Объектно-ориентированные CASE-технологии

Метамодель языка UML.

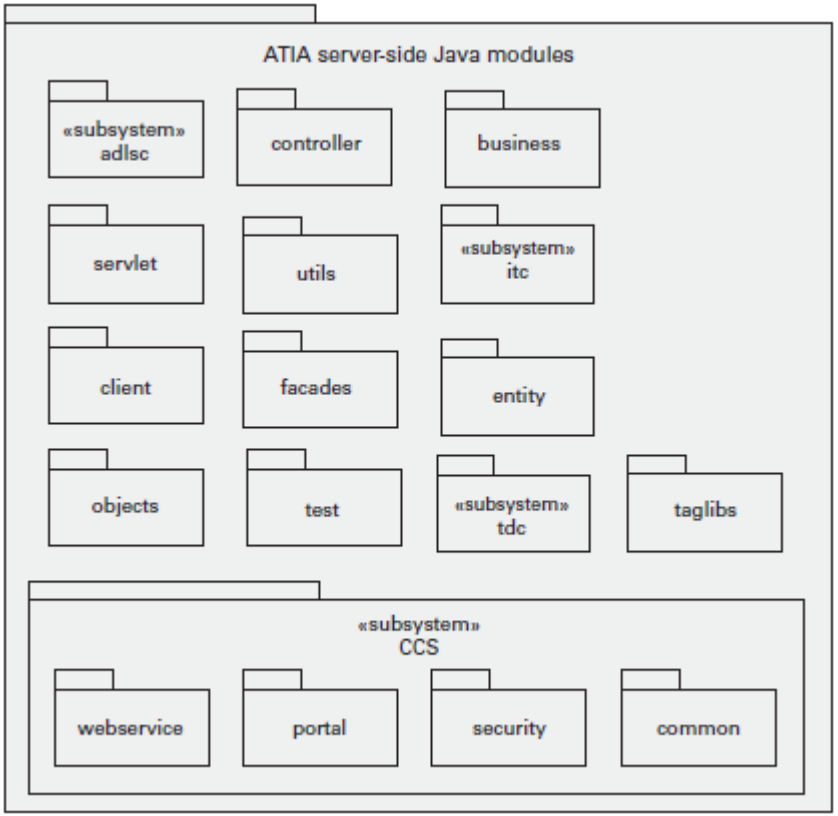
Практикум: визуализация UML-моделей программ и данных

Романов Владимир Юрьевич,
Московский Государственный Университет им. М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
vromanov@cs.msu.su,
romanov.rvy@yandex.ru

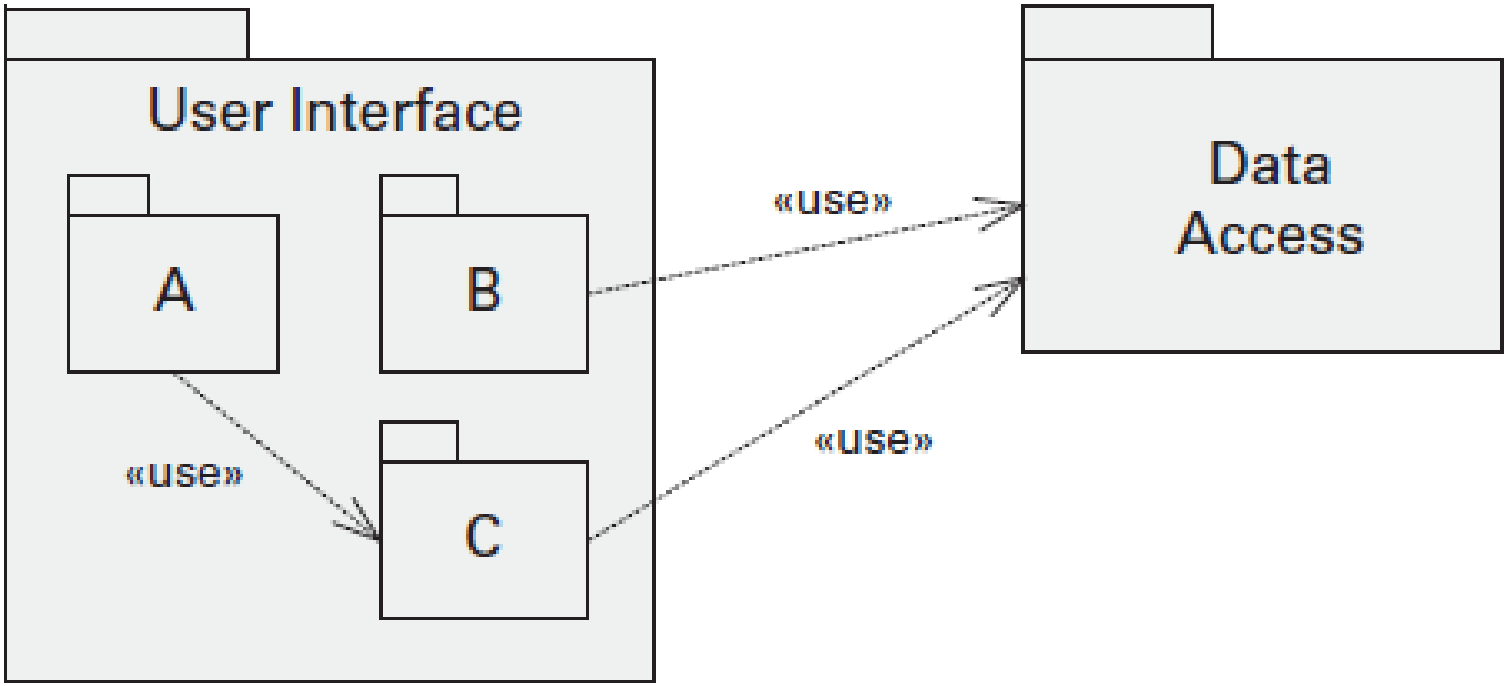
1. Визуализация программы

1.1. Нотация языка UML

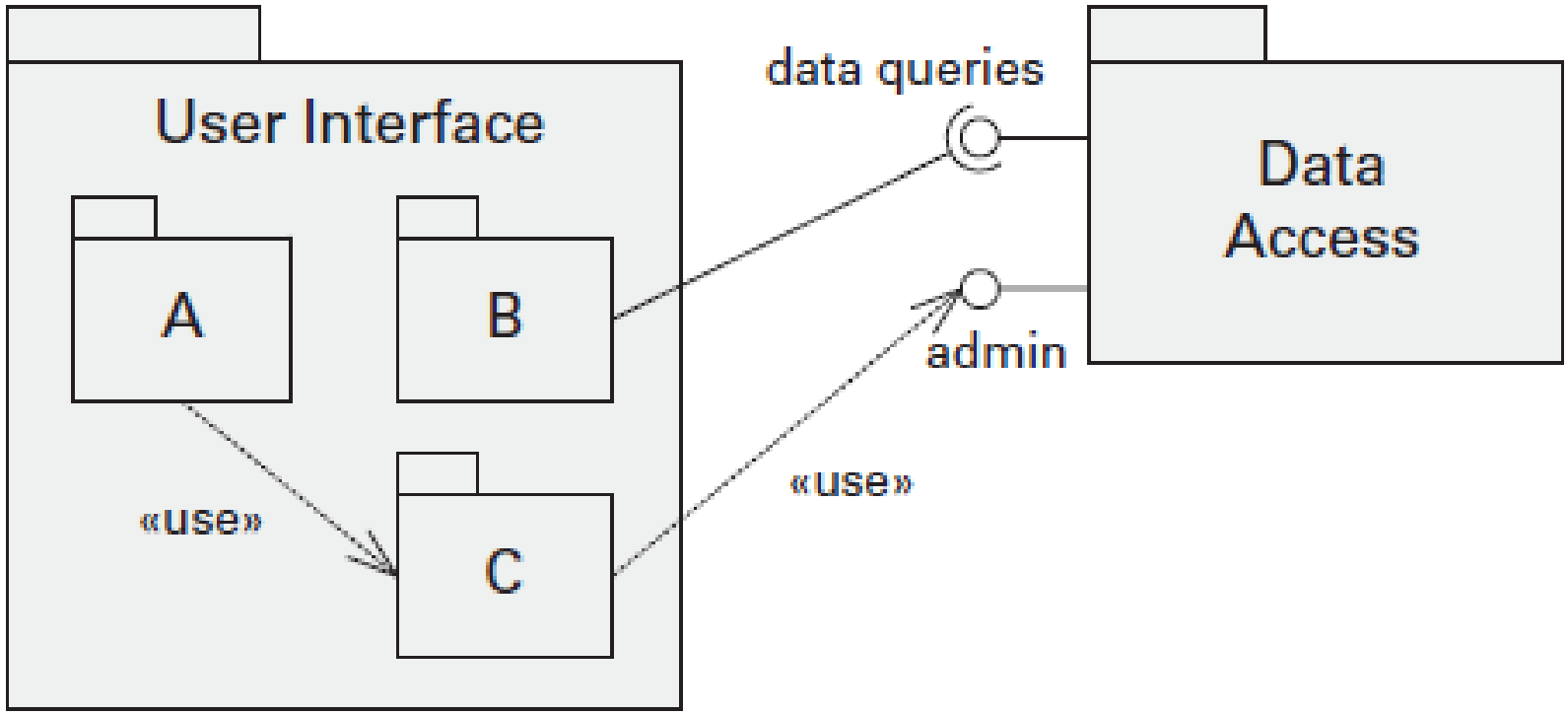
Нотация UML. Структура системы



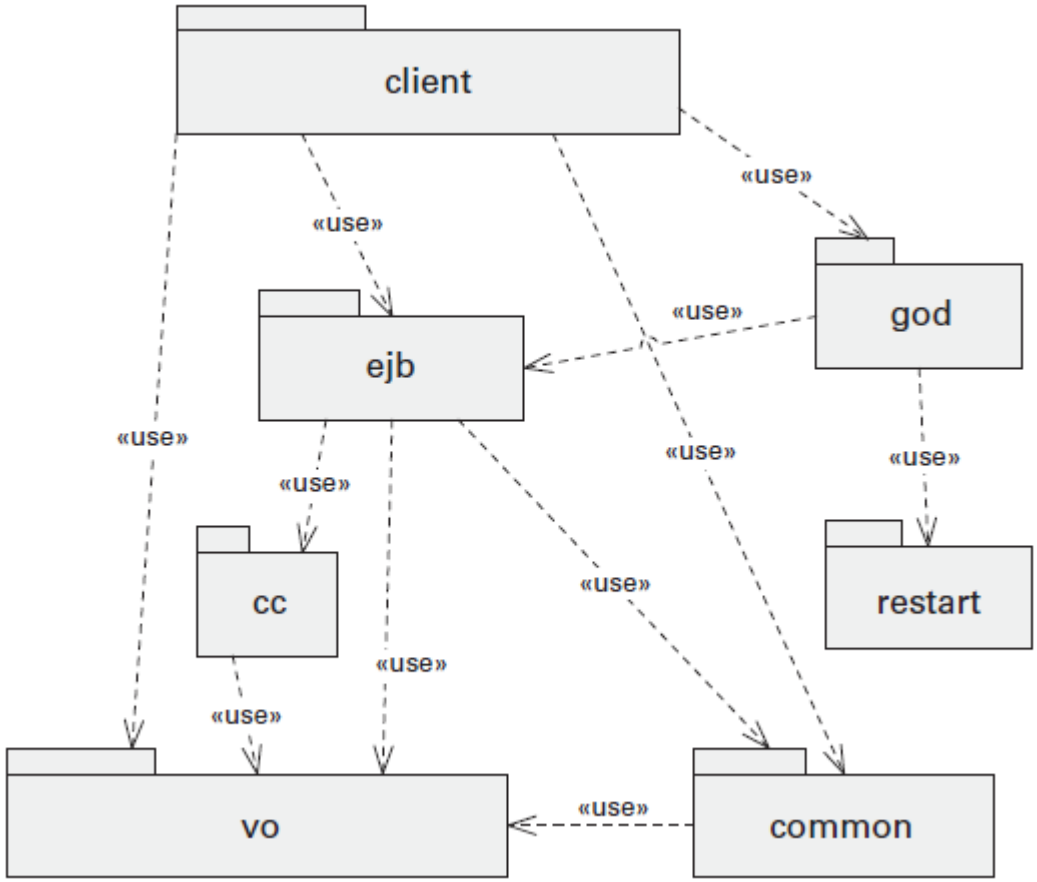
Нотация UML. Зависимости между пакетами



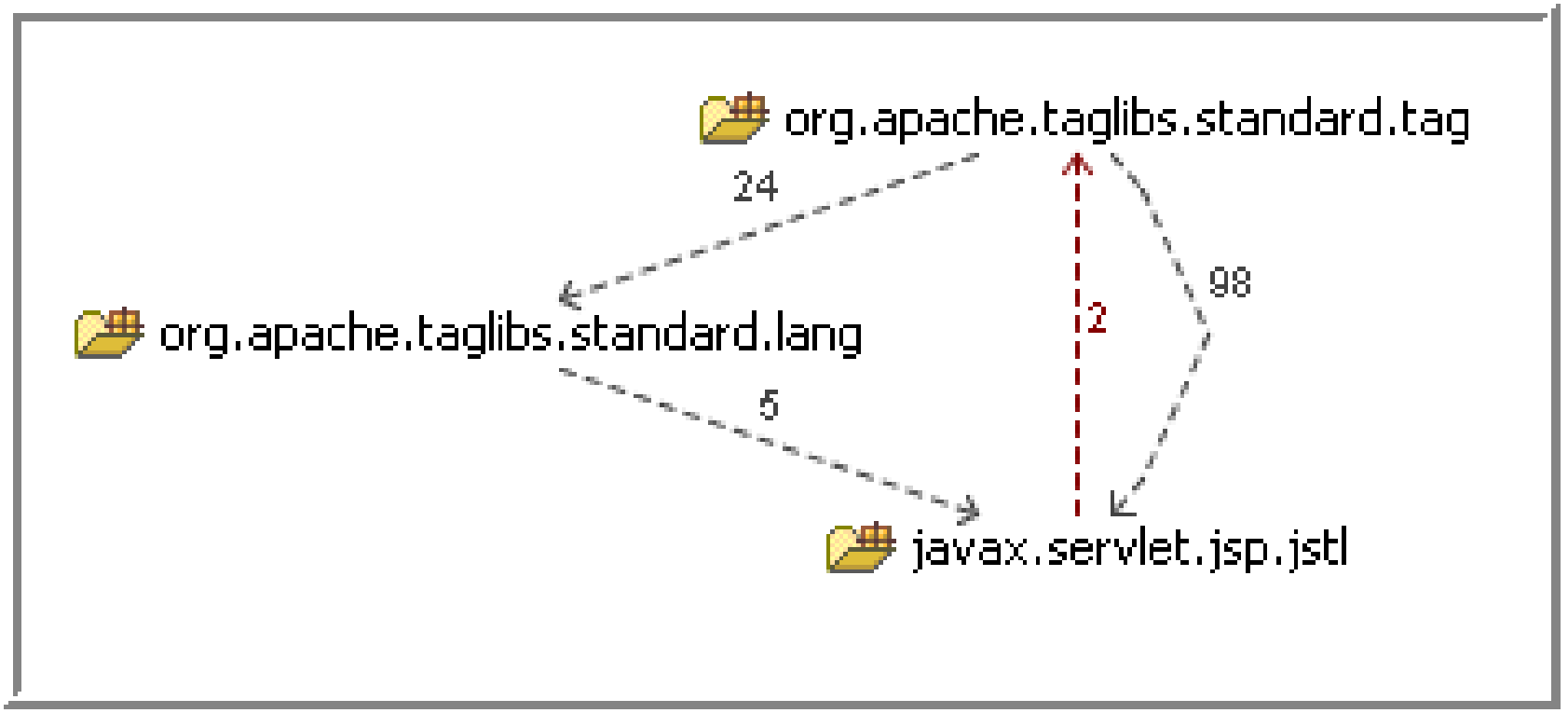
Нотация UML. Зависимости между пакетами. Уточнение - зависимость через интерфейсы



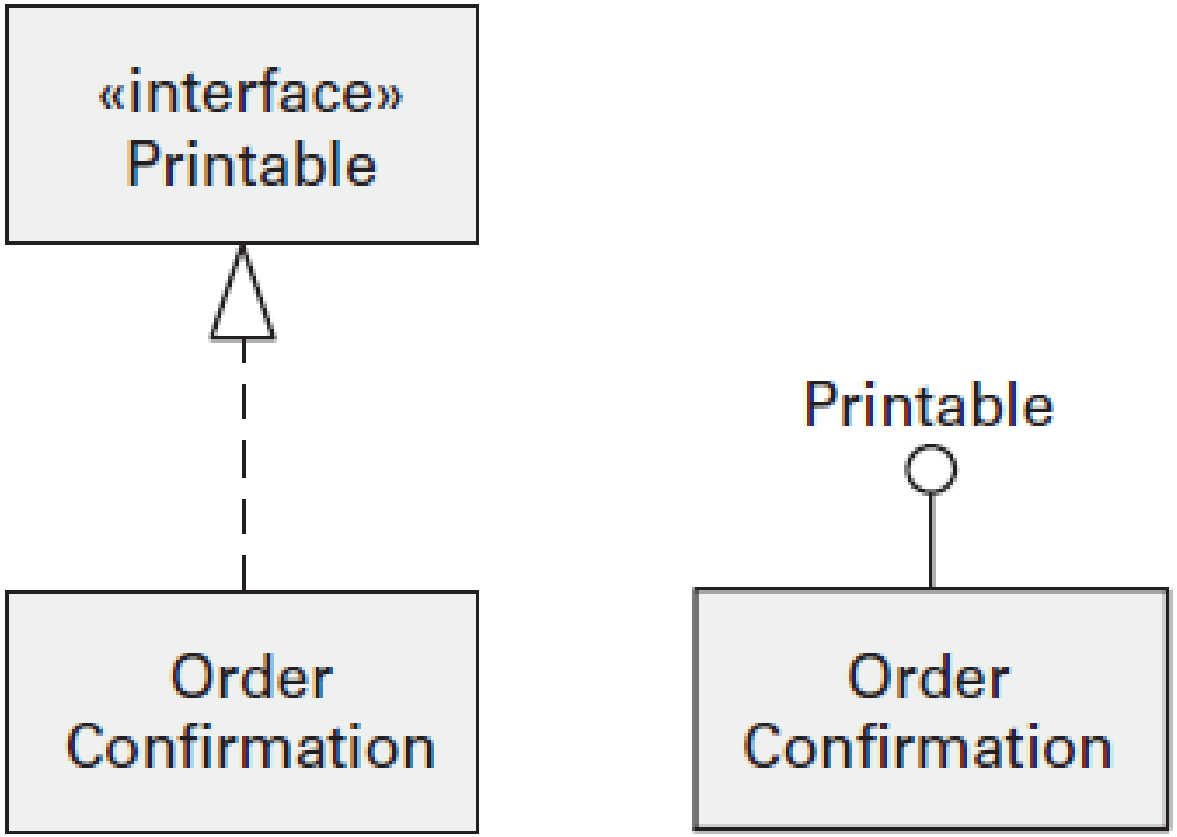
Нотация UML.
По уровневая визуализация пакетов системы



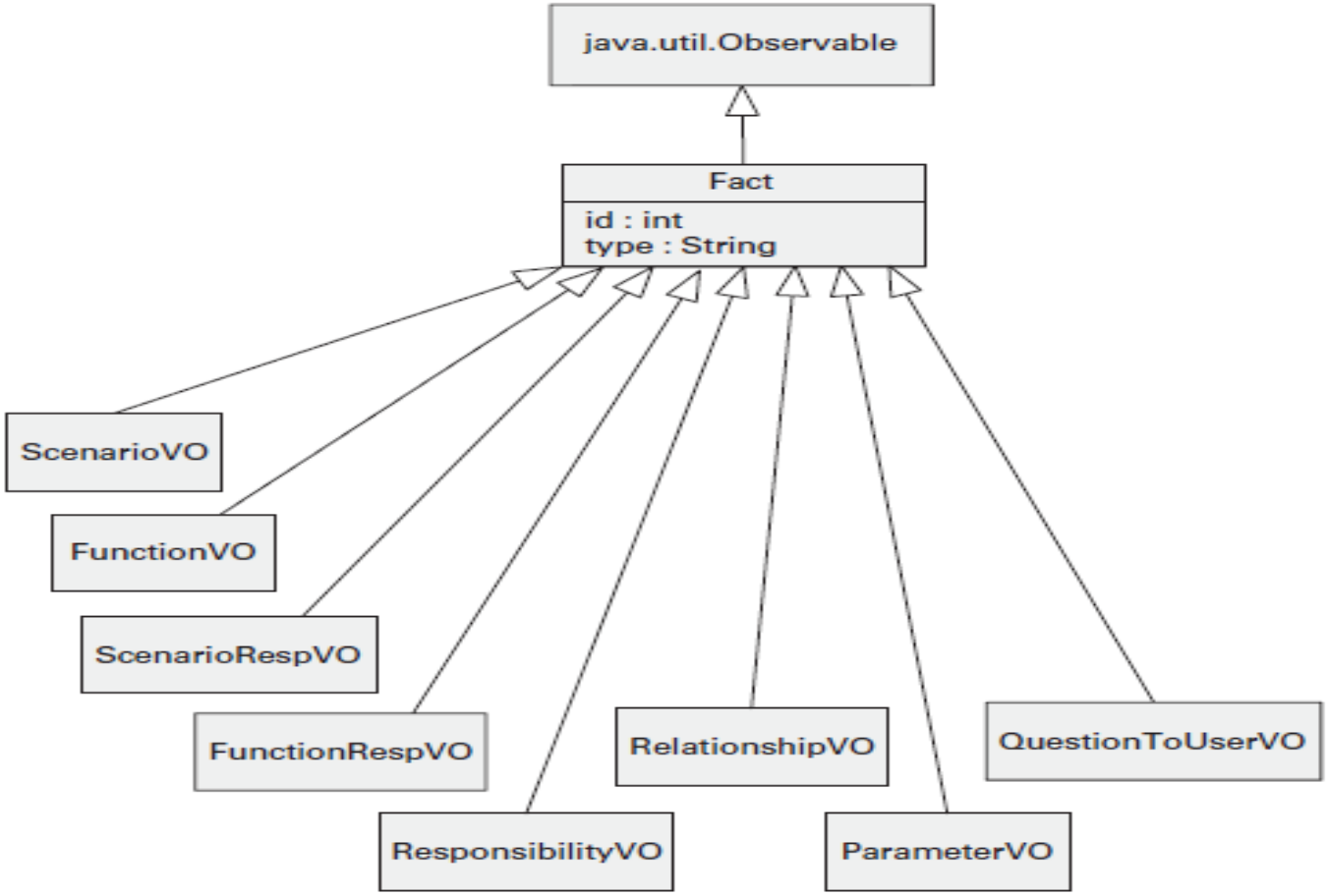
1.1. Нотация UML.
Оценка степени зависимости между пакетами системы



