ЗВЕРСКИЙ ВЗГЛЯД НА

JAVA

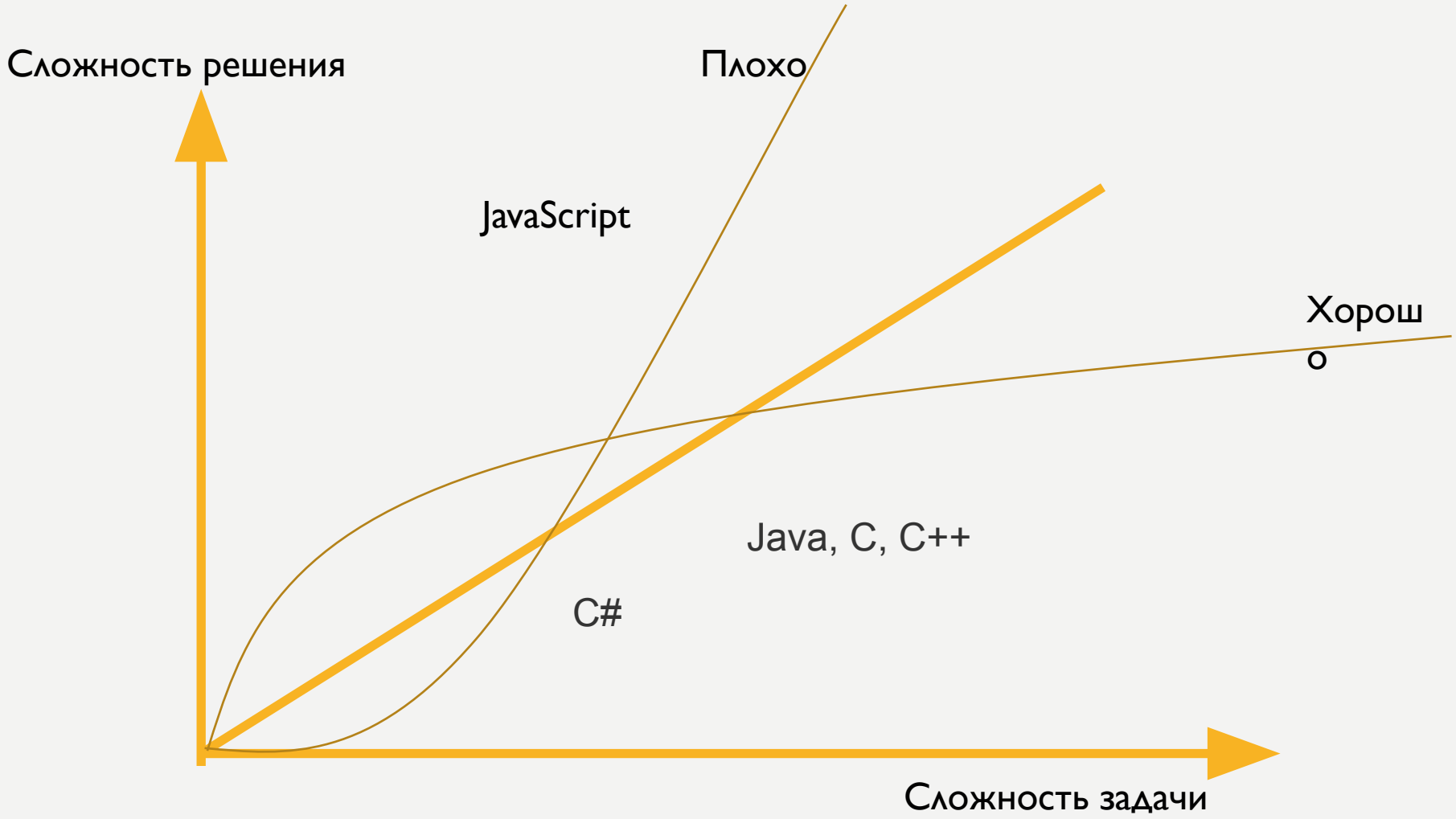
СПИСОК ЛИТЕРАТУРЫ

- «Полное руководство по Java» - Шилдт Г. Приводится подробное описание языка
- «Совершенный код» Макконнелл С. Подробно рассматриваются различные аспекты разработки ПО
- «Философия Java» Эккель Б. Рассказывается почему Джава устроена так, а не иначе.

- https://javarush.ru/ - ресурс для самообучения программированию на Java (уже платный)
- https://stepik.org/catalog - вроде как бесплатный

РАЗРАБОТКА ОДНОЙ КАРТИНКОЙ

УПРАВЛЕНИЕ СЛОЖНОСТЬЮ

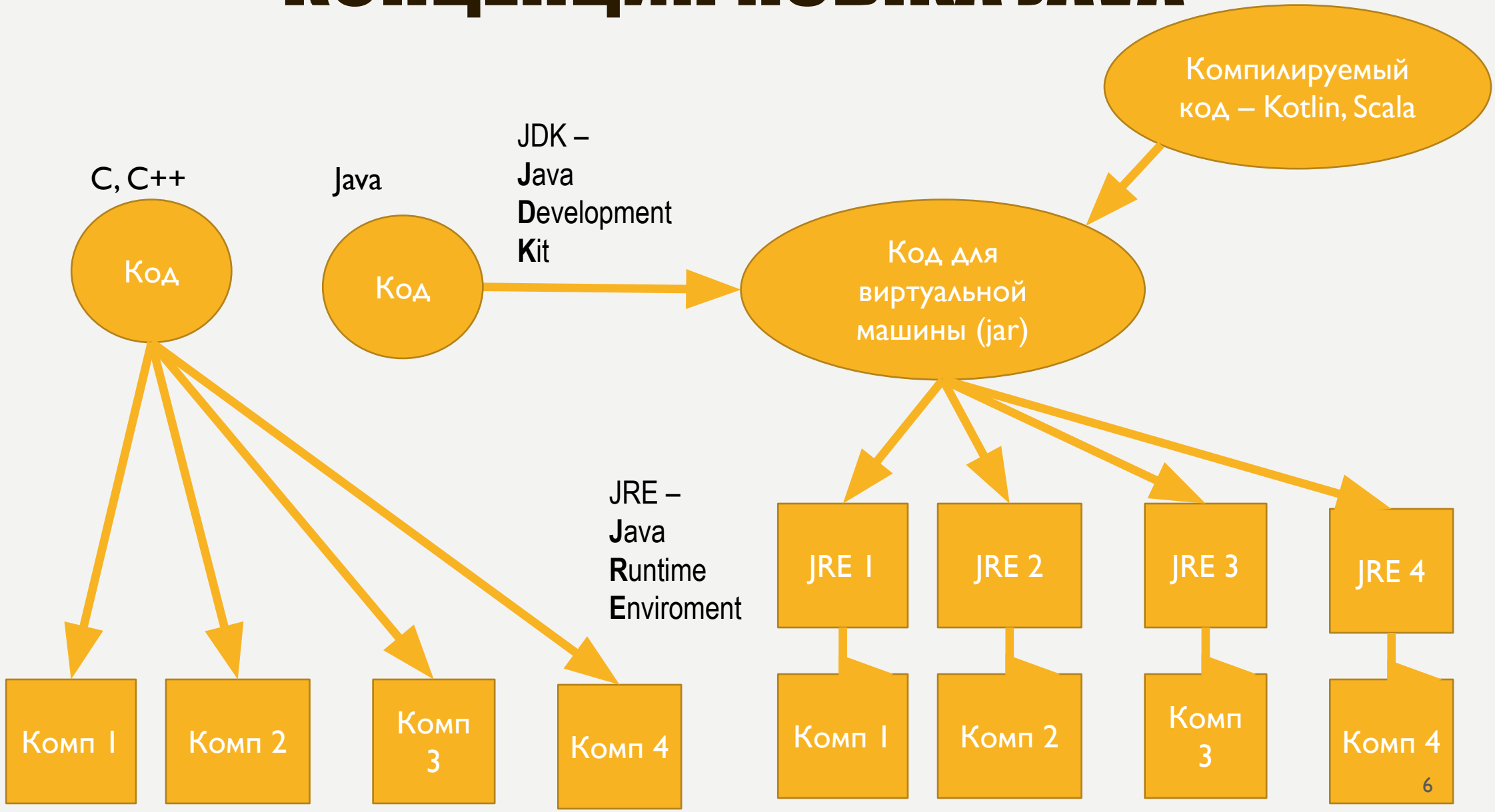


ОБЛАСТИ ПРИМЕНЕНИЯ

- Настольные приложения под ОС Windos, Linux, MacOS
  - Консольные
  - Встроенный функционал для создания оконных приложений
- web-приложения
  - сервлеты
  - ??апплеты??
- Мобильные приложения – Android

