Язык программирования Java

Библиотека **Eclipse Standard Widget Toolkit**

Разработка интерфейса пользователя
на примере программы «Шахматный блокнот».

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

Назначение SWT

- **Standard Widget Toolkit**: Инструмент для разработки интерфейса пользователя
  - Эффективный
  - Переносимый
  - Низкий уровень реализации: есть доступ к возможностям операционной системы

- Сайт проекта: eclipse.org/swt

SWT. Основные классы библиотеки: Класс **Display**

- Представляет *рабочее место* (*workstation*) мониторы, клавиатуру, мышку
- Отвечает за распределение событий в *цикле событий (event loop)*
- Содержит список *окон верхнего уровня* (*Shells*)
- Содержит список *мониторов* (*Monitors*)

SWT. Основные классы библиотеки: Класс **Shell**

- Представляет *окно* на экране
- Это корень дерева состоящего из:
  - *Composites* (составной элемент)
  - *Controls* (управляющий элемент)
- **Shell** – потомок класса **Widget**

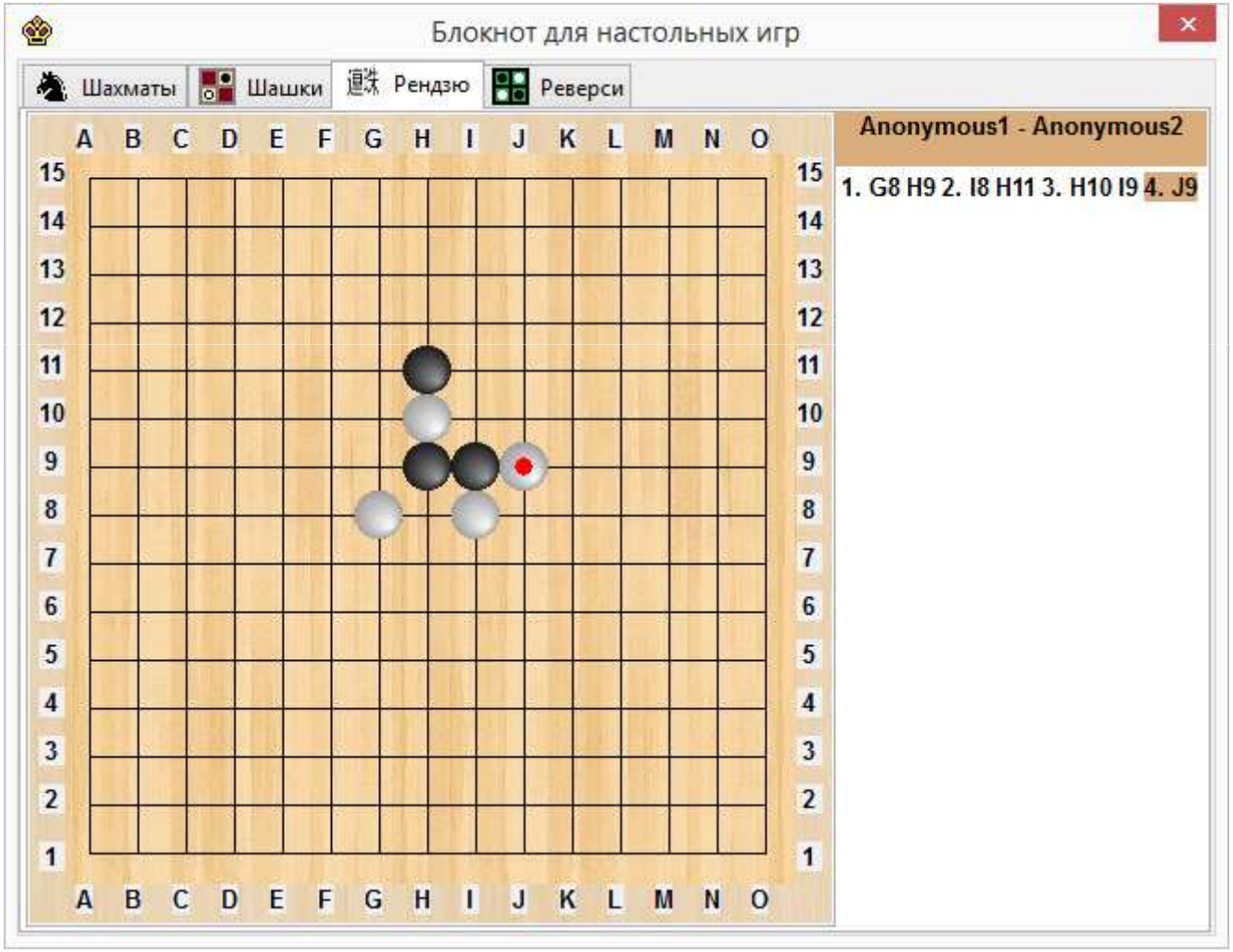
SWT. Вид класса **Shell** на экране.
Окно с заголовком *Chess Notebook*

Блокнот игр. Шахматы. Подсветка последнего хода

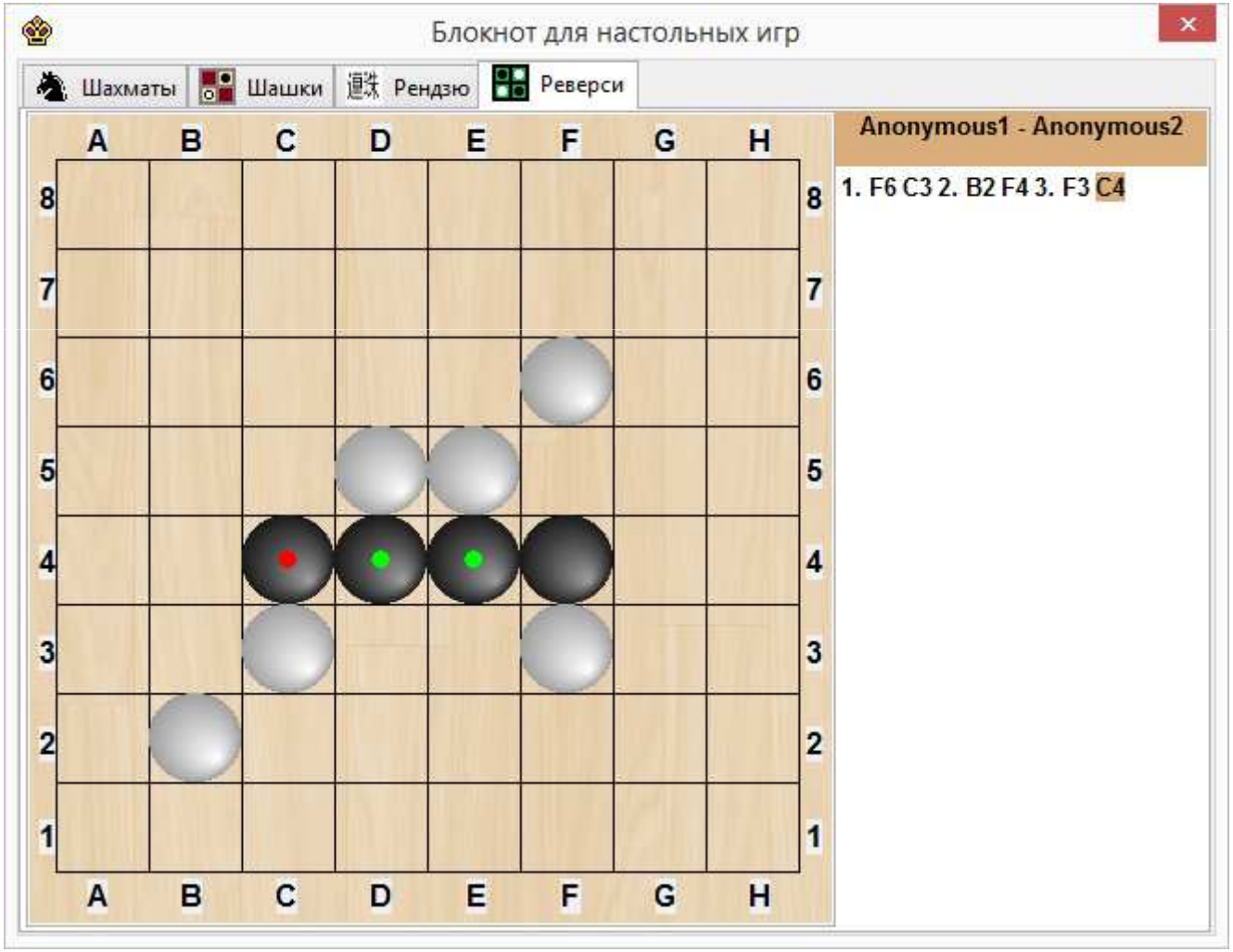
МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017

Блокнот игр. Рендзю. Подсветка последнего хода

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017



Блокнот игр. Реверси. Подсветка последнего хода



SWT. Основные классы библиотеки: Класс **Composite**

- **Composite** (составной элемент) **-** управляющий элемент (control) который может состоять их других составных элементов и управляющий элементов

- **Composite** – потомок класса **Widget**

SWT. Основные классы библиотеки: Класс **Control**

- **Control** (управляющий элемент) – представляет *легковесный* (*heavyweight*) элемент операционной системы
- Примеры управляющих элементов: Button, Label, Text, Tree, Shell, Composite, …
- **Control** – потомок класса **Widget**

SWT. Цикл событий (event loop)

- В **SWT**, цикл событий должен быть явно закодирован в приложении
- Цикл событий постоянно *читает и распределяет события* интерфейса пользователя поступающие из операционной системы и «отдает» CPU когда событий нет.
- Цикл событий завершается когда завершается приложение. Обычно когда закрывается окно.

SWT. Пример цикла событий

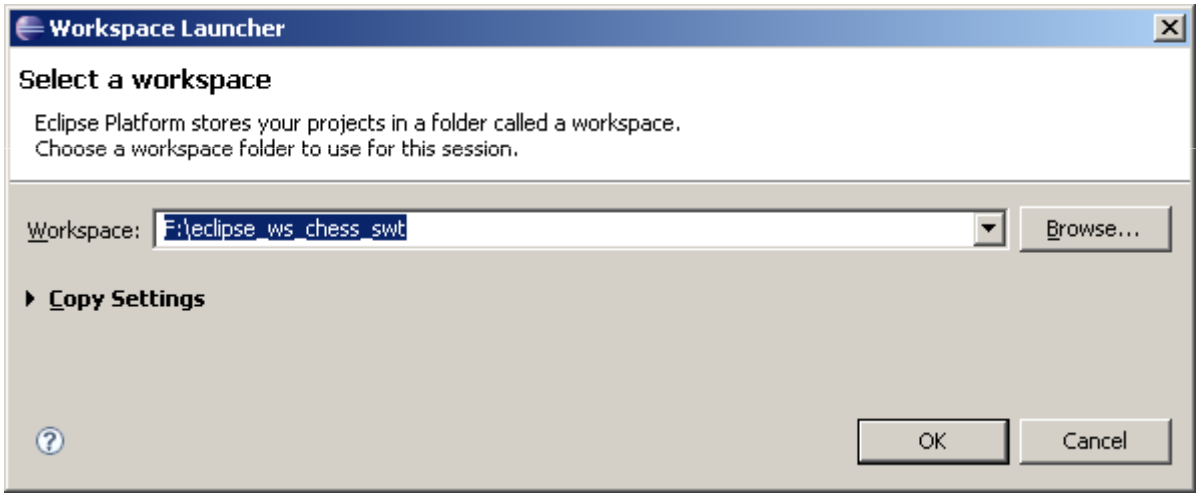

```
while(!shell.isDisposed()){

   if(!display.readAndDispatch())
      display.sleep ();
}
```

Шахматы. Минимальное «приложение»

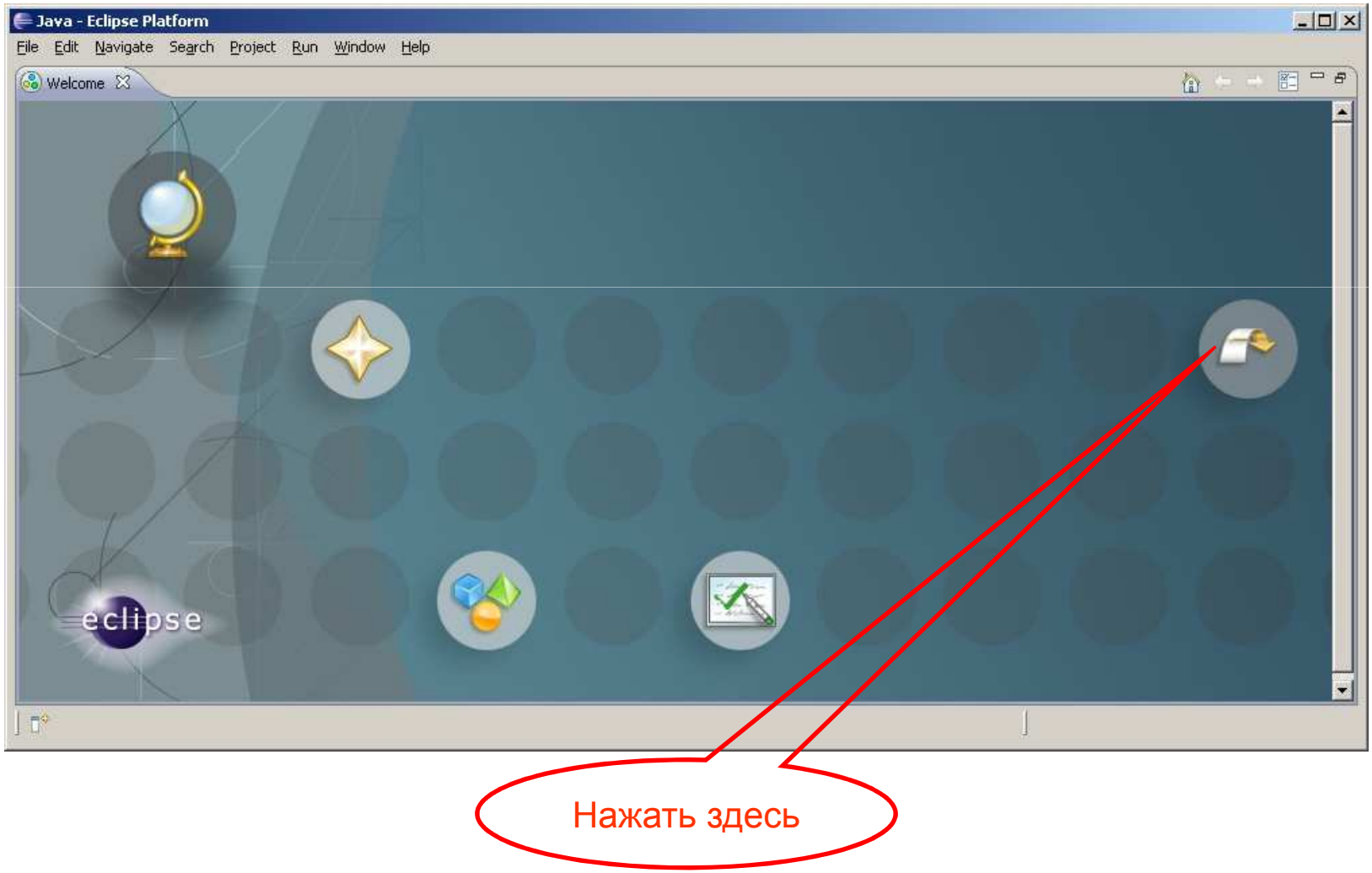

```
import org.eclipse.swt.widgets.*;

public class Chess {
    public static void main(String[] args) {
        final Display display = new Display();

        final Shell shell = new Shell(display);
        shell.setSize(600, 500);
        shell.setText("Chess Notebook");
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    } // main
} // class Chess
```

Шахматы. Создание рабочего места

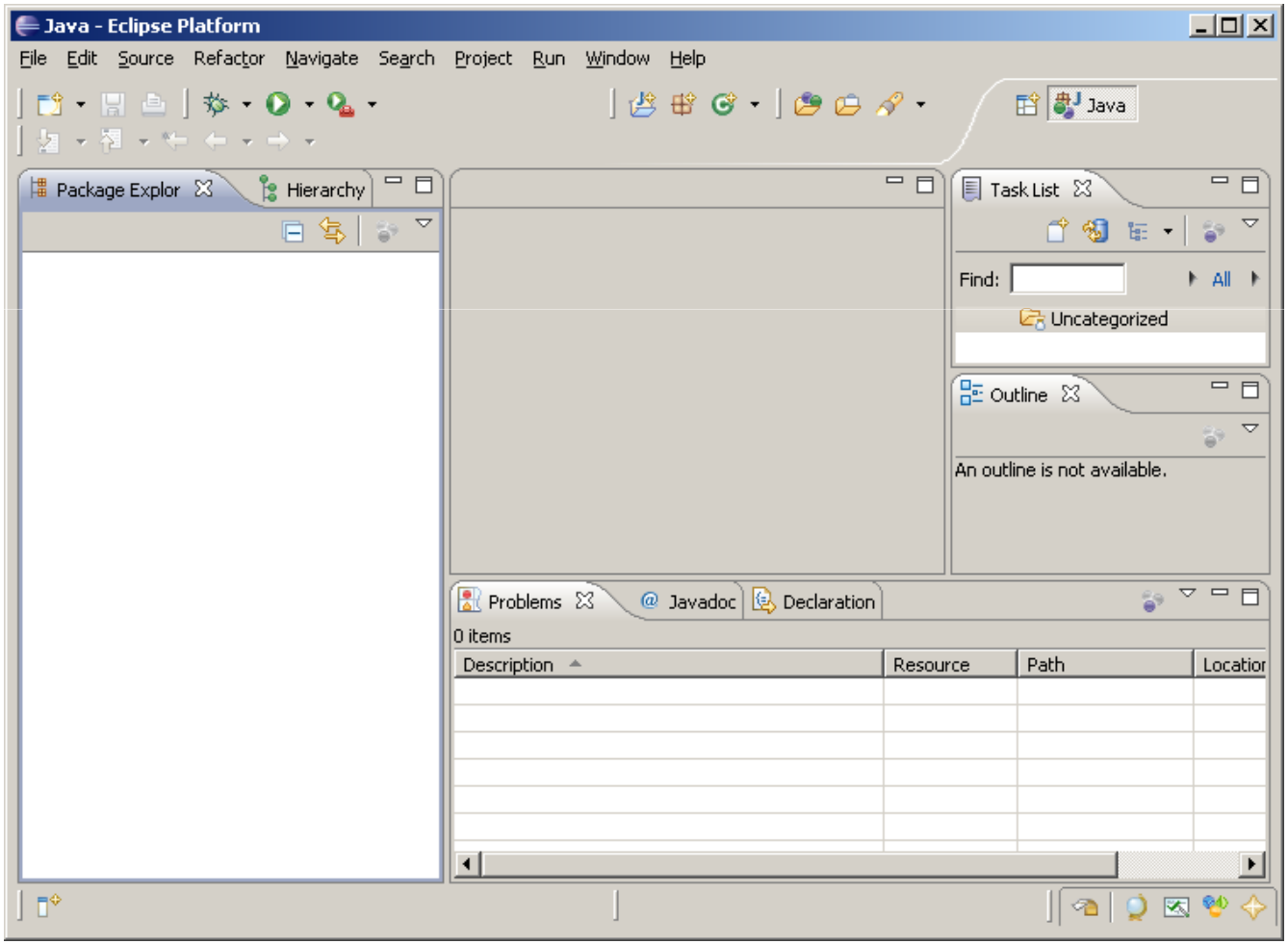
- **Files | Switch Workspace | Other …**



Шахматы. Рабочее место. Wellcome



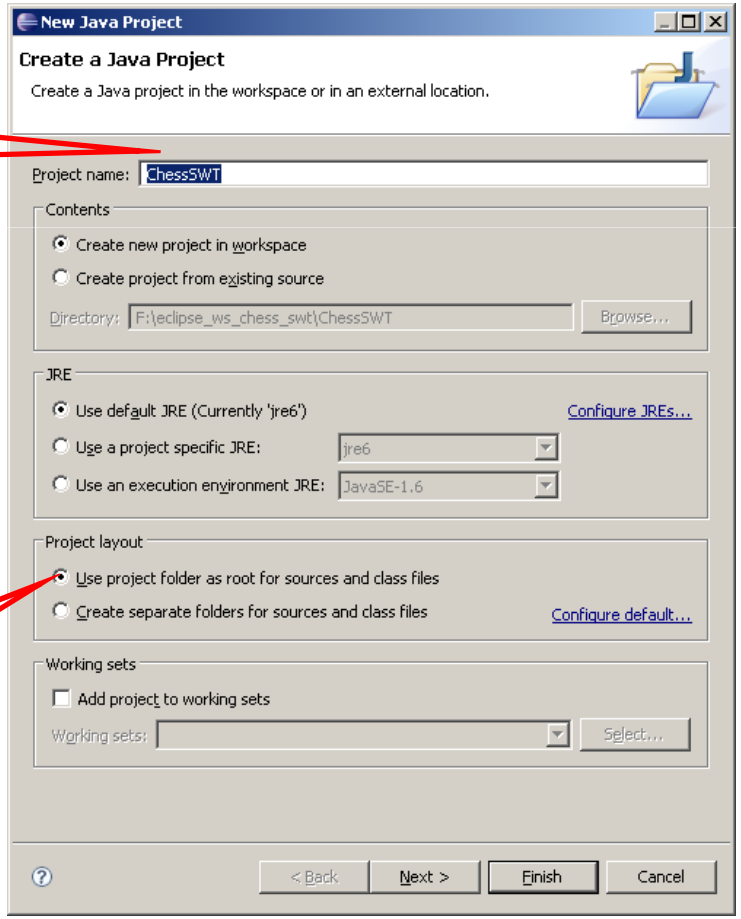
Шахматы. Workbench (Станок ??)



