MAD CONF

Учимся на плохих примерах: SOLID в Android

Даниил Попов
Ведущий инженер

Avito

Немного обо мне

- Ведущий инженер Авито
- Разрабатываю Android-приложение
- Занимаюсь Android разработкой с 2011 года

Как появилась идея доклада?

Собеседование!

S Single Responsibility Principle

S Single Responsibility Principle

O Open-closed Principle

S Single Responsibility Principle

O Open-closed Principle

L Liskov Substitution Principle

S Single Responsibility Principle

O Open-closed Principle

L Liskov Substitution Principle

I Interface Segregation Principle

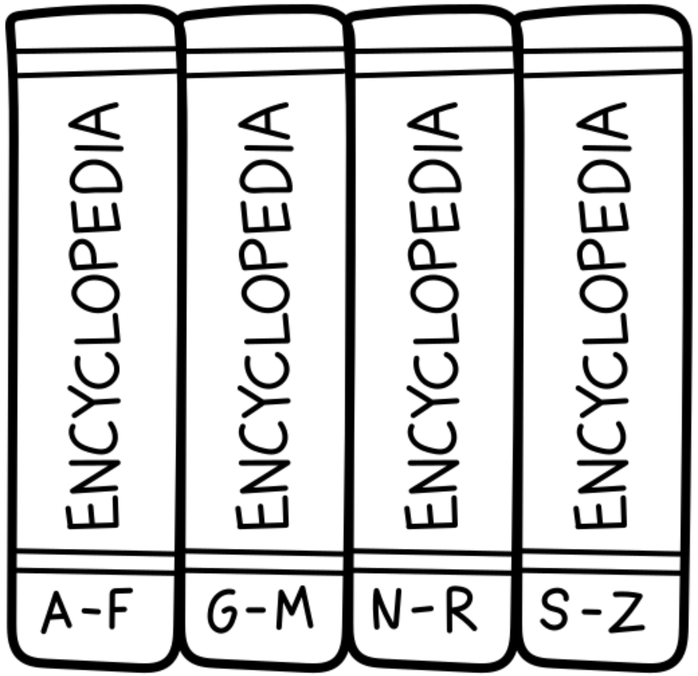
S Single Responsibility Principle

O Open-closed Principle

L Liskov Substitution Principle

I Interface Segregation Principle

D Dependency Inversion Principle


ENCYCLOPEDIA
A-F
ENCYCLOPEDIA
G-M
ENCYCLOPEDIA
N-R
ENCYCLOPEDIA
S-Z


ENCYCLOPEDIA
ENCYCLOPEDIA
ENCYCLOPEDIA
ENCYCLOPEDIA
A-F
DETECTED

Если не можешь быть хорошим примером, попробуй быть ужасающим предупреждением

Стивен Фрай

S Single Responsibility Principle

«

У класса должна быть лишь одна причина для изменения

»

Роберт С. Мартин

CONTEXT

stopService()

getDrawable()

startService()

getCacheDir()

getColor()

CONTEXT

openFileInput()

grantUriPermission()

enforcePermission()

startActivity()

getString()

fileList()

startActivities()

checkSelfPermission()

getExternalFilesDir()

getString()
getColor()
getDrawable()

checkSelfPermission()
grantUriPermission()
enforcePermission()

CONTEXT

getCacheDir()
fileList()
openFileInput()
getExternalFilesDir()

startActivity()
startActivities()
startService()
stopService()

```
fun onStartButtonClick() {
    context.componentLauncher.startActivity(MainActivity::class)
}
```

```
fun onStartButtonClick() {
    context.componentLauncher.startActivity(MainActivity::class)
}
```

```
fun onStartButtonClick() {
    context.componentLauncher.startActivity(MainActivity::class)
}


fun openUserDataFile(): FileOutputStream =
        context.files.openFileOutput("user.data", MODE_PRIVATE)
```

```
fun onStartButtonClick() {
    context.componentLauncher.startActivity(MainActivity::class)
}


fun openUserDataFile(): FileOutputStream =
        context.files.openFileOutput("user.data", MODE_PRIVATE)
```

```
fun onStartButtonClick() {
    context.componentLauncher.startActivity(MainActivity::class)
}


fun openUserDataFile(): FileOutputStream =
        context.files.openFileOutput("user.data", MODE_PRIVATE)


fun canReadCalendar() : Boolean =
        context.permissions.checkPermission(READ_CALENDAR)
```

```
fun onStartButtonClick() {
    context.componentLauncher.startActivity(MainActivity::class)
}


fun openUserDataFile(): FileOutputStream =
        context.files.openFileOutput("user.data", MODE_PRIVATE)


fun canReadCalendar() : Boolean =
        context.permissions.checkPermission(READ_CALENDAR)
```

ACTIVITY

ACTIVITY : CONTEXT

ACTIVITY : CONTEXT

onCreate() ..... onDestroy()

onStart() ..... onStop()

onResume() ..... onPause()

API level 14+

```
// Вариант №1
application.registerActivityLifecycleCallbacks(
    object : Application.ActivityLifecycleCallbacks {

        override fun onActivityCreated(activity: Activity,
                                       state: Bundle?) {}

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

API level 14+

```
// Вариант №1
application.registerActivityLifecycleCallbacks(
    object : Application.ActivityLifecycleCallbacks {

        override fun onActivityCreated(activity: Activity,
                                       state: Bundle?) {}

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

API level 14+

```
// Вариант №1
application.registerActivityLifecycleCallbacks(
    object : Application.ActivityLifecycleCallbacks {

        override fun onActivityCreated(activity: Activity,
                                       state: Bundle?) {}

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

API level 14+

```
// Вариант №1
application.registerActivityLifecycleCallbacks(
    object : Application.ActivityLifecycleCallbacks {

        override fun onActivityCreated(activity: Activity,
                                       state: Bundle?) {}

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

Jetpack Lifecycle

```
// Вариант №2
class MyObserver : LifecycleObserver {

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun connectListener() {
        // ...
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    fun disconnectListener() {
        // ...
    }
}
```

Jetpack Lifecycle

```
// Вариант №2
class MyObserver : LifecycleObserver {

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun connectListener() {
        // ...
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    fun disconnectListener() {
        // ...
    }
}
```

