

# Язык Java и разработка Java-приложений

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

# Литература

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

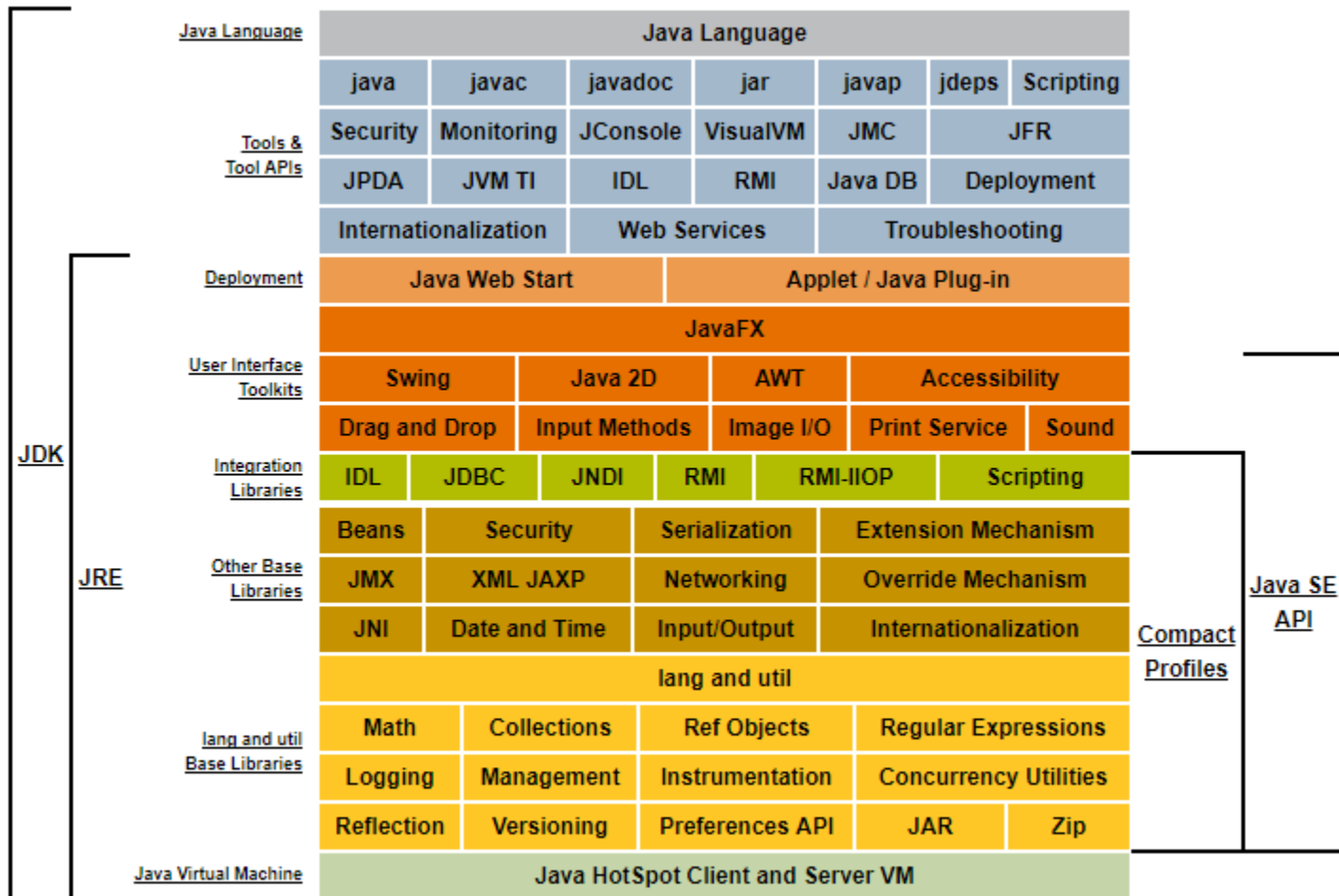
1. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley  
Java(TM) Programming Language, Java SE 8 Edition  
**<https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf>**
2. Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley  
The Java™ Virtual Machine Specification, Java SE 8 Edition  
**<https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf>**
3. Eclipse web sites: **<http://www.eclipse.org>**
4. Oracle web site:  
**<https://docs.oracle.com/javase/8/docs/>**
5. Брюс Эккель. Философия Java. 4-е издание.  
Издательство «Питер». Петербург 2016.  
ISBN 5-88782-105-1

# Документация по Java

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

<https://docs.oracle.com/javase/8/docs/>



# Цели курса

- Изучение языка Java перед изучением технологий основанных на языке Java
- Изучение среды разработки Eclipse распространяемой фирмой IBM в исходных текстах
- Подготовка к изучению курса:  
“Разработка объектно-ориентированных систем программирования в среде Eclipse”

О курсе

## “Разработка объектно-ориентированных систем программирования в среде Eclipse”

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Разработка расширений (plug-in) для среды Eclipse
- Разработка расширений среды для объектно-ориентированных систем программирования
- Разработка распознавателей для объектно-ориентированных языков программирования
- Разработка промежуточного представления компилятора с помощью Eclipse Modeling Framework

# О курсе “Разработка объектно-ориентированных систем программирования в среде Eclipse”

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Визуализация промежуточного представления компилятора с помощью *Graphic Editing Framework*
- Моделирование программ с помощью реализованной в среде Eclipse *метамодел* языка *UML 2.0*
- Генерация кода для виртуальной машины Java
- Разработка в среде Eclipse отладчиков для языков программирования

# Язык программирования Java

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

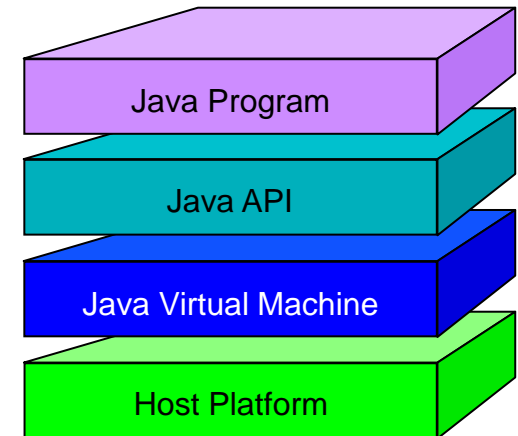
- Простота
- Язык высокого уровня
- Объектно-ориентированный
- Независим от архитектуры
- Переносим в исходных текстах и в двоичном коде
- Интерпретируемый и компилируемый
- Многопоточный

# Платформа Java

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Платформа – окружение разработки где работает программа
- Java – платформа работает на разных операционных системах
- Java – платформа состоит из виртуальной машины Java и интерфейса приложений





# Компилятор и виртуальная машина Java

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



# Установка Eclipse. Версии среды

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Среда Eclipse бесплатна
- Дистрибутив Eclipse загружается с сайта:  
<https://www.eclipse.org/downloads/packages/release/2020-06/r>

## Eclipse Modeling Tools

444 MB

9,497 DOWNLOADS



The Modeling package provides tools and runtimes for building model-based applications. You can use it to graphically design domain models, to leverage those models at design time by creating and editing dynamic instances, to collaborate via Eclipse's team support with facilities for comparing and merging models and model instances structurally, and finally to generate Java code from those models to produce complete applications. In addition, via the package's discover catalog, you can easily install a wide range of additional powerful, model-based tools and runtimes to suit your specific needs.



Windows x86\_64

macOS x86\_64

Linux x86\_64

# Установка Eclipse. Eclipse Modeling Tools

МГУ ИМ



## Eclipse IDE for JavaScript and Web Developers

156 MB - Downloaded 29,419 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**



## Eclipse IDE for Java and DSL Developers

309 MB - Downloaded 19,275 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**



## Eclipse for Parallel Application Developers

227 MB - Downloaded 18,259 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**



## Eclipse for RCP and RAP Developers

254 MB - Downloaded 13,260 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**



## Eclipse Modeling Tools

392 MB - Downloaded 8,603 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**



## Eclipse IDE for Java and Report Developers

315 MB - Downloaded 4,941 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**



## Eclipse for Testers

131 MB - Downloaded 4,115 Times

Windows **32-bit 64-bit**  
Mac Cocoa **64-bit**  
Linux **32-bit 64-bit**

# Eclipse Modeling Tools

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

<https://www.eclipse.org/downloads/packages/release/2020-06/r>

## Eclipse Modeling Tools

444 MB 9,578 DOWNLOADS



The Modeling package provides tools and runtimes for building model-based applications. You can use it to graphically design domain models, to leverage those models at design time by creating and editing dynamic instances, to collaborate via Eclipse's team support with facilities for comparing and merging models and model instances structurally, and finally to generate Java code from those models to produce complete applications. In addition, via the package's discover catalog, you can easily install a wide range of additional powerful, model-based tools and runtimes to suit your specific needs.



Windows x86\_64  
macOS x86\_64  
Linux x86\_64



# IntelliJ IDEA Community Edition

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

<https://www.jetbrains.com/idea/download/#section=windows>

## Download IntelliJ IDEA

Windows

macOS

Linux

### Ultimate

For web and enterprise development

Download

.exe



Free 30-day trial

### Community

For JVM and Android development

Download

.exe



Free, built on open source

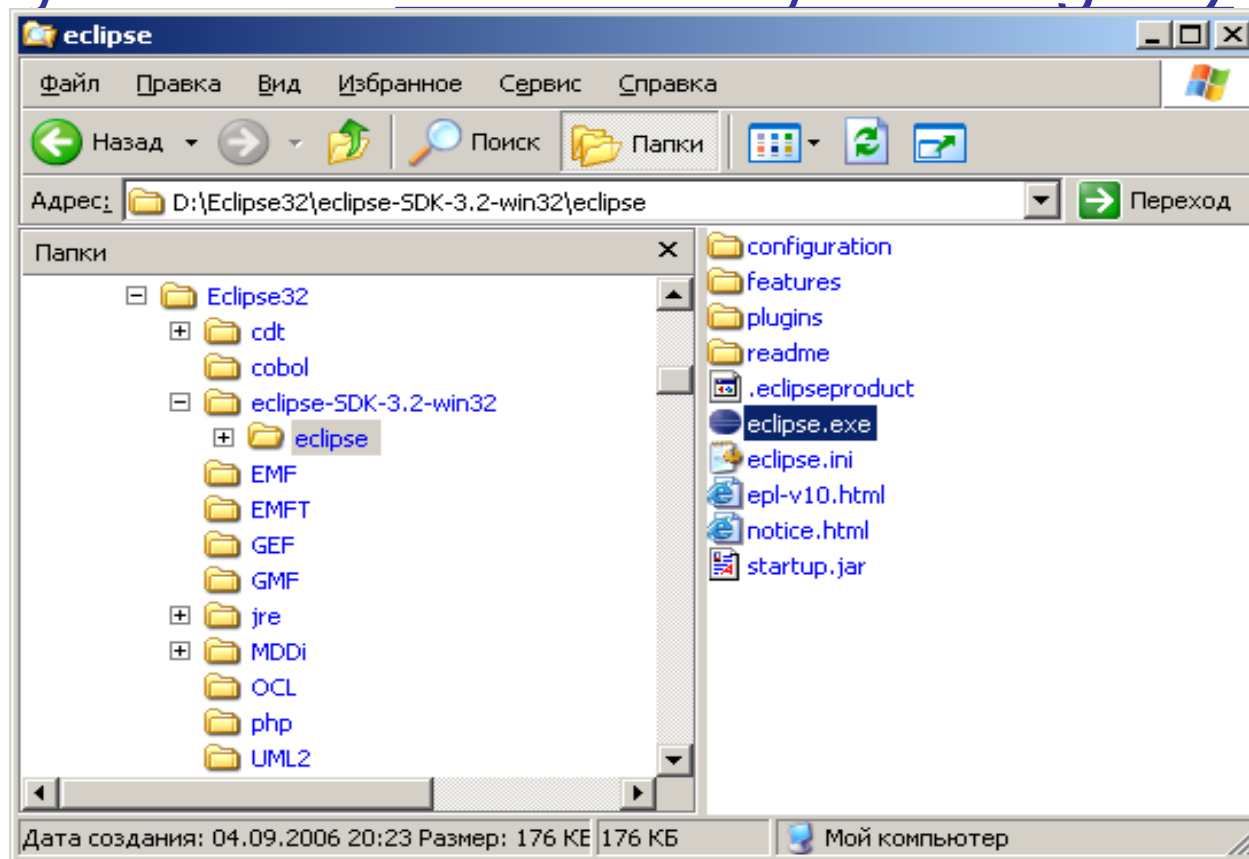
# Краткий обзор среды Eclipse для разработки на языке Java

# Запуск среды Eclipse на исполнение

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Установка – просто распаковка Zip-архива.  
При установке не используется *registry*



# Обзор среды Eclipse

- Перспективы (Perspectives)
- Виды (Views)
- Редакторы (Editors)
- Работа с файлами
- Навигация по рабочему месту (Workbench)
- Консоль (Console)

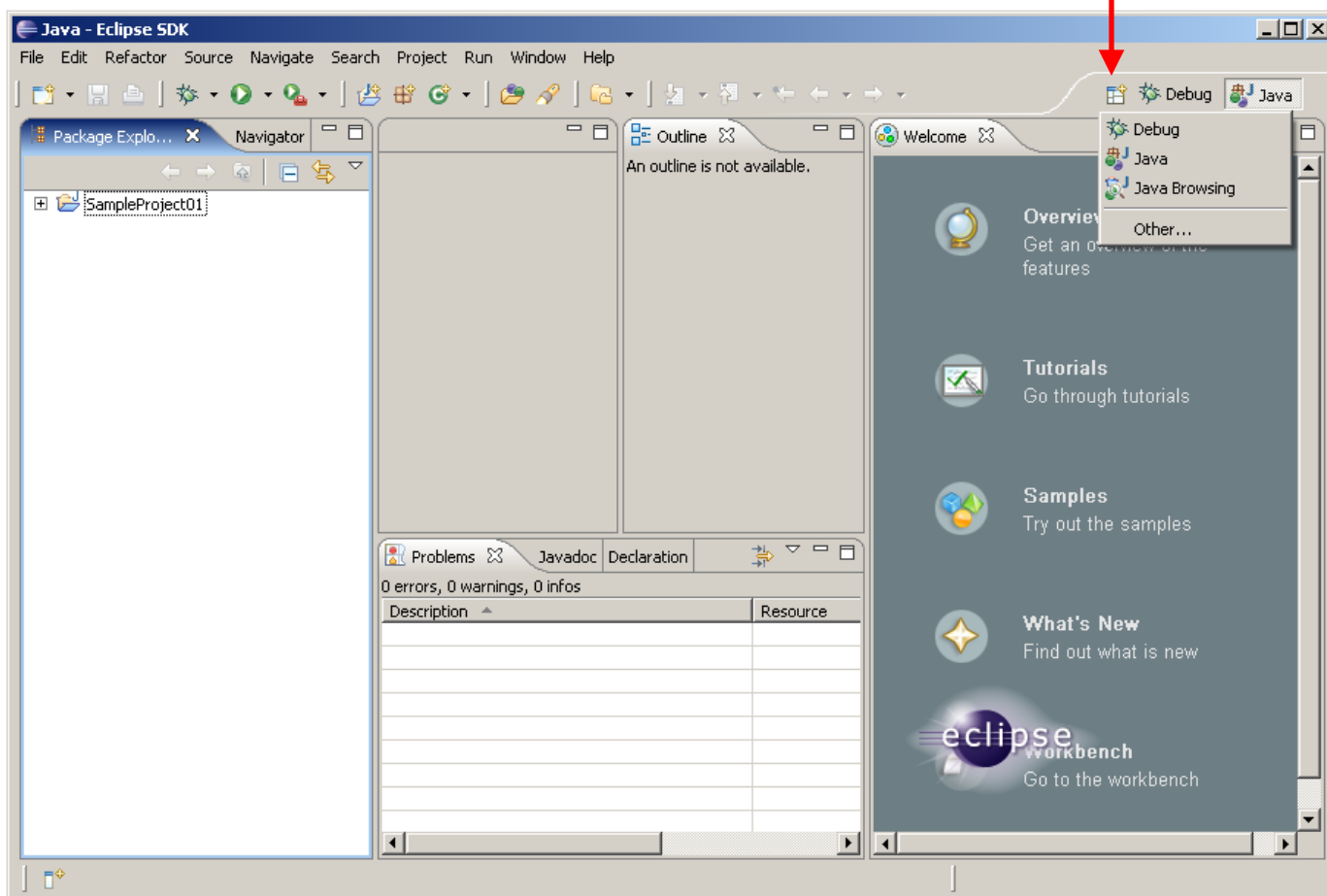


# Обзор среды Eclipse. Перспективы (Java)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

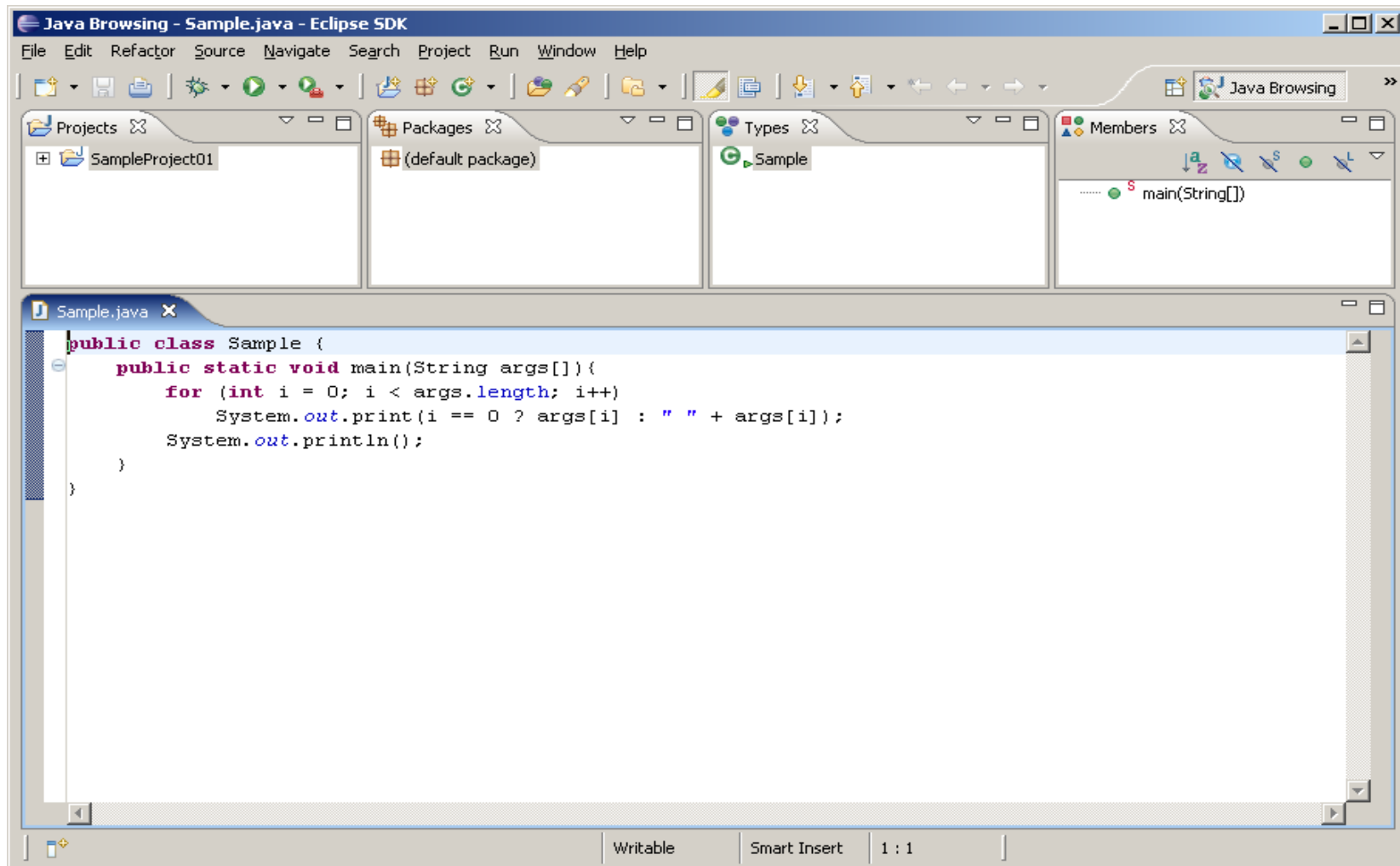
- На рабочем месте (workbench) может быть открыто несколько перспектив
- Различные перспективы могут быть выбраны из панели перспектив



# Обзор среды Eclipse. Перспективы (Java Browsing)

МГУ им. М.В.Ломоносова. Факультет ВМК.