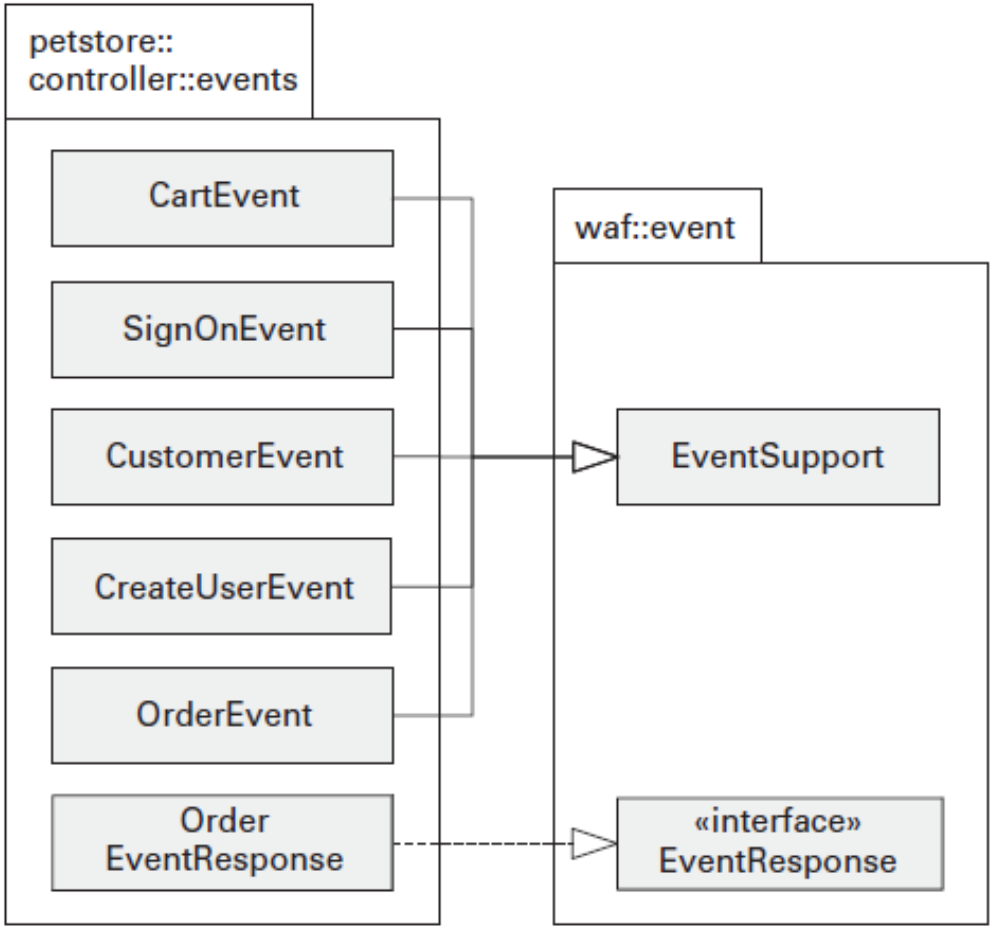
Нотация UML.
Реализация классами интерфейсов



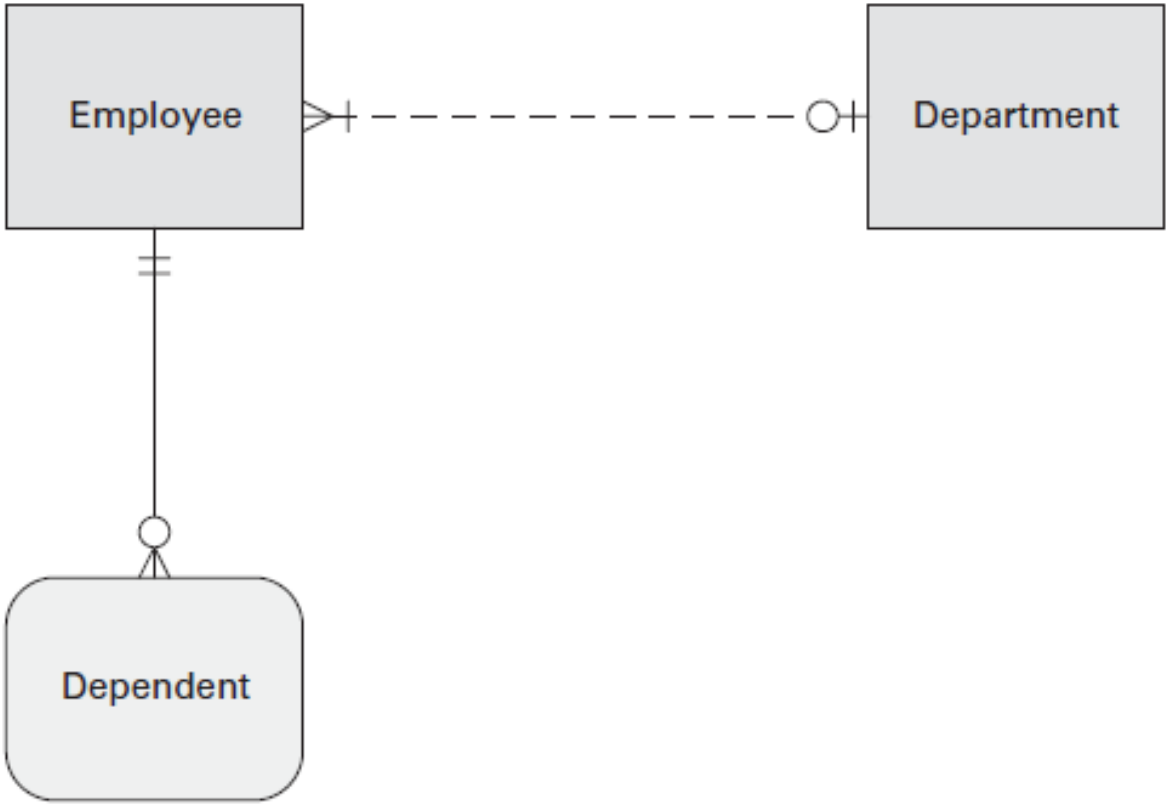
Нотация UML. Отношения наследования для класса



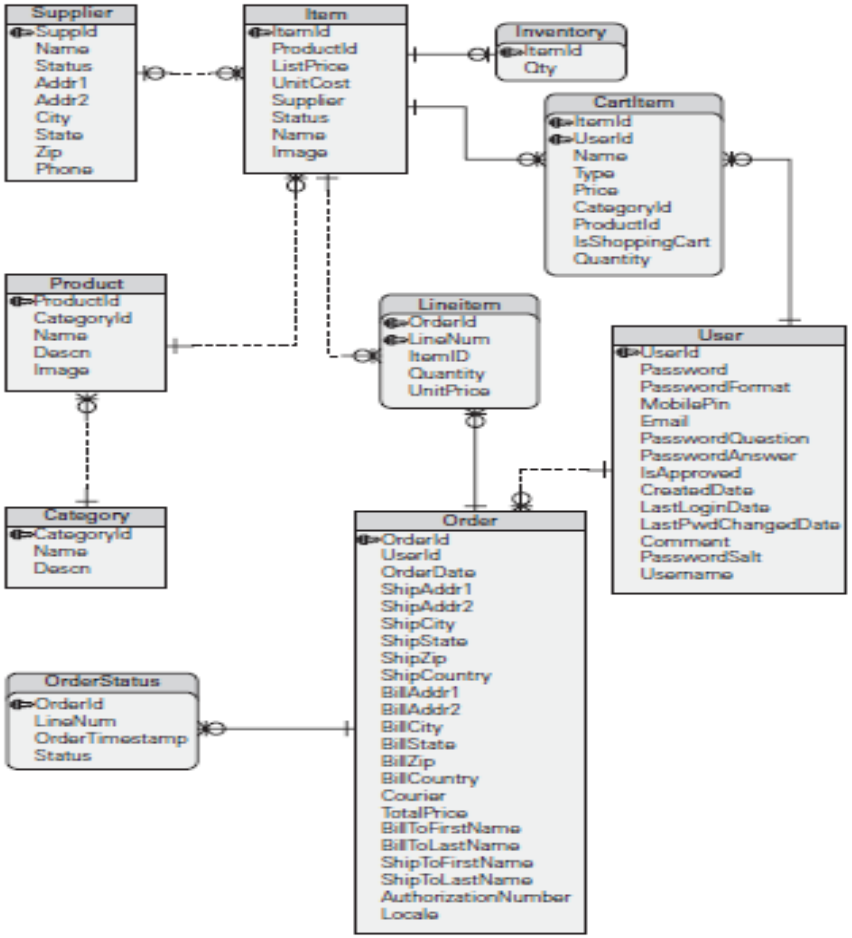
Нотация UML. Отношения наследования при визуализации пакетов



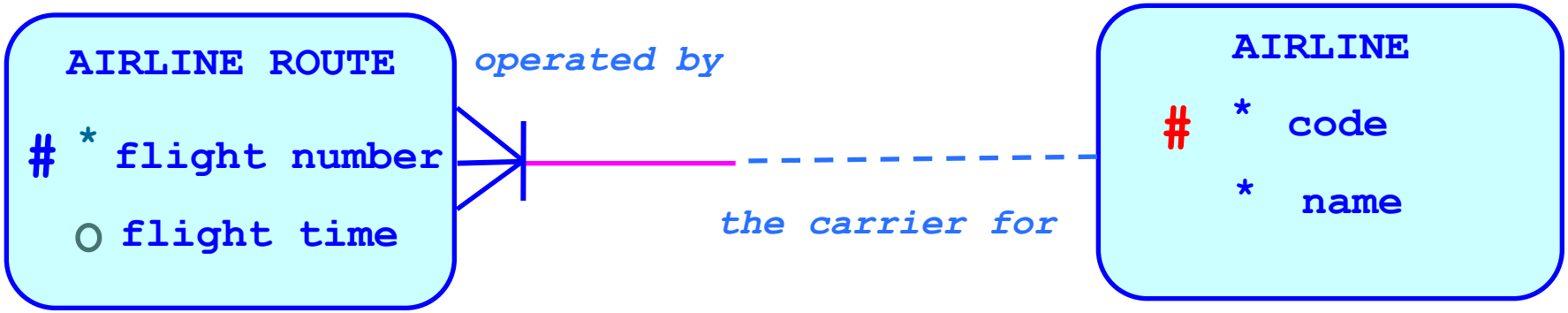
Нотация Entity-Relationship (Сущность-Отношение). Моделирование данных



Нотация Entity-Relationship (Сущность-Отношение). Моделирование данных



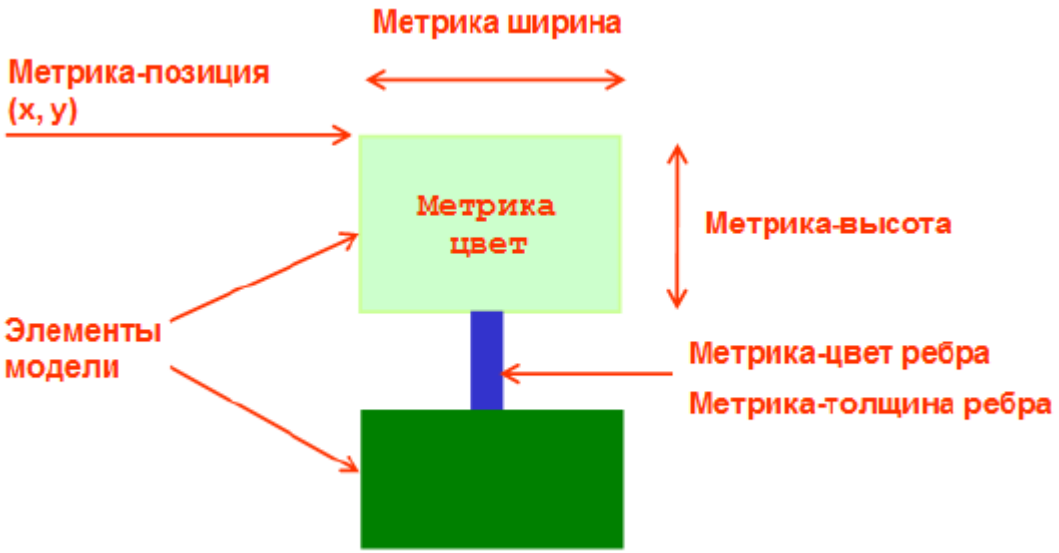
Нотация Entity-Relationship (Сущность-Отношение). от фирмы Oracle



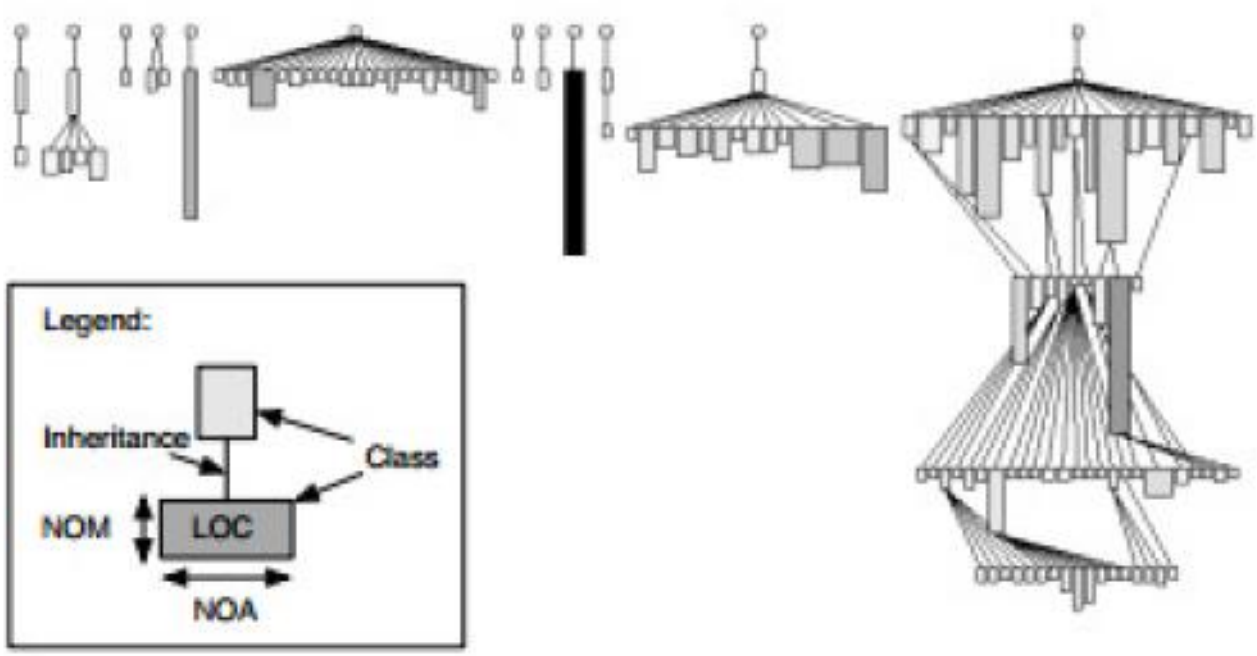
1. Визуализация программы

1.2. Полиметрические виды

1.1. Полиметрические виды. Принципы формирования

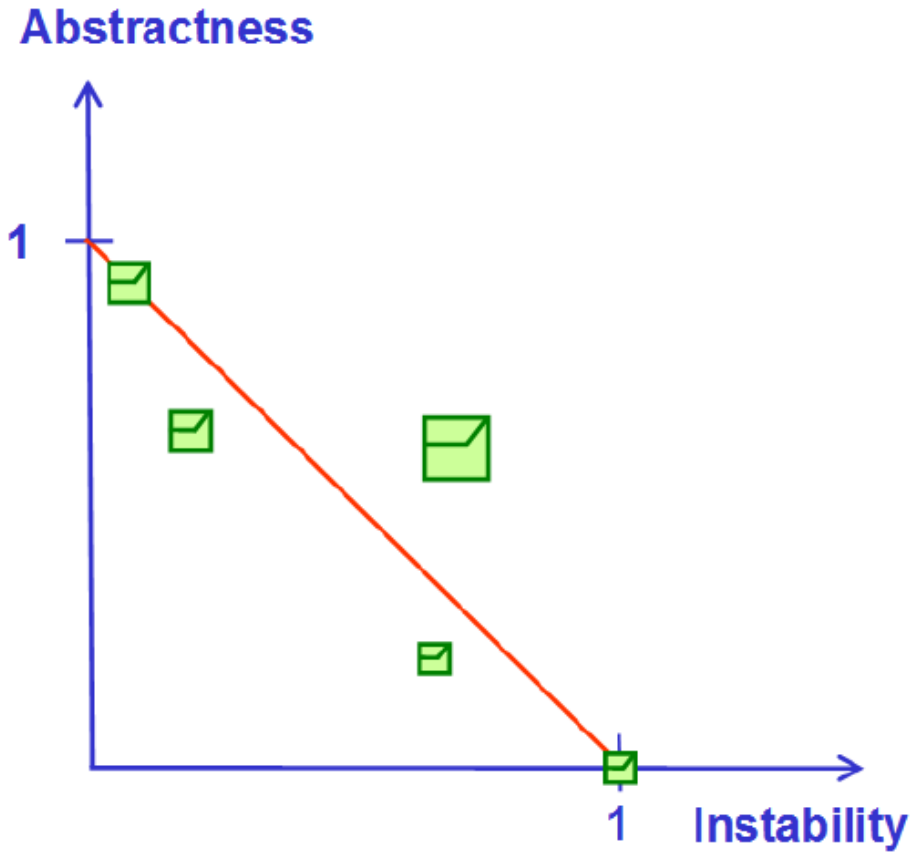


1.1. Полиметрические виды. Пример использования размера и цвета в диаграмме сложности пакета



- **LOC** – (Lines Of Code) строк кода
- **NOM** – (Number Of Methods) количество методов
- **NOA** – (Number Of Attributes) количество атрибутов

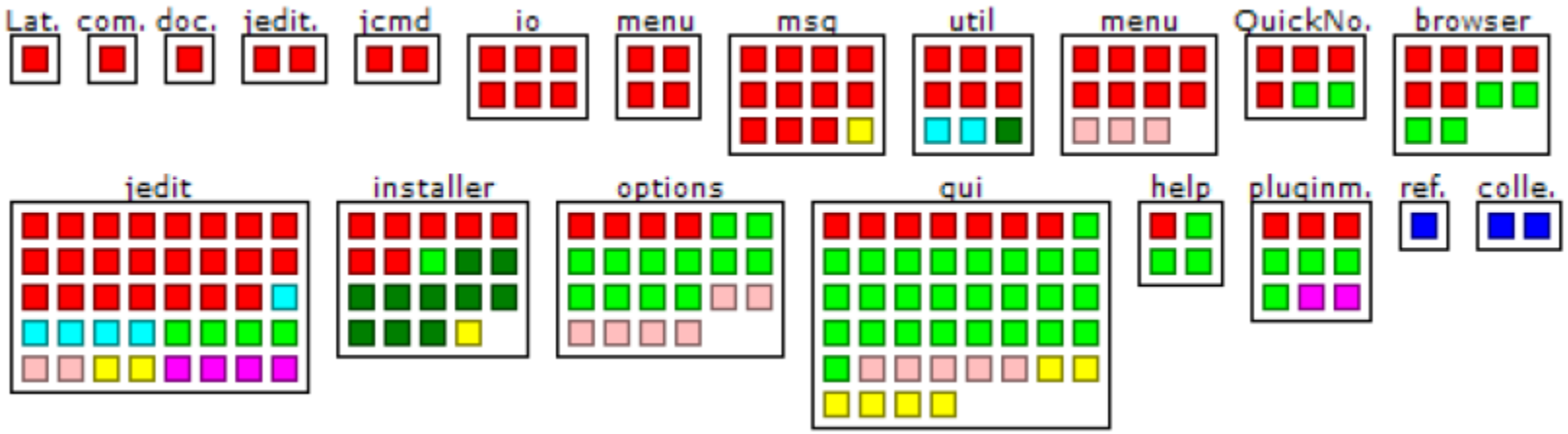
1.2. Полиметрические виды. Пример использования размера и координат для визуализации абстрактности и нестабильности пакета



1. Визуализация программы

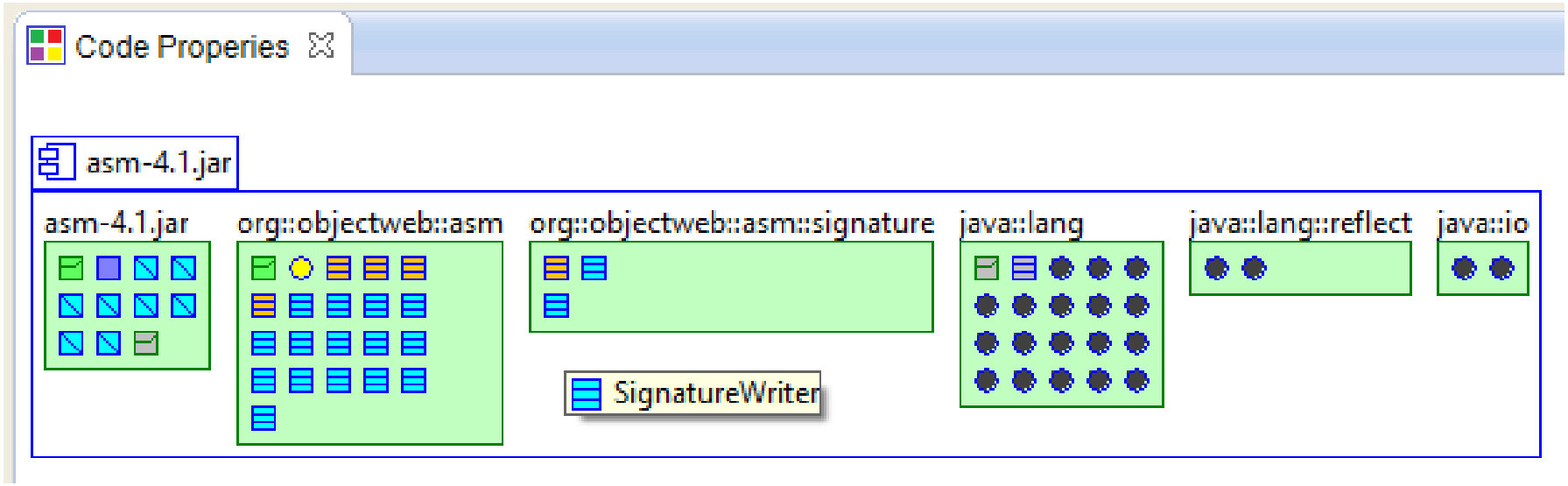
1.3. Карты распределения

Карты распределения. Распределение свойств среди кода объектов



- **Красный – классы ядра расположенные в верху иерархии наследования**
- **Синий – классы интерфейса пользователя**
- **Зеленый – внутренние классификаторы программы**