Jetpack Lifecycle

```
// Вариант №2
class MyObserver : LifecycleObserver {

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun connectListener() {
        // ...
    }

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    fun disconnectListener() {
        // ...
    }
}
```

Open-closed principle

Программные сущности должны быть открыты для расширения, но закрыты для модификации

Бертран Мейер

Activity

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    if (who == null) {
        // ...
    } else if (who.startsWith(REQUEST_PERMISSIONS_WHO_PREFIX)) {
        // ...
    } else if (who.startsWith("@android:view:")) {
        // ...
    } else if (who.startsWith(AUTO_FILL_AUTH_WHO_PREFIX)) {
        // ...
    } else {
        // ...
    }
}
```

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    if (who == null) {
        // ...
    } else if (who.startsWith(REQUEST_PERMISSIONS_WHO_PREFIX)) {
        // ...
    } else if (who.startsWith("@android:view:")) {
        // ...
    } else if (who.startsWith(AUTO_FILL_AUTH_WHO_PREFIX)) {
        // ...
    } else {
        // ...
    }
}
```

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    if (who == null) {
        // ...
    } else if (who.startsWith(REQUEST_PERMISSIONS_WHO_PREFIX)) {
        // ...
    } else if (who.startsWith("@android:view:")) {
        // ...
    } else if (who.startsWith(AUTO_FILL_AUTH_WHO_PREFIX)) {
        // ...
    } else {
        // ...
    }
}
```

Как можно сделать по-другому?

```
interface ActivityResultDispatcher {

    boolean canDispatchFrom(String who);

    void dispatchResult(String who,
                        int requestCode,
                        int resultCode,
                        Intent data);
}
```

```
interface ActivityResultDispatcher {

    boolean canDispatchFrom(String who);

    void dispatchResult(String who,
                        int requestCode,
                        int resultCode,
                        Intent data);
}
```

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    // ...
    for (ActivityResultDispatcher dispatcher : dispatchers) {
        if (dispatcher.canDispatchFrom(who)) {
            dispatcher.dispatchResult(who, requestCode, resultCode, data);
        }
    }
    // ...
}
```

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    // ...
    for (ActivityResultDispatcher dispatcher : dispatchers) {
        if (dispatcher.canDispatchFrom(who)) {
            dispatcher.dispatchResult(who, requestCode, resultCode, data);
        }
    }
    // ...
}
```

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    // ...
    for (ActivityResultDispatcher dispatcher : dispatchers) {
        if (dispatcher.canDispatchFrom(who)) {
            dispatcher.dispatchResult(who, requestCode, resultCode, data);
        }
    }
    // ...
}
```

```
void dispatchActivityResult(String who, int requestCode, int resultCode,
                            Intent data, String reason) {
    // ...
    for (ActivityResultDispatcher dispatcher : dispatchers) {
        if (dispatcher.canDispatchFrom(who)) {
            dispatcher.dispatchResult(who, requestCode, resultCode, data);
        }
    }
    // ...
}
```

View

```java
private void onProvideStructureForAssistOrAutofill(ViewStructure structure,
                                                   boolean forAutofill,
                                                   int flags) {
    // ...
    structure.setVisibility(getVisibility());
    structure.setEnabled(isEnabled());
    if (isClickable()) { structure.setClickable(true); }
    if (isFocusable()) { structure.setFocusable(true); }
    if (isFocused()) { structure.setFocused(true); }
    if (isAccessibilityFocused()) { structure.setAccessibilityFocused(true); }
    if (isSelected()) { structure.setSelected(true); }
    if (isActivated()) { structure.setActivated(true); }
    if (isLongClickable()) { structure.setLongClickable(true); }
    if (this instanceof Checkable) { /* ... */}
    if (isOpaque()) { structure.setOpaque(true); }
    if (isContextClickable()) { structure.setContextClickable(true); }
    //...
}
```

```java
private void onProvideStructureForAssistOrAutofill(ViewStructure structure,
                                                   boolean forAutofill,
                                                   int flags) {
    // ...
    structure.setVisibility(getVisibility());
    structure.setEnabled(isEnabled());
    if (isClickable()) { structure.setClickable(true); }
    if (isFocusable()) { structure.setFocusable(true); }
    if (isFocused()) { structure.setFocused(true); }
    if (isAccessibilityFocused()) { structure.setAccessibilityFocused(true); }
    if (isSelected()) { structure.setSelected(true); }
    if (isActivated()) { structure.setActivated(true); }
    if (isLongClickable()) { structure.setLongClickable(true); }
    if (this instanceof Checkable) { /* ... */}
    if (isOpaque()) { structure.setOpaque(true); }
    if (isContextClickable()) { structure.setContextClickable(true); }
    //...
}
```

```java
private void onProvideStructureForAssistOrAutofill(ViewStructure structure,
                                                   boolean forAutofill,
                                                   int flags) {
    // ...
    structure.setVisibility(getVisibility());
    structure.setEnabled(isEnabled());
    if (isClickable()) { structure.setClickable(true); }
    if (isFocusable()) { structure.setFocusable(true); }
    if (isFocused()) { structure.setFocused(true); }
    if (isAccessibilityFocused()) { structure.setAccessibilityFocused(true); }
    if (isSelected()) { structure.setSelected(true); }
    if (isActivated()) { structure.setActivated(true); }
    if (isLongClickable()) { structure.setLongClickable(true); }
    if (this instanceof Checkable) { /* ... */}
    if (isOpaque()) { structure.setOpaque(true); }
    if (isContextClickable()) { structure.setContextClickable(true); }
    //...
}
```

```java
private void onProvideStructureForAssistOrAutofill(ViewStructure structure,
                                                   boolean forAutofill,
                                                   int flags) {
    // ...
    structure.setVisibility(getVisibility());
    structure.setEnabled(isEnabled());
    if (isClickable()) { structure.setClickable(true); }
    if (isFocusable()) { structure.setFocusable(true); }
    if (isFocused()) { structure.setFocused(true); }
    if (isAccessibilityFocused()) { structure.setAccessibilityFocused(true); }
    if (isSelected()) { structure.setSelected(true); }
    if (isActivated()) { structure.setActivated(true); }
    if (isLongClickable()) { structure.setLongClickable(true); }
    if (this instanceof Checkable) { /* ... */}
    if (isOpaque()) { structure.setOpaque(true); }
    if (isContextClickable()) { structure.setContextClickable(true); }
    //...
}
```

```
interface ViewStructureFiller {
    void fill(ViewStructure structure, boolean forAutofill, int flags);
}
```

```
interface ViewStructureFiller {
    void fill(ViewStructure structure, boolean forAutofill, int flags);
}
```