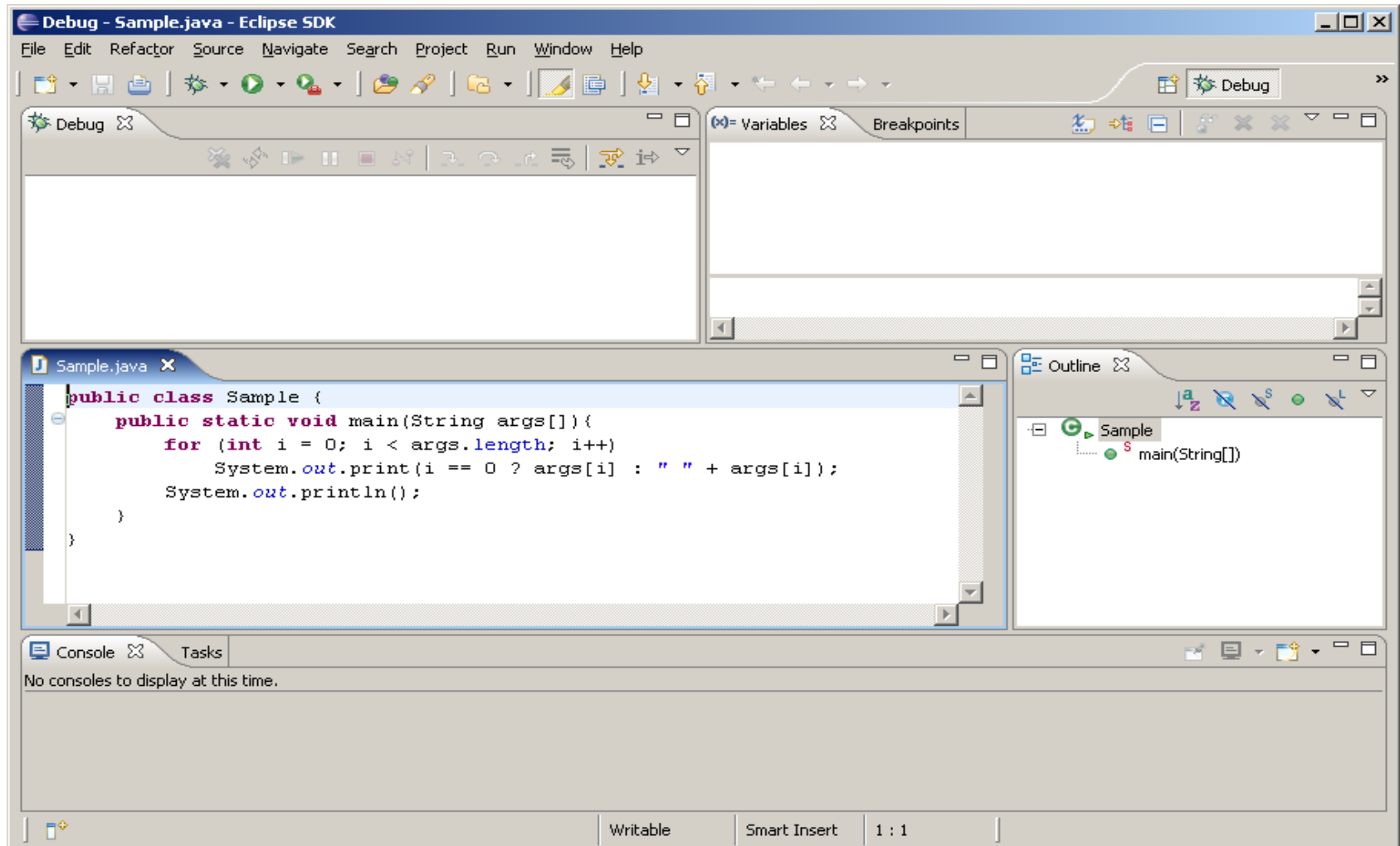
Романов Владимир Юрьевич ©2024



# Обзор среды Eclipse. Перспективы (Debug)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



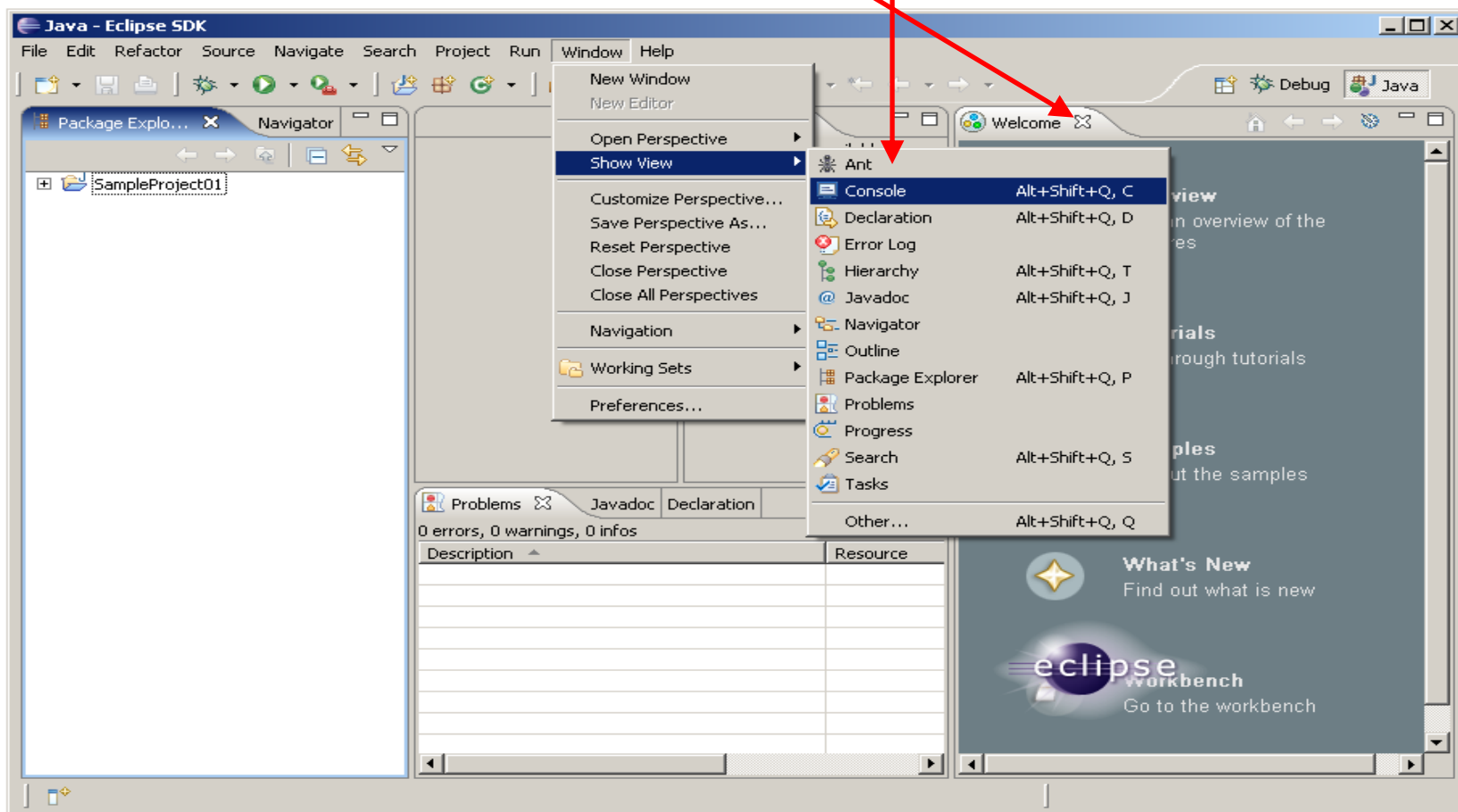
# Обзор среды Eclipse.

## Добавление видов к перспективам

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Добавление к перспективе нового вида (Console)
- Удаление вида Welcome из перспективы



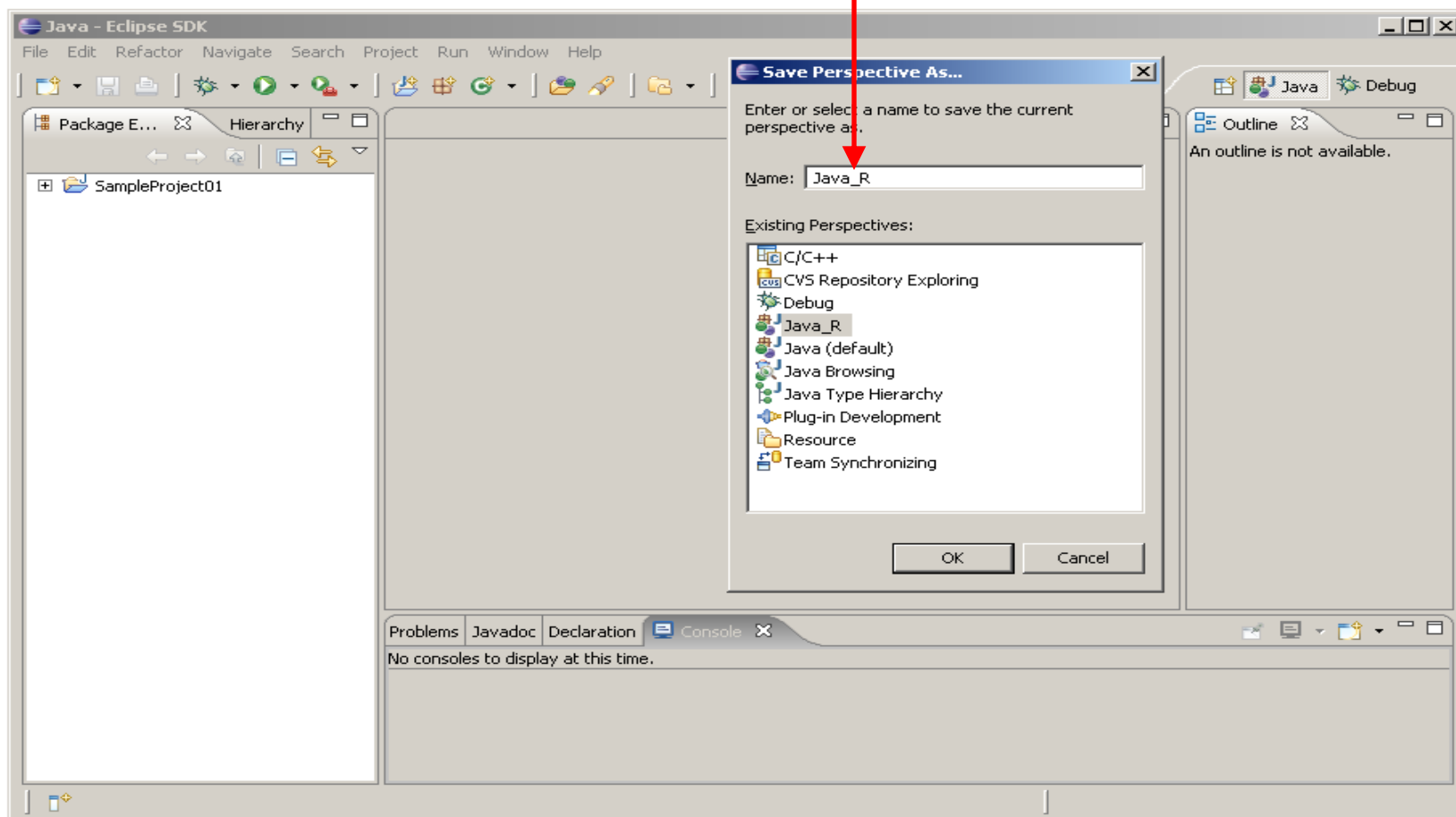
# Обзор среды Eclipse.

## Сохранение перспективы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Сохранение перспективы под новым именем



# Обзор среды Eclipse.

## Виды

МГУ им. М.В.Ломоносова. Факультет ВМК.

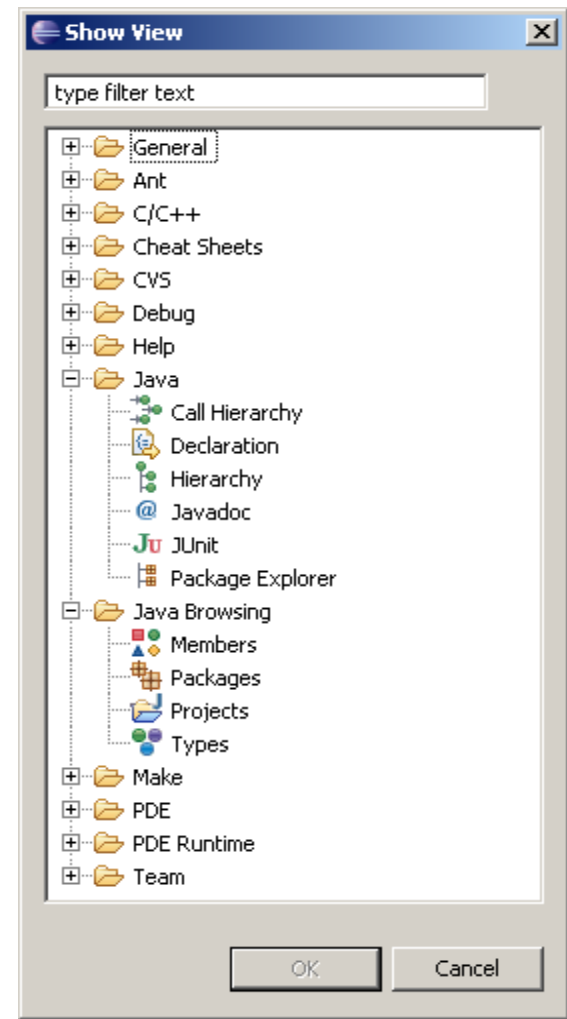
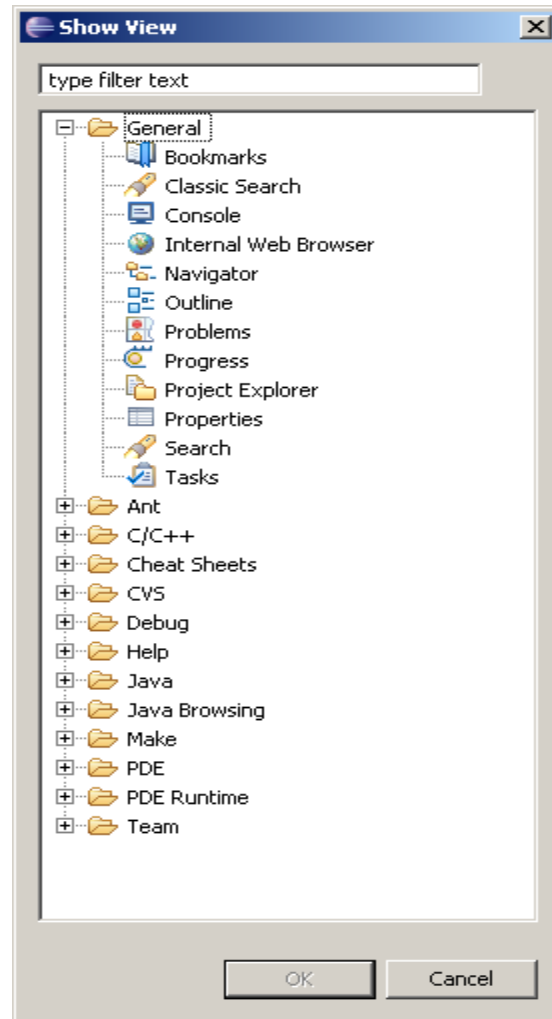
Романов Владимир Юрьевич ©2024

### Java Perspective

- Declaration
- Javadoc
- Outline
- Package Explorer
- Problems
- Welcome

### Debug Perspective

- Console
- Debug
- Outline
- Tasks
- Variables
- Welcome

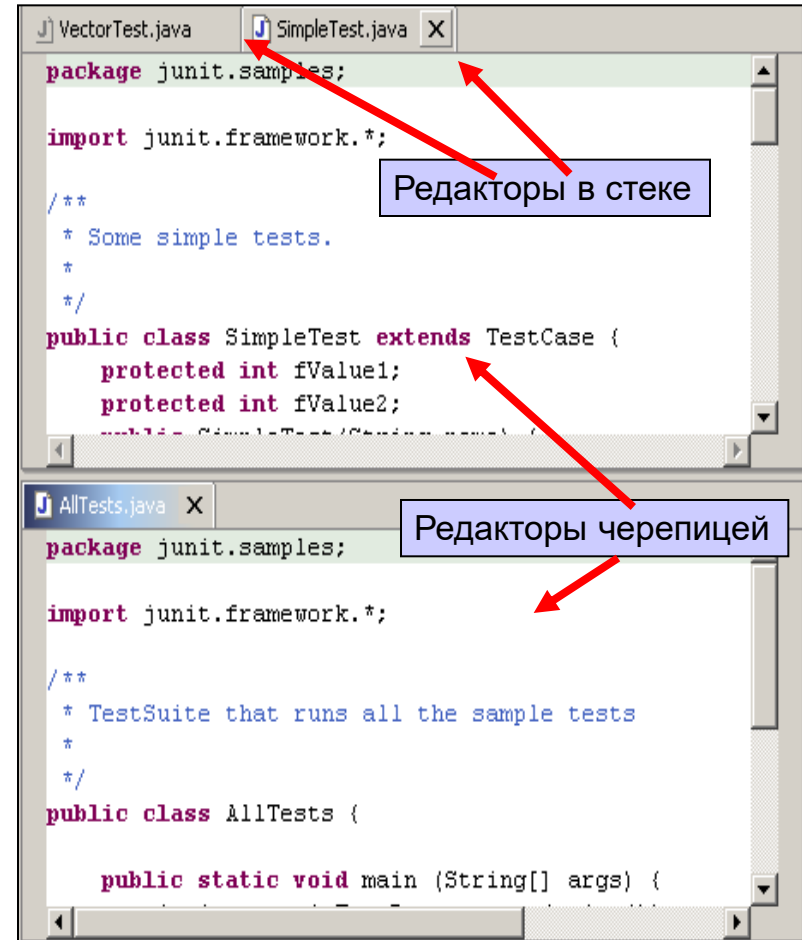


# Обзор среды Eclipse. Редакторы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Редакторы могут быть открыты для большинства ресурсов
- Несколько редакторов может быть открыто в стеке
- Если содержимое редактора было модифицировано, но не сохранено, то перед именем файла \*
- Если редактор активен, то в меню и панели инструментов содержатся операции применимые к активному редактору



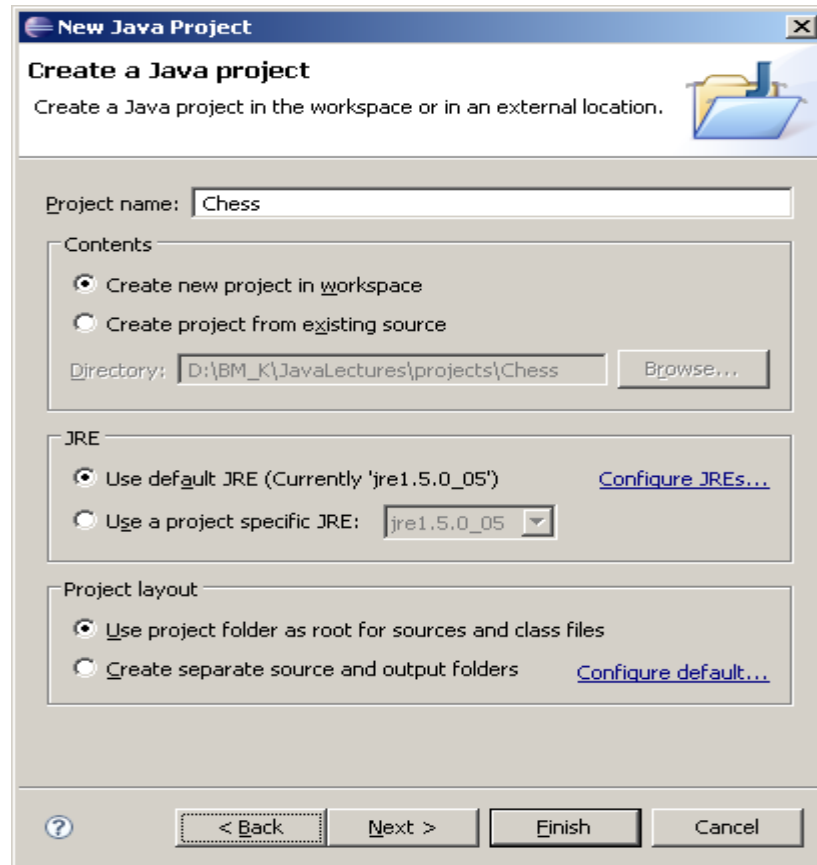
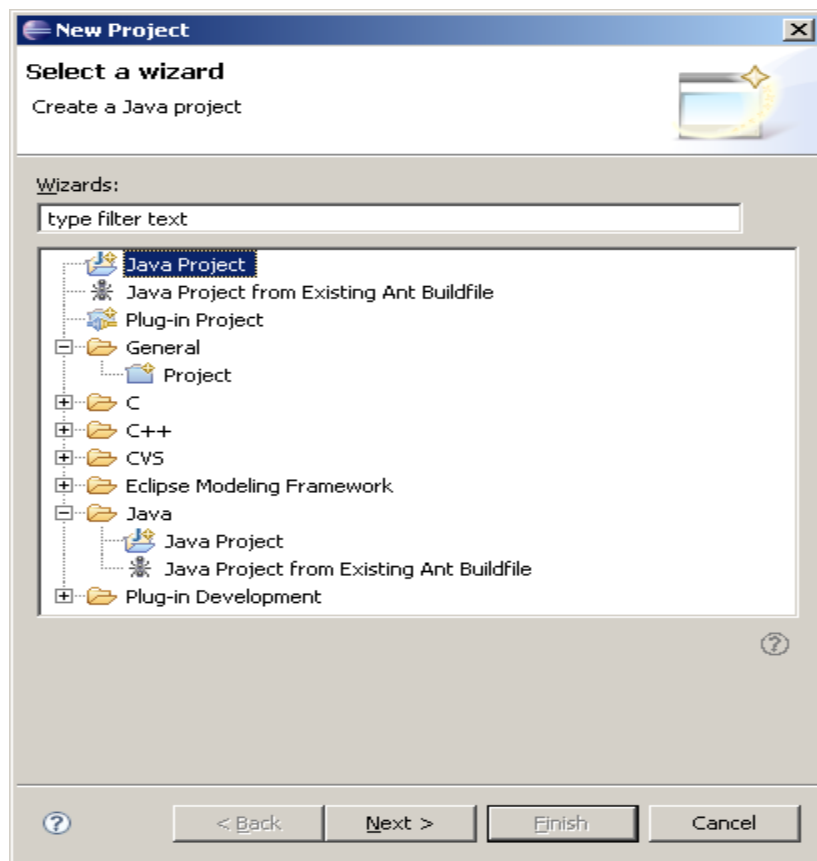
# Обзор среды Eclipse.