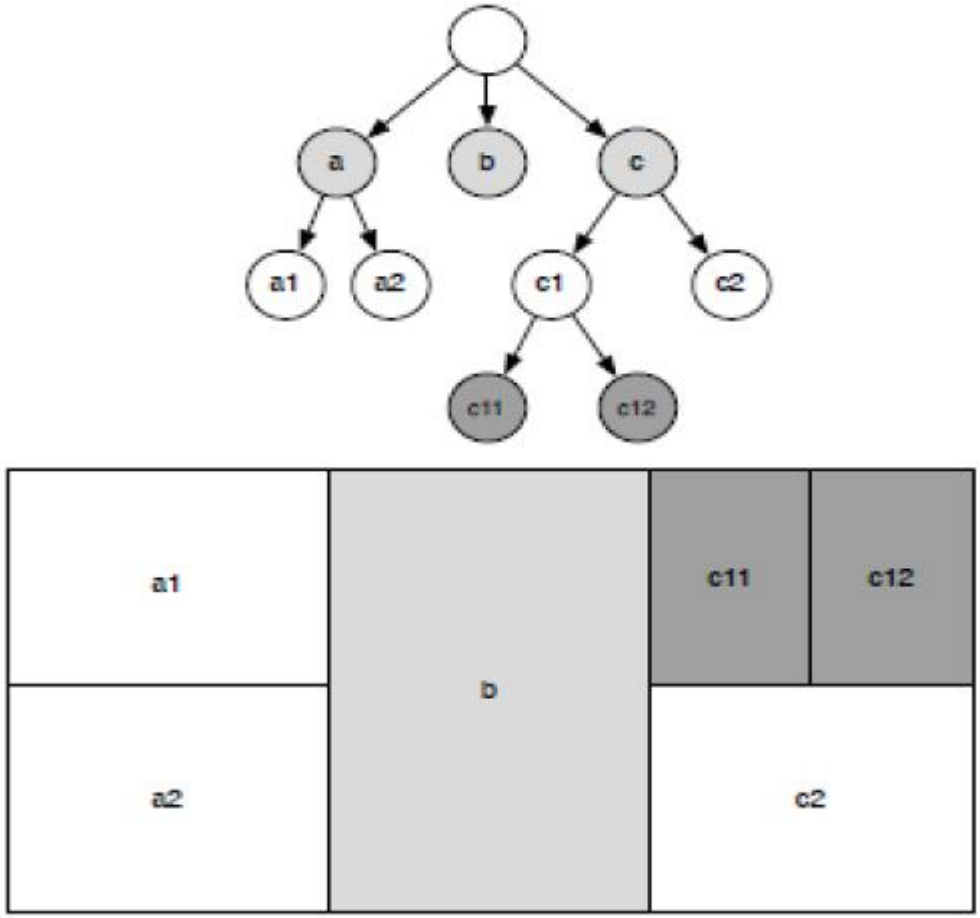
Карты распределения.
Распределение свойств среди кода объектов



1. Визуализация программы

1.4. Деревья-карты распределения

1.4. Деревья - карты. Принцип формирования

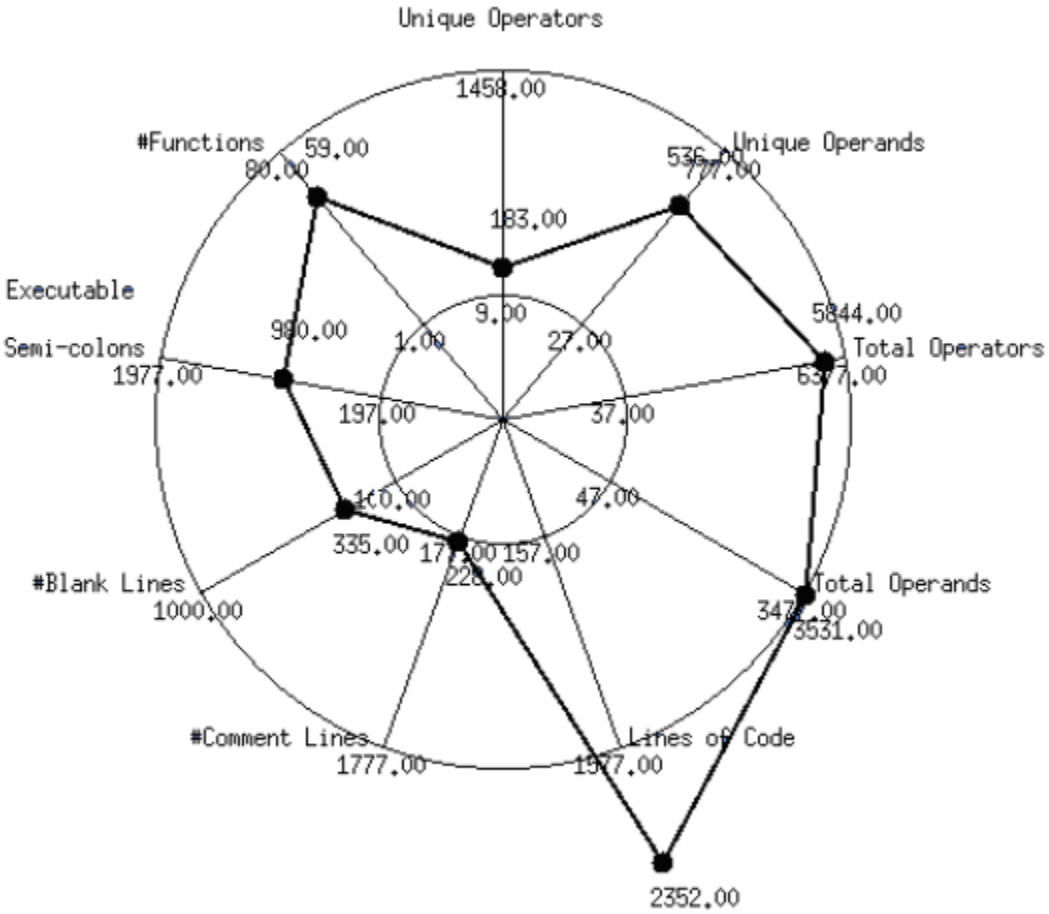


1.4.1. Деревья - карты. Визуализация свойств классов с помощью деревьев-карт

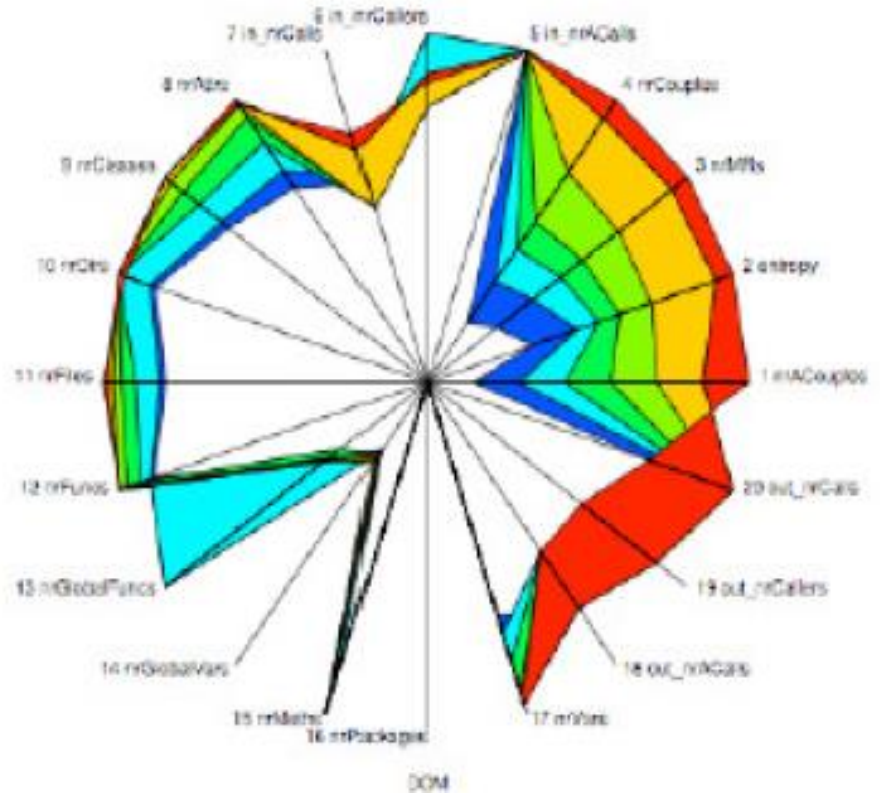
1. Визуализация программы

1.5. Полярные диаграммы

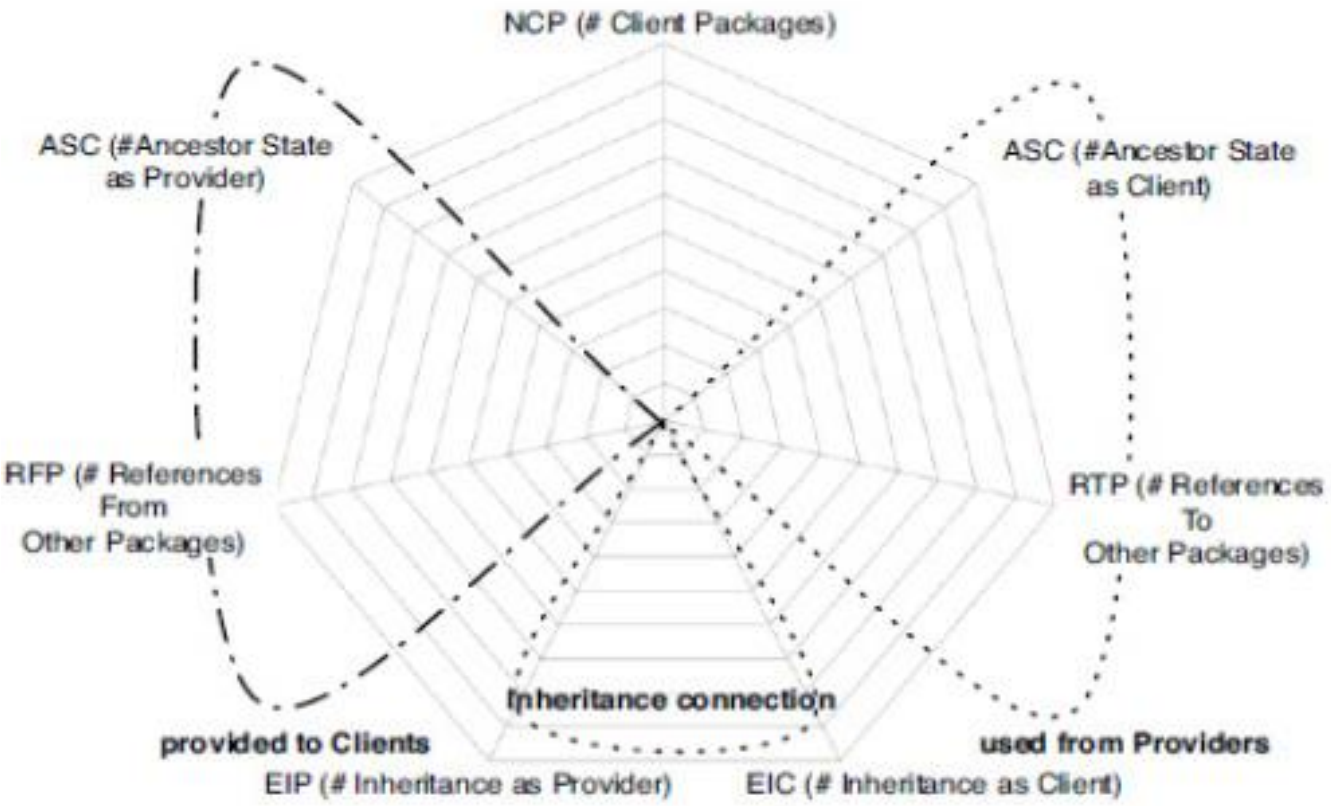
Визуализация метрик с помощью полярной диаграммы. Метрики класса



2.2. Визуализация эволюции класса с помощью полярной диаграммы.



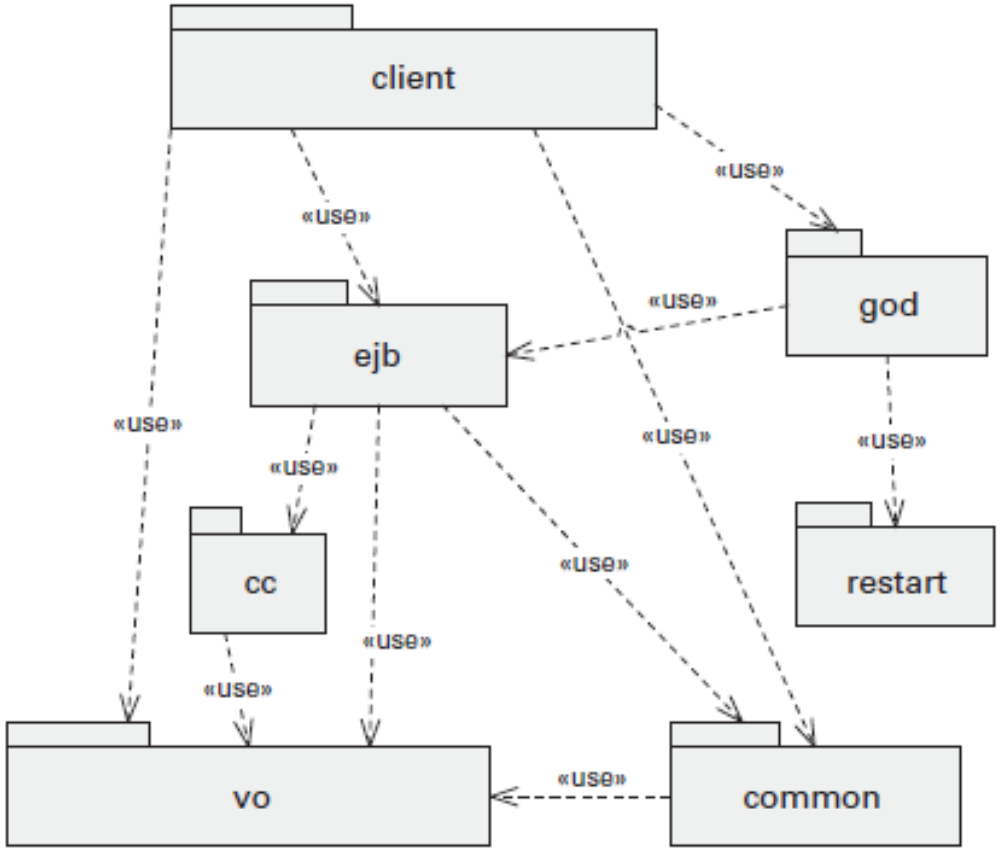
2.3. Визуализация связей и метрик пакета с помощью полярной диаграммы - бабочки.



1. Визуализация программы

1.6. Матрицы структурной зависимости

Матрица структурной зависимости. Пакеты для визуализации матрицей

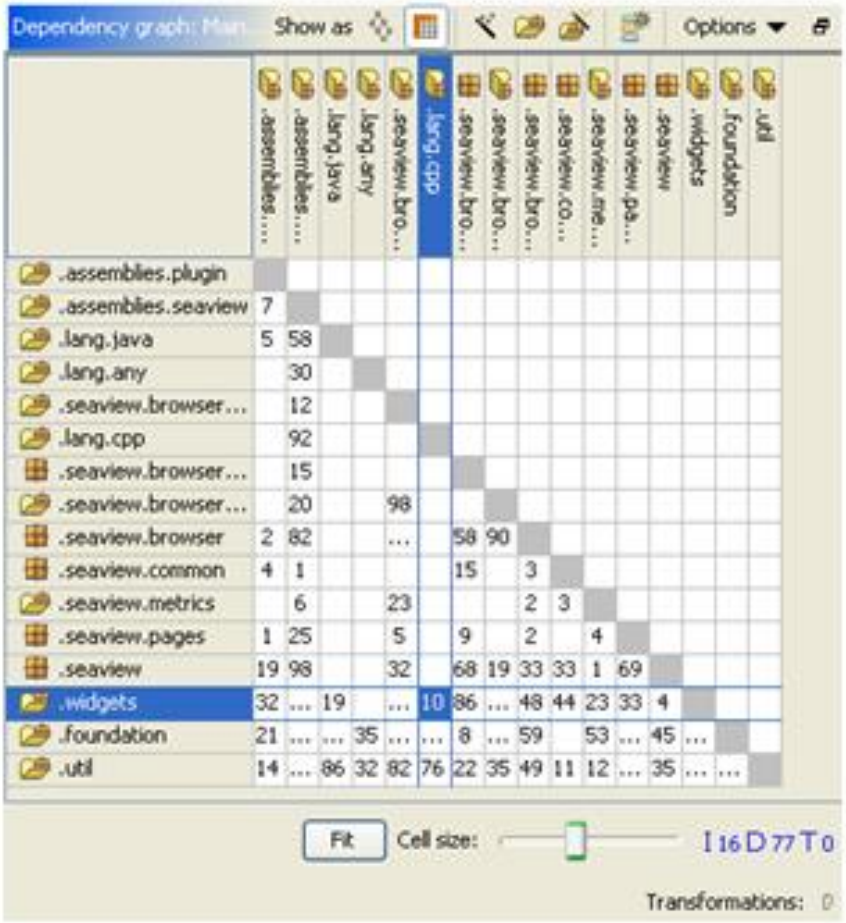


Матрица структурной зависимости (DSM). Визуализация матрицей

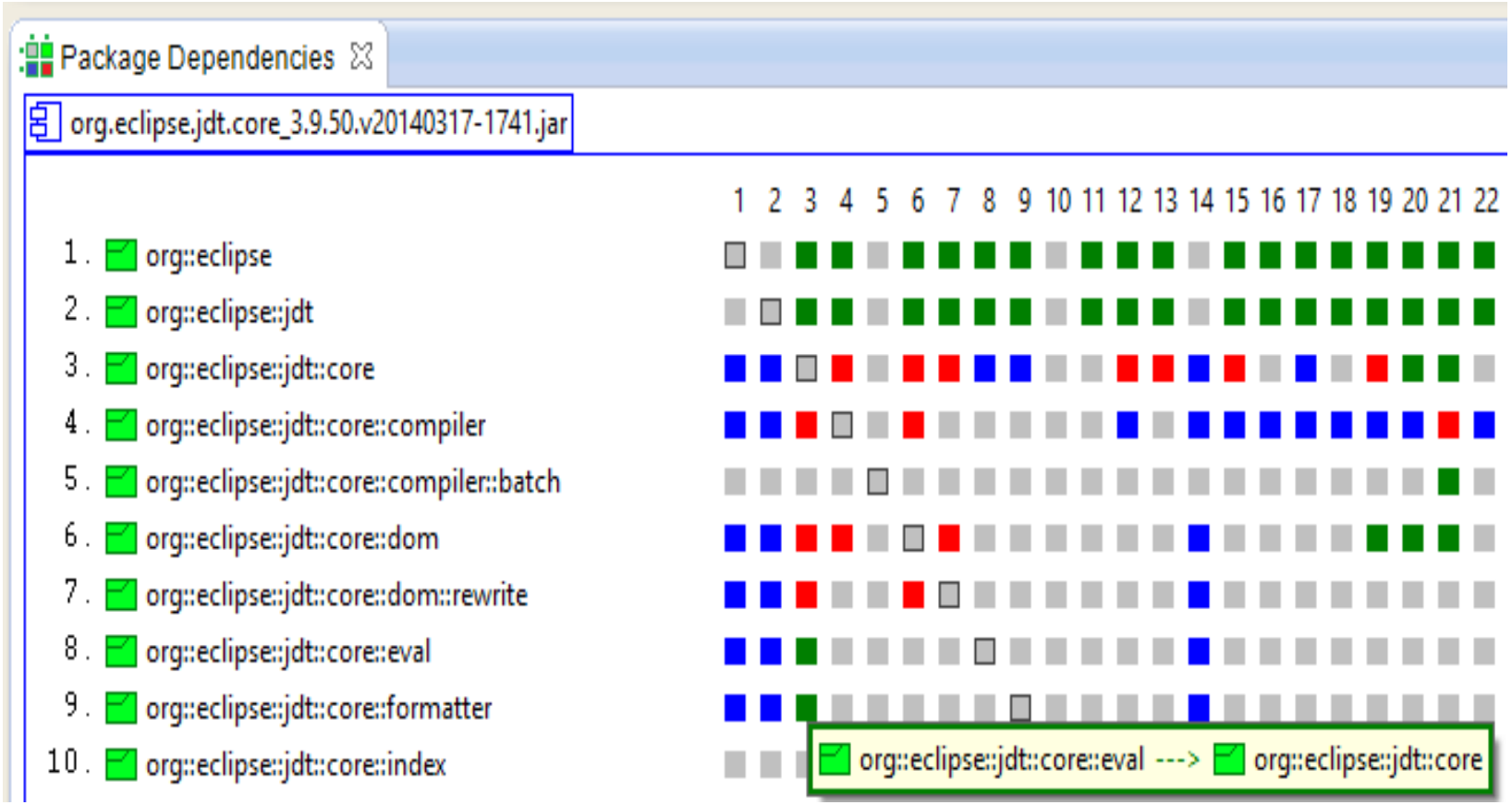
http://www.dsmweb.org/

using module / used module	client	ejb	cc	god	restart	common	vo
client	0	0	0	0	0	0	0
ejb	**1**	0	0	**1**	0	0	0
cc	0	**1**	0	0	0	0	0
god	**1**	0	0	0	0	0	0
restart	0	0	0	**1**	0	0	0
common	**1**	**1**	0	0	0	0	0
vo	**1**	**1**	**1**	0	0	**1**	0

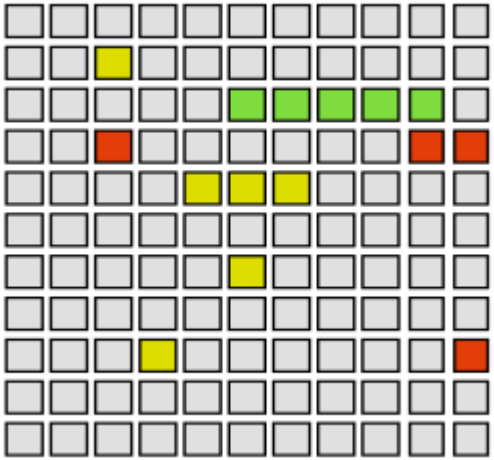
Матрица структурной зависимости
Анализ степени зависимости.



