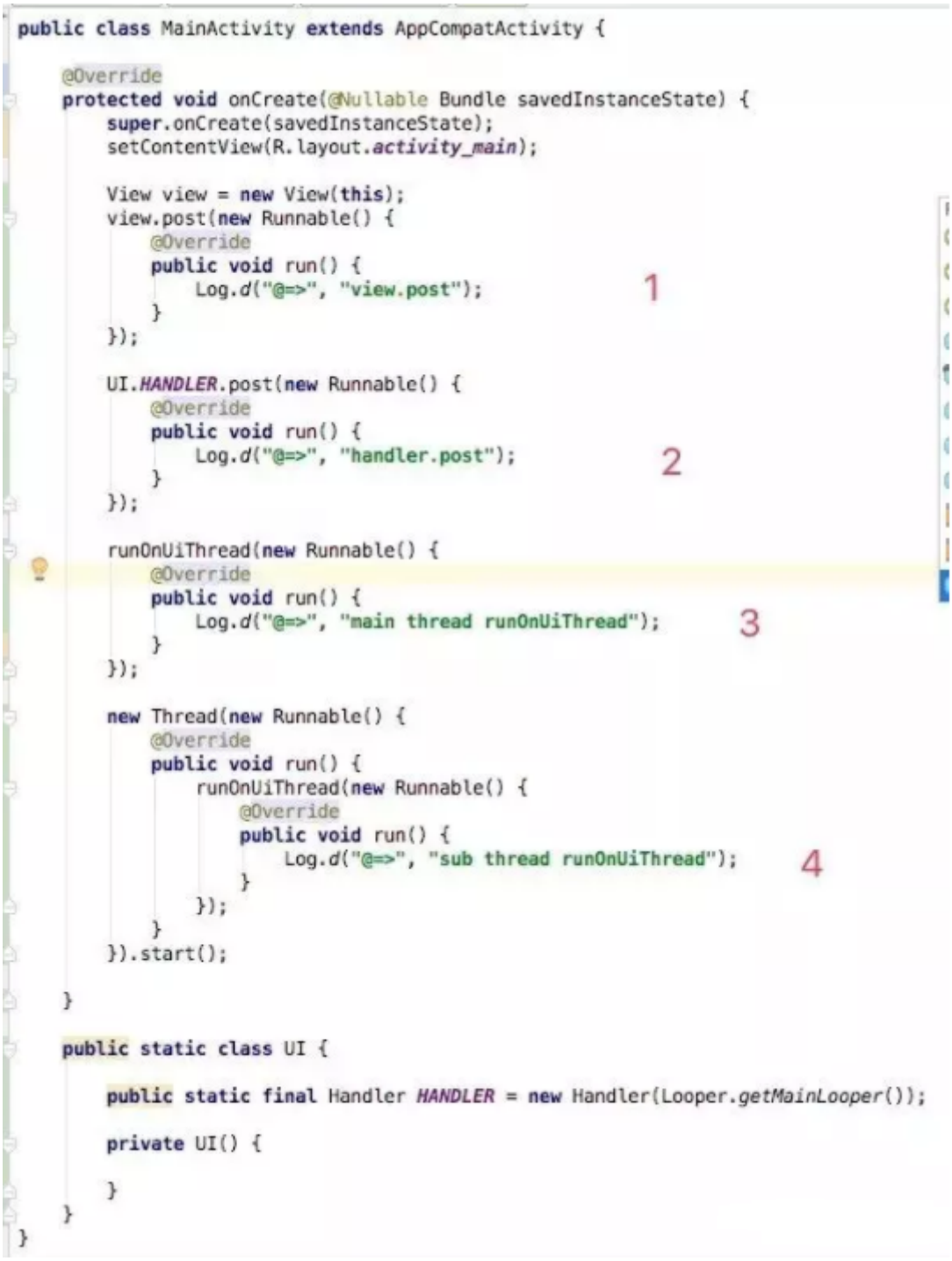
先看一下个实例程序：



```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        View view = new View(this);
        view.post(new Runnable() {
            @Override
            public void run() {
                Log.d("@=>", "view.post");
            }
        });

        UI.HANDLER.post(new Runnable() {
            @Override
            public void run() {
                Log.d("@=>", "handler.post");
            }
        });

        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Log.d("@=>", "main thread runOnUiThread");
            }
        });

        new Thread(new Runnable() {
            @Override
            public void run() {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Log.d("@=>", "sub thread runOnUiThread");
                    }
                });
            }
        }).start();

    }

    public static class UI {

        public static final Handler HANDLER = new Handler(Looper.getMainLooper());

        private UI() {

        }
    }
}
```