- У нас – OAF
- Android

КОНЦЕПЦИЯ ЯЗЫКА JAVA



ИЗ ЧЕГО СОСТОИТ JAVA

Скачать всё это можно с официального сайта Oracle

- JRE (Java Runtime Environment) – среда исполнения Java

- JDK (Java Development Kit) = JRE + инструменты разработки

- Javac – компилятор
- Javadoc – формирование документации
- Что-то ещё

Среда разработки:
IntelliJ IDEA фирмы JetBrains
Community – бесплатная. Мне хватает

В принципе для работы достаточно текстового редактора и консоли

СБОРЩИК МУСОРА

C, C++

Ручное управление памятью (new и delete)

+ Возможность более эффективного управления памятью (зависит от квалификации программиста)

– Сложнее программировать

- Выделение и освобождение памяти вручную

– Менее безопасно

- Можно обратиться вообще в любую область памяти

JAVA, C#

Виртуальная машина сама решает, когда можно удалить тот или иной объект.

+ Проще программировать

+ Безопасность – нет произвольного доступа к памяти

- Большее потребление памяти (зависит от квалификации программиста)

- Сборка мусора. Иногда приложение может зависнуть на какое-то время, так как все ресурсы тратятся только на очистку мусора

КОММЕНТАРИИ И ДОКУМЕНТАЦИЯ

```java
/**
 * A mutable sequence of characters.  This class provides an API compatible
 * with {@code StringBuffer}, but with no guarantee of synchronization.
 * This class is designed for use as a drop-in replacement for
 * {@code StringBuffer} in places where the string buffer was being
 * used by a single thread (as is generally the case).   Where possible,
 * it is recommended that this class be used in preference to
 * {@code StringBuffer} as it will be faster under most implementations.
 *
 * <p>The principal operations on a {@code StringBuilder} are the
 * {@code append} and {@code insert} methods, which are
 * overloaded so as to accept data of any type. Each effectively
 * converts a given datum to a string and then appends or inserts the
 * characters of that string to the string builder. The
 * {@code append} method always adds these characters at the end
 * of the builder; the {@code insert} method adds the characters at
 * a specified point.
 * <p>
 * For example, if {@code z} refers to a string builder object
 * whose current contents are "{@code start}", then
 * the method call {@code z.append("le")} would cause the string
 * builder to contain "{@code startle}", whereas
 * {@code z.insert(4, "le")} would alter the string builder to
 * contain "{@code starlet}".
 * <p>
 * In general, if sb refers to an instance of a {@code StringBuilder},
 * then {@code sb.append(x)} has the same effect as
 * {@code sb.insert(sb.length(), x)}.
 * <p>
 * Every string builder has a capacity. As long as the length of the
 * character sequence contained in the string builder does not exceed
 * the capacity, it is not necessary to allocate a new internal
 * buffer. If the internal buffer overflows, it is automatically made larger.
 *
 * <p>Instances of {@code StringBuilder} are not safe for
 * use by multiple threads. If such synchronization is required then it is
 * recommended that {@link java.lang.StringBuffer} be used.
 *
 * <p>Unless otherwise noted, passing a {@code null} argument to a constructor
 * or method in this class will cause a {@link NullPointerException} to be
 * thrown.
 *
 * @author      Michael McCloskey
 * @see         java.lang.StringBuffer
 * @see         java.lang.String
 * @since       1.5
 */
public final class StringBuilder
    extends AbstractStringBuilder
    implements java.io.Serializable, CharSequence
{
```

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP — Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

Class StringBuilder

java.lang.Object
  java.lang.StringBuilder

All Implemented Interfaces:

`Serializable, Appendable, CharSequence`

```
public final class StringBuilder
extends Object
implements Serializable, CharSequence
```

A mutable sequence of characters. This class provides an API compatible with `StringBuffer`, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for `StringBuffer` in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to `StringBuffer` as it will be faster under most implementations.

The principal operations on a `StringBuilder` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string builder. The `append` method always adds these characters at the end of the builder; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string builder object whose current contents are `"start"`, then the method call `z.append("le")` would cause the string builder to contain `"startle"`, whereas `z.insert(4, "le")` would alter the string builder to contain `"starlet"`.

In general, if sb refers to an instance of a `StringBuilder`, then `sb.append(x)` has the same effect as `sb.insert(sb.length(), x)`.

Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger.

Instances of `StringBuilder` are not safe for use by multiple threads. If such synchronization is required then it is recommended that `StringBuffer` be used.

Unless otherwise noted, passing a `null` argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.

Since:

1.5

See Also:

`StringBuffer, String, Serialized Form`

```java
/**
 * Returns the current capacity. The capacity is the amount of storage
 * available for newly inserted characters, beyond which an allocation
 * will occur.
 *
 * @return  the current capacity
 */
public int capacity() {
    return value.length;
}
```

to this sequence.

`int`	`capacity()` Returns the current capacity.