## Создание нового проекта

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- File | New | Project



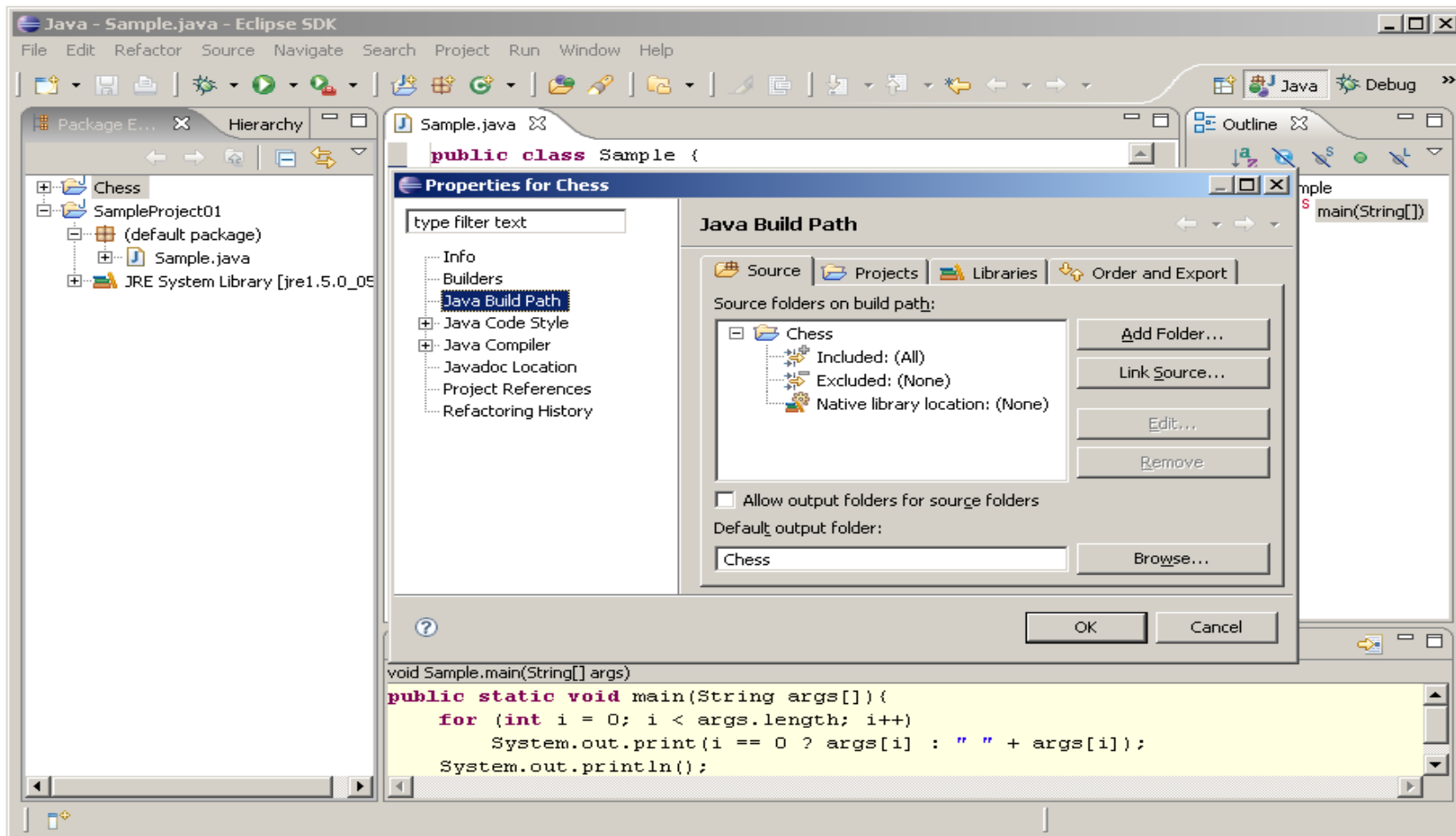


# Обзор среды Eclipse. Свойства

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Все ресурсы имеют свойства.

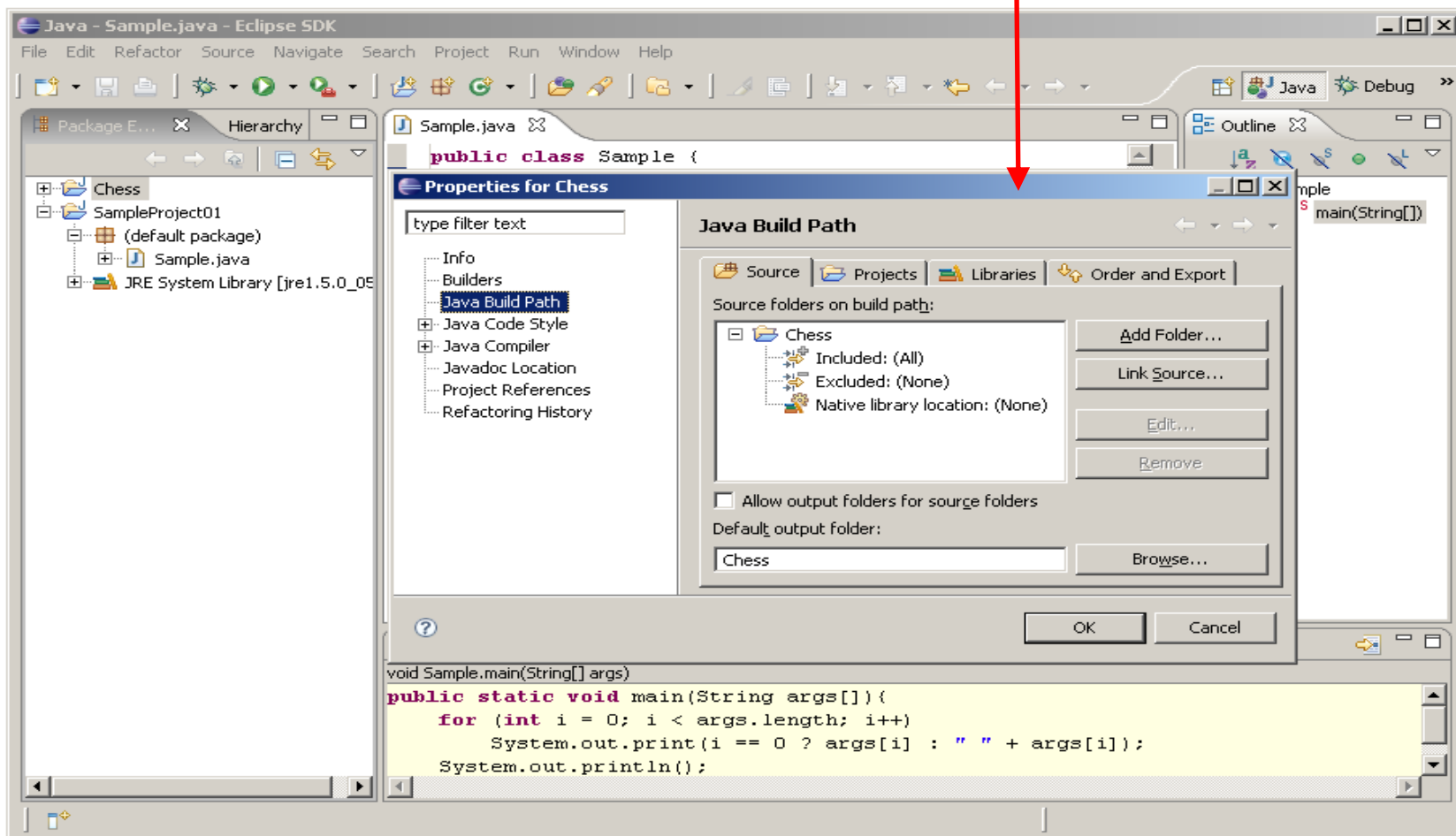


# Обзор среды Eclipse. Свойства

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Все ресурсы имеют свойства. Свойства проекта *Chess*:



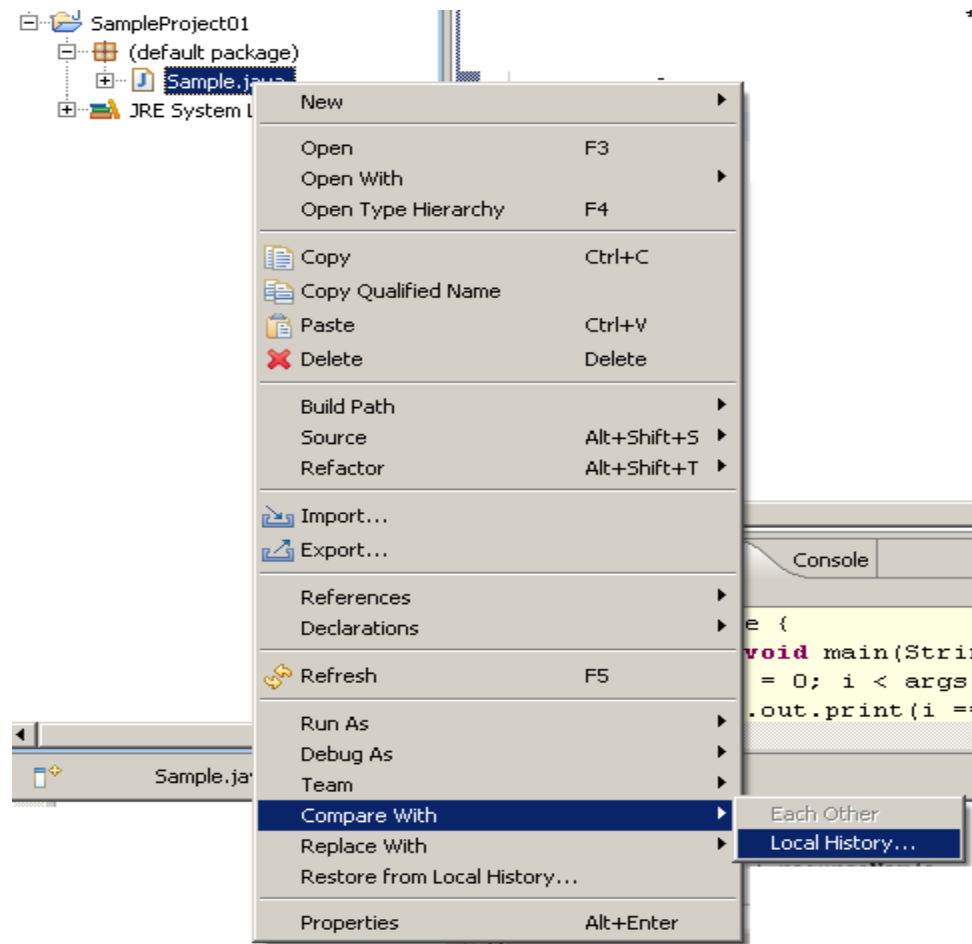
# Обзор среды Eclipse.

## Локальная история (1)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Для каждого файла хранится *локальная история* файла



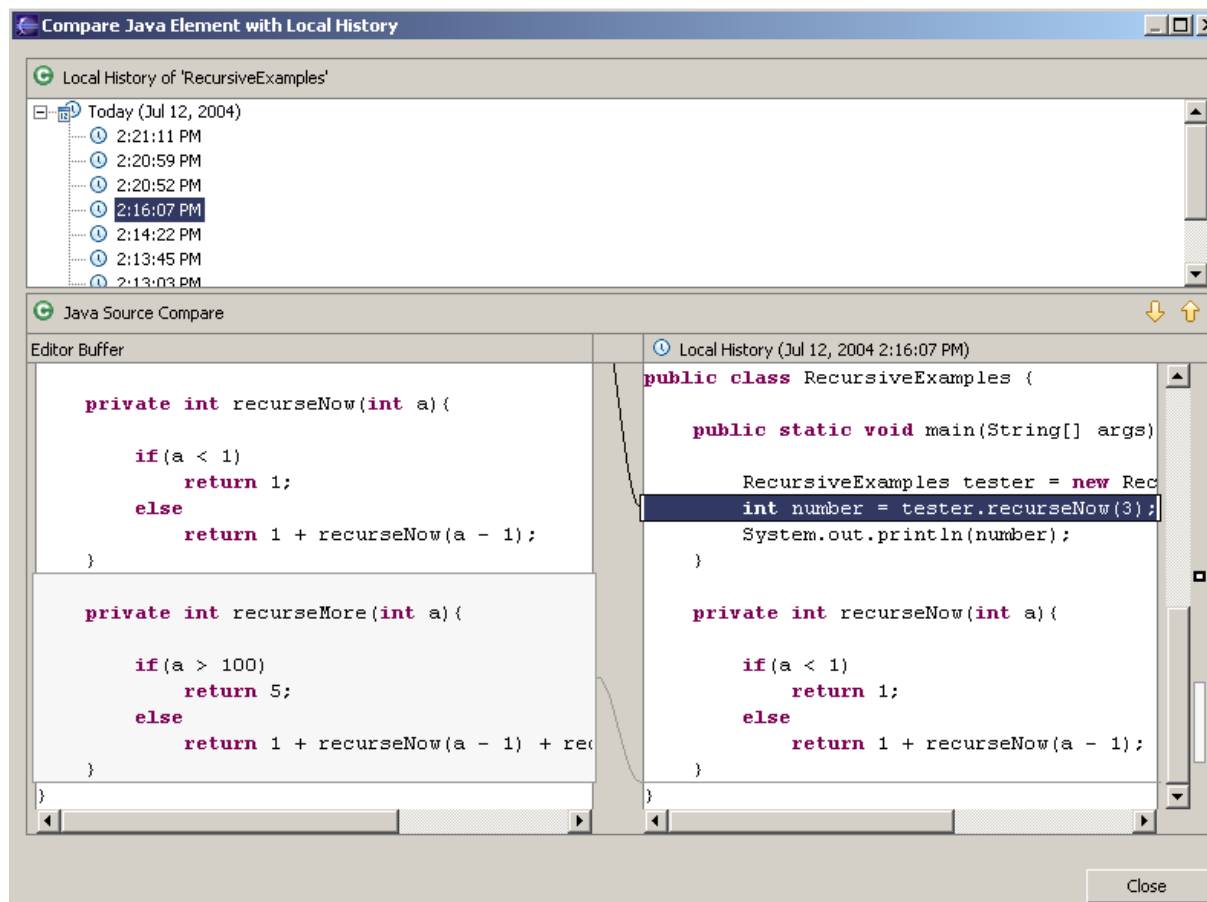
# Обзор среды Eclipse.

## Локальная история (2)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

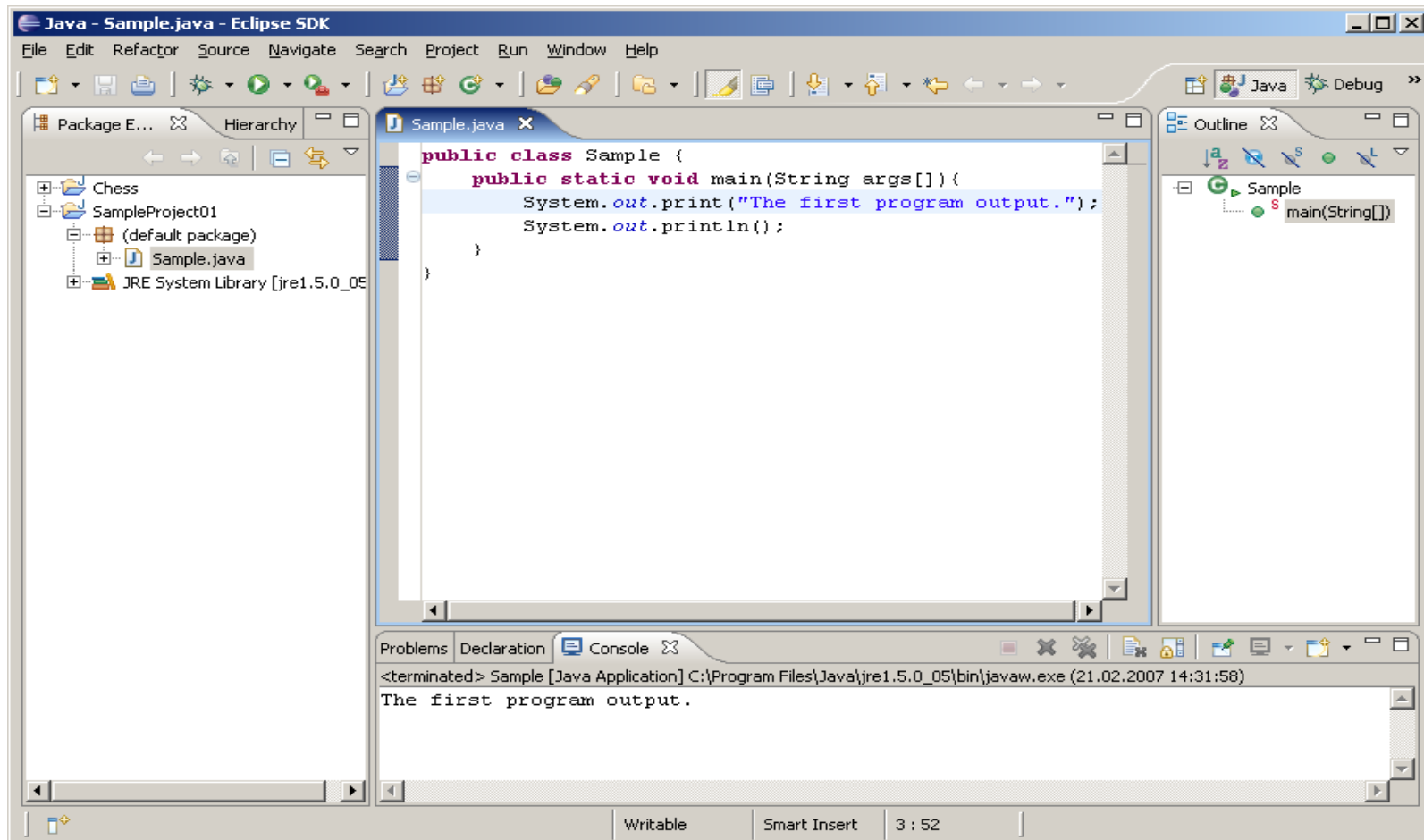
- Каждое использование команды **Save** сохраняет редакцию файла
- Редакции файлов можно сравнить



# Вывод программы на консоль

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



# Введение в язык Java

# Классы Java и их синтаксис

## Члены класса

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы инкапсулируют атрибуты (поля) и поведение (методы)
- Поля и методы являются *членами* класса
- Члены класса могут принадлежать всем экземплярам класса. В этом случае поля и методы помечаются ключевым словом *static*
- Члены класса могут принадлежать конкретным экземплярам класса. В этом случае они называются полями и методами экземпляров класса
- В одном файле с расширением *\*.java* не может быть более одного публичного класса

# Классы Java и их синтаксис.

## Члены классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы инкапсулируют атрибуты (поля) и поведение (методы)
- Атрибуты и методы являются *членами* класса

```
class Square {  
    int h;  
    int v;  
    boolean isNear(Square s) {  
        return  
            Math.abs(h - s.h) <= 1 &&  
            Math.abs(v - s.v) <= 1;  
    }  
}
```



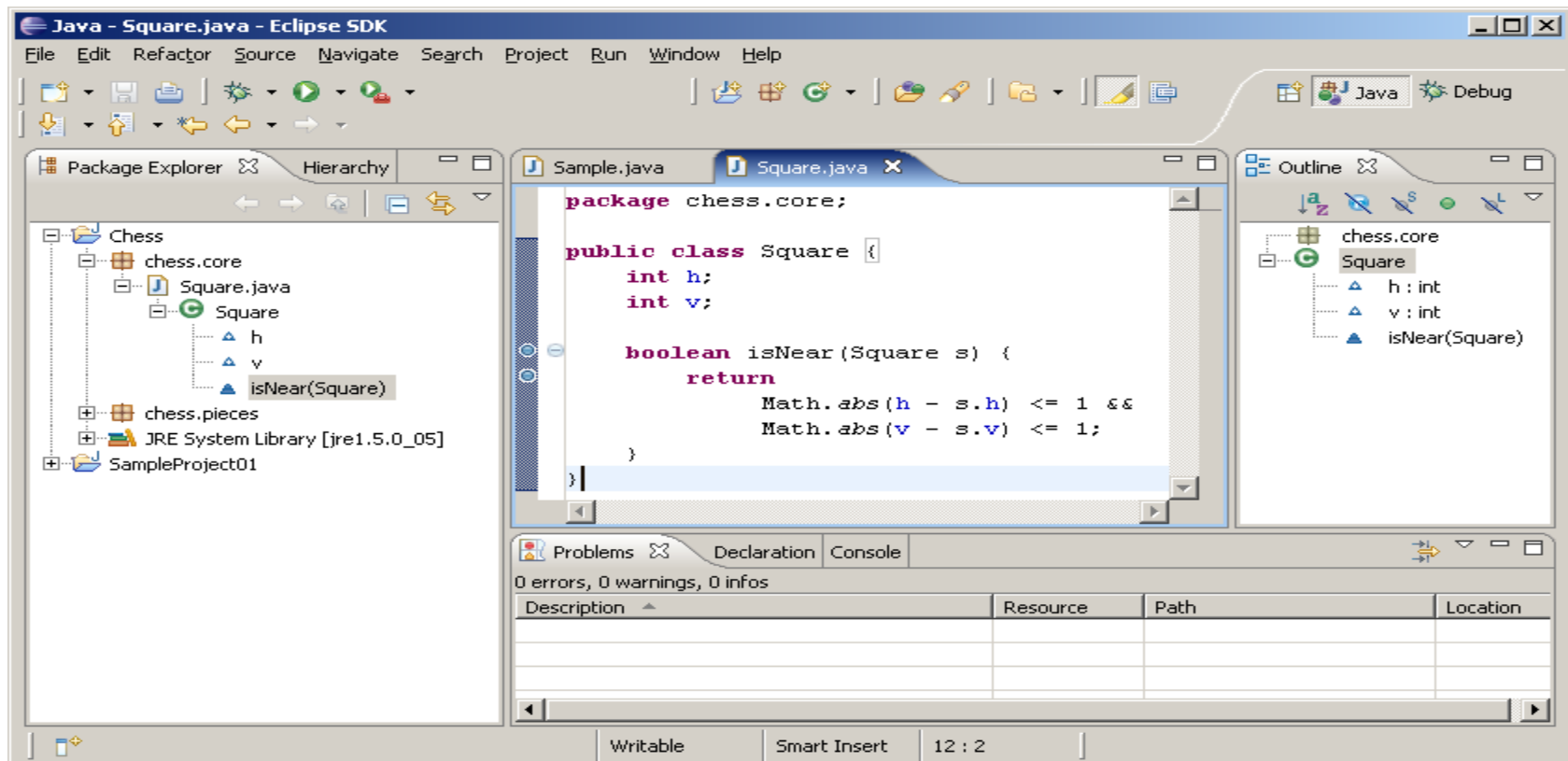
# Классы Java и их синтаксис.