Ещё немного о View

```java
public final void setLeft(int left) {
    if (left != mLeft) {
        // ...
        mLeft = left;
        // ...
        sizeChange(mRight - mLeft, height, oldWidth, height);
        //...
    }
}
```

```java
public final void setLeft(int left) {
    if (left != mLeft) {
        // ...
        mLeft = left;
        // ...
        sizeChange(mRight - mLeft, height, oldWidth, height);
        //...
    }
}
```

```java
public final void setLeft(int left) {
    if (left != mLeft) {
        // ...
        mLeft = left;
        // ...
        sizeChange(mRight - mLeft, height, oldWidth, height);
        //...
    }
}
```

```java
public final void setLeft(int left) {
    if (left != mLeft) {
        // ...
        mLeft = left;
        // ...
        sizeChange(mRight - mLeft, height, oldWidth, height);
        onLeftChange(left)
        //...
    }
}
```

```java
public final void setLeft(int left) {
    if (left != mLeft) {
        // ...
        mLeft = left;
        // ...
        sizeChange(mRight - mLeft, height, oldWidth, height);
        onLeftChange(left)
        //...
    }
}
```

L Liskov substitution principle

Абстракции можно заменить конкретными реализациями и ничего не сломается

Диванный эксперт

```
fun foo(list: List<String>) {
    // ...
}

fun bar(map: Map<Int, Boolean>) {
    // ...
}

fun foobar(view: View) {
    // ...
}
```

```kotlin
fun foo(list: List<String>) {
    // ...
}

fun bar(map: Map<Int, Boolean>) {
    // ...
}

fun foobar(view: View) {
    // ...
}
```

```kotlin
fun foo(list: ArrayList<String>) {
    // ...
}

fun bar(map: TreeMap<Int, Boolean>) {
    // ...
}

fun foobar(view: Button) {
    // ...
}
```

```kotlin
fun foo(list: List<String>) {
    // ...
}

fun bar(map: Map<Int, Boolean>) {
    // ...
}

fun foobar(view: View) {
    // ...
}
```

```kotlin
fun foo(list: ArrayList<String>) {
    // ...
}

fun bar(map: TreeMap<Int, Boolean>) {
    // ...
}

fun foobar(view: Button) {
    // ...
}
```

```kotlin
fun foo(list: List<String>) {
    // ...
}

fun bar(map: Map<Int, Boolean>) {
    // ...
}

fun foobar(view: View) {
    // ...
}
```

```kotlin
fun foo(list: ArrayList<String>) {
    // ...
}

fun bar(map: TreeMap<Int, Boolean>) {
    // ...
}

fun foobar(view: Button) {
    // ...
}
```

```kotlin
fun foo(list: List<String>) {
    // ...
}

fun bar(map: Map<Int, Boolean>) {
    // ...
}

fun foobar(view: View) {
    // ...
}
```

```kotlin
fun foo(list: ArrayList<String>) {
    // ...
}

fun bar(map: TreeMap<Int, Boolean>) {
    // ...
}

fun foobar(view: Button) {
    // ...
}
```

unmodifiableList()
unmodifiableSet()
unmodifiableMap()

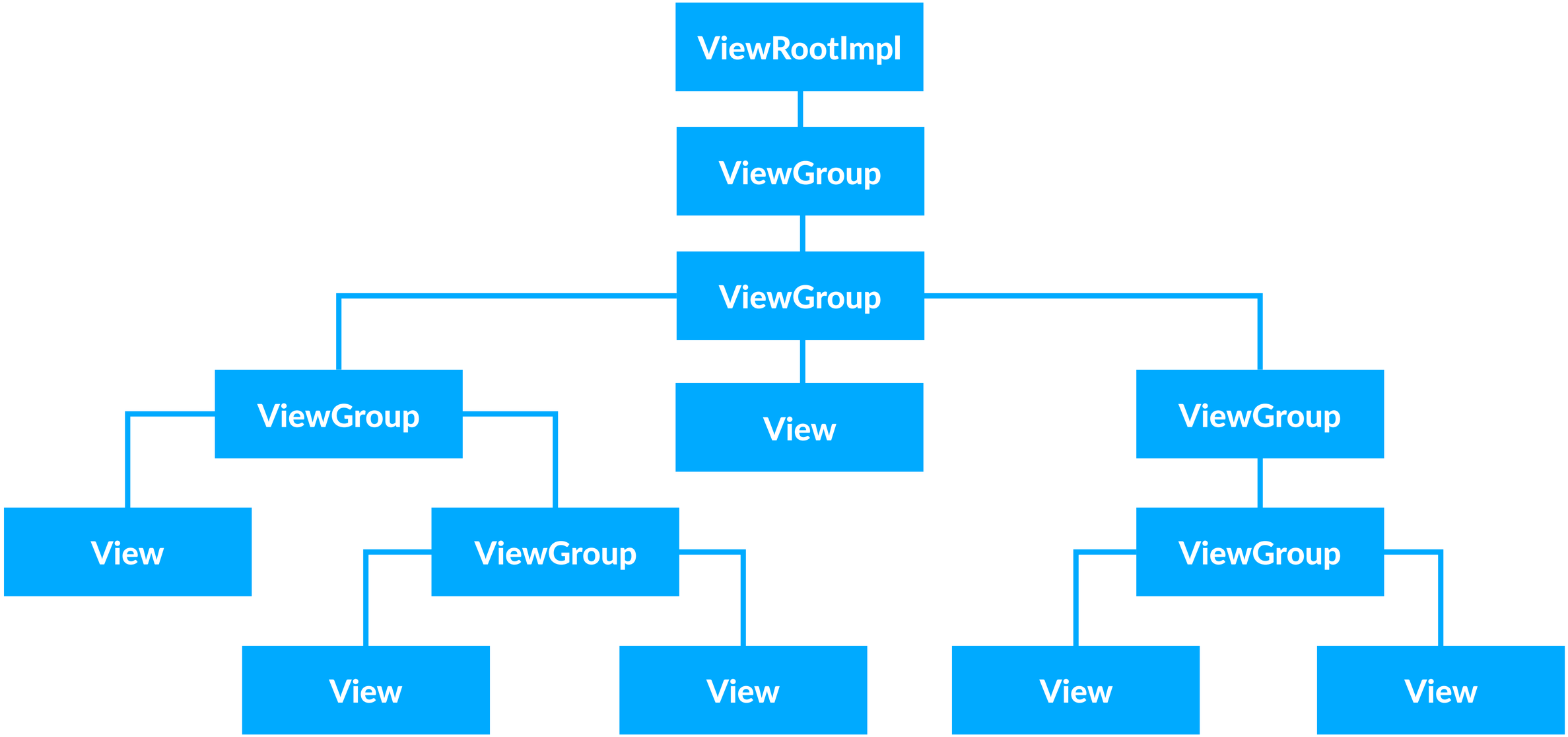
List/MutableList
Set/MutableSet
Map/MutableMap