capacity

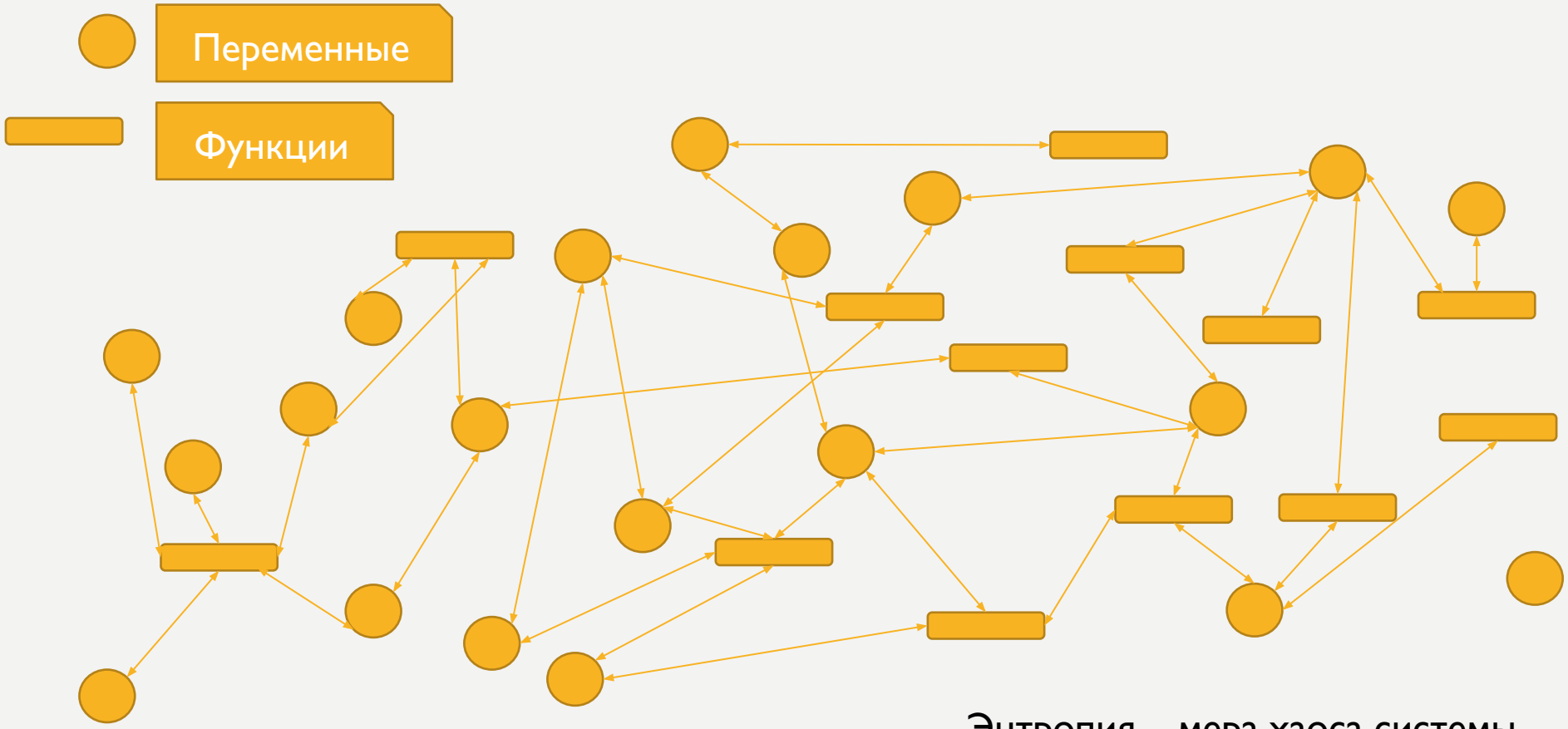
`public int capacity()`

Returns the current capacity. The capacity is the amount of storage available for newly inserted characters, beyond which an allocation will occur.

Returns:

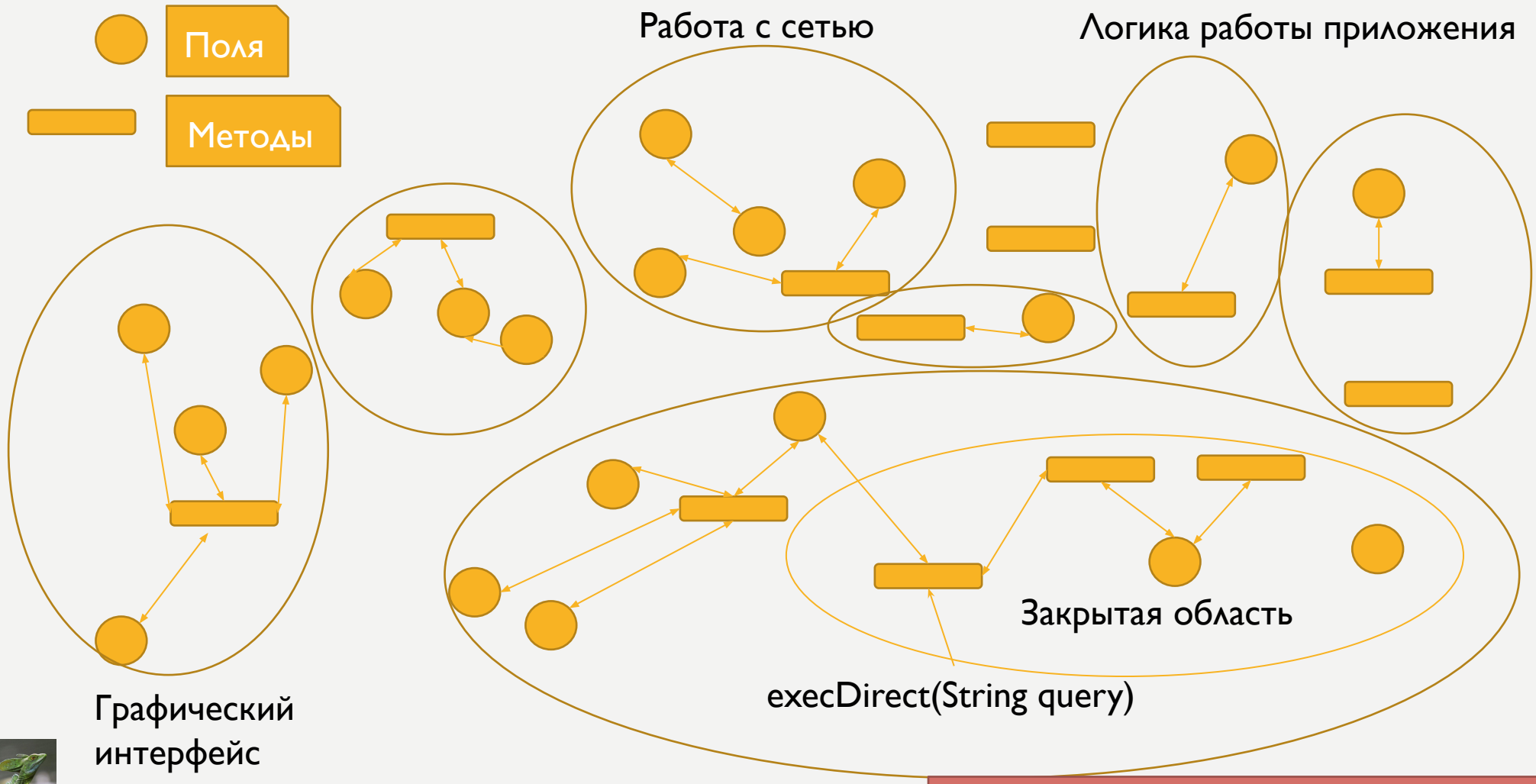
the current capacity

ЭНТРОПИЯ



Энтропия – мера хаоса системы.
Количественно можно описать возможным количеством микросостояний

