Даниил Попов
Ведущий инженер

Учимся на плохих примерах: SOLID в Android

Стачка **ИННОПОЛИС** innopolis university

12/13 октября
Татарстан, Иннополис

Немного обо мне

- Ведущий инженер Авито
- Разрабатываю Android-приложение
- Занимаюсь Android разработкой с 2011 года
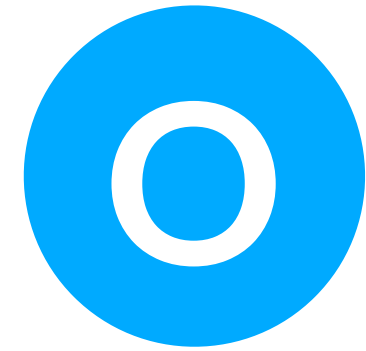
Как появилась идея доклада?

Собеседование!

5

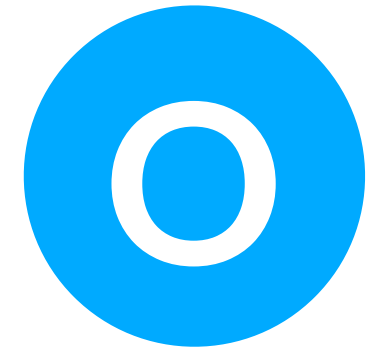**S** Single Responsibility Principle
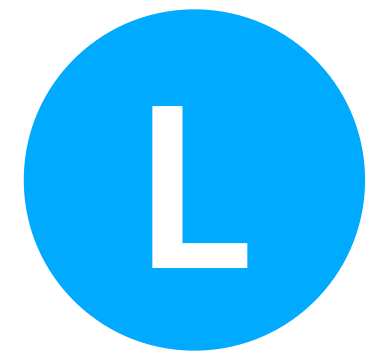
S Single Responsibility Principle

O Open-closed Principle

S Single Responsibility Principle

O Open-closed Principle

L Liskov Substitution Principle

S Single Responsibility Principle
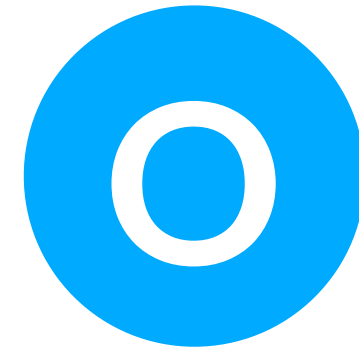
O Open-closed Principle
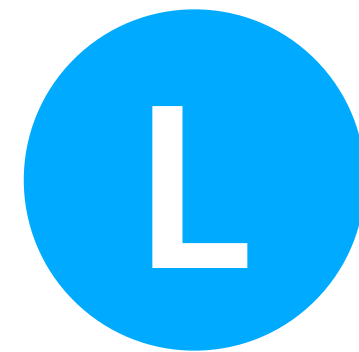
L Liskov Substitution Principle

I Interface Segregation Principle

S Single Responsibility Principle
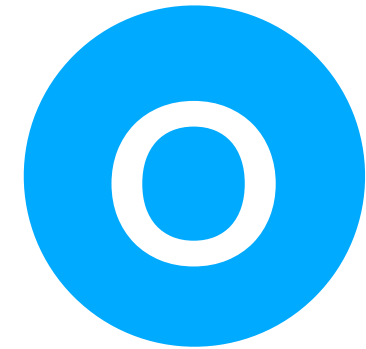
O Open-closed Principle