## Члены классов. Eclipse

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы инкапсулируют атрибуты (поля) и поведение (методы)
- Поля и методы являются *членами* класса



# Классы Java и их синтаксис.

## Объявление полей класса и экземпляра

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Поля класса могут принадлежать всем экземплярам класса. В этом случае поля помечаются ключевым словом *static*.

```
class Point {  
    public int x, y;  
  
    static public final  
    Point ORIGIN = new Point(0,0);  
  
    public Point(int newX, int newY)  
        { x = newX; y = newY; }  
  
    public Point(Point p)  
        { x = p.x; y = p.y; }  
}
```

Поле экземпляра класса

Поле всех экземпляров класса

# Классы Java и их синтаксис.

## Использование полей класса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Для доступа к статическому полю (класса) используется имя класса, для доступа к полю экземпляра - имя экземпляра.

```
// Использование поля класса.  
Point p1 = Point.ORIGIN;  
  
// Создание экземпляра класса.  
Point p2 = new Point(3, 4);  
Point p3 = new Point(5, 6);  
  
// Использование поля экземпляра.  
p2.y = 100;  
p3.y = p2.y;  
}
```

# Классы Java и их синтаксис.

## Объявление методов класса и экземпляра

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Методы класса принадлежат всем экземплярам класса. В этом случае методы помечаются ключевым словом *static*

```
class Point {  
    static  
    public int distanse(Point p1, Point p2) {  
        int dx = p1.x - p2.x;  
        int dy = p1.y - p2.y;  
        return Math.sqrt(dx * dx + dy * dy);  
    }  
    public int distanse(Point p) {  
        int dx = p.x - x;  
        int dy = p.y - y;  
        return Math.sqrt(dx * dx + dy * dy);  
    }  
}
```

# Классы Java и их синтаксис.

## Использование методов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Для доступа к статическому методу (класса) используется имя класса, для доступа к методу экземпляра - имя экземпляра класса.

```
Point p1 = new Point(3, 4);
Point p2 = new Point(5, 6);

// Использование метода класса.
int d1 = Point.distanse(p1, p2);

// Использование метода экземпляра.
int d2 = p2.distanse(p1);
}
```

# Классы Java и их синтаксис.

## Область видимости статических методов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- В статических методах класса можно использовать только статические поля класса и вызывать только статические методы класса.

```
class Test {
    public int p;

    public void process() {
    }
    static public void main(String s[]) {
        p = 1; // Ошибка
        process(); // Ошибка

        Test test = new Test();
        test.p = 1;
        test.process();
    }
}
```

# Классы Java и их синтаксис.

## Наследование классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы могут быть независимы друг от друга
- Классы могут быть связаны *отношением наследования*. (суперкласс/подкласс).

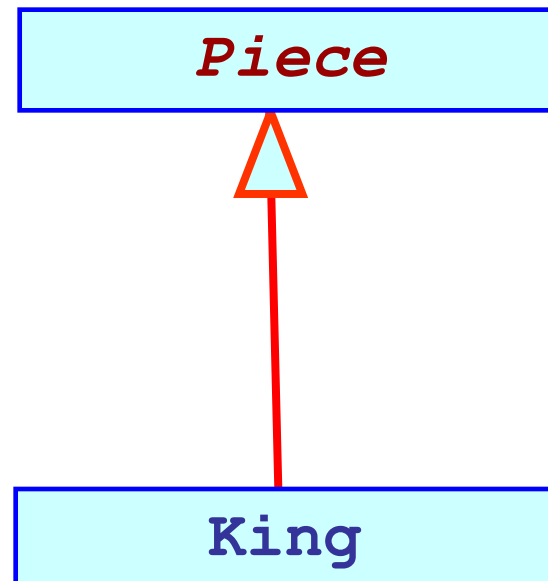
```
public class King extends Piece {  
    void move(Square s) {  
    }  
}
```

# Изображение наследования классов Java на UML-диаграммах

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public class King extends Piece {  
    void move(Square s) {  
    }  
}
```





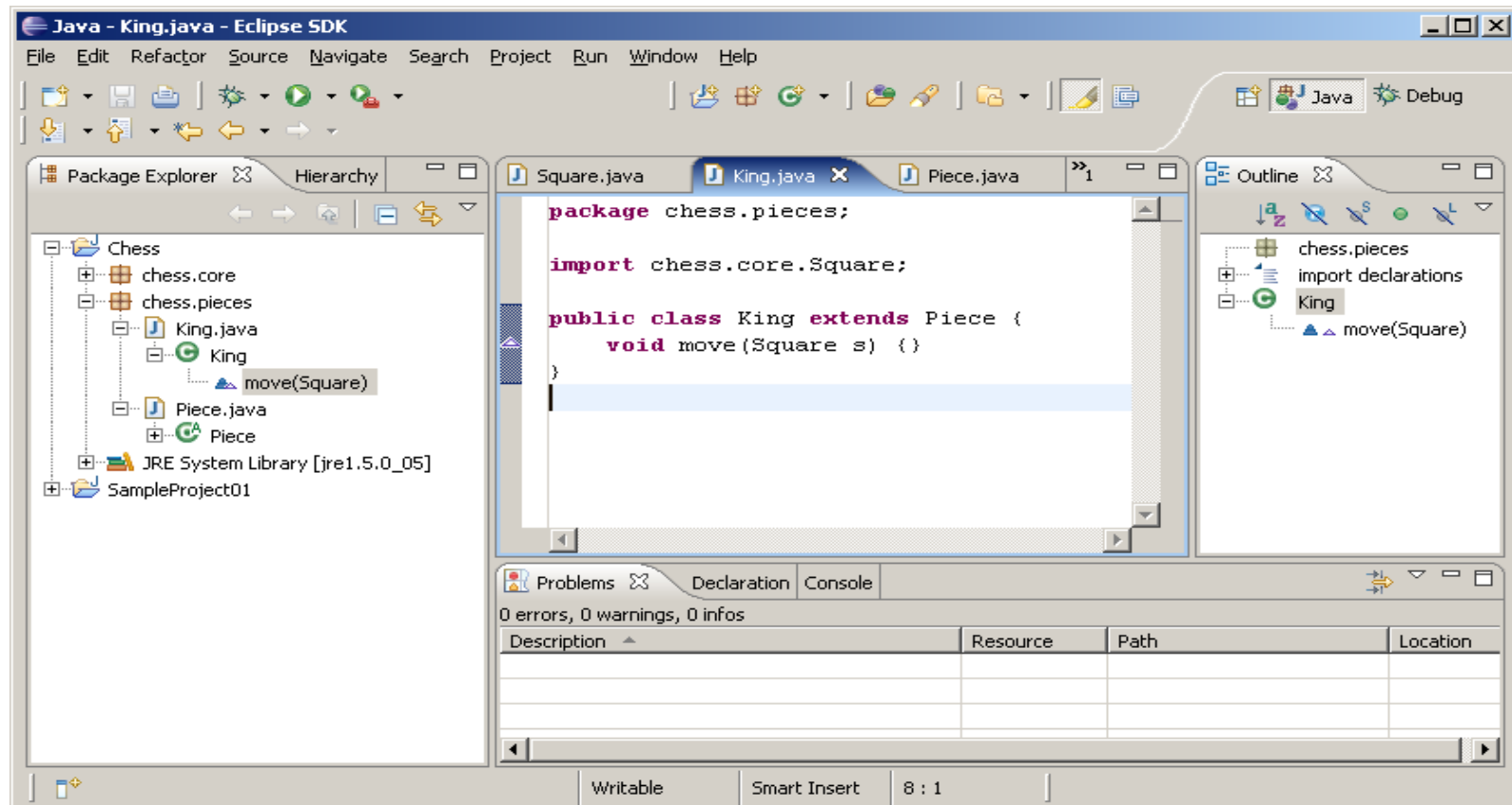
# Классы Java и их синтаксис.

## Наследование классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы могут быть связаны отношением наследования (суперкласс/подкласс).



# Интерфейсы в Java и их синтаксис.

## Объявление интерфейсов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Интерфейсы описывают методы, которые должны быть реализованы в классах
- Могут описывать неизменяемые (**final**) переменные
- Метафора – что должен уметь делать класс

```
// Интерфейс для классов представляющих
// шахматные ходы
public interface Move {
    // Передвинуть фигуры на доске.
    void doMove();

    // Вернуть фигуры в исходное состояние.
    void undoMove();
}
```

# Интерфейсы в Java и их синтаксис.

## Реализация интерфейсов классами

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы реализуют методы интерфейса
- Абстрактные классы могут реализовать не все методы интерфейса. Остальные реализуют потомки класса.

```
// Шахматный ход – рокировка.  
public class Castling implements Move {  
    void doMove() {  
        // Передвинуть короля и ладью  
    }  
  
    void undoMove() {  
        // Вернуть короля и ладью  
    }  
}
```

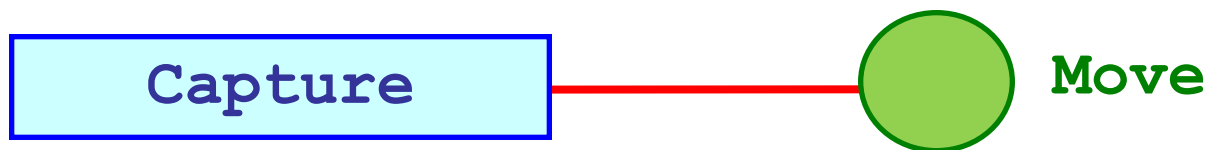
# Интерфейсы в Java и их синтаксис.

## Реализация интерфейсов классами

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
// Шахматный ход – захват чужой фигуры
public class Capture implements Move {
    void doMove() {
        // Убрать чужую фигуру.
        // Передвинуть на ее место свою.
    }
    void undoMove() {
        // Передвинуть свою фигуру на прежнее место.
        // Поставить чужую фигуру.
    }
}
```



# Интерфейсы в Java и их синтаксис. Не изменяемые поля интерфейса.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public interface BaseColors {  
    int RED = 1, GREEN = 2, BLUE = 4;  
}
```

```
int color = BaseColors.RED;
```

# Интерфейсы в Java и их синтаксис.

## Статические методы интерфейсов в Java 8.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
// Фигура ходит по диагонали.  
public interface DiagonalPiece {  
  
    static boolean isDiagonalPined(Square target) {  
        // Может ли фигура пойти на поле target?  
        // Откроется ли королю шах если диагональная  
        // фигура пойдет на клетку target  
    }  
  
}
```

# Интерфейсы в Java и их синтаксис.

## Статические методы интерфейсов в Java 8.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
// Фигура ходит по вертикали или горизонтали.  
public interface LinePiece {  
    static  
    boolean isVerticalPined(Square target) {  
        // Может ли фигура пойти на поле target?  
        // Откроется ли королю шах если  
        // фигура пойдет на клетку target  
    }  
  
    static  
    boolean isHorizontalPined(Square target) {  
        // Может ли фигура пойти на поле target?  
        // Откроется ли королю шах если  
        // фигура пойдет на клетку target  
    }  
}
```

# Интерфейсы в Java и их синтаксис

## Использование статических методов в Java 8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public class Queen extends Piece
    implements DiagonalPiece, LinePiece
{
    boolean isCorrectMove(Square s) {
        if (isHorizontalPined(s))
            return false;
        if (isVerticalPined(s))
            return false;
        if (isDiagonalPined(s))
            return false;
        // ...
    }
    // ...
}
```



# Интерфейсы в Java и их синтаксис.

## Использование статических методов в Java 8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public class Bishop extends Piece
    implements DiagonalPiece
{
    void isCorrectMove (Square s) {
        if (isDiagonalPined(s))
            return false;

        // ...
    }
    // ...
}
```

# Интерфейсы в Java и их синтаксис

## Использование статических методов в Java 8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public class Rook extends Piece
    implements LinePiece
{
    boolean isCorrectMove (Square s) {
        if (isHorizontalPined(s))
            return false;
        if (isVerticalPined(s))
            return false;
        // ...
    }
    // ...
}
```

# Интерфейсы в Java и их синтаксис

## Расширение интерфейсов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
// Интерфейс для классов представляющих  
// шахматные ходы  
public interface Move {  
    void doMove();  
    void undoMove();  
  
    Board getBoard(); // Ошибка!!!  
}
```

Классы в уже оттранслированных  
библиотеках не смогут реализовать  
новый метод интерфейса

# Интерфейсы в Java и их синтаксис

## Использование умалчиваемых методов в Java 8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
// Интерфейс для классов представляющих
// шахматные ходы
public interface Move {
    void doMove();
    void undoMove();

    default Board getBoard() {
        // ...
    }
}
```

Для классов в уже  
оттранслированных библиотеках  
используется умалчиваемая  
реализация метода интерфейса

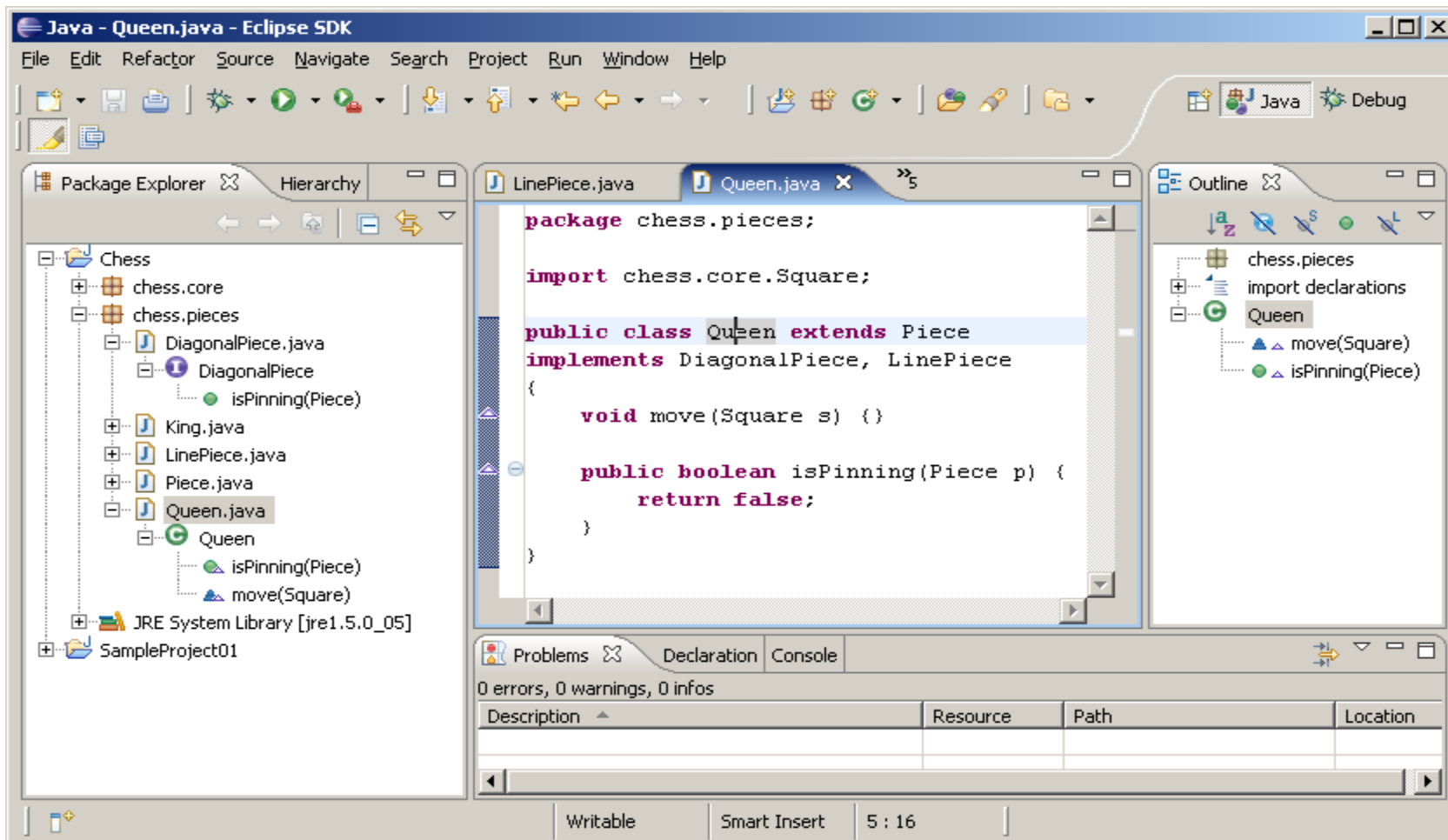
# Классы Java и их синтаксис.

## Реализация интерфейсов. Eclipse

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Классы могут реализовывать один и более *интерфейсов*



The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows a project structure with a package named `chess.pieces` containing classes `DiagonalPiece.java`, `King.java`, `LinePiece.java`, `Piece.java`, and `Queen.java`. The `Queen` class is highlighted.
- Main Editor:** Displays the source code for `Queen.java`. The code is as follows:

```
package chess.pieces;

import chess.core.Square;

public class Queen extends Piece
implements DiagonalPiece, LinePiece
{
    void move(Square s) {}

    public boolean isPinning(Piece p) {
        return false;
    }
}
```
- Outline:** Shows the class hierarchy and methods for `Queen`, including `move(Square)` and `isPinning(Piece)`.
- Problems:** Shows 0 errors, 0 warnings, and 0 infos.
- Bottom Bar:** Shows the status bar with "Writable", "Smart Insert", and "5 : 16".

# Классы Java и их синтаксис.

## Группирование классов в пакеты

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Логически связанные классы группируются в *пакеты*

```
package chess.pieces;  
  
import chess.core.Square;  
  
public class Queen extends Piece  
    implements DiagonalPiece, LinePiece  
{  
    void move(Square s) {}  
  
    public boolean isPinning(Piece p) {  
        return false;  
    }  
}
```

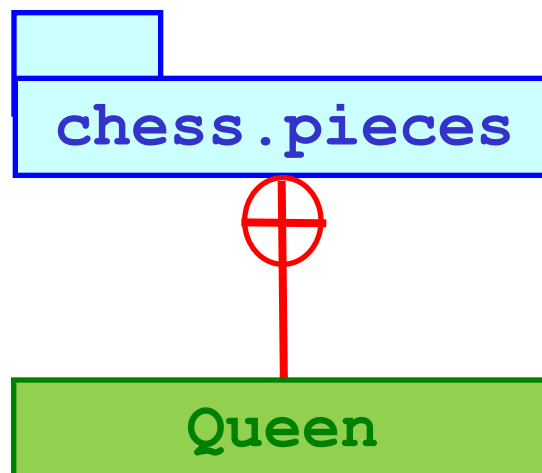
# Изображение пакетов на UML-диаграммах

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
package chess.pieces;
```

```
public class Queen extends Piece  
implements DiagonalPiece, LinePiece  
{  
}
```



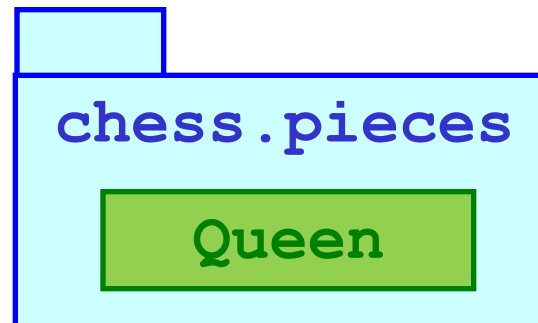
# Изображение пакетов на UML-диаграммах

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
package chess.pieces;
```

```
public class Queen extends Piece  
implements DiagonalPiece, LinePiece  
{  
}
```





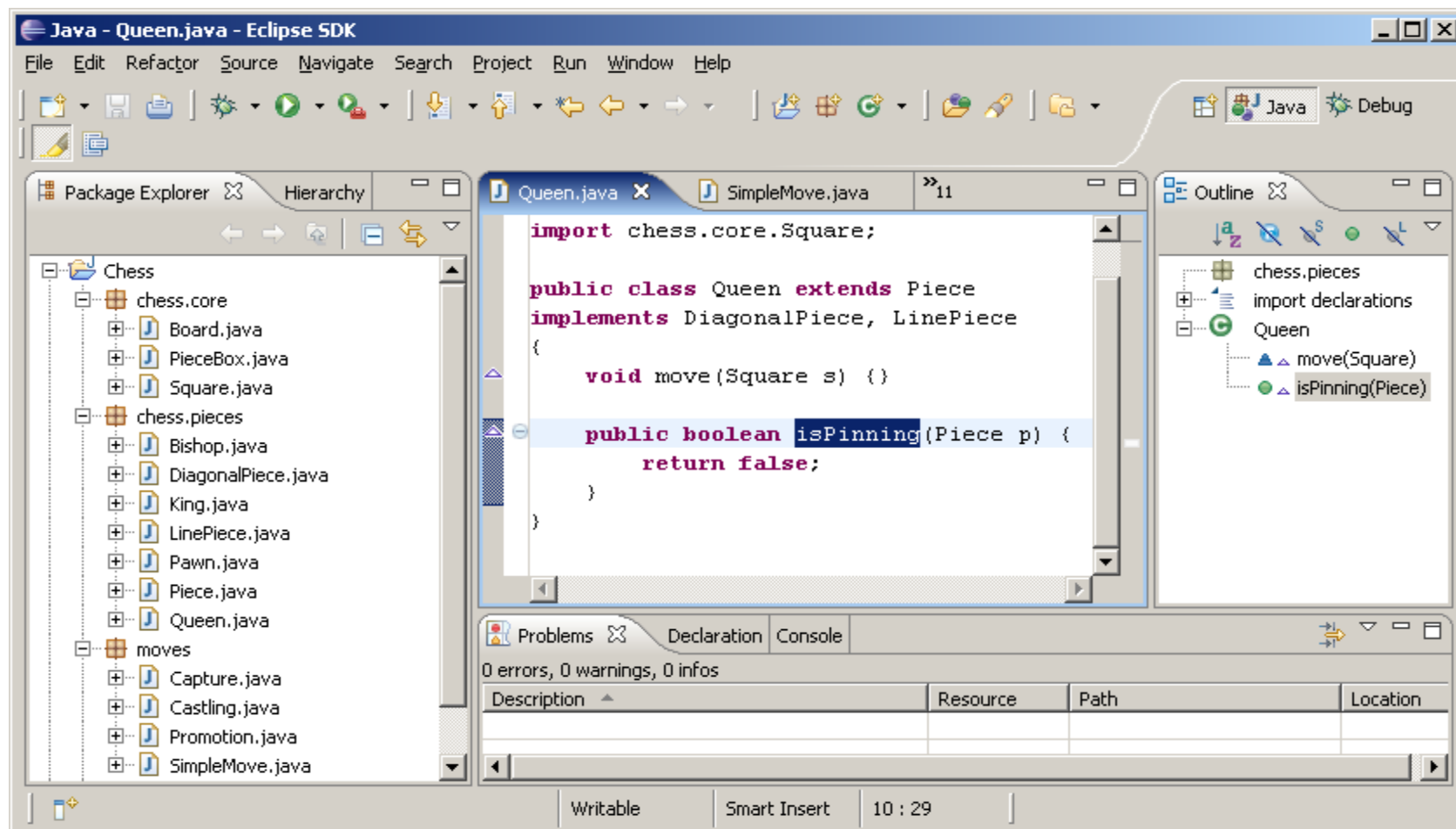
# Классы Java и их синтаксис.