3.1. Визуализация с помощью точечной диаграммы. Визуализация связей между пакетами.



3.2. Визуализация с помощью точечной диаграммы. Визуализация связей между компонентами (jar-файлами).

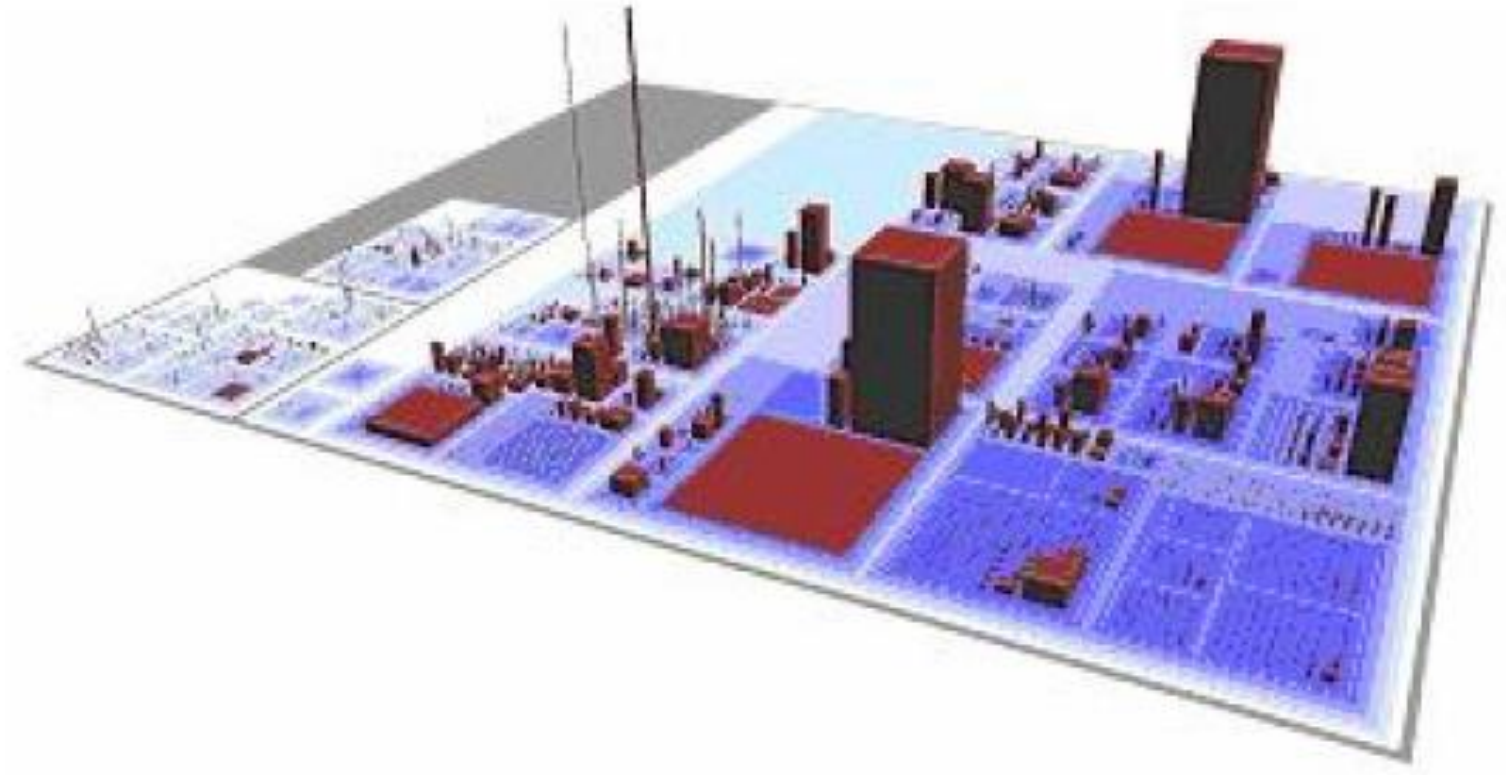


1. Визуализация программы

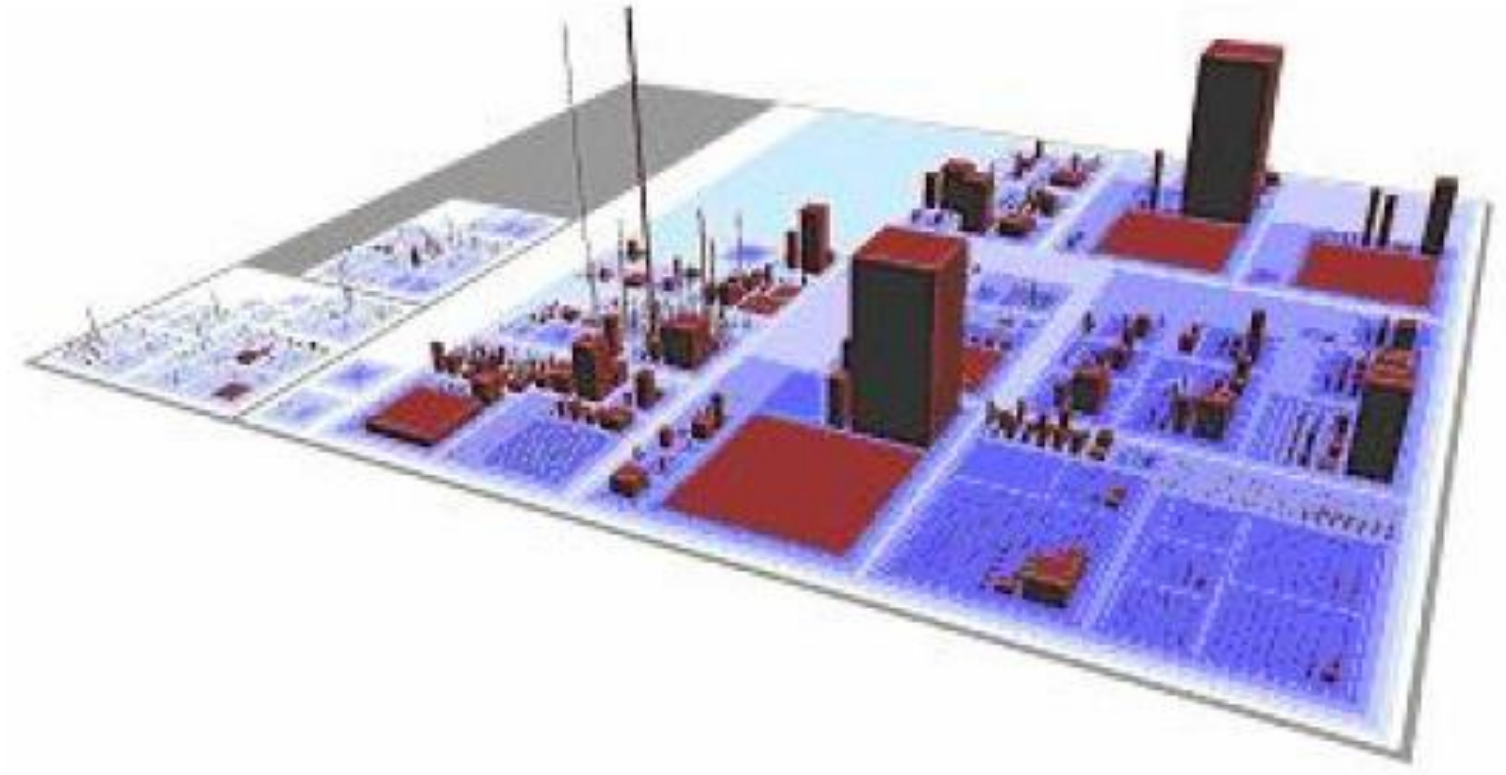
1.7. Метафора города при визуализации программы

1.7. 1. Кварталы города – это пакеты UML (Java)

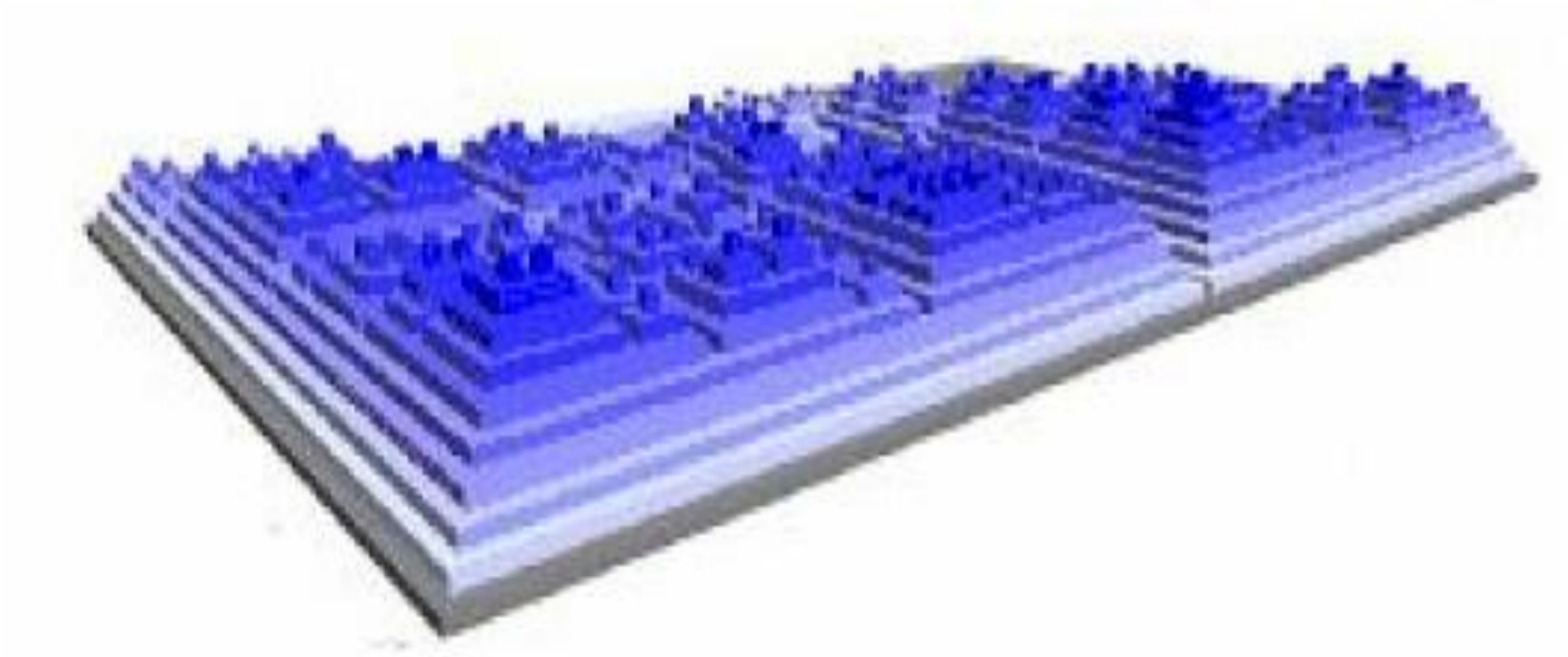
4.1. Визуализация структуры и метрик с помощью карты города.



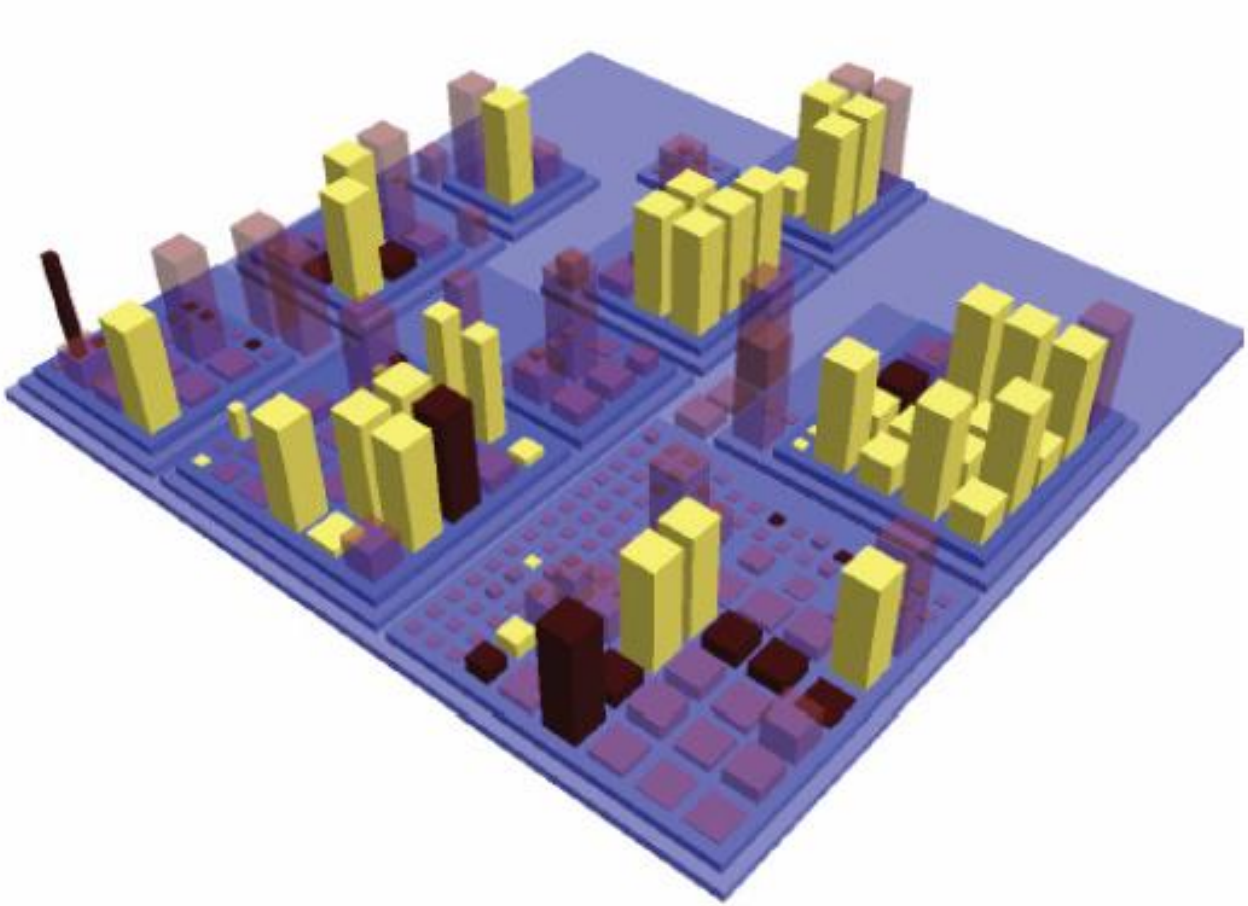
Визуализация структуры и метрик с помощью карты программы-города.



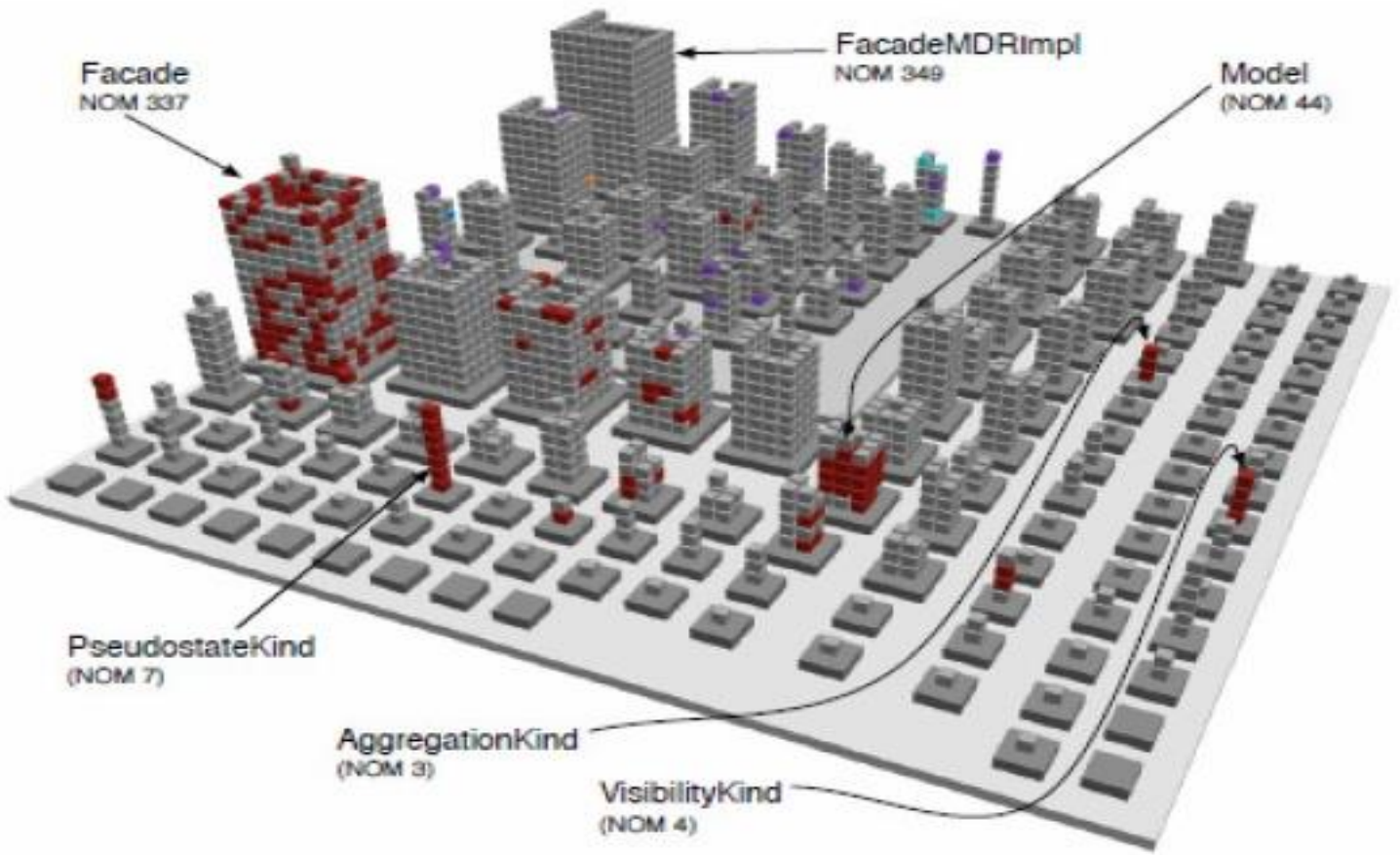
4.1.1. Визуализация топологии программы-города.



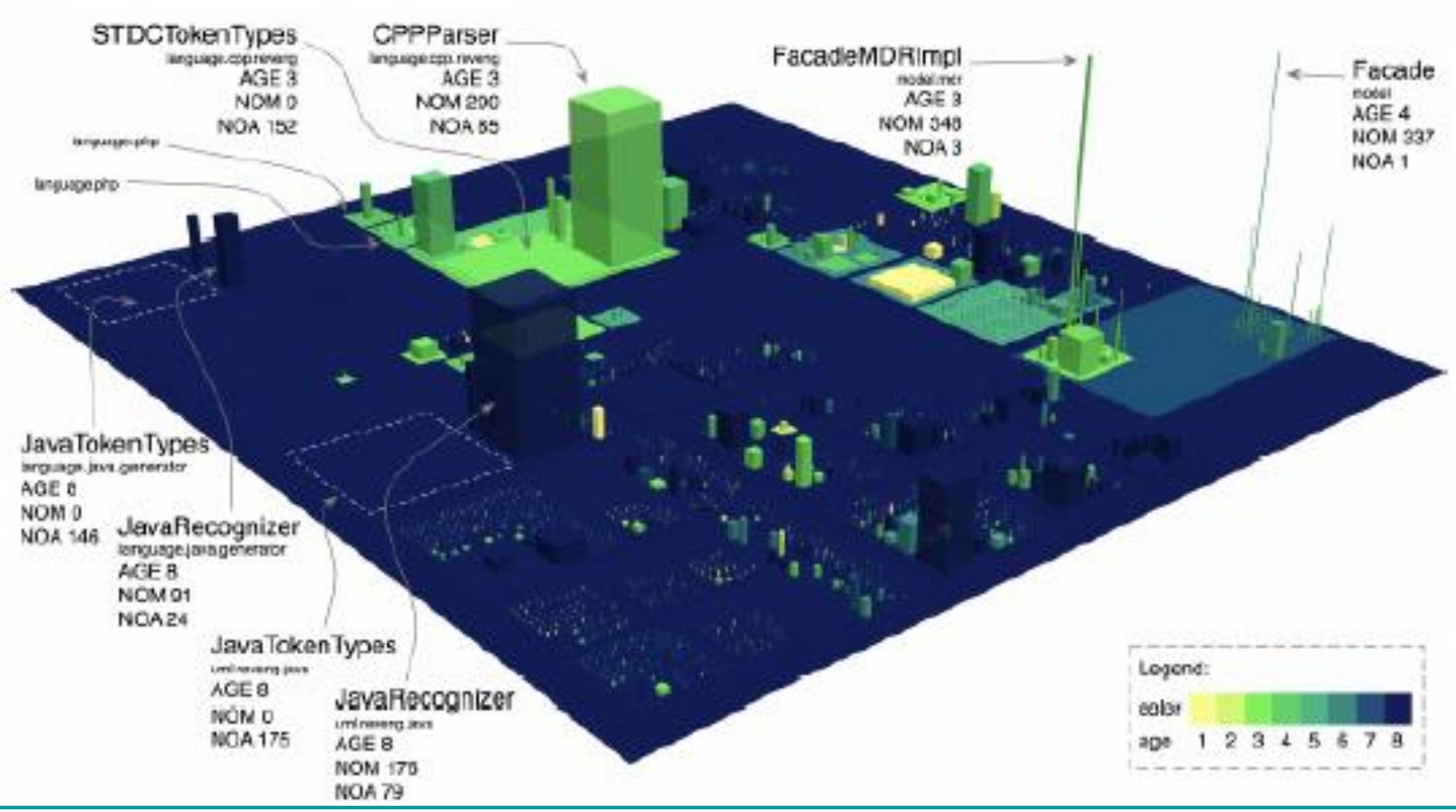
Визуализация свойств классов с помощью расцветки программы-города.



