fragment切换步骤：

1. 定义各个Fragment对象

fragment = new Fragment();

1. 得到FragmentManager对象

manager = context.getFramentManager();

1. 得到FragmentTransaction对象

transaction = manager.beginTransaction();

1. 调用FragmentTransaction对象的replace()方法切换fragment

transaction.replace(帧布局id, fragment);

1. 调用FragmentTransaction对象的commit()提交事务

transaction.commit();

19-> PopupMenu:

1. 定义PopupMenu所依附的View
2. 通过构造方法定义PopupMenu

PopupMenu menu = new PopupMenu(上下文，所依附View);

1. 使用xml文件填充PopupMenu布局

menu.getMenuInflater().inflate(R.menu.menu\_list, menu.getMenu());

1. 为PopupMenu注册监听器

menu.setOnMenuItemClickListener();

1. 显示PopupMenu

menu.show();