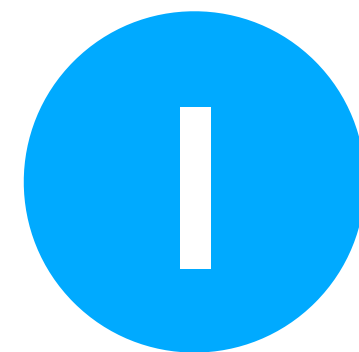**L** Liskov Substitution Principle
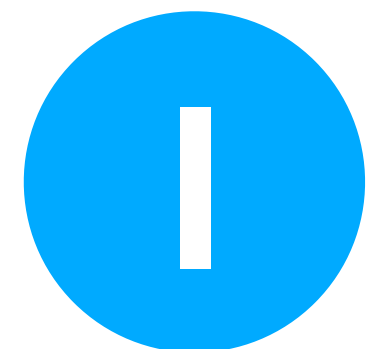
I Interface Segregation Principle
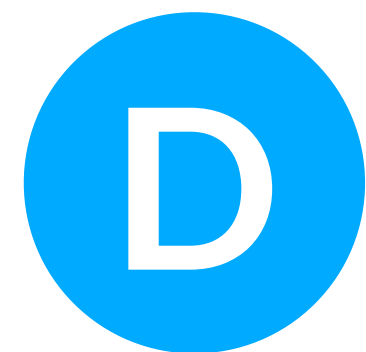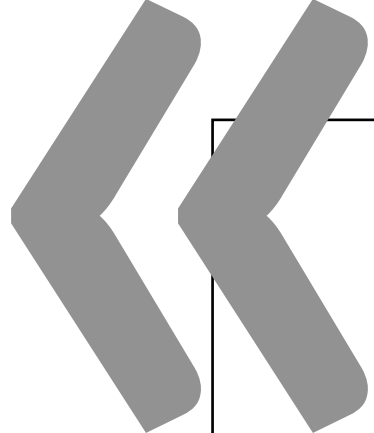
D Dependency Inversion Principle