## Группирование классов в пакеты

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Логически связанные классы группируются в пакеты



# Классы Java и их синтаксис.

## Модификаторы классов

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Модификатор *public*

Класс доступен для всех других классов. Отсутствие модификатора означает доступность класса только классам внутри пакета содержащего данный класс.

- Модификатор *private*

Этот модификатор допустим только для классов, которые *вложены (nested)* в другие классы

- Модификатор *abstract*

Запрет на создание экземпляров класса

- Модификатор *final*

Запрет на создание подклассов данного класса

# Изображение модификаторов класса на UML

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
abstract public class Piece  
{  
}
```

*Piece*

*{abstract}*

# Конструкторы классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- *Конструктор* – метод который создает экземпляр класса. Имя конструктора совпадает с именем класса. Допустимы несколько конструкторов с различными параметрами
- Конструктор используется для инициализации объектов
- Тело класса содержит по меньшей мере один конструктор
- Конструктор возвращает указатель на созданный объект. Оператор **return** в конструкторе отсутствует
- Для создания *экземпляров* класса используется ключевое слово **new** с именем конструктора

```
King piece = new King();
```

# Умалчиваемые конструкторы классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Умалчиваемый конструктор не имеет аргументов.

Предоставляется по умолчанию платформой Java если нет ни одного явно определенного конструктора

- При определении хотя бы одного явного конструктора необходимо объявить явно и умалчиваемый конструктор

```
class King {}  
King piece = new King();
```

```
class King {  
    King(Square s) {}  
}  
King piece = new King(); // Ошибка компиляции!!!
```

```
class King {  
    King() {}  
    King(Square s) {}  
}  
King piece = new King(); // Ошибки компиляции теперь нет
```

# Цепочка конструкторов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- *Цепочка конструкторов* - вызов внутри класса одного конструктора другим конструктором. Цепочки используются для разделения общего кода между конструкторами
- Вызов цепочки конструкторов:  
`this` (список аргументов)

```
class Piece {
    boolean isWhite;
    Square square;

    Piece(Square square, boolean isWhite)
        { this.square = square; this.isWhite = isWhite; }
    Piece(Square square)
        { this(square, true); }
    Piece()
        { this(null, true); }
}
```

# Конструкторы суперкласса

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Объекты суперкласса строятся до построения объекта-подкласса
- Для инициализации членов суперкласса используется вызов:
  - **super**(список-аргументов)
- Первой строкой конструктора могут быть:
  - **super**(список-аргументов)
  - **this**(список-аргументов)
- В одном конструкторе нельзя использовать одновременно и **super** и **this**
- Компилятор предоставляет *неявный вызов конструктора* **super()** для всех конструкторов подклассов

# Освобождение памяти объектов

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- В Java отсутствует понятие *деструктора* для неиспользуемых объектов
- Освобождение памяти выполняется автоматически виртуальной машиной Java
- Сборщик мусора освобождает память объектов, на которые нет ссылок
- Связь между объектом и ссылкой на объект уничтожается при задании нового значения ссылке на объект

`objectReference = null;`

- Объект без ссылок – кандидат на освобождение при сборке мусора



# Сборка мусора

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Можно явно запросить сборку мусора:

```
System.gc()
```

- Метод объекта *finalize* будет выполняться непосредственно перед сборкой мусора.

```
@Override  
protected void finalize() {  
    // TODO Здесь освободить захваченные ресурсы  
}
```

Используется при:

- освобождении памяти выделенной с помощью *native*-методов
- закрытии открытых объектом файлов перед тем, как память объекта будет освобождена.

# Поля классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- *Поля*
  - Часть определения классов
  - Состояние объекта хранится в полях
  - Каждый экземпляр получает собственную копию переменных экземпляра
- В месте объявления поля могут быть *инициализированы*
- Если поля не инициализируются явно, то используются *умалчиваемые значения*

```
public class Piece {  
    public boolean isWhite;  
    protected Square square;  
}
```

модификатор  
доступа

ТИП

ИМЯ

# Методы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- *Методы* определяют
  - Как объект отвечает на сообщение
  - Поведение класса
  - Все методы принадлежат классу

```
public class Square {  
    private boolean isNear(Square s) { ... }  
}
```

модификатор  
доступа

тип

имя

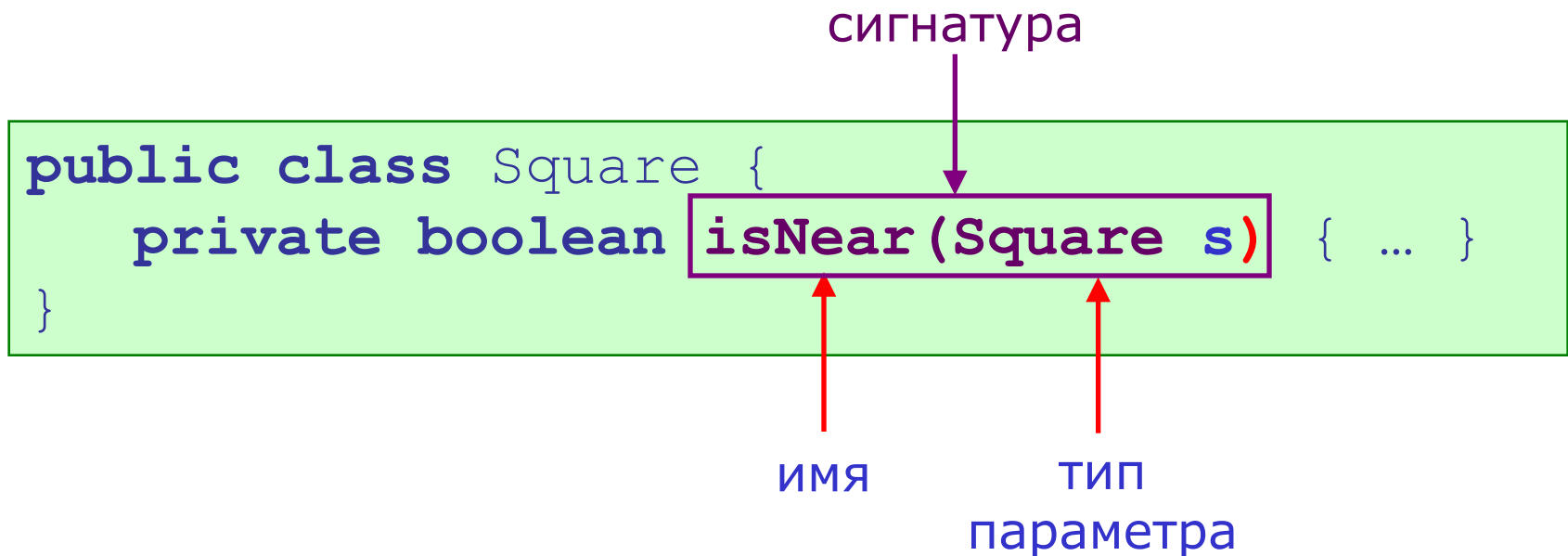
список  
параметров

# Сигнатура метода

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Класс может иметь множество методов с одинаковыми именами
- Каждый метод должен иметь другую сигнатуру
- Сигнатура – количество аргументов и типы аргументов



# Параметры метода

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Аргументы (параметры) пересылаются:
  - примитивные типы по значению
  - ссылки на объекты для ссылочных типов
- Примитивные значения не могут быть модифицированы при пересылке в качестве аргументов

```
public void method1 () {  
    int a = 0;  
    System.out.println(a); // вывод 0  
    method2(a);  
    System.out.println(a); // вывод 0  
}  
  
void method2(int a) {  
    a = a + 1;  
}
```

# Вызов метода и возврат из метода

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- И для вызова метода используется оператор точка
  - для методов экземпляра и для методов класса
  - если вызываемый метод в том же классе, то оператор точка не требуется

## Вызов через оператор точка метода класса

```
public class King {  
    boolean wasCastling() { ... }  
  
    boolean isCorrectMove(Square newSquare) {  
        Square s = King.oppositeKing().square;  
        return !s.isNear(newSquare) && !wasCastling();  
    }  
}
```

Вызов через оператор точка  
метода экземпляра

метод определен в  
том же классе

# Перекрытие метода

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Метод суперкласса перекрывается методом подкласса с той же сигнатурой

Перекрытый метод

```
public class Piece {  
    boolean isCorrectMove (Square newSquare) {...}  
}  
  
public class King extends Piece {  
    boolean isCorrectMove (Square newSquare) {  
        if (!super.isCorrectMove (newSquare))  
            return false;  
  
        Square s = King.oppositeKing().square;  
        return !s.isNear(square) && !wasCastling();  
    }  
}
```

Перекрывающий метод

Вызов перекрытого метода

# Метод *Main*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Приложение не может быть выполнено без хотя бы одного класса с методом *main*
- Виртуальная машина Java загружает класс и начинает выполнение с метода *main*

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

- **public** – метод может быть вызван любым объектом
  - **public**
- **static** – нет необходимости сначала создавать объект
- **void** – этот метод ничего не возвращает



# Инкапсуляция

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Объект – содержит данные и действия, которые можно выполнить над данными.
- Принцип сокрытия информации – объект знает о себе все, другие объекты запрашивают информацию об этом объекте.
- Все объекты отличаются друг от друга и программа – это обмен сообщениями между объектами
- Для сокрытия информации используется модификатор доступа *private*

# Статические члены

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Статические поля и методы принадлежат классу
- При изменении статического значения одним из объектов данного класса изменяется значение для всех объектов данного класса
- Статические методы и поля могут быть доступны без создания экземпляров класса

# Конечные (*final*) члены класса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Поле с модификатором *final* не может быть модифицировано. Это аналог констант в языке Java
- Константы связанные с классом обычно для простоты доступа объявляются с модификаторами *static final*
- Общепринято константы записывать большими буквами

```
public class Piece {  
    static final public KING = 1;  
    static final public QUEEN = 2;  
    static final public PAWN = 3;  
}
```

# Абстрактные классы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Невозможно создать экземпляр абстрактного класса. Предполагается, что они будут суперклассами для других классов
- Методы с модификатором *abstract* не имеют реализации
- Если класс имеет абстрактные методы, то он должен быть объявлен абстрактным

```
abstract public class Piece {  
    boolean isCorrectMove(Square s) { ... }  
    abstract void makeMove(Square s);  
}
```

# Пакеты классов

- Классы могут быть сгруппированы в пакеты

```
package chess.pieces;  
  
abstract public class Piece {  
    boolean isCorrectMove(Square s) { ... }  
    abstract void makeMove(Square s);  
}
```

- Различные пакеты могут иметь классы с одинаковыми именами
- По соглашению имена пакетов задаются в нижнем регистре

# Видимость классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Для ссылки на классы в том же пакете можно использовать только имя класса
- Для ссылки на классы из других пакетов необходимо использовать полностью квалифицированное имя класса

```
package chess.movies;

public class Castling extends Move {
    void doMove(Square s) {
        if (chess.pieces.King.wasCasling())
            return;
    }
}
```

# Импорт классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Предложение *import* используется для того, что бы сделать классы непосредственно доступными

```
package chess.movies;

import game.core.*;
import chess.pieces.King;
import chess.pieces.Rook;

public class Castling extends Move {
    void doMove(Square s) {
        if (King.wasCasling()) return;
        if (Rook.wasMoved()) return;
    }
}
```

# Импорт классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Предложение *import* используется для того, что бы сделать классы непосредственно доступными

```
package chess.movies;

import chess;

public class Castling extends Move {
    void doMove(Square s) {
        if (pieces.King.wasCasling()) return;
        if (pieces.Rook.wasMoved()) return;
    }
}
```



# Импорт статических методов классов в Java8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Пакет `java.lang` импортируется по умолчанию.

```
package samples;

//import java.lang.Math;

public class StaticImportSample {

    public static void main(String[] args) {
        Math.sin(Math.PI);
    }

}
```

# Импорт статических методов классов в Java8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Предложение ***import*** используется для импорта статической функции ***sin*** и статического поля **PI** класса **Math**.

```
package samples;

import static java.lang.Math.sin;
import static java.lang.Math.PI;

public class StaticImportSample {

    public static void main(String[] args) {
        sin(PI);
    }

}
```

# Импорт статических методов классов в Java8

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Предложение *import* используется для импорта всех статических методов и полей класса **Math**.

```
package samples;

import static java.lang.Math.*;

public class StaticImportSample {

    public static void main(String[] args) {
        sin(PI);
    }

}
```

# Пакеты ядра Java (1)

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- **java.lang**
  - неявно импортируется во все пакеты
  - предоставляет фундаментальные классы языка программирования Java (**Object**, **String**, **StringBuffer**, ...)
- **java.util**
  - библиотека классов-коллекций
  - модель для программирования событий
  - классы для работы с датами и временем
  - классы для локализации программ на различных национальных языках
- **java.io**
  - работа через потоки ввода/вывода
  - сериализация
  - работа с файловой системой

# Пакеты ядра Java (2)

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- **java.math**
  - стандартные математические функции
  - работа с большими целыми числами **BigInteger**
  - работа с большими вещественными числами **BigDecimal**
- **java.sql**
  - классы для анализа структуры реляционной базы данных
  - классы для выполнения запросов к базе данных на языке SQL
- **java.text**

классы и интерфейсы для обработки текста, дат, чисел способом независимым от национальных языков

# Наследование в языке Java

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

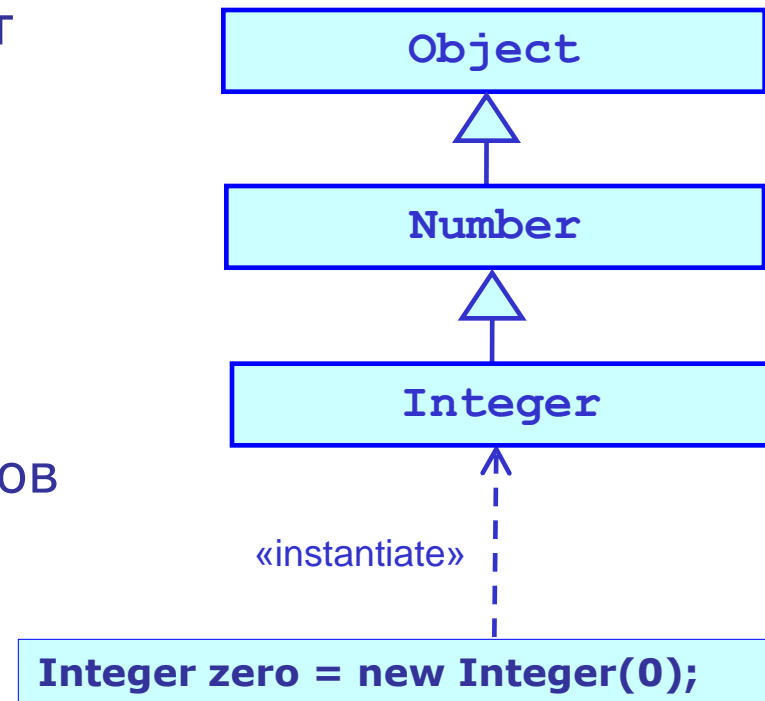
- Понимание наследования полей и методов
- Знакомство с иерархией классов
- Как подклассы специализируют классы
- Как выполняется поиск метода
- Как создаются и используются подклассы
- Понимание полиморфизма
- Рефакторинг в иерархии наследования

# Иерархии классов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Каждый объект принадлежит классу (является экземпляром класса)
- Каждый класс (кроме класса **Object**) имеет суперкласс
- Корень всей иерархии классов – класс **Object**.
- При определении нового класса

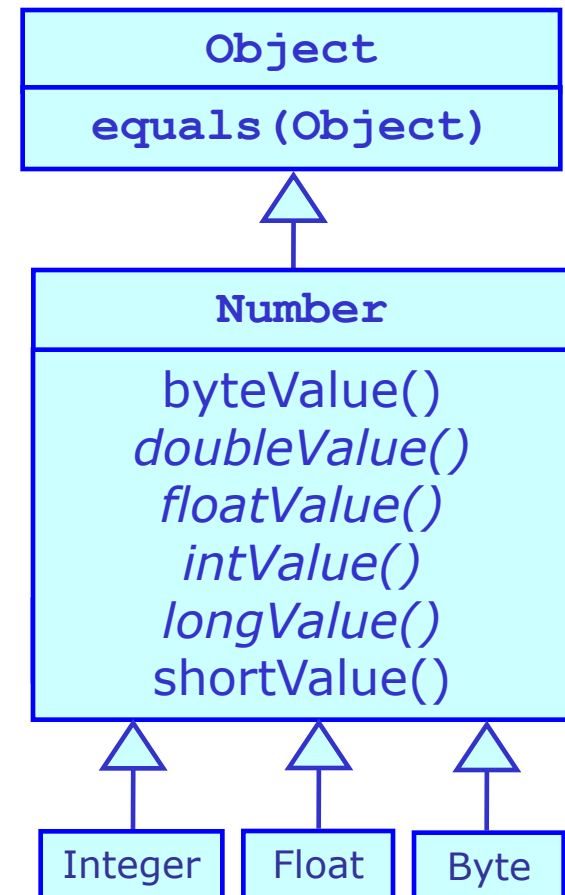


# Специализация и обобщение

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Подкласс есть специализация его суперкласса
- Суперкласс есть обобщение его подклассов.
- Общее состояние и поведение подкласса перемещается в суперкласс и становится доступным всем подклассам



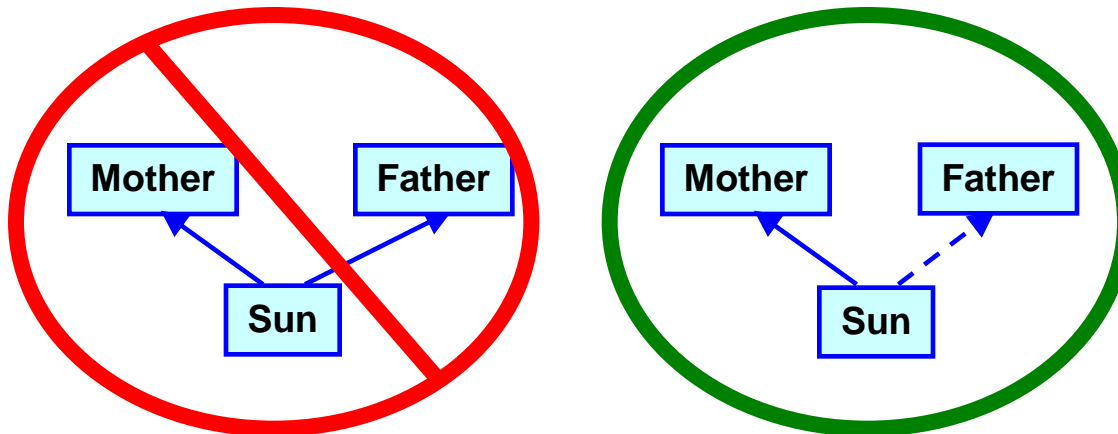


# Множественное наследование

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Множественное наследование в языке Java не поддерживается
- Каждый класс, за исключением класса **Object**, имеет только один непосредственный суперкласс
- Для реализации множества общих методов различными классами можно использовать не только наследование, но и реализацию классами интерфейсов



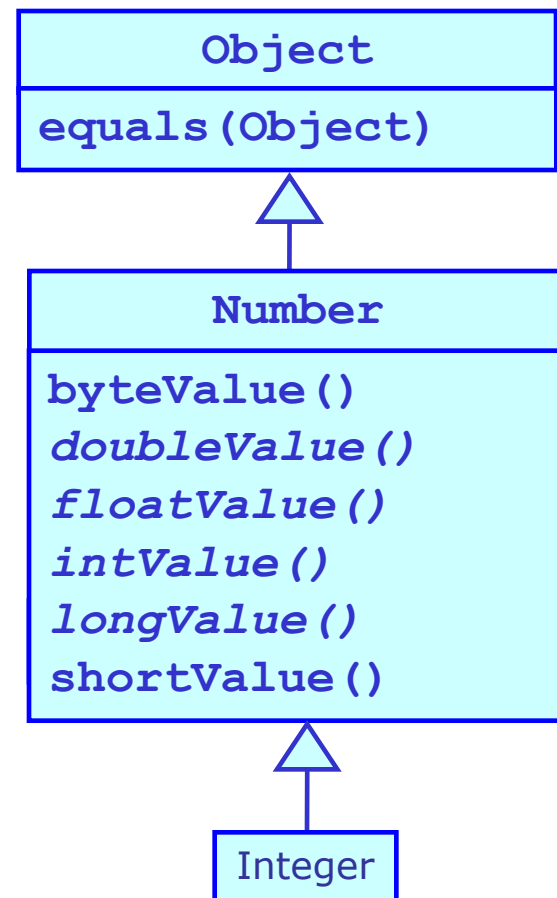
# Наследование полей и методов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Каждый подкласс наследует поля суперкласса и всех классов расположенных выше в иерархии наследования
- Каждый подкласс наследует методы суперкласса. Объект будет понимать все сообщения (вызов методов) его класса и суперклассов

```
Integer zero = new Integer(0);  
if (zero.equals(x)) {  
    byte b = zero.byteValue();  
    ...  
}
```



# Модификаторы доступа

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

• Поля и методы в Java имеют ограничения по доступу, описываемые следующими модификаторами:

- private – доступ ограничен классом в котором объявлен данный член класса

```
private int x;
```

- *умалчиваемый* (без модификатора) – доступ ограничен пакетом, в котором данный класс объявлен

```
int y;
```

- protected – доступ ограничен пакетом, в котором данный класс объявлен, и подклассами данного класса

```
protected void setName(String name) {...}
```

- public – доступ для всех классов всех пакетов

```
public String getName() {...}
```

# Перекрытие наследуемых методов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Имеется возможность дополнить или изменить поведение суперкласса перекрытием в подклассе унаследованного метода
- Перекрывающий метод должен иметь то же имя и список параметров (сигнатуру)
- Метод подкласса может заменять или уточнять метод суперкласса