Шахматы. Создание проекта

- **Files | New | Java Project …**

Ввести имя проекта ***ChessSWT***



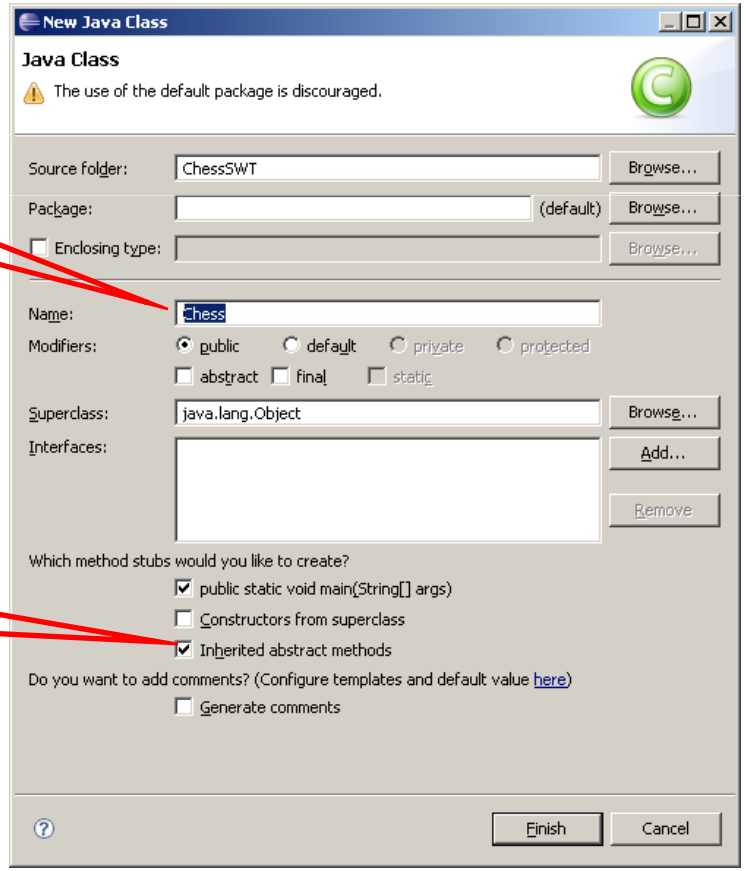
Текстовые и бинарные файлы в одной папке

Шахматы. Создание “главного” класса

- **Files | New | Class ...**

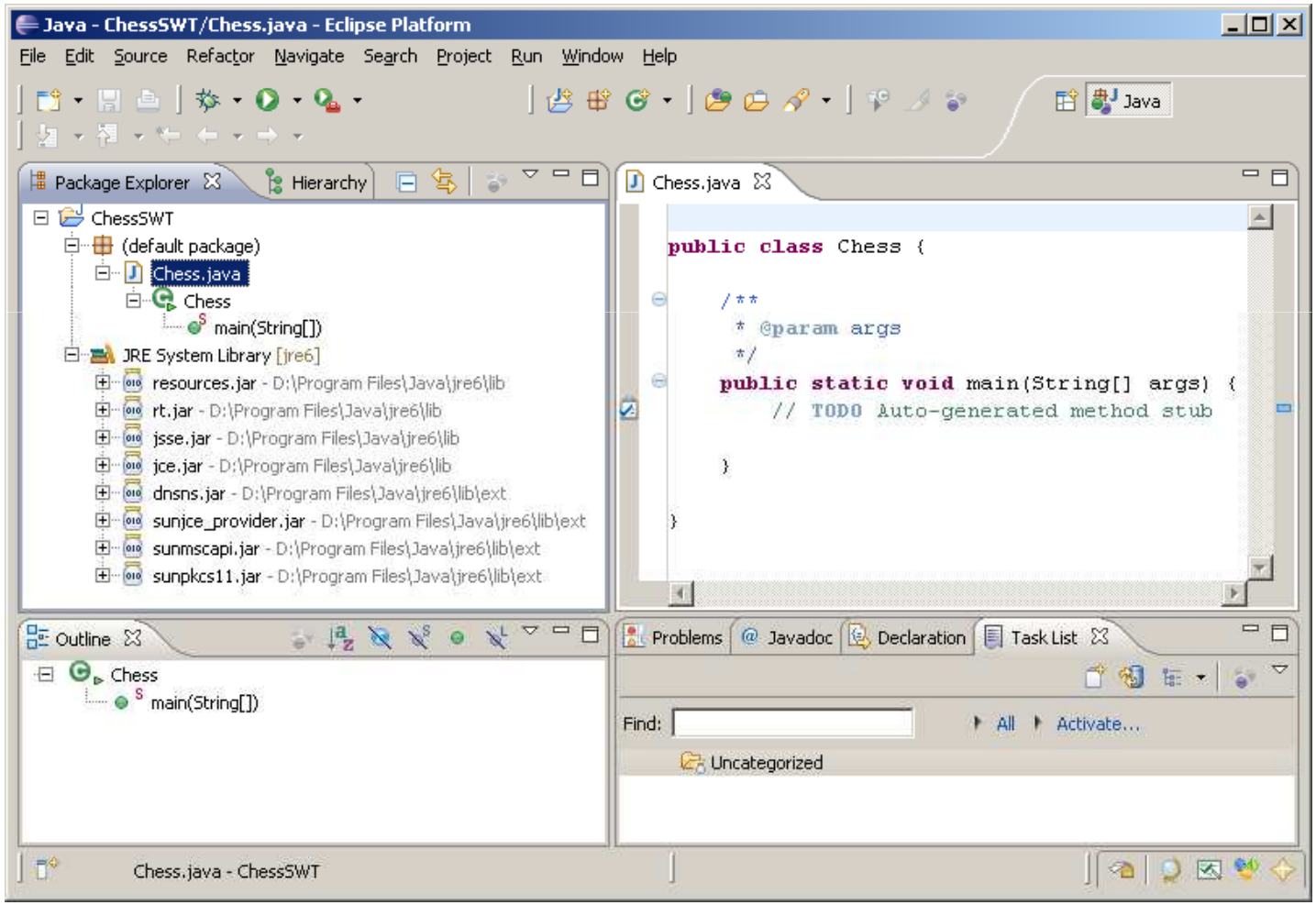
Ввести имя класса ***Chess***

Попросить создать метод main()



Шахматы. Сгенерированный «главный» класс **Chess**

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017

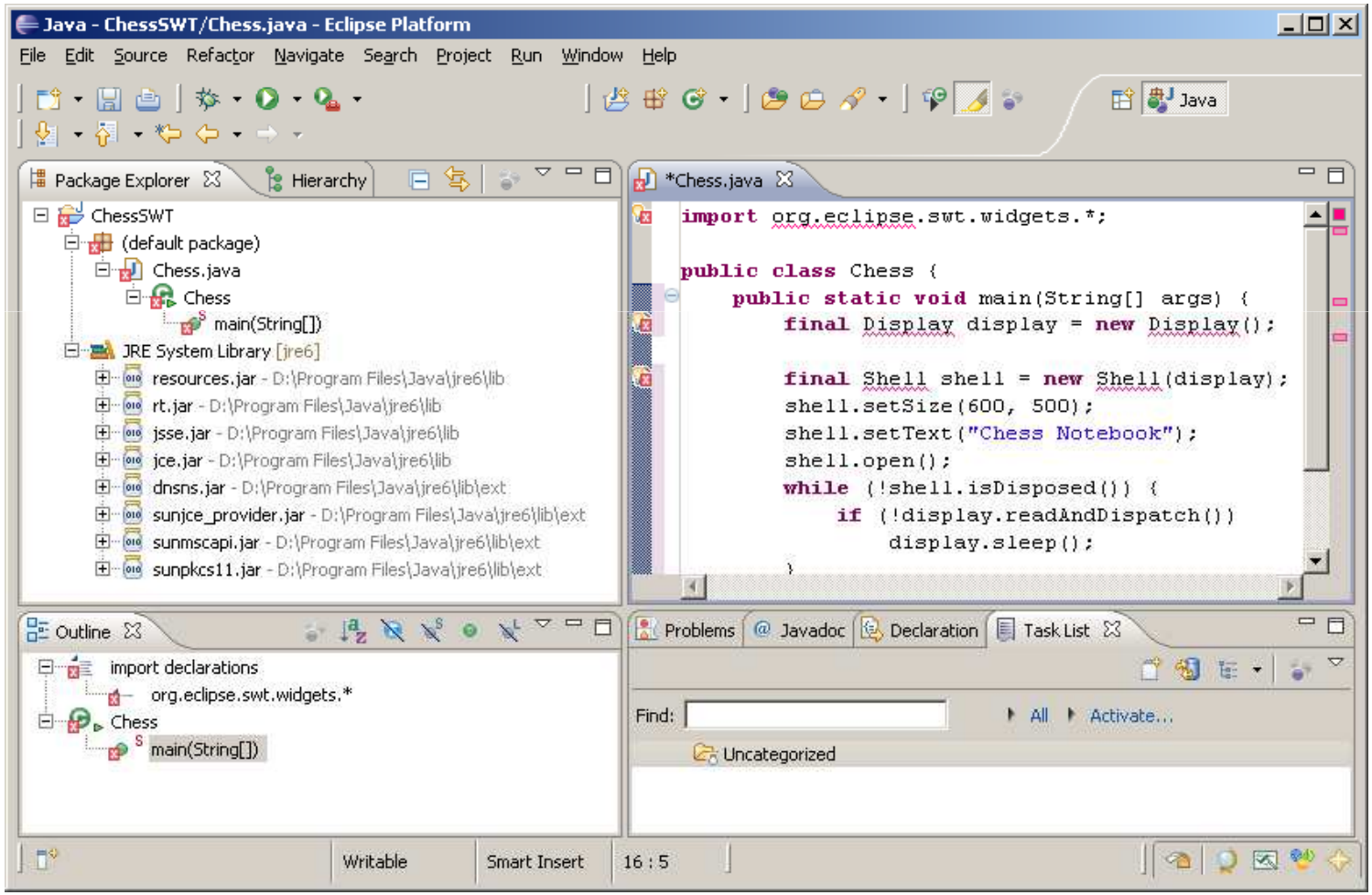


Шахматы. Минимальное приложение


```
import org.eclipse.swt.widgets.*;

public class Chess {
    public static void main(String[] args) {
        final Display display = new Display();

        final Shell shell = new Shell(display);
        shell.setSize(600, 500);
        shell.setText("Chess Notebook");
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    } // main
} // class Chess
```

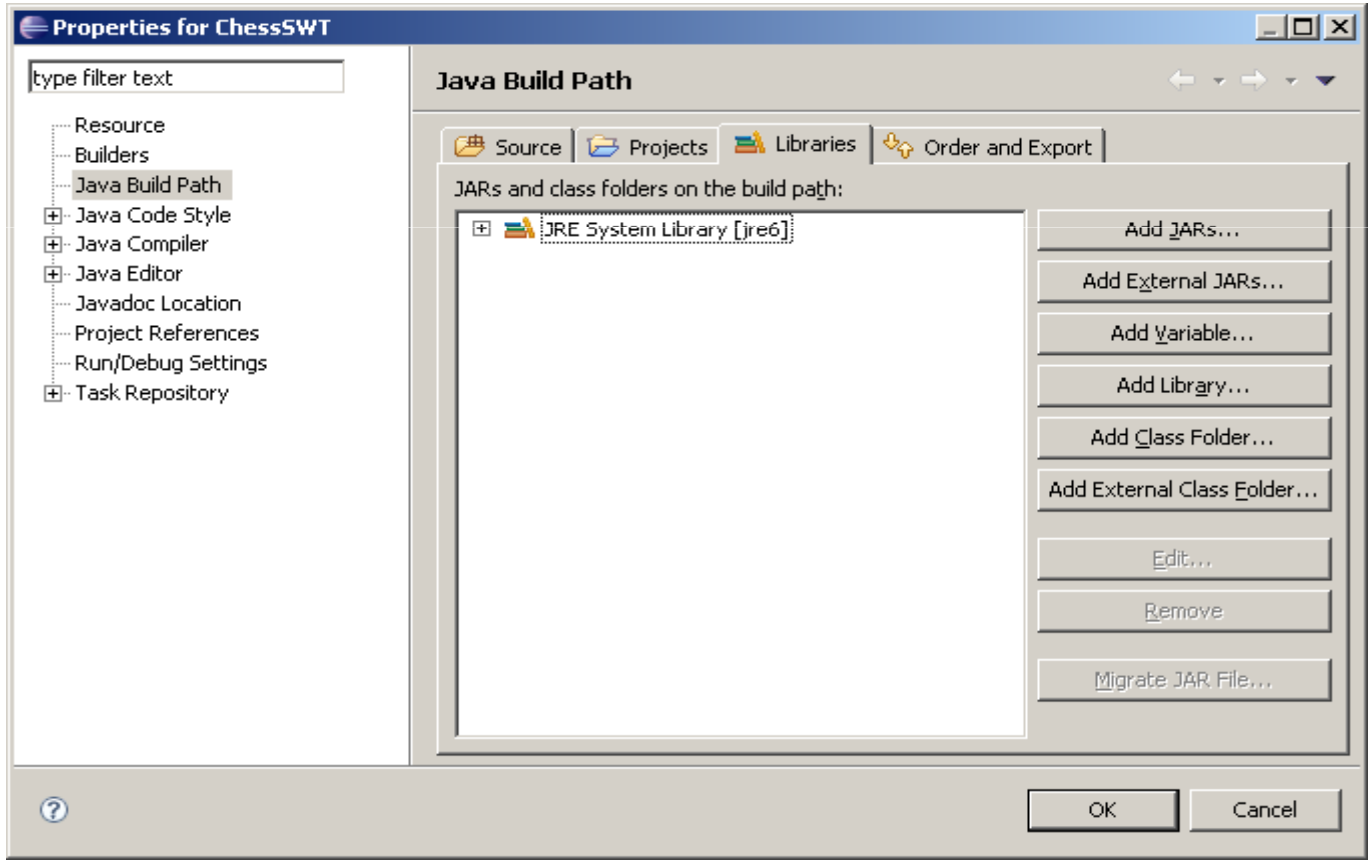
Шахматы. «Минимальные» шахматы в окне редактора

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017



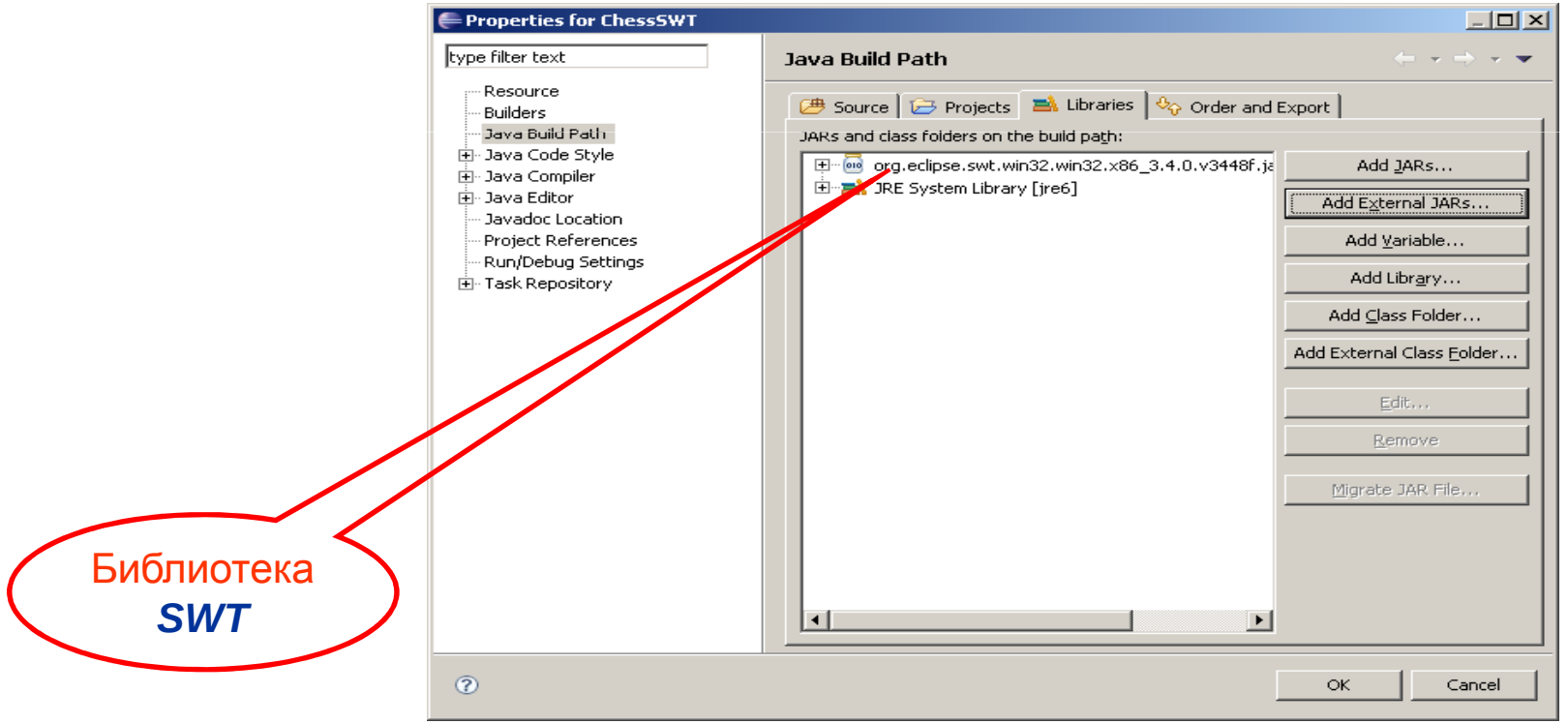
Шахматы. Подключение библиотеки Standard Window Toolkit

- **Project | Properties | Project Build Path …**



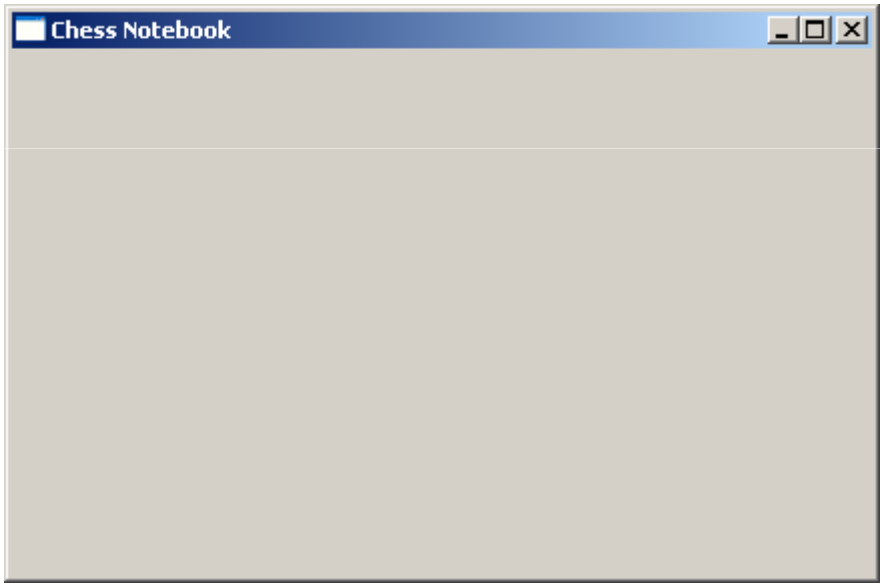
Шахматы. Поиск библиотеки SWT папке расширений (plugins) Eclipse

- **Add External JARs … | c:/eclipse/plugins/ | org.eclipse.swt.win32.win32.x86_3.4.0.v3448f.jar**



Шахматы. «Минимальные» шахматы на экране

- **Нажать “запуск в отладочном режиме” F11**



Пиктограмма приложения


```
import org.eclipse.swt.widgets.*;

public class Chess {
    public static void main(String[] args) {
        final Display display = new Display();
        ChessImages.load(display);
        final Shell shell = new Shell(display);
        shell.setSize(600, 500);
        shell.setText("Chess Notebook");
        shell.setImage(ChessImages.iconChessNotebook);
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    } // main
} // class Chess
```

Шахматы. Создание класса *ChessImages* в режиме «исправления ошибок»

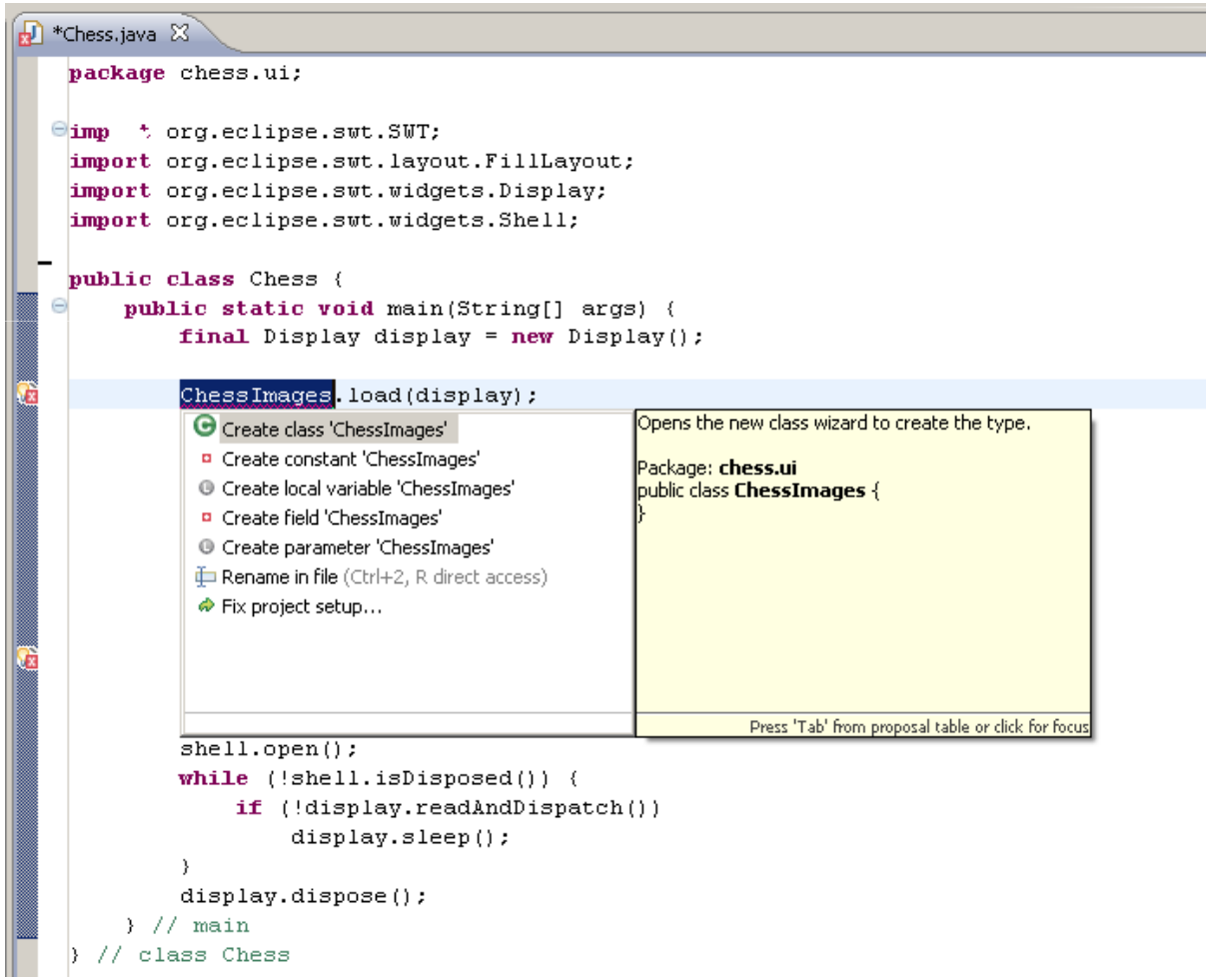
МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017

Нажатие мышью
или
CTRL+1

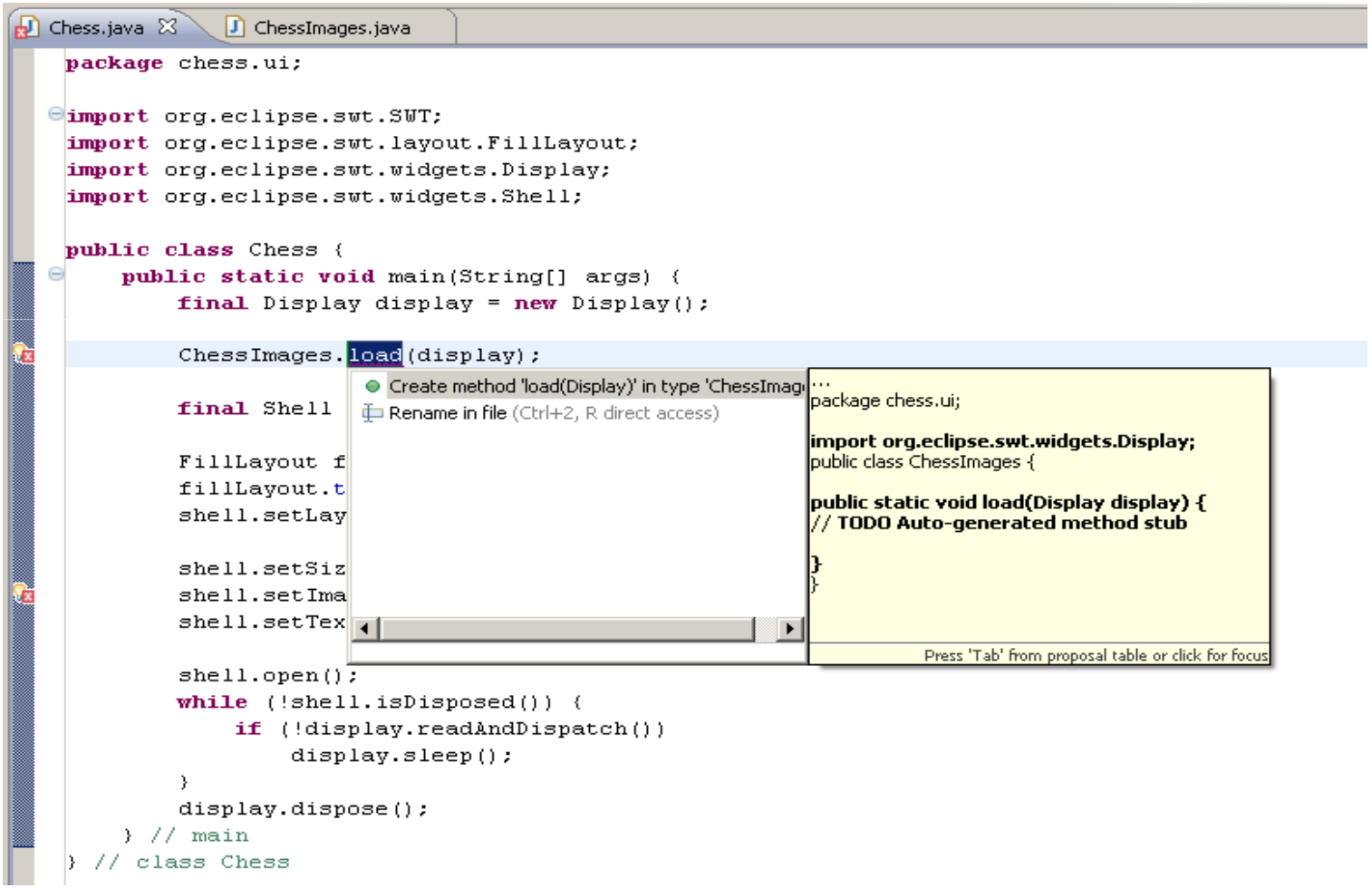
*Chess.java

```
package chess.ui;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class Chess {
    public static void main(String[] args) {
        final Display display = new Display();

        ChessImages.load(display);

        final Shell shell = new Shell(display);

        FillLayout fillLayout = new FillLayout();
        fillLayout.type = SWT.VERTICAL;
        shell.setLayout(fillLayout);

        shell.setSize(600, 500);
        shell.setImage(ChessImages.iconChessNotebook);
        shell.setText("Chess Notebook");

        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    } // main
} // class Chess
```