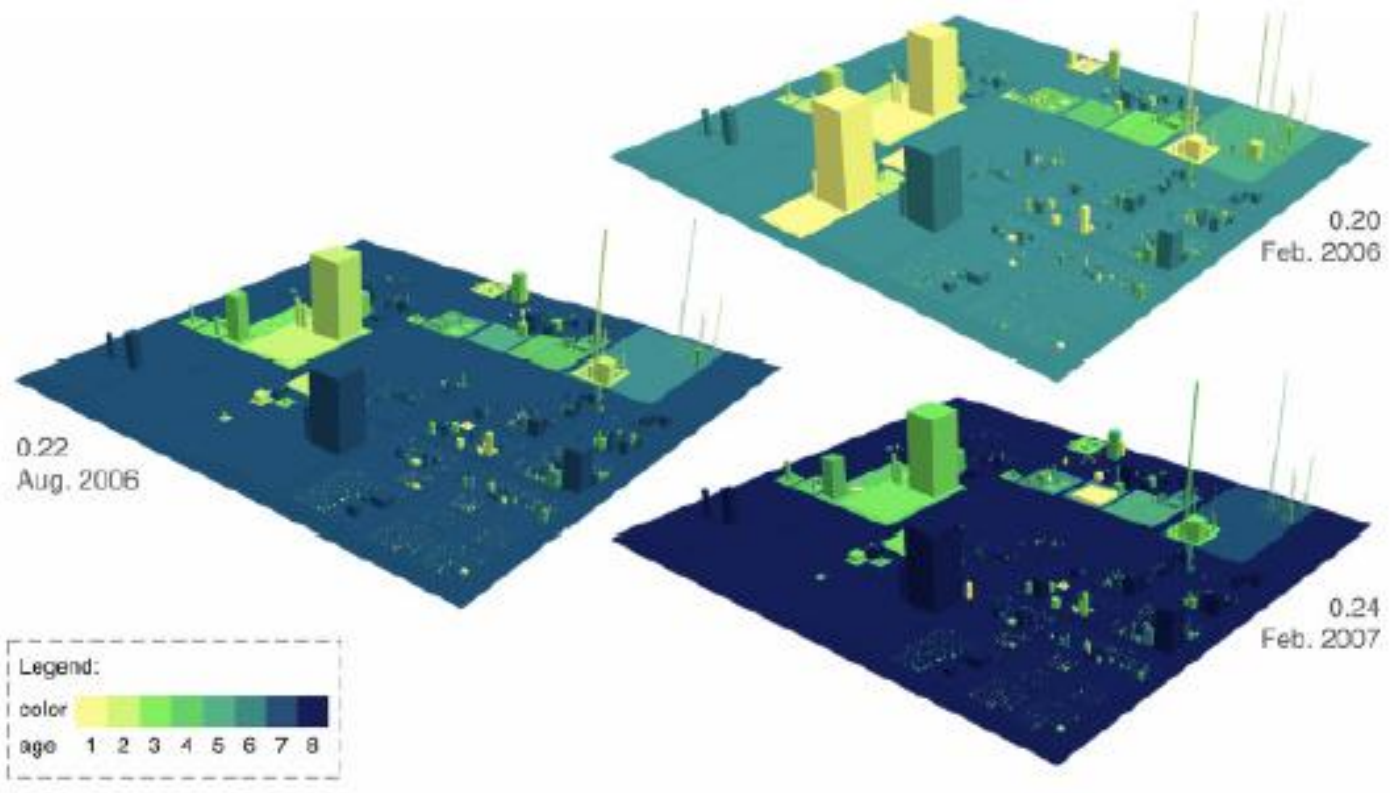
4.1.2. Визуализация методов классов и их дефектов помощью расцветки программы-города.



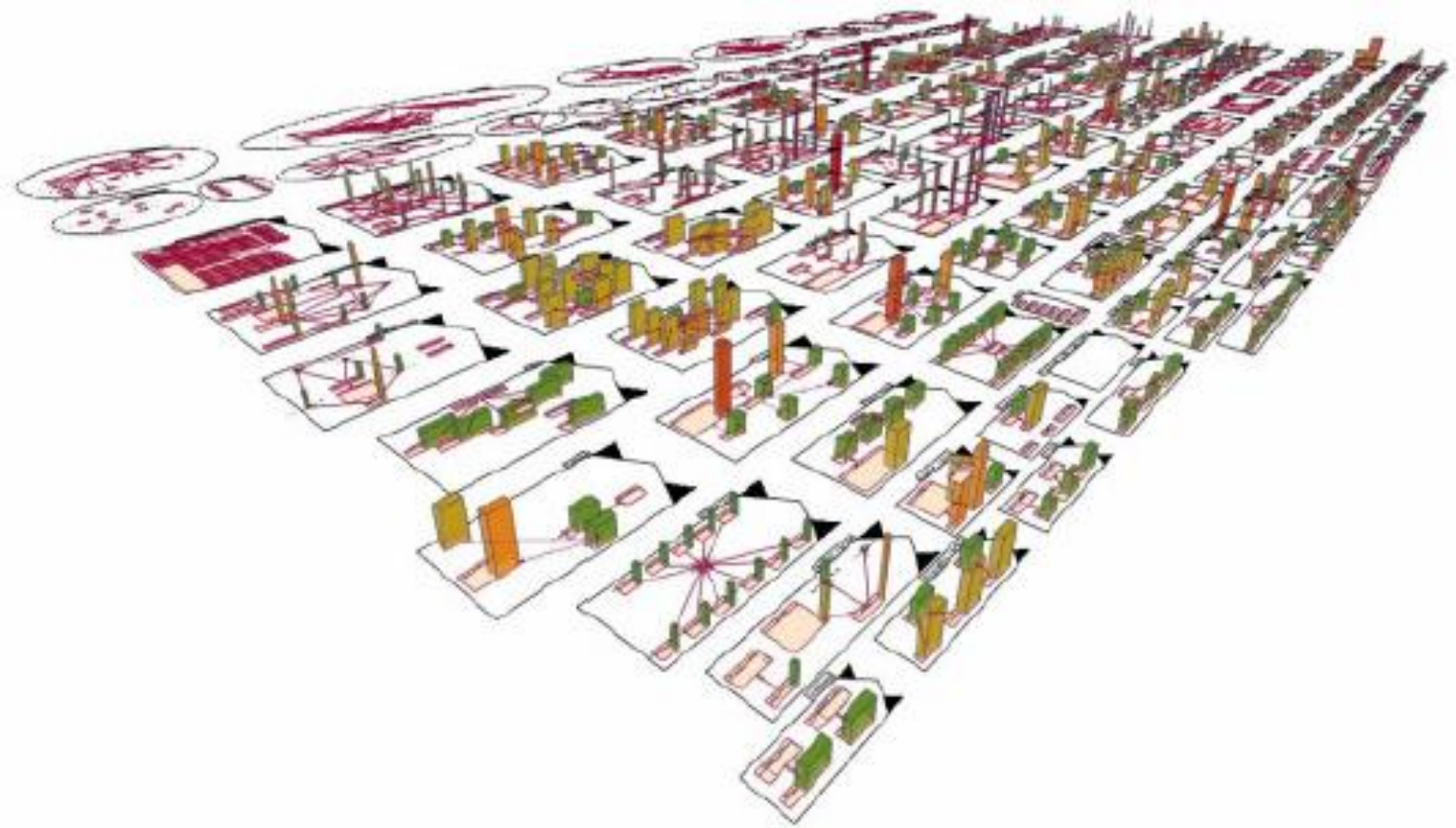
4.2. Визуализация эволюции программы с помощью расцветки программы-города.



4.2.2. Сочетание карты возраста программы и путешествия во времени

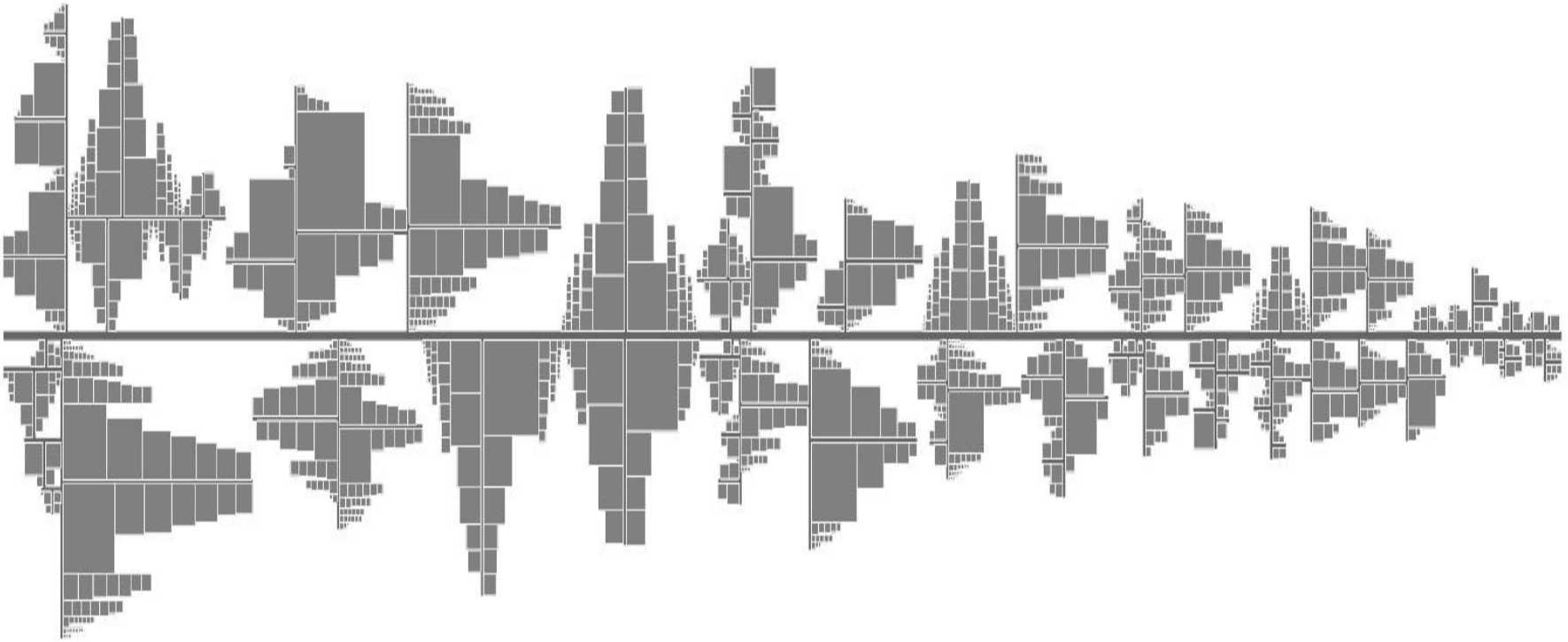


4.3. Сочетание карты программы-города и диаграммы UML

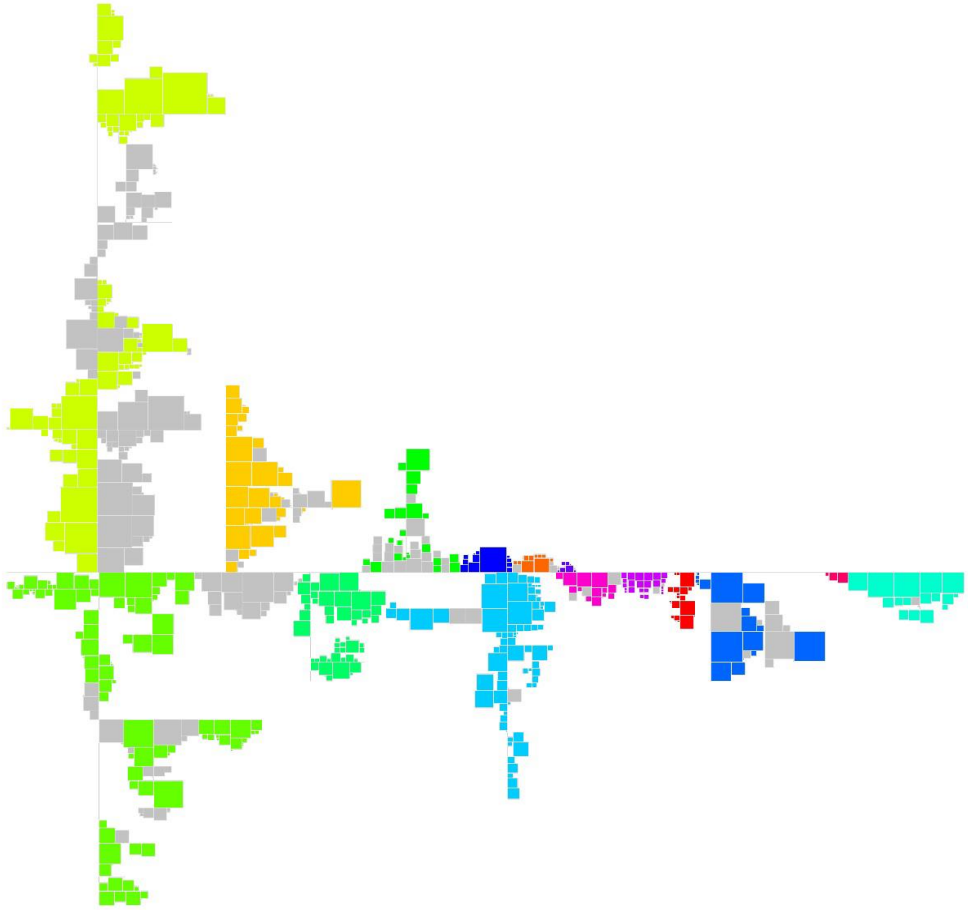


1.7. 1. Улицы города – это пакеты UML (Java)

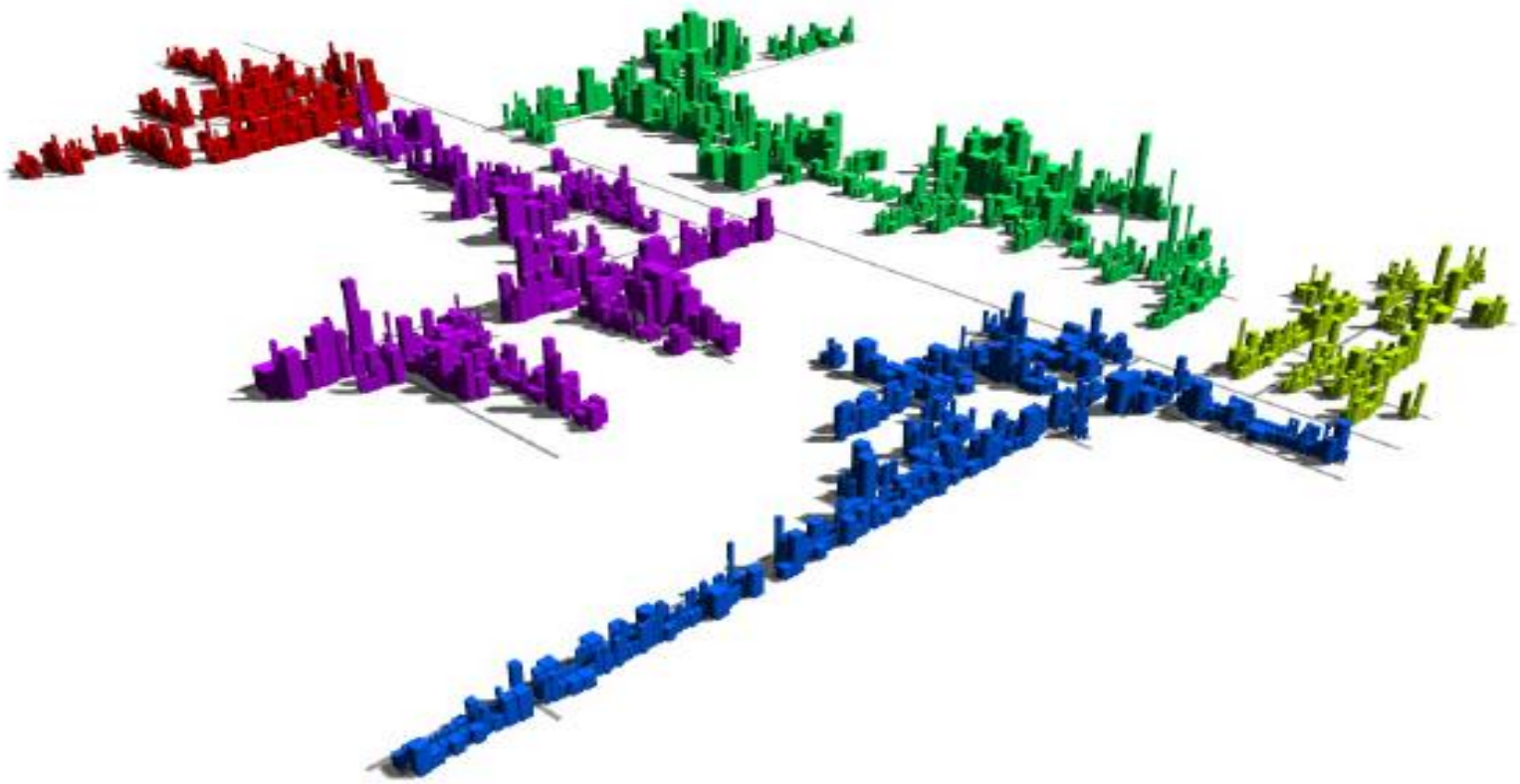
Двумерное изображение улиц города.



Двумерное изображение улиц города. Раскраска свойств зданий - классов.



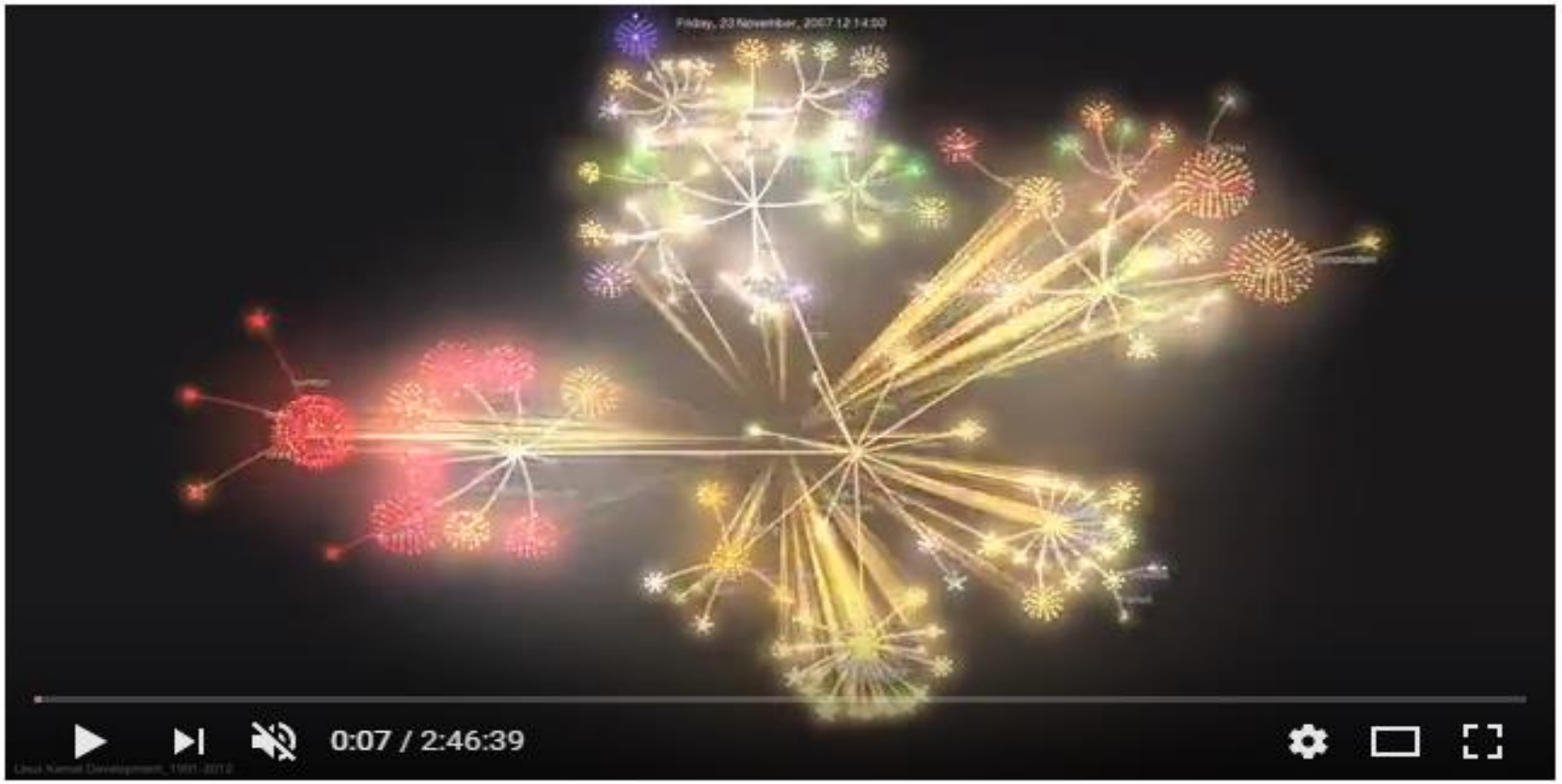
Трехмерное изображение улиц города.