НАДО БЫ КАК-ТО УПОРЯДОЧИТЬ ВСЁ ЭТО



Нужен ещё более высокий уровень абстракции

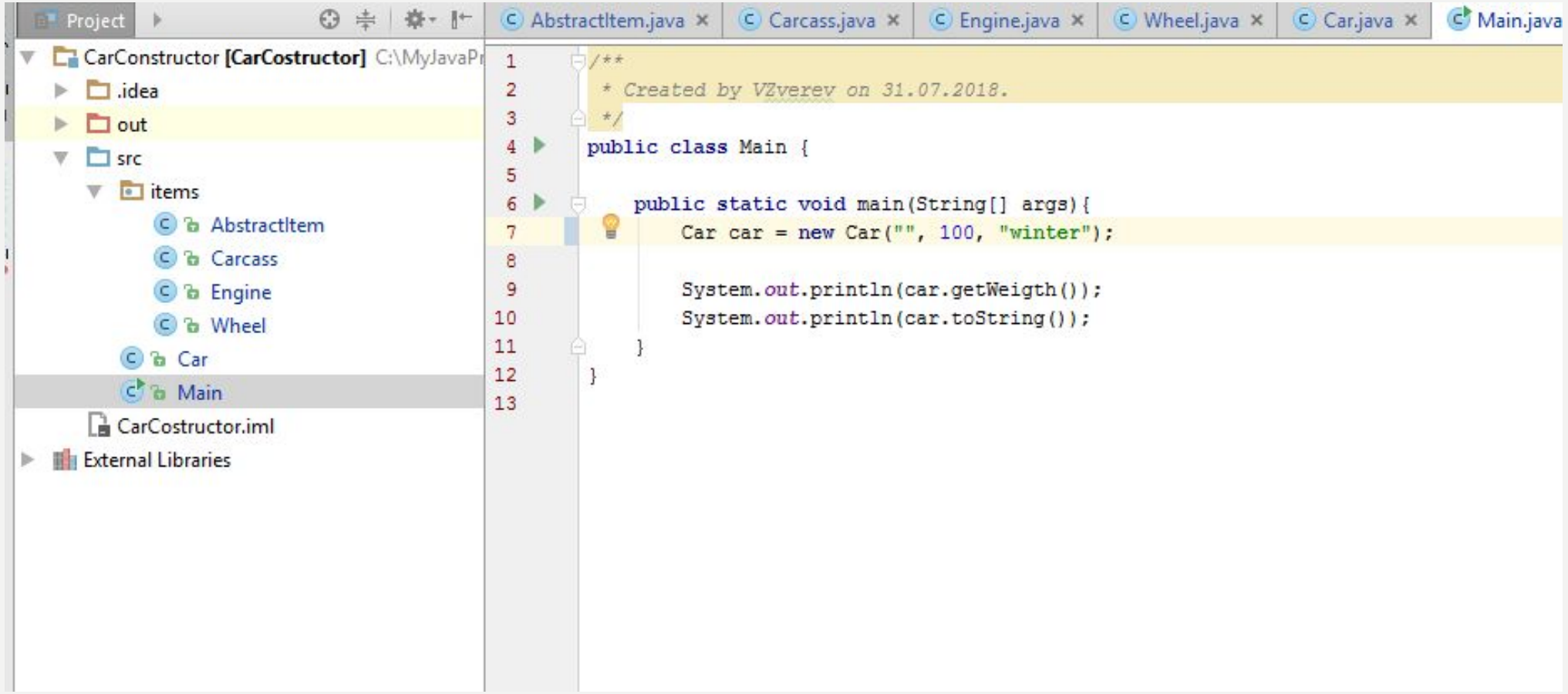
ЗАЧЕМ ТАК СЛОЖНО?

```
4  public class Car {
5
6      public String carcassName = "Lada sedan";
7      public double carcassWeight = 1562.8;
8      public String carcassColor = "baklajan";
9
10     public String engineName = "1.8";
11     public double engineWeigth = 89;
12     public double enginePower = 120;
13
14     public String wheelName = "Gardian 170/65";
15     public double wheelWeigth = 24;
16     public String wheelType = "winter";
17
18     public double getWeigth(){
19         return carcassWeight + engineWeigth + wheelWeigth * 4;
20     }
21
22     @Override
23     public String toString() {
24         return carcassName + ", colour: " + carcassColor
25                 + ", engine: " + engineName
26                 + ", wheels:"  + wheelName;
27     }
28 }
```

```java
public class Car {

    private String carcassName = "Lada sedan";
    private double carcassWeight = 1562.8;
    private String carcassColor = "baklajan";

    private String engineName = "1.8";
    private double engineWeigth = 89;
    private double enginePower = 120;

    private String wheelName = "Gardian 170/65";
    private double wheelWeigth = 24;
    private String wheelType = "winter";

    public boolean setCarcass(String name, double weigth, String colour){
        carcassName = name;
        carcassWeight = weigth;
        carcassColor = colour;

        return true;
    }

    public boolean setEngine(String name, double weigth, double power){
        engineName = name;
        engineWeigth = weigth;
        enginePower = power;

        return true;
    }

    public boolean setWheels(String name, double weigth, String type){
        wheelName = name;
        wheelWeigth = weigth;
        wheelType = type;

        return true;
    }

    public double getWeigth(){
        return carcassWeight + engineWeigth + wheelWeigth * 4;
    }

    @Override
    public String toString() {
        return carcassName + ", colour: " + carcassColor
                + ", engine: " + engineName
                + ", wheels:"  + wheelName;
    }
}
```

СОГЛАСОВАННОЕ РЕДАКТИРОВАНИЕ ПОЛЕЙ

ПАКЕТЫ
1 ОТКРЫТЫЙ КЛАСС – 1 ФАЙЛ



ОБЛАСТЬ ВИДИМОСТИ

	Блок кода	Функция	Класс	Наследники	Пакет	Любые другие классы
private	-	-	+			
protected	-	-	+	+		
-	-	-	+	+	+	
public	-	-	+	+	+	+

```
preferences = getSharedPreferences( name: "Download", MODE_PRIVATE);
preferenceEditor = preferences.edit();
//Проверяем, что все данные с сервера загружены
{
    StringBuilder errors = new StringBuilder();
    for (RequestType requestType : RequestType.values()) {
        if (!preferences.getBoolean(requestType.getFileName(), b: false)) {
            errors.append(requestType.getName()).append("\n");
        }
    }

    if (errors.length() > 0) {
        AlertDialog.Builder alertBuilder = new AlertDialog.Builder( context: this);
        alertBuilder
                .setTitle("Не полностью загруженые данные")
                .setMessage(errors.toString());

        AlertDialog alert = alertBuilder.create();
        alert.show();

        stateLevelRecordList.add(new StateLevelRecord(StateLevel.ERROR, message: "Данные не полность загружены. Рекоментдуется запустить загрузку данных"));
    }
}
```

РАННЕЕ И ПОЗДНЕЕ СВЯЗЫВАНИЕ ИЛИ ПАРА СЛОВ О ПРИВАТНОСТИ

- private – нельзя переопределить, то есть можно выполнить связывание на этапе компиляции. Несколько выше производительность
- Все остальные – связывание на этапе выполнения. То есть выполняемый вариант будет выполняться во время выполнения программы

ИНИЦИАЦИЯ ПОЛЕЙ

```java
public class Car {

    //Инициация при объявлении
    private String carcassName = "Lada sedan";
    private double carcassWeight = 1562.8;
    private String carcassColor = "baklajan";

    private String engineName;
    private double engineWeigth, enginePower;

    //Блок инициализации. Выполняется до конструктора
    {
        //Загружаем откуда-нибудь
        engineName = "1.8";
        engineWeigth = 89;
        enginePower = 120;
    }

    //Инициация в конструкторе
    private final String wheelName;
    private final double wheelWeigth;
    private final String wheelType;


    public Car(String wheelName, double wheelWeigth, String wheelType) {
        this.wheelName = wheelName;
        this.wheelWeigth = wheelWeigth;
        this.wheelType = wheelType;
    }
```

Объявление создание переменной. Можно везде, но до первого использования

Инициализация присвоение значения

МНОГОКРАТНОЕ ИСПОЛЬЗОВАНИЕ КОДА

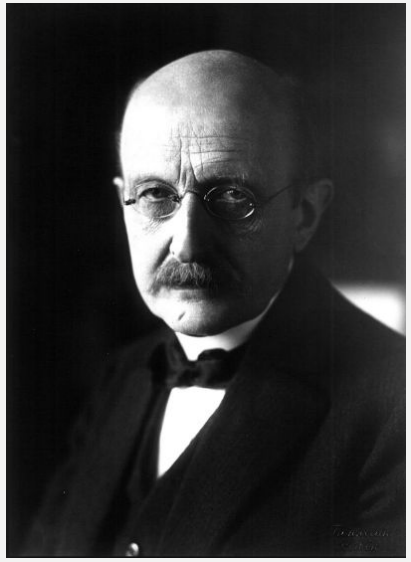
ОДНОКРАТНО ИСПОЛЬЗУЕМЫЙ КОД

- Много кода
- Сложно поддерживать одинаковую работу
- Низкое качество кода

МНОГОКРАТНО ИСПОЛЬЗУЕМЫЙ КОД

- Мало кода
- Высокое качество кода

ДАННЫЕ СУЩЕСТВУЮТ И ПЕРЕДАЮТСЯ ЦЕЛЫМИ ОБЪЕКТАМИ



Макс Планк – ввёл понятие «кванта действия», что стало основой квантовой механики. Можно провести аналогию, что части программ тоже передаются и существуют целыми кусками

```
public Dialog onCreateDialog(Bundle savedInstanceState) {
    Bundle b = getArguments();

    //ID записи
    mCurrRecordId = b.getLong(ARG_REC_ID,  defaultValue: -1);
    mTableName = b.getString(ARG_TABLE);

    //Для безопасности (пока так)
    if (mCurrRecordId == -1 || mTableName.equals(DB.ONHAND_TABLE_NAME)) {
        GLog.L(LOG_TAG,  text: "Onhand table Stub");
        //mLotNumber = "Ошибка! ID строки не получен";
        mSnItem = new SnItem();
        mSnItem.setSnCtrl("1");
        mSnItem.setLotCtrl("1");
    } else {
        mSnItem = MySingleton.getInstance(null).getDb().getSnItemRecById(mTableName, mCurrRecordId);
        //mLotNumber = mSnItem.getLotNumber();
    }

    builder = new AlertDialog.Builder(getActivity());

    LayoutInflater inflater = getActivity().getLayoutInflater();
    final View view = inflater.inflate(R.layout.context_dialog,  root: null);
```