```
public class MyClass extends Object {  
    public boolean equals(Object o) {  
        if (o==null)  
            ...  
    }  
}
```

# Ограничения на перекрытие методов

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Список параметров должен совпадать со списком параметров перекрываемого наследуемого метода суперкласса
- Тип возвращаемого результата должен совпадать с типом возвращаемого результата метода суперкласса
- Модификатор доступа в суперклассе не может быть более ограничительным, чем модификатор доступа в подклассе

Пример. При перекрытии метода с модификатором *protected*, новый метод может быть *protected* или *public*, но не *private*

# Наследование и статические методы

МГУ им. М.В.Ломоносова. Факультет ВМК.


Романов Владимир Юрьевич ©2024

- Класс может использовать статические методы унаследованные от суперклассов и собственные статические методы
- Статические методы не могут быть перекрыты

```
static String t = "test";

public static String superTest(String s) {
    s += " was the arg.";
    return s;
}
```

```
public static void main(String[] args){
    System.out.println(superTest(t));
}
```

A green arrow points from the `superTest(t)` call in the `main` method to the `superTest` method definition above.

# Наследование и конструкторы

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Вызываться могут только конструкторы класса, экземпляр которого создается, либо конструктор непосредственного суперкласса
- Для вызова конструктора суперкласса используется ключевое слово *super* и список параметров конструктора
- Для вызова конструктора того же класса используется ключевое слово *this* и список параметров конструктора
- Первой строкой конструктора может быть одна из:
  - `this ()`
  - `super ()`
- Вызов конструктора через `this()` или `super()` допустим только в конструкторе

# Блоки инициализации. Инициализация экземпляров класса

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Блок инициализации экземпляра вызывается до вызова любого из конструкторов класса
- В классе может быть неограниченное количество блоков инициализации
- Порядок расположения блоков инициализации в классе не имеет значения



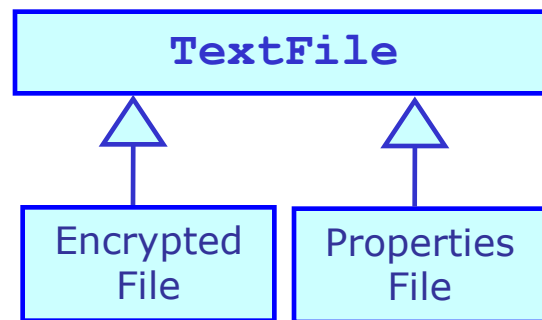
# Полиморфизм

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Переменной может быть присвоен объект типа переменной, либо объект типа – подкласс типа переменной:

```
TextFile file = new TextFile();  
TextFile file = new EncryptedFile();  
TextFile file = new PropertiesFile();
```



- Любой объект может быть присвоен переменной типа **Object**, поскольку он самый верхний в иерархии объектов

```
Object anything = new AnyType();
```

# Пример инициализации экземпляра класса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
package geometry;

public class Point {
    int x, y;

    {
        x = 1;
    }

    public Point() { }

    {
        y = 1;
    }

}
```

Блок  
инициализации  
экземпляра

Порядок расположения  
блоков инициализации  
не важен

# Блоки инициализации. Инициализация статических полей класса

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Блок инициализации статических полей класса вызывается до вызова любого из конструкторов класса
- В классе может быть неограниченное количество блоков инициализации статических полей классов
- Порядок расположения в классе блоков инициализации статических полей классов не имеет значения

# Пример инициализации статических полей класса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
package graph;

public class Node {
    static Color defaultLineColor, defaultColorFill;

    static {
        defaultLineColor = Color.Blue;
    }

    public Node() { }

    static {
        defaultFillColor = Color.Cyan;
    }
}
```

Блок инициализации статических полей

Порядок расположения блоков инициализации не важен

# Вложенность классов

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Способ группирования классов используемых только в одном месте. Вспомогательные классы внутри
- Увеличение инкапсуляции – вложенный класс работает со *скрытыми* (*private*) полями и методами *внешнего* (*outer*) класса.
- Получаемый код проще понимать – вложенные классы обычно небольшие
- Получаемый код проще сопровождать – место использования класса расположено близко к месту объявления класса

# Статические вложенные и внутренние классы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
class OuterClass {
```

```
...
```

```
static class StaticNestedClass {
```

```
...
```

```
}
```

```
class InnerClass {
```

```
...
```

```
}
```

```
}
```

Внешний класс

Статический вложенный  
класс

Внутренний класс

# Доступ к статическим вложенным классам

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
class OuterClass {  
    static class StaticNestedClass {  
    }  
} // class OuterClass  
  
class MainClass {  
    public static void main(String[] args) {  
  
        OuterClass.StaticNestedClass nestedInstance  
            = new OuterClass.StaticNestedClass ();  
  
    } // main  
} // class MainClass
```

Внешний класс

Статический вложенный класс

# Внутренние классы

- Внутренние классы могут существовать только внутри экземпляров внешнего класса
- Внутренние классы имеют прямой доступ к полям и методам экземпляра класса
- Перед созданием экземпляра внутреннего класса необходимо создать экземпляр внешнего класса
- Для внутренних классов можно задавать видимости: **public, private, protected**



# Доступ к внутренним классам

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
class OuterClass {  
    class InnerClass {  
    }  
} // class OuterClass  
  
class MainClass {  
    public static void main(String[] args) {  
  
        OuterClass outerInstance = new OuterClass();  
        OuterClass.InnerClass innerInstance  
            = outerInstance.new InnerClass();  
  
    } // main  
} // class MainClass
```

Внешний класс

Внутренний класс

Экземпляр внешнего класса

# Внутренние локальные классы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- *Внутренние локальные классы* могут объявляться внутри методов класса
- Внутренние локальные классы имеют доступ к параметрам и локальным переменным класса

```
class OuterClass {  
    public void outerMethod(int x) {  
        int y;  
  
        class InnerClass {  
        } // class InnerClass  
  
    } // outerMethod  
} // class OuterClass
```

# Внутренние анонимные классы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- *Внутренние анонимные классы* могут объявляться внутри методов класса
- Имя анонимного класса не указывается

```
class OuterClass {  
    public void outerMethod(int x) {  
  
        new Thread(new Runnable() {  
            public void run() {  
                ...  
            }  
        }).start();  
  
    } // outerMethod  
} // class OuterClass
```

Анонимный внутренний класс реализующий метод run интерфейса *Runnable*

# Перечисления в языке Java

# Типы-перечисления

- *Типы-перечисления* – это типы поля которых содержат фиксированное множество констант

```
public enum BaseColor {  
    RED, GREEN, BLUE  
}  
  
public class Test {  
    static public void main(String[] args) {  
        BaseColor color = BaseColor.RED;  
        switch (color) {  
            case RED: System.out.println("Red"); break;  
            case BLUE: System.out.println("Blue"); break;  
        } // switch  
    } // main  
} // class Test
```

# Поля и методы типов-перечислений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public enum Color {
    RED(255, 0, 0),
    GREEN(0, 255, 0),
    BLUE(0, 0, 255),
    WHITE(255, 255, 255),
    BLACK(0,0,0);

    public int red, green, blue;

    public int getIntensity()
        { return red + green + blue; }

    Color(int red, int green, int blue) {
        this.red = red;
        this.green = green;
        this.red = blue;
    }
} // enum Color
```

Возможно использование  
вне типа-перечисления

Возможно  
использование  
только внутри  
перечисления

# Поля и методы типов-перечислений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public enum Dirs {
    LEFT_UP (-1, -1),    UP (+0, -1),    RIGHT_UP (+1, -1),
    LEFT (-1, +0),      RIGHT (+1, +0),
    LEFT_DOWN(-1, +1), DOWN(+0, +1), RIGHT_DOWN (+1, +1);

    public static final
    Dirs[] ALL = {
        LEFT_UP,    UP,    RIGHT_UP,
        LEFT,       RIGHT,
        LEFT_DOWN, DOWN, RIGHT_DOWN
    };

    public int dv, dh;

    Dirs(int dv, int dh) {
        this.dv = dv;
        this.dh = dh;
    }
}
```

# Рефакторинг программ на языке Java

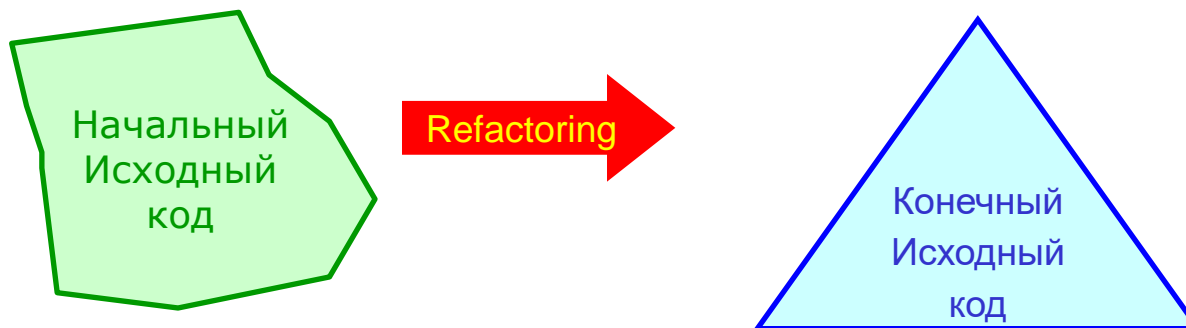


# Что такое рефакторинг

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Рефакторинг – процесс изменения программной системы таким образом, что внешнее поведение кода не изменяется, но внутренняя структура и архитектура улучшаются
- Рефакторинг – преобразование исходного кода сохраняющего поведение



# Для чего необходим рефакторинг

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Улучшение читаемости и понятности
- Улучшение расширяемости кода
- Добавление гибкости
- Улучшение производительности

# Как и когда выполняется рефакторинг

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Основа рефакторинга – обобщение. Абстракции находятся снизу вверх проверкой множества конкретных примеров
- Поиск методов с различными именами, но имеющими схожее поведение
- Параметризация различий у методов
- Разделение больших методов на методы меньшего размера, но допускающие большее переиспользование
- Выполняется во время сопровождения, тестирования, кодирования

# Цикл рефакторинга

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Исходный код программы должен проходить через фазы расширения и упорядочения
  - Фаза расширения – код добавляется для реализации новых функциональных требований
  - Фаза упорядочивания – код удаляется и преобразуется для улучшения структуры кода и архитектуры системы
- За время жизни программы этот цикл повторяется многократно

# Распространенные причины отказа от рефакторинга

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Изменение кода/проекта системы может привести его нерабочее состояние
  - Автоматизация тестирования позволит устранить эту причину отказа системы
- Непонятно как система работает в данный момент
  - В процессе рефакторинга система может быть изучена
  - Процесс рефакторинга можно документировать
- Недостаточно времени для выполнения рефакторинга
  - Проведение рефакторинга позволит существенно сократить время разработки на более поздних фазах работы с программой

# Методы рефакторинга

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

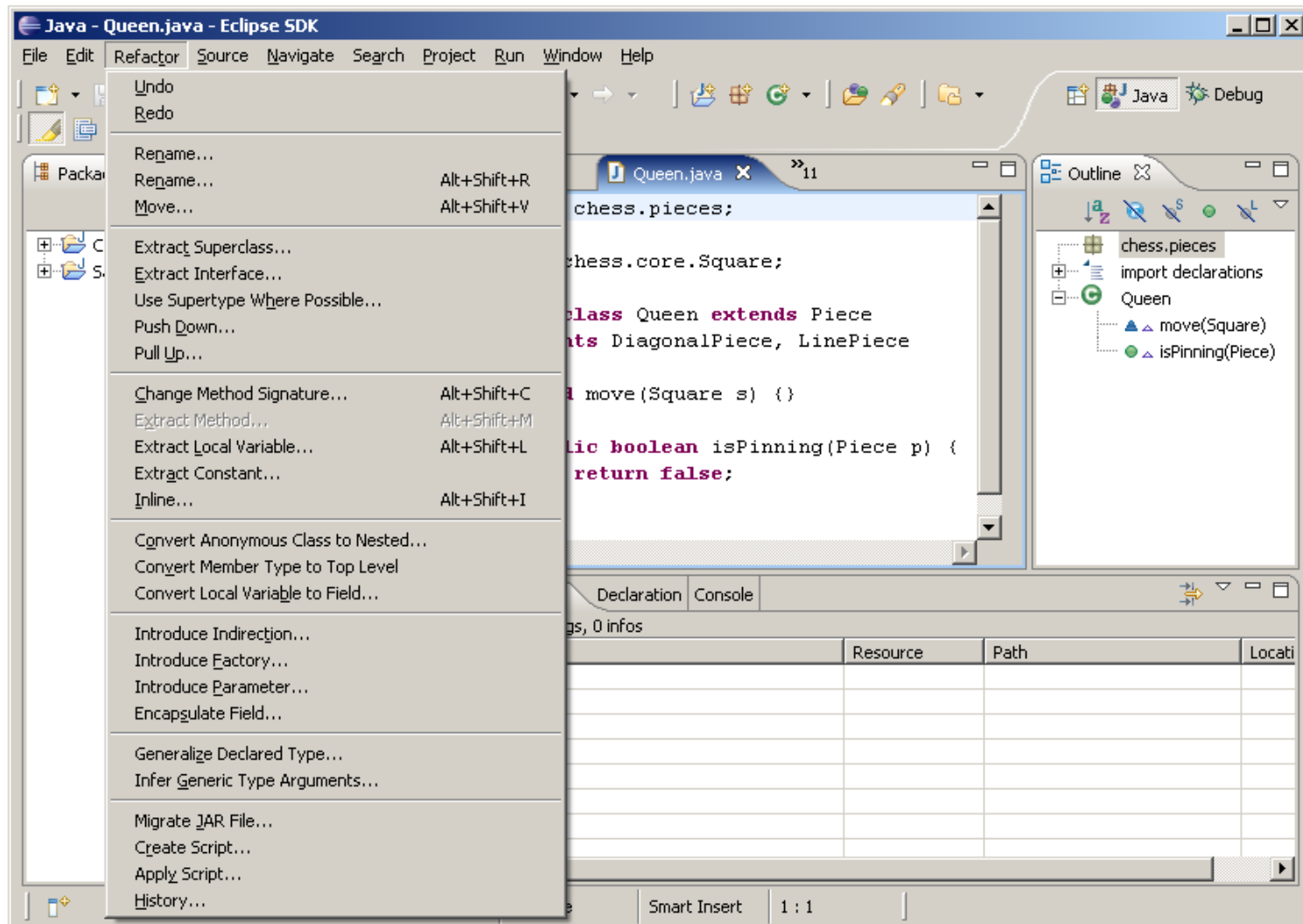
Романов Владимир Юрьевич ©2024

- Создание и удаление
  - классов, методов, переменных
- Перемещение методов и переменных:
  - вниз/вверх по иерархии наследования
  - перемещение в другой класс
- Реорганизация
  - Иерархии наследования
  - Кода методов

# Использование среды Eclipse как инструмента рефакторинга

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



# Рефакторинг в Eclipse. Переименование типа (1)

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

## Среда Eclipse существенно упрощает проведение рефакторинга

- Поиск в программе всех мест, где используется переименовываемый тип
- Отображение предполагаемых изменений и их влияния на программу
- Выполнение изменений



# Рефакторинг в Eclipse.

## Переименование типа (2)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

The screenshot shows the Eclipse IDE interface with the 'Refactor' menu open. The 'Rename...' option is selected, and the 'Rename Type' dialog is displayed in the foreground. The dialog shows the new name 'ChessPiece' and several options for updating references and occurrences.

**Refactor Menu:**

- Undo
- Redo
- Rename...
- Rename... Alt+Shift+R
- Move... Alt+Shift+V
- Extract Superclass...
- Extract Interface...
- Use Supertype Where Possible...
- Push Down...
- Pull Up...
- Change Method Signature... Alt+Shift+C
- Extract Method... Alt+Shift+M
- Extract Local Variable... Alt+Shift+L
- Extract Constant...
- Inline... Alt+Shift+I
- Convert Anonymous Class to Nested...
- Convert Member Type to Top Level
- Convert Local Variable to Field...
- Introduce Indirection...
- Introduce Factory...
- Introduce Parameter...
- Encapsulate Field...
- Generalize Declared Type...
- Infer Generic Type Arguments...
- Migrate JAR File...
- Create Script...
- Apply Script...
- History...

**Rename Type Dialog:**

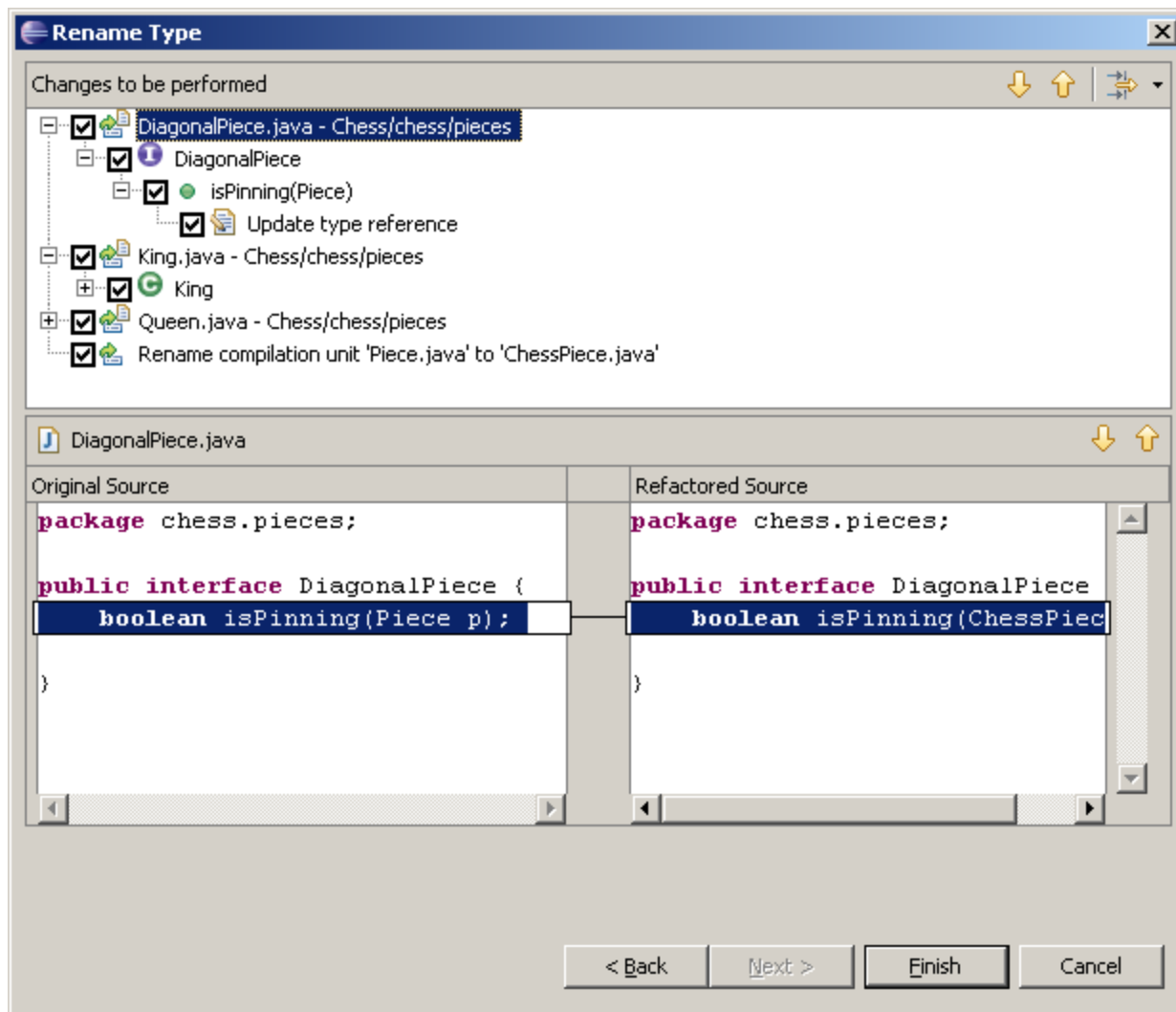
- New name: ChessPiece
- Update references
- Update similarly named variables and methods [Configure...](#)
- Update textual occurrences in comments and strings (forces preview)
- Update fully qualified names in non-Java text files (forces preview)
- File name patterns:
- The patterns are separated by commas (\* = any string, ? = any character)
- < Back
- Next >
- Finish
- Cancel

# Рефакторинг в Eclipse.

## Переименование типа (3)

МГУ им. М.В.Ломоносова. Факультет ВМК.