Трехмерное изображение улиц города. Изображение эволюции как рельефа города

Визуализация эволюции программы

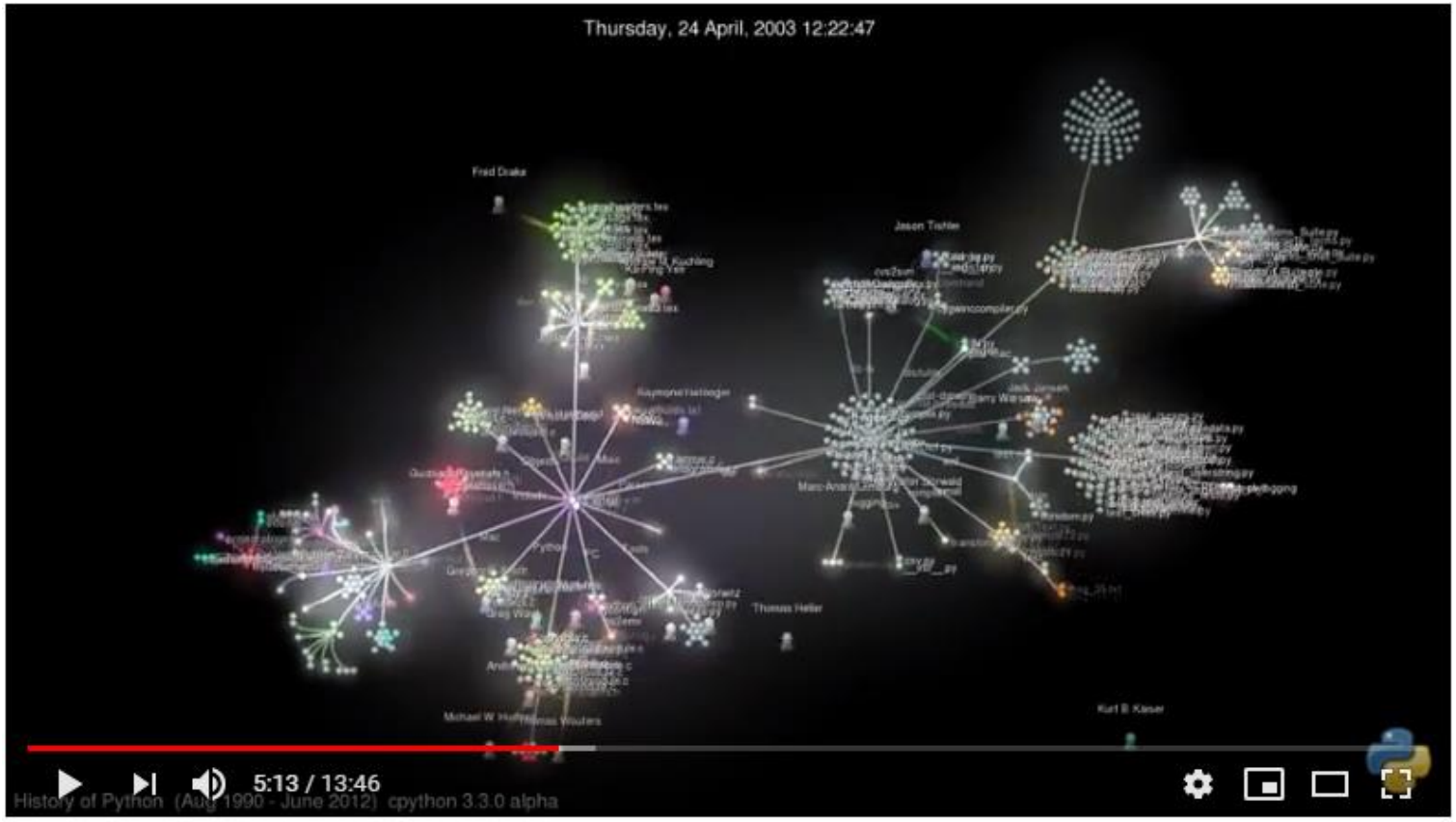
Эволюция программа в Git-репозитории (история ядра Linux)



Linux Kernel Development, 1991-2012

https://www.youtube.com/watch?v=pOSqctHH9vY

Эволюция программы в Git-репозитории (история языка Python)

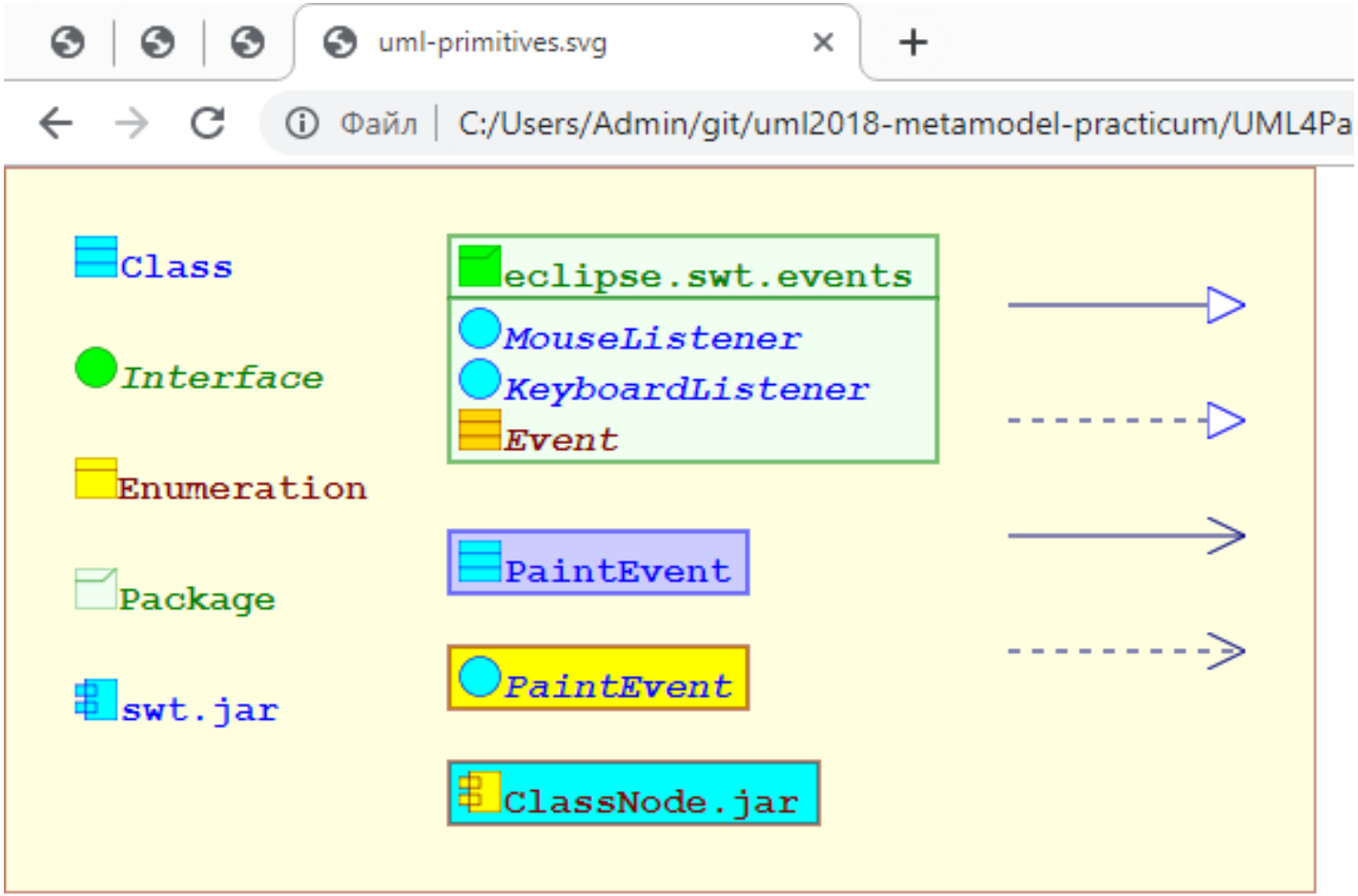


History of Python - Gource - development visualization (august 1990 - june 2012)

https://www.youtube.com/watch?v=cNBtDstOTmA

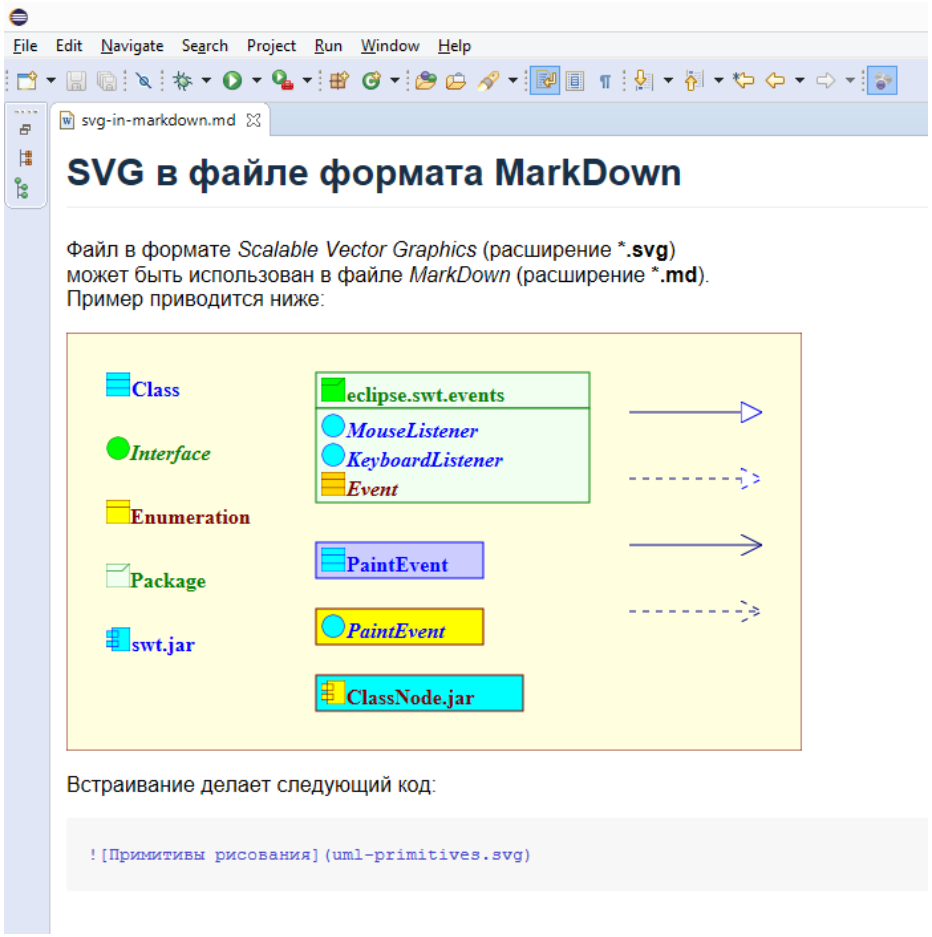
2. Генерация диаграмм в формате HTML5+SVG

Диаграмма в файле формата Scalable Vector Graphics (SVG)



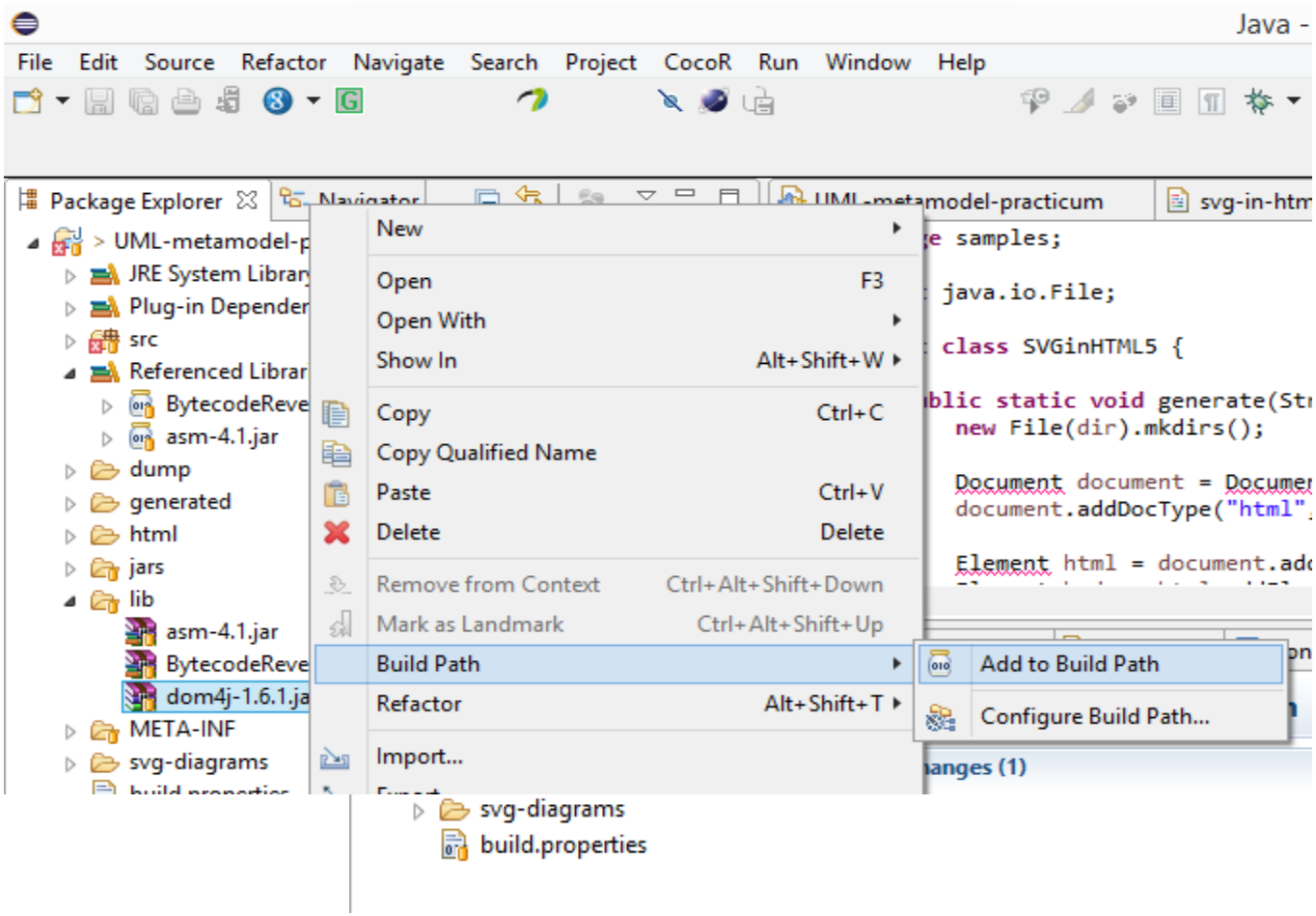
Файл в формате HTML5+SVG
Scalable Vector Graphics встроен в HTML5

Файл в формате SVG (Scalable Vector Graphics) встроен в файл формата MarkDown

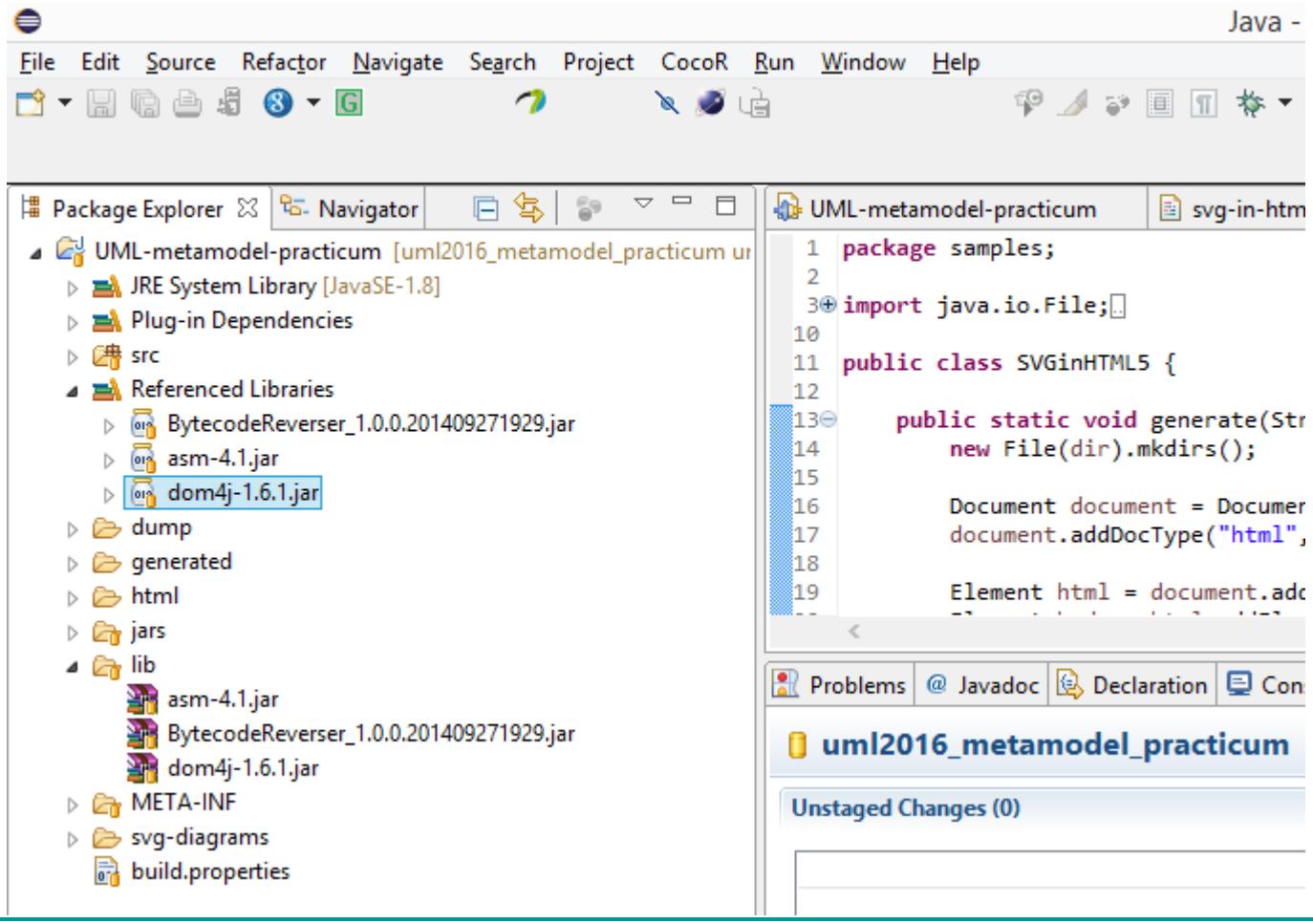


2.1. Использование пакета Dom4j

Библиотека Dom4J
Подключение библиотеки к проекту (1).



Библиотека Dom4J
Подключение библиотеки к проекту (2).



Файл в формате HTML5+SVG
Scalable Vector Graphics


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>

<html>
  <body>
    <h1>SVG in HTML5</h1>

    <svg width="100" height="100">
            <circle cx="50" cy="50" r="40"
                                stroke="green" stroke-width="4" fill="yellow"/>
    </svg>

  </body>
</html>
```

Файл в формате HTML5+SVG
Scalable Vector Graphics

Генерация заголовка файла


```
package samples;

import java.io.File;

import org.dom4j.Document;
import org.dom4j.DocumentFactory;
import org.dom4j.Element;

import uml.diagram.svg.SVG;

public class SVGinHTML5 {

public static void generate(String dir, String fileName) {
   new File(dir).mkdirs();

   Document document = DocumentFactory.getInstance().createDocument();
   document.addDocType("html", null, null);
```

Генерация HTML-тегов


```
Element  html = document.addElement("html");

Element  body = html.addElement("body");

Element  h1 = body.addElement("h1");
h1.addText("SVG in HTML5");
```

Генерация SVG-тегов


```
Element svg = body.addElement("svg");
svg.addAttribute("width", "100");
svg.addAttribute("height", "100");

Element circle = svg.addElement("circle");
circle.addAttribute("cx", "50");
circle.addAttribute("cy", "50");
circle.addAttribute("r", "40");
circle.addAttribute("stroke", "green");
circle.addAttribute("stroke-width", "4");
circle.addAttribute("fill", "yellow");
```

Сохранение документа в файл

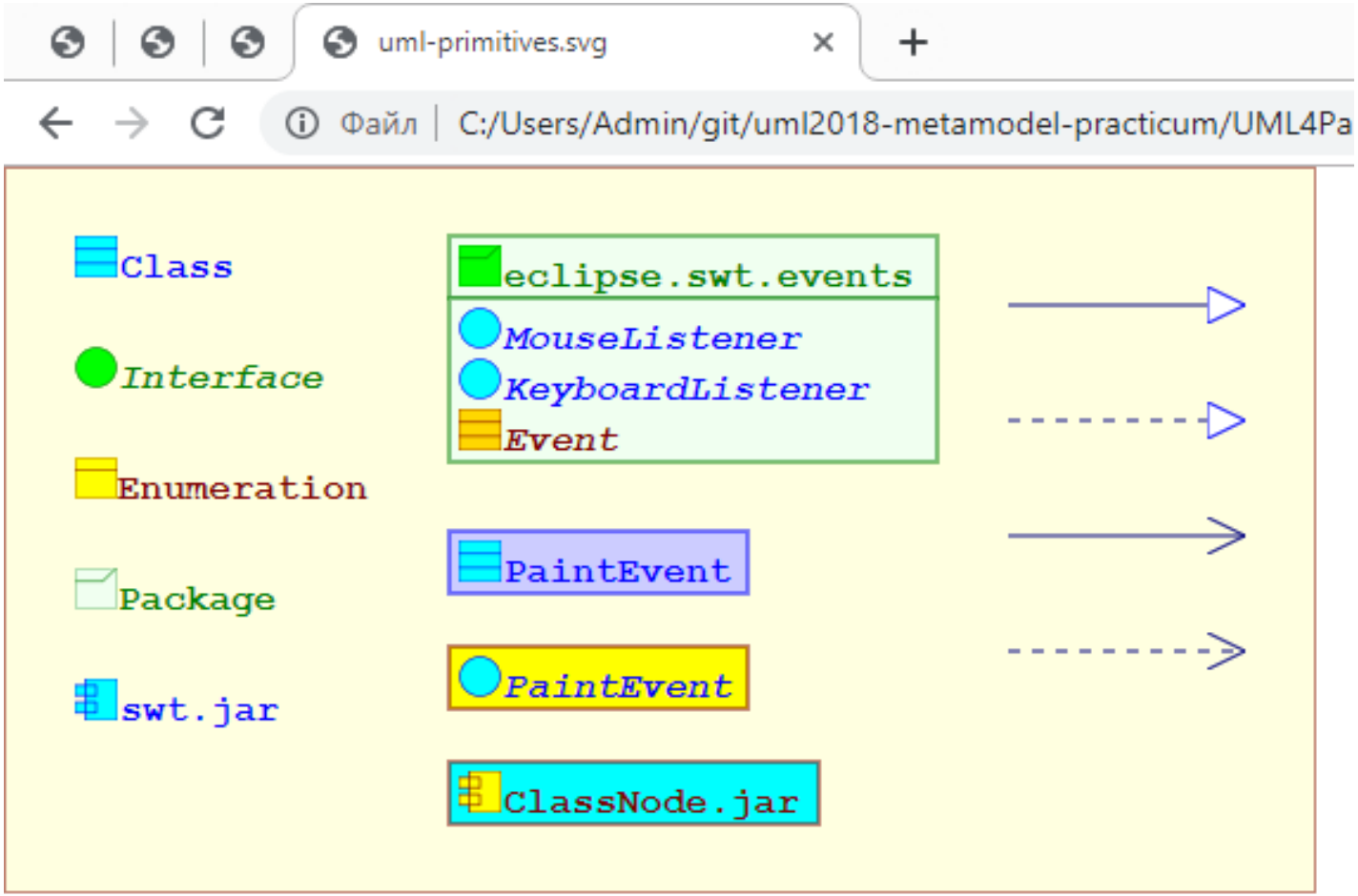