НАСЛЕДОВАНИЕ

Абстрактный класс – нельзя создать объект.
Определяет функционал

```java
public class AbstractItem {
    private double weigth;
    private String name;

    public AbstractItem(double weigth, String name) {
        this.weigth = weigth;
        this.name = name;
    }

    public boolean setValues(double weigth, String name){
        this.weigth = weigth;
        this.name = name;

        return true;
    }

    public double getWeigth() { return weigth; }

    public String getName() {
        return name;
    }
}
```

```java
List<AbstractItem> itemList = new ArrayList<>();
```

```java
public class Carcass extends AbstractItem {
    private String colour;

    public Carcass(double weigth, String name, String colour) {
        super(weigth, name);
        this.colour = colour;
    }

    public void setColour(String colour) { this.colour = colour; }

    public String getColour() { return colour; }
}
```

```java
public class Engine extends AbstractItem {
    double power;

    public Engine(double weigth, String name, double power) {
        super(weigth, name);
        this.power = power;
    }
}
```

```java
public class Wheel extends AbstractItem {

    String type;

    public Wheel(double weigth, String name, String type) {
        super(weigth, name);
        this.type = type;
    }
}
```

ВСЁ ЕСТЬ OBJECT

Method Summary

Methods

Modifier and Type	Method and Description
`protected Object`	`clone()` Creates and returns a copy of this object.
`boolean`	`equals(Object obj)` Indicates whether some other object is "equal to" this one.
`protected void`	`finalize()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class<?>`	`getClass()` Returns the runtime class of this `Object`.
`int`	`hashCode()` Returns a hash code value for the object.
`void`	`notify()` Wakes up a single thread that is waiting on this object's monitor.
`void`	`notifyAll()` Wakes up all threads that are waiting on this object's monitor.
`String`	`toString()` Returns a string representation of the object.
`void`	`wait()` Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
`void`	`wait(long timeout)` Causes the current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
`void`	`wait(long timeout, int nanos)` Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

NULL

- Все объекты могут быть преобразованы в null
- NullPointerExeption – нет объекта

ПАРА СЛОВ О ХРУПКОСТИ

- А теперь в базовом классе нам нужна ещё и цена
  - Добавляем в базовом классе поле и добавляем ещё один параметр в конструктор (пока всё хорошо)
  - Меняем соответственным образом конструкторы все потомков (более - менее нормально)
  - Исправляем все использования в коде (а вот здесь начинается самая жесть)

НАСЛЕДОВАНИЕ И КОМПОЗИЦИЯ

Пример композиции

```
import items.Carcass;
import items.Engine;
import items.Wheel;

public class Car_composition {
    private Carcass carcass;
    private Engine engine;

    Wheel[] forwardWheels   = new Wheel[2];
    Wheel[] backWheels      = new Wheel[2];
}
```

	Наследование	Композиция
Количество	1	множество
Можно менять во время исполнения	X	V
Доступ с защищенным полям класса и методам	V	X
Возможность использовать вместо базового класса	V	X

ПРИМИТИВЫ

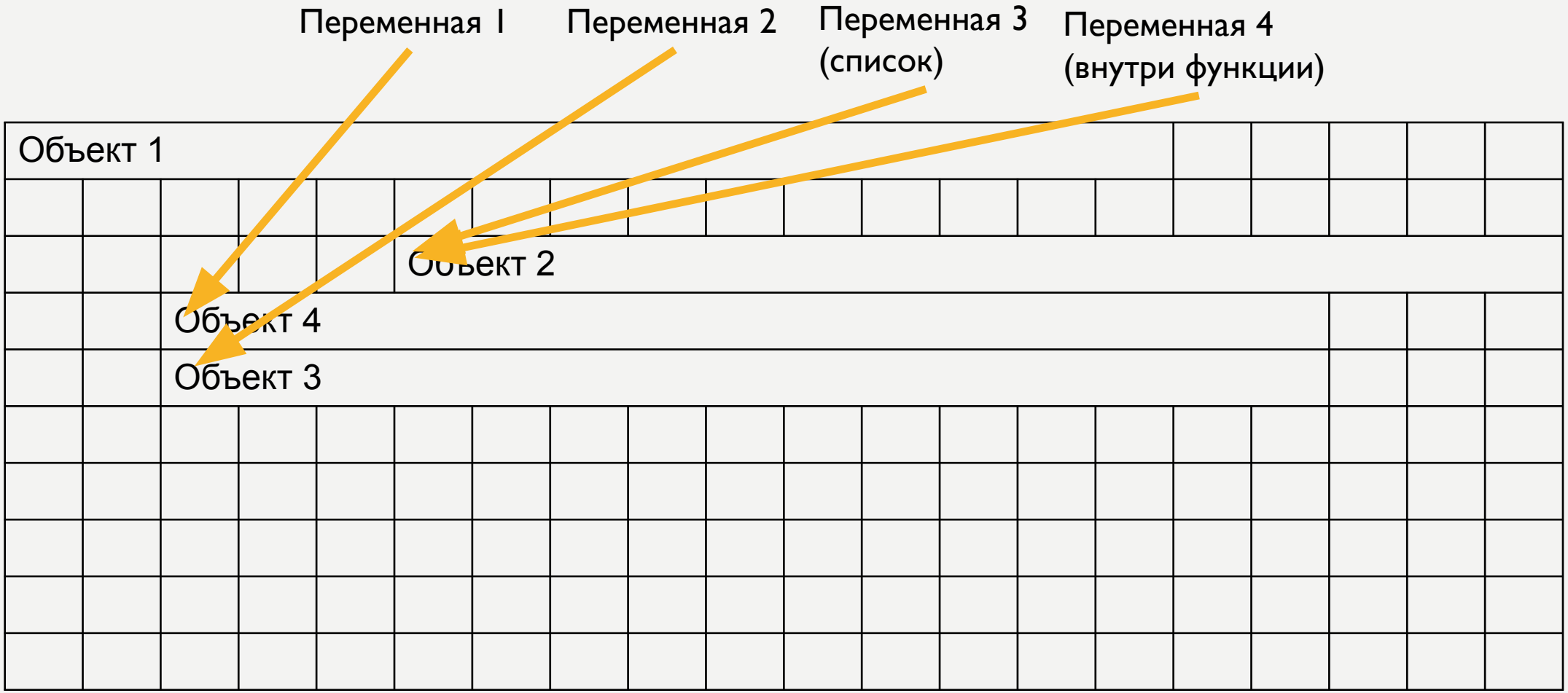
Указатель

byte								-128 до 127
short								-32768 до 32767
int								-2147483648 до 2147483647
long								
float								$1,4*10^{-45}$ до $\sim 3,4*10^{38}$
double								$4,9*10^{-324}$ до $\sim 1,8*10^{308}$
char								16 разрядное целое (UTF-16)
boolean								True/false

```
double periodInt    = 100_000/120_000;  periodInt: 0.0
double periodD      = 100_000D/120_000D;  periodD: 0.8333333333333334
```

```
double doubleVal = new Double(12.0).doubleValue();
double parcedDouble = Double.parseDouble("12.0");
```

ОБЪЕКТЫ В ПАМЯТИ



СРАВНЕНИЕ

Оператор == работает для примитивов именно так, как и предполагается
Для объектов сравнивается ссылка.
Как сравнить две даты? Что считать равными датами в текущей задаче?
С точностью до часа? Минуты? Попадают в один период?

- equals()

Для сравнения объектов в коде

По умолчанию сравниваются ссылка на объекты.

Для всех других случаев надо метод переопределить

- hashCode()

Для корректной работы карт (HashMap<?>, HashTable<?>)

НЕМНОГО СТАТИЧНОСТИ

- static – элемент класса, не нужен объект.
  - public static void main(String[] args)
- Блок инициализации тоже может быть статичным
  - static { статичный код }

- Может использовать только статичные поля и методы

- Часто используется для работы со сложными объектами, когда создание большого их количества нежелательно
- Единый объект на все экземпляры. Часто используется для реализации констант.

