

Разработка инструментов моделирования
объектно-ориентированных программных систем
основанных на языке UML и
интегрированных среду Eclipse.

Владимир Юрьевич Романов,
Московский Государственный Университет им. М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
vromanov@cmc.msu.ru,
vladimir.romanov@gmail.com

Раздел 3.

Graphical Editing Framework

базовый инструмент построения
графического интерфейса
для систем моделирования

Graphical Editing Framework

GEF

- Интерактивный уровень
- Отображение Модель-Вид
- Интеграция с верстаком

Draw2d

- Перерисовка
- Планировка
- Печать

SWT
canvas

Уровень взаимодействия
с аппаратурой

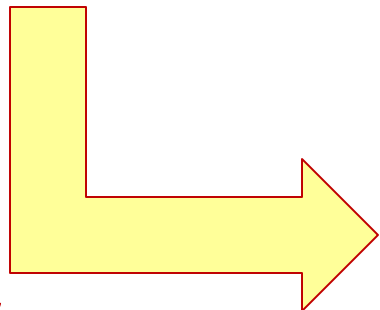
Пакет Draw2d

Введение в Draw2d

- Представляется отдельным подключаемым модулем
- Легковесная графическая система (работает на одном потоке управления)
- Используется GEF для рисования графических примитивов
- Обрабатывает события от мыши
- Изображения строятся из фигур – аналогов окон
- Фигуры располагаются на различных уровнях изображения

Draw2d. Фигуры

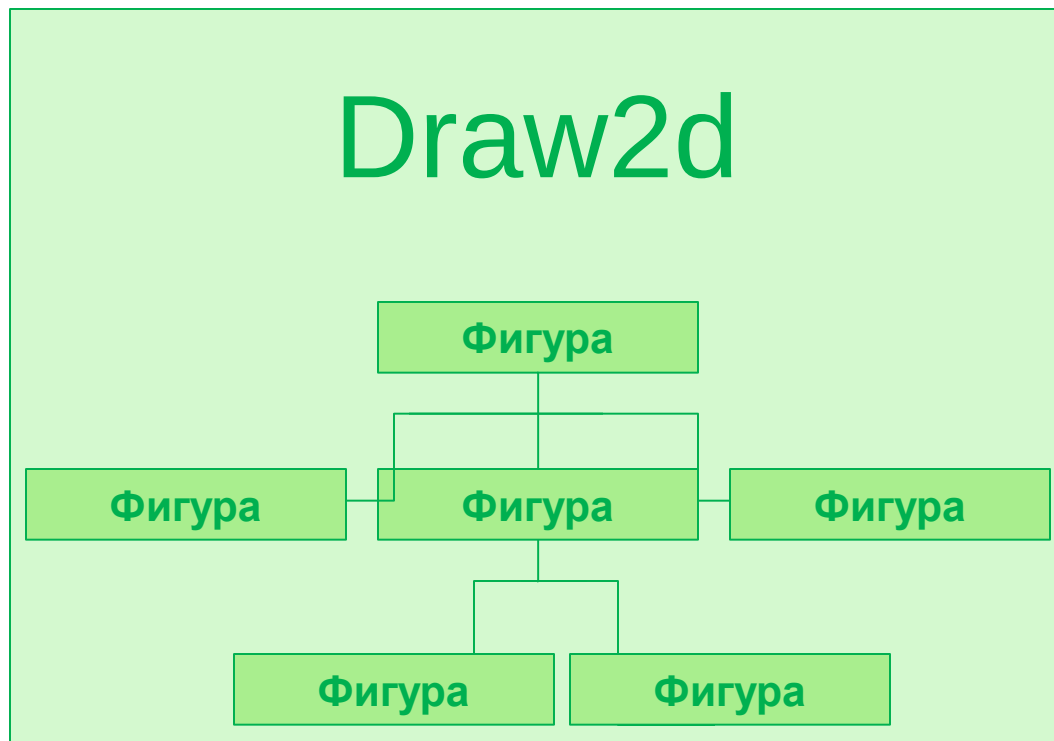
SWT
Canvas



События:

Paint
Mouse
Key
Focus

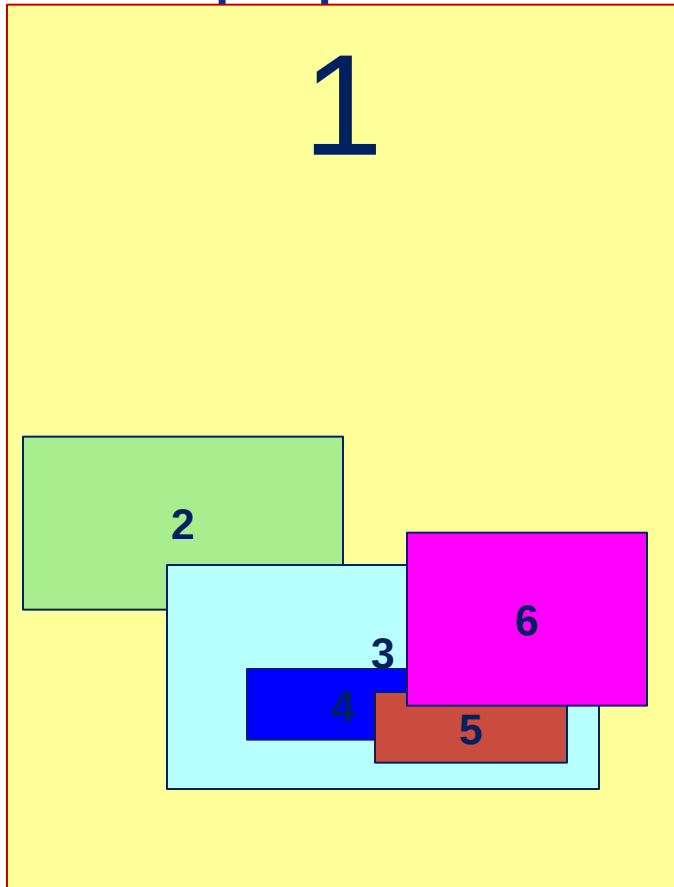
Draw2d



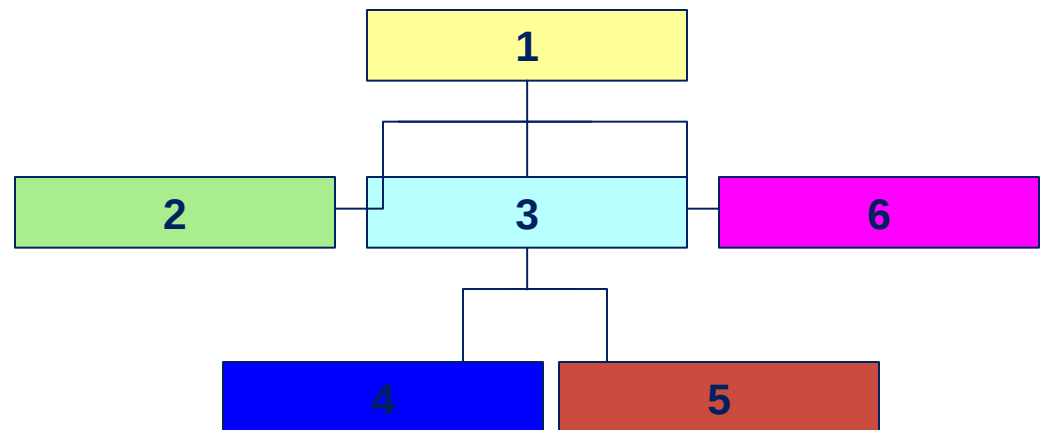
Draw2d – легковесная система

- Графическая система работает в рамках одного легковесного потока управления
- Графические объекты (фигуры) трактуются как окна не прямоугольной формы
- **В отличие от «прямоугольных» графических систем фигуры позволяют создавать сложные изображения не потребляя много ресурсов системы**
- Фигуры могут быть вложены друг в друга
- Фигуры могут принимать фокус ввода
- Фигуры могут принимать события от мыши
- Фигуры имеют собственную координатную систему
- Фигуры имеют собственный курсор

Порядок рисования и поиска фигур в иерархии



- Фигуры образуют дерево
- Родители отсекают детей по границам
- Последний нарисованный наверху
- Поиск нажатой мышью фигуры в обратном порядке



Функциональные возможности фигур (1)

- Регистрация и deregистрация «слушателей» фигур
- Уведомление слушателей фигур о нажатии мыши над фигурой
- Уведомление слушателей о структурных изменениях в иерархии фигур, в перемещении и изменении размера фигур
- Манипуляция иерархией фигур
 - *Добавление и удаление фигур-детей*
 - *Операции доступа к фигурам-родителям и фигурам-детям*
- Задание планировщика размещения фигур
- Задание положения и размера фигуры

Функциональные возможности фигур (2)

- Задание курсора мыши, показываемого при проходе мыши над фигурой
- Задание подсказки, показываемой при проходе мыши над фигурой
- Задание фокуса ввода и считывание его текущего положения
- Описание прозрачности и видимости фигуры
- Выполнение преобразования координат
- Рисование фигуры

Подкласс фигур Shape

Shape (Шейп)

- ❑ *Могут сами себя заполнить и нарисовать границы с конфигурируемыми толщиной и типом линии границы*
- ❑ *Возможно рисование в режиме исключающего ИЛИ (XOR)*
- ❑ *Примеры подклассов класса Shape:*
 - Ellipse
 - Triangle
 - Rectangle
 - Rounded Rectangle
 - Polyline
 - Polygon

Подкласс фигур Widget

Widget (Управляющий элемент)

- ❑ *Фигуры, позволяющие выполнять ввод информации в приложения использующие подключаемый модуль Draw2D*
- ❑ *Примеры управляющих элементов*
 - Button (Кнопка)
 - Check Box (Переключатель)
 - Label (Метка)

Подклассы фигур Layer и Pane

Layer (Уровень) и Pane (Панель)

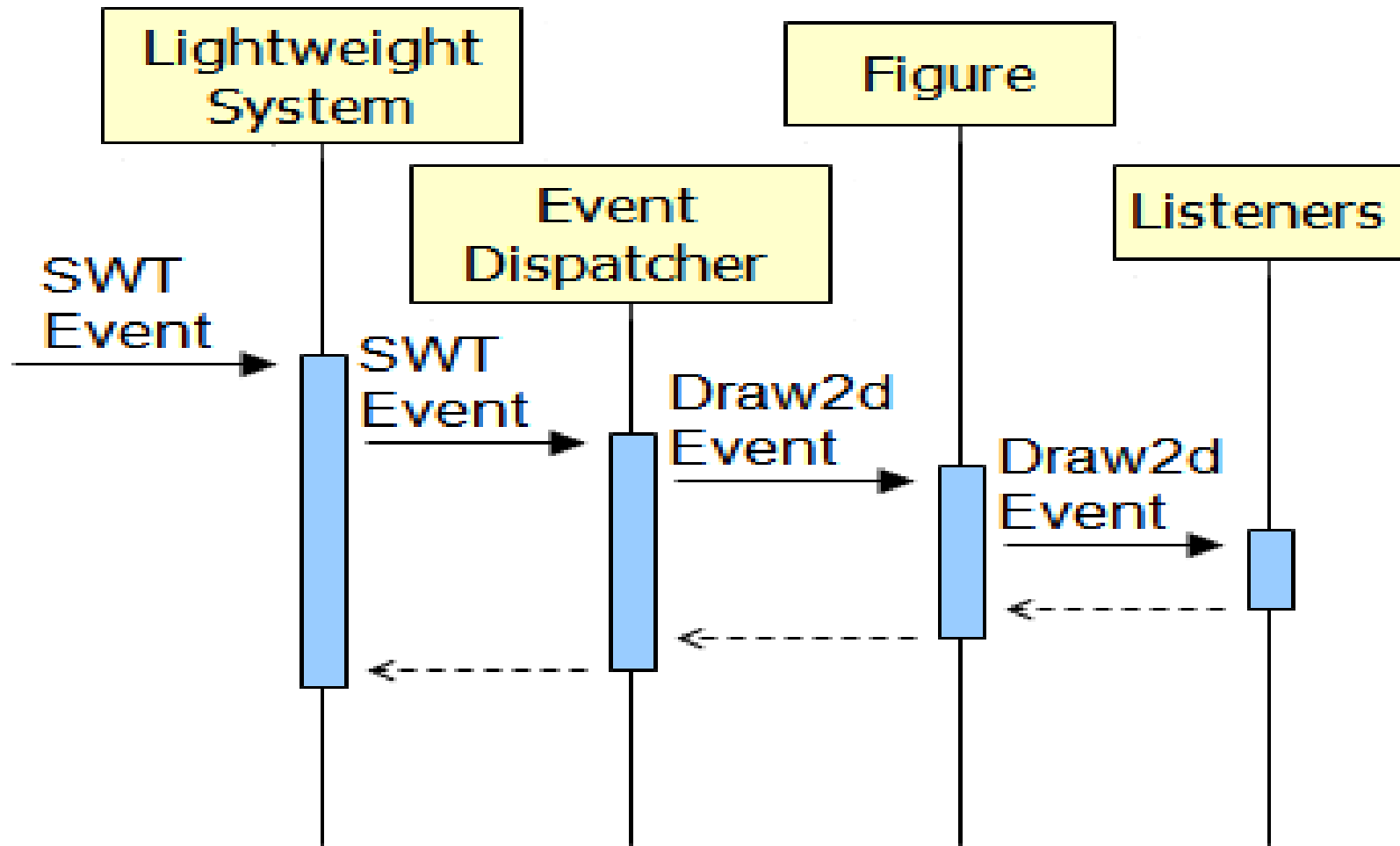
- ❑ *Эти фигуры предназначены для хранения фигур-детей.*
- ❑ *Позволяют выполнять масштабирование фигур*
- ❑ *Позволяют выполнять скруллинг (пролистывание) фигур*
- ❑ *Позволяют размещать фигуры на разных уровнях*

Класс `LightweightSystem`

■ **Выполняет отображение панелей (*canvas*) SWT в модуль *Draw2d*. Содержит 3 компоненты:**

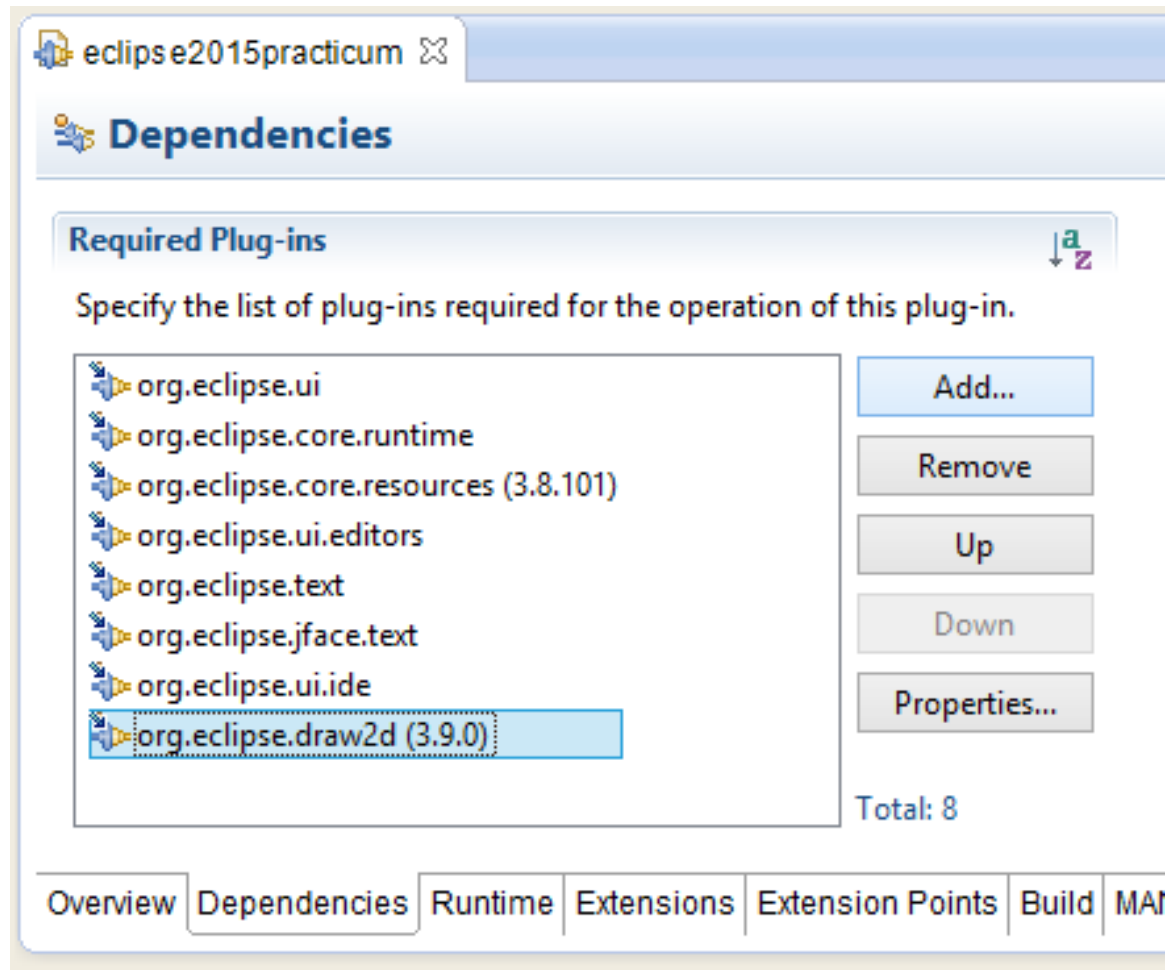
- Корневая фигура класса `LightweightSystem`. Является «родительской» фигурой для корневой фигуры приложения. Наследует графическое окружение SWT: шрифт, основной и фоновый цвета
- Диспетчер событий: класс `SWTEventDispatcher` транслирует события SWT в соответствующие события *Draw2D*. Отслеживает какие события в фокусе, в какую фигуру направлены события мыши, для какой фигуры запрошена подсказка
- Менеджер обновления. Отвечает за перерисовку и обновление фигур. Метод `performUpdate()` вызывается когда приходит запрос на перерисовку *canvas* на которых рисуются фигуры. Менеджер хранит список фигур которые изменены или должны быть перерисованы. Умалчиваемый менеджер `DeferredUpdateManager` позволяет выполнять обновления асинхронно, запрашивая выполнение этой работы через поток управления интерфейса пользователя.

Обработка событий



Подключение плагина *draw2d*

(подключение jar-файла к проекту Eclipse)



Использование draw2d в оконном приложении.