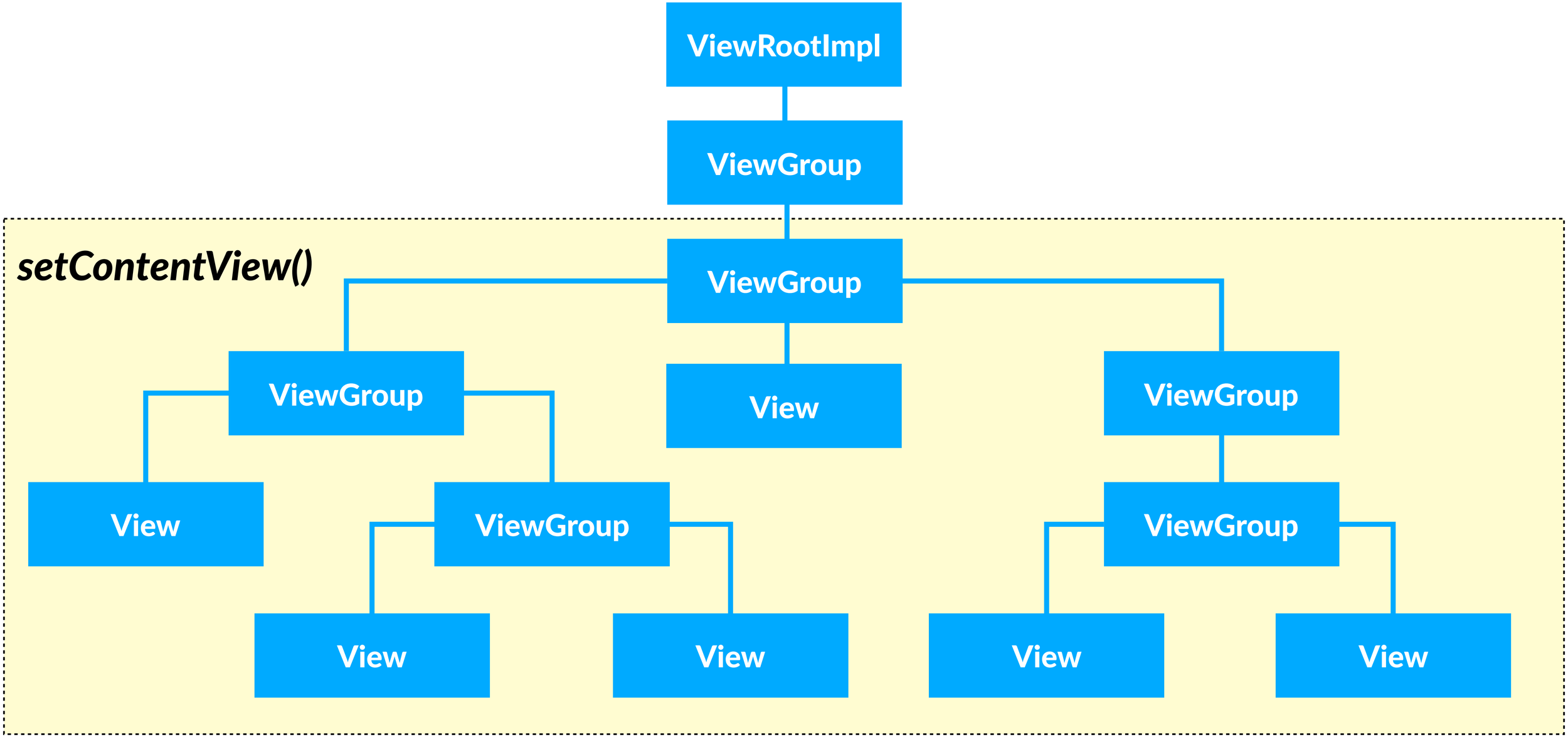
```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```

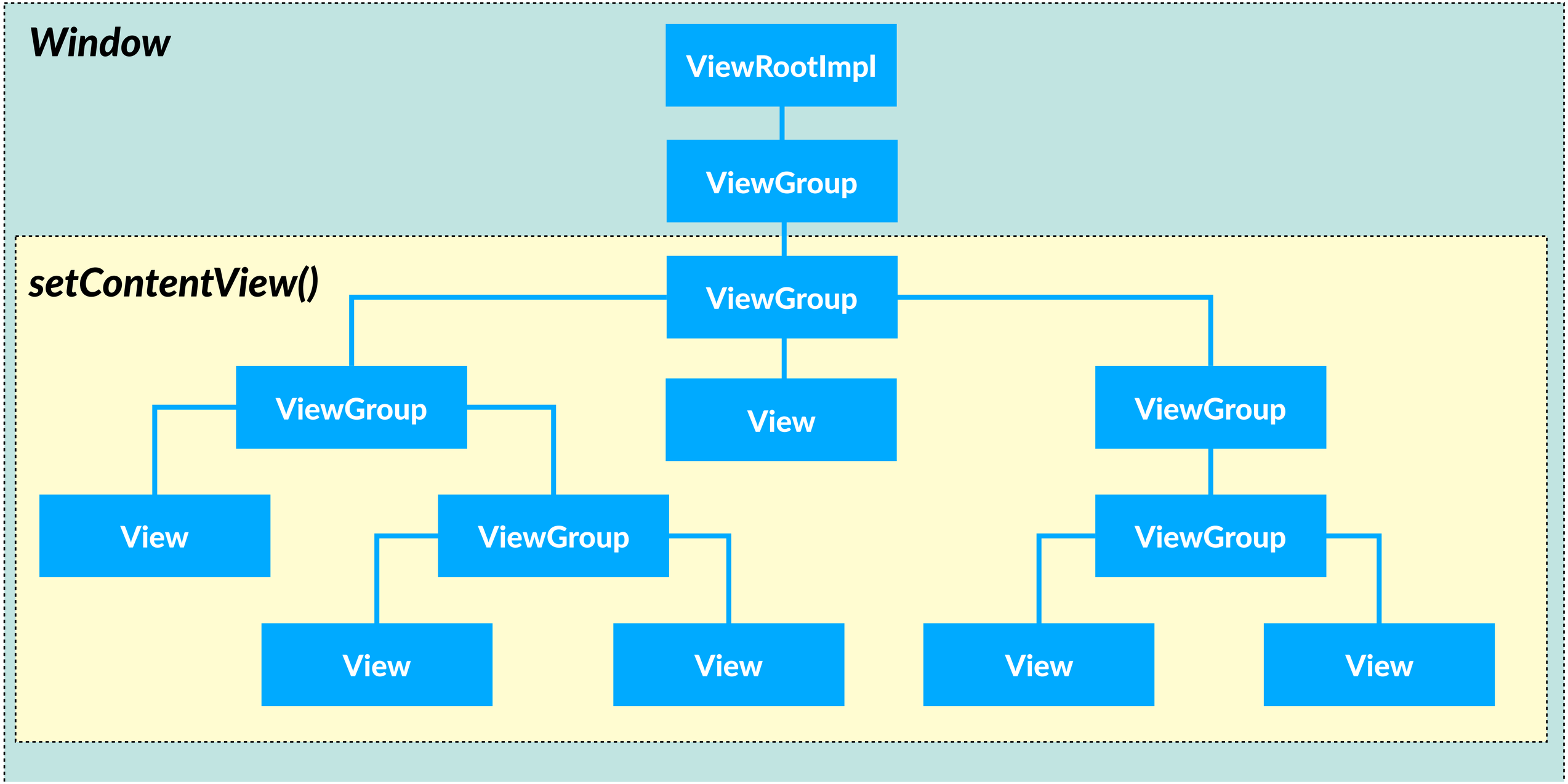
```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```