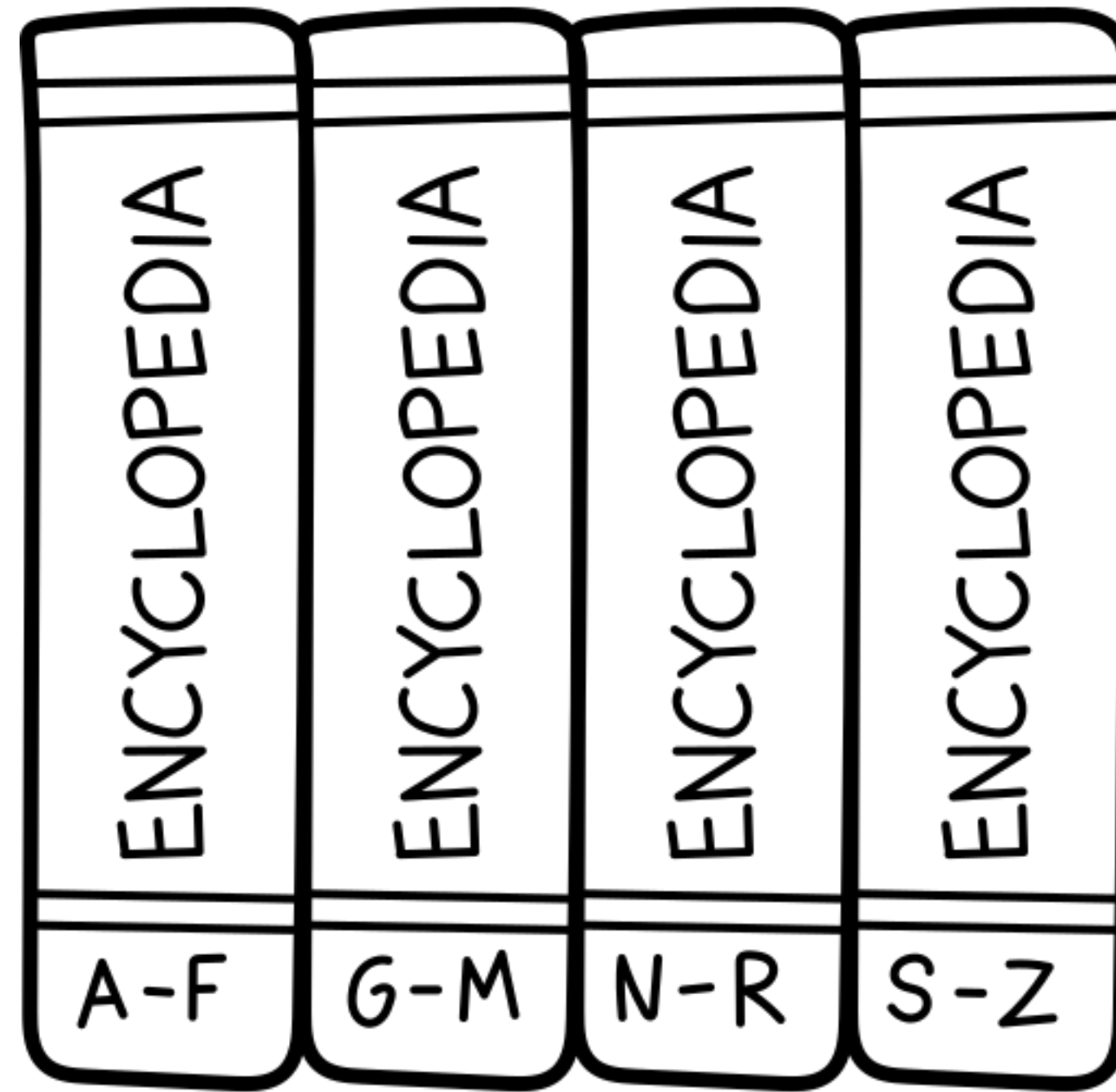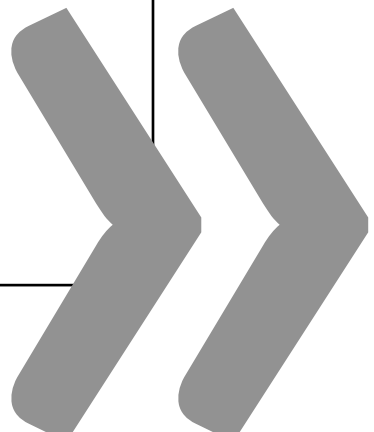
« Если не можешь быть хорошим примером, попробуй быть ужасающим предупреждением »
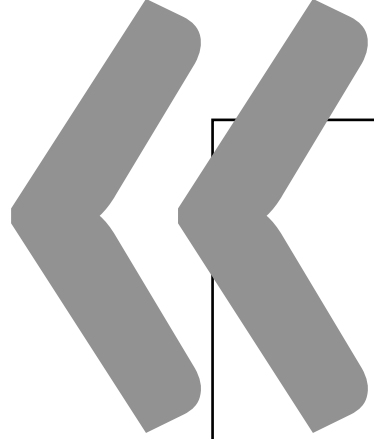
Стивен Фрай
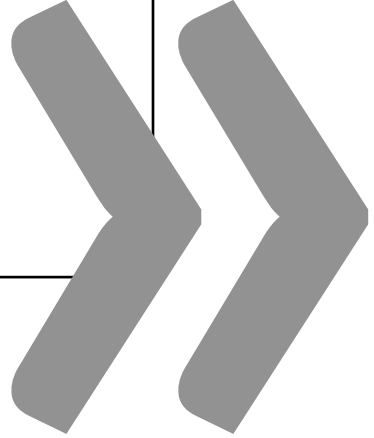
Single Responsibility Principle