```
        SVG.save(document, dir, fileName);
}
```

Сохранение документа в файл


```
public static void save (Document document, String dir, String fileName) {
        FileOutputStream fos;
        try {
                fos = new FileOutputStream(dir + "/" + fileName);

                OutputFormat format = OutputFormat.createPrettyPrint();

                XMLWriter writer = new XMLWriter (fos, format);

                writer.write(document);
                writer.flush();
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
}
```

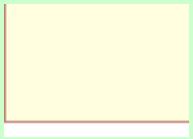
2.2. Примитивы для построения UML-диаграмм

Из чего строится UML-диаграмма (примитивы для рисования)



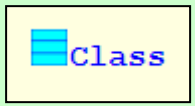
Рисование примитивов для построения диаграмм на языке Java


```
public static void showPrimitives(String diagramDir, String fileName) {
    new File(diagramDir).mkdirs();

    Document document = DocumentFactory.getInstance().createDocument();
    document.addDocType("svg", null,
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd");

    Element diagram = SVG.svgHead(document);

    //
    // Бумага на которой рисуем.
    //
    Element paper = diagram.addElement("rect");
    paper.addAttribute("fill", "LightYellow");
    paper.addAttribute("stroke", "maroon");
```



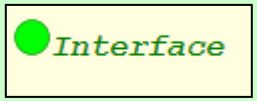
Рисование пиктограммы для класса

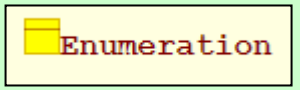

```
//
// Узлы-пиктограммы на диаграммах UML.
//
int x = X;
int y = Y;
int columnWidth = 0;
int diagramWidth = 0;
int diagramHeight = 0;

IconNode icon;

icon = new ClassIconNode(diagram, x, y, "Class");
icon.setX(x);
icon.setY(y);
columnWidth = Math.max(columnWidth, icon.getWidth());
```



Рисование пиктограммы для интерфейса и перечисления

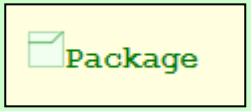

```
y += (icon.getHeight() + DY);
icon = new InterfaceIconNode(diagram, x, y, "Interface");
icon.setColor("green");
icon.setBackgroundColor("lime");
icon.setTextColor("green");
columnWidth = Math.max(columnWidth, icon.getWidth());

y += (icon.getHeight() + DY);
icon = new EnumerationIconNode(diagram, x, y, "Enumeration");
icon.setColor("maroon");
icon.setBackgroundColor("yellow");
icon.setTextColor("maroon");
columnWidth = Math.max(columnWidth, icon.getWidth());
```

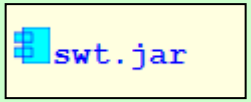




Рисование пиктограммы для пакета и компонента

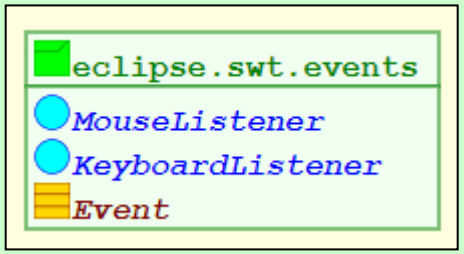

```
y += (icon.getHeight() + DY);
icon = new PackageIconNode(diagram, x, y, "Package");
icon.setColor("green");
icon.setBackgroundColor("#F0FFF0");
icon.setTextColor("green");
columnWidth = Math.max(columnWidth, icon.getWidth());

y += (icon.getHeight() + DY);
icon = new ComponentIconNode(diagram, x, y, "swt.jar");
icon.setX(x);
icon.setY(y);
columnWidth = Math.max(columnWidth, icon.getWidth());
```





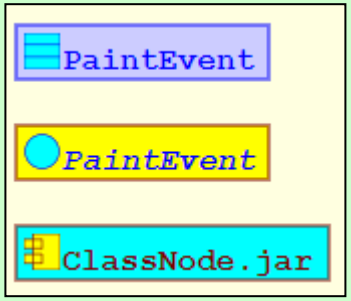
Рисование пиктограммы пакета с вложенными пиктограммами интерфейсов и класса


```
// Составные узлы на диаграммах UML.
//
x += (columnWidth + DX);
y = Y;
columnWidth = 0;

PackageNode pn = new PackageNode(diagram, "eclipse.swt.events", x, y);

pn.createInterfaceIconNode("MouseListener");
pn.createInterfaceIconNode("KeyboardListener");

IconNode in = pn.createClassIconNode("Event");
in.setBackgroundColor("gold");
in.setColor("maroon");
in.setTextColor("maroon");
in.setItalic(true);
columnWidth = Math.max(columnWidth, pn.getWidth());

y += (pn.getHeight() + DY);
```



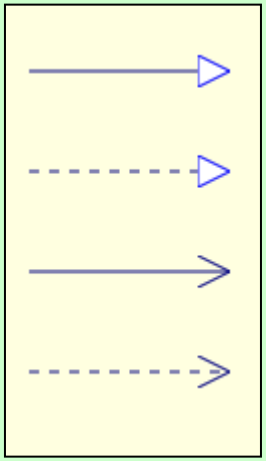
Рисование узлов для класса, интерфейса и компонента


```
ClassifierNode cn = new ClassNode(diagram, "PaintEvent", x, y);
cn.header.setBackground("#CCCCFF");
cn.header.setColor("blue");
columnWidth = Math.max(columnWidth, cn.getWidth());

y += 50;
InterfaceNode inn = new InterfaceNode(diagram, "PaintEvent", x, y);
inn.header.setBackground("yellow");
inn.header.setColor("maroon");
columnWidth = Math.max(columnWidth, inn.getWidth());

y += 50;
ComponentNode cmn = new ComponentNode(diagram, "ClassNode.jar", x, y);
cmn.iconNode.setColor("maroon");
cmn.iconNode.setTextColor("maroon");
cmn.iconNode.setBackgroundColor("yellow");
cmn.header.setColor("maroon");
columnWidth = Math.max(columnWidth, cmn.getWidth());

diagramHeight = Math.max(diagramHeight, y + cmn.getHeight());
```



Рисование отношений наследования, реализации, ассоциации и зависимости


```
//
// Отношения на диаграммах UML (ребра графа).
//
int L = 100;
columnWidth = L;

double[ ] xy; // Траектория линии.

xy = new double[ ] { x, y, x+L, y };
new InheritanceEdge (diagram, xy );

y += 50;
xy = new double[ ] { x, y, x+L, y };
new ImplementationEdge (diagram, xy );

y += 50;
xy = new double[ ] { x, y, x+L, y };
new AssociationEdge (diagram, xy );

y += 50;
xy = new double[ ] { x, y, x+L, y };
new DependencyEdge (diagram, xy );
```



Примеры диаграмм

Матрица структурной зависимости пакета Design Structure Matrix (DSM) для JHotDraw

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2025

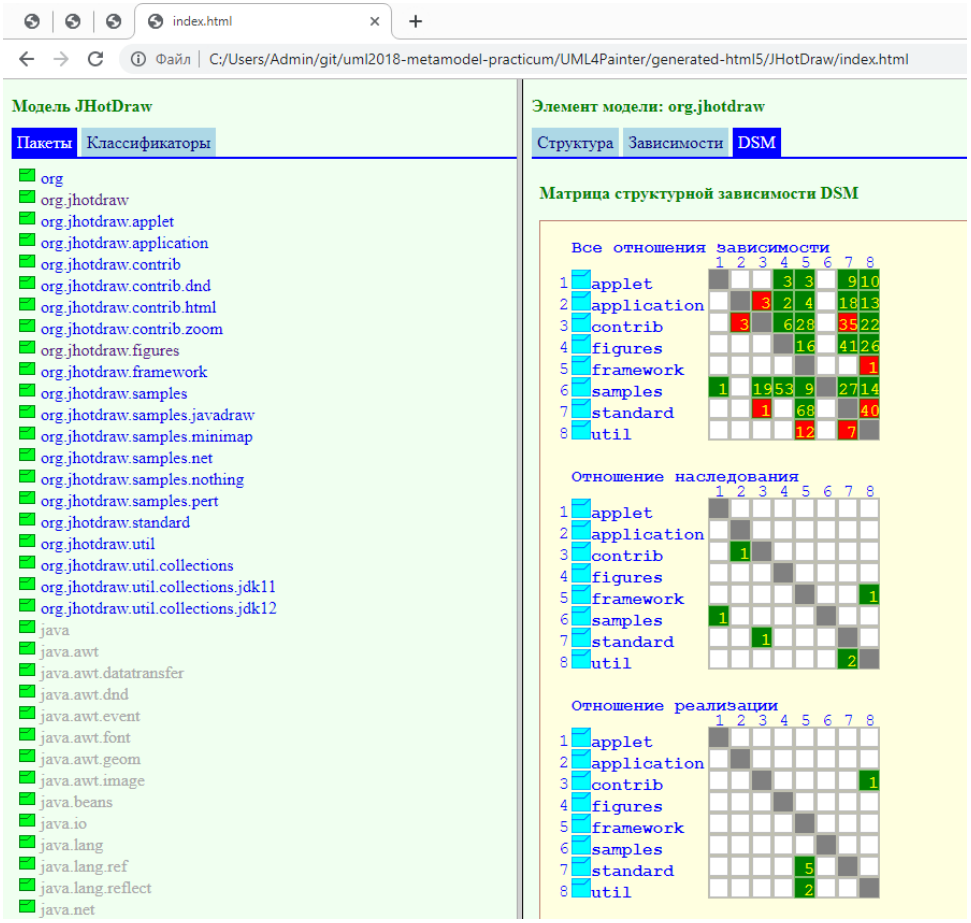
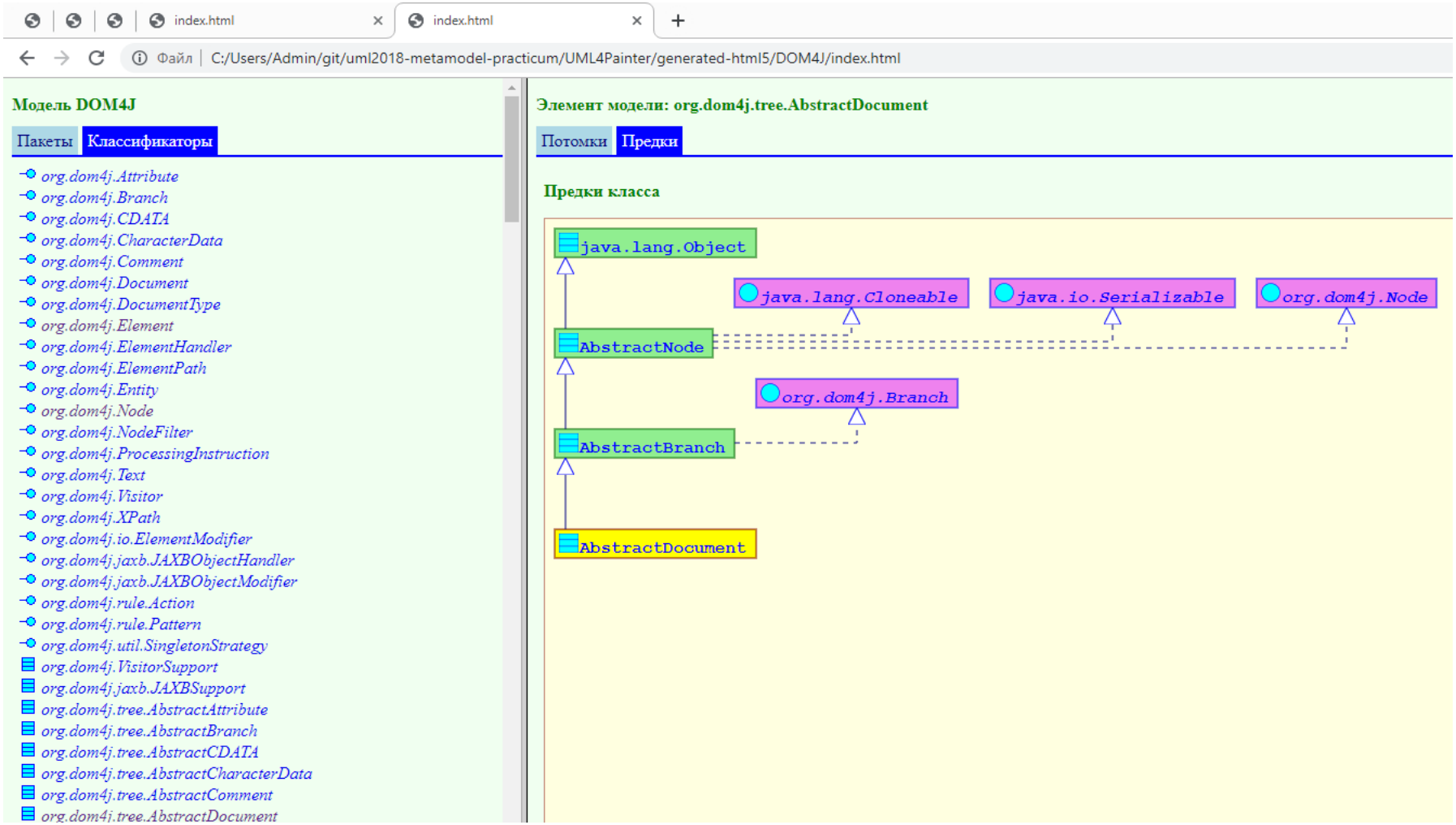


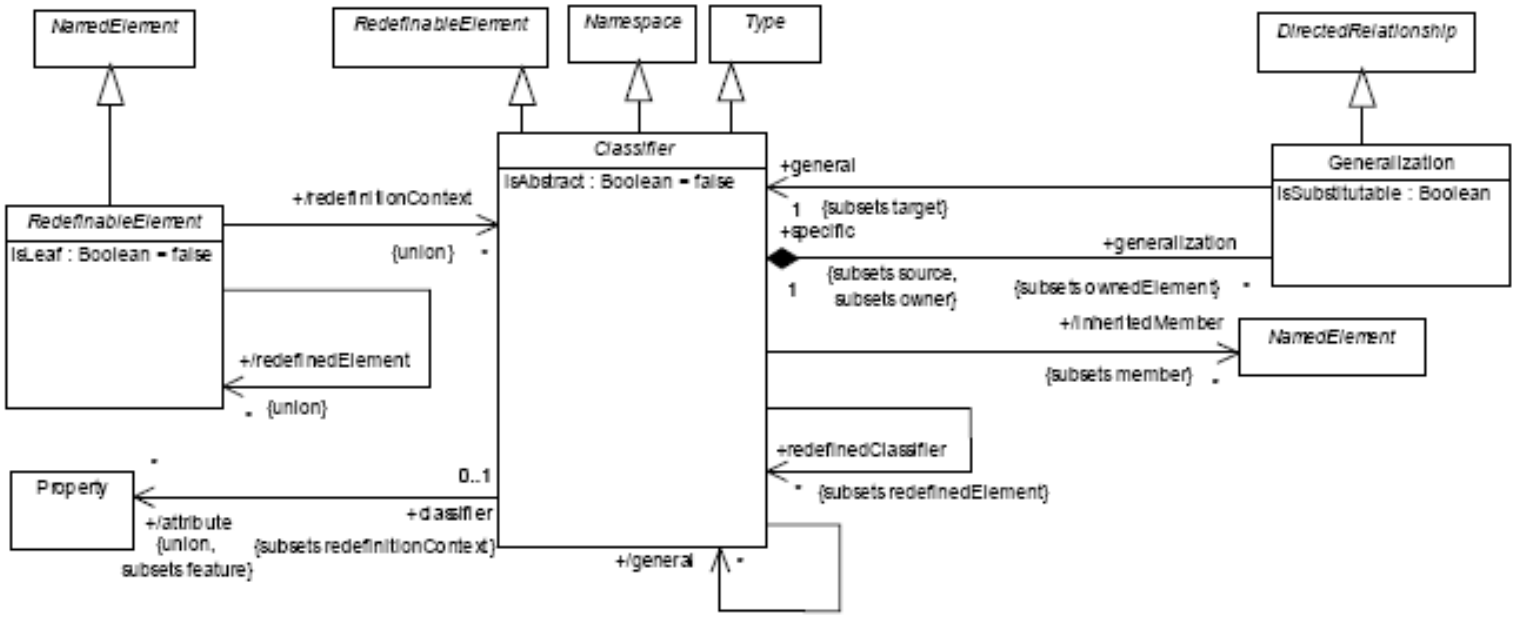
Диаграмма предков класса org.dom4j.tree.AbstractDocument

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2025

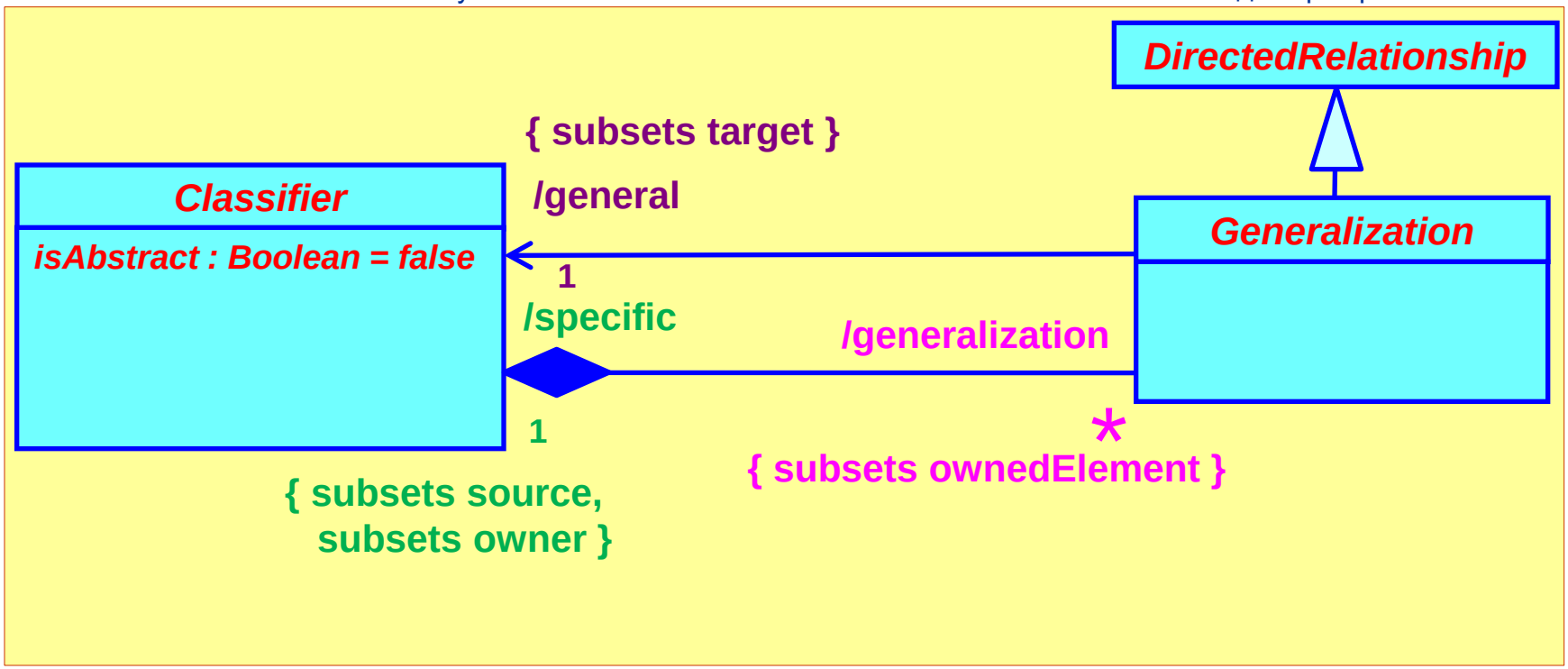


Вспомогательные UML-модели для визуализации

Классификаторы в метамодели языка UML



Моделирование отношения наследования (обобщения) метамодели языка UML



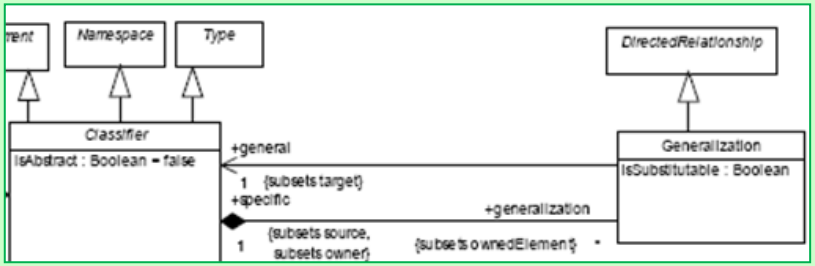

```
class Node extends View {

}
```

Поиск и сохранение потомков класса


```
public class UMLRelations {

    @SuppressWarnings("serial")
    static public class UMLClassifiers extends HashSet<Classifier> {
    }

    @SuppressWarnings("serial")
    static public class UMLClasses extends HashSet<Class> {
    }

    public static Map<Interface, UMLClasses> implementations = new HashMap<>();

    public static Map<Classifier, UMLClassifiers> generalizations = new HashMap<>();

    public static void load(Model model) {
        loadGeneralizations(model);
        loadImplementations(model);
    }
```

Поиск и сохранение потомков класса


```
private static void loadGeneralizations (Model model) {
        Predicate<Element> vizitFilter = e -> true;
        Predicate<Element> actionFilter = e -> e instanceof Generalization;

        Consumer<? super Element> action = e -> {
                Generalization g = (Generalization) e;

                Classifier parent = g.getGeneral();
                Classifier child  = g.getSpecific();     }

                UMLClassifiers childs = g.get(parent);
                if (childs == null) {
                        childs = new UMLClassifiers();
                        g.put(parent, childs);
                }

                childs.add(child);
        };