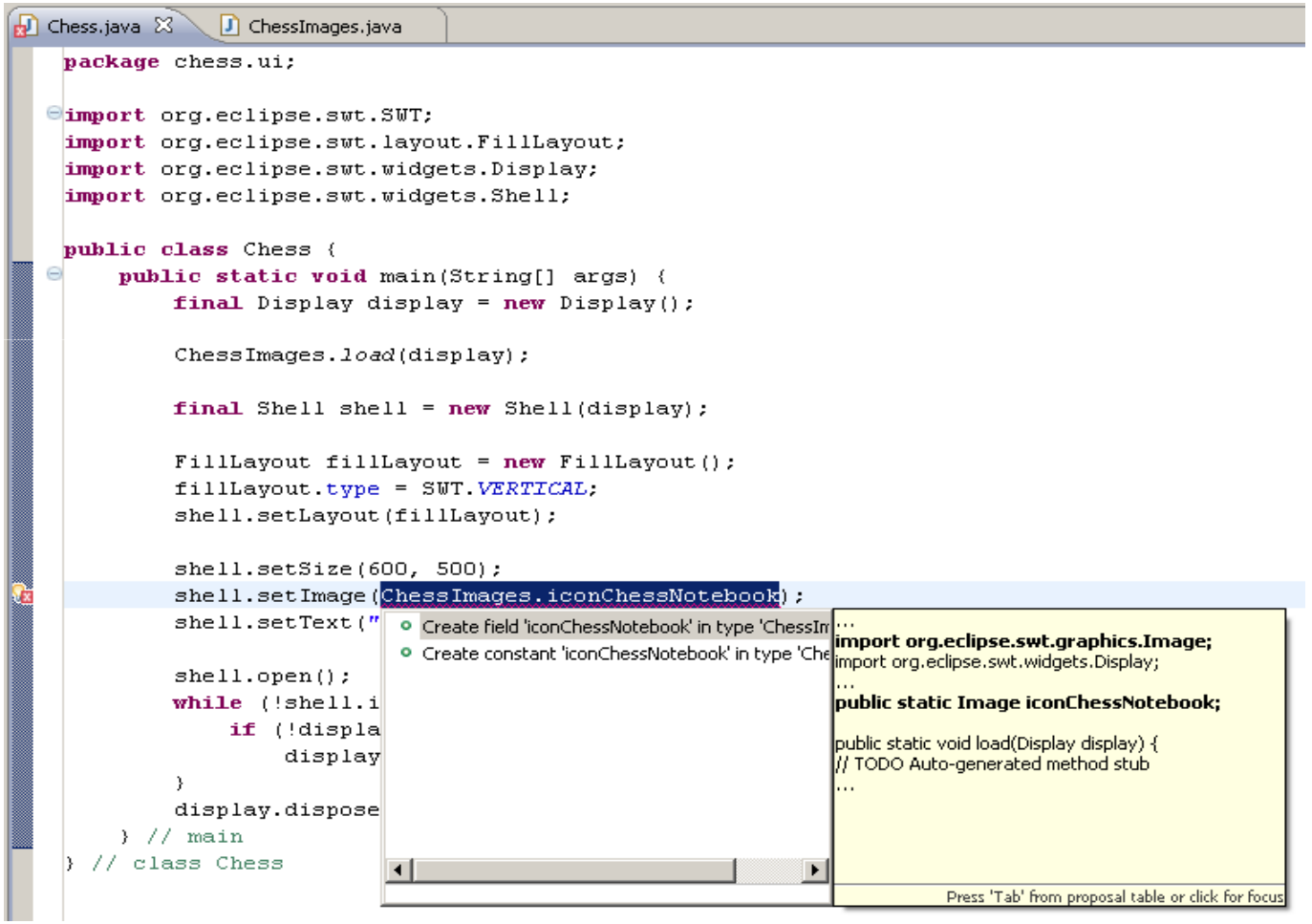
Шахматы. Варианты «исправления ошибки» предлагаемые в среде Eclipse



Шахматы. Создание метода *load* в классе *ChessImages*



Создание поля *iconChessNotebook* в классе *ChessImages*

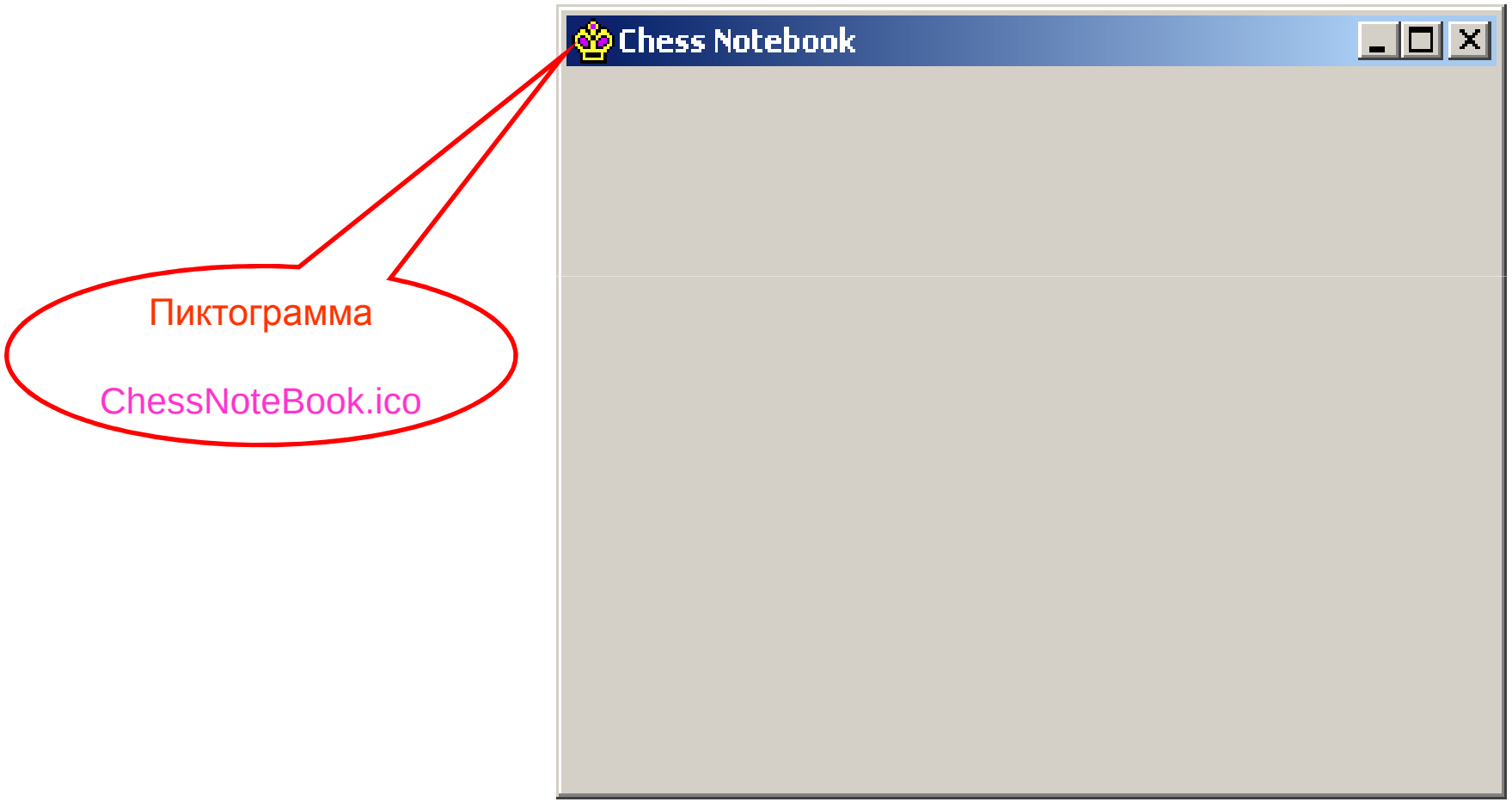


Сгенерированный класс *ChessImages*


```
package chess.ui;

import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Display;

public class ChessImages {

        public static Image iconChessNotebook;

        public static void load(Display display) {
            iconChessNotebook = new Image(display,
              ChessImages.class.getResourceAsStream("ChessNoteBook.ico"));
        }
}
```

- **Файл** ChessNoteBook.ico **должен находится в той же папке, что и файл ChessImages.class**

Шахматы. Блокнот с пиктограммой



SWT. Пакеты библиотеки

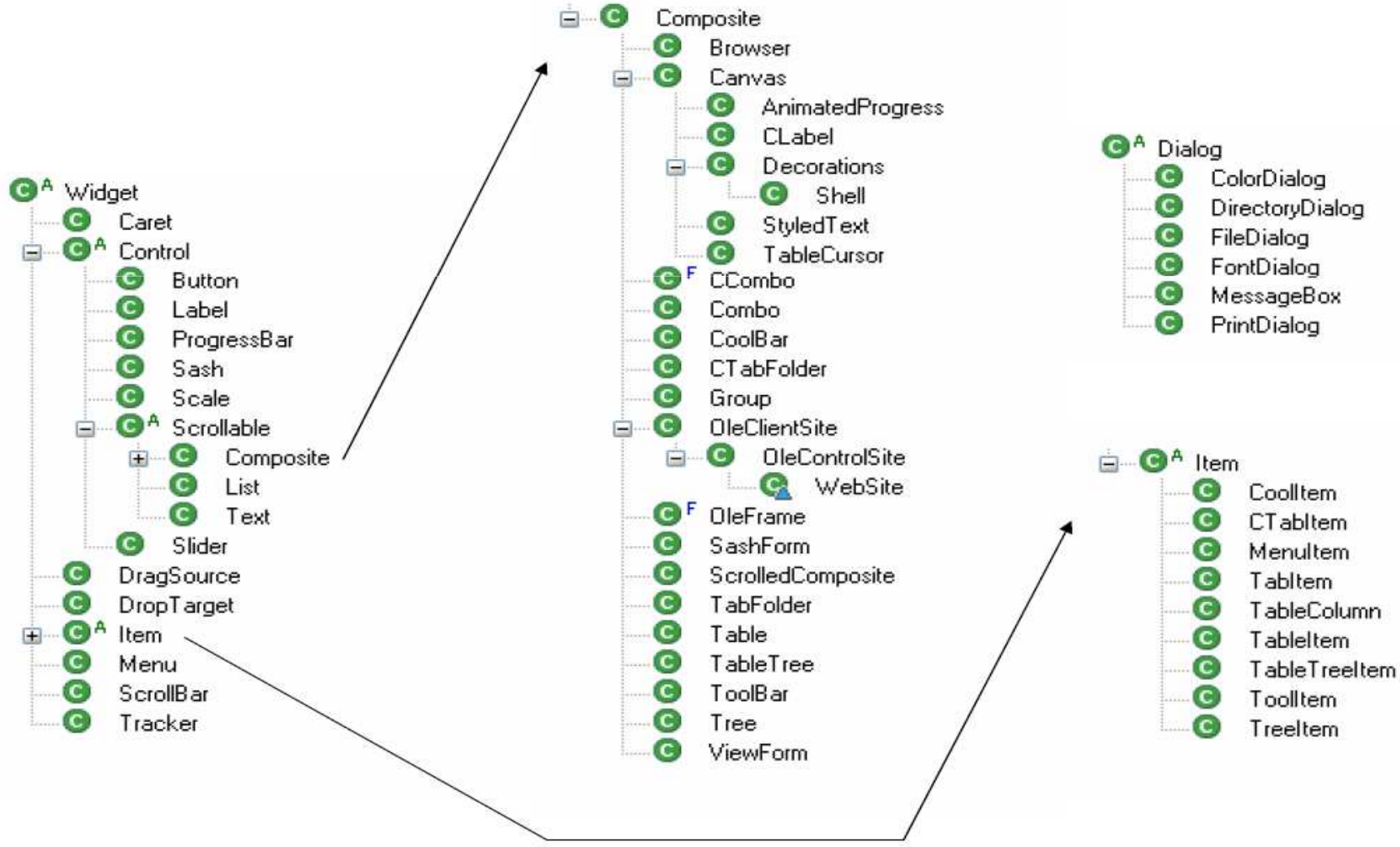
- **Пакеты библиотеки SWT на языке Java**

  - org.eclipse.swt
  - org.eclipse.swt.widgets
  - org.eclipse.swt.graphics
  - org.eclipse.swt.events
  - org.eclipse.swt.layout
  - org.eclipse.swt.dnd
  - org.eclipse.swt.printing
  - org.eclipse.swt.program
  - org.eclipse.swt.accessibility
  - org.eclipse.swt.custom
  - org.eclipse.swt.browser
  - org.eclipse.swt.awt
  - org.eclipse.swt.internal

Класс SWT.

- Содержит все константы библиотеки
  - SWT.PUSH, SWT.RADIO
  - SWT.Selection
- Содержит универсальные методы
  - getPlatform()
  - getVersion()
  - error()

Иерархия управляющих элементов (widgets).



Конструкторы и стили управляющих элементов.

- *Управляющий элемент (widget)* всегда имеет предка
- Вид типичного конструктора:
  Widget(Composite parent, int style)
- Стили задаются с помощью констант из класса *SWT*
- Примеры:
  - **new** Label(shell, SWT.NONE);
  - Button push = **new** Button(shell, SWT.PUSH);
  - Button radio = **new** Button(parent, SWT.RADIO);
  - Text text = **new** Text(group, SWT.SINGLE | SWT.BORDER);

- Исключение. Класс *Shell* всегда имеет предка *Shell* или *Display*
  - Shell shell = **new** Shell(display, SWT.SHELL_TRIM);
  - Shell dialog = **new** Shell(shell, SWT.DIALOG_TRIM);

Класс Widget.

- Абстрактный суперкласс для всех элементов интерфейса пользователя
- Создается с помощью конструкторов (без фабрик)
- При создании занимает ресурсы операционной системы
- Ресурсы освобождаются программно с помощью метода dispose()
- Уведомляет слушателей когда происходят с этим управляющим элементом происходят события
- Позволяет хранить специфичные для приложения данные
  - setData(Object)
  - setData(String, Object)
- Событие
  - Dispose

Освобождение ресурсов графики и управляющих элементов.

- Вы должны явно освобождать объекты потребляющие ресурсы:
  - Класс Widget и его подклассов
  - Классы Color, Cursor, Font, GC, Image, Region,
  - Класс Device и его подклассы (Display, Printer)

- **Правило 1: *“Если вы создали его, вы его освобождаете”***
  - Программист должен освободить шрифт:

```
Font font = new Font (display, "Courier", 10, SWT.NORMAL);
font.dispose ();
```

  - Программист не должен освободить шрифт:

```
Font font = control.getFont();
```

Освобождение ресурсов управляющих элементов их предками.

- **Правило 2: *“Освобождение предка освобождает его потомков “***
  - shell.dispose(); // Освобождает всех потомков окна
  - menu.dispose(); // Освобождает все элементы меню
  - tree.dispose(); // Освобождает все элементы дерева

- Заметим, что:
  - control.dispose();
  - menuItem.dispose();
  - Освобождает элемент меню созданный с помощью setMenu(menu);

Класс Control

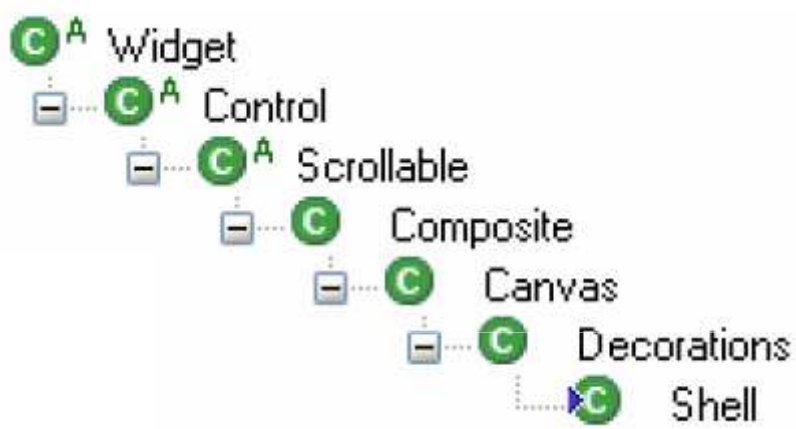
- Абстрактный суперкласс для всех легковесных (heavyweight) элементов интерфейса пользователя

- Стили
  - BORDER, LEFT_TO_RIGHT, RIGHT_TO_LEFT

- События
  - FocusIn, FocusOut
  - KeyDown, KeyUp
  - Traverse
  - MouseDown, MouseUp, MouseDoubleClick
  - MouseEnter, MouseExit, MouseMove, MouseHover
  - Move, Resize
  - Paint
  - Help

Класс Shell

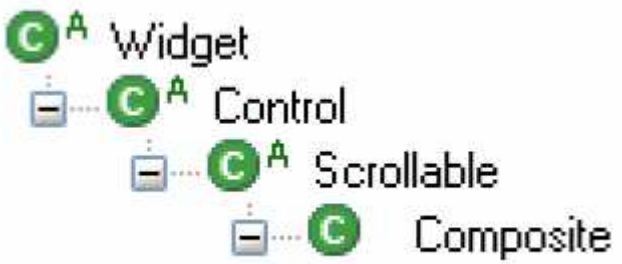
- Конструкторы
  - **new** Shell(display, SWT.SHELL_TRIM);
  - **new** Shell(shell, SWT.DIALOG_TRIM);



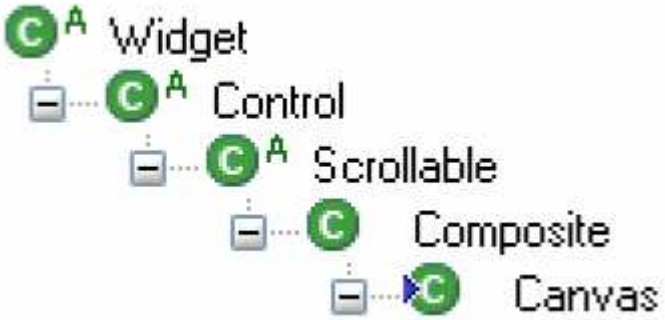
- Стили
  - BORDER, CLOSE, MIN, MAX, NO_TRIM, RESIZE, TITLE
  - APPLICATION_MODAL, MODELESS, PRIMARY_MODAL, SYSTEM_MODAL
- События
  - Close, Activate, Deactivate, Iconify, Deiconify
- Характерные методы
  - open(), close(), setActive()
- Замечания
  - Предок для *shell* верхнего уровня всегда Display
  - Предок для *shell* - диалогового окна всегда *Shell* верхнего уровня

Класс Composite

- Конструкторы
  - **new** Composite(parent, SWT.NONE);
- Стили
  - NO_BACKGROUND, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE, NO_RADIO_GROUP
- События
  - Close, Activate, Deactivate, Iconify, Deiconify
- Характерные методы
  - getChildren()
  - setLayout(Layout), layout(boolean)
  - setTabList(Control[])
- Замечания
  - может иметь потомков – управляющие элементы (*controls*)
  - может использовать класс Layout для задания положения потомков
  - используется как суперкласс для создания управляющих элементов (*widgets*) пользователя



Класс Canvas

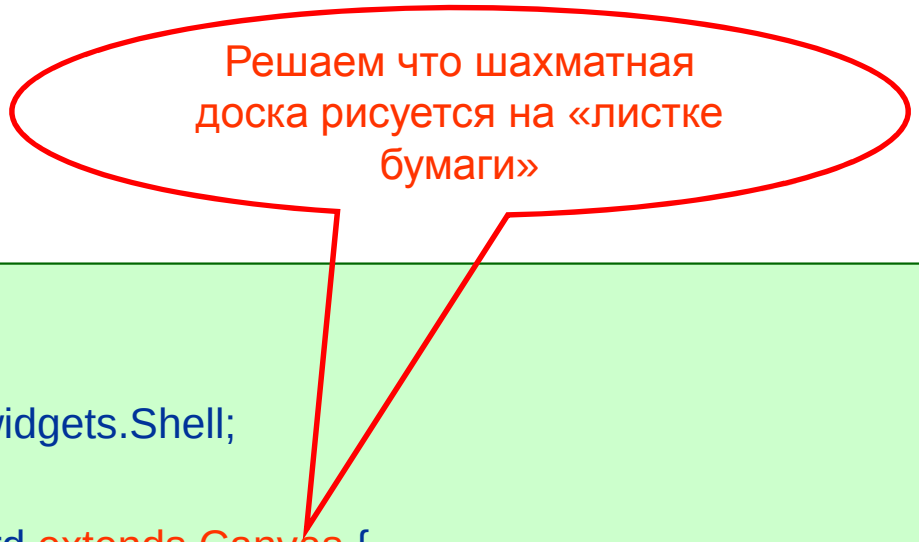


- Конструкторы
  - **new** Canvas(parent, SWT.NONE);
- Стили
  - NO_BACKGROUND, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE
- Характерные методы
  - scroll(int, int, int, int, int, int, boolean)
  - setCaret(Caret)
- Замечания
  - Обычно используется как «чистый лист бумаги» для рисования графики
  - используется как суперкласс для создания управляющих элементов (*widgets*) пользователя

Шахматы. Создание шахматной доски


```
public class Chess {
    public static void main(String[] args) {
        final Display display = new Display();
        ChessImages.load(display);
         final Shell shell = new Shell(display);
        shell.setSize(600, 500);
        shell.setText("Chess Notebook");
         shell.setImage(ChessImages.iconChessNotebook);

         new ChessBoard(shell, SWT.NONE);
        shell.open();
        while (!shell.isDisposed()) {
                if (!display.readAndDispatch())
                    display.sleep();
        }
       display.dispose();
    } // main
} // class Chess
```

Шахматы. Сгенерированный класс ChessBoard

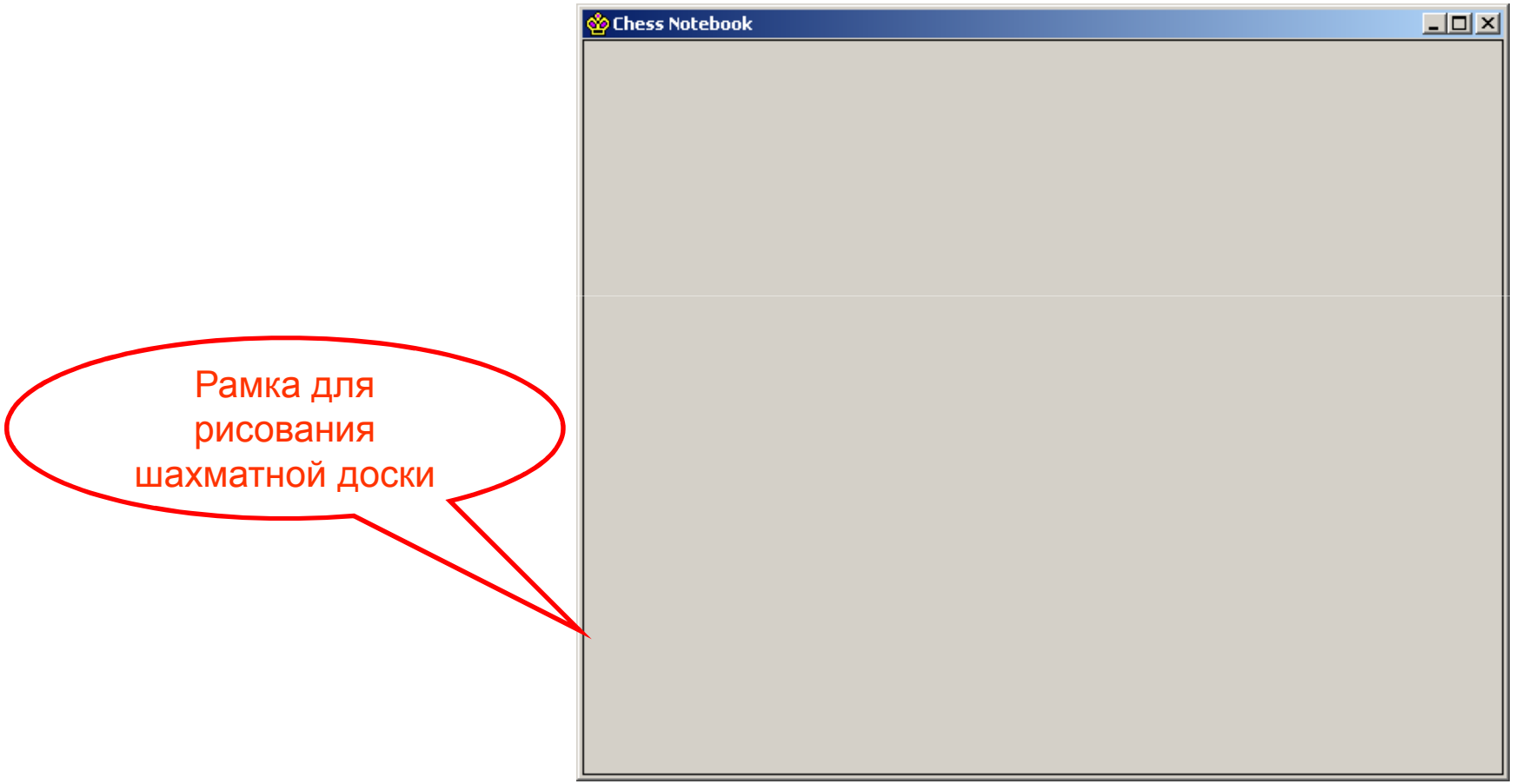



```
package chess.ui;

import org.eclipse.swt.widgets.Shell;

public class ChessBoard extends Canvas {

        public ChessBoard(Shell parent, int style) {
            // TODO Auto-generated constructor stub
            super(parent, style);
        }
}
```