У класса должна быть лишь одна причина для изменения

Роберт С. Мартин

stopService()

startService()

getDrawable()

getColor()

getCacheDir()

CONTEXT

openFileInput()

grantUriPermission()

enforcePermission()

startActivity()

getString()

fileList()

startActivities()

checkSelfPermission()

getExternalFilesDir()

11

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

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

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

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

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


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

```kotlin
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)
```

```kotlin
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()**

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

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


        override fun onActivityStarted(activity: Activity) {}


        override fun onActivityResumed(activity: Activity) {}


        // ...
    })
```

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

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

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

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

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

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

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

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

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

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


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


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

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

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


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

14

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

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


    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    fun disconnectListener() {
        // ...
    }
}
```
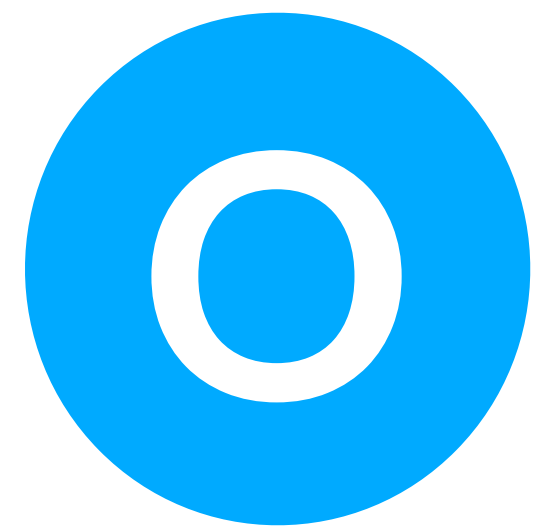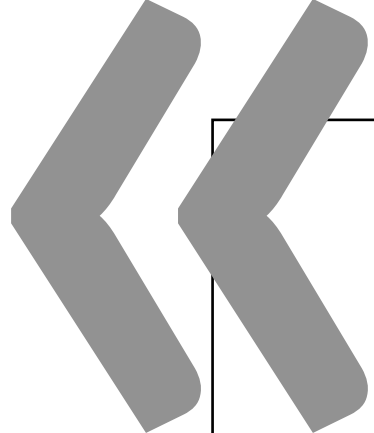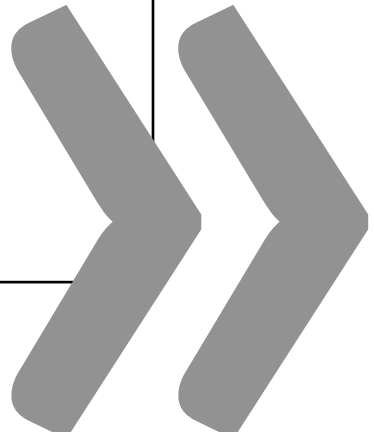
Open-closed principle

«
»

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

Бертран Мейер

Activity

```java
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 {
        // ...
    }
}
```

```java
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 {
        // ...
    }
}
```

```java
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 {
        // ...
    }
}
```

18

Как можно сделать по-другому?

```java
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);
}
```

```java
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);
        }
    }
    // ...
}
```

```java
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);
        }
    }
    // ...
}
```

```java
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);
        }
    }
    // ...
}
```

```java
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); }
    //...
}
```

```java
interface ViewStructureFiller {

    void fill(ViewStructure structure, boolean forAutofill, int flags);

}
```

```
interface ViewStructureFiller {

    void fill(ViewStructure structure, boolean forAutofill, int flags);

}
```

Ещё немного о View

```
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);
        //...
    }
}
```

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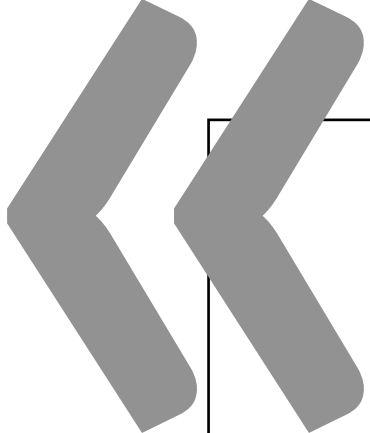
```
public final void setLeft(int left) {
    if (left != mLeft) {
        // ...
        mLeft = left;
        // ...
        sizeChange(mRight - mLeft, height, oldWidth, height);
        onLeftChange(left)
        //...
    }
}
```
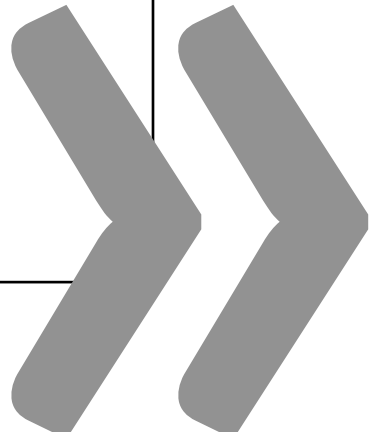
L Liskov substitution principle

« Абстракции можно заменить конкретными реализациями и ничего не сломается »

Диванный эксперт

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

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

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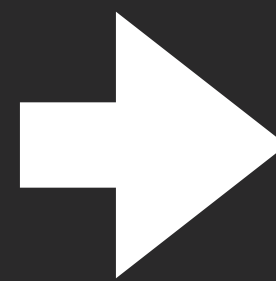
```kotlin
fun foo(list: List<String>) {
    // ...
}