Создание приложения в функции *main()*

```
package samples.draw2d;

import org.eclipse.draw2d.ColorConstants;
import org.eclipse.draw2d.Figure;
import org.eclipse.draw2d.LightweightSystem;
import org.eclipse.draw2d.LineBorder;
import org.eclipse.draw2d.XYLayout;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

import uml.images.UMLImages;

public class Draw2DWindows {
    public static void main(String args[ ]) {
```

Создание главного окна приложения

```
// ...
```

```
Display d = new Display();
```

```
// Загрузка изображений из файлов в память.  
UMLImages.load();
```

```
// Создаем на мониторе окно с пиктограммой.  
final Shell shell = new Shell(d);  
shell.setSize(600, 400);  
shell.setText("Draw2D Samples");  
shell.setImage(UMLImages.diagramImage);
```

```
LightweightSystem lws = new LightweightSystem(shell);
```

```
// ...
```

Вставка в окно простейшей фигуры

Корневая фигура

```
// Создаем корневую фигуру.  
Figure contents = new Figure();  
contents.setOpaque(true); // Фигура не прозрачна.  
contents.setBackgroundColor( ColorConstants.yellow );  
contents.setBorder( new LineBorder(new Color(d, 192, 0, 192)) );
```

```
// Задаем планировщик для всех фигур вложенных  
// в корневую фигуру.  
// При вставке вложенной фигуры должны указываться  
// абсолютные координаты X и Y этой фигуры.
```

```
XYLayout contentsLayout = new XYLayout();  
contents.setLayoutManager(contentsLayout);
```

```
// Вставка вложенных фигур делается здесь.
```

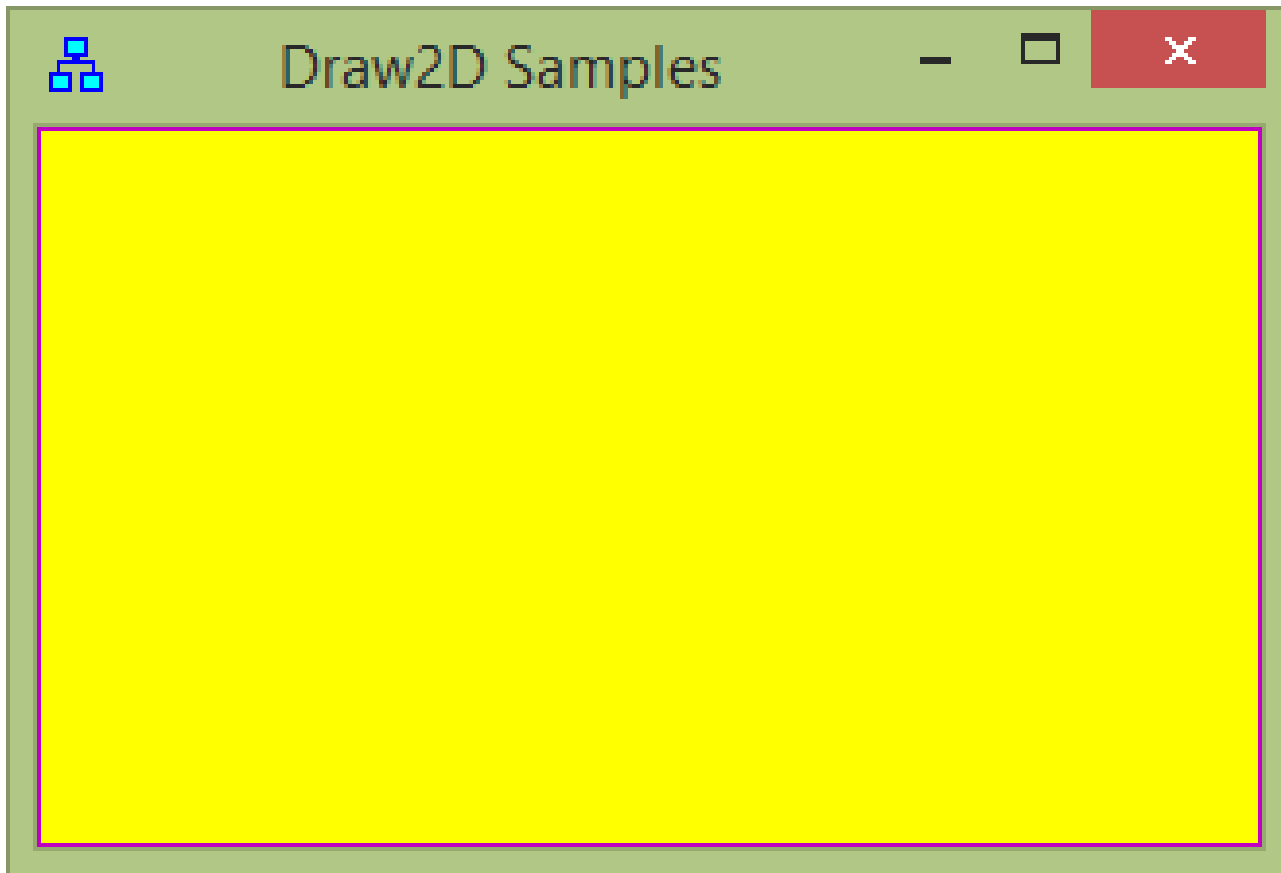
```
lws.setContents(contents);
```

Фигуры – дети могут
иметь произвольные
абсолютные
координаты

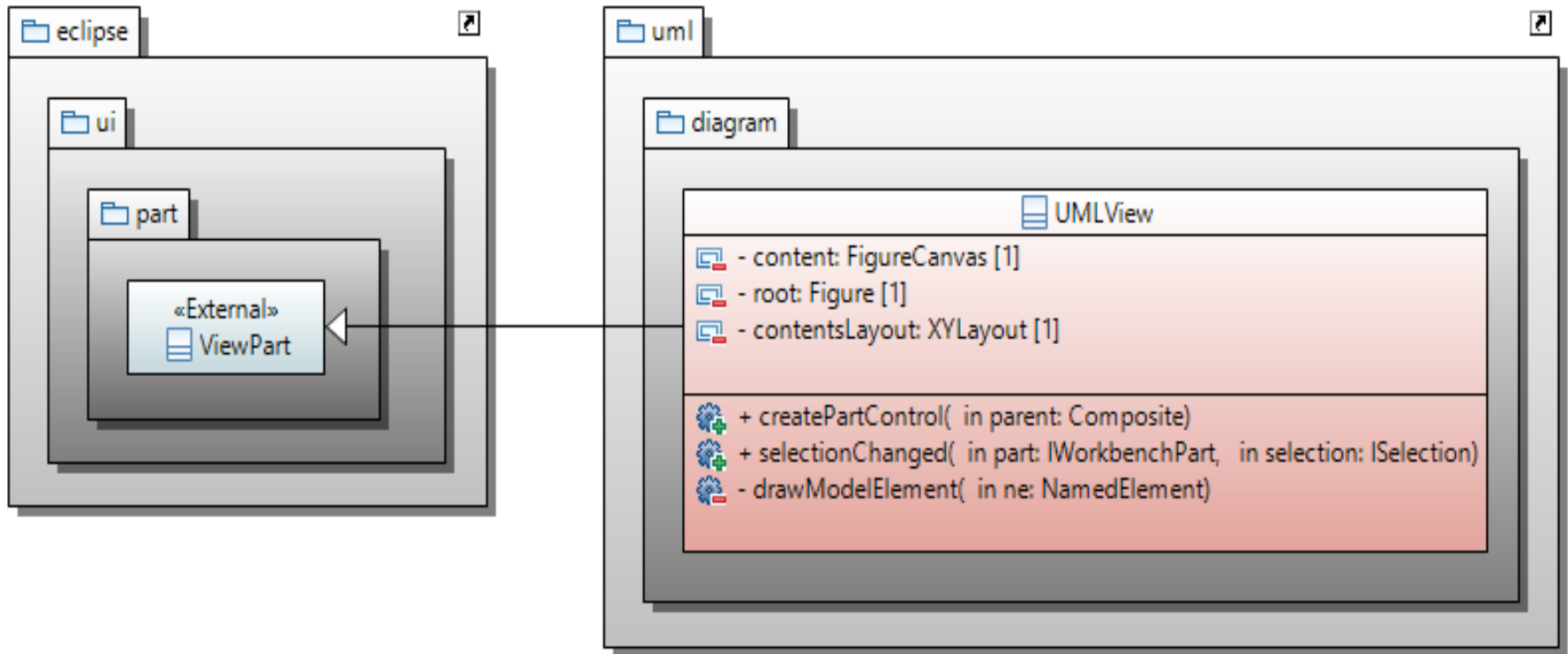
Цикл в оконном приложении

```
shell.open();  
  
while (!shell.isDisposed())  
    while (!d.readAndDispatch())  
        d.sleep();  
    }  
}
```

Приложение с простейшей фигурой – желтым прямоугольником

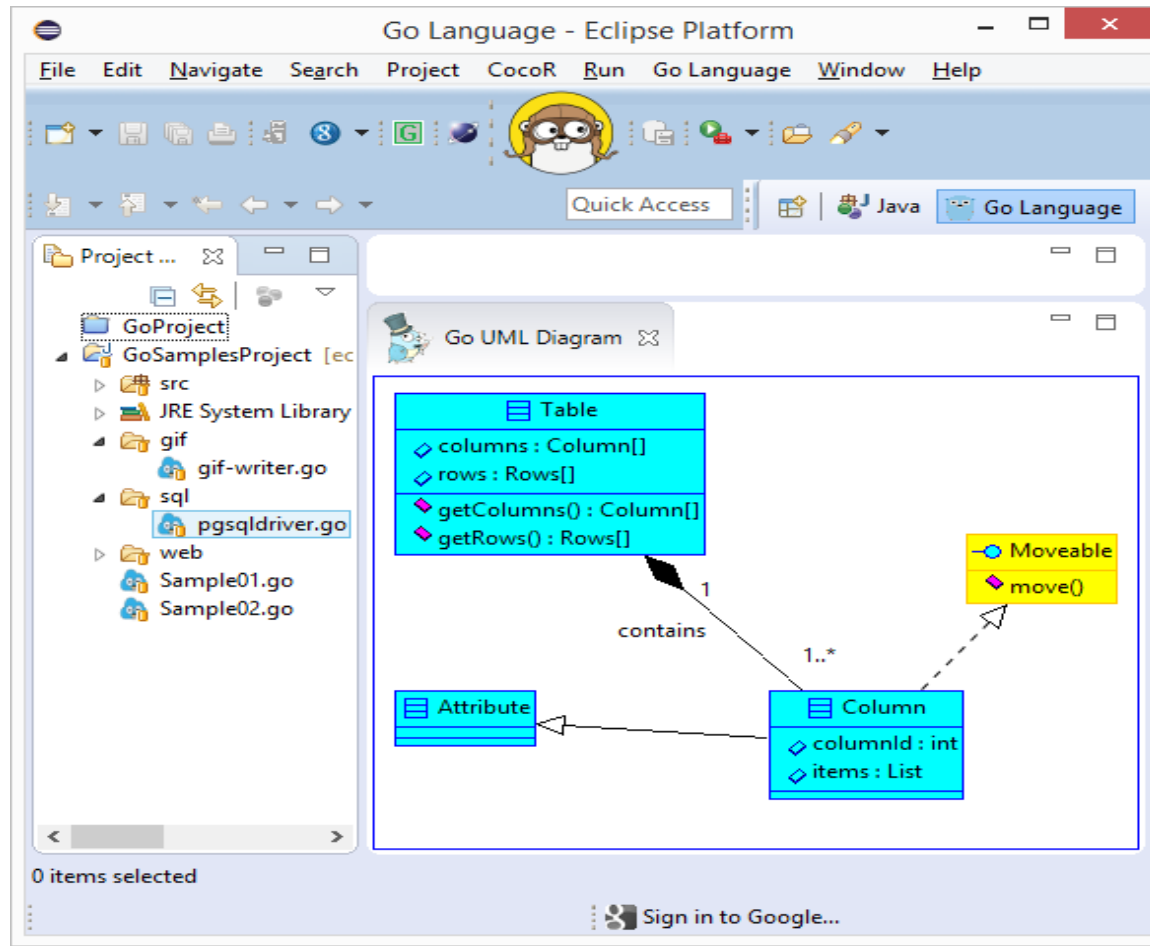


Расширение среды Eclipse для визуализации UML диаграмм



Расширение Eclipse для UML-диаграмм.

Пример UML-диаграммы



Использование Draw2D в плагине (расширении) среды Eclipse

```
package uml.diagram;  
  
import org.eclipse.draw2d.ColorConstants;  
import org.eclipse.draw2d.Figure;  
import org.eclipse.draw2d.FigureCanvas;  
import org.eclipse.draw2d.LineBorder;  
import org.eclipse.draw2d.XYLayout;  
import org.eclipse.draw2d.geometry.Rectangle;  
import org.eclipse.swt.graphics.Color;  
import org.eclipse.swt.widgets.Composite;  
import org.eclipse.ui.part.ViewPart;  
  
public class UMLDiagramView extends ViewPart {  
    private FigureCanvas content;  
    private Figure root;  
    private XYLayout contentsLayout;  
  
    public UMLDiagramView() {  
    }  
}
```

Использование библиотеки *Draw2D* в плагине среды Eclipse (фон для UML-диаграмм)

@Override

```
public void createPartControl (Composite parent) {  
    content = new FigureCanvas (parent, 0);  
    content.setBackground( new Color(null, 0, 255, 192) );  
  
    root = new Figure();  
    content.setContents(root);  
  
    contentsLayout = new XYLayout();  
    root.setLayoutManager(contentsLayout);  
  
    root.setBorder(new LineBorder(ColorConstants.blue, 2));  
    root.setOpaque(true);  
    root.setBackgroundColor(ColorConstants.white);  
}
```

Основные возможности draw2d.

Границы

- ❑ **GroupBoxBorder** - предоставляет границы используемые для группирования управляющих элементов
- ❑ **TitleBarBorder** – границы аналогичные границам окон
- ❑ **CompoundBorder** – граница, состоящая из двух границ
- ❑ **FrameBorder** – аналог TitleBarBorder. Может использоваться для создания фигур внутри границ
- ❑ **FocusBorder** – окружает фигуру границами, аналогичными изображению границ фигуры в фокусе
- ❑ **LineBorder** – окружает фигуру линией заданной ширины
- ❑ **MarginBorder** – невидимые границы (отступ от изображения фигуры)
- ❑ **SchemeBorder** – границы имитирующие тени
- ❑ **ButtonBorder** – границы имитирующие кнопку

Основные возможности draw2d.

Планировщики (Layout)

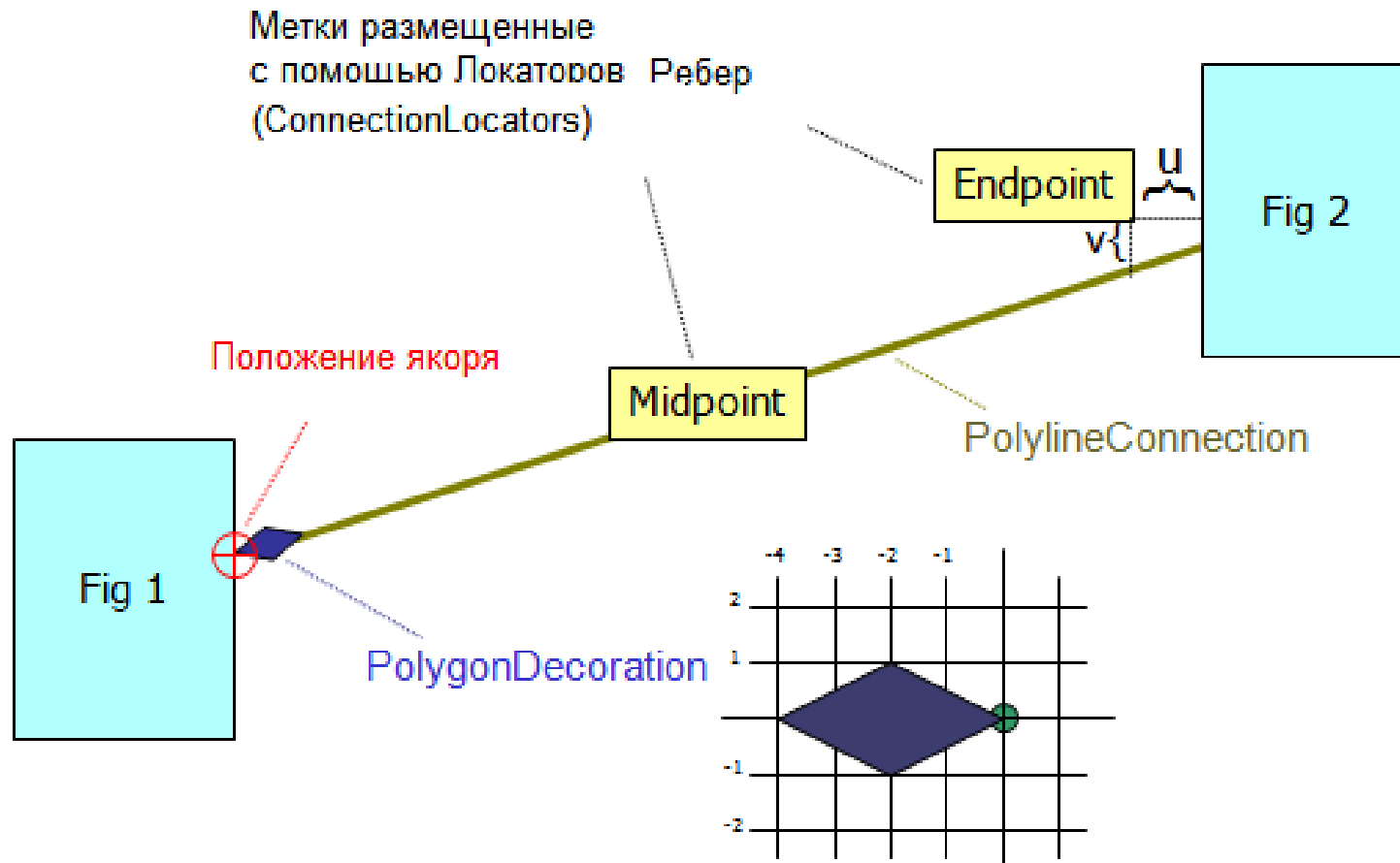
- **Планировщики используются для задания размеров и положения фигур-детей данной фигуры.**
 - **FlowLayout** – располагает фигуры в строки или колонки
 - **DelegatingLayout** – делегирует размещение фигур *локаторам* этих фигур. Фигура должна предоставить экземпляр подкласса класса *Locator* в качестве *ограничения* (constraint) этой фигуры
 - **XYLayout** – помещает фигуру в пределы (положение и размеры) заданные прямоугольником
 - **ScrollPaneLayout** – используется для прокрутки изображения фигур с помощью линеек прокрутки и поля просмотра (*viewport*).
 - **ViewportLayout** – используется для управления полем просмотра

Основные возможности draw2d.

Уровни

- ❑ **Уровни (Layers)** – прозрачные фигуры, предназначенные для использования панелях уровней (LayersPanels). Методы `containsPoint()` и `findFigureAt()` переопределяются так, что бы нажатие мышью «проходило сквозь» уровень
- ❑ **FreeformLayer** – предоставляет дополнительные возможности уровня, который не имеет границ по всем четырем направлениям. Они не имеют фиксированных границ и начала. Их фигуры-дети могут иметь отрицательные координаты
- ❑ **ConnectionLayer** – разновидность `FreeformLayer`, позволяющий добавлять фигуры-ребра (`connection`).
- ❑ **LayerPanels** – фигуры предназначенные для хранения уровней. Предназначены для создания, добавления, вставки, хранения, переупорядочивания уровней

Ребра, декорации ребер, якоря и локаторы



Основные возможности draw2d.