        UMLUtil.walkAll(model, vizitFilter, actionFilter, action);
}
```



Поиск и сохранение потомков класса


```
private static void dumpInheritance (String dumpPath) {
        new File(dumpPath).mkdirs();

        PrintStream ps = null;
        try {
                ps = new PrintStream(dumpPath + "/_inheritance.txt");
        } catch (FileNotFoundException e) {
                return;
        }
```

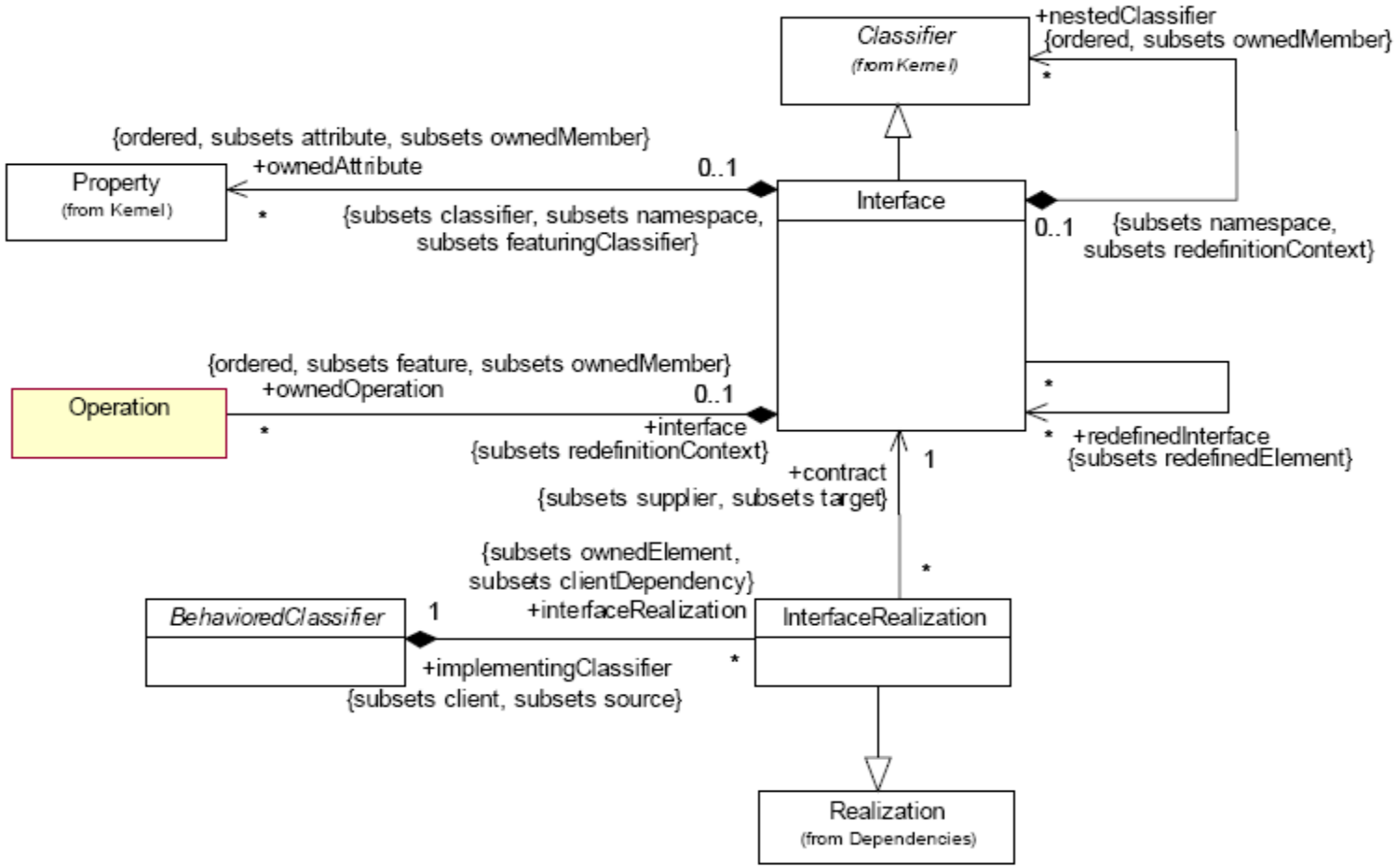
Поиск и сохранение потомков класса


```
// …
        for (Entry<Classifier, UMLClassifiers> g : generalizations.entrySet()) {
                Classifier parent = g.getKey();

                String parentName = UMLUtil.getShortName(parent).replace("::", ".");
                String parentKind = "";

                if (parent   instanceof   Class)
                   parentKind = "class";
                else
                if (parent    instanceof   Interface)
                   parentKind = "interface";

                String  childsNames  =  g.getValue()
                        .stream()
                        .map(child -> UMLUtil.getShortName(child))
                        .map(name -> name.replace("::", "."))
                        .sorted()
                        .collect(  Collectors.joining(",\n   ", "\n   ", "\n")  );

                ps.format("%s %s%n  isParentFor%s %n", parentKind, parentName, childsNames);
        }

        ps.close();
}
```

Поиск и сохранение потомков класса

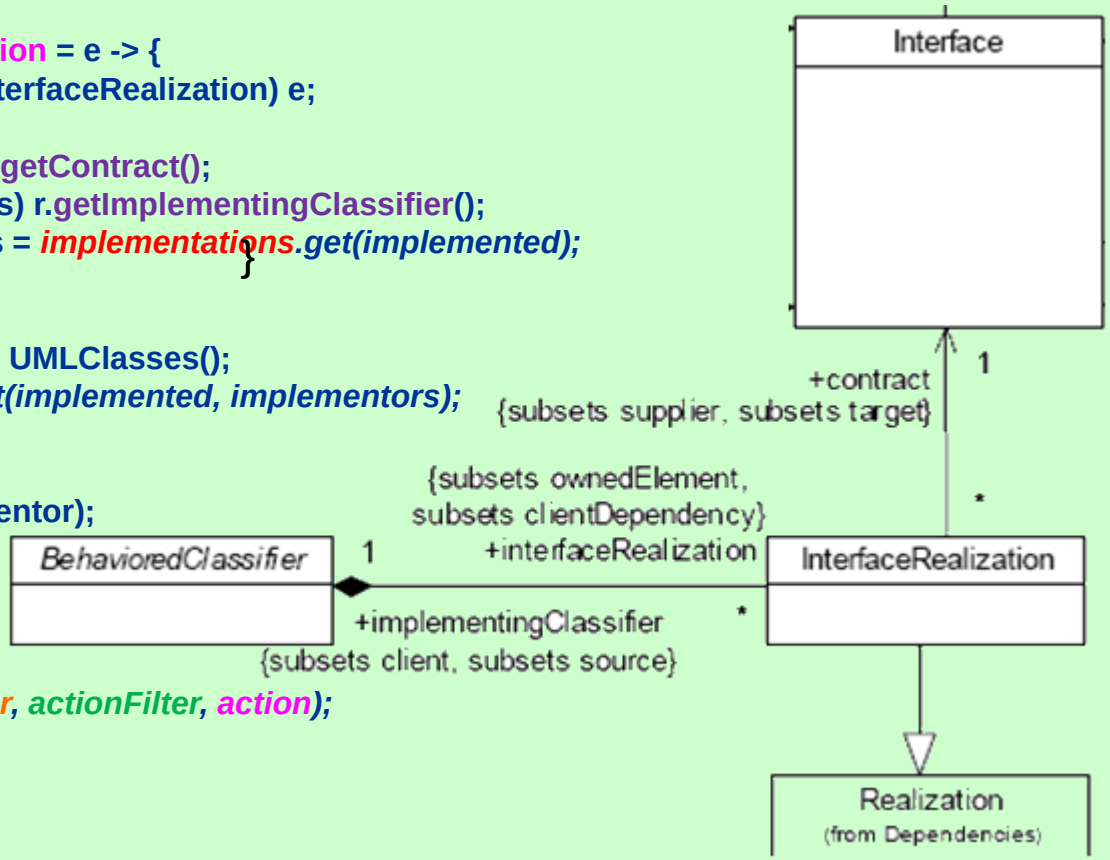

```
class javax.swing.JButton
  isParentFor
   org.jhotdraw.util.CommandButton,
   org.jhotdraw.util.PaletteButton

interface org.jhotdraw.contrib.html.AttributeContentProducerContext
  isParentFor
   org.jhotdraw.contrib.html.HTMLContentProducerContext

class org.dom4j.tree.AbstractBranch
  isParentFor
   org.dom4j.tree.AbstractDocument,
   org.dom4j.tree.AbstractElement

interface org.jhotdraw.framework.Figure
  isParentFor
   org.jhotdraw.contrib.Layoutable,
   org.jhotdraw.contrib.html.GeometricFigure,
   org.jhotdraw.framework.ConnectionFigure
```

Представление интерфейсов в метамодели языка UML



Поиск и сохранение реализаций интерфейса


```
private static void loadImplementations (Model model) {
        Predicate<Element> vizitFilter = e -> true;
        Predicate<Element> actionFilter = e -> e instanceof InterfaceRealization;

        Consumer<? super Element> action = e -> {
                InterfaceRealization r = (InterfaceRealization) e;

                Interface implemented  = r.getContract();
                Class implementor = (Class) r.getImplementingClassifier();
                UMLClasses implementors = implementations.get(implemented);

                if (implementors == null) {
                        implementors = new UMLClasses();
                        implementations.put(implemented, implementors);
                }

                implementors.add(implementor);

        };

        UMLUtil.walkAll( model, vizitFilter, actionFilter, action);
}
```



Поиск и сохранение реализаций интерфейса


```
private static void dumpImplementations (String dumpPath) {
        new File(dumpPath).mkdirs();

        PrintStream ps = null;
        try {
                ps = new PrintStream(dumpPath + "/_implementations.txt");
        } catch (FileNotFoundException e) {
                return;
        }
                                                        }
```

Поиск и сохранение реализаций интерфейса


```
        for (Entry<Interface, UMLClasses> g : implementations.entrySet()) {
                Interface implemented = g.getKey();

                String implementedName = UMLUtil.getShortName(implemented).replace("::", ".");

                String implementorsNames = g.getValue()
                        .stream()
                        .map(child -> UMLUtil.getShortName(child))
                        .map(name -> name.replace("::", "."))
                        .sorted()
                        .collect( Collectors.joining(",\n   ", "\n   ", "\n"));

                ps.format("interface %s%n  isImplementedBy classes%s %n",
                         implementedName, implementorsNames);
        }

        ps.close();
}
```

Поиск и сохранение потомков класса

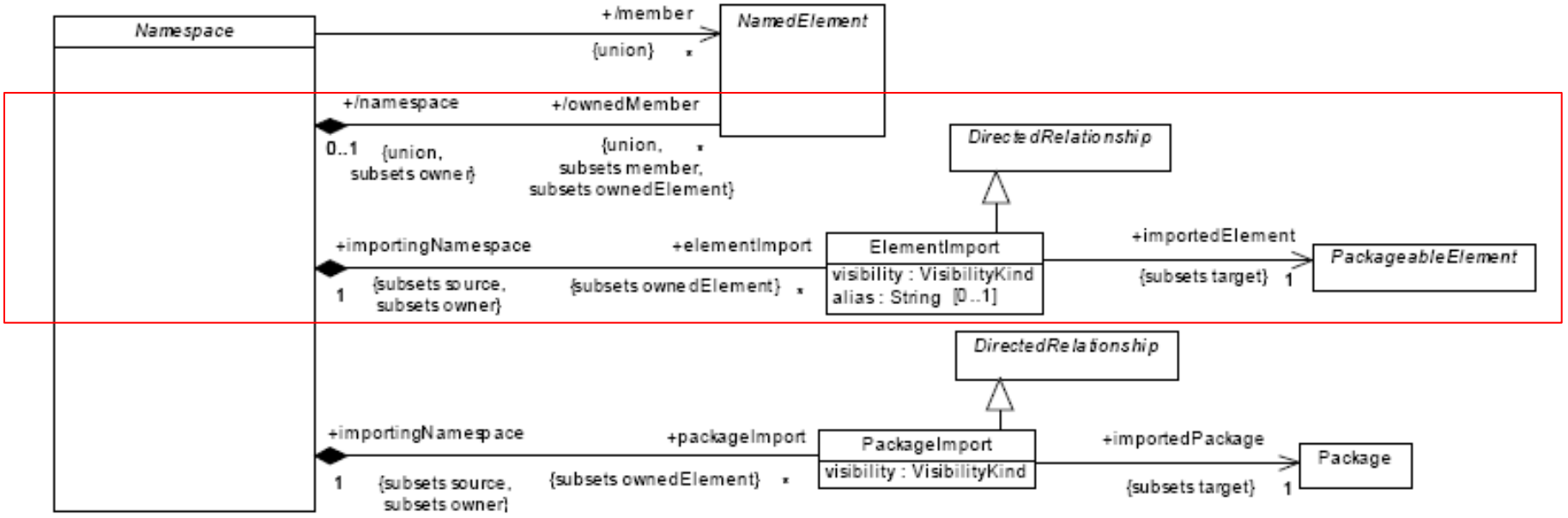
interface org.jhotdraw.framework.DrawingEditor
  isImplementedBy classes
   org.jhotdraw.applet.DrawApplet,
   org.jhotdraw.application.DrawApplication,
   org.jhotdraw.samples.javadraw.JavaDrawViewer

interface java.awt.image.ImageObserver
  isImplementedBy classes
   org.jhotdraw.figures.ImageFigure

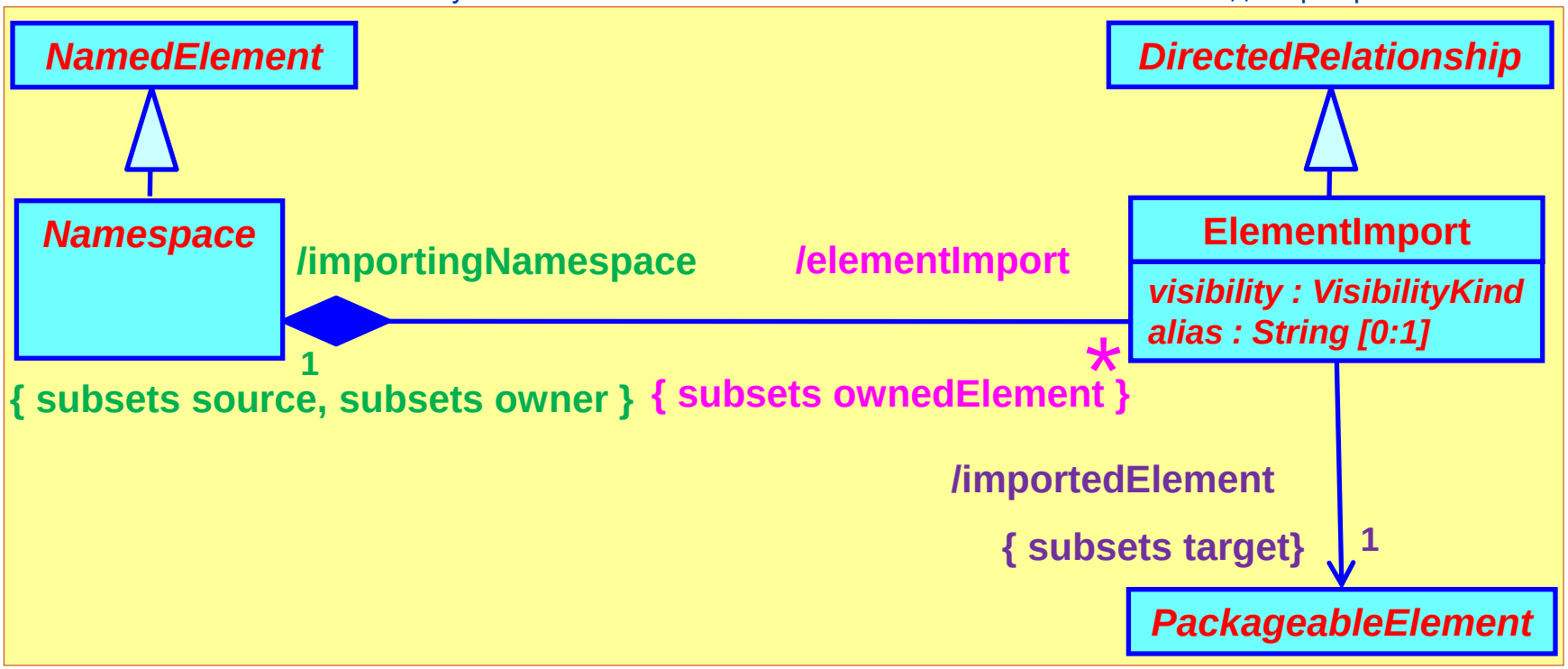
interface org.jhotdraw.contrib.html.HTMLContentProducerContext
  isImplementedBy classes
   org.jhotdraw.contrib.html.HTMLTextAreaFigure

Импорт в пространства имен в метамодели языка UML

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2025



Моделирование импорта элементов в метамодели языка UML



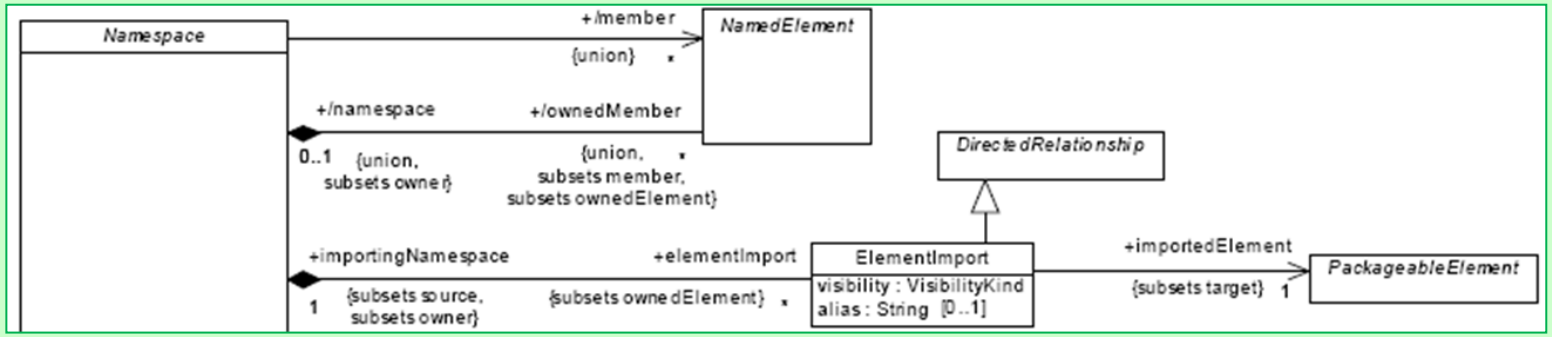

```
import graphics.Color;

class Node {
  Color borderColor;
}
```

Матрица зависимости между пакетами


```
public class DSM {
        /**
         * Размер квадратной матрицы.
         */
        public int size;

        /**
         * Пакеты для который строится матрица.
         */                                                    }
        private List<Package> packages;

        /**
         * Матрица ячейки которой содержат множество импортов
         * из пакета в колонке матрицы в пакет в строке матрицы.
         */
        private DSMCell [ ][ ] matrix;
```

Матрица зависимости между пакетами. Ячейка матрицы


```
@SuppressWarnings("serial")
public class DSMCell extends HashSet<ElementImport> {
        public List<PackageableElement> getImported() {
                return stream()
                    .map(im -> im.getImportedElement())
                    .distinct() // Убрали повторяющиеся элементы.
                    .collect( Collectors.toList());
        }
        public long getAmount() {
                return stream()
                    .map(im -> im.getImportedElement())
                    .distinct() // Убрали повторяющиеся элементы.
                    .count();
        }
}
```

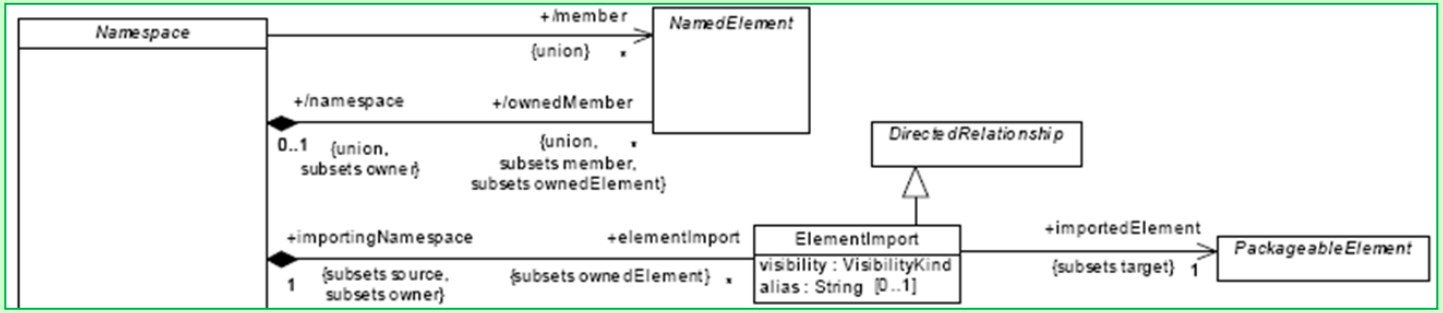


Построение матрицы зависимости между пакетами


```
private void build() {
        if (size == 0)
           return;

        Predicate<Element> vizitFilter     = e -> true;
        Predicate<Element> actionFilter =  e -> e  instanceof   ElementImport;

        Consumer<? super Element> action = e -> {



            // …



        };

        Model model = packages.get(0).getModel();

        UMLUtil.walkAll( model, vizitFilter, actionFilter, action);
}
```

Построение матрицы зависимости между пакетами


```
Consumer<? super  Element> action = e -> {
        // Действие для найденного в модели импорта.
        ElementImport elementImport = (ElementImport) e;

        // Кто импортирует и что импортируется.
        final Namespace importer = elementImport.getImportingNamespace();
        final PackageableElement imported = elementImport.getImportedElement();

        // Пакеты для элементов которых выполняется импорт.
        final Package importerPackage = importer.getNearestPackage();
        final Package exporterPackage = imported.getNearestPackage();

        // Пакеты для которых строится матрица.
        Package importerOwner = findOwner(importerPackage);
        Package exporterOwner = findOwner(exporterPackage);

        if ((importerOwner == null) || (exporterOwner == null))
           return;

        DSMCell imports = getCell(importerOwner, exporterOwner);
        imports.add(elementImport);
};
```



Матрица зависимости между пакетами для библиотеки Dom4j