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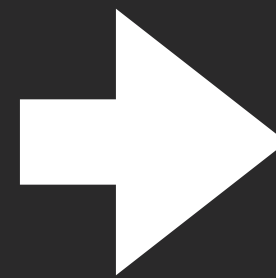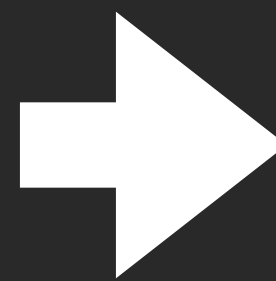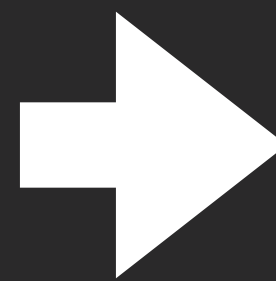
```
fun foo(list: List<String>) {         fun foo(list: ArrayList<String>) {
    // ...                                 // ...
}                                     }

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

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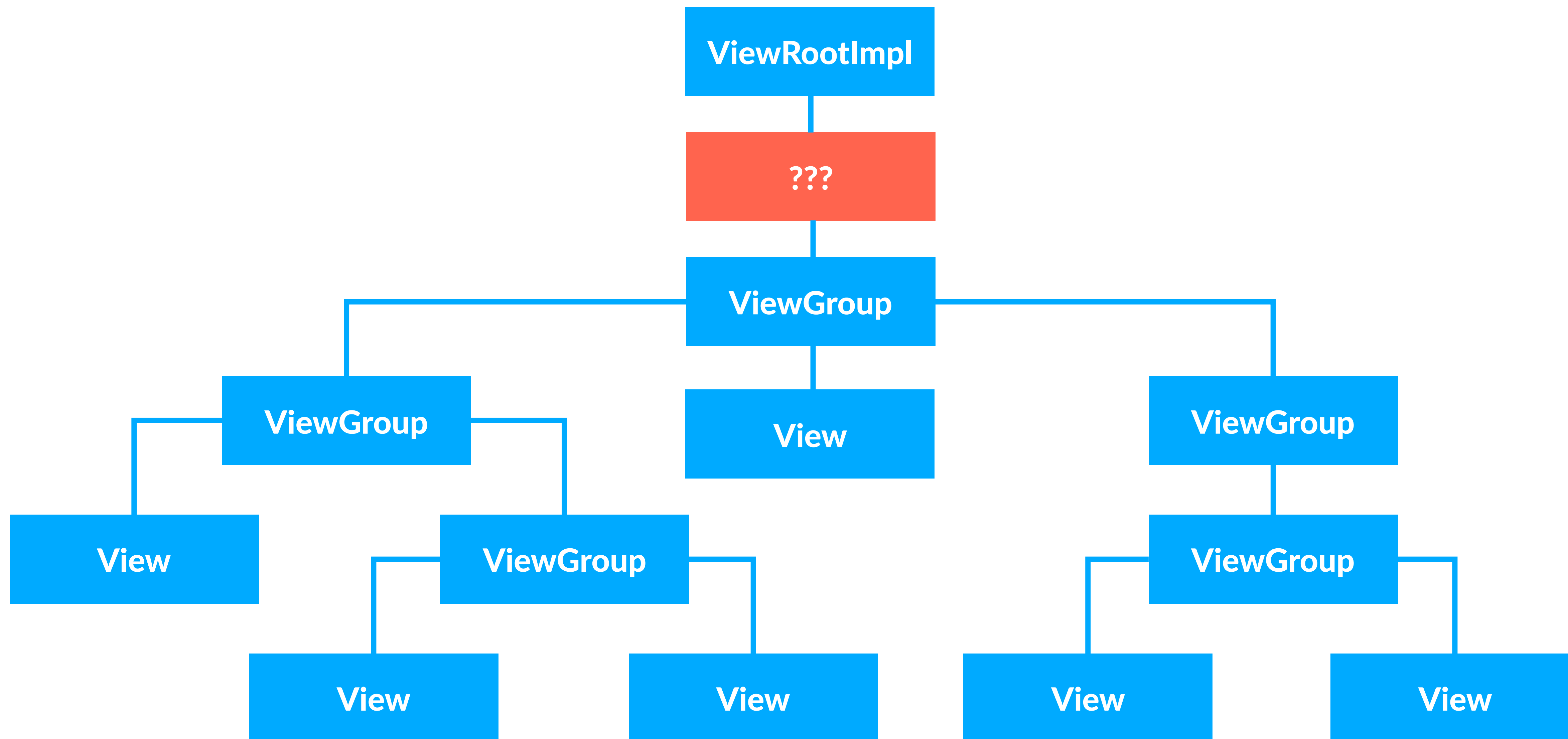
ResourcesImpl

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

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

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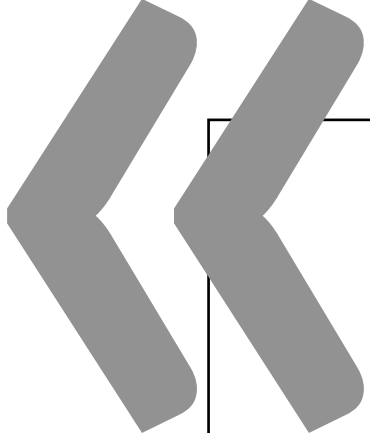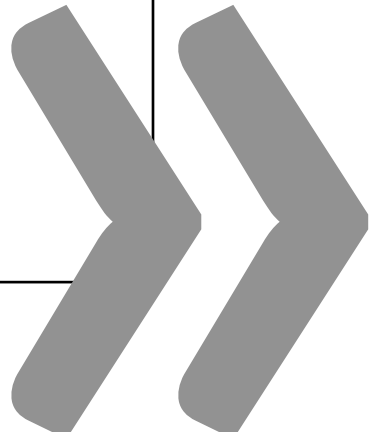
I Interface segregation principle

«

Много узкоспециализированных интерфейсов лучше, чем один интерфейс общего назначения

Ребята у кулера

»