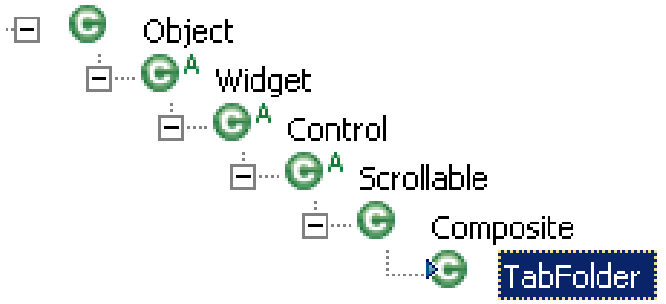
Шахматы. Отрисовка шахматной доски


```
package chess.ui;

import org.eclipse.swt.events.PaintEvent;
import org.eclipse.swt.events.PaintListener;
import org.eclipse.swt.widgets.Canvas;
import org.eclipse.swt.widgets.Shell;

public class ChessBoard extends Canvas implements PaintListener {

        public ChessBoard(Shell parent, int style) {
            super(parent, style);
            addPaintListener(this);
        }
        public void paintControl(PaintEvent e) {
            Rectangle clientArea = getClientArea();
            e.gc.drawRectangle(0, 0, clientArea.width - 1, clientArea.height - 1);
        }
}
```

Шахматы. Рамка шахматной доски



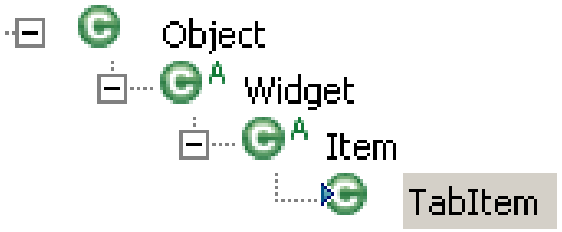
Класс TabFolder



- Конструкторы
  - **new** TabFolder(parent, SWT.TOP);
- Стили
  - SWT.TOP , SWT.BOTTOM, SWT.LEFT, SWT.RIGHT
- Характерные методы
  - setSelection(int, boolean)
  - setSelection(TabItem)
  - TabItem getSelection()
  - int getSelectionIndex()
- Замечания
  - используется как «книга» с закладками

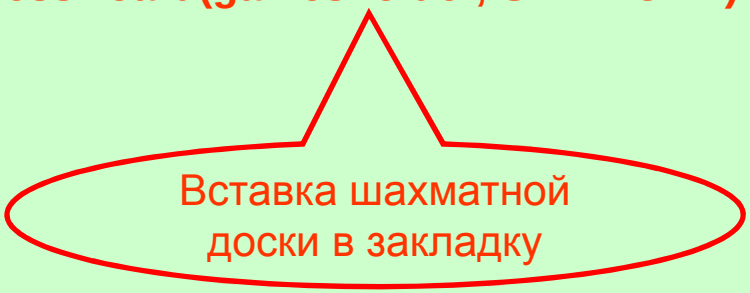
Класс TabItem

- Конструкторы
  - **new** TabItem(parent, SWT.NONE);
- Стили
  - Унаследованные от родителей
- Характерные методы
  - setControl(Control)
  - setImage(Image)
  - setText(String)
  - setToolTipText(String)
- Замечания
  - используется как закладка в «книге»

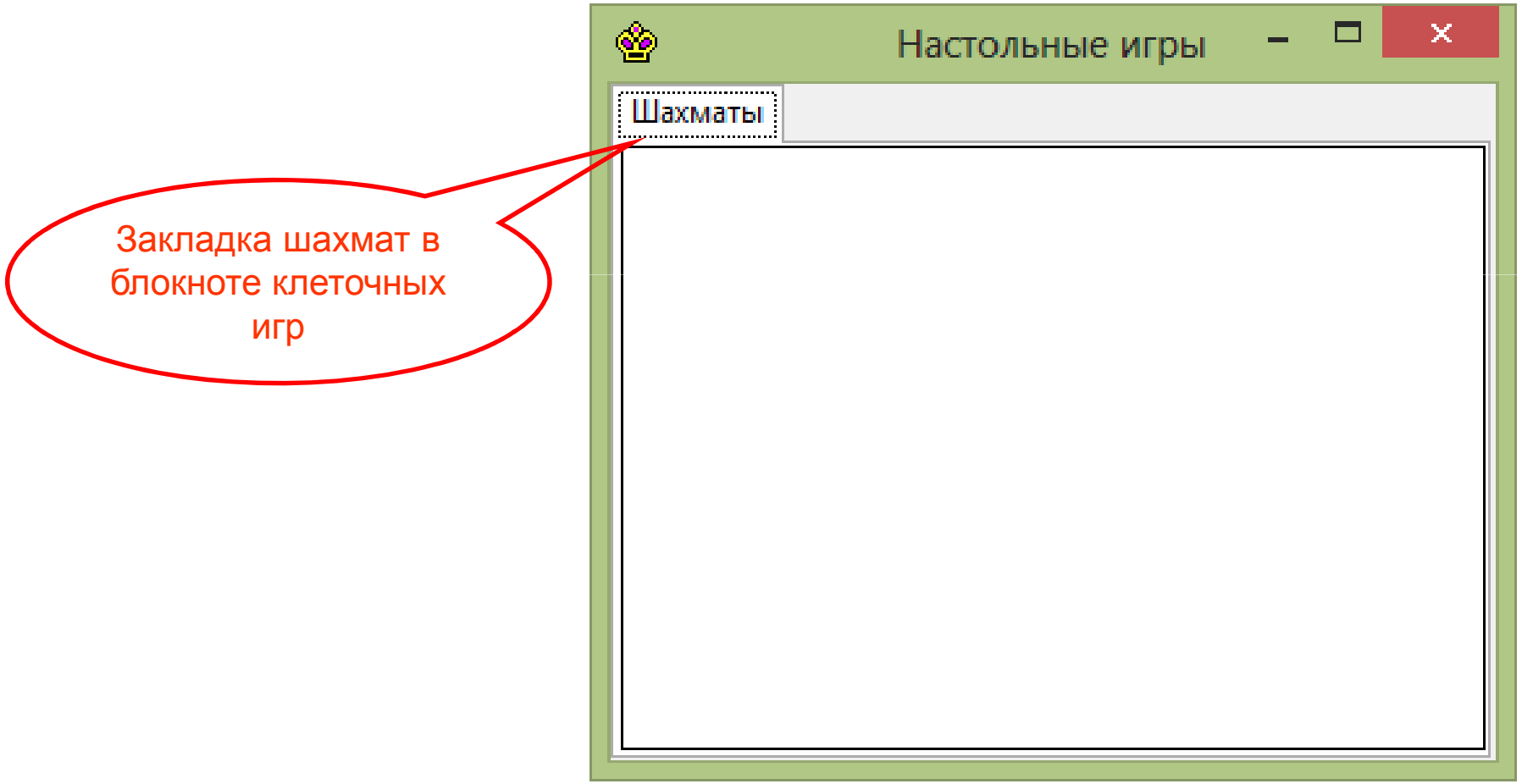


Шахматы. Шахматы – закладка в книге


```
public static void main(String[] args) {
        // …

        final TabFolder gamesFolder = new TabFolder(shell, SWT.TOP);

        TabItem chessItem = new TabItem(gamesFolder, SWT.NULL);
        chessItem.setText("Шахматы");
        chessItem.setControl( new ChessBoard(gamesFolder, SWT.NONE) );

        // …
} // main
```



Шахматы. Закладка шахмат в блокноте клеточных игр



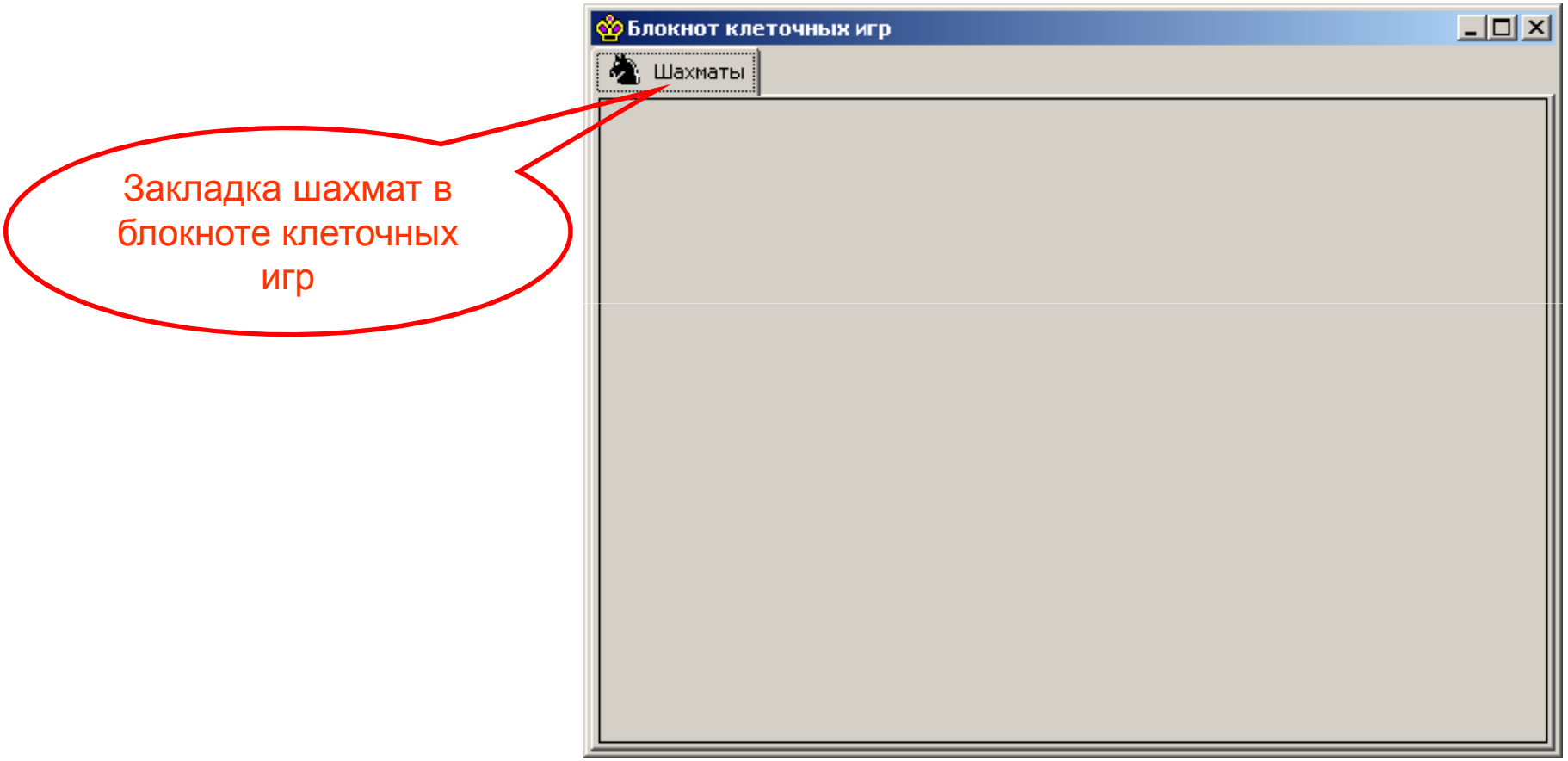
Шахматы. Шахматы – пиктограмма для закладки в блокноте игр


```
public static void main(String[] args) {
        // …

        final TabFolder gamesFolder = new TabFolder(shell, SWT.TOP);

        final Image chessTabImage = new Image(display,
                        ChessImages.imageKnightBlack.getImageData().scaledTo(20, 20)    );

        TabItem chessItem = new TabItem(gamesFolder,  SWT.NULL);
        chessItem.setText("Шахматы");
        chessItem.setImage(chessTabImage);
        chessItem.setControl(  new ChessBoard(gamesFolder,  SWT.NONE)  );

        // …
} // main
```

Масштабирование рисунка

Пиктограмма для закладки шахмат в блокноте клеточных игр

Шахматы. Закладка шахмат

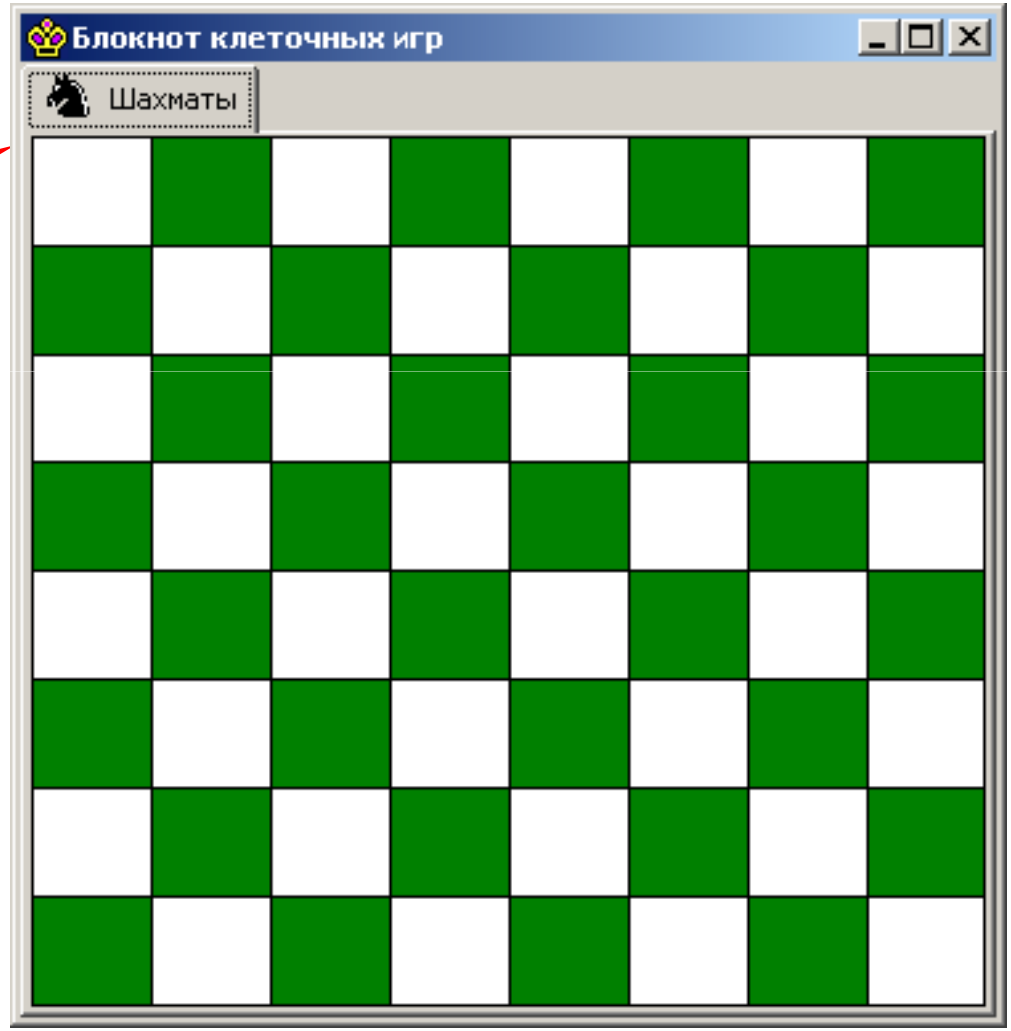


Шахматы. Отрисовка клеток доски


```
public void paintControl(PaintEvent e) {
Rectangle clientArea = getClientArea();
GC gc = e.gc;
for (int v = 0; v < nV; v++)
  for (int h = 0; h < nH; h++) {
     int squareWidth  = getClientArea().width  / nV;
     int squareHeight = getClientArea().height / nH;

     boolean isWhiteSquare = ((v + h) % 2 == 0);
     Color squareColor = isWhiteSquare ? new Color(null, 255, 255,255) : new Color(null, 0, 192, 0);

     gc.setBackground(squareColor);
     gc.fillRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);

     gc.setForeground(new Color(null, 0, 0, 0));
     gc.drawRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);
} // for
e.gc.drawRectangle(0, 0, clientArea.width - 1, clientArea.height - 1);
} // paintControl
```

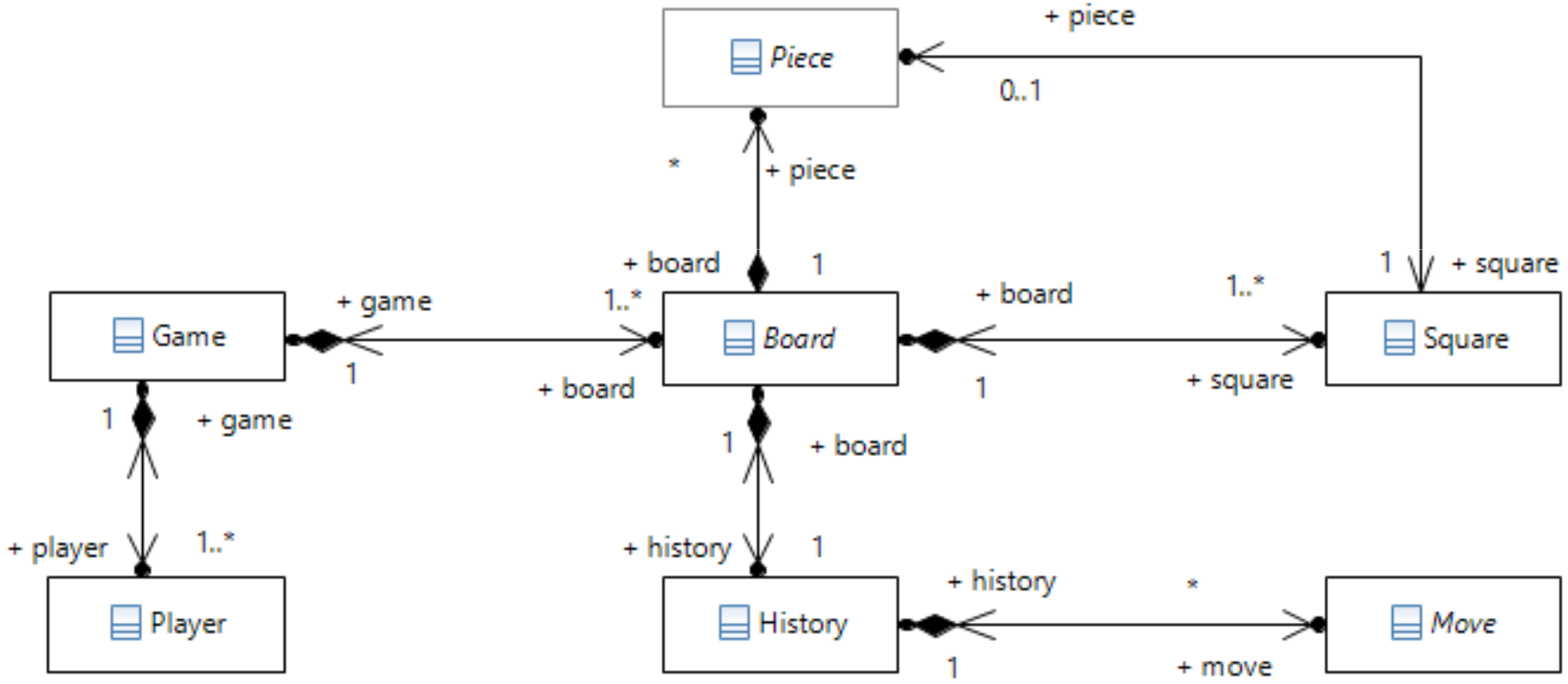
Шахматы. Клетки шахматной доски

Клетки шахматной доски

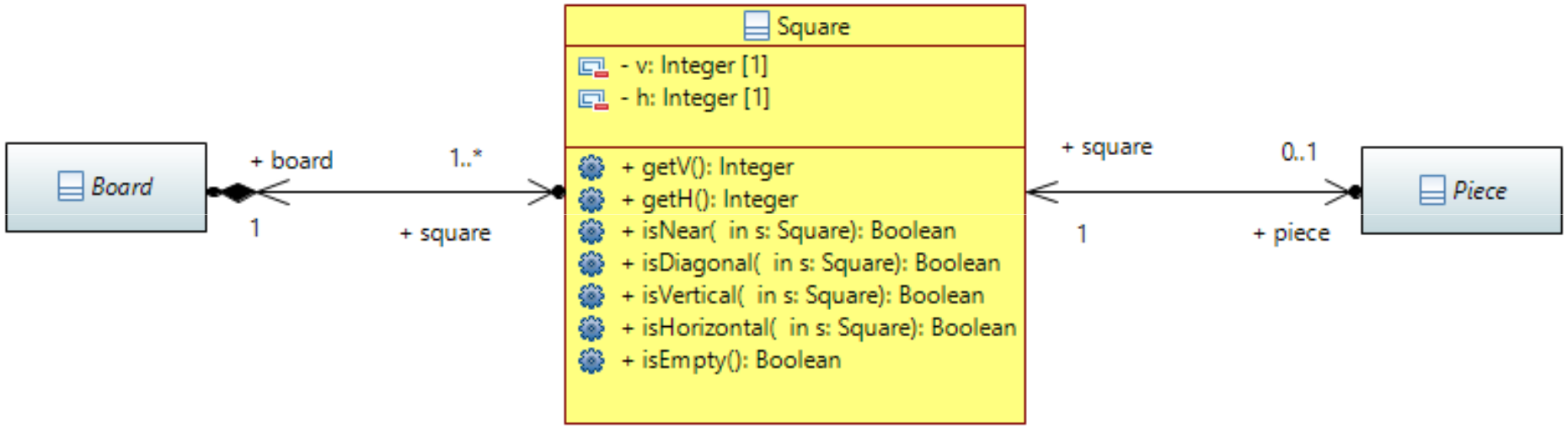


Проект на языке UML

Пакет games.core - ядро настольных игр. Содержимое пакета. Диаграмма наUML



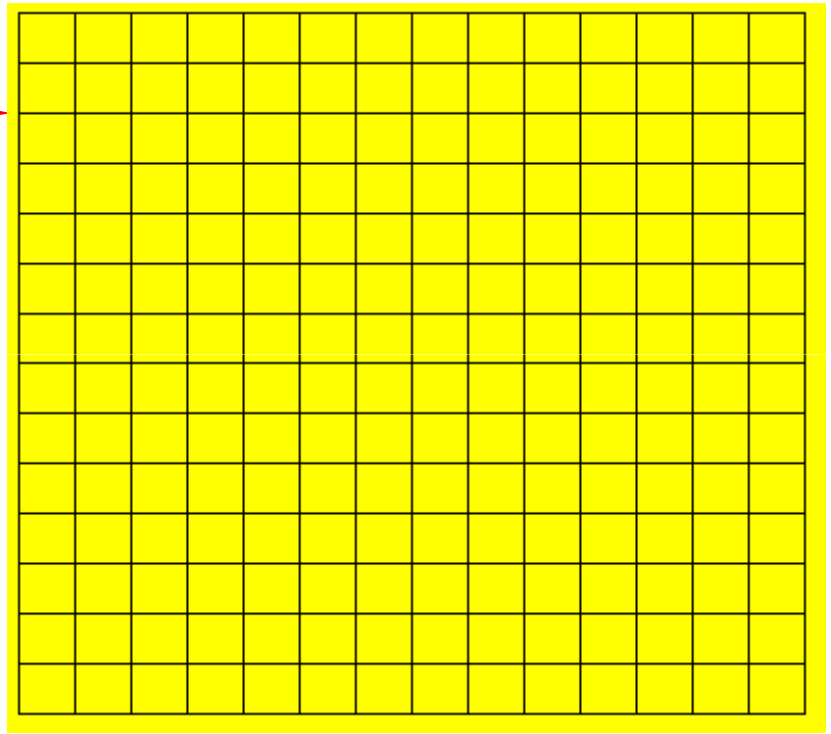
Класс games.core.Square
Диаграмма наUML



Рефакторинг программы. Разновидности досок для игр

Клетки доски для игр го и рендзю

Фигуры располагаются на пересечении линий



- **Цель – преобразовать программу для рисования досок различных настольных игр**

Шахматы. Отрисовка клеток доски


```
public void paintControl(PaintEvent e) {
Rectangle clientArea = getClientArea();
GC gc = e.gc;
for (int v = 0; v < nV; v++)
  for (int h = 0; h < nH; h++) {
     int squareWidth  = getClientArea().width  / nV;
     int squareHeight = getClientArea().height / nH;

     boolean isWhiteSquare = ((v + h) % 2 == 0);
     Color squareColor = isWhiteSquare ? new Color(null, 255, 255,255) : new Color(null, 0, 192, 0);

     gc.setBackground(squareColor);
     gc.fillRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);

     gc.setForeground(new Color(null, 0, 0, 0));
     gc.drawRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);
} // for
e.gc.drawRectangle(0, 0, clientArea.width - 1, clientArea.height - 1);
} // paintControl
```