Локаторы (Locators)

- ❑ *Классы, реализующие интерфейс **Locator**, предназначены для размещения фигур присоединенных к ребрам. Единственный метод интерфейса:*

void relocate(IFigure target);

- ❑ *Подклассы класса **ConnectionLocator** используются для размещения фигур, которые присоединены к ребрам (connection). Они могут быть использованы для размещения стрелок на концах ребер или для размещения на ребрах других обозначений (adorns) или надписей.*
- ❑ *Локаторы предполагают, что фигура остается присоединенной к ребру в определенном месте при перемещении ребра*

Основные возможности draw2d.

Предопределенные локаторы

- ❑ **ArrowLocator** – предназначен для размещения стрелок на ребрах
- ❑ **EndpointLocator** – предназначен для размещения конечных точек ребра
- ❑ **MidpointLocator** – предназначен для размещения фигур в середине ребра
- ❑ **ConnectionEndpointLocator** - предназначен для размещения фигур в начале или конце ребра
- ❑ **RelativeLocator** – предназначен для задания положения фигуры в относительных координатах (от 0 до 1)

Основные возможности draw2d.

Якоря ребер (Anchors)

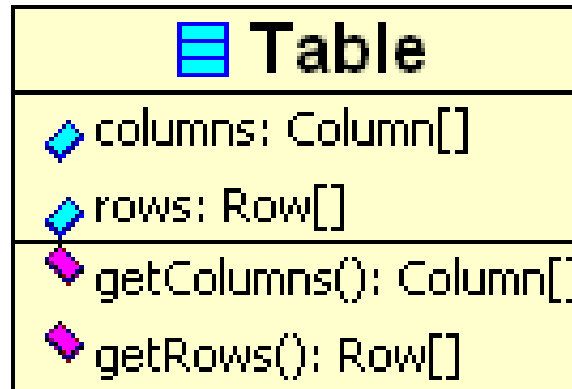
- ❑ **Anchor (якорь)** – представляет способ присоединения ребра к фигуре. Вычисляет точку соединения и сообщает зарегистрированным слушателям что конец ребра передвинут
- ❑ **ChorboxAnchor** – вычисляет точку пересечения ребра и фигуры, при условии что ребро направлено к центру фигуры
- ❑ **LabelAnchor** – подкласс класса ChorboxAnchor. Ребро должно быть направлено к текстовой метке (label). Положение якоря зависит от центра метки.
- ❑ **EllipseAnchor** – вычисляет точку пересечения ребра и эллипса, при условии что ребро направлено к центру эллипса
- ❑ **XYAnchor** – для задания якорей с фиксированной позицией

Основные возможности draw2d.

Маршрутизаторы ребер (Routers)

- ❑ **Router** (маршрутизатор) – используется для задания пути от одного якоря ребра до другого якоря ребра
- ❑ **NullConnectionRouter** – просто рисует прямую линию от одного якоря до другого якоря
- ❑ **AutomaticRouter** – базовый класс маршрутизаторов, которые предотвращают пересечение ребер
- ❑ **BendpointConnectionRouter** – маршрутизатор для ломаной линии, точки которой задает пользователь
- ❑ **ManhattanConnectionRouter** – маршрутизатор для ломаной линии, отрезки которой всегда расположено под углом 90 градусов относительно друг друга

Построение диаграммы классов языка UML



Узел графа на UML – диаграмме классов

Простой узел UML-диаграммы.

```
public class UMLClassNode extends Figure {  
  
    public UMLClassNode (String string) {  
        ToolbarLayout layout = new ToolbarLayout();  
        setLayoutManager(layout);  
  
        setBorder(new LineBorder(ColorConstants.black, 1));  
        setOpaque(true); // Фигура не прозрачна.  
  
        add(new Label(string));  
    }  
}
```

Сборка UML-диаграммы в оконном приложении

```
public class UMLClassFigureTest {  
  
    public static void main (String args[ ]) {  
        Display d = new Display();  
  
        final Shell shell = new Shell(d);  
        shell.setSize(400, 400);  
        shell.setText("UMLClassFigure Test");  
  
        LightweightSystem lws = new LightweightSystem(shell);  
  
        Figure contents = new Figure();  
  
        XYLayout contentsLayout = new XYLayout();  
        contents.setLayoutManager(contentsLayout);  
  
        // Вставка изображения  
  
        lws.setContents(contents);  
  
        shell.open();  
        while (!shell.isDisposed())  
            while (!d.readAndDispatch())  
                d.sleep();  
    }  
}
```

Фигуры – дети могут
иметь произвольные
координаты

Корневая фигура

Вставка узлов-классов в корневую фигуру UML-диаграммы

```
public class UMLDiagramFactory {  
    public static void addDiagram (Figure contents) {  
        XYLayout contentsLayout = new XYLayout();  
        contents.setLayoutManager(contentsLayout);  
  
        Figure classFigure1 = new UMLClassNode("Table");  
        contents.add(classFigure1);  
  
        contentsLayout.setConstraint(classFigure1,  
            new Rectangle(10, 10, -1, -1));  
  
        Figure classFigure2 = new UMLClassNode("Column");  
        contents.add(classFigure2);  
  
        contentsLayout.setConstraint(classFigure2,  
            new Rectangle(200, 200, -1, -1));  
    }  
}
```

Вставка узла-класса в корневую фигуру UML-диаграммы

```
Figure umlClass = new UMLClassNode("Table");
```

```
contents.add(umlClass) ;
```

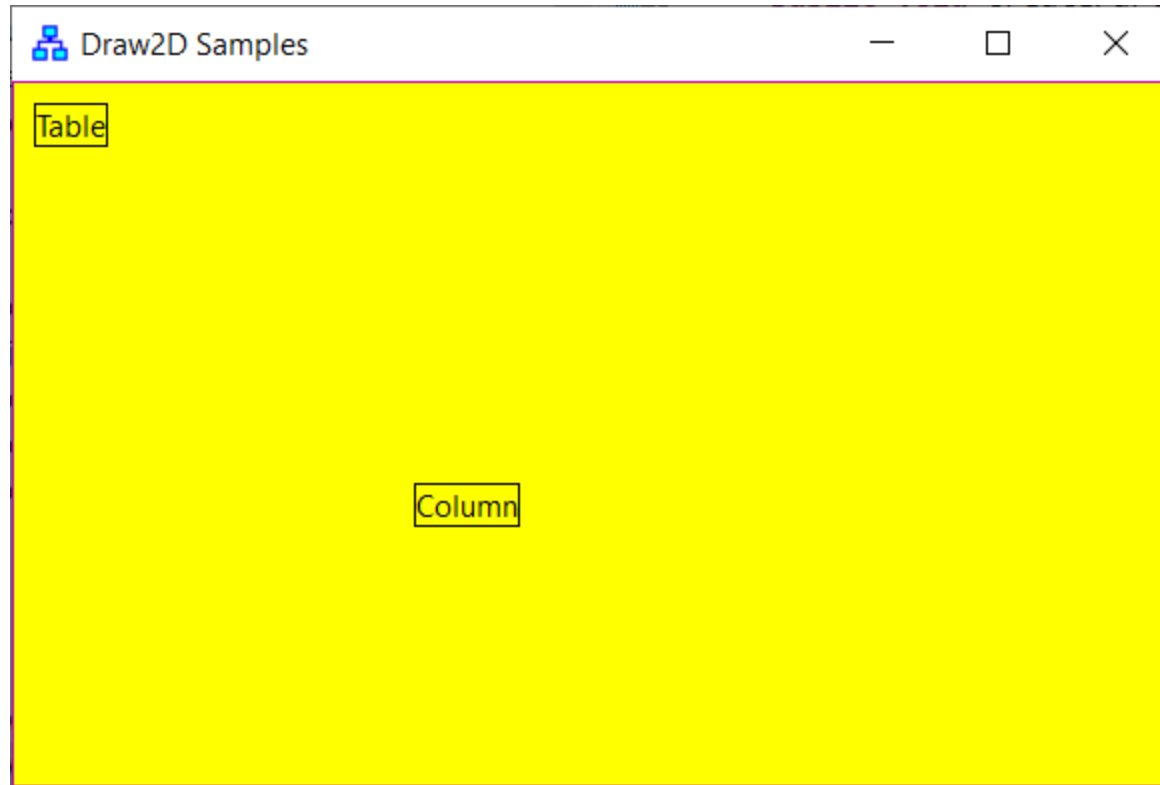
```
contentsLayout.setConstraint(umlClass,  
    new Rectangle(10,10, -1,-1));
```

Вставка узла-класса
в корневую фигуру

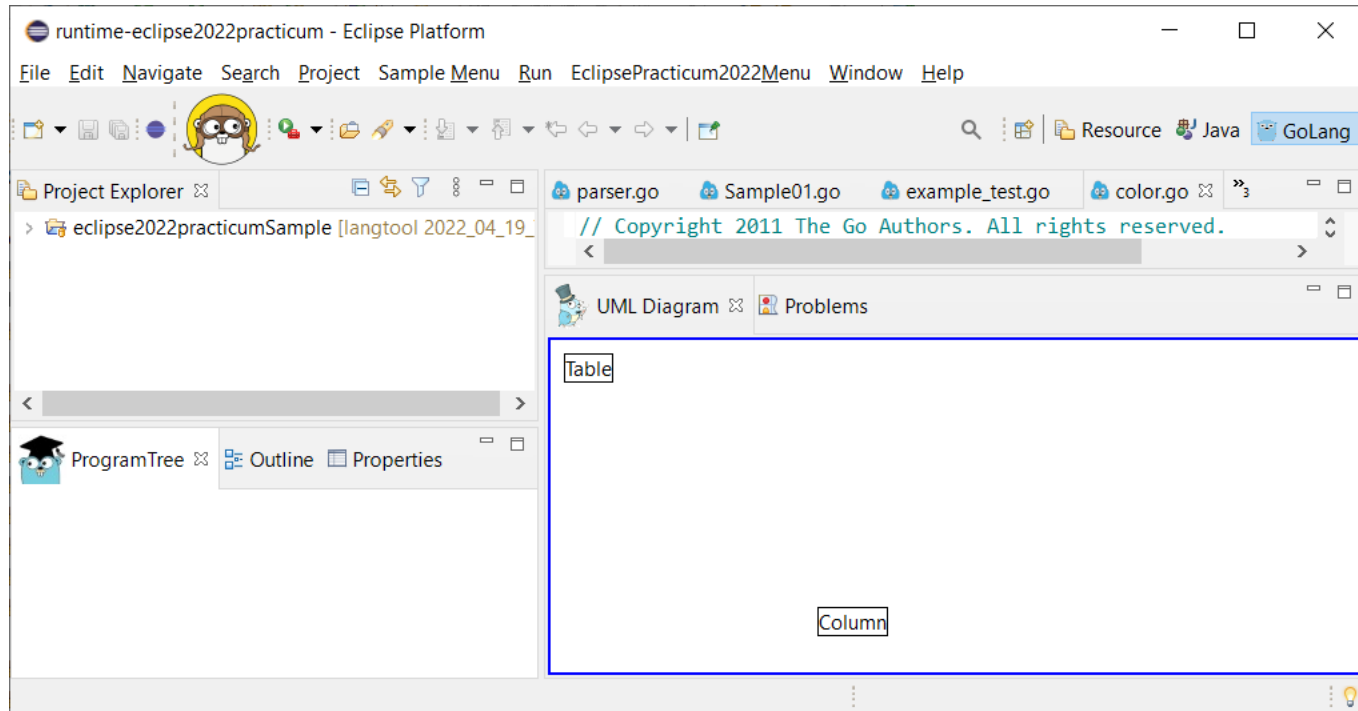
Задание
координат узла

Задание размеров узла
(-1 – настройка на размер
содержимого)

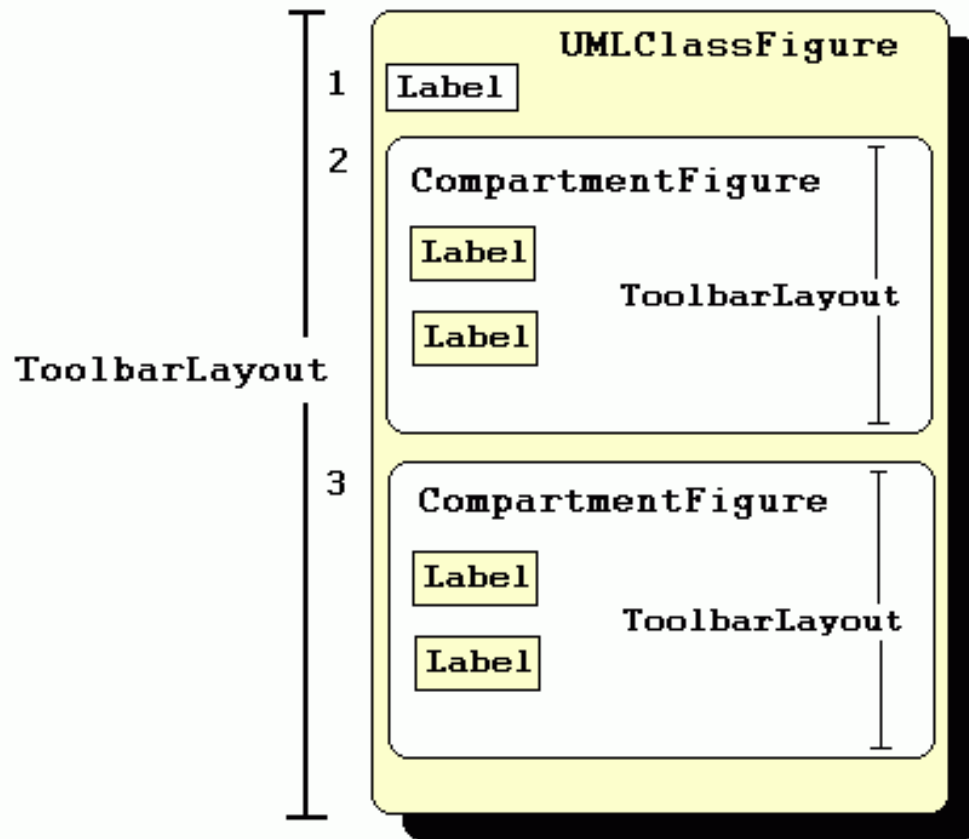
Отрисовка на UML-диаграмме простейших узлов-классов.



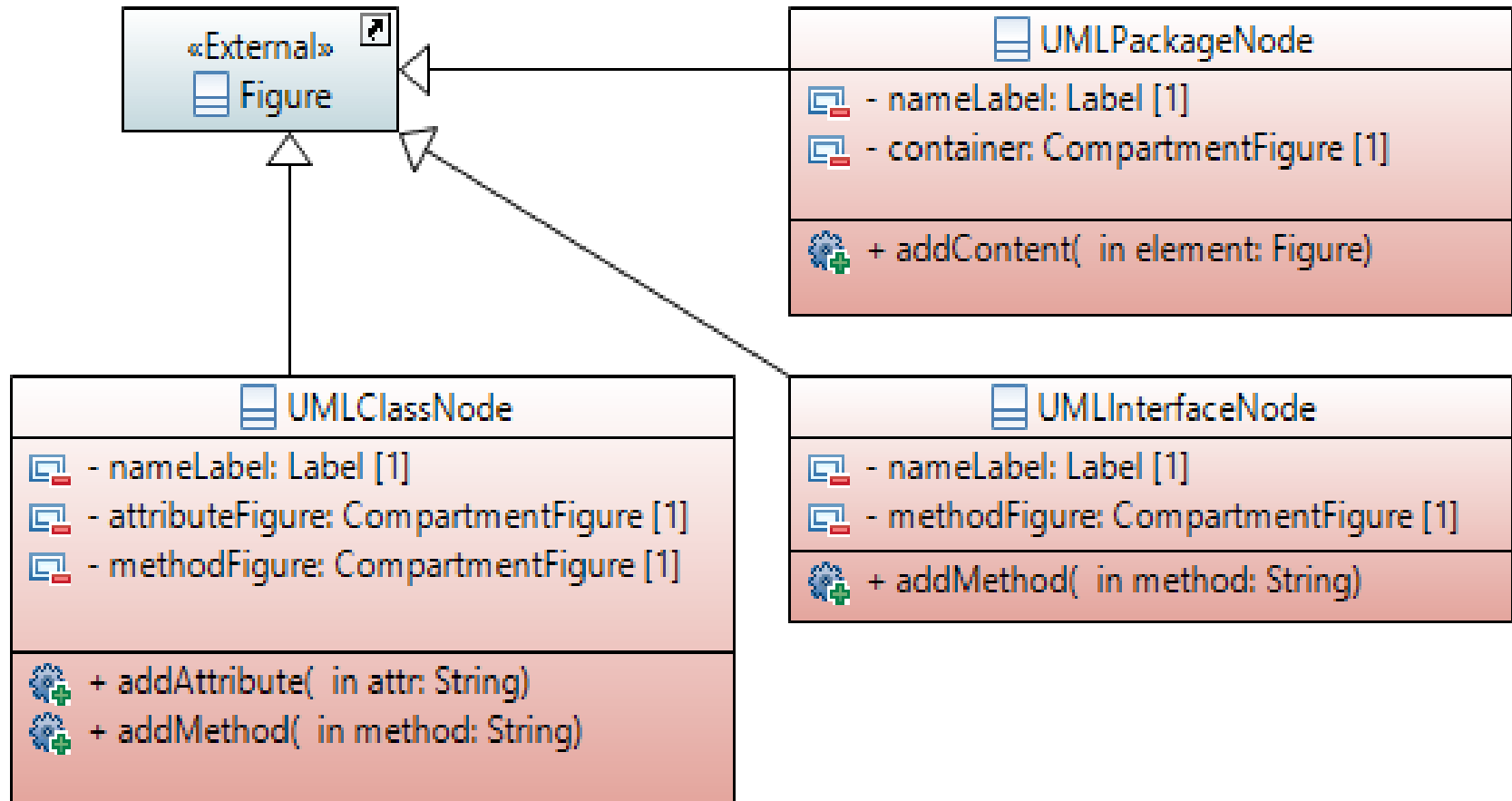
Отрисовка на UML-диаграмме простейших узлов-классов.