```
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

42

```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);

}
```

```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);

}
```

```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);

}
```

```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) {}

}
```

```java
public static interface AnimatorPauseListener {

    void onAnimationPause(Animator animation);

    void onAnimationResume(Animator animation);

}
```
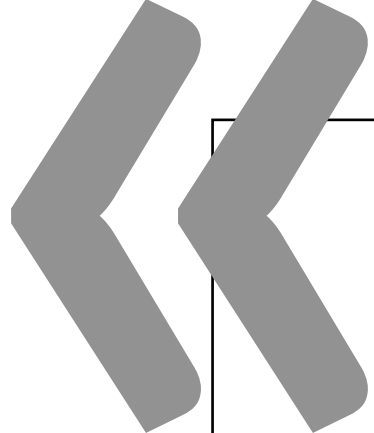
```java
public interface AnimatorStartEndListener {

    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);

}
```
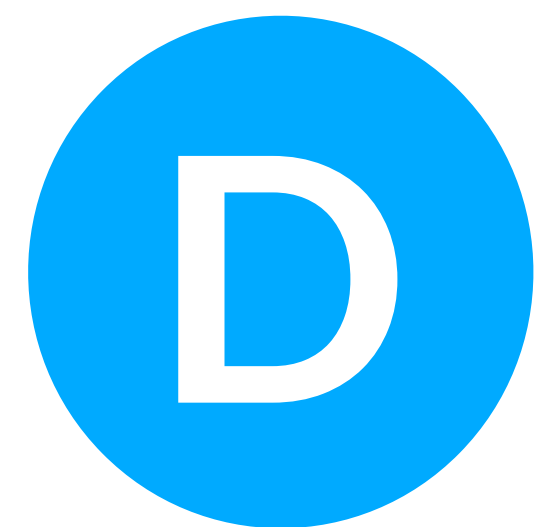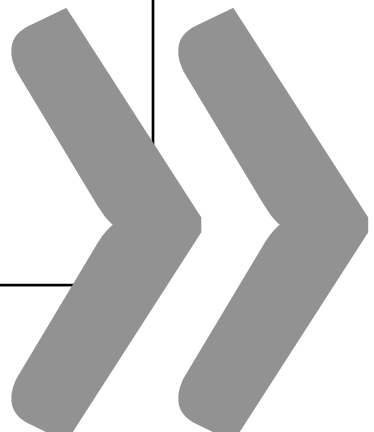
D Dependency inversion principle