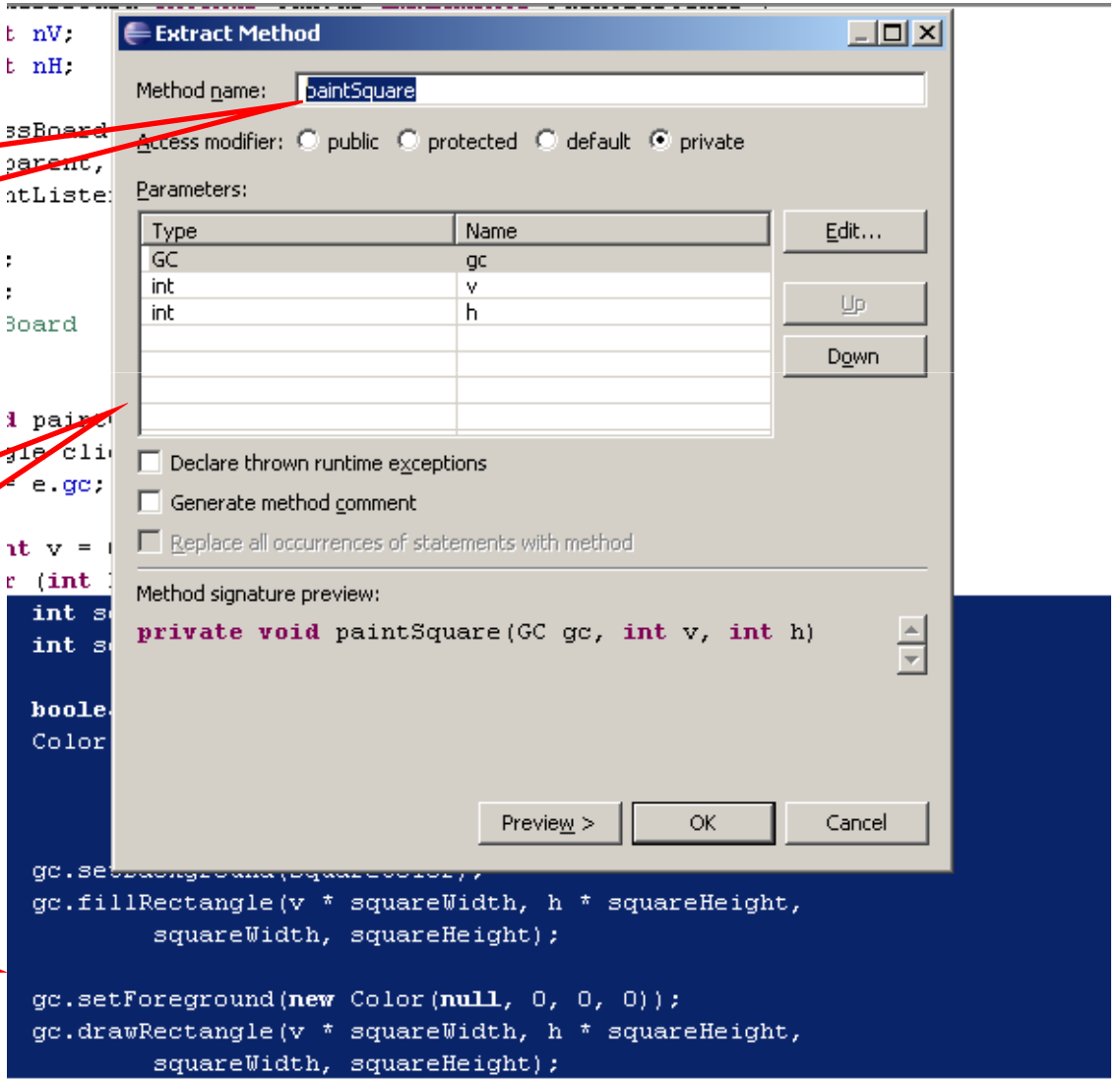
Алгоритм рисования клетки шахматной доски вынесем в отдельный метод

Рефакторинг. Выделение метода

3. напечатать имя метода

2. Потом открыть диалоговое окно ALT + Shift + M

1. Сначала выделение текста



Шахматы. Выделенный метод рисования клетки шахмат (двухцветная доска)


```
private void paintSquare(GC gc, int v, int h) {
        int squareWidth  = getClientArea().width  / nV;
        int squareHeight = getClientArea().height / nH;

        boolean isWhiteSquare = ((v + h) % 2 == 0);
        Color squareColor = isWhiteSquare
              ? new Color(null, 255, 255,255) : new Color(null, 0, 192, 0);

        gc.setBackground(squareColor);
        gc.fillRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);

        gc.setForeground(new Color(null, 0, 0, 0));
        gc.drawRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);
}
```

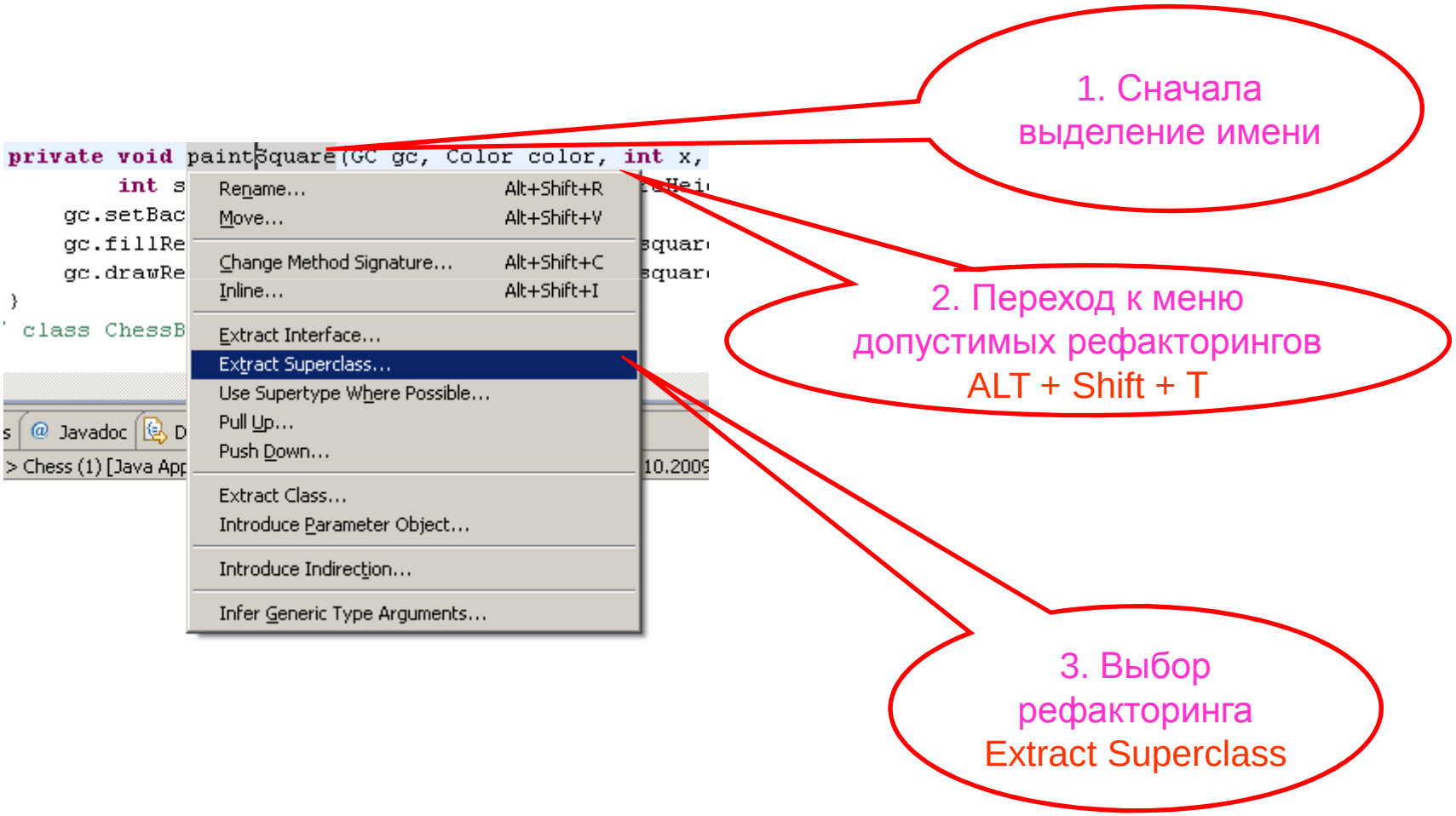
Шахматы. Вызов выделенного метода в универсальном методе рисования доски


```
public void paintControl(PaintEvent e) {
        Rectangle clientArea = getClientArea();

        GC gc = e.gc;

        for (int v = 0; v < nV; v++)
            for (int h = 0; h < nH; h++) {
                paintSquare(gc, v, h);
            }

        gc.drawRectangle(0, 0, clientArea.width - 1, clientArea.height - 1);
}
```

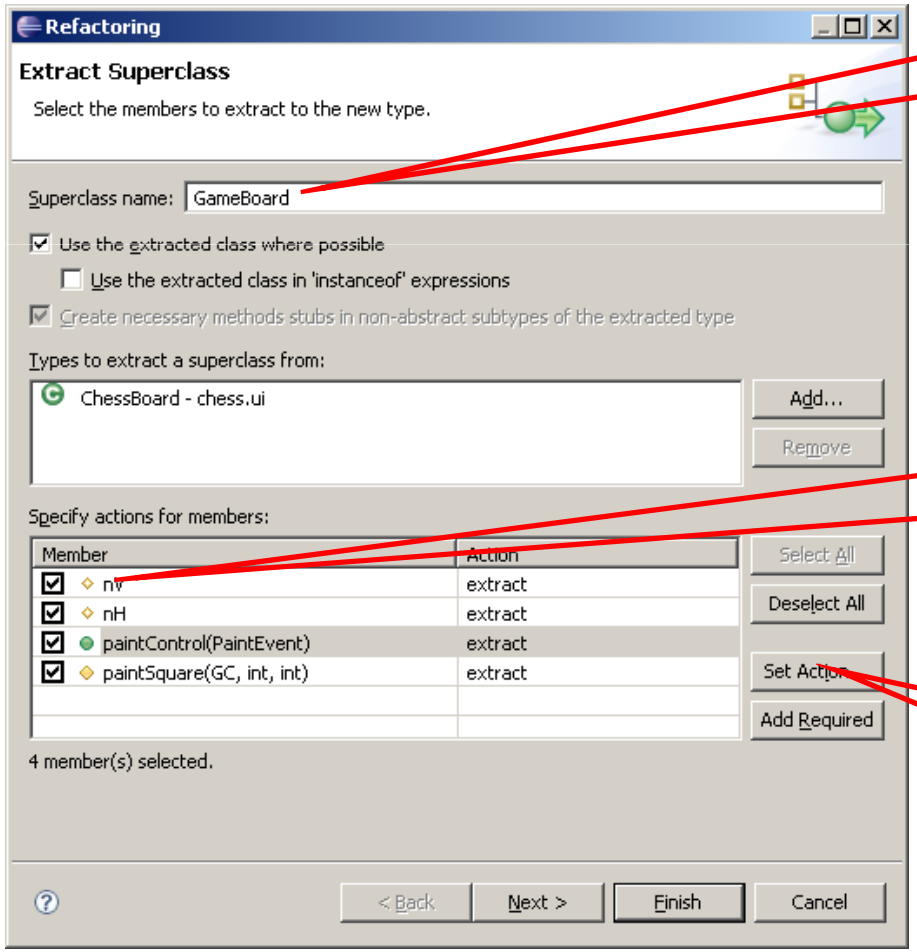
Рефакторинг. Вынос универсальных полей и методов в базовый класс

- Предпосылки
  - Размерность доски - поля nH и nV общие для всех клеточных игр
  - Алгоритм рисования в цикле всех клеток доски универсален

Рефакторинг. Создание абстактного класса *GameBoard*



Рефакторинг. Создание абстрактного класса *GameBoard*

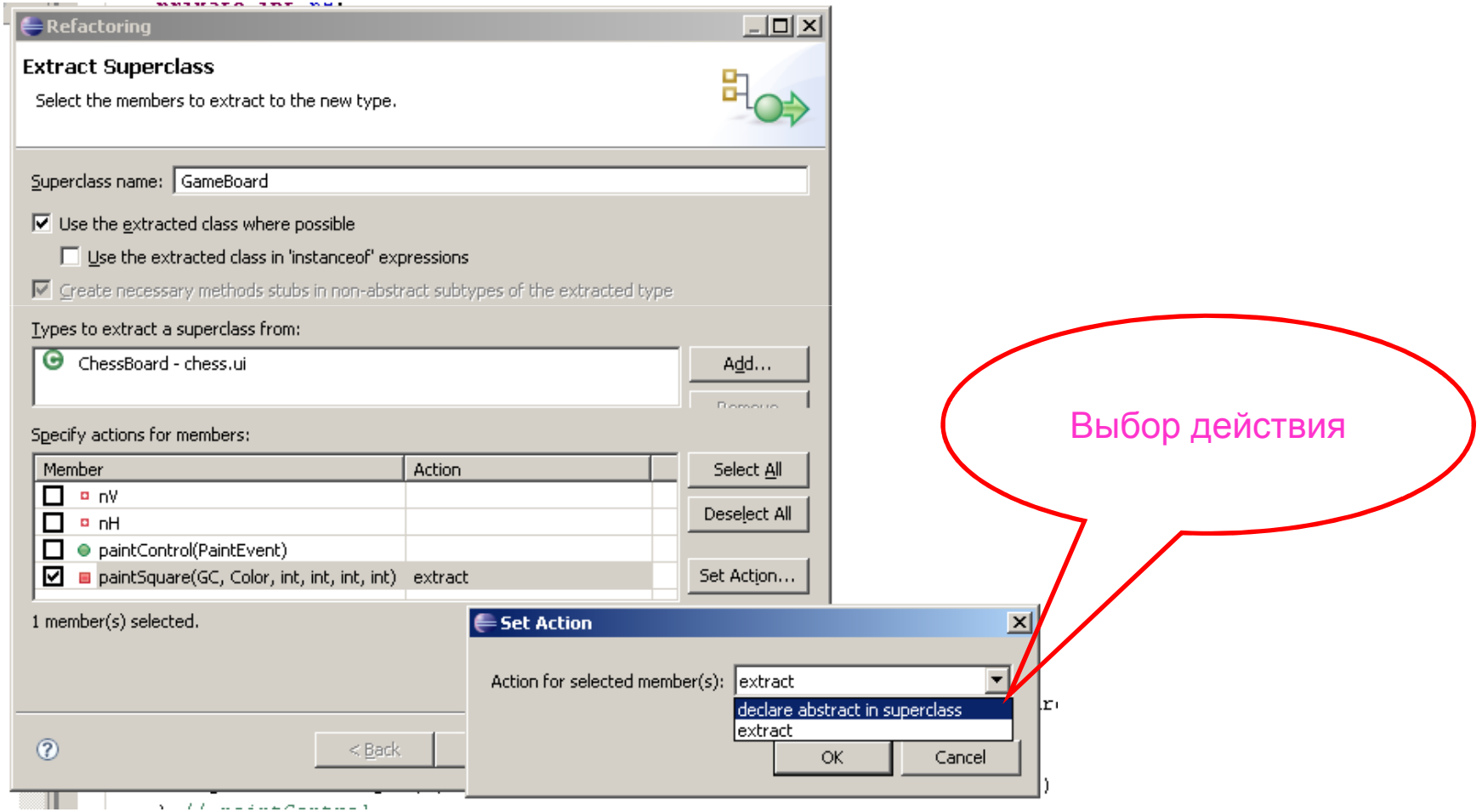


1. Ввод имени суперкласса

2. Выбор переносимых в суперкласс полей и методов

3. Выбор действия умалчиваемое действие extract

Рефакторинг. Создание абстрактного метода в классе *GameBoard*



Шахматы. Выделенный суперкласс *GameBoard*

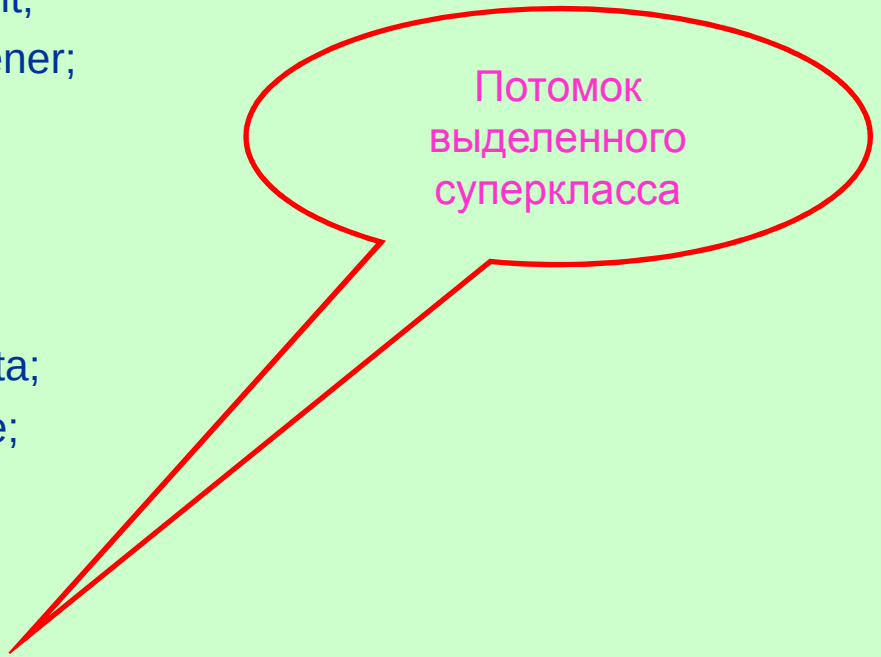

```
public abstract class GameBoard extends Canvas {
        protected int nV;
        protected int nH;
        protected abstract void paintSquare(GC gc, int v, int h);

        public GameBoard(Composite parent, int style)
            { super(parent, style); }

        public void paintControl(PaintEvent e) {
            Rectangle clientArea = getClientArea();
            GC gc = e.gc;
            for (int v = 0; v < nV; v++)
                    for (int h = 0; h < nH; h++)
                            paintSquare(gc, v, h);
            e.gc.drawRectangle(0, 0, clientArea.width - 1, clientArea.height - 1);
        } // paintControl
}
```

Предок выделенного суперкласса

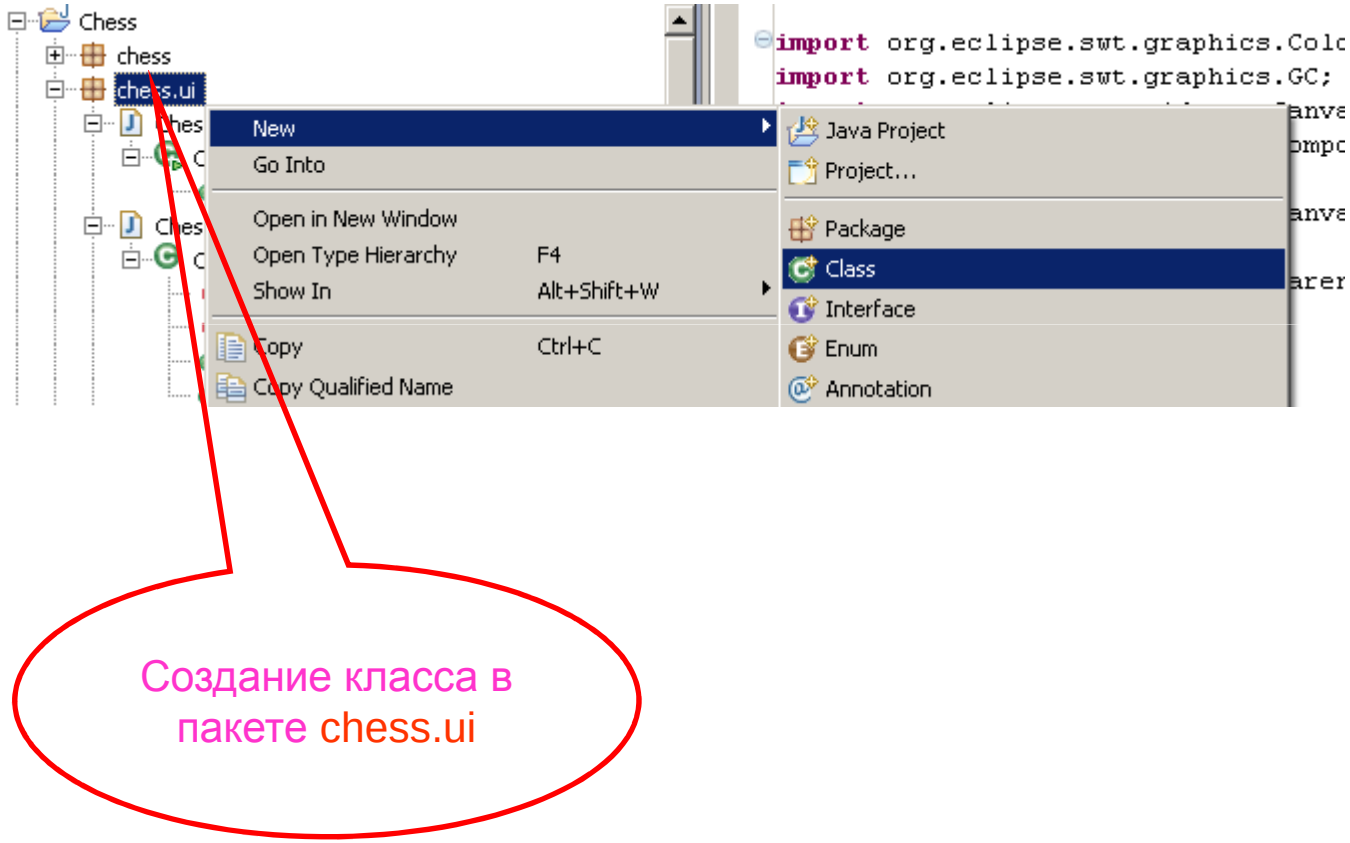
Переопределяемый метод *paintSquare*

Шахматы. *ChessBoard* – потомок выделенного суперкласса *GameBoard*


```
import org.eclipse.swt.events.MouseEvent;
import org.eclipse.swt.events.MouseListener;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.graphics.Cursor;
import org.eclipse.swt.graphics.GC;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.graphics.ImageData;
import org.eclipse.swt.widgets.Composite;

import chess.ChessImages;

public class ChessBoard extends GameBoard {

// …
}
```



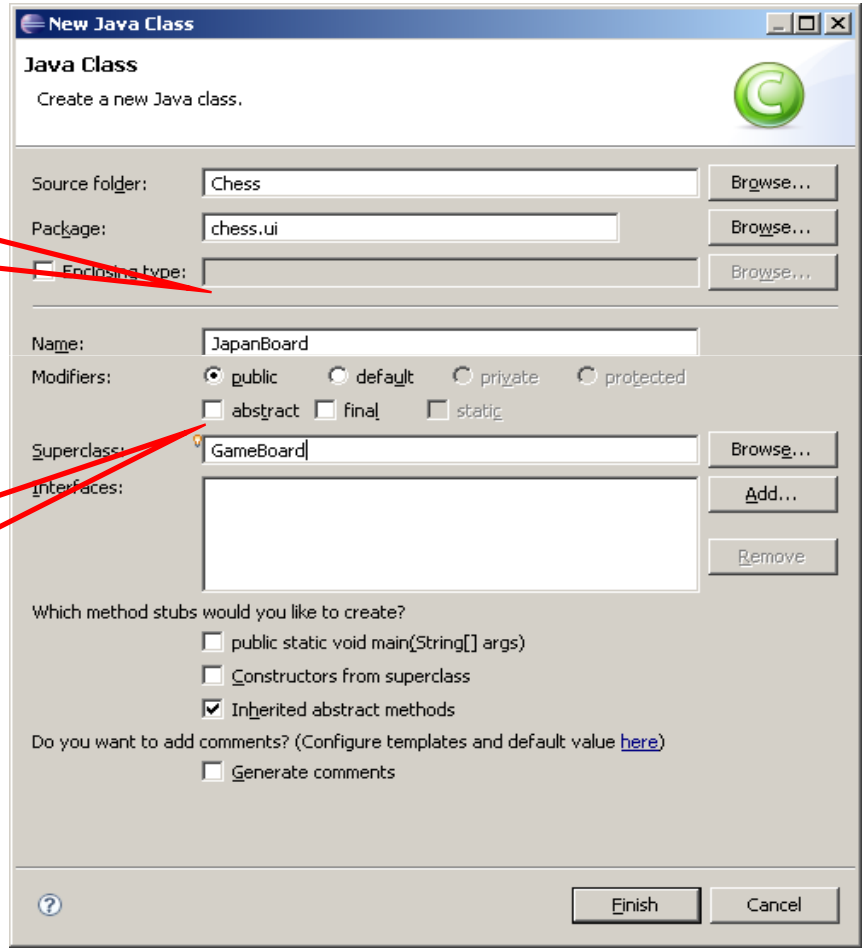
Создание класса JapanBoard - потомка класса *GameBoard* (wizard)