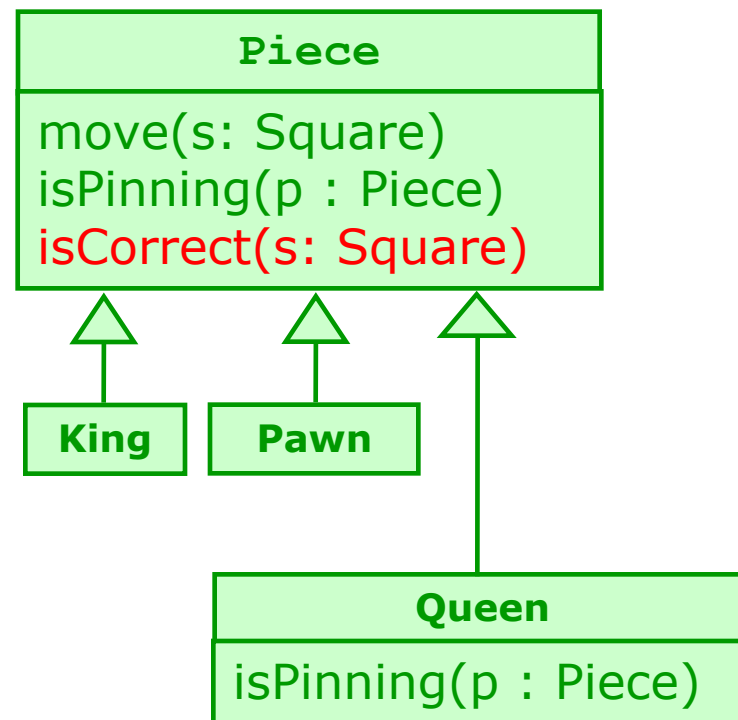
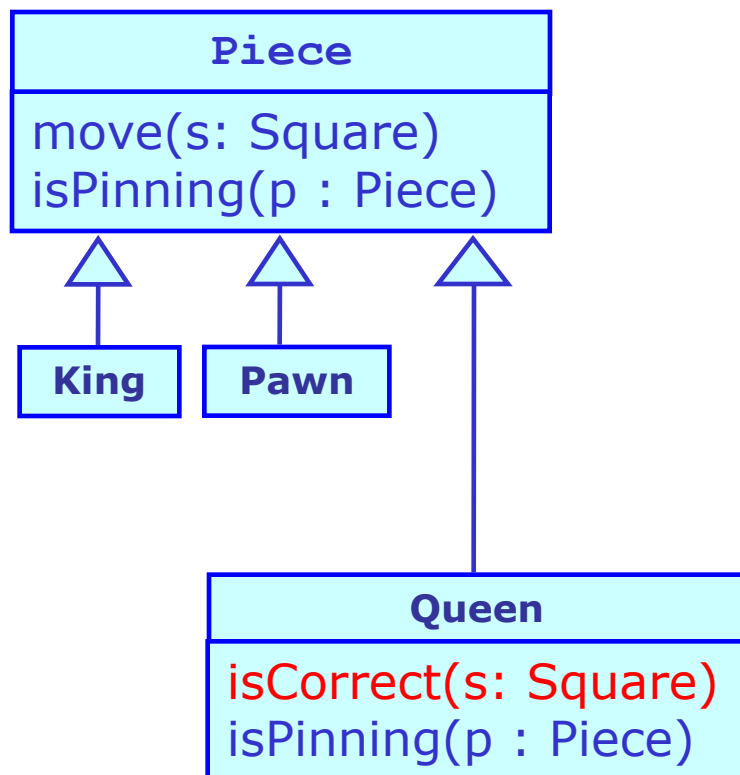
Романов Владимир Юрьевич ©2024



# Рефакторинг в Eclipse. Перемещение метода (1)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



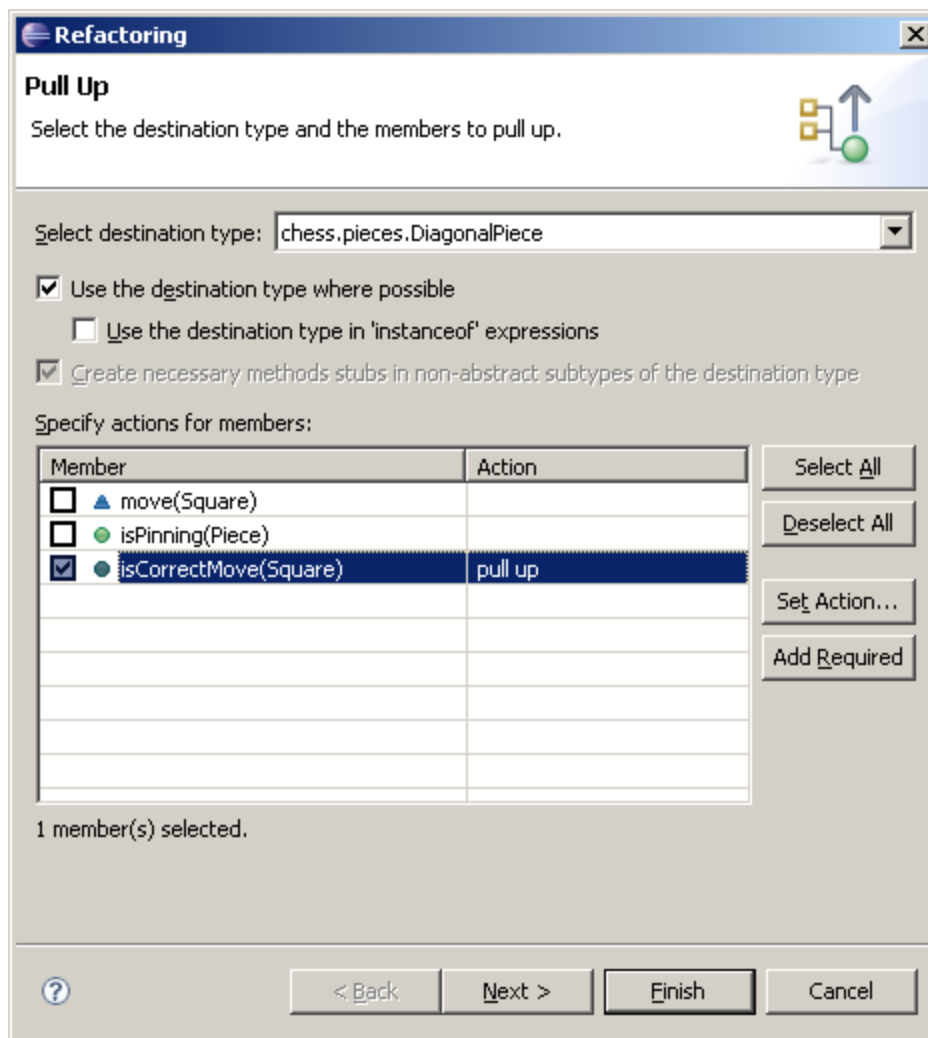


# Рефакторинг в Eclipse.

## Перемещение метода (3)

МГУ им. М.В.Ломоносова. Факультет ВМК.

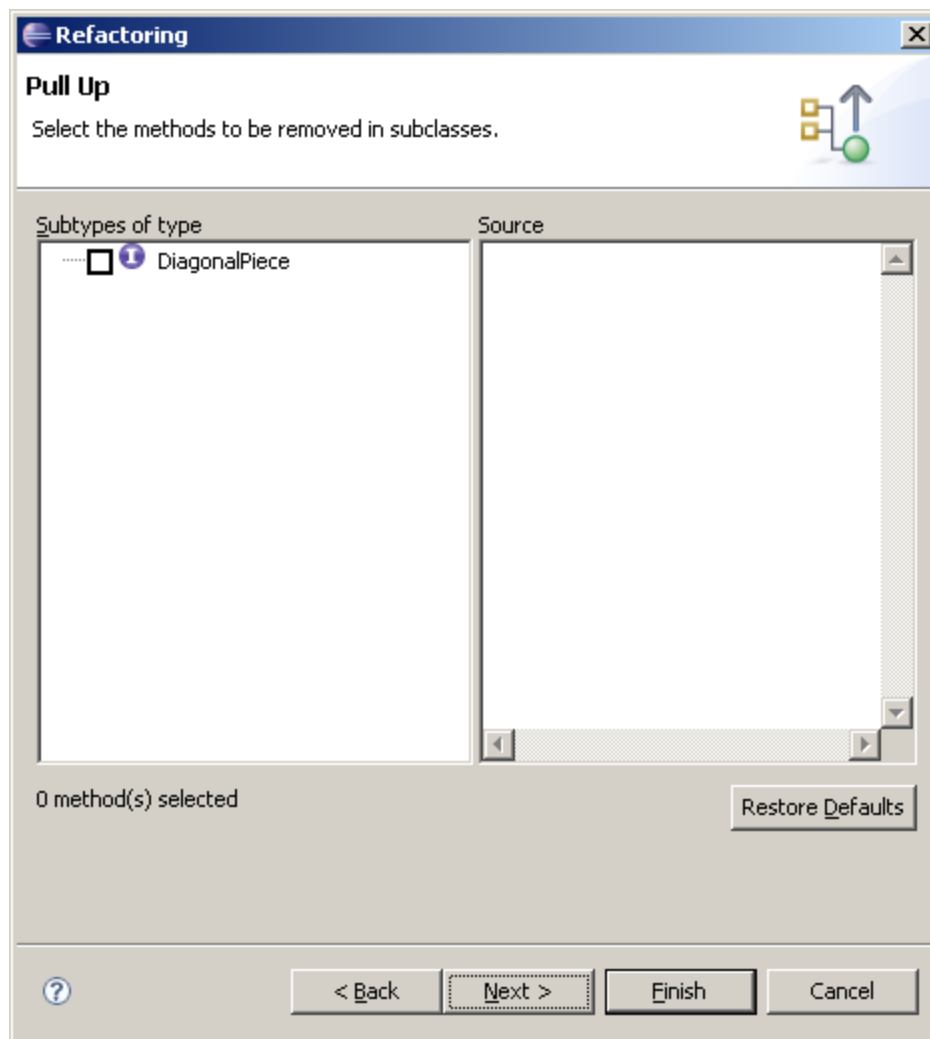
Романов Владимир Юрьевич ©2024



# Рефакторинг в Eclipse. Перемещение метода (4)

МГУ им. М.В.Ломоносова. Факультет ВМК.

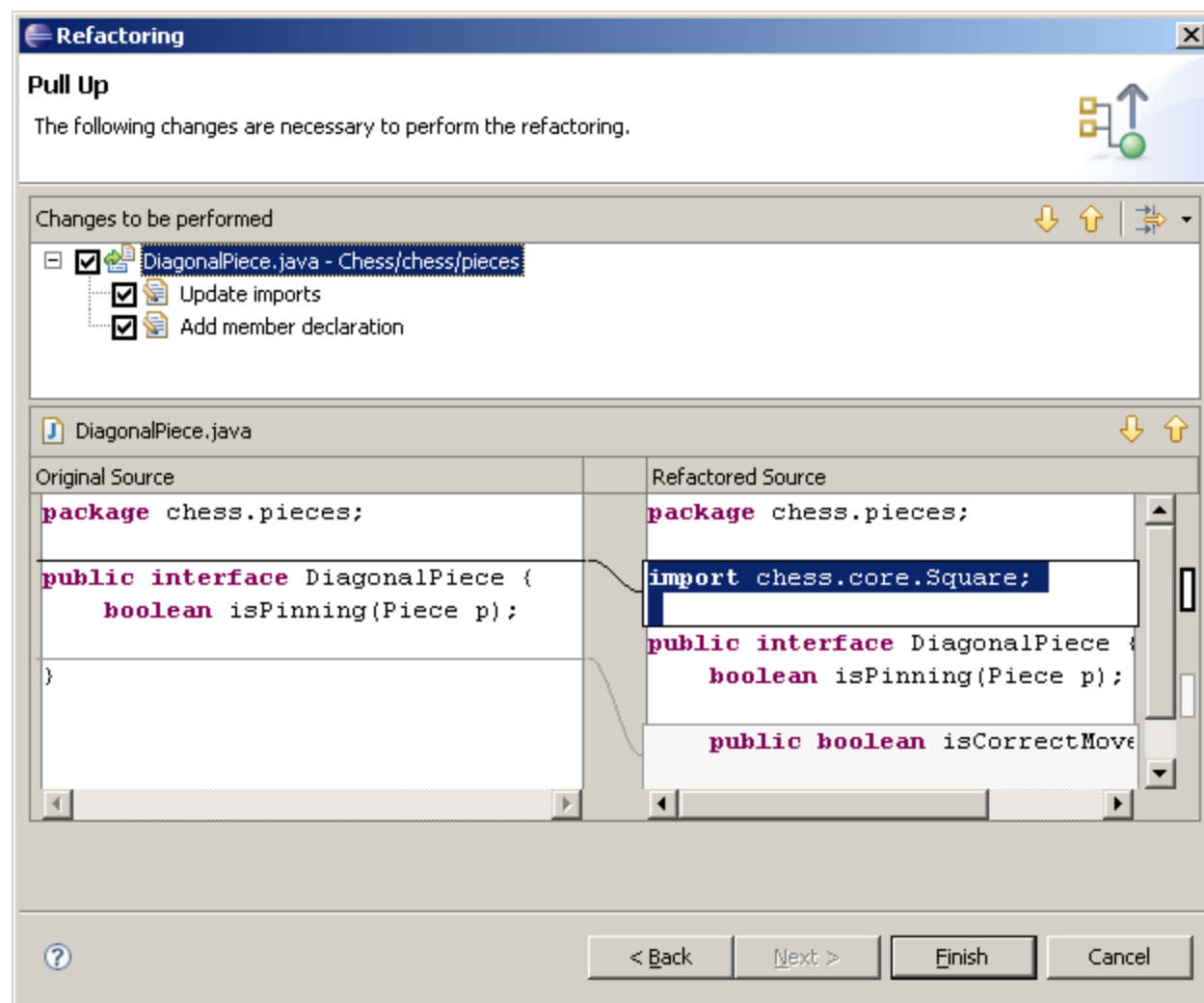
Романов Владимир Юрьевич ©2024



# Рефакторинг в Eclipse. Перемещение метода (5)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



# Рефакторинг в Eclipse.

## Использование полиморфизма (1)

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Замена имени класса-потомка на имя класса-предка
- После этого программа работает со всеми потомками класса-предка

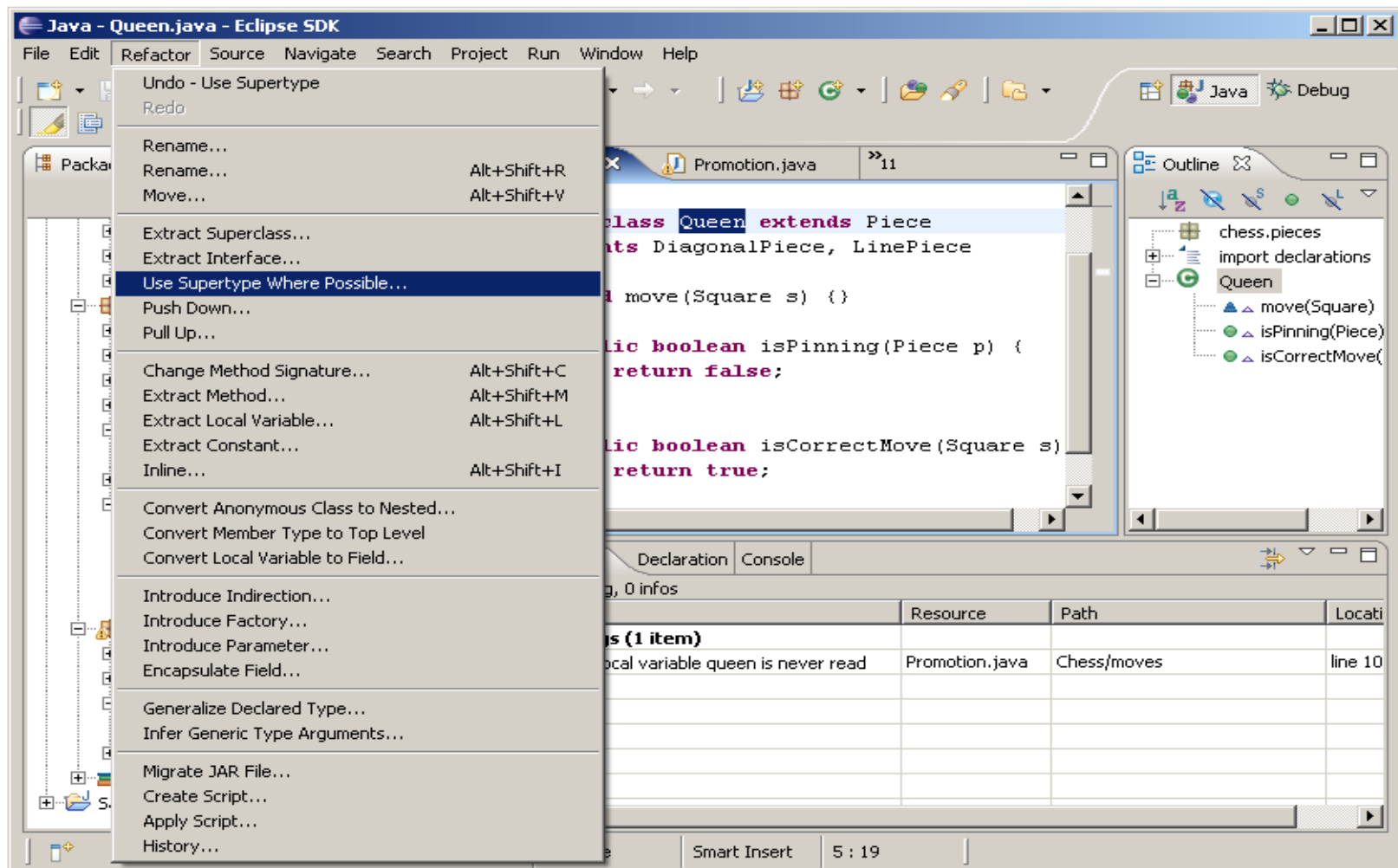


# Рефакторинг в Eclipse.

## Использование полиморфизма (2)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

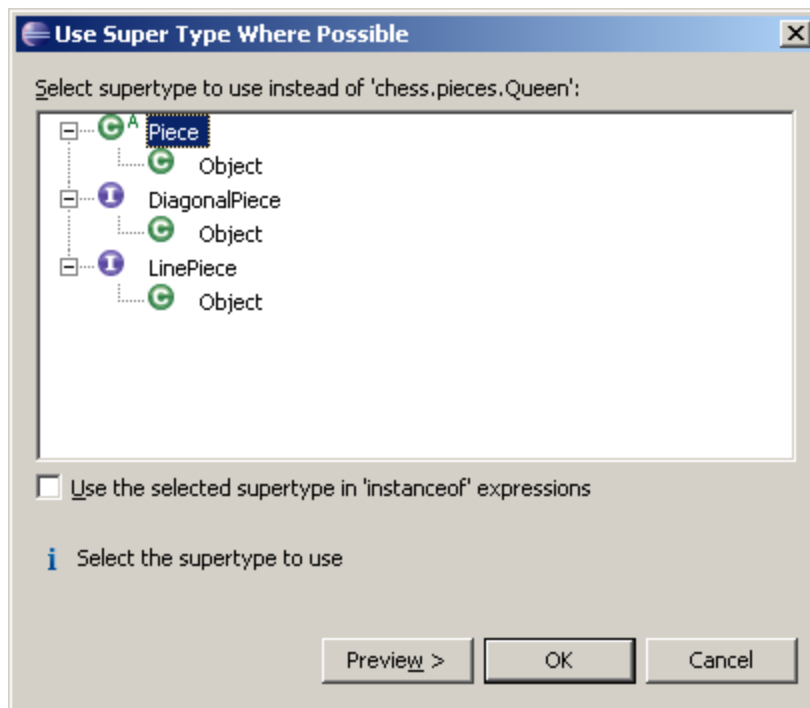
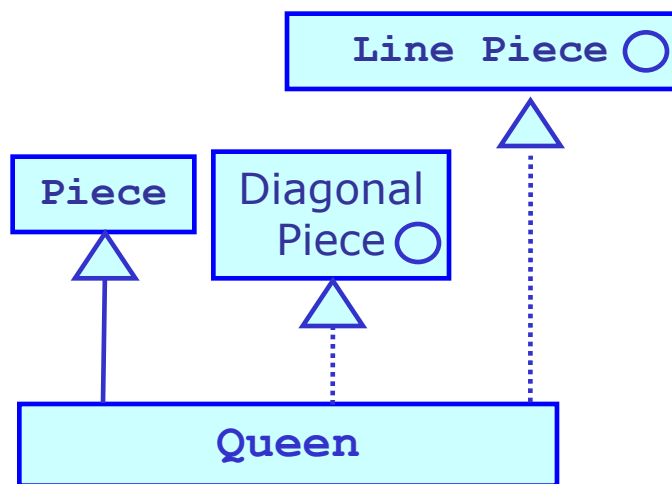