ПОЛИМОРФИЗМ

- Наследование
- Реализация интерфейсов
- Перегрузка функций
- Список аргументов переменной длинны

```
public void addDiscout(String name, double ... values){

    values[0] = 1;

}
```

Method Summary

Methods

Modifier and Type	Method and Description
StringBuilder	append(boolean b) Appends the string representation of the boolean argument to the sequence.
StringBuilder	append(char c) Appends the string representation of the char argument to this sequence.
StringBuilder	append(char[] str) Appends the string representation of the char array argument to this sequence.
StringBuilder	append(char[] str, int offset, int len) Appends the string representation of a subarray of the char array argument to this sequence.
StringBuilder	append(CharSequence s) Appends the specified character sequence to this Appendable.
StringBuilder	append(CharSequence s, int start, int end) Appends a subsequence of the specified CharSequence to this sequence.
StringBuilder	append(double d) Appends the string representation of the double argument to this sequence.
StringBuilder	append(float f) Appends the string representation of the float argument to this sequence.

append() – метод. Много сигнатур.

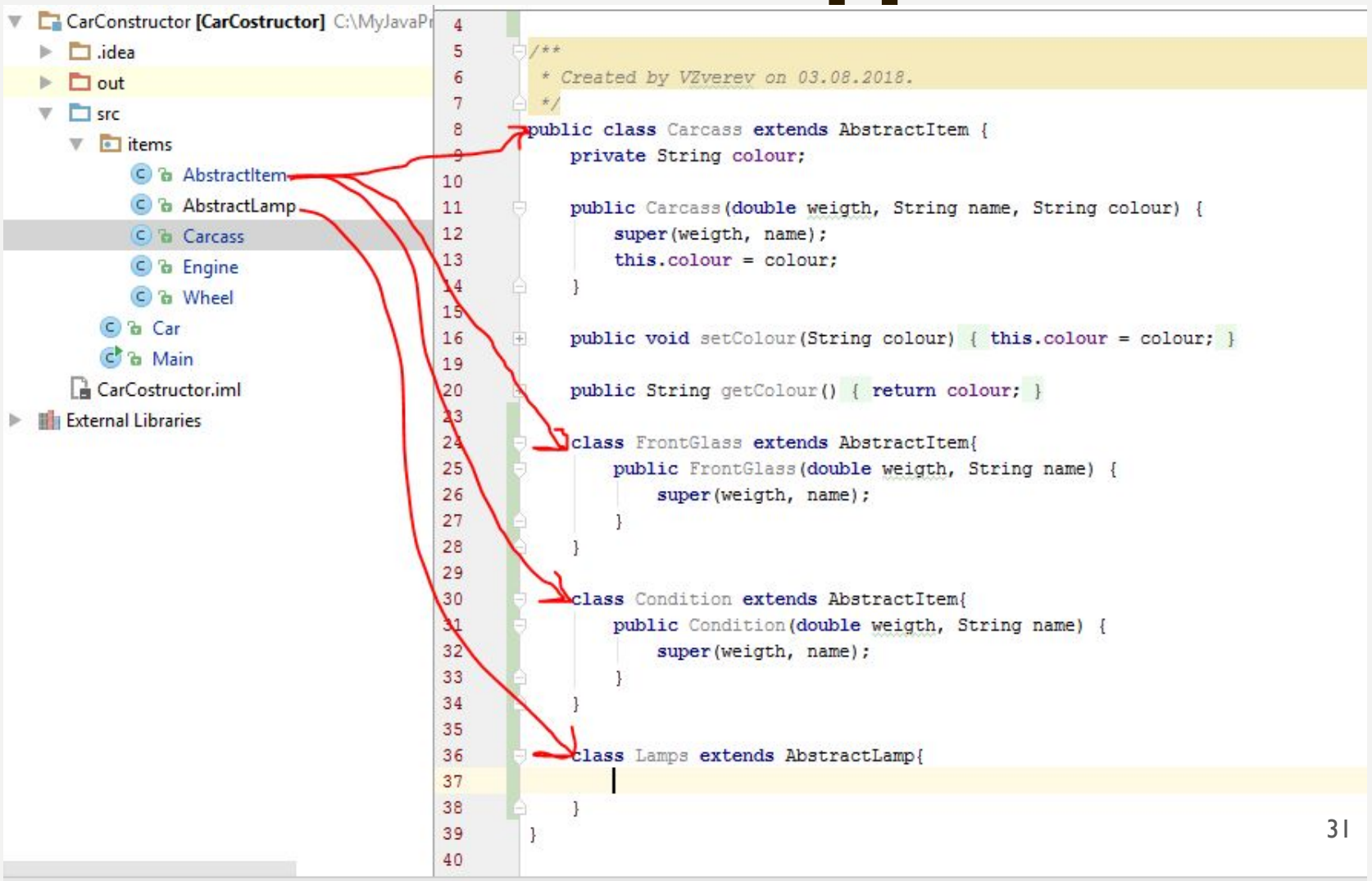
ЗАПУСК ПРИЛОЖЕНИЯ

- Вначале было: static public void main(String[] args){}

  Этот метод – точка входа в программу.

  – static – в начале нет ни одного объекта
  – public – нужен доступ для запуска
  – void – это есть начали и конец, зачем возвращать значение?
  – String[] args – список параметров запуска. Используется при запуске из консоли.

ВЛОЖЕННЫЕ КЛАССЫ И МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ



КОНВЕНЦИЯ ИМЕНОВАНИЯ

Для Java это официальный документ (https://www.oracle.com/technetwork/java/codeconventions-135099.html).

Имеет статус рекомендаций, то есть отклонения не являются ошибками синтаксиса.

Если кратко:

КОНСТАНТЫ

ИмяКласса

имяПеременной, имяМетода()

ВЕТВЛЕНИЯ

IF

- Ну всё как везде
- Тернарный оператор
  - If true ? 2 :0

SWITCH

- Фиксированные строки
- Числа
- Элементы перечислений

```
List<String> months = new ArrayList<>(12);
months.add("Январь");
months.add("Февраль");
months.add("Март");
months.add("Апрель");
months.add("Май");
months.add("Июнь");
months.add("Июль");
months.add("Август");
months.add("Сентябрь");
months.add("Октябрь");
months.add("Ноябрь");
months.add("Декабрь");

check:
for(int iYear = 2016; iYear < 2020; iYear++)
    for(String month: months){
        if(true){
            break;
        }
        else
            break check;
    }
```