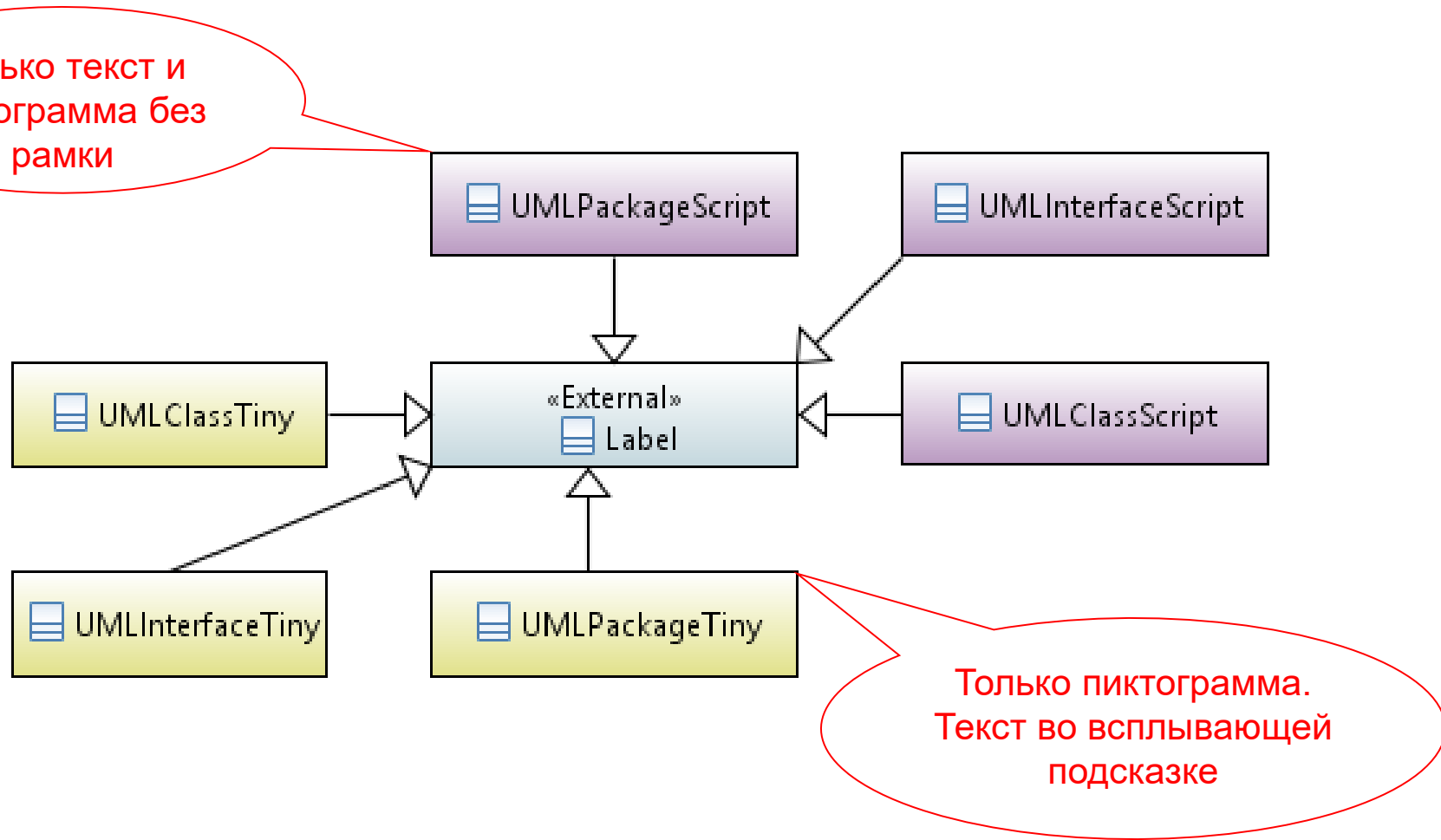
Структура узла графа на диаграмме классов



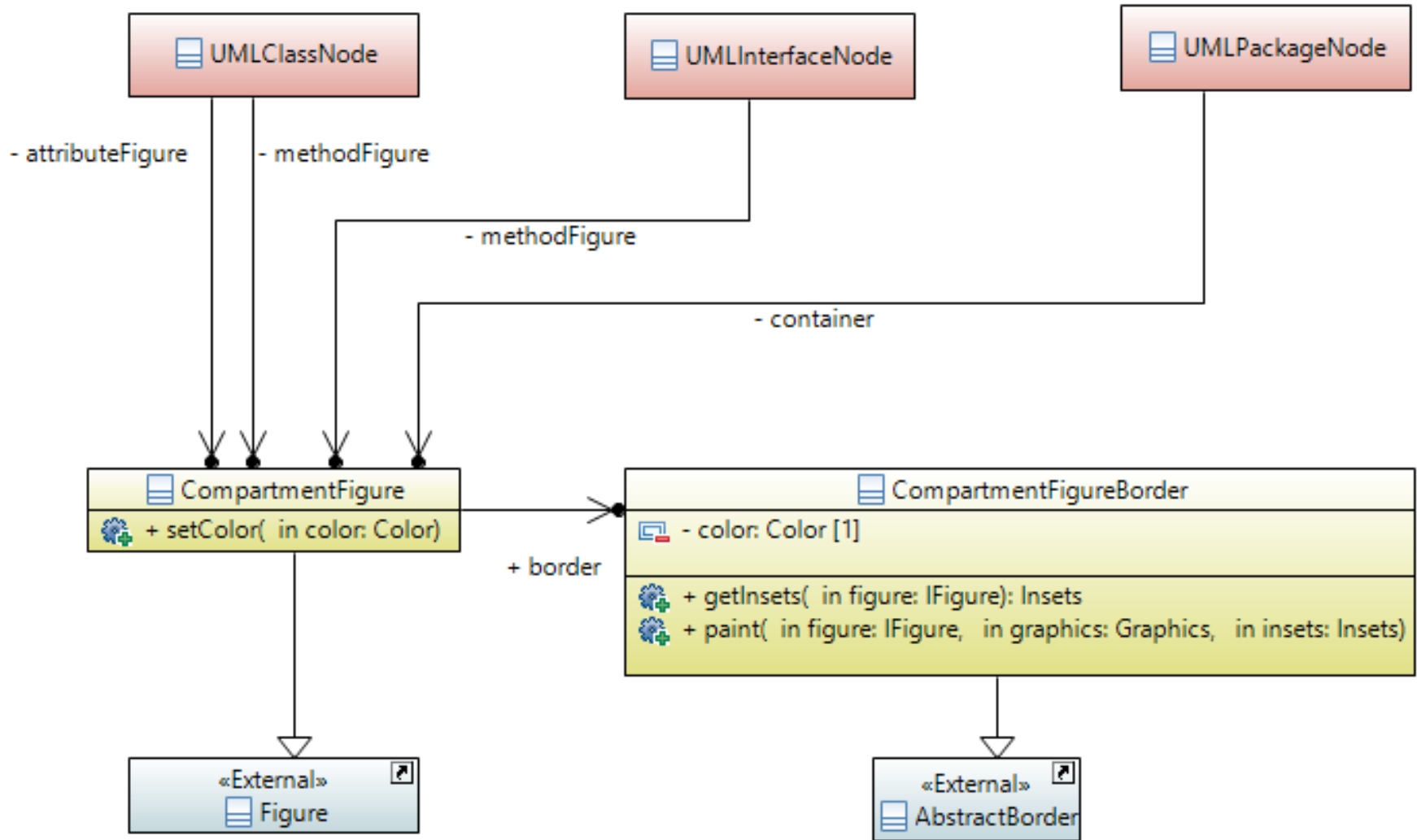
Узлы UML-диаграммы. Узлы с секциями



Узлы UML-диаграммы. Узлы без секций



Секции и границы UML-диаграммы



Граница содержимого узла

```
public class CompartmentFigureBorder extends AbstractBorder {  
  
    public Insets getInsets(IFigure figure)  
        { return new Insets(1,0,0,0); }  
  
    public void paint(IFigure figure, Graphics graphics, Insets insets) {  
        graphics.drawLine(getPaintRectangle(figure, insets).getTopLeft(),  
            tempRect.getTopRight());  
    }  
}
```

Фигура – секция узла

```
public class CompartmentFigure extends Figure {  
  
    public CompartmentFigure() {  
        ToolBarLayout layout = new ToolBarLayout();  
        layout.setMinorAlignment(ToolBarLayout.ALIGN_TOPLEFT);  
        layout.setStretchMinorAxis(false);  
        layout.setSpacing(2);  
        setLayoutManager(layout);  
        setBorder( new CompartmentFigureBorder() );  
    }  
  
}
```

Создание фигуры – узла графа

```
public class UMLClassNode extends Figure {  
  
    public static Color classColor = new Color(null, 255, 255, 206);  
  
    private CompartmentFigure attributeFigure = new CompartmentFigure();  
    private CompartmentFigure methodFigure = new CompartmentFigure();  
  
    public UMLClassNode (String name) {  
        ToolbarLayout layout = new ToolbarLayout();  
        setLayoutManager(layout);  
  
        setBorder( new LineBorder(ColorConstants.black, 1) );  
        setBackgroundColor(classColor);  
        setOpaque(true); // Фигура не прозрачна.  
  
        add( new Label(name, UMLImages.classImage) );  
        add(attributeFigure);  
        add(methodFigure);  
    }  
    public CompartmentFigure getAttributesCompartment()  
        { return attributeFigure; }  
    public CompartmentFigure getMethodsCompartment()  
        { return methodFigure; }  
}
```

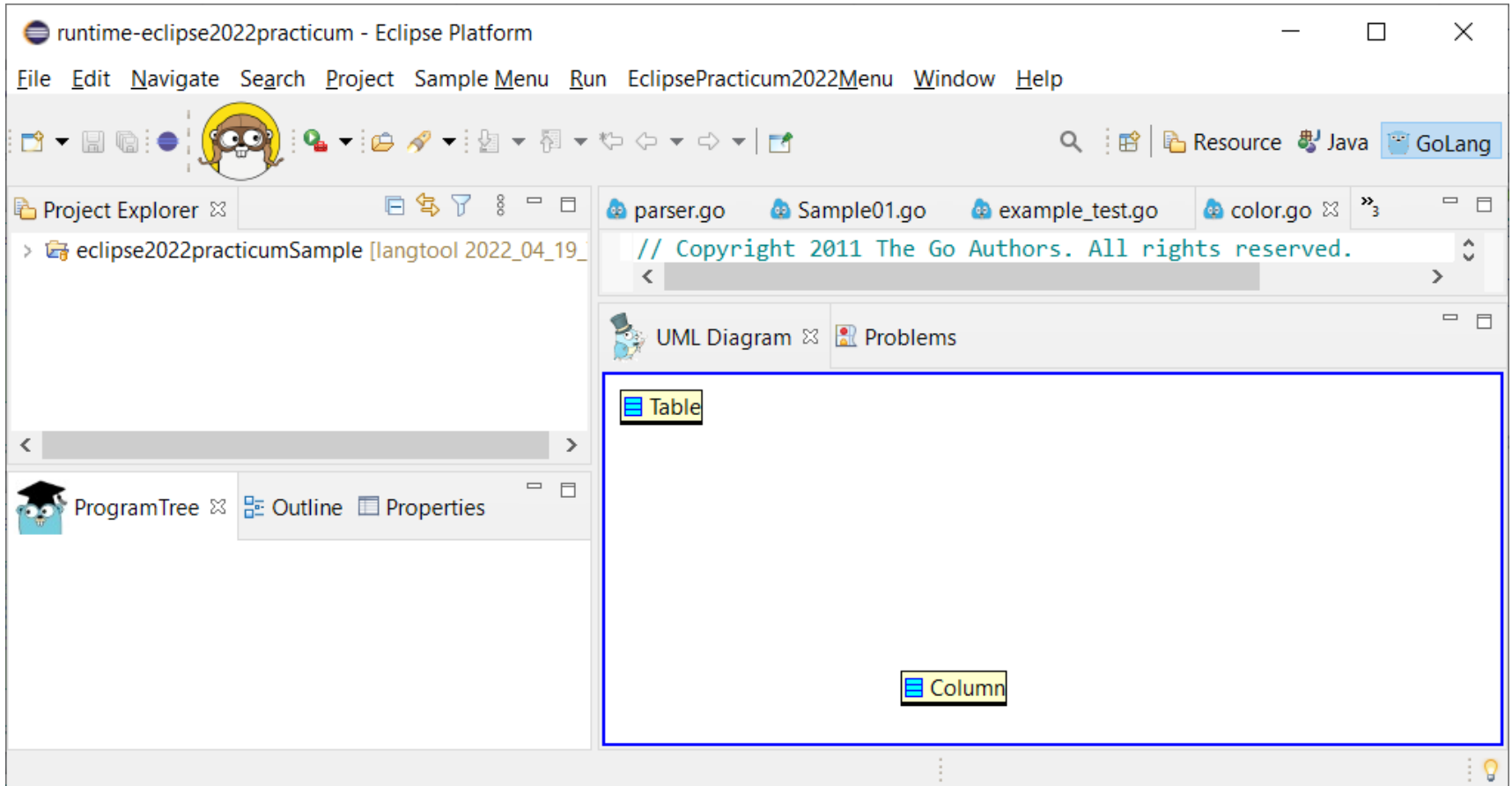
Добавление пиктограммы для узла графа

```
public class UMLClassNode extends Figure {  
  
    public static Color classColor = new Color(null, 255, 255, 206);  
  
    private CompartmentFigure attributeFigure = new CompartmentFigure();  
    private CompartmentFigure methodFigure = new CompartmentFigure();  
  
    public UMLClassNode (String name) {  
        Image classImage = new Image(null,  
            UMLImages.class.getResourceAsStream("classIcon.gif"));  
  
        ToolbarLayout layout = new ToolbarLayout();  
        setLayoutManager(layout);  
        setBorder( new LineBorder(ColorConstants.black, 1) );  
        setBackgroundColor(classColor);  
        setOpaque(true); // Фигура не прозрачна.  
  
        add( new Label(name, classImage) );  
        add(attributeFigure);  
        add(methodFigure);  
    }  
}
```

Добавление пиктограммы для узла графа

```
public class UMLClassNode extends Figure {  
  
    public static Color classColor = new Color(null, 255, 255, 206);  
  
    private CompartmentFigure attributeFigure = new CompartmentFigure();  
    private CompartmentFigure methodFigure = new CompartmentFigure();  
  
    public UMLClassNode (String name) {  
        ToolbarLayout layout = new ToolbarLayout();  
        setLayoutManager(layout);  
        setBorder( new LineBorder(ColorConstants.black, 1) );  
        setBackgroundColor(classColor);  
        setOpaque(true); // Фигура не прозрачна.  
  
        add( new Label(name, UMLImages.classImage) );  
        add(attributeFigure);  
        add(methodFigure);  
    }  
}
```

Узлы-классы с пиктограммами на UML-диаграмме.



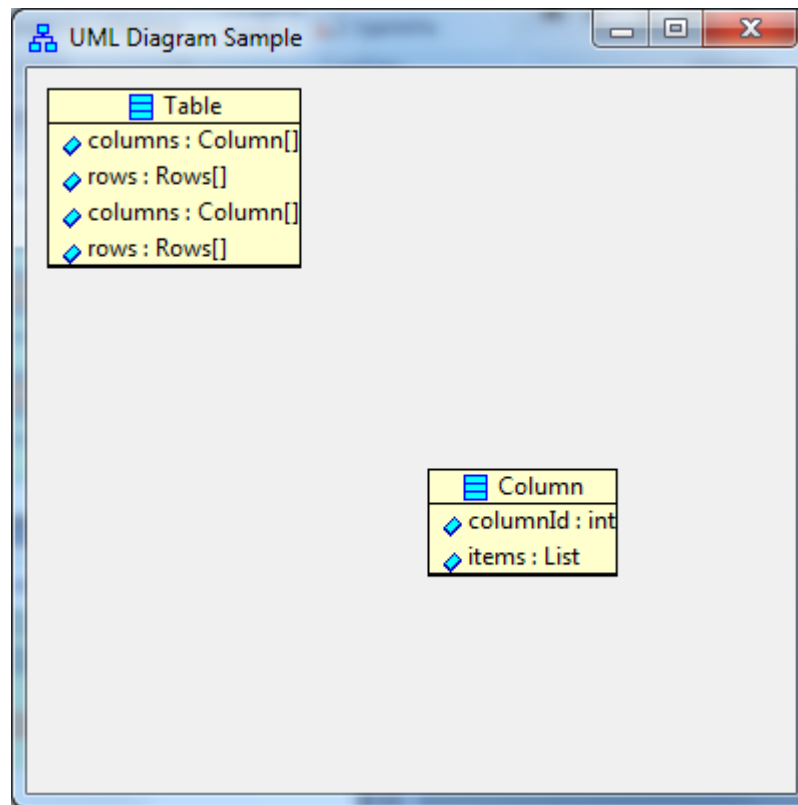
Добавление атрибутов класса

```
public static void addDiagram(Figure contents) {  
    XYLayout contentsLayout = new XYLayout();  
    contents.setLayoutManager(contentsLayout);  
  
    UMLClassNode classFigure1 = new UMLClassNode ("Table");  
    classFigure1.addAttribute("columns : Column[]");  
    classFigure1.addAttribute("rows : Rows[]");  
    contents.add(classFigure1);  
  
    contentsLayout.setConstraint(classFigure1,  
                                new Rectangle(10, 10, -1, -1));  
  
    UMLClassNode classFigure2 = new UMLClassNode ("Column");  
    classFigure2.addAttribute("columnId : int");  
    classFigure2.addAttribute("items : List");  
    contents.add(classFigure2);  
  
    contentsLayout.setConstraint(classFigure2,  
                                new Rectangle(200, 200, -1, -1));  
}
```