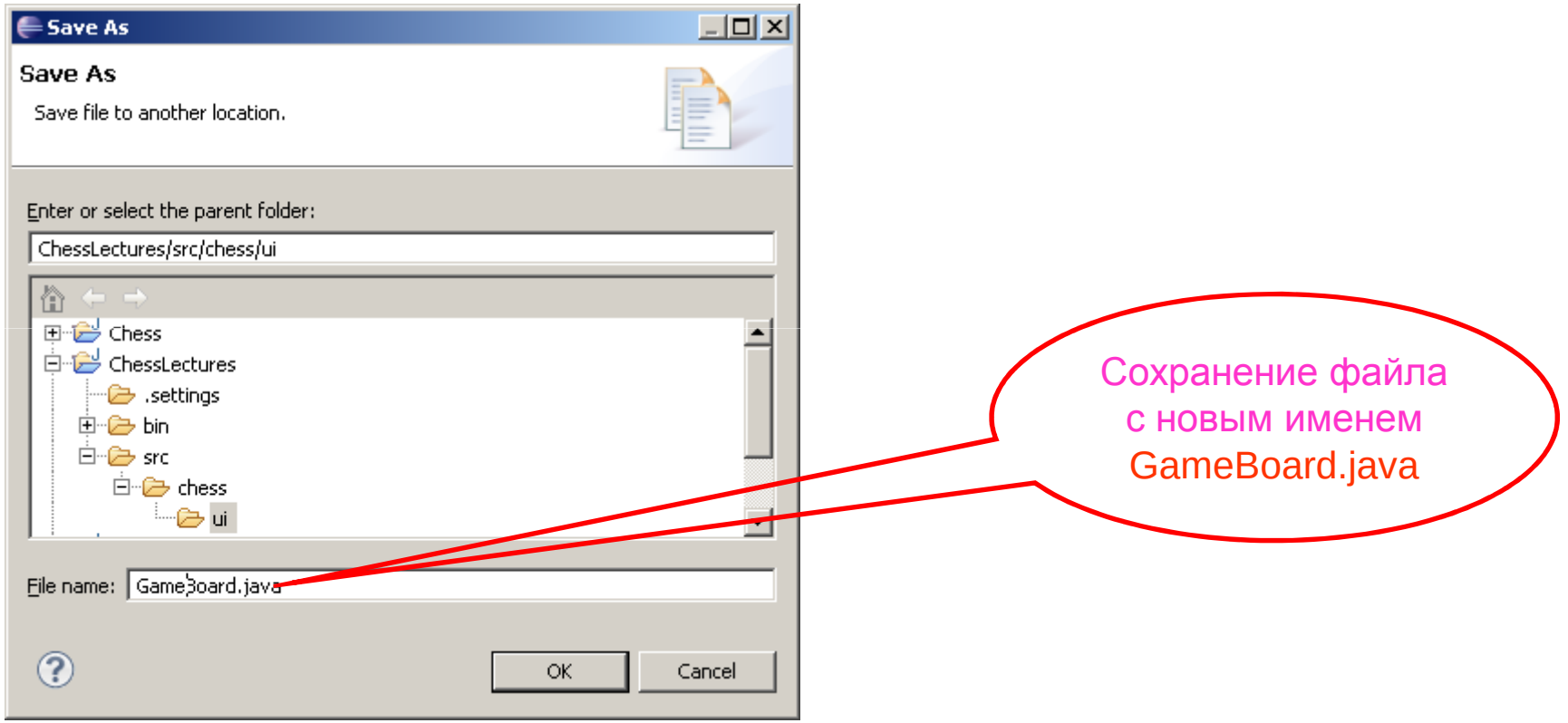
Создание класса JapanBoard - потомка класса *GameBoard* (wizard)

Задание имени класса

Задание имени базового класса

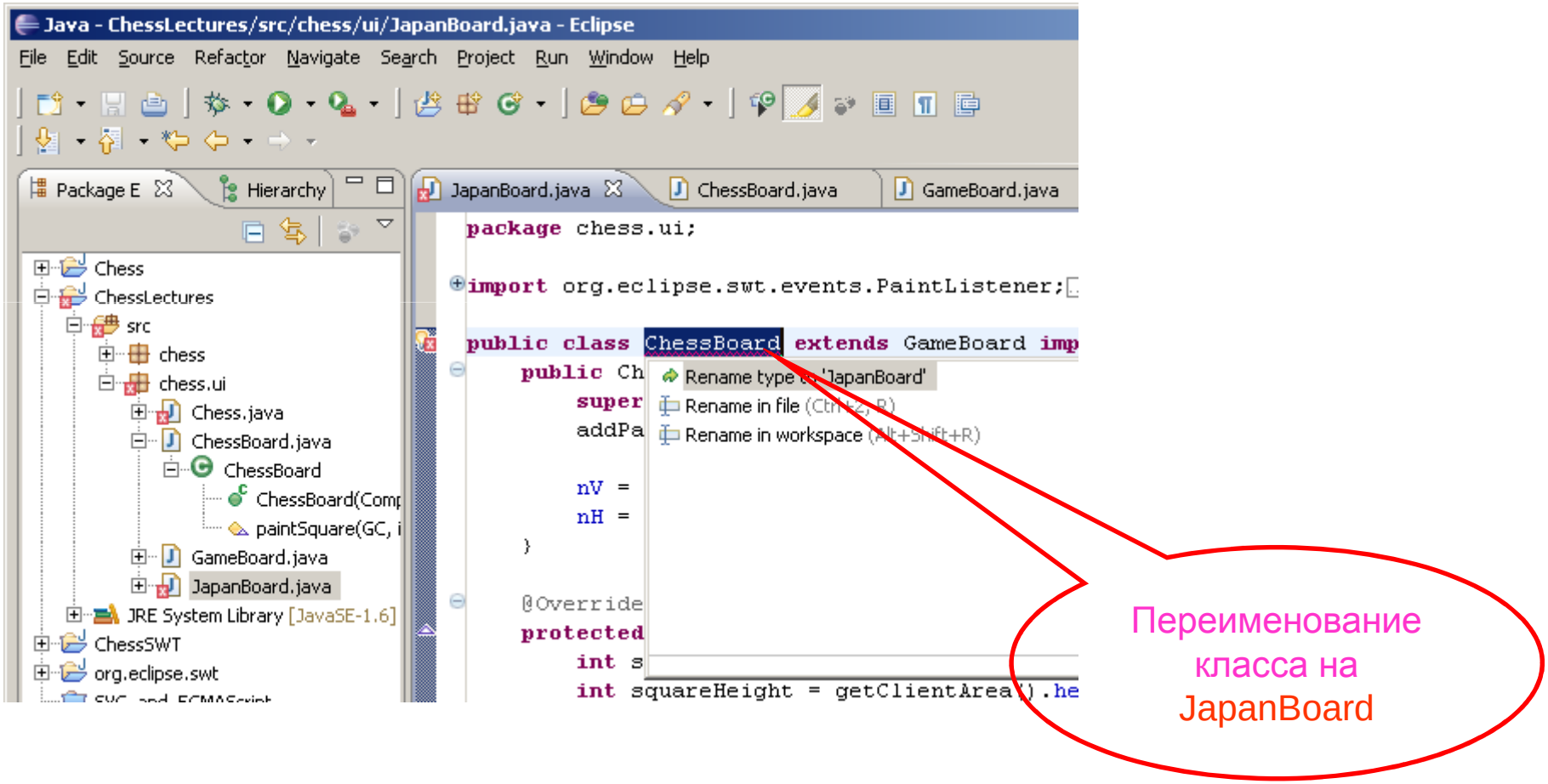


Создание класса JapanBoard - потомка класса *GameBoard* (рефакторинг)



Создание класса JapanBoard - потомка класса *GameBoard* (рефакторинг)

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017



Шахматы. *JapanBoard* – потомок суперкласса *GameBoard* (1)


```
public class JapanBoard extends GameBoard {

public JapanBoard(Composite parent, int style) {
        super(parent, style);

        nV = 15;
        nH = 15;
}

// …
```

Потомок суперкласса GameBoard

Шахматы. *JapanBoard* – потомок суперкласса *GameBoard* (2)


```
@Override
protected void paintSquare(GC gc, int v, int h) {
        int squareWidth  = getClientArea().width  / nV;
        int squareHeight = getClientArea().height / nH;

        gc.setBackground(new Color(null, 255, 255,0) );
        gc.fillRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);

        int x = v * squareWidth  + squareWidth/2;
        int y = h * squareHeight + squareHeight/2;

        if (v !=      0) gc.drawLine(x, y, x - squareWidth/2, y);
        if (v != nV-1) gc.drawLine(x, y, x + squareWidth/2, y);

        if (h !=      0) gc.drawLine(x, y, x, y - squareHeight/2);
        if (h != nH-1) gc.drawLine(x, y, x, y + squareHeight/2);
}
}  // class JapanBoard
```

Реализация абстрактного метода GameBoard.paintSquare

Блокнот игр. Игры на шахматной доске


```
final TabFolder gamesFolder = new TabFolder(shell, SWT.TOP);

final Image chessTabImage = new Image(display,
        ChessImages.imageKnightBlack.getImageData().scaledTo(20, 20));

TabItem chessItem = new TabItem(gamesFolder, SWT.NULL);
chessItem.setText("Шахматы");
chessItem.setImage(chessTabImage);
chessItem.setControl(new ChessBoard(gamesFolder, SWT.NONE));

final Image checkersTabImage = new Image(display,
        ChessImages.imageCheckersNotebook.getImageData().scaledTo(20, 20));

TabItem checkersItem = new TabItem(gamesFolder, SWT.NULL);
checkersItem.setText("Шашки");
checkersItem.setImage(checkersTabImage);
checkersItem.setControl(new ChessBoard(gamesFolder, SWT.NONE));
```

Блокнот

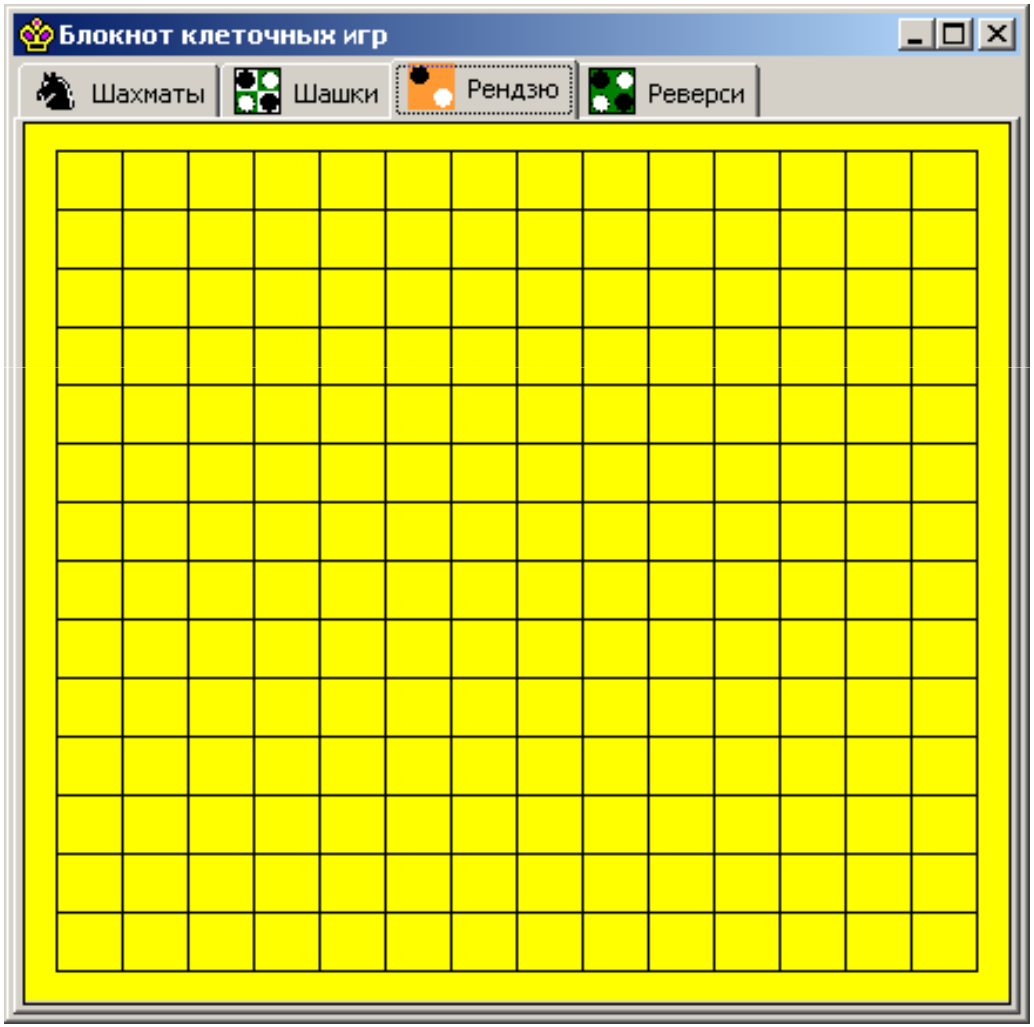
Блокнот игр. Игры на «японской» доске


```
final Image renjuTabImage = new Image(display,
        ChessImages.imageRenjuNotebook.getImageData().scaledTo(20, 20));

TabItem renjuItem = new TabItem(gamesFolder, SWT.NULL);
renjuItem.setText("Рендзю");
renjuItem.setImage(renjuTabImage);
renjuItem.setControl(new JapanBoard(gamesFolder, SWT.NONE));

final Image reversiTabImage = new Image(display,
        ChessImages.imageReversiNotebook.getImageData().scaledTo(20, 20));

TabItem reversiItem = new TabItem(gamesFolder, SWT.NULL);
reversiItem.setText("Реверси");
reversiItem.setImage(reversiTabImage);
reversiItem.setControl(new JapanBoard(gamesFolder, SWT.NONE));
```

Блокнот

Блокнот игр. Доска для игры

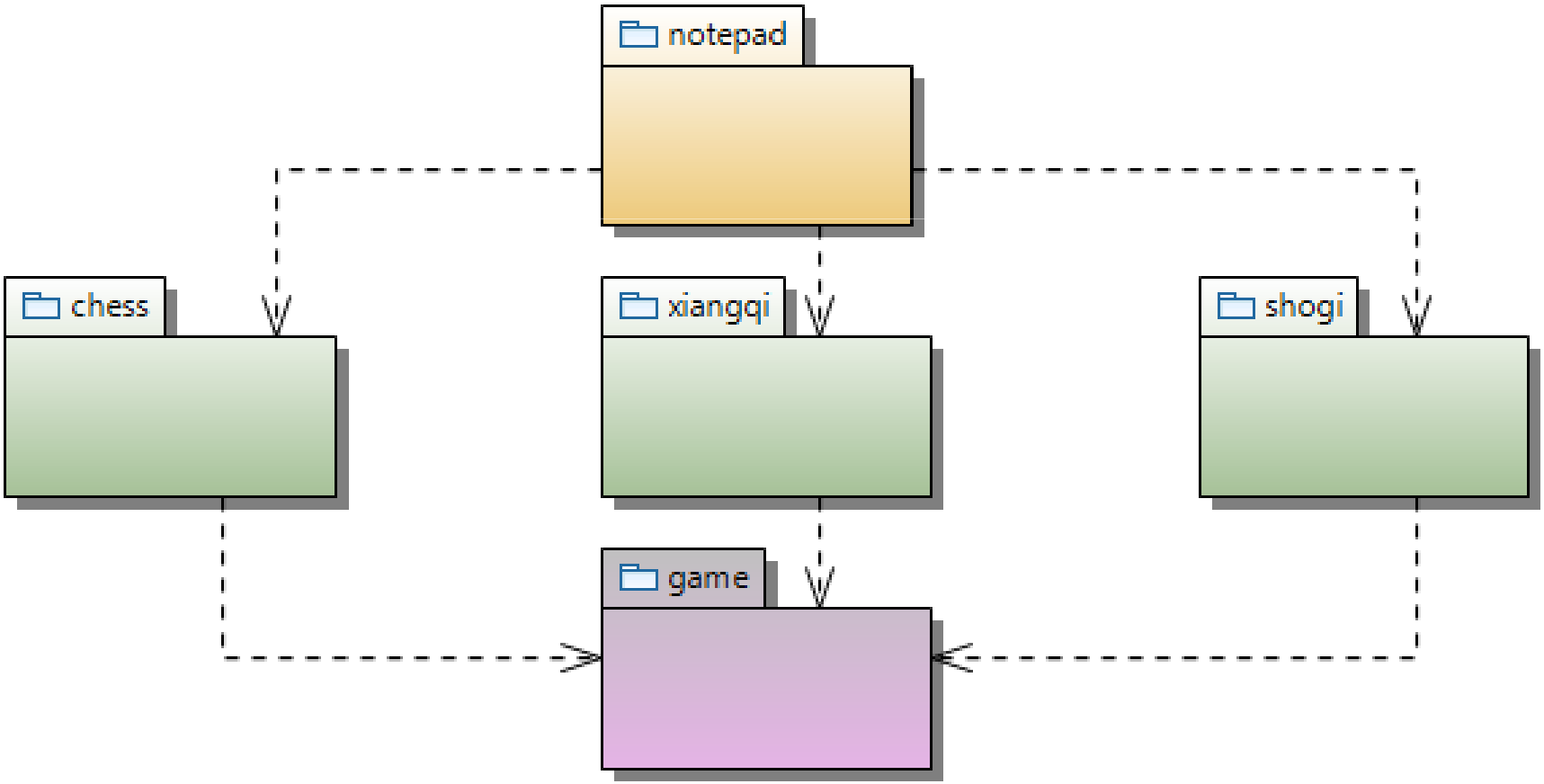
Рендзю, Го, Сянци (китайские шахматы)



Проектирование архитектуры.
Выделение уровней в настольных играх

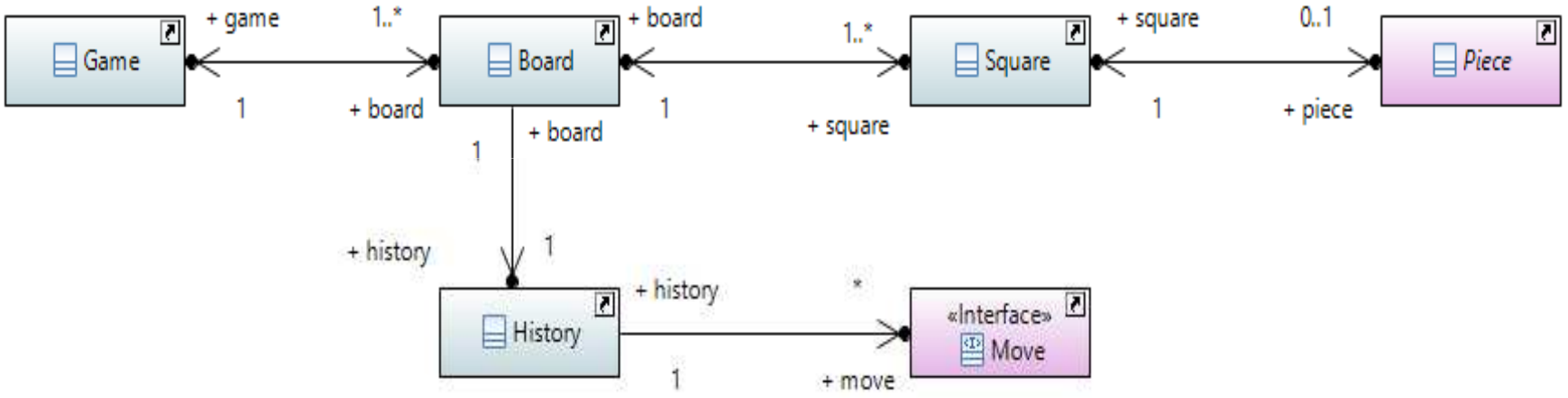
МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017

Идентификация *пакетов верхнего уровня*

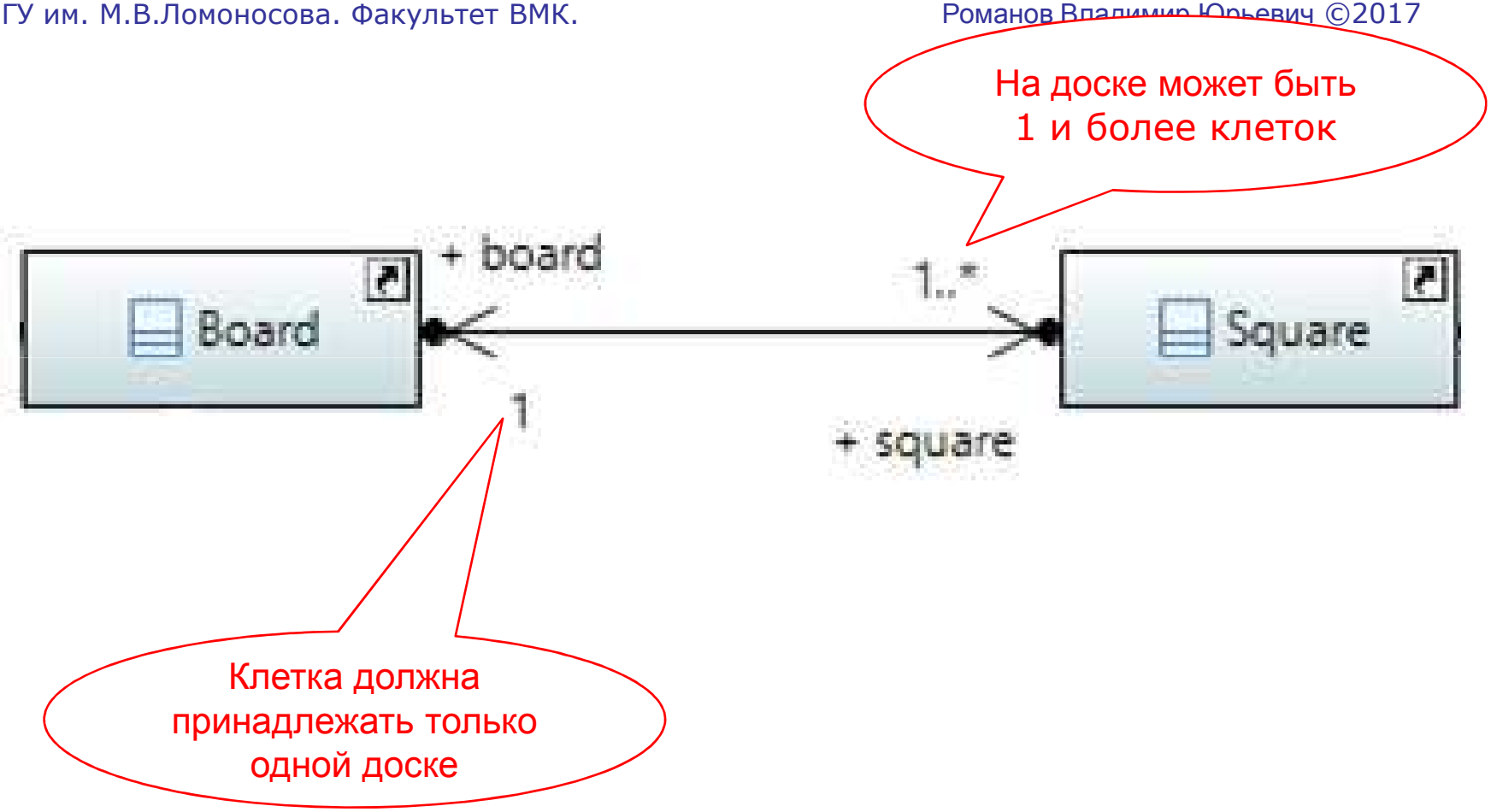


Проектирование архитектуры.