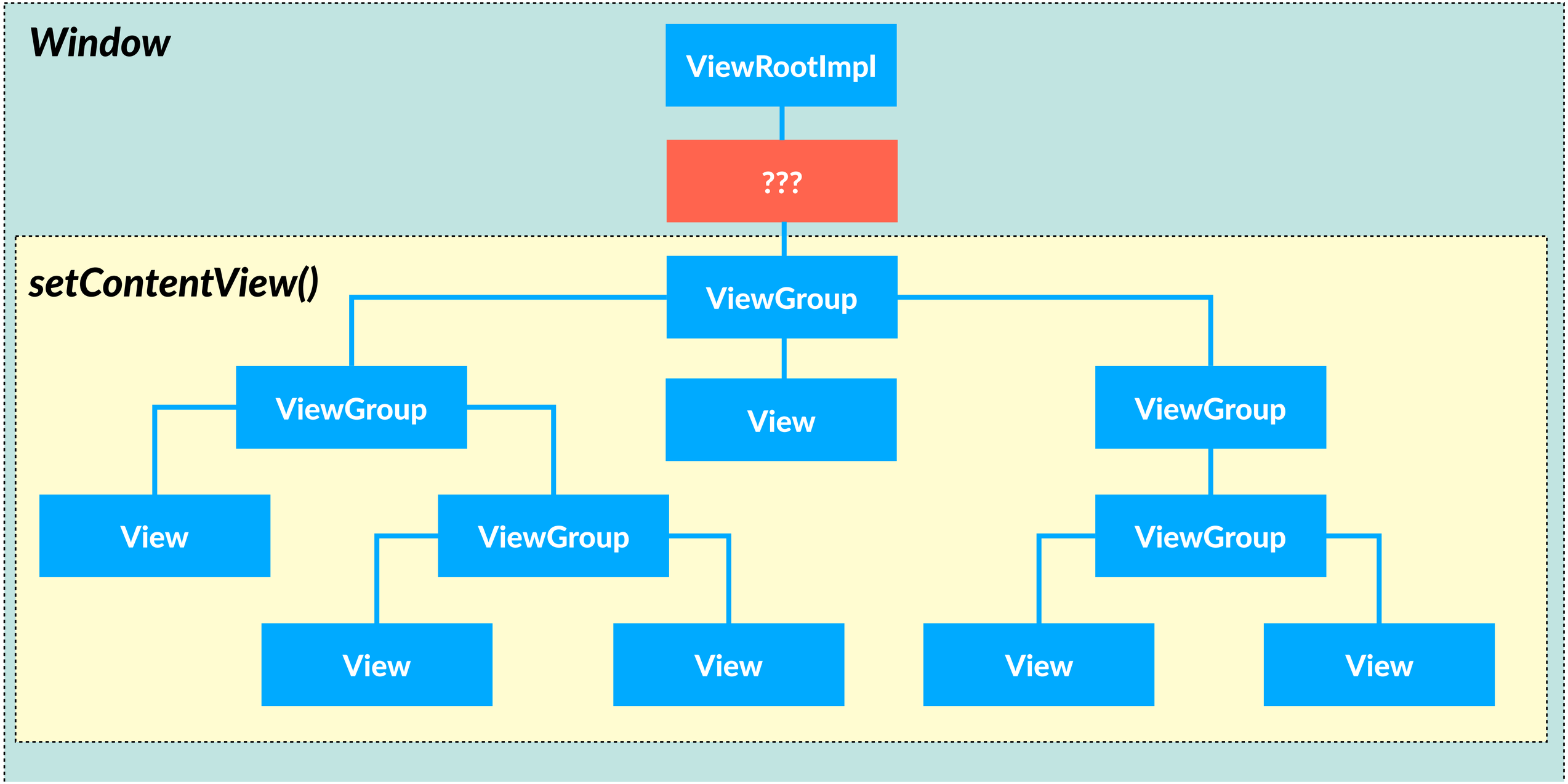
```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```