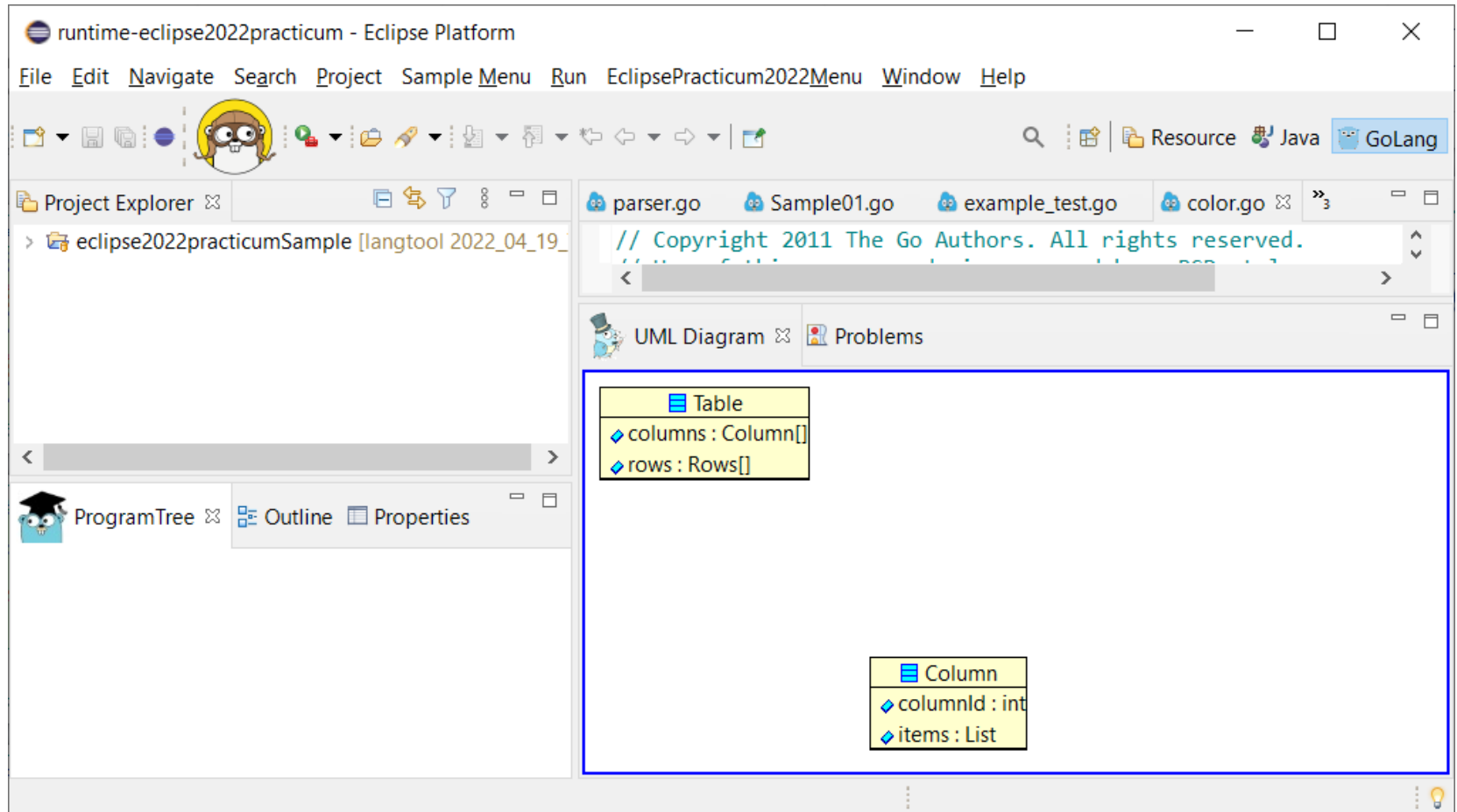
Создание атрибута класса — узла

```
public class UMLClassNode extends Figure {  
  
    public static Color classColor = new Color(null, 255, 255, 206);  
  
    private CompartmentFigure attributeFigure = new CompartmentFigure();  
    private CompartmentFigure methodFigure = new CompartmentFigure();  
  
    // ...  
  
    public void addAttribute(String attr) {  
        attributeFigure.add( new Label(attr, UMLImages.attributeImage) );  
    }  
}
```

На UML-диаграмме показываются атрибуты класса



На UML-диаграмме показываются атрибуты класса



Добавление методов класса

```
public static void addDiagram(Figure contents) {
    XYLayout contentsLayout = new XYLayout();
    contents.setLayoutManager(contentsLayout);

    UMLClassNode classFigure1 = new UMLClassNode ("Table");
    classFigure1.addAttribute("columns : Column[]");
    classFigure1.addAttribute("rows : Rows[]");
    classFigure1.addMethod("getColumns() : Column[]");
    classFigure1.addMethod("getRows() : Rows[]");
    contents.add(classFigure1);

    contentsLayout.setConstraint(classFigure1, new Rectangle(10, 10, -1, -1));

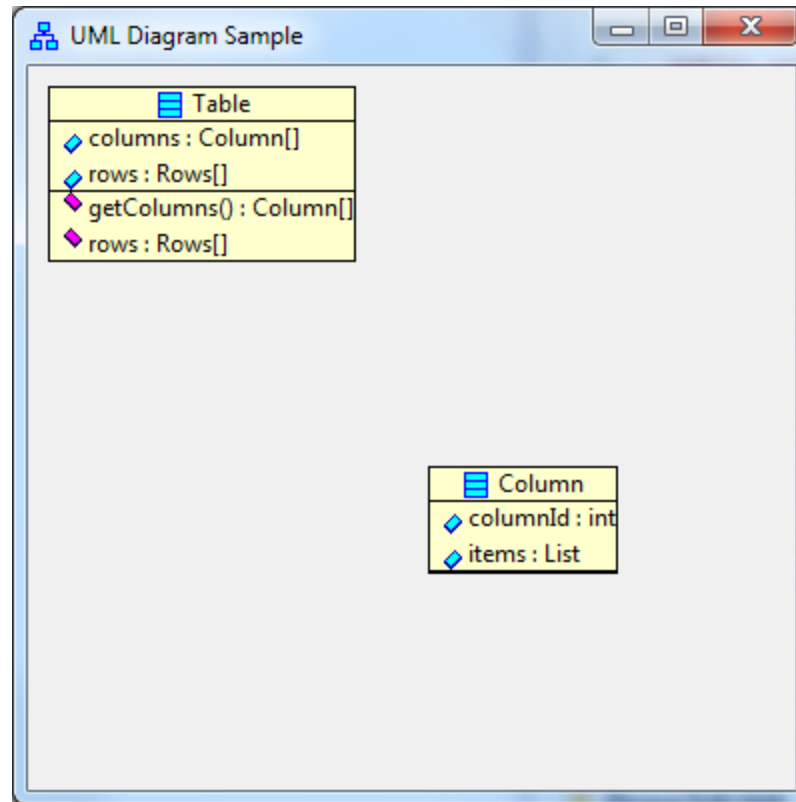
    UMLClassNode classFigure2 = new UMLClassNode ("Column");
    classFigure2.addAttribute("columnId : int");
    classFigure2.addAttribute("items : List");
    contents.add(classFigure2);

    contentsLayout.setConstraint(classFigure2, new Rectangle(200, 200, -1, -1));
}
```

Создание метода для класса – узла

```
public class UMLClassNode extends Figure {  
  
    public static Color classColor = new Color(null, 255, 255, 206);  
  
    private CompartmentFigure attributeFigure = new CompartmentFigure();  
    private CompartmentFigure methodFigure = new CompartmentFigure();  
  
    // ...  
  
    public void addAttribute(String attr) {  
        attributeFigure.add(new Label(attr, UMLImages.attributeImage));  
    }  
  
    public void addMethod(String method) {  
        methodFigure.add(new Label(method, UMLImages.methodImage));  
    }  
  
}
```

На UML-диаграмме показываются методы класса



На UML-диаграмме показываются методы класса

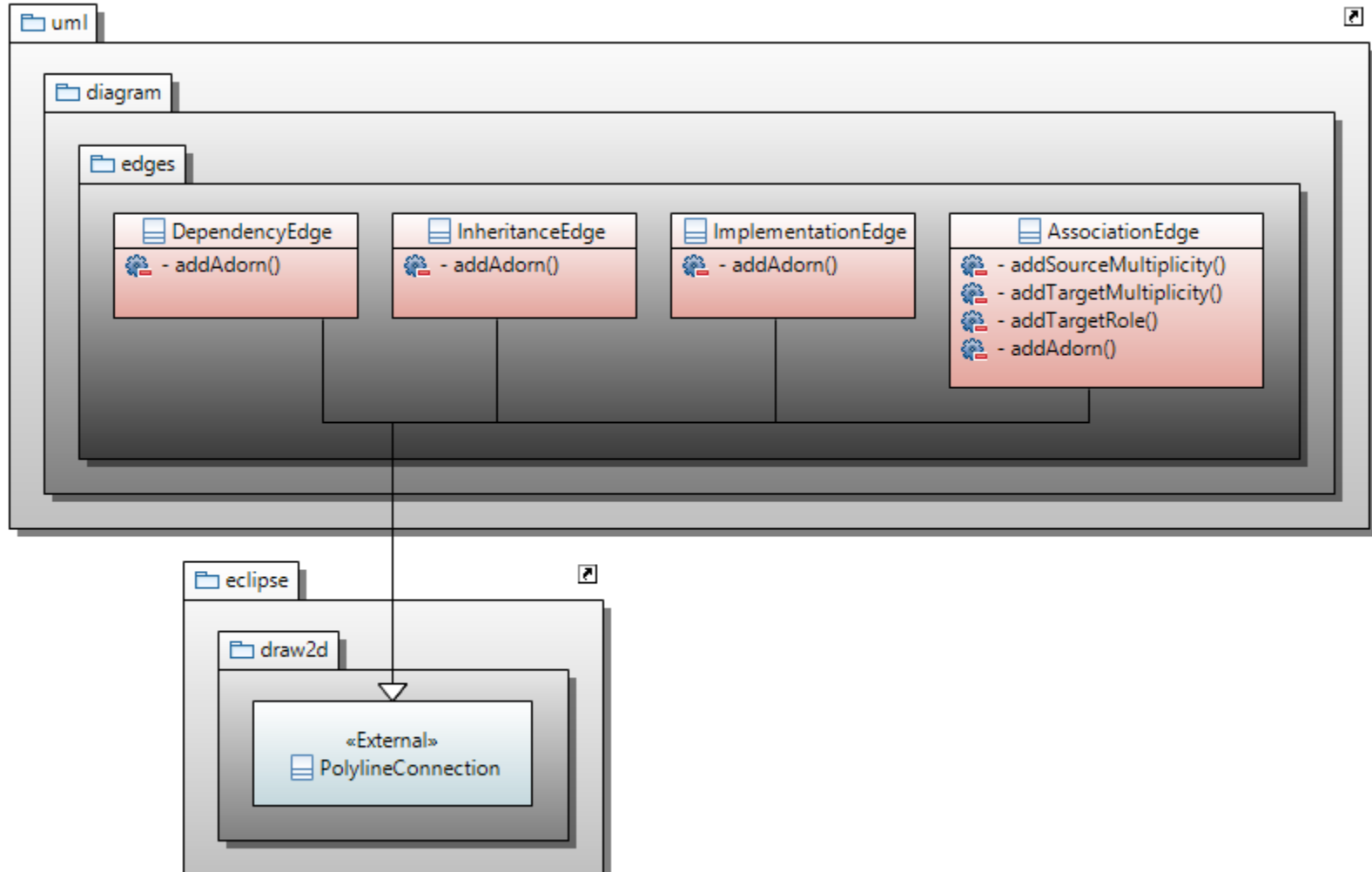
The screenshot shows the Eclipse IDE interface for a GoLang project named 'runtime-eclipse2022practicum'. The main editor displays a Go file with the following content:

```
// Copyright 2011 The Go Authors. All rights reserved.  
// ...
```

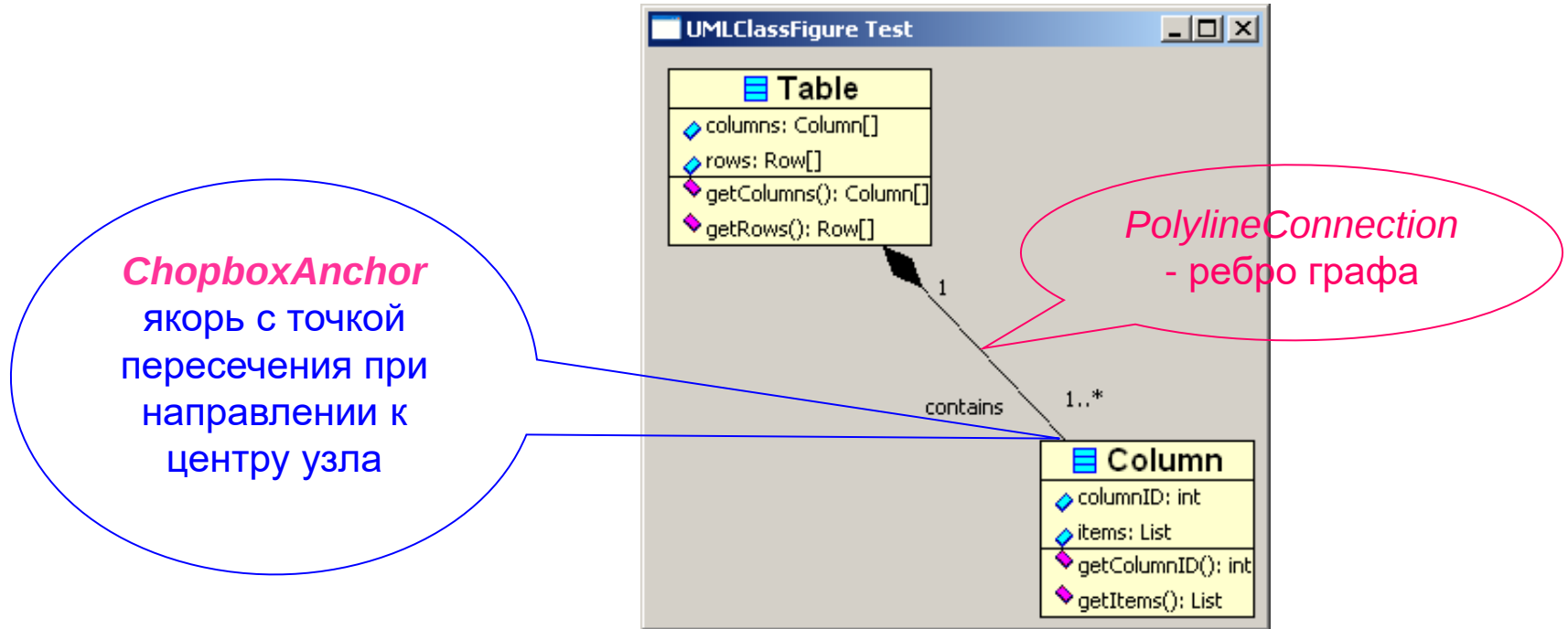
The UML Diagram window shows the following classes:

- Table**
 - columns : Column[]
 - rows : Rows[]
 - getColumns() : Column[]
 - getRows() : Rows[]
- Column**
 - columnId : int
 - items : List

Ребра на диаграмме статической структуры языка UML



Ребро – отношение ассоциации языка UML



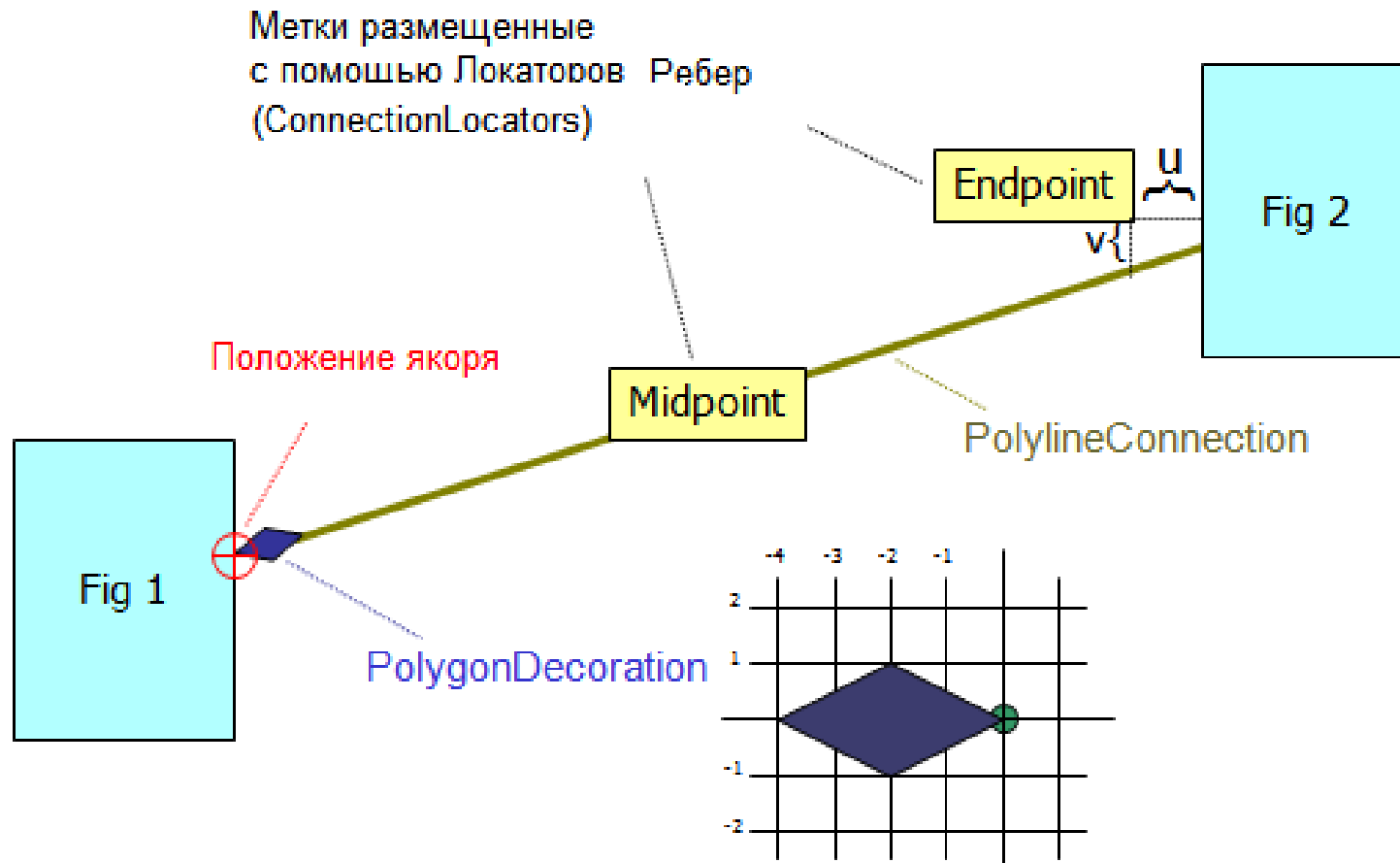
Добавление отношения ассоциации

```
public static void addDiagram(Figure contents) {  
    XYLayout contentsLayout = new XYLayout();  
    contents.setLayoutManager(contentsLayout);  
  
    UMLClassNode classFigure1 = new UMLClassNode ("Table");  
    classFigure1.addAttribute("columns : Column[]");  
    classFigure1.addAttribute("rows : Rows[]");  
    classFigure1.addMethod("getColumns() : Column[]");  
    classFigure1.addMethod("getRows() : Rows[]");  
    contents.add(classFigure1);  
    contentsLayout.setConstraint(classFigure1,  
                                new Rectangle(10, 10, -1, -1));  
  
    UMLClassNode classFigure2 = new UMLClassNode ("Column");  
    classFigure2.addAttribute("columnId : int");  
    classFigure2.addAttribute("items : List");  
    contents.add(classFigure2);  
    contentsLayout.setConstraint(classFigure2,  
                                new Rectangle(200, 200, -1, -1));  
  
    AssociationEdge as = new AssociationEdge(classFigure1, classFigure2);  
    contents.add(as);  
}
```