ЦИКЛЫ И ~~GOTO~~ МЕТКИ

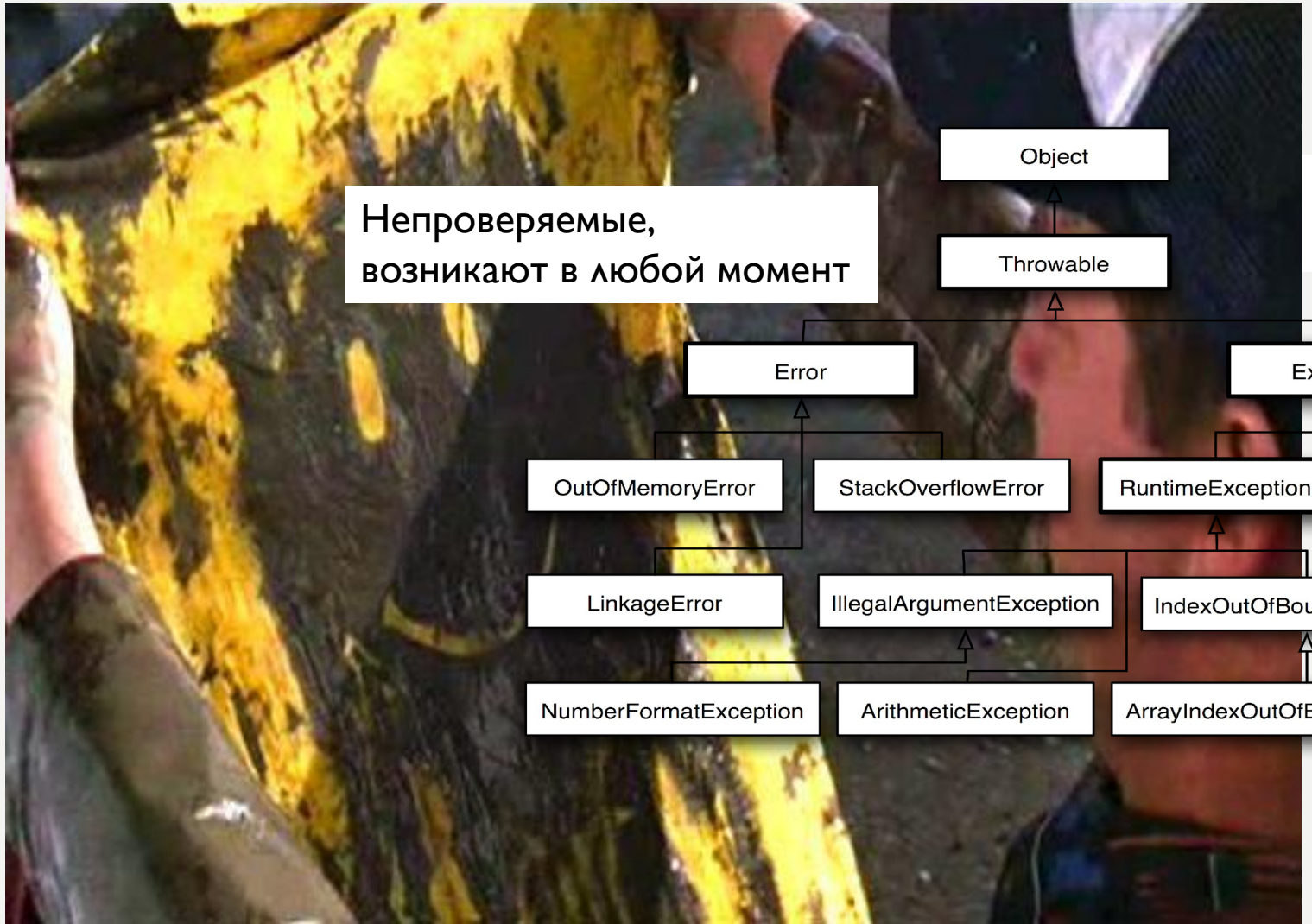
FINAL

- Поля класса – обязательно должны быть инициированы ровно 1 раз:
  - либо при объявлении
  - либо в блоке инициирования
  - либо во всех конструкторах

  final – не константа, пишем маленькими.

  - Не могут больше менять значение. При этом объект может изменяться!
- Переменные метода – значит в методе не меняется ссылка
- Методы – нельзя переопределить.
- Классы – нельзя переопределить. Например String.

ИСКЛЮЧЕНИЯ СЛУЧАЮТСЯ



Непроверяемые,
возникают в любой момент

Проверяемые,
проверка на этапе компиляции

ИСКЛЮЧЕНИЯ СЛУЧАЮТСЯ

```
try {
    DB db = mySing.getDb();
    db.UpdateReceiveQty(itemList.getCursor(), qltFloat, s_row, s_col, s_descr, TvTermOfRec.getText().toString());
    setResult(RESULT_OK, intent);
    finish();
} catch (IOException e) {
    e.getCause().printStackTrace();
    //Toast.makeText(this, e.getMessage().toString(), Toast.LENGTH_LONG).show();
}
```

- Исключение в конструкторе – объект не создаётся
- Исключение либо обрабатывается, либо передаётся в сигнатуру метода
- При переопределении методов исключения можно только добавлять

```
Thread.setDefaultUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler() {
    @Override
    public void uncaughtException(Thread t, Throwable e) {

    }
});
```

TRY И TRY С РЕСУРСАМИ

```
public boolean createColumnIfNeed(String tableName, String columnName, String type) {
    try (Cursor cursor = getCursor( "Select * from " + tableName)) {
        if (cursor.getColumnIndex(columnName) == - 1) {
            mDB.execSQL("alter table " + tableName + " add column " + columnName + " " + type + ";");
            mDB.execSQL("alter table " + tableName + TMP + " add column " + columnName + " " + type + ";");

            return true;
        }

        return false;
    }
}
```

Можно объявить переменные, реализующие интерфейс AutoCloseable. Метод close() будет вызван при выходе из блока.

```
try {
//код
}
catch{Exeption e}
finally{
//Выполняется всегда после блока кода
}
```

КОЛЛЕКЦИИ И МАССИВЫ

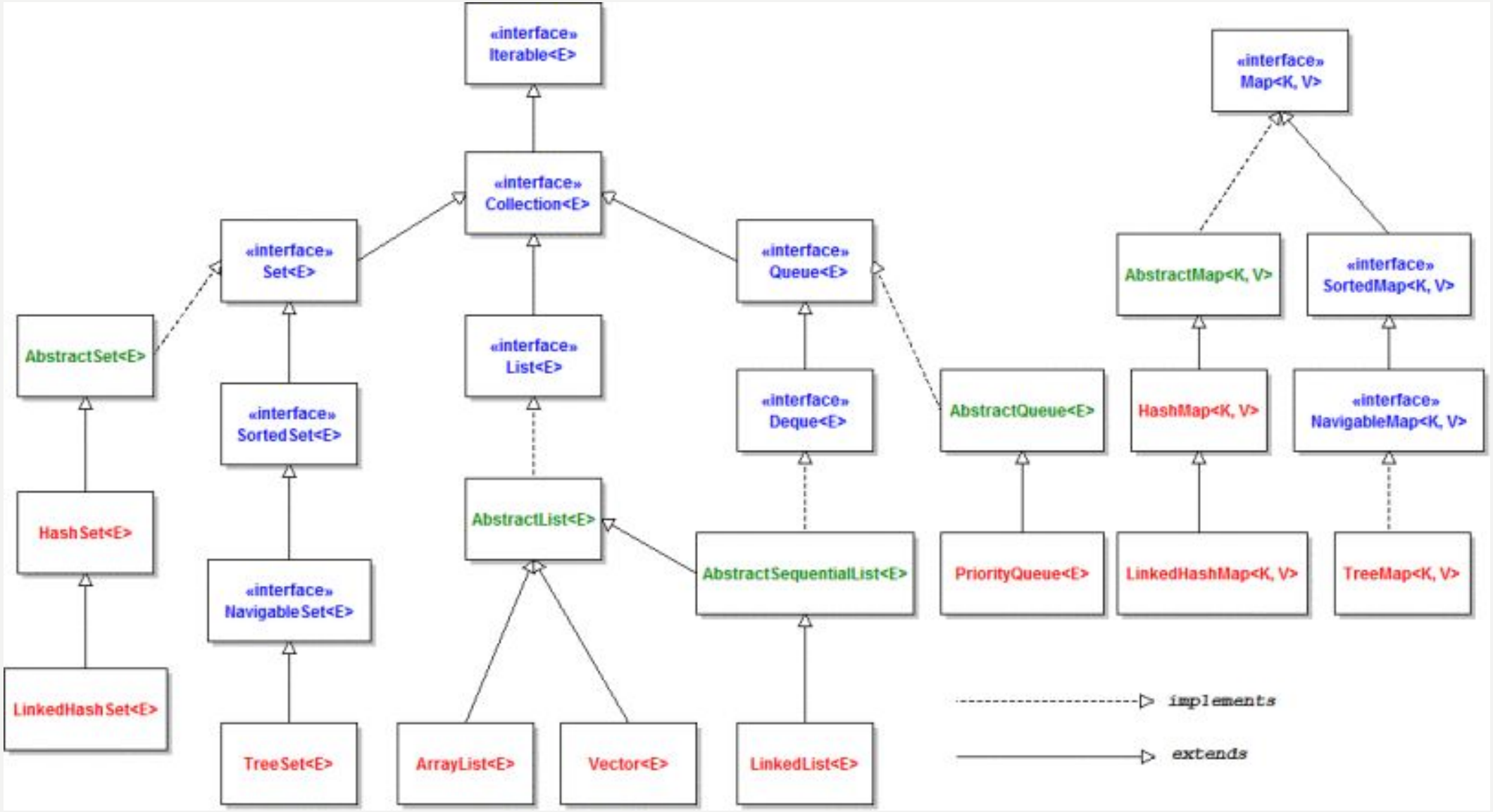
КОЛЛЕКЦИИ

- Произвольное количество элементов (можно задавать начальное кол-во)
- Случайное распределение в памяти, как следствие – меньшая производительность
- Не уменьшаются, если не используются элементы