想得到答案，就得知道 runOnUiThread 、Handler.post、View.post 三者的区别。

从难易程度来讲，我们先说下 runOnUiThread 和 Handler.post 的区别，先看看 runOnUiThread 的源码：

```
/**
 * Runs the specified action on the UI thread. If the current thread is the UI
 * thread, then the action is executed immediately. If the current thread is
 * not the UI thread, the action is posted to the event queue of the UI thread.
 *
 * @param action the action to run on the UI thread
 */
public final void runOnUiThread(Runnable action) {
    if (Thread.currentThread() != mUiThread) {
        mHandler.post(action);
    } else {
        action.run();
    }
}
```

如果当前不是 UI 线程，那么由主线程的 Handler 扔个消息给 MessageQueue；如果当前是 UI 线程，则立刻执行。

知道这样的话，之前的那个题目就能回答一部分了，【234】的顺序应该是【324】，因为第一个 runOnUiThread 会立刻执行，而【24】就依据进入 MessageQueue 的先后顺序执行。

相信给【234】排序应该大部分人都错不了，那么第一个 View.post 呢？这个就略坑了。

题目设了个陷阱大家应该都看到了，View 实例化后并没有被加进任何能 attachToWindow 的 ViewGroup 中，这是一个很大的坑点。

我们先来看看 View.post 的源码，值得注意的是，View.post 的实现从 api24 开始有了变化，也就是说文章最开始的那道题在 Android 7.0（api24）和 7.0 以下的设备上输出的答案会不同。

先看看 api24 上的源码：

```
public boolean post(Runnable action) {
    final AttachInfo attachInfo = mAttachInfo;
    if (attachInfo != null) {
        return attachInfo.mHandler.post(action);
    }

    // Postpone the runnable until we know on which thread it needs to run.
    // Assume that the runnable will be successfully placed after attach.
    getRunQueue().post(action);
    return true;
}
```

再看看 api23 上的源码：

```
public boolean post(Runnable action) {
    final AttachInfo attachInfo = mAttachInfo;
    if (attachInfo != null) {
        return attachInfo.mHandler.post(action);
    }
    // Assume that post will succeed later
    ViewRootImpl.getRunQueue().post(action);
    return true;
}
```

看到区别了吧，当 mAttachInfo 为 null 的时候，源码变了。

我们先分析下相同部分的代码，也就是当 mAttachInfo 不为 null 的情况，我们看看它是啥时候被赋值的：

View#dispatchAttachedToWindow

```
void dispatchAttachedToWindow(AttachInfo info, int visibility) {
    //System.out.println("Attached! " + this);
    mAttachInfo = info;
    if (mOverlay != null) {
        mOverlay.getOverlayView().dispatchAttachedToWindow(info, visibility);
    }
    mWindowAttachCount++;
```

再看看 View 的dispatchAttachedToWindow 是哪位哥哥调的，显然是 ViewGroup：

ViewGroup#dispatchAttachedToWindow

```
@Override
void dispatchAttachedToWindow(AttachInfo info, int visibility) {
    mGroupFlags |= FLAG_PREVENT_DISPATCH_ATTACHED_TO_WINDOW;
    super.dispatchAttachedToWindow(info, visibility);
    mGroupFlags &= ~FLAG_PREVENT_DISPATCH_ATTACHED_TO_WINDOW;

    final int count = mChildrenCount;
    final View[] children = mChildren;
    for (int i = 0; i < count; i++) {
        final View child = children[i];
        child.dispatchAttachedToWindow(info,
                combineVisibility(visibility, child.getVisibility()));
    }
    final int transientCount = mTransientIndices == null ? 0 : mTransientIndices.size();
    for (int i = 0; i < transientCount; ++i) {
        View view = mTransientViews.get(i);
        view.dispatchAttachedToWindow(info,
                combineVisibility(visibility, view.getVisibility()));
    }
}
```

那么这个 ViewGroup 是谁？又是谁调用的?