Отношение ассоциации на диаграмме классов

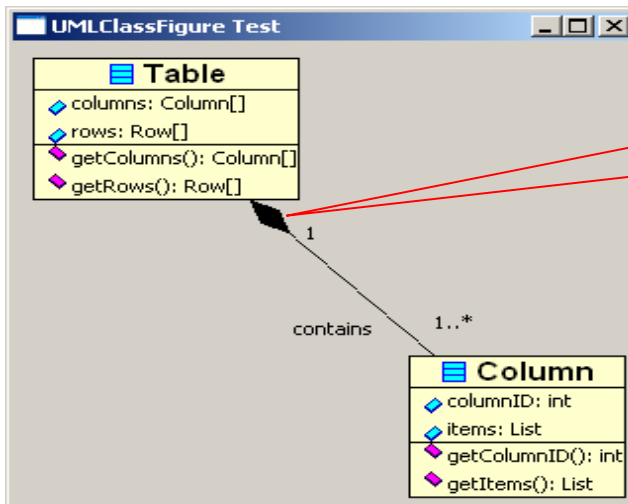
```
public class AssociationEdge extends PolylineConnection {  
  
    public AssociationEdge(UMLClassNode classFigure1, UMLClassNode classFigure2) {  
        ChopboxAnchor sourceAnchor = new ChopboxAnchor(classFigure1);  
        ChopboxAnchor targetAnchor = new ChopboxAnchor(classFigure2);  
  
        setSourceAnchor(sourceAnchor);  
        setTargetAnchor(targetAnchor);  
  
        addAdorn();  
    }  
}
```

Ребра, декорации ребер, якоря и локаторы



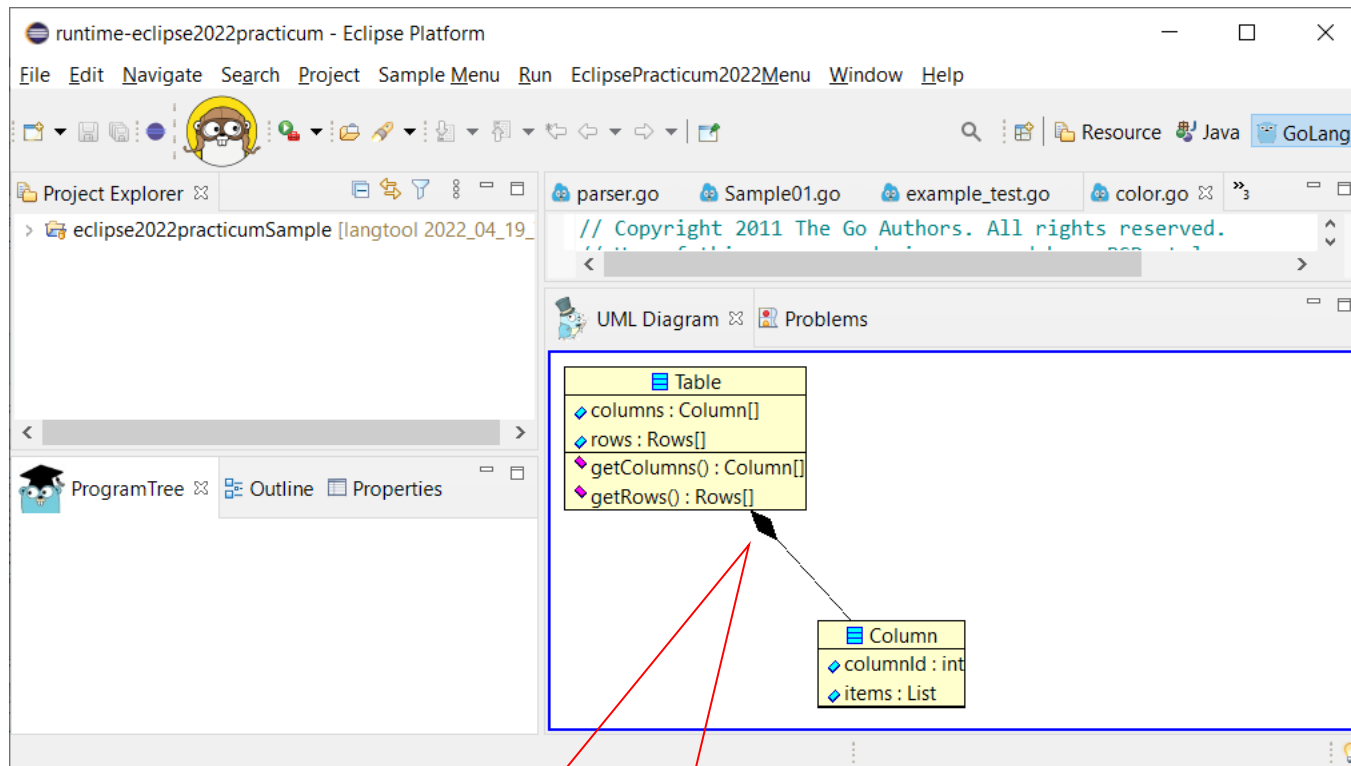
Создание декорации для ребра графа

```
private void addAdorn() {  
    PolygonDecoration decoration = new PolygonDecoration();  
  
    PointList decorationPointList = new PointList();  
    decorationPointList.addPoint( 0, 0);  
    decorationPointList.addPoint(-2, 2);  
    decorationPointList.addPoint(-4, 0);  
    decorationPointList.addPoint(-2, -2);  
  
    decoration.setTemplate(decorationPointList);  
    setSourceDecoration(decoration);  
}
```



PolygonDecoration
декорация в виде
ромба

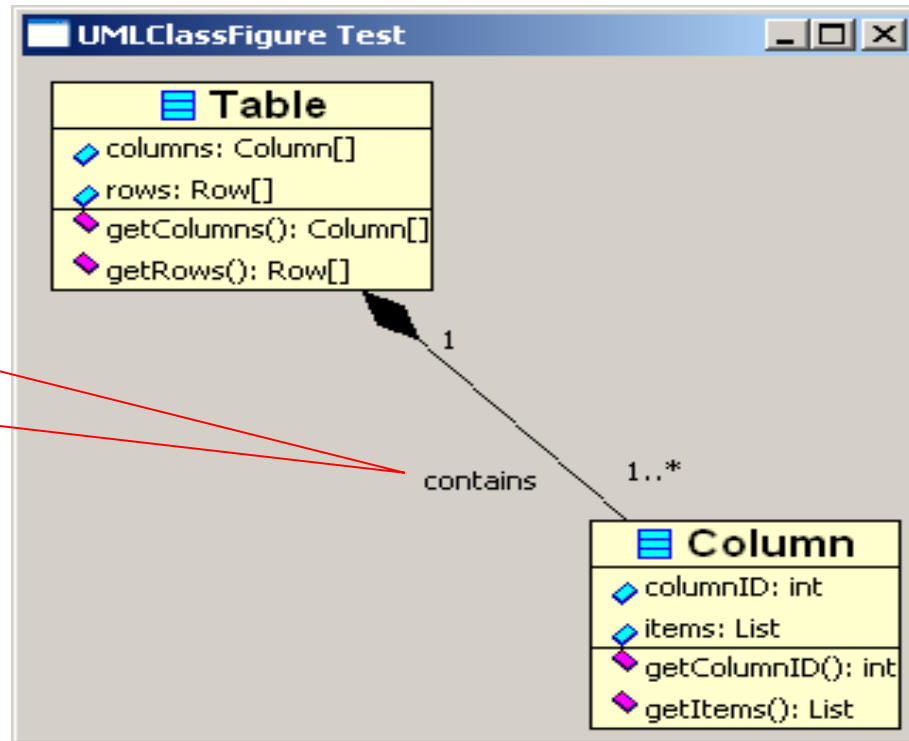
Создание декорации для ребра графа



PolygonDecoration
декорация в виде ромба

Создание меток для ребер графа

Метка – роль
окончания
отношение
ассоциации



Добавление множественности для отношения ассоциации

```
public class AssociationEdge extends PolylineConnection {  
  
    public AssociationEdge(UMLClassNode classFigure1, UMLClassNode classFigure2)  
    {  
        ChopboxAnchor sourceAnchor = new ChopboxAnchor(classFigure1);  
        ChopboxAnchor targetAnchor = new ChopboxAnchor(classFigure2);  
        setSourceAnchor(sourceAnchor);  
        setTargetAnchor(targetAnchor);  
  
        addAdorn();  
        addSourceMultiplicity("1..*");  
        addTargetMultiplicity("1..*");  
    }  
}
```

Задание положения множественности для отношения ассоциации

Label targetMultiplicityLabel

```
public void addTargetMultiplicity(String multiplicity) {  
    ConnectionEndpointLocator targetEndpointLocator =  
        new ConnectionEndpointLocator(this, true);  
  
    targetEndpointLocator.setVDistance(15);  
  
    targetMultiplicityLabel = new Label(multiplicity);  
    add(targetMultiplicityLabel, targetEndpointLocator);  
}  
  
public void setTargetMultiplicity(String multiplicity) {  
    targetMultiplicityLabel.setText(multiplicity);  
}
```

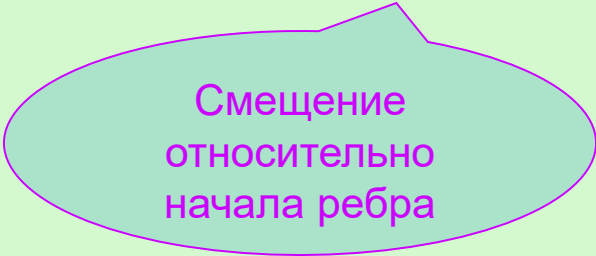
Смещение по
вертикали

Смещение
относительно
конца ребра

Задание положения множественности для отношения ассоциации

```
Label sourceMultiplicityLabel;
```

```
public void addSourceMultiplicity(String multiplicity) {  
    ConnectionEndpointLocator sourceEndpointLocator =  
        new ConnectionEndpointLocator(this, false);  
    sourceEndpointLocator.setVDistance(15);  
  
    sourceMultiplicityLabel = new Label("1");  
    add(sourceMultiplicityLabel, sourceEndpointLocator);  
}
```



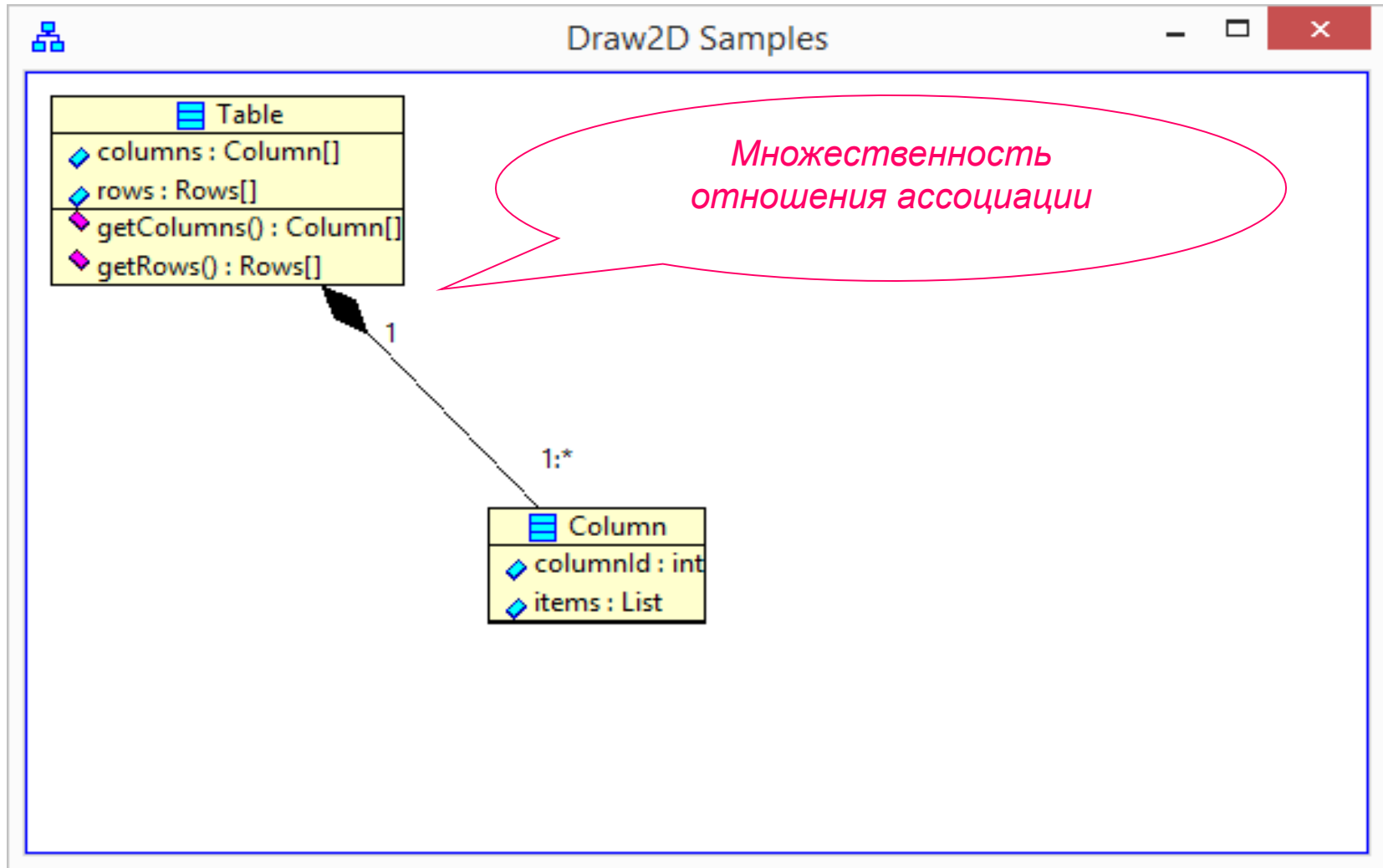
Смещение
относительно
начала ребра

```
public void setSourceMultiplicity(String multiplicity) {  
    sourceMultiplicityLabel.setText(multiplicity);  
}
```

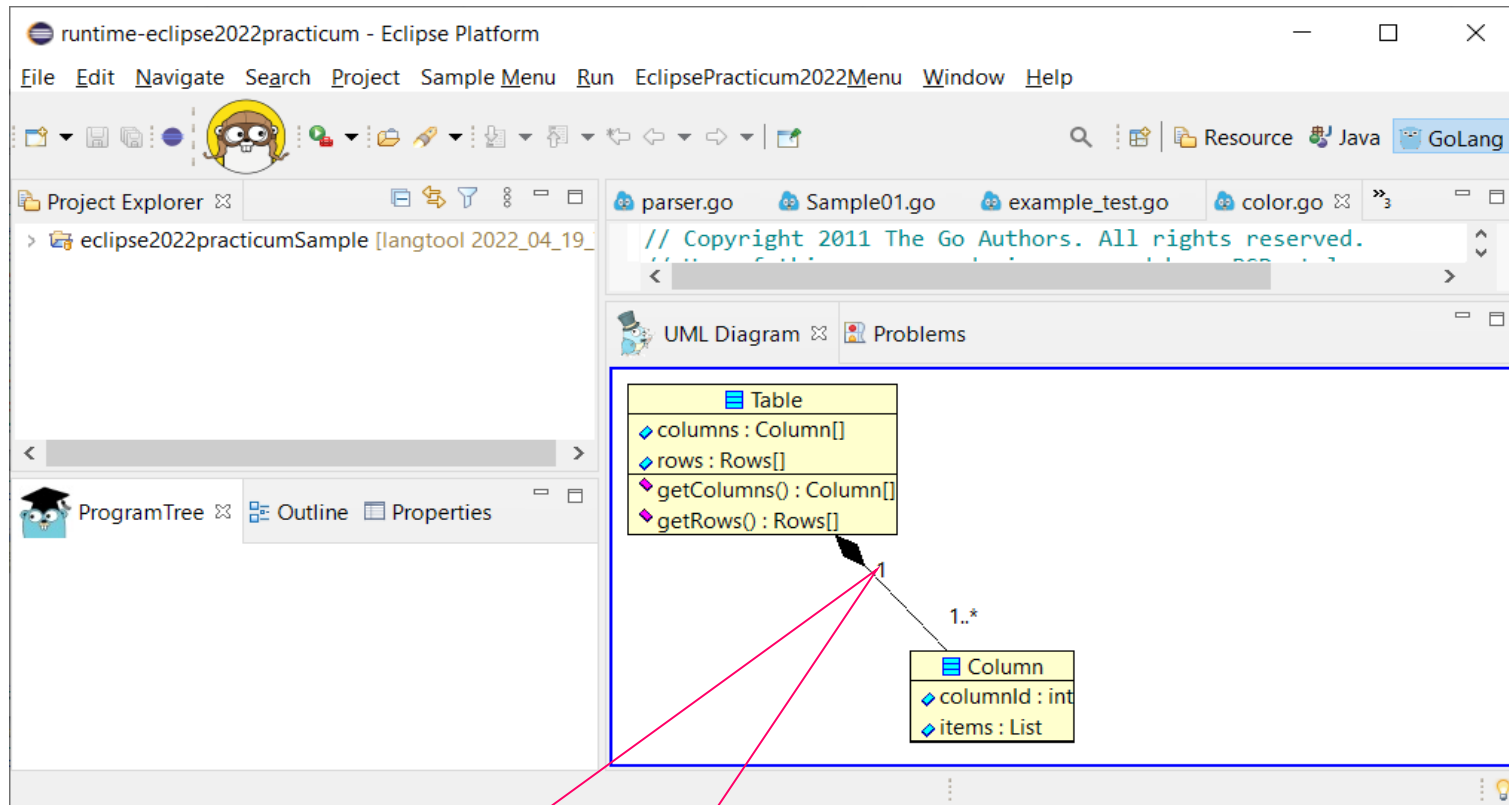
Множественность ассоциации

```
public static void addDiagram(Figure contents) {  
    XYLayout contentsLayout = new XYLayout();  
    contents.setLayoutManager(contentsLayout);  
  
    UMLClassNode classFigure1 = new UMLClassNode ("Table");  
    classFigure1.addAttribute("columns : Column[]");  
    classFigure1.addAttribute("rows : Rows[]");  
    classFigure1.addMethod("getColumns() : Column[]");  
    classFigure1.addMethod("getRows() : Rows[]");  
    contents.add(classFigure1);  
    contentsLayout.setConstraint(classFigure1, new Rectangle(10, 10, -1, -1));  
  
    UMLClassNode classFigure2 = new UMLClassNode ("Column");  
    classFigure2.addAttribute("columnId : int");  
    classFigure2.addAttribute("items : List");  
    contents.add(classFigure2);  
    contentsLayout.setConstraint(classFigure2, new Rectangle(200, 200, -1, -1));  
  
    AssociationEdge as = new AssociationEdge (classFigure1, classFigure2);  
    contents.add(as);  
  
    as.setSourceMultiplicity("1");  
    as.setTargetMultiplicity("1..*");  
}
```

Множественность отношения ассоциации



Множественность отношения ассоциации



Множественность
отношения ассоциации

Создание меток роли для отношения ассоциации

Label targetRole;

```
public void addTargetRole(String role) {  
    ConnectionEndpointLocator relationshipLocator  
                                = new ConnectionEndpointLocator(this, true);  
    relationshipLocator.setUDistance(30);  
    relationshipLocator.setVDistance(-20);  
  
    targetRole = new Label(role);  
    add(targetRole, relationshipLocator);  
}  
  
public void setTargetRole(String role) {  
    targetRole.setText(role);  
}
```

Конечная (target)
точка ребра?