МАССИВЫ

- Фиксированное кол-во элементов
- В памяти располагаются последовательно, как следствие большая производительность

ПАРА СЛОВ О КОЛЛЕКЦИЯХ



ПЕРЕЧИСЛЕНИЯ

Список констант
+ можно сделать цикл по всему множеству
+ Дополнительное ограничение при передаче параметров / возврате значений

```
public enum RequestType{
    ORGANISATIONS{...},

    DEPARTMENTS{...},

    EMPOLOYEES{...},

    PHONES{...},

    MOBILE{
        @Override
        public String getLink() { return "/api/phone/all"; }

        @Override
        public String getFileName() { return "MobilePhones"; }

        @Override
        public String getFieldsList() {
            return "fields=organization_id,department_id,employee_id,country_code,city_code,number,type,typeToString";
        }

        @Override
        public String getExpand() { return null; }

        @Override
        public String getName() {
            return "Мобильные телефоны";}

        @Override
        String getFilter() { return "filter[type]=2"; }

        @Override
        String getSort() { return null; }
    };

    abstract public String getLink();
    abstract public String getFileName();
    abstract public String getFieldsList();
    abstract String getExpand();
    abstract String getSort();
    abstract public String getName();
    abstract String getFilter();

    public File getFolder(File folder){...}

    String getLinkWithParams(){...}
}
```

```
for(RequestType requestType: RequestType.values()){

    horizontLinear = new LinearLayout( context: MainActivity.this);
    horizontLinear.setOrientation(LinearLayout.HORIZONTAL);

    progressBar = new ProgressBar( context: MainActivity.this);
    horizontLinear.addView(progressBar);

    textView = new TextView( context: MainActivity.this);
    textView.setText(requestType.getName());
    horizontLinear.addView(textView);

    layout.addView(horizontLinear);

    dataLoadList.add(new AsynkDataLoad(requestType, progressBar, textView, alertDialog));
}
```

АННОТАЦИИ

- @Override – проверяет, что есть что переопределять
- @Deprecated - устарело

НЕМНОГО О СТРОКАХ

STRING

- Фиксированный, неизменяемый текст

STRINGBUILDER

- Класс для редактирования строк
- Операции по редактированию строк проходят намного быстрее

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ДЛЯ НАСТОЛЬНЫХ ПРИЛОЖЕНИЙ

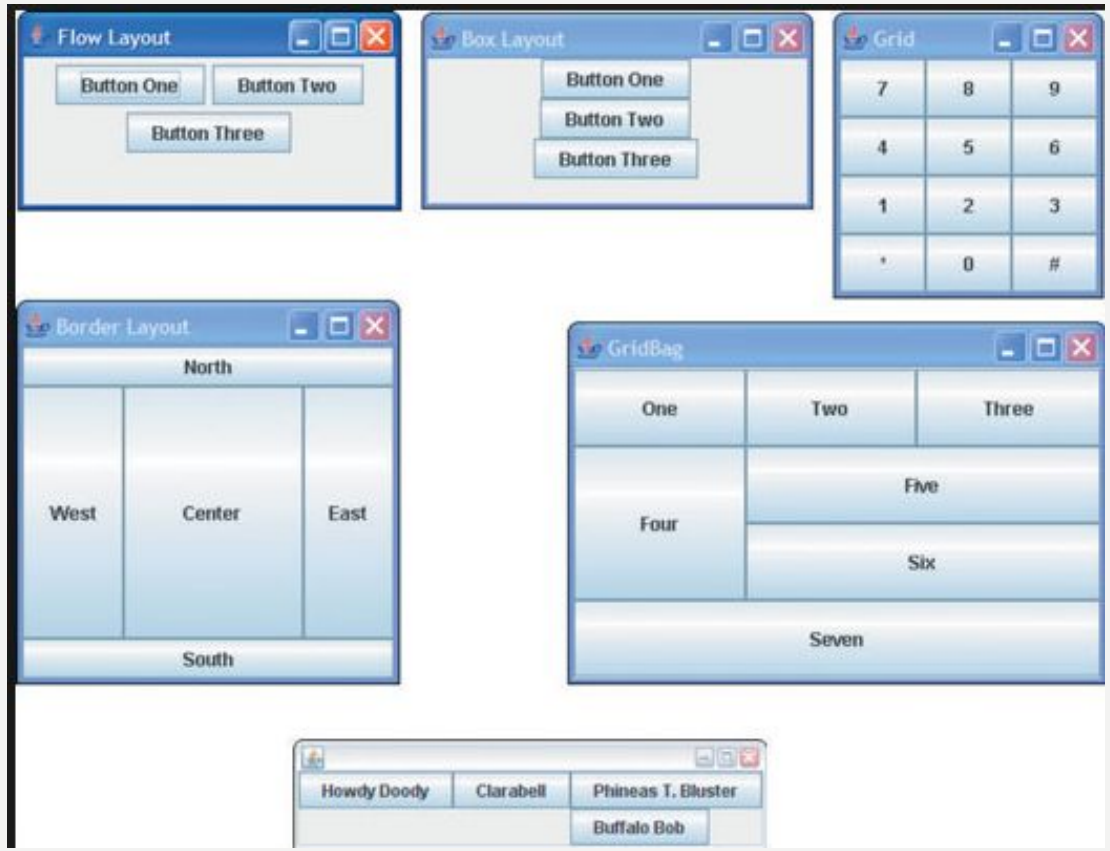
- AWT – старый, не рекомендуется для использования
- Swing – получше
- FX – начиная с Java8, сам не пробовал.

Заявлена кроссплатформенность.

Frame - окно

Контейнер и менеджер раскладки

Текст1 Текст2 Текст3



ПОТОКИ И ПОТОКОБЕЗОПАСНОСТЬ

java.lang

Class StringBuilder

java.lang.Object
    java.lang.StringBuilder

All Implemented Interfaces:

Serializable, Appendable, CharSequence

```
public final class StringBuilder
extends Object
implements Serializable, CharSequence
```

A mutable sequence of characters. This class provides an API compatible with `StringBuffer`, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for `StringBuffer` in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to `StringBuffer` as it will be faster under most implementations.

The principal operations on a `StringBuilder` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string builder. The `append` method always adds these characters at the end of the builder; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string builder object whose current contents are "`start`", then the method call `z.append("le")` would cause the string builder to contain "`startle`", whereas `z.insert(4, "le")` would alter the string builder to contain "`starlet`".

In general, if sb refers to an instance of a `StringBuilder`, then `sb.append(x)` has the same effect as `sb.insert(sb.length(), x)`. Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger.

Instances of `StringBuilder` are not safe for use by multiple threads. If such synchronization is required then it is recommended that `StringBuffer` be used.

Since:

1.5

See Also:

`StringBuffer`, `String`, Serialized Form

СЕРИАЛИЗАЦИЯ

- Передача объектов на другой компьютер
- Сохранение между запусками

СОКРАЩЕННЫЕ ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

- if(true1 & false 2 & false 3) – выполнятся все проверки
- if(true1 && false 2 && false 3) – выполнятся только первые 2, на втором шаге значение всего выражения уже понятно)
- if(true1 | false 2 | false 3) – выполнятся все проверки
- if(false1 || true 2 || false 3) – выполнится только 2

УПРОЩЕННОЕ ПРИСВАИВАНИЕ

- totalPayment += getOptionPrice(Option);
- totalPayment -= getFixDiscount();
- totalPatment *= 1 – getPercentDiscount()/100;
- totalPayment /= getSeasonKoefficient;

```
--curStep;
++curStep;

curStep = 0;
++curStep * 2 + curStep-- = ?
```