```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```


ViewRootImpl
ViewGroup
ViewGroup
ViewGroup
View
ViewGroup
View
ViewGroup
ViewGroup
View
View
View
View


ViewRootImpl
ViewGroup
setContentView()
ViewGroup
ViewGroup
View
ViewGroup
View
ViewGroup
ViewGroup
View
View
View
View


Window
ViewRootImpl
ViewGroup
setContentView()
ViewGroup
ViewGroup
View
ViewGroup
View
ViewGroup
ViewGroup
View
View
View
View


Window
ViewRootImpl
???
setContentView()
ViewGroup
ViewGroup
View
ViewGroup
View
ViewGroup
ViewGroup
View
View
View
View

ResourcesImpl

```
TypedArray obtainStyledAttributes(Resources.Theme wrapper,
                                  AttributeSet set,
                                  int[] attrs,
                                  int defStyleAttr,
                                  int defStyleRes) {
    synchronized (mKey) {
        // ...
        final XmlBlock.Parser parser = (XmlBlock.Parser) set;
        // ...
    }
}
```

```
TypedArray obtainStyledAttributes(Resources.Theme wrapper,
                                  AttributeSet set,
                                  int[] attrs,
                                  int defStyleAttr,
                                  int defStyleRes) {
    synchronized (mKey) {
        // ...
        final XmlBlock.Parser parser = (XmlBlock.Parser) set;
        // ...
    }
}
```

```
TypedArray obtainStyledAttributes(Resources.Theme wrapper,
                                  AttributeSet set,
                                  int[] attrs,
                                  int defStyleAttr,
                                  int defStyleRes) {
    synchronized (mKey) {
        // ...
        final XmlBlock.Parser parser = (XmlBlock.Parser) set;
        // ...
    }
}
```

I Interface segregation principle

Много узкоспециализированных интерфейсов лучше, чем один интерфейс общего назначения

Ребята у кулера

```java
public interface TextWatcher extends NoCopySpan {

    public void beforeTextChanged(CharSequence s,
                                  int start,
                                  int count,
                                  int after);

    public void onTextChanged(CharSequence s,
                              int start,
                              int before,
                              int count);

    public void afterTextChanged(Editable s);

}
```

```java
public interface TextWatcher extends NoCopySpan {

    public void beforeTextChanged(CharSequence s,
                                  int start,
                                  int count,
                                  int after);

    public void onTextChanged(CharSequence s,
                              int start,
                              int before,
                              int count);

    public void afterTextChanged(Editable s);

}
```

```java
public interface OnTextChangedListener {

    public void beforeTextChanged(CharSequence s,
                                  int start,
                                  int count,
                                  int after);

    public void onTextChanged(CharSequence s,
                              int start,
                              int before,
                              int count);

    public void afterTextChanged(Editable s);

}
```

АНИМАЦИИ

АНИМАЦИИ

```java
public static interface AnimatorListener {

    default void onAnimationStart(Animator animation, boolean isReverse) {
        onAnimationStart(animation);
    }

    default void onAnimationEnd(Animator animation, boolean isReverse) {
        onAnimationEnd(animation);
    }

    void onAnimationStart(Animator animation);

    void onAnimationEnd(Animator animation);

    void onAnimationCancel(Animator animation);

    void onAnimationRepeat(Animator animation);

}
```

```
public static interface AnimatorListener {

    default void onAnimationStart(Animator animation, boolean isReverse) {
        onAnimationStart(animation);
    }

    default void onAnimationEnd(Animator animation, boolean isReverse) {
        onAnimationEnd(animation);
    }

    void onAnimationStart(Animator animation);

    void onAnimationEnd(Animator animation);

    void onAnimationCancel(Animator animation);

    void onAnimationRepeat(Animator animation);

}
```

```
public static interface AnimatorListener {

    default void onAnimationStart(Animator animation, boolean isReverse) {
        onAnimationStart(animation);
    }

    default void onAnimationEnd(Animator animation, boolean isReverse) {
        onAnimationEnd(animation);
    }

    void onAnimationStart(Animator animation);

    void onAnimationEnd(Animator animation);

    void onAnimationCancel(Animator animation);

    void onAnimationRepeat(Animator animation);

}
```

```java
public abstract class AnimatorListenerAdapter implements Animator.AnimatorListener,
        Animator.AnimatorPauseListener {

    @Override
    public void onAnimationCancel(Animator animation) {}

    @Override
    public void onAnimationEnd(Animator animation) {}

    @Override
    public void onAnimationRepeat(Animator animation) {}

    @Override
    public void onAnimationStart(Animator animation) {}

    @Override
    public void onAnimationPause(Animator animation) {}

    @Override
    public void onAnimationResume(Animator animation) {}

}
```

```java
public abstract class AnimatorListenerAdapter implements Animator.AnimatorListener,
        Animator.AnimatorPauseListener {

    @Override
    public void onAnimationCancel(Animator animation) {}

    @Override
    public void onAnimationEnd(Animator animation) {}

    @Override
    public void onAnimationRepeat(Animator animation) {}

    @Override
    public void onAnimationStart(Animator animation) {}

    @Override
    public void onAnimationPause(Animator animation) {}

    @Override
    public void onAnimationResume(Animator animation) {}

}
```

```
public static interface AnimatorPauseListener {
    void onAnimationPause(Animator animation);
    void onAnimationResume(Animator animation);
}
```

```java
public interface AnimatorStartEndListener {
    void onAnimationStart(Animator animation, boolean isReverse);
    void onAnimationEnd(Animator animation, boolean isReverse);
}
public interface AnimatorCancelListener {
    void onAnimationCancel(Animator animation);
}
public interface AnimatorRepeatListener {
    void onAnimationRepeat(Animator animation);
}
```