Надпись на ребре

Создание меток роли для отношения ассоциации

Label sourceRole;

```
public void addSourceRole(String role) {  
    ConnectionEndpointLocator relationshipLocator  
                                = new ConnectionEndpointLocator(this, false);  
    relationshipLocator.setUDistance(10);  
    relationshipLocator.setVDistance(-10);  
  
    sourceRole = new Label(role);  
    add(sourceRole, relationshipLocator);  
}  
  
public void setSourceRole(String role) {  
    sourceRole.setText(role);  
}
```

Конечная (target)
точка ребра?

Надпись на ребре

Создание меток роли для отношения

```
public static void addDiagram(Figure contents) {
    XYLayout contentsLayout = new XYLayout();
    contents.setLayoutManager(contentsLayout);

    UMLClassNode classFigure1 = new UMLClassNode ("Table");
    classFigure1.addAttribute("columns : Column[]");
    classFigure1.addAttribute("rows : Rows[]");
    classFigure1.addMethod("getColumns() : Column[]");
    classFigure1.addMethod("getRows() : Rows[]");
    contents.add(classFigure1);
    contentsLayout.setConstraint(classFigure1, new Rectangle(10, 10, -1, -1));

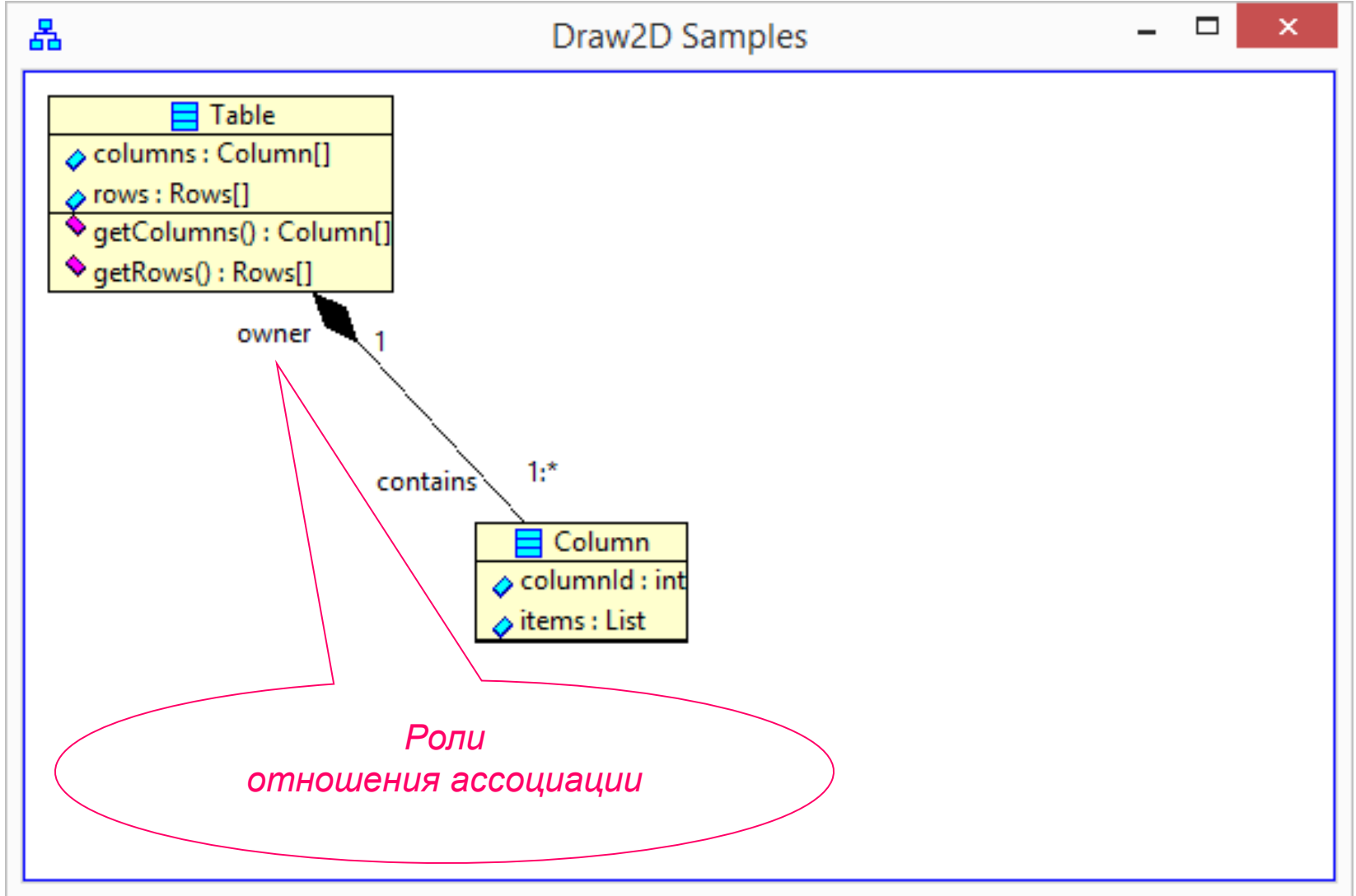
    UMLClassNode classFigure2 = new UMLClassNode ("Column");
    classFigure2.addAttribute("columnId : int");
    classFigure2.addAttribute("items : List");
    contents.add(classFigure2);
    contentsLayout.setConstraint(classFigure2, new Rectangle(200, 200, -1, -1));

    AssociationEdge as = new AssociationEdge (classFigure1, classFigure2);
    contents.add(as);

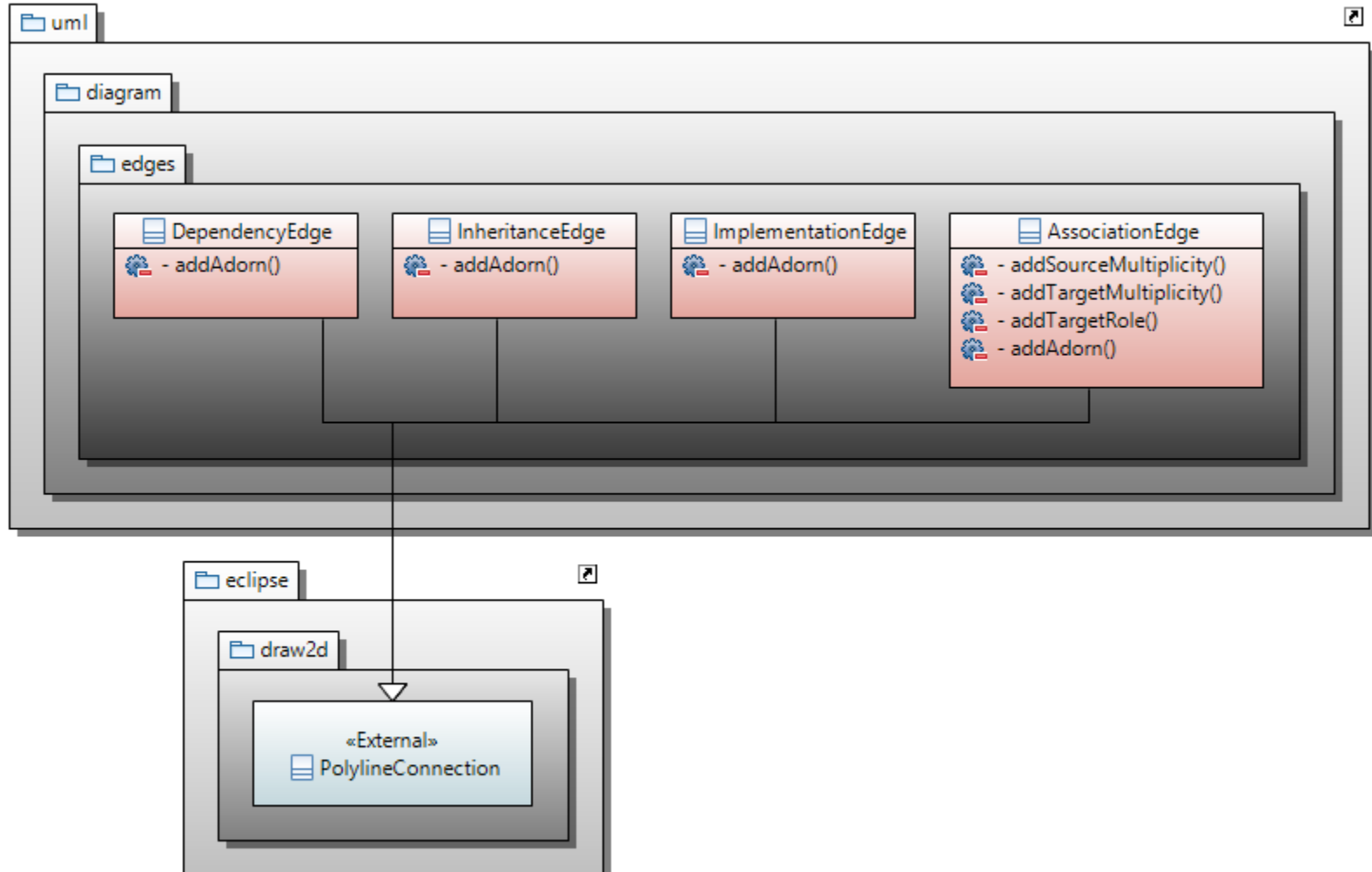
    as.setSourceMultiplicity("1");
    as.setTargetMultiplicity("1..*");

    as.addSourceRole("owner");
    as.addTargetRole("contains");
}
```

Роли отношения ассоциации



Ребра на диаграмме статической структуры языка UML



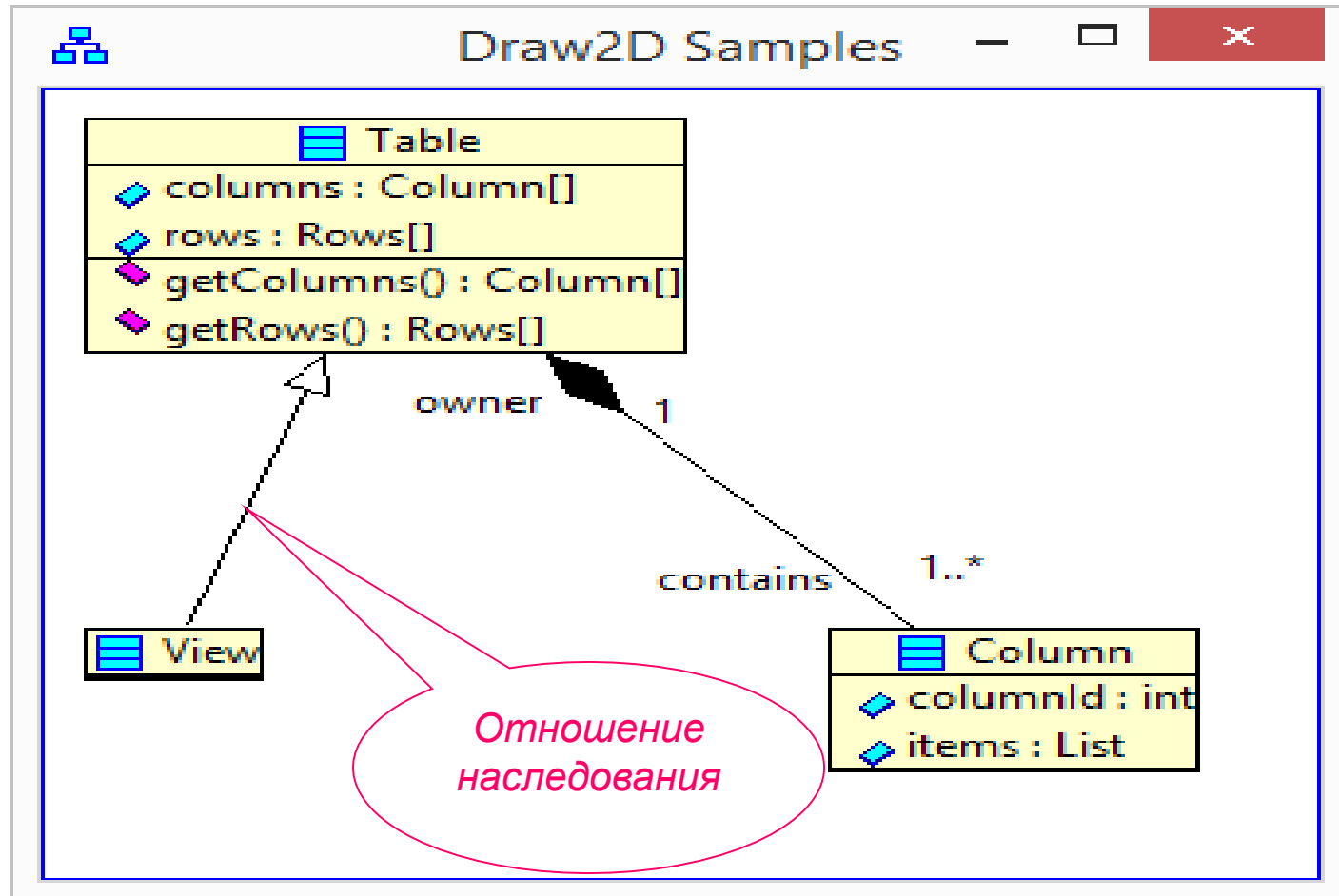
Создание ребра для отношения наследования

```
private static void addDiagram(Figure contents) {  
  
    // ...  
  
    UMLClassNode viewNode = new UMLClassNode("View");  
    contents.add(viewNode);  
  
    Rectangle constraintView = new Rectangle(10, 200, -1, -1);  
    contentsLayout.setConstraint(viewNode, constraintView);  
  
    InheritanceEdge inhittance = new InheritanceEdge(viewNode, tableNode);  
    contents.add(inhittance);  
}
```

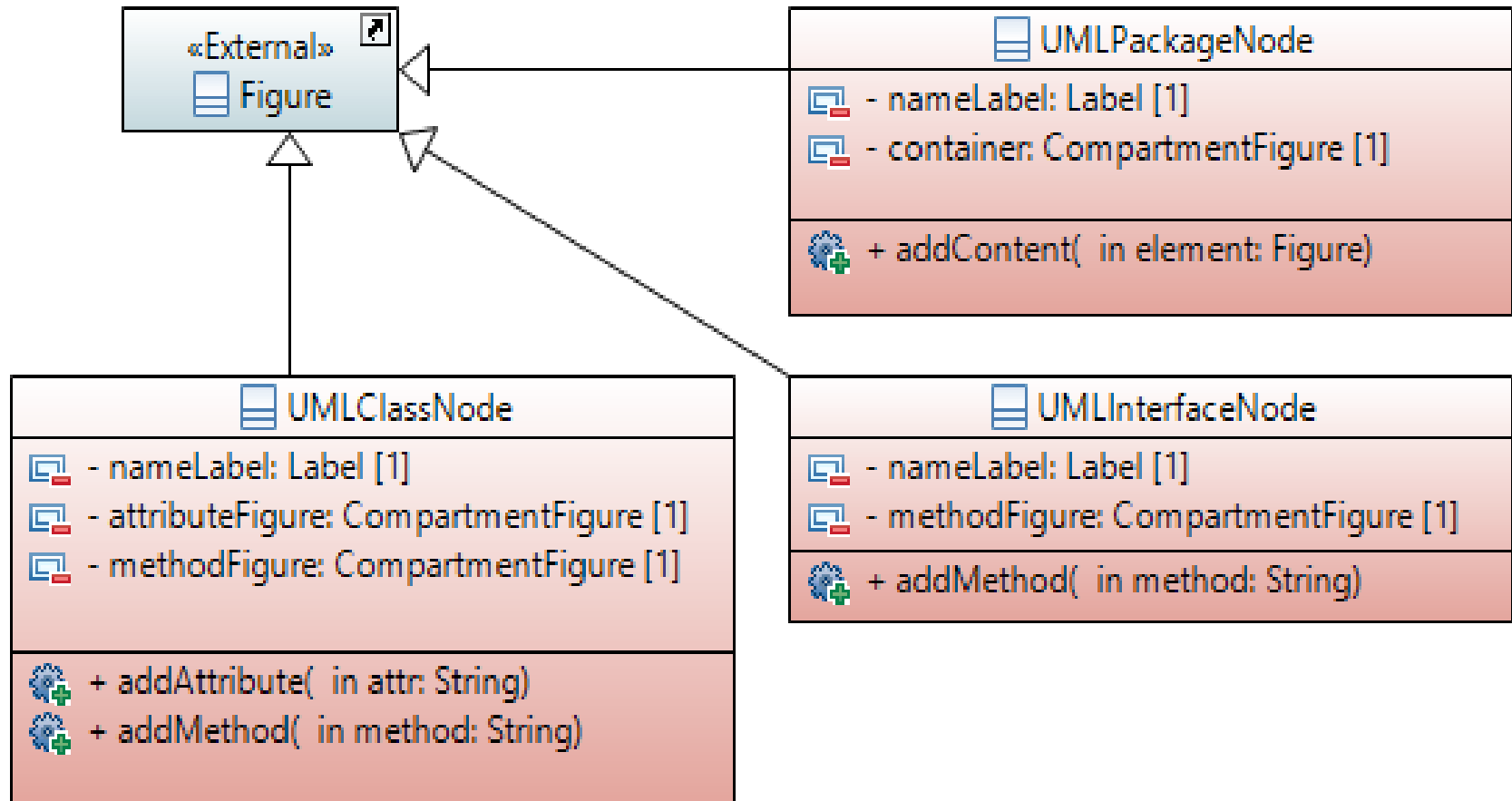
Создание ребра отношения наследования

```
public class InheritanceEdge extends PolylineConnection {  
    public InheritanceEdge (UMLClassNode source, UMLClassNode target) {  
        // ChopboxAnchor - ребра будут направлены в центр узла.  
        ChopboxAnchor sourceAnchor = new ChopboxAnchor(source);  
        ChopboxAnchor targetAnchor = new ChopboxAnchor(target);  
        setSourceAnchor(sourceAnchor);  
        setTargetAnchor(targetAnchor);  
  
        addAdorn();  
    }  
  
    private void addAdorn() {  
        PolygonDecoration decoration = new PolygonDecoration();  
        decoration.setBackgroundColor(ColorConstants.white);  
  
        PointList decorationPointList = new PointList();  
        decorationPointList.addPoint(0, 0);  
        decorationPointList.addPoint(-2, 2);  
        decorationPointList.addPoint(-2, -2);  
        decoration.setTemplate(decorationPointList);  
        setTargetDecoration(decoration);  
    }  
}
```