« **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;
    }
    // ...
}
```

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

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

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```

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

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

        override fun onActivityStarted(activity: Activity) {}

        override fun onActivityResumed(activity: Activity) {}

        // ...
    })
```
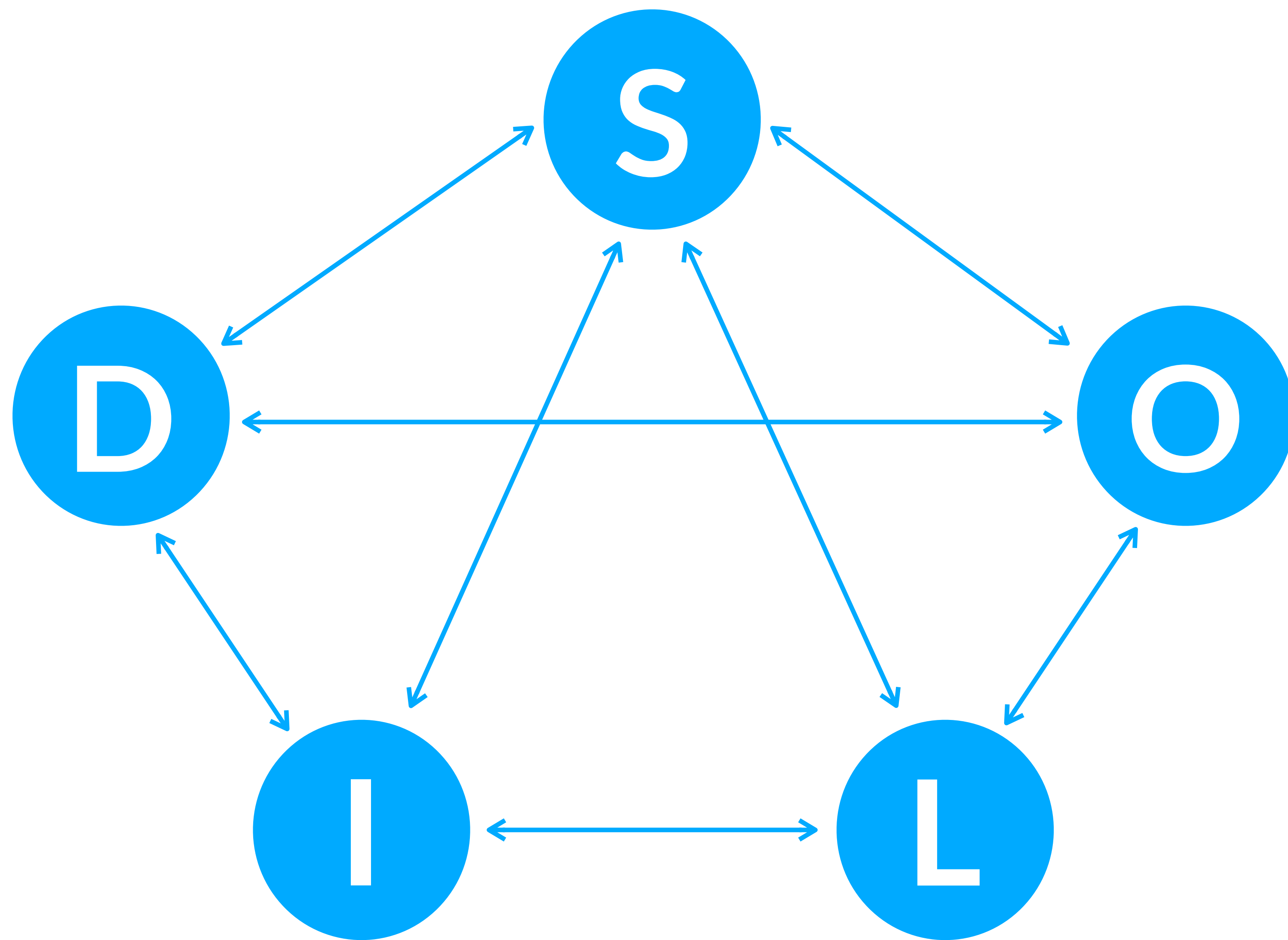
→ **еще 18 методов!**
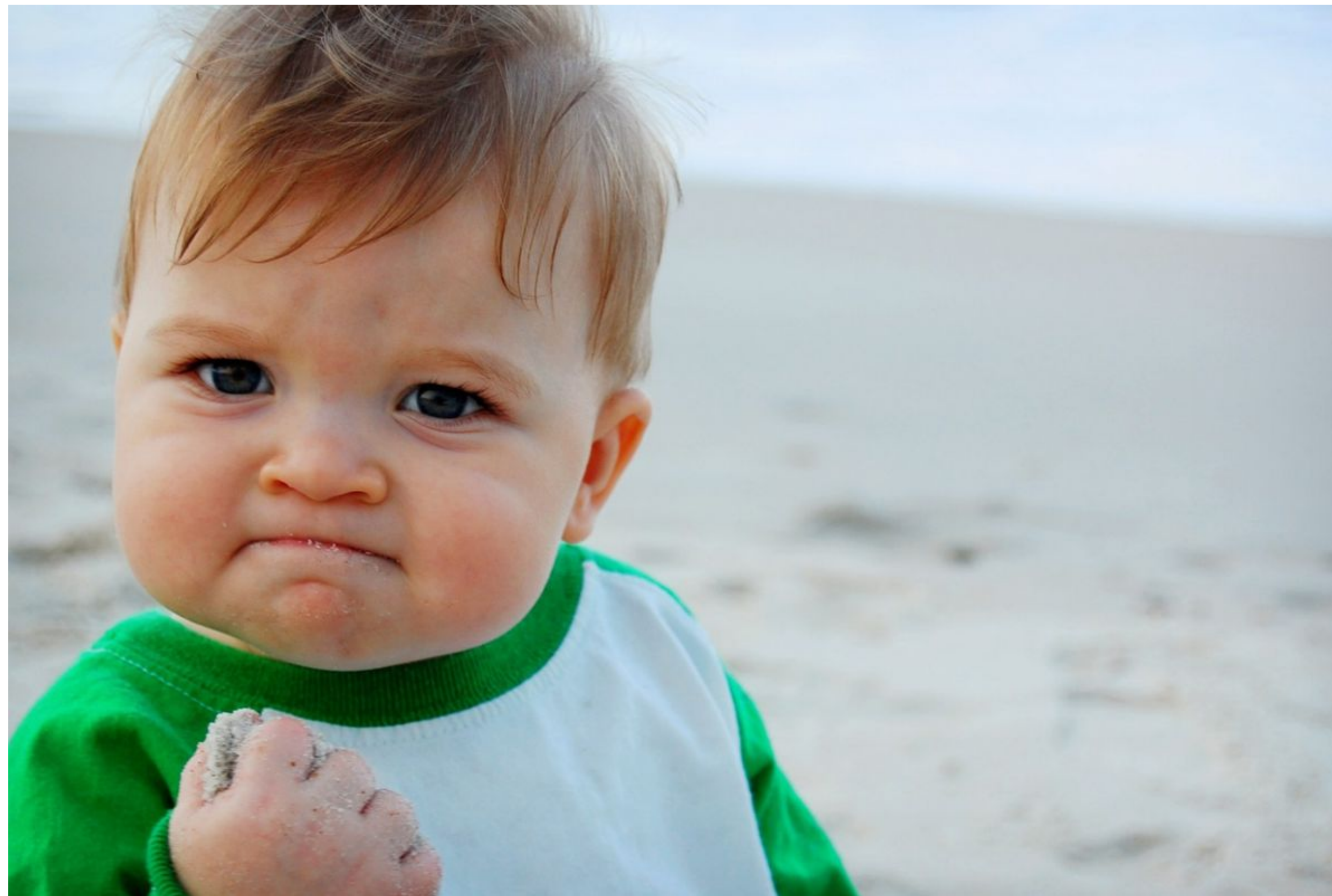
S O L I D

НЕ СТРОГИЕ ПРАВИЛА

S O L I D

НЕ СЕРЕБРЯНАЯ ПУЛЯ

КОГДА ВСПОМНИЛ И ПРИМЕНИЛ

ПРИНЦИПЫ SOLID

Спасибо

Стачка **ИННОПОЛИС** **INNOPOLIS UNIVERSITY**

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

Попов Даниил
Ведущий инженер

Стачка 2019

Avito