**1.abstract 和 interface 區別**

Abstract 類是抽象共有的行為 extents （一對一）

Interface 是類特有的行為 implements （一對多）

**2.Cloneable：**

Java的淺拷貝和深拷貝

Implements Cloneable

實現clone()函數。

Object obj1 = Object obj2；

obj1 == obj2; //true

obj1.equals(obj2); //true

Object obj2 = obj1.clone();

Obj2 == obj1; //false

Obj2.equals(obj1); // true;

**3. Java中finalize()**

1.class Object 中的一個protected方法，子類可以重寫該方法。

2.垃圾回收器要回收对象的时候，首先要调用这个类的finalize方法。

3.一般純java對象不需要重寫該函數，但是當java對象中調用了JNI方法而且new 對象，需要在該函數中釋放該對象。

4.android中調用堆棧：

at com.example.testfl.MainActivity$TestFinalize.finalize(MainActivity.java:99)

at java.lang.Daemons$FinalizerDaemon.doFinalize(Daemons.java:198)

at java.lang.Daemons$FinalizerDaemon.run(Daemons.java:181)

at java.lang.Thread.run(Thread.java:818)

5.java中的FinalizerDaemon 守護線程調用該函數。

6. 除此以外,正常情况下,当某个对象被系统收集为无用信息的时候,finalize()将被自动调用,但是jvm不保证finalize()一定被调用,也就是说,finalize()的调用是不确定的,这也就是为什么sun不提倡使用finalize()的原因。只有在创建匿名对象并且显示地调用了 System.gc()方法的时候，finalize方法才被调用了。

**4.Java中變量的初始化：**

局部變量報錯；成員變量設置默認值；

構造函數中調用init函數；

（static）靜態成員的定義和初始化

**5.java 中static關鍵字：**

**類的成員變量：**

**方法：**

動態的內部類不能有static方法和static變量，靜態的內部類可以有static變量和static方法。

在外部類中定義內部類的靜態成員變量，需要內部類為static class。

static是不允许用来修饰局部变量。

**域塊**：

**類**：

**6.java中怎樣使用jar包中的組件：**

使用classloader

创建一个带参数的jar包中的类对象。

public static Handler.Callback createKeyguardViewMediator\_HTC(Context context) {

final String packageName = "com.htc.lockscreen";

final String htcMediatorClass = "com.htc.lockscreen.keyguard.KeyguardViewMediator";

final int flags = Context.CONTEXT\_IGNORE\_SECURITY | Context.CONTEXT\_INCLUDE\_CODE;

Context targetContext = null;

try {

targetContext = context.createPackageContext(packageName, flags);

Class<?> clazz = getTargetClass(targetContext, htcMediatorClass);

if (clazz != null) {

Class<?>[] paramTypes = new Class[]{Context.class, Context.class};

Constructor<?> constructor = clazz.getConstructor((Class[])paramTypes);

Object[] params = new Object[]{context, targetContext};

Object instance = constructor.newInstance(params);

Handler.Callback mediator = null;

if (instance != null) {

if (instance instanceof Handler.Callback) {

mediator = (Handler.Callback)instance;

} else {

Log.d(TAG, "stranger interface");

}

}

if (mediator != null) {

return mediator;

}

}

} catch (NameNotFoundException ex) {

Log.d(TAG, "NameNotFoundException - " + ex.getMessage(), ex);

} catch (Exception ex) {

Log.d(TAG, "other exception - " + ex.getMessage(), ex);

}

return null;

}

1. java中的ThreadLocal的使用：

1.定义：

//线程第一次调用mThreadSession.get()会调用initialValue函数创建thread local对象。

private final ThreadLocal<SQLiteSession> mThreadSession = new ThreadLocal<SQLiteSession>() {

@Override

protected SQLiteSession initialValue() {

return createSession();

}

};

//线程第一次调用mThreadSession.get()会返回null，只能调用sThreadLocal.set(new Looper(quitAllowed));来创建thread local对象。

static final ThreadLocal<Looper> sThreadLocal = new ThreadLocal<Looper>();