Отношение наследования



Узлы UML-диаграммы. Узлы с секциями



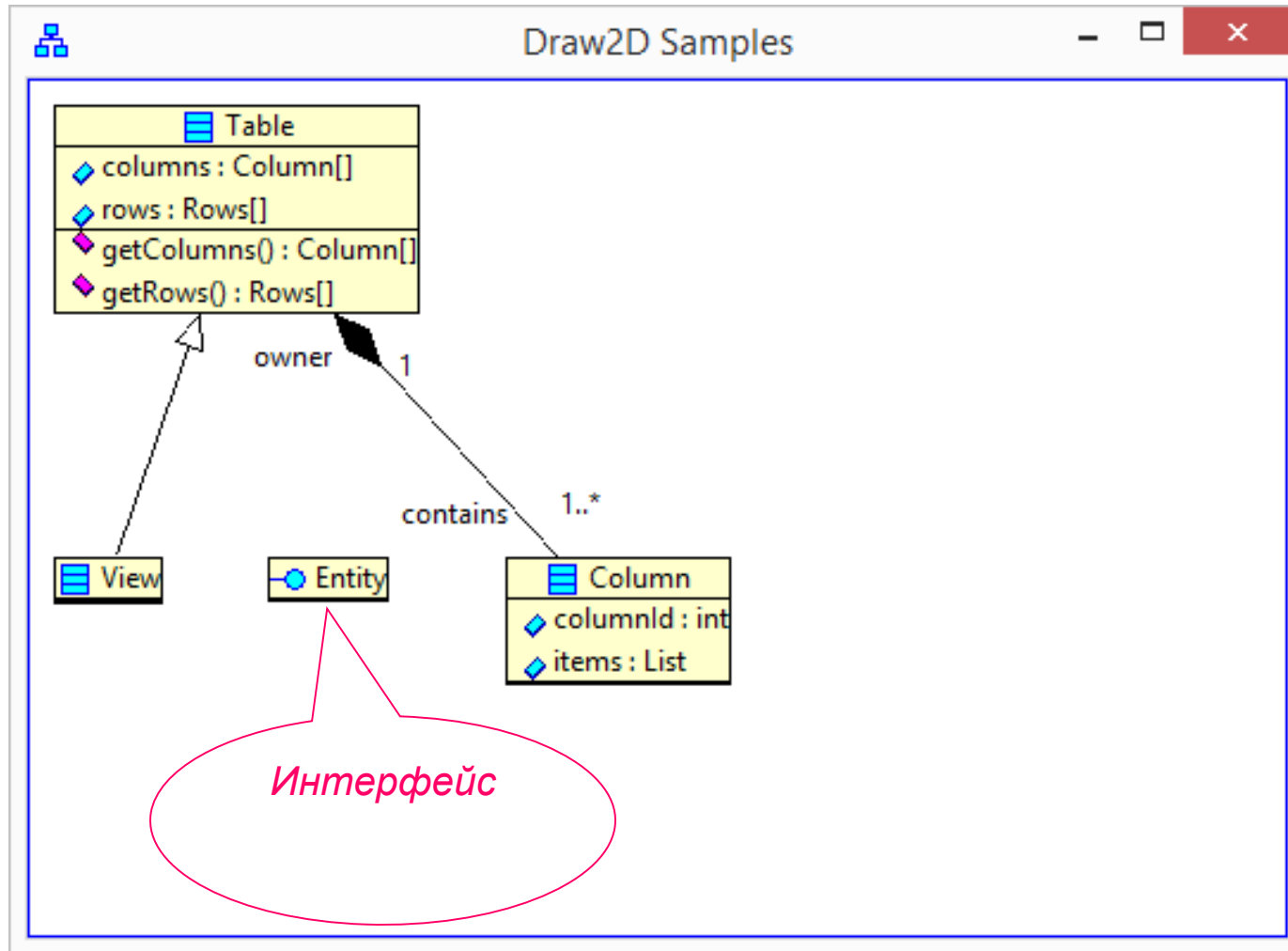
Узел диаграммы классов – интерфейс класса

```
private static void addDiagram(Figure contents) {  
  
    // ...  
  
    UMLInterfaceNode entityNode = new UMLInterfaceNode ("Entity");  
    contents.add(entityNode);  
  
    Rectangle constraintEntity = new Rectangle(100, 200, -1, -1);  
    contentsLayout.setConstraint(entityNode, constraintEntity);  
}
```

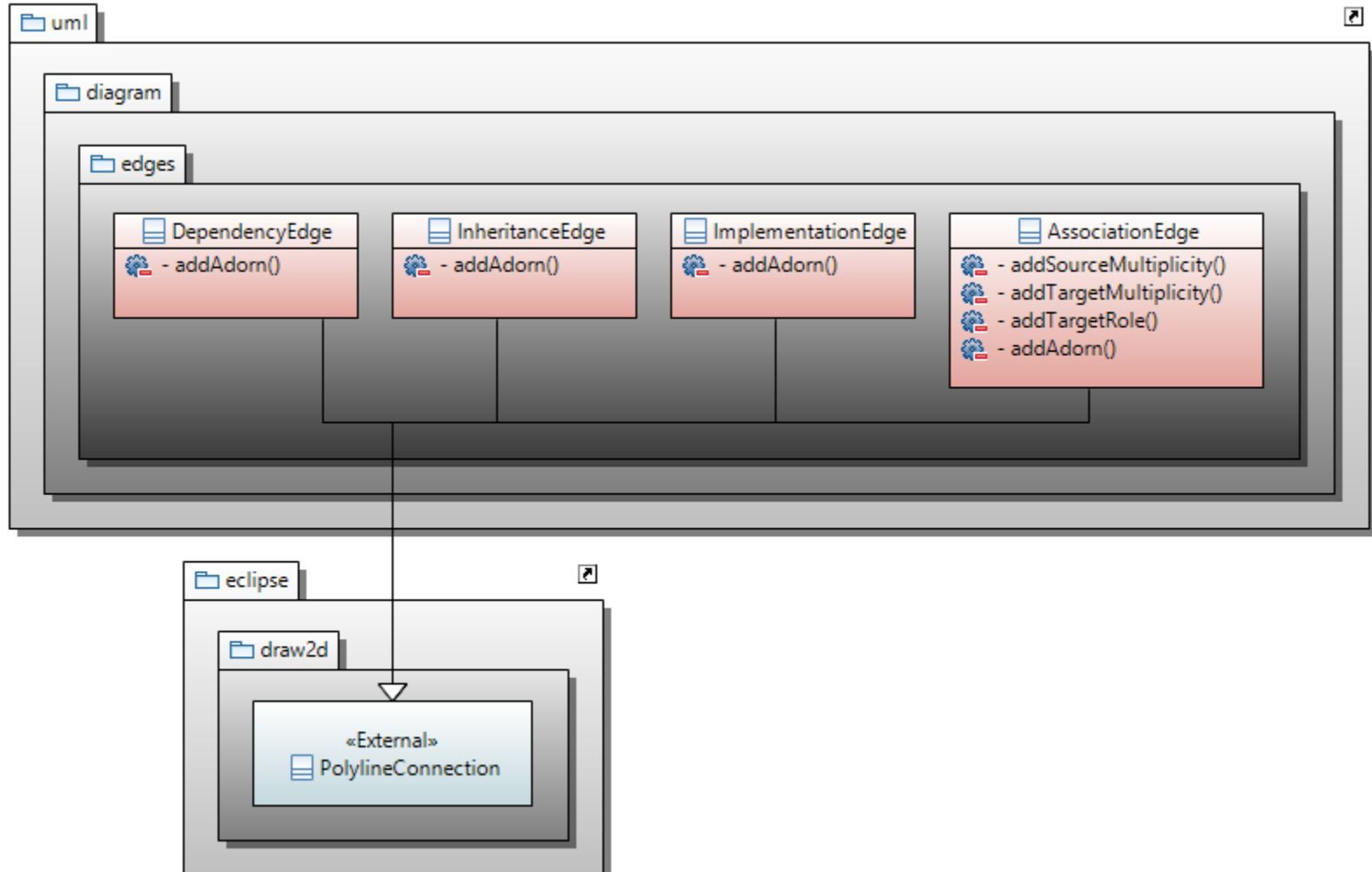
Узел диаграммы классов - интерфейс класса

```
public class UMLInterfaceNode extends Figure {  
    public static Color interfaceColor = new Color(null, 255, 255, 206);  
  
    private CompartmentFigure methodFigure = new CompartmentFigure();  
  
    public UMLInterfaceNode (String name) {  
        ToolbarLayout layout = new ToolbarLayout();  
        setLayoutManager(layout);  
  
        setBackgroundColor(interfaceColor);  
        setBorder(new LineBorder(ColorConstants.black, 1));  
        setOpaque(true); // Фигура не прозрачна.  
  
        add(new Label(name, UMLImages.interfacelImage));  
        add(methodFigure);  
    }  
  
    public void addMethod(String method) {  
        methodFigure.add(new Label(method, UMLImages.methodImage));  
    }  
}
```

Изображение интерфейса класса



Ребра на диаграмме статической структуры языка UML



Создание ребра для отношения реализации

```
private static void addDiagram(Figure contents) {
```

```
// ...
```

```
UMLInterfaceNode entityNode = new UMLInterfaceNode ("Entity");  
contents.add(entityNode);
```

```
Rectangle constraintEntity = new Rectangle(100, 200, -1, -1);  
contentsLayout.setConstraint(entityNode, constraintEntity);
```

```
ImplementationEdge implement = new ImplementationEdge(viewNode, entityNode);  
contents.add(implement);
```

```
}
```

```
}
```

Отношение реализации классом интерфейса

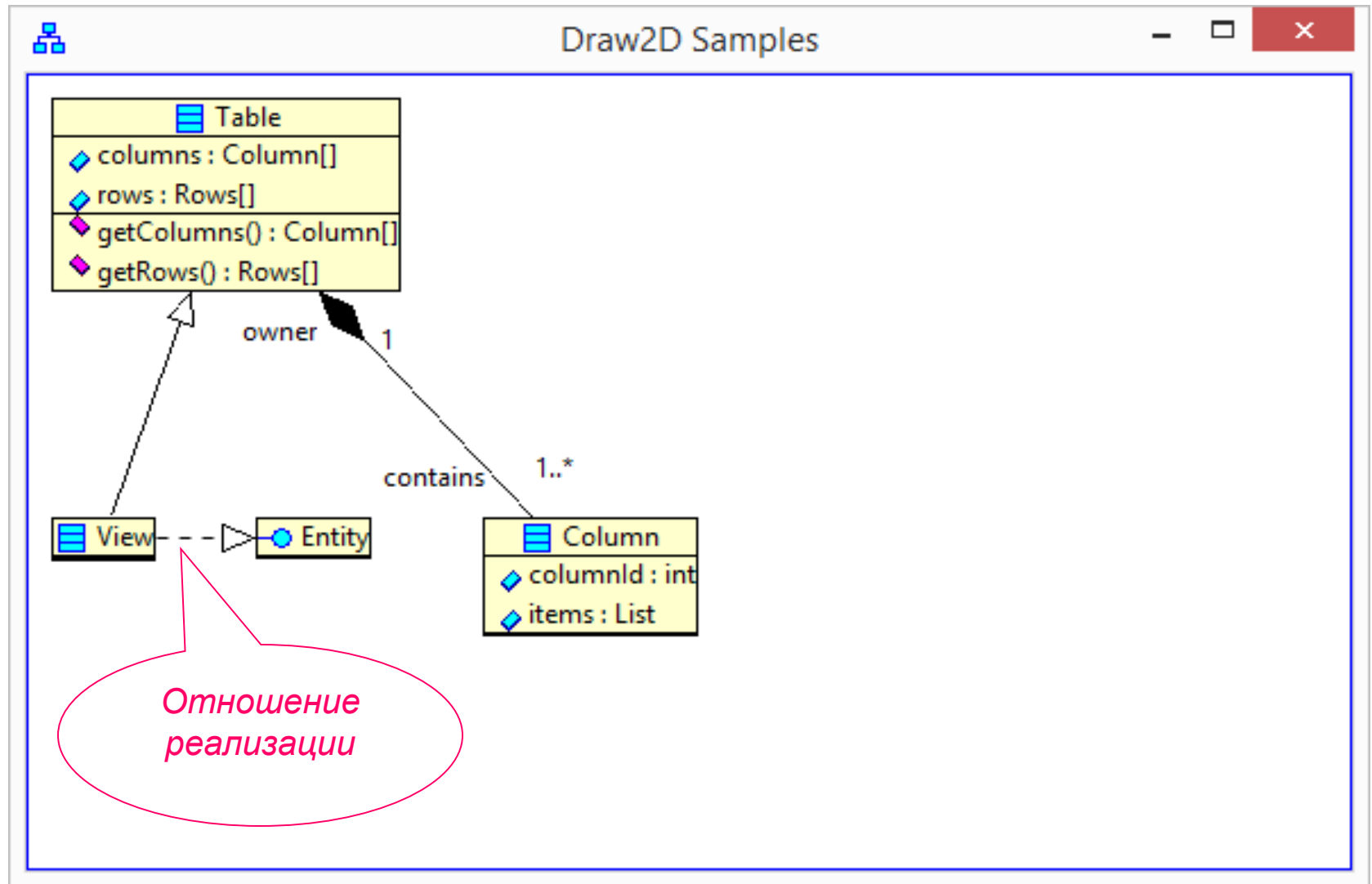
```
public class ImplementationEdge extends PolylineConnection {  
  
    public ImplementationEdge(UMLClassNode source, UMLInterfaceNode target) {  
        // ChopboxAnchor - ребра будут направлены в центр узла.  
        ChopboxAnchor sourceAnchor = new ChopboxAnchor(source);  
        ChopboxAnchor targetAnchor = new ChopboxAnchor(target);  
        setSourceAnchor(sourceAnchor);  
        setTargetAnchor(targetAnchor);  
  
        setLineStyle(SWT.LINE_CUSTOM); // штриховая линия отношения  
        setAntialias(SWT.ON);  
        setLineDash(new float[] { 5.0f, 5.0f });  
  
        addAdorn(); // стрелка отношения  
    }  
  
    // ...  
}
```

Отношение реализации классом интерфейса

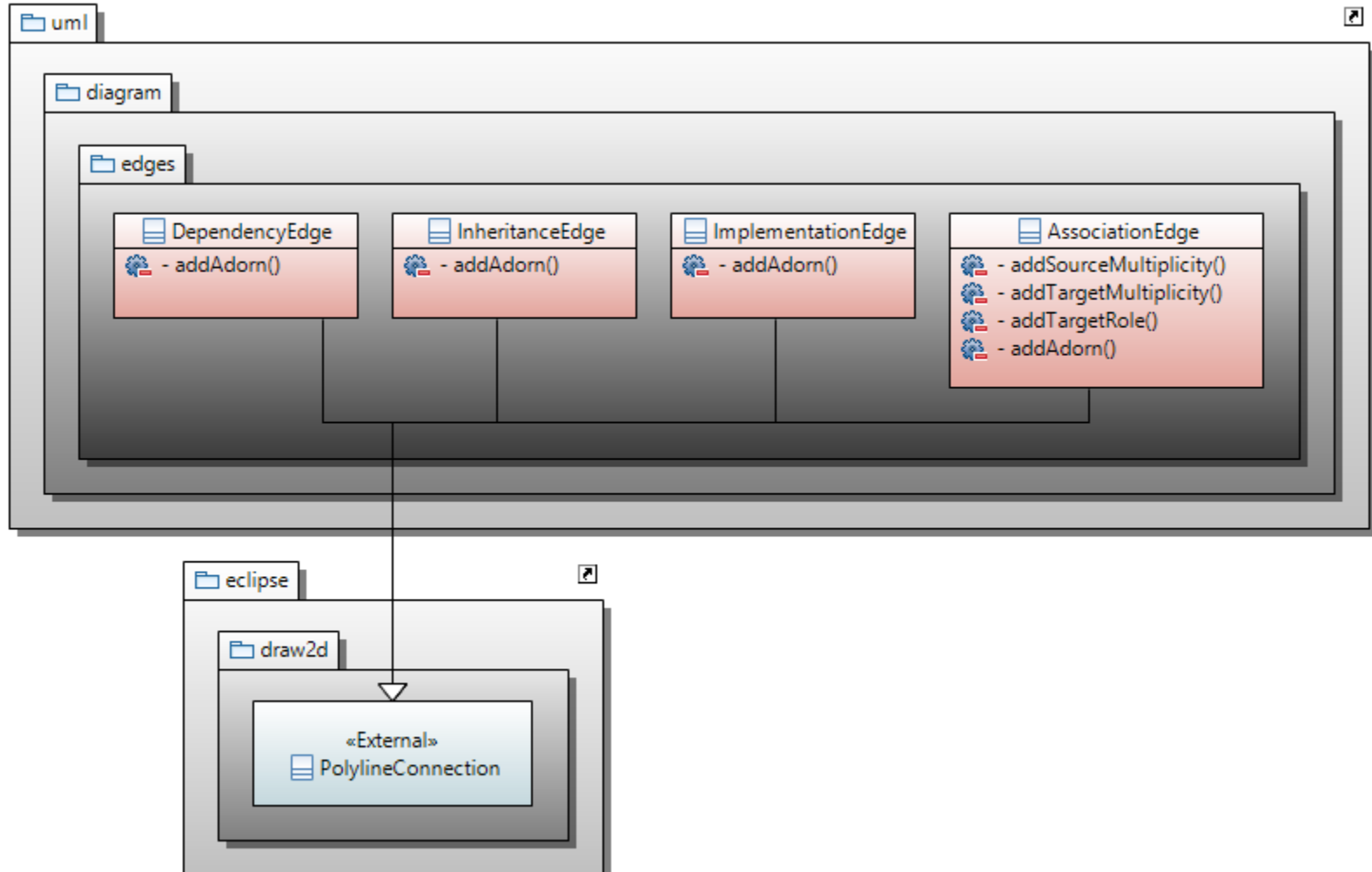
```
// ...
```

```
private void addAdorn() {  
    PolygonDecoration decoration = new PolygonDecoration();  
    decoration.setBackgroundColor(ColorConstants.white);  
  
    PointList decorationPointList = new PointList();  
    decorationPointList.addPoint(-2, 2);  
    decorationPointList.addPoint( 0, 0);  
    decorationPointList.addPoint(-2, -2);  
    decoration.setTemplate(decorationPointList);  
  
    setTargetDecoration(decoration);  
}  
}
```

Изображение отношения реализации



Ребра на диаграмме статической структуры языка UML



Отношение зависимости

произвольных элементов диаграмм

```
public class DependencyEdge extends PolylineConnection {  
  
    public DependencyEdge(Figure source, Figure target) {  
        // ChopboxAnchor - ребра будут направлены в центр узла.  
        ChopboxAnchor sourceAnchor = new ChopboxAnchor(source);  
        ChopboxAnchor targetAnchor = new ChopboxAnchor(target);  
        setSourceAnchor(sourceAnchor);  
        setTargetAnchor(targetAnchor);  
  
        setLineStyle(SWT.LINE_CUSTOM);  
        setAntialias(SWT.ON);  
        setLineDash(new float[] { 5.0f, 5.0f });  
  
        addAdorn();  
    }  
    // ...  
}
```

Создание ребра для отношения зависимости (2)

```
// ...
```

```
private void addAdorn() {  
    PolylineDecoration decoration = new PolylineDecoration();  
  
    PointList decorationPointList = new PointList();  
    decorationPointList.addPoint(-2, 2);  
    decorationPointList.addPoint( 0, 0);  
    decorationPointList.addPoint(-2, -2);  
    decoration.setTemplate(decorationPointList);  
    setTargetDecoration(decoration);  
}  
}
```

Узлы UML-диаграммы. Узлы с секциями