D Dependency inversion principle

Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Роберт С. Мартин

```java
public abstract class FragmentHostCallback<E> extends FragmentContainer {
    // ...

    final FragmentManagerImpl mFragmentManager = new FragmentManagerImpl();

    // ...

    FragmentManagerImpl getFragmentManagerImpl() {
        return mFragmentManager;
    }

    // ...
}
```

```java
public abstract class FragmentHostCallback<E> extends FragmentContainer {
    // ...

    final FragmentManagerImpl mFragmentManager = new FragmentManagerImpl();

    // ...

    FragmentManagerImpl getFragmentManagerImpl() {
        return mFragmentManager;
    }

    // ...
}
```

```java
public abstract class FragmentHostCallback<E> extends FragmentContainer {
    // ...

    final FragmentManagerImpl mFragmentManager = new FragmentManagerImpl();

    // ...

    FragmentManagerImpl getFragmentManagerImpl() {
        return mFragmentManager;
    }

    // ...
}
```

```java
public class Fragment implements ...
    // ...

    FragmentManagerImpl mFragmentManager;
    FragmentManagerImpl mChildFragmentManager;

    // ...

    void instantiateChildFragmentManager() {
        // ...
        mChildFragmentManager = new FragmentManagerImpl();
        mChildFragmentManager.attachController(mHost, new FragmentContainer() {
            // ...
        }, this);
    }

    // ...
}
```

```java
public class Fragment implements ...
    // ...

    FragmentManagerImpl mFragmentManager;
    FragmentManagerImpl mChildFragmentManager;

    // ...

    void instantiateChildFragmentManager() {
        // ...
        mChildFragmentManager = new FragmentManagerImpl();
        mChildFragmentManager.attachController(mHost, new FragmentContainer() {
            // ...
        }, this);
    }

    // ...
}
```

```java
public class Fragment implements ...
    // ...

    FragmentManagerImpl mFragmentManager;
    FragmentManagerImpl mChildFragmentManager;

    // ...

    void instantiateChildFragmentManager() {
        // ...
        mChildFragmentManager = new FragmentManagerImpl();
        mChildFragmentManager.attachController(mHost, new FragmentContainer() {
            // ...
        }, this);
    }

    // ...
}
```

```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```

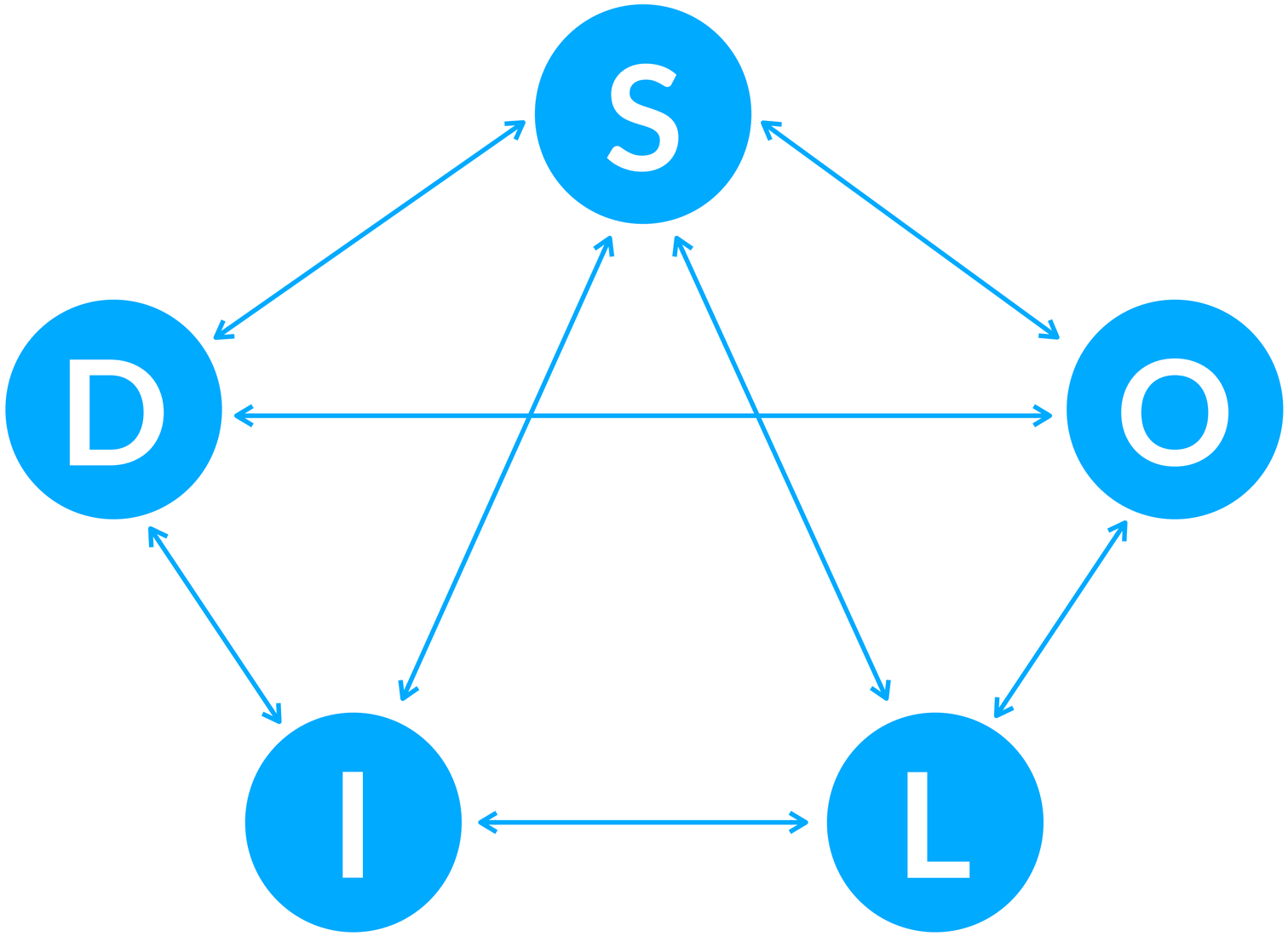
```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```

```java
public void transformFromViewToWindowSpace(int[] inOutLocation) {
    // ...
    ViewParent viewParent = mParent;
    while (viewParent instanceof View) {
        final View view = (View) viewParent;
        // ...
        position[0] += view.mLeft;
        position[1] += view.mTop;

        viewParent = view.mParent;
    }

    if (viewParent instanceof ViewRootImpl) {
        // *cough*
        final ViewRootImpl vr = (ViewRootImpl) viewParent;
        position[1] -= vr.mCurScrollY;
    }
    // ...
}
```

```
// Вариант №1
application.registerActivityLifecycleCallbacks(
    object : Application.ActivityLifecycleCallbacks {

        override fun onActivityCreated(activity: Activity,
                                       state: Bundle?) {}

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

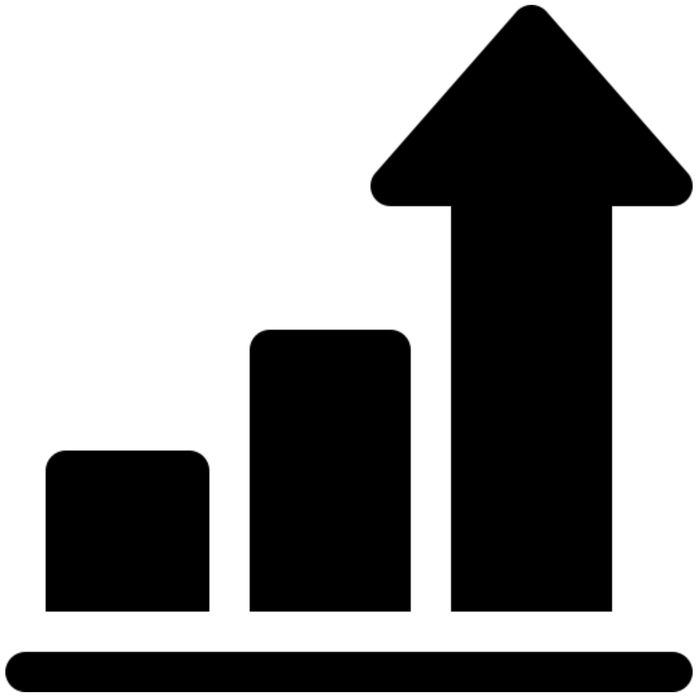
```
// Вариант №1
application.registerActivityLifecycleCallbacks(
    object : Application.ActivityLifecycleCallbacks {