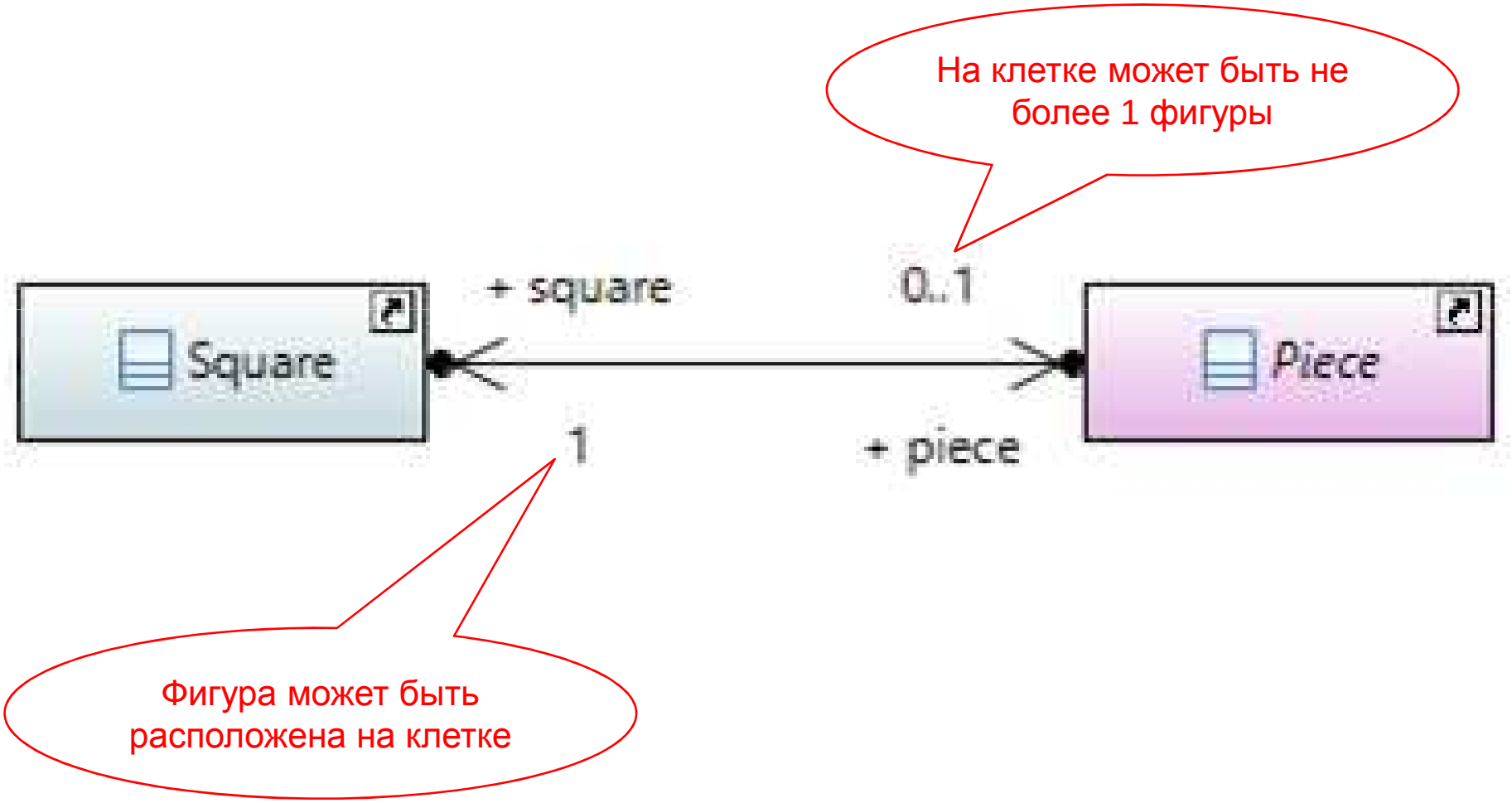
Классы пакета game.core



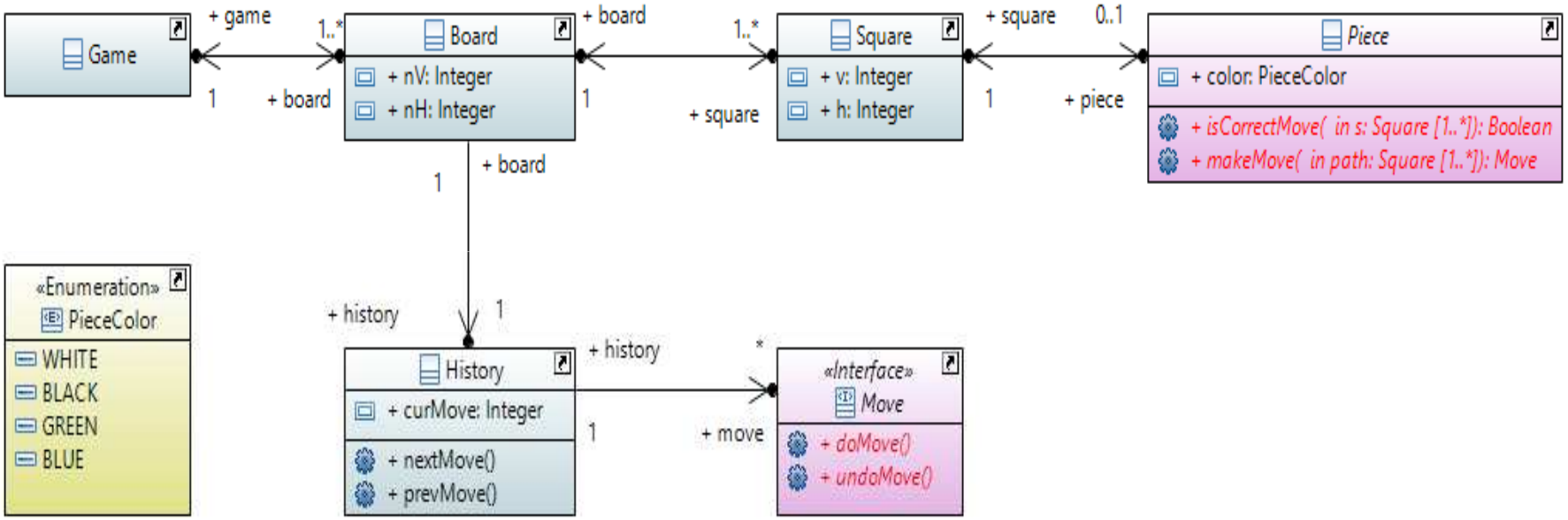
Отношение ассоциации между классам *Board* и *Square*



Отношение ассоциации между классам *Piece* и *Square*

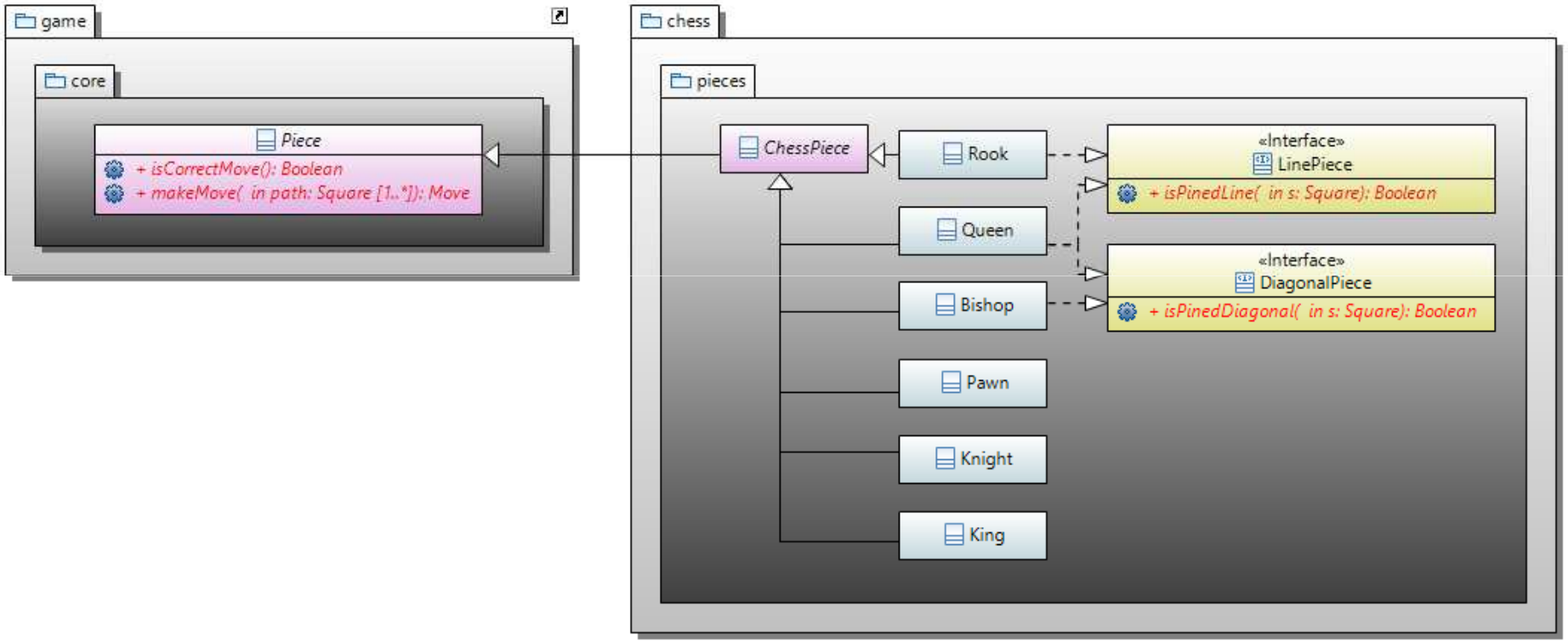


Атрибуты и методы классов ядра настольных игр



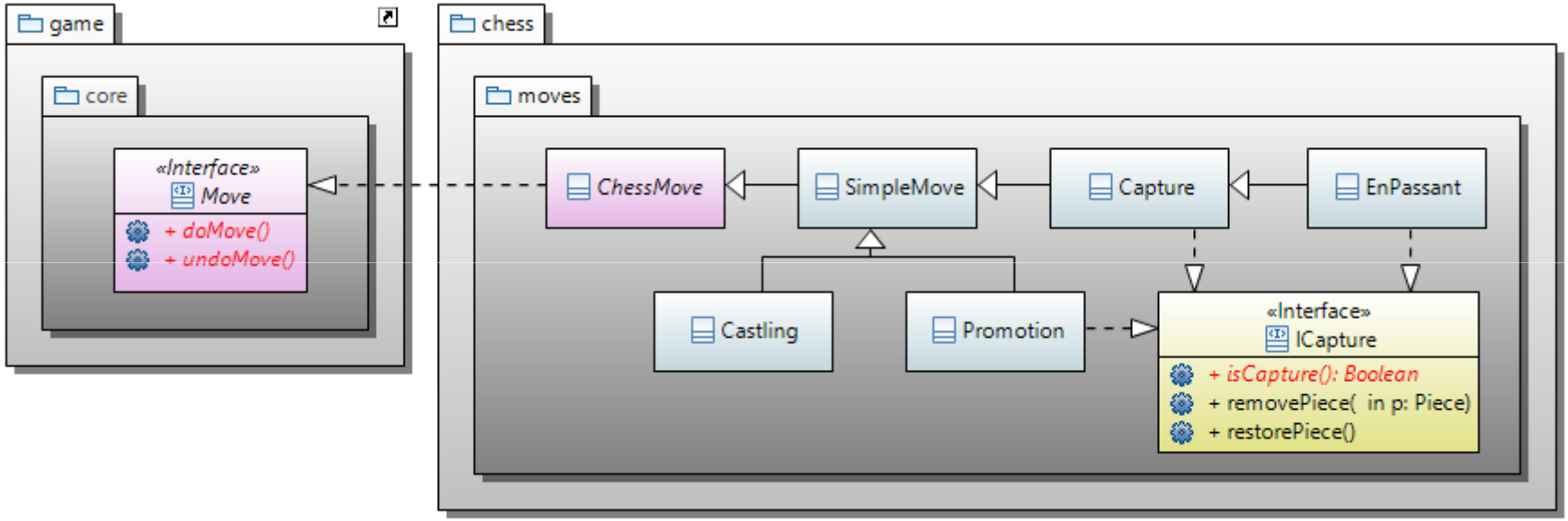
Проектирование классов

Классы-фигуры в шахматной программе



Проектирование классов

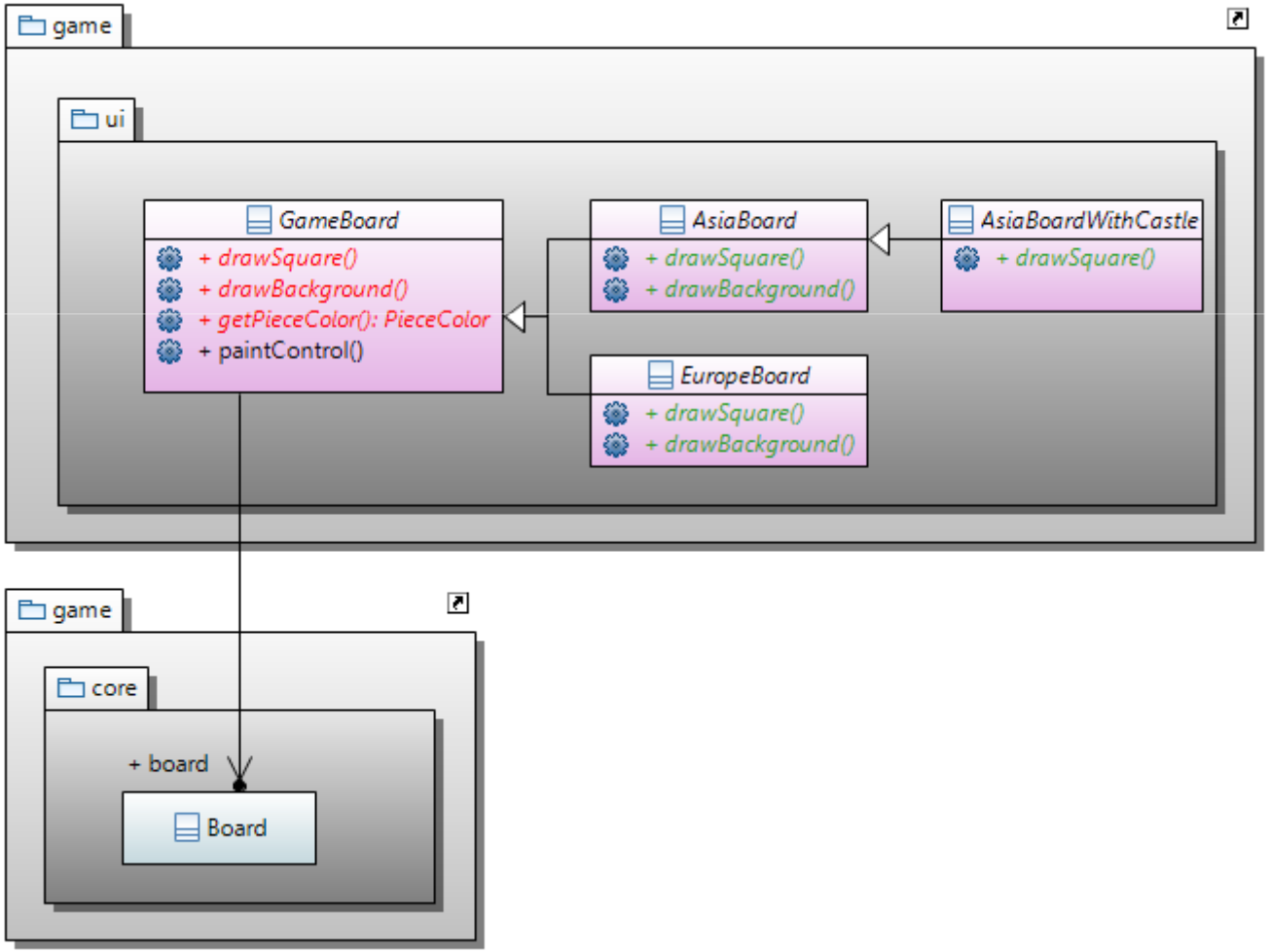
Классы-ходы в шахматной программе



Проектирование классов

Базовые классы интерфейса пользователя

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017



Объявление абстрактных методов в абстрактном классе


```
public abstract class GameBoard extends Canvas implements PaintListener  {
// …
        protected Board board;

       public GameBoard(Composite parent, Board board) {
             super(parent, SWT.NONE);

             this.board = board;

             addPaintListener(this);
       }
// …
}
```

Объявление абстрактных методов в абстрактном классе


```
public abstract class GameBoard extends Canvas implements PaintListener  {
// …
        abstract
        protected void drawBackground (GC gc, Rectangle area);

        abstract
        public Image getPieceImage (Piece piece);

        abstract
        public void drawSquare (GC gc, int v, int h, int squareWidth, int squareHeight);
// …
}
```

Использование абстрактных методов в абстрактном классе

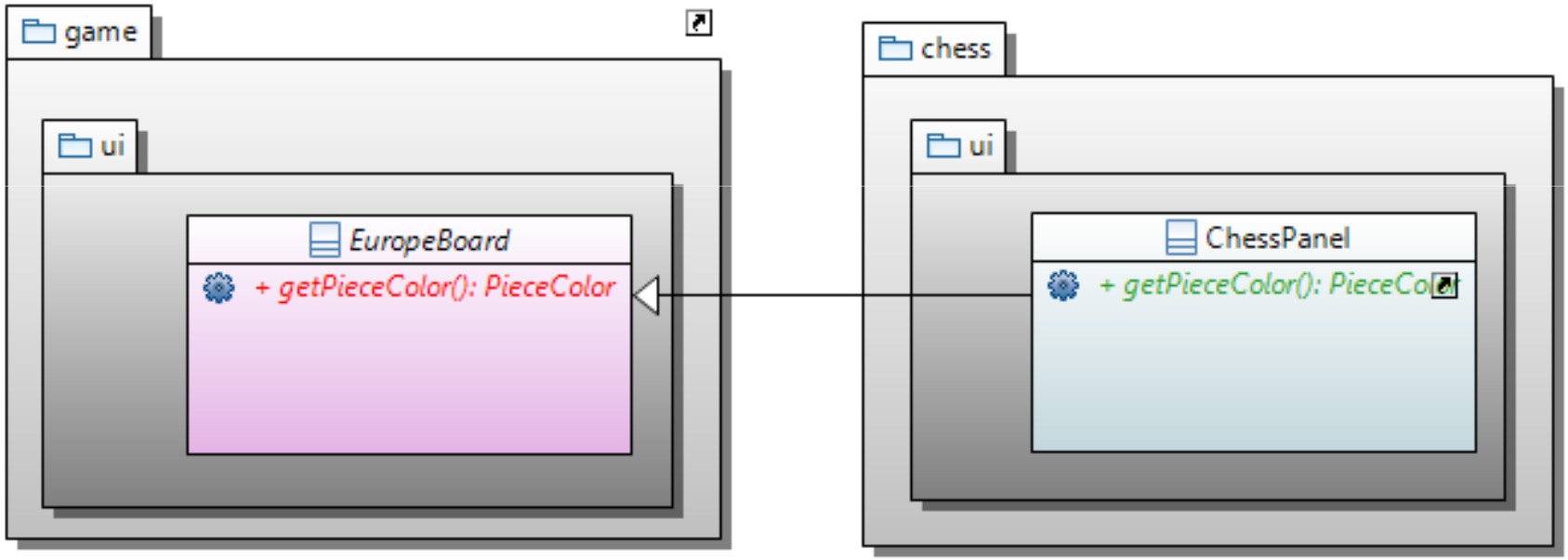

```
public abstract class GameBoard extends Canvas implements PaintListener  {
// …
@Override
public void paintControl (PaintEvent e) {
        GC gc = e.gc;
        Rectangle clientArea = getClientArea();

        drawBackground(gc, clientArea);

        int squareWidth  = getClientArea().width  / board.nV;
        int squareHeight = getClientArea().height / board.nH;

        for (int v = 0; v < board.nV; v++) {
                for (int h = 0; h < board.nH; h++)  {
                        drawSquare(gc, v, h, squareWidth, squareHeight);

                        drawPiece   (gc, v, h, squareWidth, squareHeight);
                }
        }
}
// …
```

Использование абстрактных методов в абстрактном классе


```
public abstract class GameBoard extends Canvas implements PaintListener  {
// …
      private void drawPiece(GC gc, int v, int h, int squareWidth, int squareHeight) {
            Piece piece = board.getSquare(v, h).getPiece();
            if (piece == null) return;

            int dx = squareWidth  /8;
            int dy = squareHeight /8;

            int x = v * squareWidth  + dx;
            int y = h * squareHeight + dy;

            Image image = getPieceImage(piece);
            Rectangle bounds = image.getBounds();
            gc.drawImage(image,
                  0, 0, bounds.width, bounds.height,
                  x, y, squareWidth - 2*dx, squareHeight - 2*dy);
      }
// …
```

Проектирование классов

Классы интерфейса пользователя шахмат



Реализация некоторых абстрактных методов в абстрактном классе ***EuropeBoard***


```
public abstract class EuropeBoard extends GameBoard {
    public EuropeBoard (Composite composite, Board board) {
        super(composite, board);
    }

    @Override
    protected void drawBackground (GC gc, Rectangle area) {
        // Рисуем фон – деревянную доску.
        Image image = GameImages.woodMedium;
        Rectangle bounds = image.getBounds();

        gc.drawImage(image,
            0, 0, bounds .width, bounds.height,
            area.x, area.y, area.width, area.height);
    }
}
```

Реализация некоторых абстрактных методов в абстрактном классе **EuropeBoard**

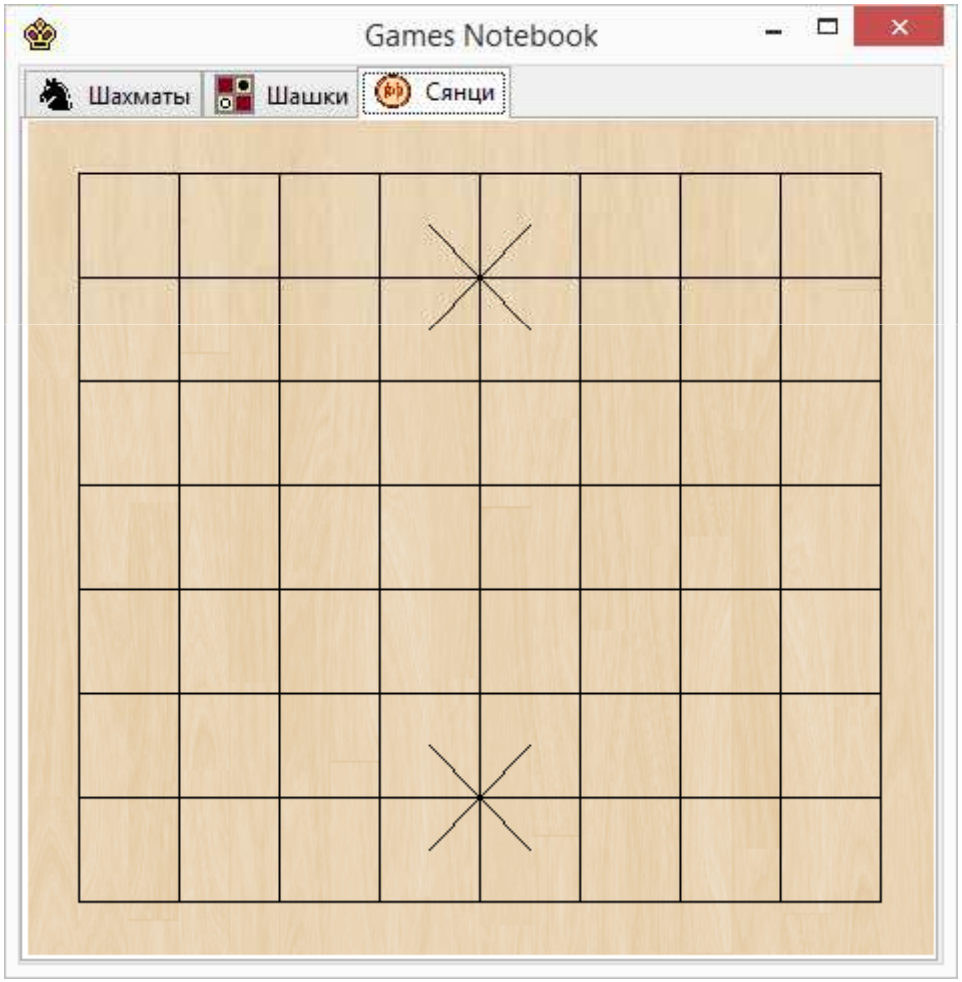

```
public abstract class EuropeBoard extends GameBoard {

    @Override
    public void drawSquare (GC gc, int v, int h, int squareWidth, int squareHeight) {
        boolean isWhiteSquare = ((v + h) % 2 == 0);

        Rectangle bounds = GameImages.woodDark.getBounds();

        if ( !isWhiteSquare )
            gc.drawImage(GameImages.woodDark,
                    0, 0, bounds.width, bounds.height,
                    v * squareWidth, h * squareHeight,
                    squareWidth, squareHeight);

        gc.setForeground( new Color(null, 0, 0, 0) );
        gc.drawRectangle(v * squareWidth, h * squareHeight, squareWidth, squareHeight);
    }
}
```

Реализация некоторых абстрактных методов в абстрактном классе ***AsiaBoard***


```
public abstract class AsiaBoard extends GameBoard {

    @Override
    protected void drawBackground (GC gc, Rectangle area) {
        // Рисуем фон – деревянную доску.
        Rectangle bounds = GameImages.woodLight.getBounds();

        gc.drawImage(GameImages.woodLight,
                     0, 0, bounds .width, bounds.height,
                     area.x, area.y, area.width, area.height);
    }

    @Override
    public void drawSquare (GC gc, int v, int h, int squareWidth, int squareHeight)
    {
        // Рисуем клетку – пересечение линий в центре клетки
        // ...
    }

}
```

Блокнот игр. Доска для игры в азиатские игры *Рендзю, Го, Сянци (китайские шахматы)*

МГУ им. М.В.Ломоносова. Факультет ВМК. Романов Владимир Юрьевич ©2017



Класс *AbstractBoardPanel*. Базовый класс для досок всех игр


```
public abstract class AbstractBoardPanel extends Canvas
        implements PaintListener, MouseListener, Observer
{

//…

protected Board board;

public AbstractBoardPanel(Composite parent, int v, int h) {
    super(parent, SWT.TRANSPARENT);

    nV = v;
    nH = h;
    board = new Board (nV, nH);

    board.addObserver( this );

//…

}
```

Класс *AbstractBoardPanel*. Базовый класс для досок всех игр


```
public abstract class AbstractBoardPanel extends Canvas
        implements PaintListener, MouseListener, Observer
{
//…

protected Board board;

public AbstractBoardPanel(Composite parent, int v, int h) {
    super(parent, SWT.TRANSPARENT);

    nV = v;
    nH = h;
    board = new Board (nV, nH);

    board.addObserver( this );

//…
}

//…
```

Класс *AbstractBoardPanel*. Абстрактные методы реализуемые в потомках


```
//...

/**
 * Отрисовать клетку.
 */
protected abstract void drawSquare(GC gc, int v, int h, int sw, int sh);

/**
 * Отрисовать доску.
 */
protected abstract void drawBoard(GC gc, Rectangle clientArea);

/**
 * Получить изображение для заданной фигуры
 */
protected abstract Image getImage(Piece piece);

//...
```

Класс *AbstractBoardPanel*. Рисование доски, клеток, фигур, последнего хода.


```
@Override
public void paintControl (PaintEvent e) {
        Rectangle clientArea = getClientArea();
        int squareWidth  = clientArea.width  / nV;
        int squareHeight = clientArea.height / nH;

        GC gc = e.gc;

        drawBoard (gc, clientArea);

        for (int v = 0; v < nV; v++)
                for (int h = 0; h < nH; h++) {
                        drawSquare (gc, v, h, squareWidth, squareHeight);
                        drawPiece (gc, v, h, squareWidth, squareHeight);
                }

        markLastMove (gc);
}
```