先来看看是谁调用的：

ViewRootImpl#performTraversals

```
        viewVisibilityChanged = false;
        mLastConfiguration.setTo(host.getResources().getConfiguration());
        mLastSystemUiVisibility = mAttachInfo.mSystemUiVisibility;
        // Set the layout direction if it has not been set before (inherit is the default)
        if (mViewLayoutDirectionInitial == View.LAYOUT_DIRECTION_INHERIT) {
            host.setLayoutDirection(mLastConfiguration.getLayoutDirection());
        }
        host.dispatchAttachedToWindow(mAttachInfo, 0);
        mAttachInfo.mTreeObserver.dispatchOnWindowAttachedChange(true);
        dispatchApplyInsets(host);
        //Log.i(TAG, "Screen on initialized: " + attachInfo.mKeepScreenOn);
```

调用方知道了，那么 host 是谁？host 其实就是 Activity 的 DecorView，再往下就不在这篇文章里扯了，我们只要知道，要想让子 View 能调用 dispatchAttachedToWindow，那么一定得是 DecorView 的子 View，而我们【1】中创建的 View 并不是 DecorView 的子 View，因此【1】中的 View 的 dispatchAttachedToWindow 方法并不会被执行到，所以最开始说的 mAttachInfo 在我们这道题里是为 null 的。

理了半天，只是把 api23 和 api23 以上相同部分的源码解释了下，那么不相同的源码呢？也就是当 mAttachInfo 为 null 的情况，别急，慢慢道来。

上面说了，我们【1】的例子就会使 mAttachInfo 为 null，在 api23 及以下就会执行 ViewRootImpl.getRunQueue().post(action); 而 api24 开始会执行 getRunQueue().post(action); 。

来看看两者的区别，api23 中是使用的 ViewRootImpl 的 RunQueue：

```
static RunQueue getRunQueue() {
    RunQueue rq = sRunQueues.get();
    if (rq != null) {
        return rq;
    }
    rq = new RunQueue();
    sRunQueues.set(rq);
    return rq;
}
```

sRunQueues 是个静态的 ThreadLocal 对象，关于 ThreadLocal 也是面试常问题，之后单开文章讲解，这里要知道的就是，主线程用的是同一个 RunQueue，这个 RunQueue 里的 Runnable 啥时候会执行?

ViewRootImpl#performTraversals

```
// Execute enqueued actions on every traversal in case a detached view enque
getRunQueue().executeActions(mAttachInfo.mHandler);
```

performTraversals 方法是整个 View 的绘制流程的开始，走到这里时，api23 及一下的 View.post 代码会执行，也就是题目中的答案在 api23 及以下设备会是【3241】，那么 api24 以后呢？直接使用 View 里的 getRunQueue：

```
/**
 * Returns the queue of runnable for this view.
 *
 * @return the queue of runnables for this view
 */
private HandlerActionQueue getRunQueue() {
    if (mRunQueue == null) {
        mRunQueue = new HandlerActionQueue();
    }
    return mRunQueue;
}
```

他已经不是使用的主线程 RunQueue，而是自己这个对象里的，那自己的 RunQueue 啥时候执行呢？之前说了，在 dispatchAttachedToWindow 里会执行到，而【1】的 View 并不会执行 dispatchAttachedToWindow，也就是说在 api24 开始题目的答案是 【324】。

全部分析完了，正确答案应该是【324】/【3241】，需要区分 api 版本。