        override fun onActivityCreated(activity: Activity,
                                       state: Bundle?) {}

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

→ **еще 18 методов!**


S
D
O
I
L

S O L I D

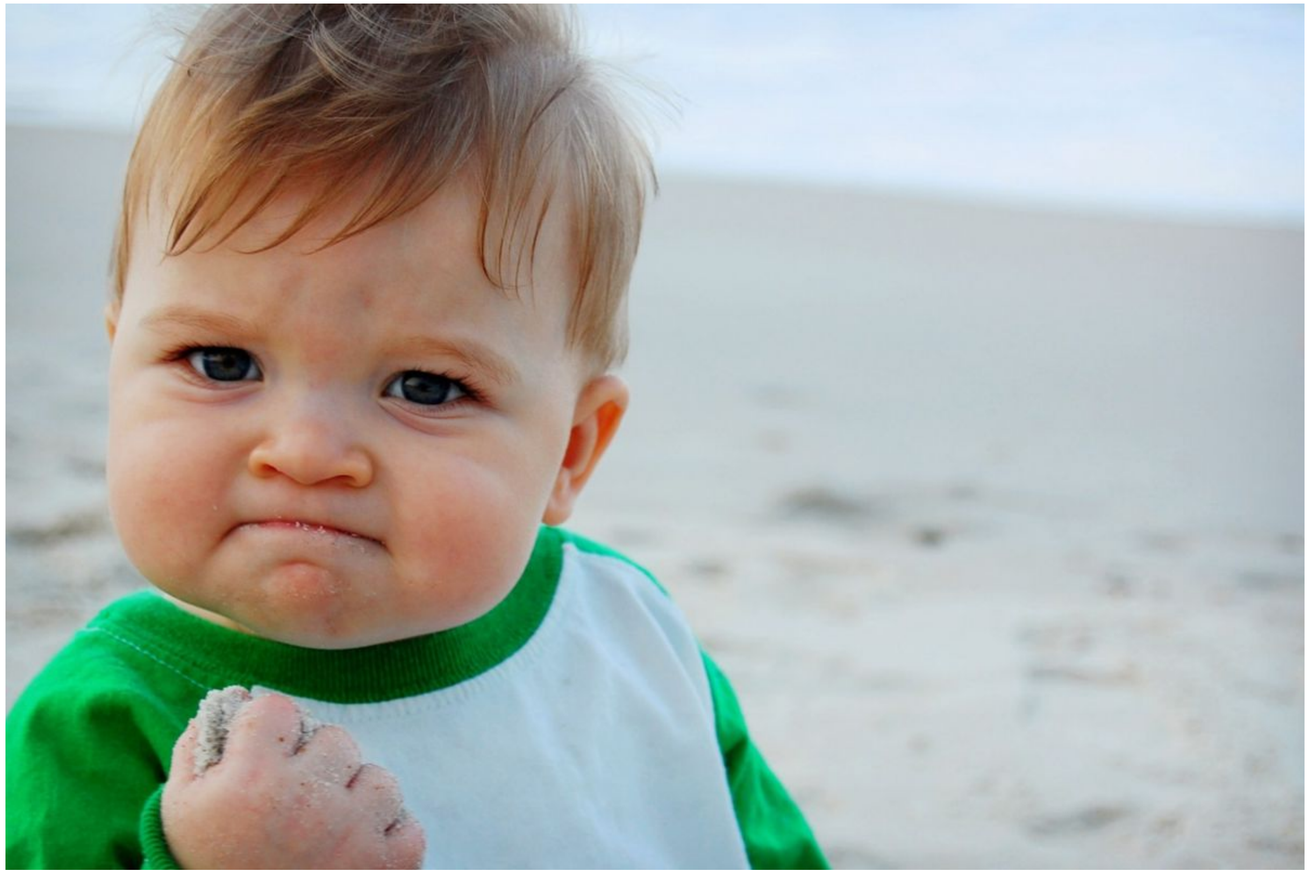
НЕ СТРОГИЕ ПРАВИЛА



S O L I D

НЕ СЕРЕБРЯНАЯ ПУЛЯ

КОГДА ВСПОМНИЛ И ПРИМЕНИЛ



ПРИНЦИПЫ SOLID

Спасибо

dalpopov@avito.ru
http://t.me/int02h
https://twitter.com/int02h

Попов Даниил
Ведущий инженер

Avito