Рисование доски

Рисование клетки

Рисование фигуры

Рисование последнего хода

Класс *AbstractBoardPanel*. Рисование фигуры


```
    private void drawPiece (GC gc, int v, int h, int sw, int sh) {
        if (board.isEmpty(v, h))
            return;

        Square square = board.getSquare(v, h);
        Piece piece = square.getPiece();

        Image pieceImage = getImage (piece);

        Rectangle bounds = pieceImage.getBounds();
        int iw = bounds.width;
        int ih = bounds.height;

        gc.drawImage (pieceImage, 0, 0, iw, ih, v * sw, h * sh, sw, sh);
    }
}
```

Вызов абстрактного метода

Реализация некоторых абстрактных методов в классе EuropeanBoard


```
/**
 * Двухцветная деревянная доска.
*/
public abstract class EuropeanBoard extends AbstractBoardPanel {
        @Override
        protected void drawBoard (GC gc, Rectangle clientArea) {}

        @Override
        protected void drawSquare (GC gc, int v, int h, int sw, int sh) {
                boolean isWhite = ((v + h) % 2 == 0);

                Image squareImage = isWhite
                        ? GameImages.imageWoodMedium
                        : GameImages.imageWoodDark;

                gc.drawImage(squareImage, 0, 0, sw, sh, v*sw, h*sh, sw, sh);

                gc.setForeground(COLOR_BLACK);
                gc.drawRectangle(v * sw, h * sh, sw, sh);
        }
}
```

Реализация абстрактного метода

Класс *ChessBoard*. Размещение фигур (1)


```
/**
 * Доска с расположенными на ней шахматными фигурами.
*/
public class ChessBoardPanel extends EuropeanBoard {
	public ChessBoardPanel(Composite parent) {
		super(parent, 8, 8);

		Chess.putPieces (board, PieceColor.WHITE);
		Chess.putPieces (board, PieceColor.BLACK);
	}

	@Override
	public Image getImage (Piece piece) {
		return getChessPieceImage(piece);
	}

//...
```

Класс *ChessBoard*. Размещение фигур


```
public class Chess extends Game {
    static public void putPieces (Board board, PieceColor color) {
        int nV = board.getWidth();
        int nH = board.getHeight();

        int hPawns = (color == PieceColor.WHITE ? nH-2 : 1);
        int hPiece  = (color == PieceColor.WHITE ? nH-1 : 0);

        for (int v = 0; v < nV; v++)
            board.putPiece(v, hPawns, new Pawn (color));

        board.putPiece(0, hPiece, new Rook(color));
        board.putPiece(1, hPiece, new Knight(color));
        board.putPiece(2, hPiece, new Bishop(color));
        board.putPiece(3, hPiece, new Queen(color));
        board.putPiece(4, hPiece, new King(color));
        board.putPiece(5, hPiece, new Bishop(color));
        board.putPiece(6, hPiece, new Knight(color));
        board.putPiece(7, hPiece, new Rook(color));
    }
}
```

Расположение пешек

Перечисление *PieceColor* - цвет фигуры


```
package game.core;

public enum  PieceColor {
        WHITE,
        BLACK,
        GREEN,
        BLUE
}
```

Класс *Piece* – *базовый класс для всех клеточных игр.*


```
abstract public class Piece {
        public PieceColor color = PieceColor.WHITE;

        protected Square square;

        public Piece(PieceColor color) {
                this.color = color;
        }

        /**
         * Допустим ли ход фигурой на заданную клетку.
         */
        abstract public boolean isCorrectMove (Square s);

        /**
         * Передвигуть фигуру на заданную клетку.
         */
        abstract public Move moveTo (Square s);
```

Класс ChessPiece – *базовый класс для всех клеточных игр.*


```
/**
 * Базовый класс для всех шахматных фигур.
*/
public abstract class ChessPiece extends Piece {
	public ChessPiece(PieceColor color) {
		super(color);
	}

	/*
	 * Общее свойство шахматных фигур - нельзя бить фигуру своего цвета.
	*/
	@Override
	public boolean isCorrectMove (Square s) {
		if (s.isEmpty()) return true;

		// Фигуры разного цвета?
		return s.getPiece().color != color;
	}


}
```

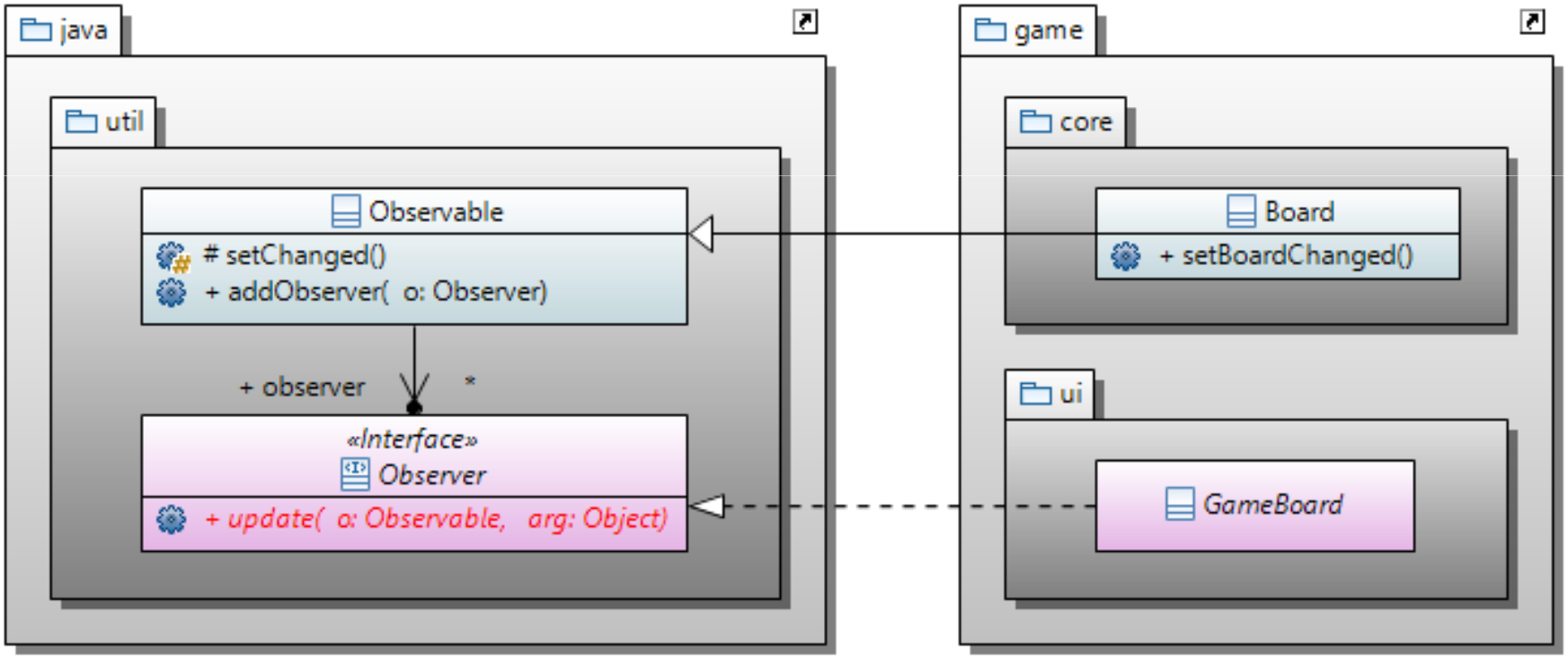
Класс *Pawn - пешка.* Реализация абстрактных методов


```
public class Pawn extends ChessPiece {
        public Pawn(PieceColor color) {
            super(color);
        }
        @Override
        public boolean isCorrectMove (Square target) {
            // ...
        }

        @Override
        public Move moveTo (Square target) {
            // ...
        }

        @Override
        public String toString() { return ""; }
```

Блокнот игр. Шахматные фигуры

Класс *Observable*
и
интерфейс *Observer*

Использование класса ***Observable*** и интерфейса ***Observer***



Обозреваемый класс **Board**.


```
public class Board extends Observable {

// ...

        public void setBoardChanged ( ) {
                super.setChanged();
        }

// ...

}
```

Можно добавлять обозревателей

Изменения произошли. **Уведомляем обозревателей**

Класс-обозреватель **GameBoard**


```
public abstract class GameBoard extends Canvas
        implements PaintListener, MouseListener, Observer
{
    public GameBoard(Composite parent, Board board) {
        super(parent, SWT.NONE);

        this.board = board;
        addPaintListener(this);

        addMouseListener( this );
        board.addObserver( this );
        board.setBoardChanged();
    }

    @Override
    public void update (Observable o, Object arg) {
        update();
        redraw();
    }
```

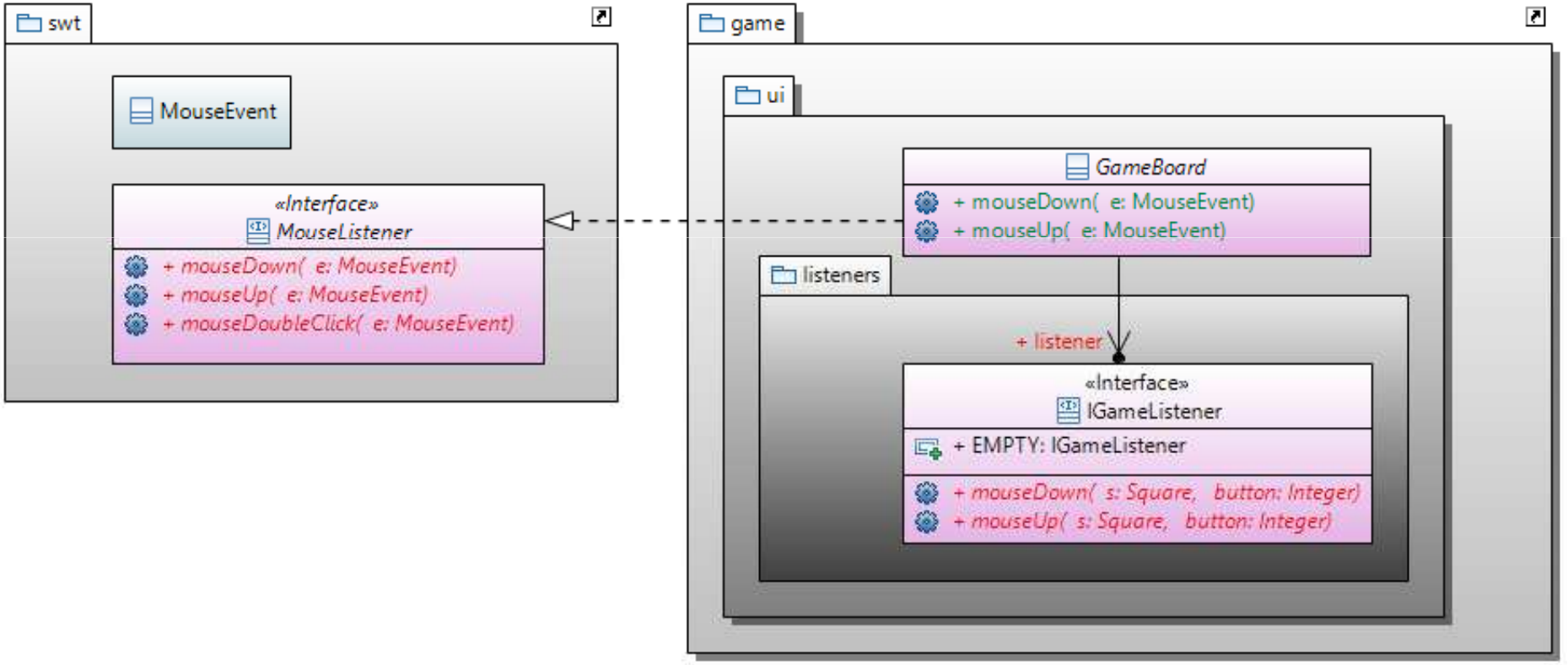
Класс **GameBoard** будет слушать изменения на **доске**

Класс **GameBoard** добавил себя как слушателя изменения на **доске**

Действия при изменении на **доске**

Программирование взаимодействия с пользователем

Использование класса ***Observable*** и интерфейса ***Observer***



SWT. Программирование клавиш мыши. Перемещение фигуры на доске


```
public abstract class GameBoard extends Canvas
        implements PaintListener, MouseListener, Observer
{
    protected Board board;

    public GameBoard(Composite parent, Board board) {
        super(parent, SWT.NONE);

        this.board = board;

        addPaintListener( this );
        addMouseListener( this );
    }
```

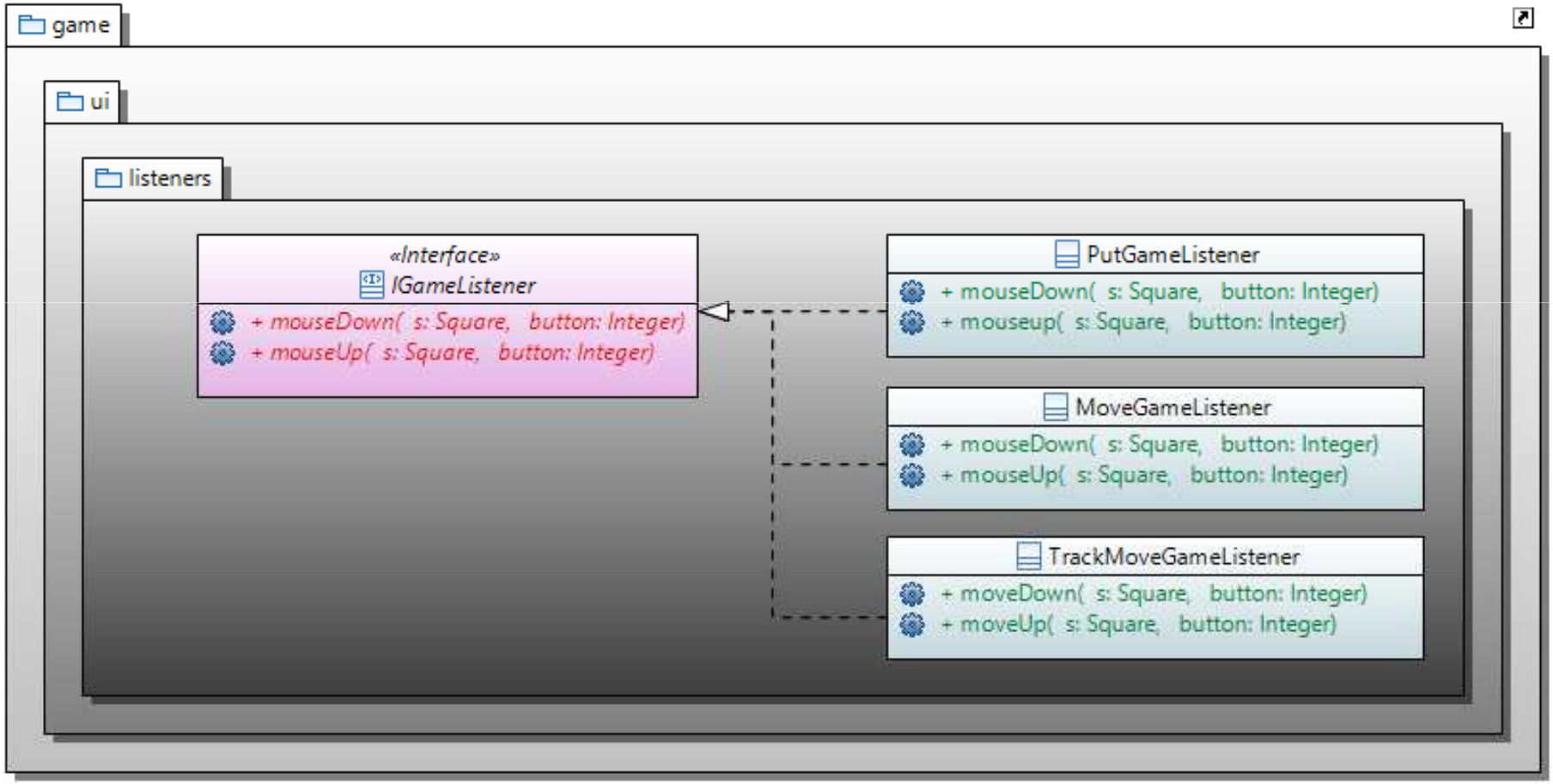
Класс **GameBoard** будет слушать клавиши мыши

Класс **GameBoard** добавил себя как слушателя клавиш мыши

Блокнот игр. Реакция на события от мыши


```
public abstract class GameBoard extends Canvas
        implements PaintListener, MouseListener, Observer  {
// ...
    protected IGameListner listener = IGameListner.EMPTY;

    @Override
    public void mouseDown(MouseEvent e) {
        Square s = getSquare(e);
        listener.mouseDown(s, e.button);
    }

    @Override
    public void mouseUp(MouseEvent e) {
        Square s = getSquare(e);
        listener.mouseUp(s, e.button);
    }

   @Override
   public void mouseDoubleClick(MouseEvent e) {}
// ...
```

Слушатели ходов на доске



Класс *PutPieceListener*.
Фигуры игры ставятся на доску


```
abstract
public class PutPieceListener implements IGameListner {

        private PieceColor moveColor = PieceColor.WHITE;

        private Board board;

        private GameBoard panel;

        public PutPieceListener(GameBoard panel) {
                this.board = panel.board;
                this.panel = panel;
        }

        @Override
        public void mouseUp(Square s, int button) {}

// ...
```

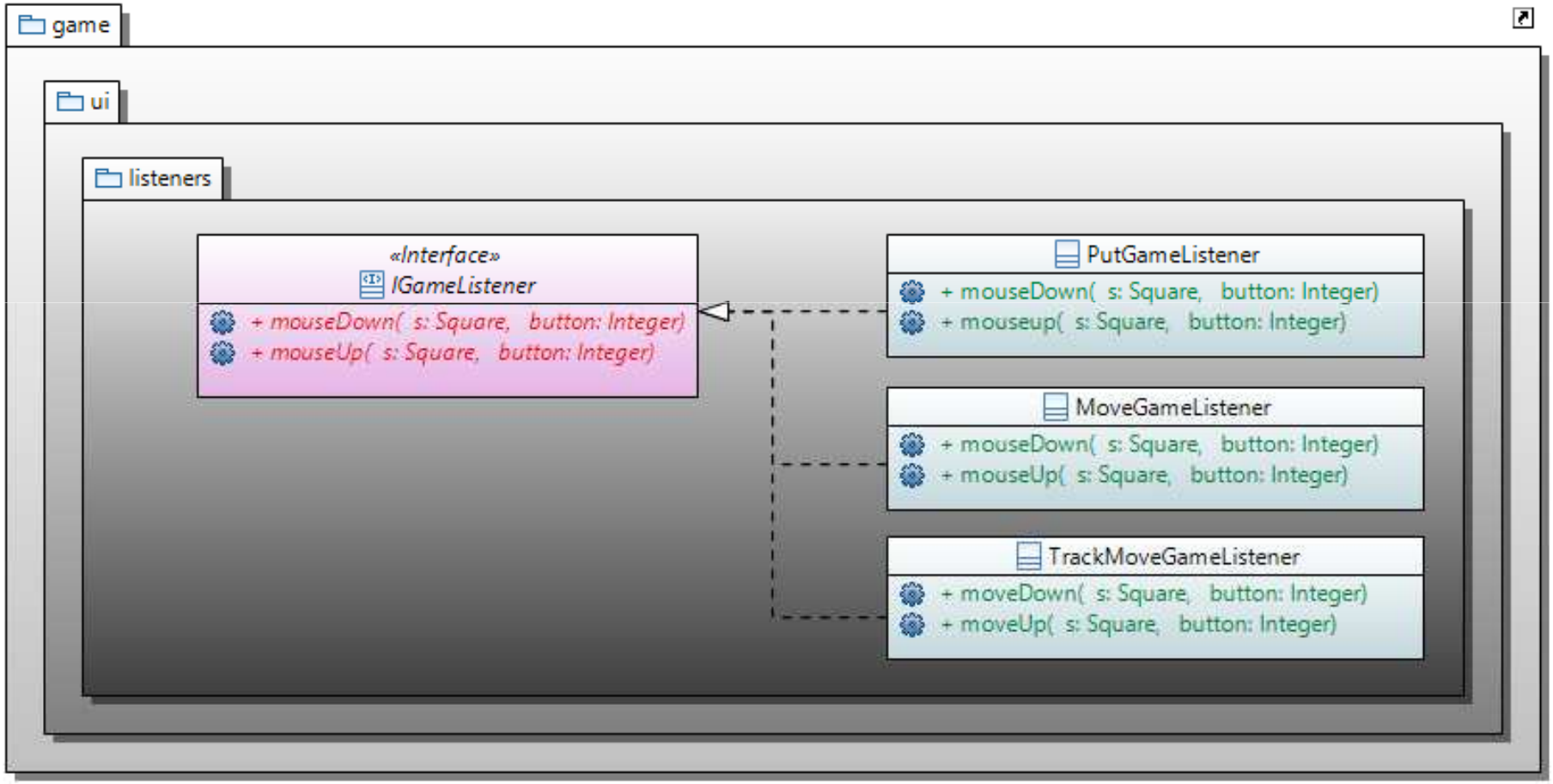
Класс *PutPieceListener*.
Фигуры игры ставятся на доску


```
        @Override
        public void mouseDown(Square mouseSquare, int button) {
                if (! mouseSquare.isEmpty()) return;

                Piece piece = getPiece (mouseSquare, moveColor);

                if  (! piece.isCorrectMove  (mouseSquare)) {
                        piece.remove(); return;
                }

                Move move = piece.makeMove (mouseSquare);
                move.doMove();
                board.history.addMove(move);

                moveColor = getOponentColor(moveColor);
                panel.imageToCursor( getPieceImage(piece, moveColor) );

                board.setBoardChanged();
                panel.redraw();
        }

        abstract public Piece getPiece(Square square, PieceColor color);
//…
```

Использование класса *PutPieceListener* для установки фигур игры Реверси


```
public class ReversiBoardPanel extends GreenBoard {

    public ReversiBoardPanel(Composite composite, int nHoles) {
        super(composite, Reversi.getInitBoard(nHoles) );

        listener = new PutPieceListener (this) {
            @Override
            public Image getPieceImage (Piece piece, PieceColor color) {
                return ReversiBoardPanel.this.getPieceImage(piece, color);
            }

            @Override
            public Piece getPiece (Square square, PieceColor color) {
                return new Stone(square, color);
            }
        };
    }

    // ...
```

Слушатели ходов на доске



Класс *MovePieceListener*.
Фигура перемещается на одно поле


```
abstract
public class MovePieceListener implements IGameListner {
    private Piece selectedPiece;

    private Square selectedSquare;

    private Cursor savedCursor;

    private PieceColor moveColor = PieceColor.WHITE;

    private Board board;

    private GameBoard panel;

    public MovePieceListener (GameBoard panel) {
        this.board = panel.board;
        this.panel = panel;
    }

    // ...
```

Класс *MovePieceListener*.
Установка фигуры на новое поле


```
@Override
public void mouseDown(Square mouseSquare, int button) {
    if ( mouseSquare.isEmpty())
        return;

    selectedPiece = mouseSquare.getPiece();
    if (selectedPiece.getColor() != moveColor)
        return;

    savedCursor = panel.getCursor();

    selectedSquare = mouseSquare;
    selectedSquare.removePiece();

    panel.imageToCursor( getPieceImage(selectedPiece, moveColor) );

    board.setBoardChanged();
    panel.redraw();
}
```

Класс *MovePieceListener*.
Выбор фигуры для перемещения


```
@Override
public void mouseUp(Square mouseSquare, int button) {
        if (selectedSquare == null) return;

        selectedSquare.setPiece(selectedPiece);

        if ( selectedPiece.isCorrectMove (mouseSquare) ) {
                Move move = selectedPiece.makeMove (selectedSquare, mouseSquare);
                move.doMove();
                board.history.addMove(move);
                moveColor = getOponentColor();
        }

        selectedPiece = null; selectedSquare = null;

        panel.setCursor(savedCursor);

        board.setBoardChanged();
        panel.redraw();
}
```

Класс *MovePieceListener*.
Выбор фигуры для перемещения


```
@Override
public void mouseUp(Square mouseSquare, int button) {
        if (selectedSquare == null) return;

        selectedSquare.setPiece(selectedPiece);

        if ( selectedPiece.isCorrectMove (mouseSquare) ) {
                Move move = selectedPiece.makeMove (selectedSquare, mouseSquare);
                move.doMove();
                board.history.addMove(move);
                moveColor = getOponentColor();
        }

        selectedPiece = null; selectedSquare = null;

        panel.setCursor(savedCursor);

        board.setBoardChanged();
        panel.redraw();
}
```

Использование класса *MovePieceListener* для перемещения фигур игры Шахматы


```
public class ChessBoardPanel extends EuropeBoard {

// ...

        public ChessBoardPanel  (Composite composite) {
                super(composite, Chess.getInitBoard());

                listener = new MovePieceListener (this) {
                                @Override
                                public Image getPieceImage (Piece piece, PieceColor color) {
                                        return ChessBoardPanel.this.getPieceImage(piece, color);
                                }
                };
        }

// ...
```