# Рефакторинг в Eclipse.

## Использование полиморфизма (3)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

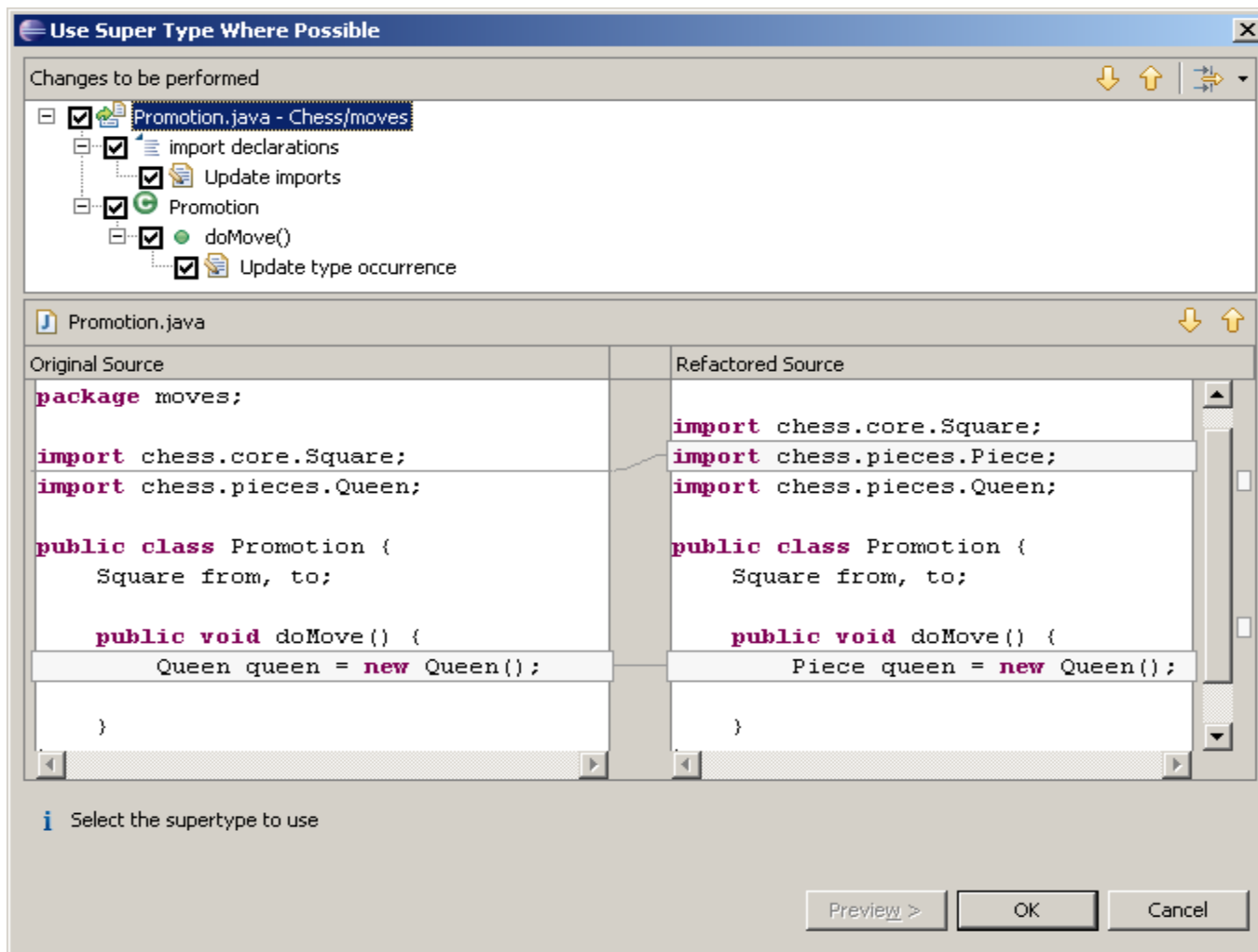


# Рефакторинг в Eclipse.

## Использование полиморфизма (4)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



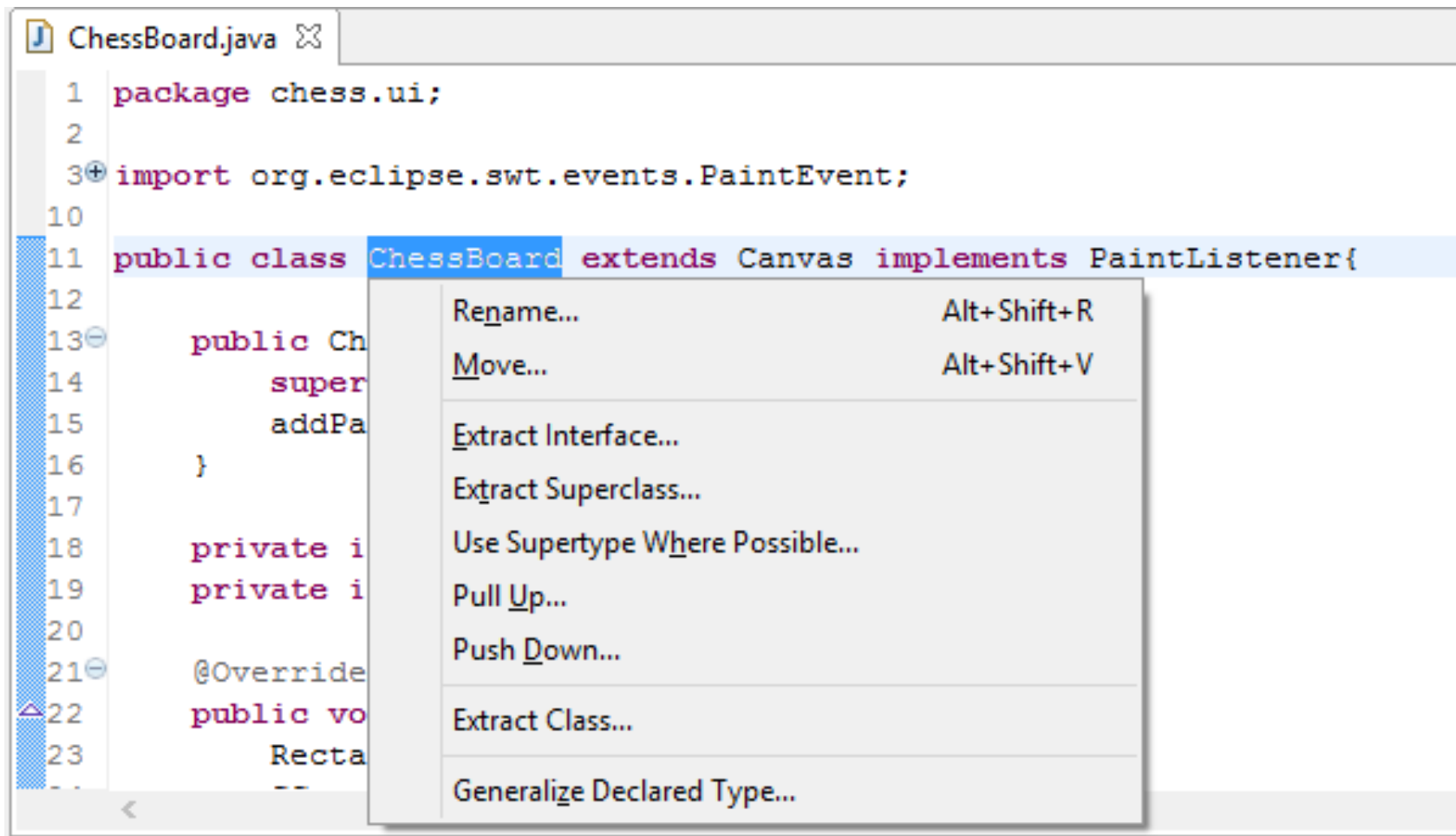
# Контексто-зависимый рефакторинг.

## Клавиши **Alt + Shift + T**

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Преобразования применимые к классу **ChessBoard**



The screenshot shows a code editor window titled 'ChessBoard.java'. The code is as follows:

```
1 package chess.ui;
2
3 import org.eclipse.swt.events.PaintEvent;
4
5
6
7
8
9
10
11 public class ChessBoard extends Canvas implements PaintListener{
12
13     public ChessBoard() {
14         super();
15         addPaintListener(this);
16     }
17
18     private int width;
19     private int height;
20
21     @Override
22     public void paint(Canvas canvas, Graphics g) {
23         Recta
```

A context menu is open over the 'ChessBoard' class declaration on line 11. The menu items and their keyboard shortcuts are:

- Rename... (Alt+Shift+R)
- Move... (Alt+Shift+V)
- Extract Interface...
- Extract Superclass...
- Use Supertype Where Possible...
- Pull Up...
- Push Down...
- Extract Class...
- Generalize Declared Type...

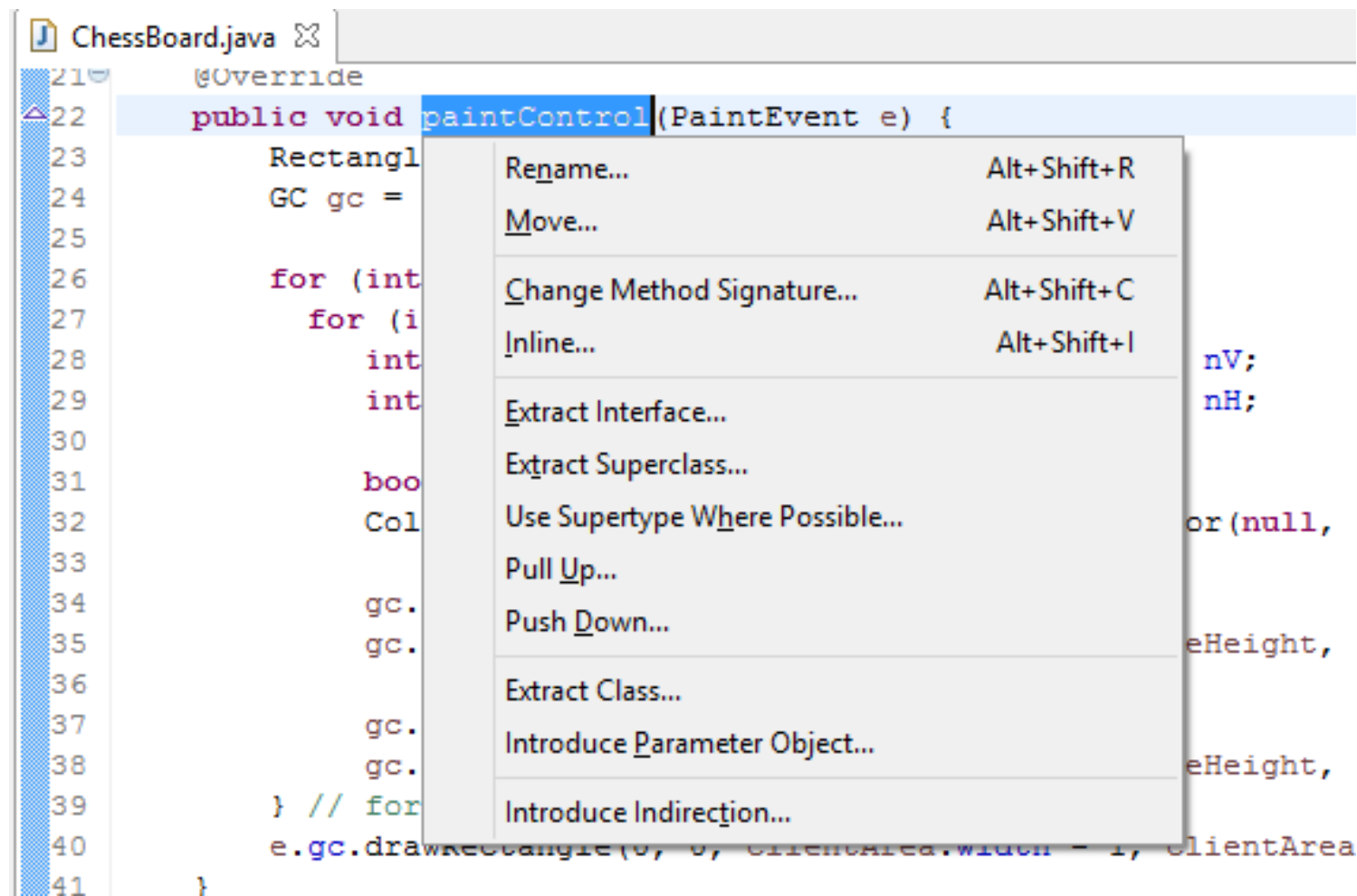
# Контексто-зависимый рефакторинг.

## Клавиши **Alt + Shift + T**

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Преобразования применимые к методу **paintControl**



```
ChessBoard.java x
21 @Override
22 public void paintControl(PaintEvent e) {
23     Rectangl
24     GC gc =
25
26     for (int
27         for (i
28         int
29         int
30
31     boo
32     Col
33
34     gc.
35     gc.
36
37     gc.
38     gc.
39     } // for
40     e.gc.drawRectangle(0, 0, clientArea.width - 1, clientArea
41 }
```

Rename...	Alt+Shift+R
Move...	Alt+Shift+V
Change Method Signature...	Alt+Shift+C
Inline...	Alt+Shift+I
Extract Interface...	
Extract Superclass...	
Use Supertype Where Possible...	
Pull Up...	
Push Down...	
Extract Class...	
Introduce Parameter Object...	
Introduce Indirection...	

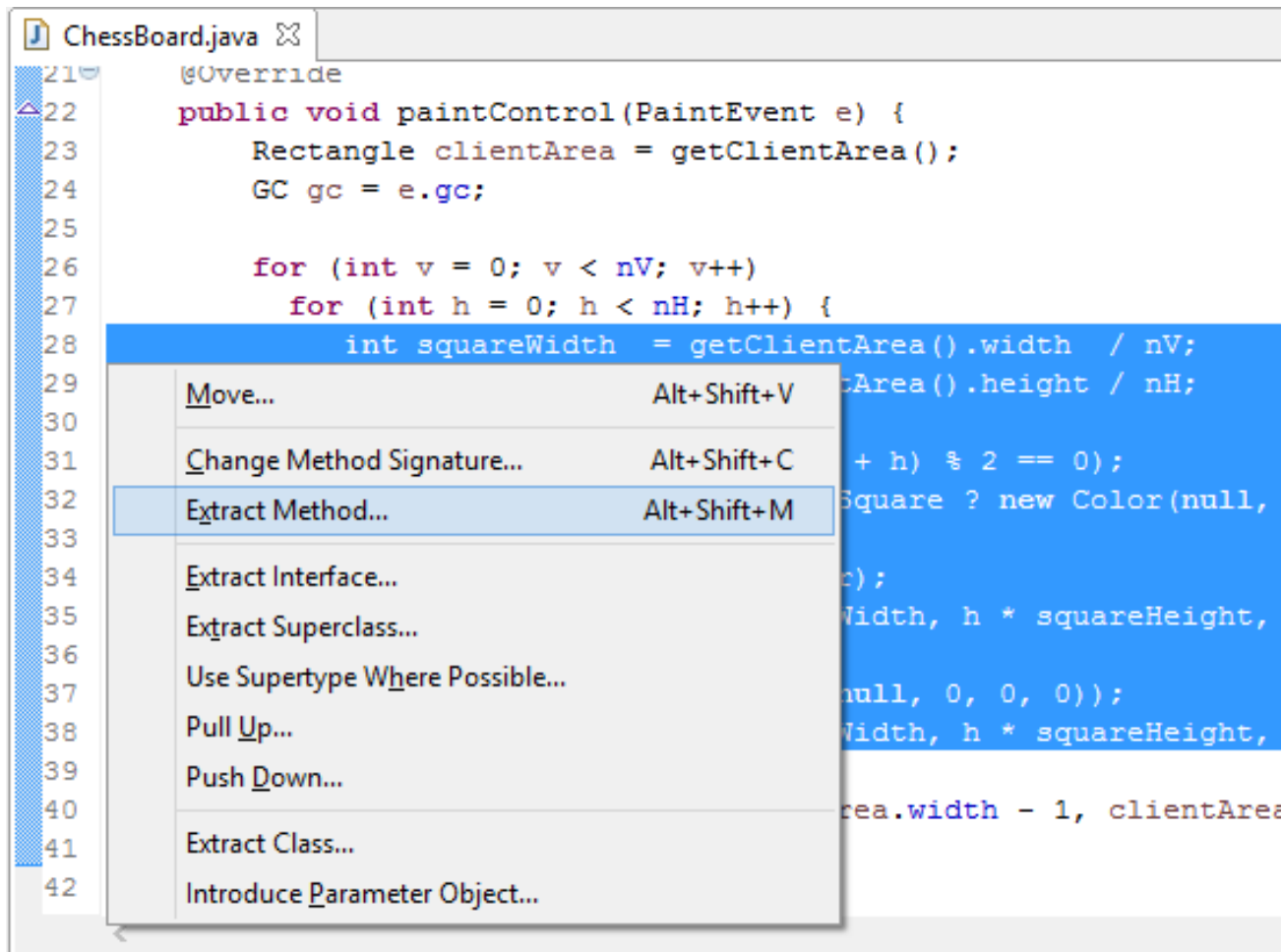
# Контексто-зависимый рефакторинг.

## Клавиши **Alt + Shift + T**

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Преобразования применимые к выделенному тексту



# Исключения и утверждения в языке Java

# Исключения в языке Java.

## Что изучается

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Как исключения используются для сигнализации об ошибках
- Как использовать конструкции ***try*** и ***catch*** для обработки исключений
- Как породить исключения
- Как использовать предложение ***assert***



# Исключения в языке Java.

## Определение

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Исключение
  - Событие или условие которое нарушает нормальный ход выполнения программы
  - Условие, которое приводит к *порождению* (**throw**) исключения системой
  - Поток управления прерывается и обработчик исключения будет *перехватывать* (**catch**) исключение
- Обработка исключения объектно-ориентированная
  - Локализует в объекте стандартные условия выполнения программы
  - Предоставляет простой способ сделать программу более надежной
  - Позволяет разделить нормальный и ненормальный ход выполнения программы

# Исключения в языке Java.

## Источники исключений


МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Источник исключений виртуальная машина Java
  - Не может быть загружен класс
  - Используется нулевая (***null***) ссылка на объект
- Ситуацию может породить и код, который пишет программист, и класс, который программист использует
  - **IOException**
  - Деление на ноль
  - Проверка корректности данных
  - Исключение обусловленное логикой работы программы
- Если исключение не перехвачено, оно завершает работу программы

```
float sales = getSales();  
int staffsize = getStaff().size;  
float avg_sales = sales/staffsize;  
System.out.println(avg_sales);
```

Деление на 0  
порождает ситуацию



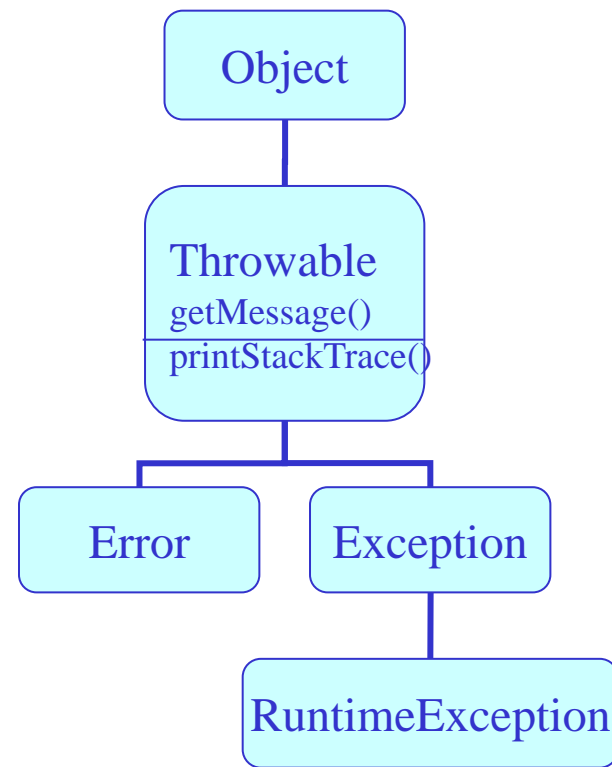
# Исключения в языке Java.

## Иерархия исключений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- **Throwable** – базовый класс, предоставляющий общий интерфейс и реализацию большинства исключений
- **Error** – отмечает серьезные проблемы, которые не могут быть перехвачены
- **Exception** – описывает класс условий, которые должны быть перехвачены или описаны как порождаемые
  - **RuntimeException** – это исключение может быть порождено при нормальном выполнении виртуальной машины
    - **ArithmeticException**
    - **BufferOverflowException**



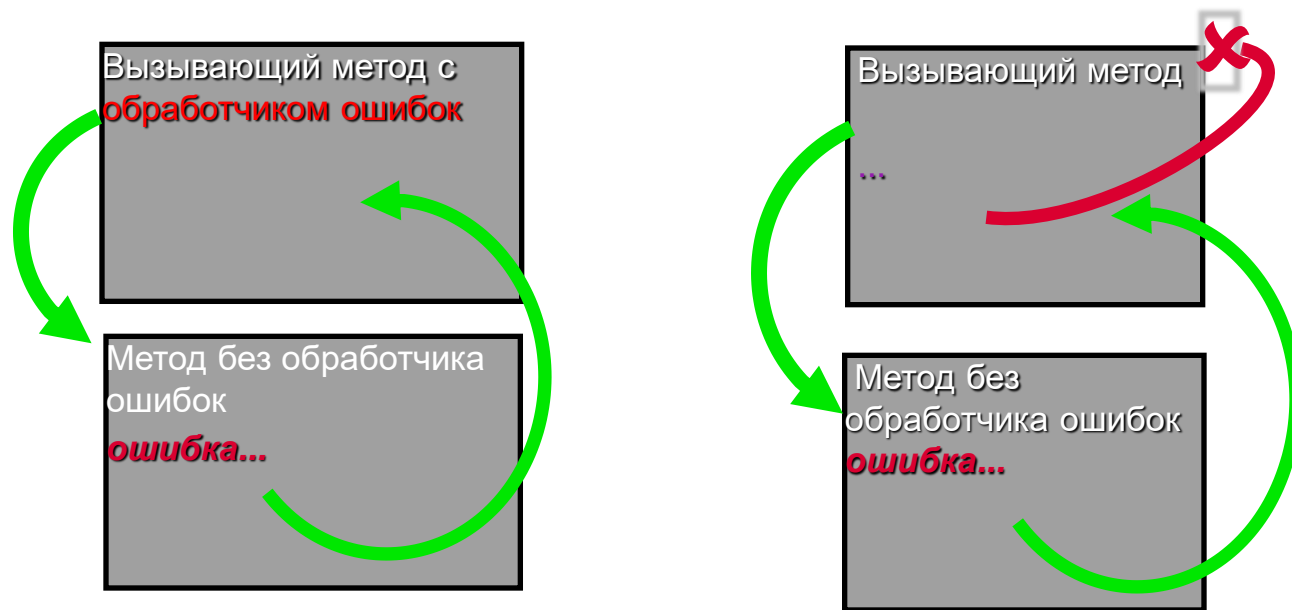
# Исключения в языке Java.

## Обработка исключений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Порожденные исключения обрабатываются либо в методе, в котором они были порождены, либо делегируются в вызывающий метод



# Исключения в языке Java.

## Конструкции обработки исключений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- ***throws*** – фрагмент объявления метода, содержащий список исключений который могут быть делегированы вверх по стеку вызовов
  - `public int doIt() throws SomeException, ...`
- ***try*** – представляет блок кода с присоединенными обработчиками ошибок. Ошибки в try-блоке будут обработаны обработчиками ошибок
- ***catch*** – блок кода для обработки конкретного исключения
- ***finally*** – необязательный блок который следует после блоков catch. Выполняется всегда независимо от того, какое исключение порождено и было ли оно порождено
- ***throw*** – явно порождает исключение
  - `throw new SomeException();`

# Исключения в языке Java.

## Блоки *try* и *catch*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Используются для обработки исключений
- Код, который может породить ошибку, заключается в блок ***try***
- Сразу за блоком ***try*** должен идти блок ***catch***

```
try {  
    // код выполняющий чтение из файла  
}  
catch (IOException ioe){  
    // код обрабатывающий ошибки ввода/вывода  
}
```

# Исключения в языке Java.

## Блок *catch*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Блок **catch** всегда содержит один аргумент определяющий тип перехватываемого исключения
- Аргументом может быть ссылка на объект класса **Throwable** или подкласс этого класса
- За одним блоком **try** может идти несколько блоков **catch**

```
try {  
    // код выполняющий чтение из файла  
}  
catch (FileNotFoundException fnf) {  
    // код обрабатывающий ошибку файл не найден  
}  
catch (IOException ioe){  
    System.out.println("I/O error " + e.getMessage() );  
}
```

# Исключения в языке Java.

## Блок *finally*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Блок ***finally*** необязателен. Выполняет действия независимо от того, было порождено исключение или нет
- Могут быть блоки ***try*** и ***finally*** без блоков ***catch***
- Блок ***catch*** выполняется после любого блока ***catch***, даже после того, который содержит предложение ***return***

```
try {  
    // код выполняющий чтение из файла  
}  
catch (FileNotFoundException fnf) {  
    // код обрабатывающий ошибку файл не найден  
}  
finally{  
    // закрытие файла  
}
```



# Исключения в языке Java.

## Предложение *throw*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Предложение может быть использовано в блоке ***try*** при необходимости порождения исключения
- Предложение ***throw*** требует только одного аргумента – объекта класса, являющегося потомком класса ***Throwable***
- Для того, что бы инкапсулировать условие, создается новый экземпляр класса ***Throwable***
- Поток управления завершается немедленно после предложения ***throw***

```
throw new java.io.IOException("msg");
```

# Исключения в языке Java.

## Предложение *throw*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Перехват исключения в вызывающем методе

```
public class ThrowDemo {  
  
    public void demoThrowMethod(int n) throws Exception {  
        if (n < 0)  
            throw new Exception("n < 0");  
    }  
  
    public static void main(String[] args) {  
        ThrowDemo app = new ThrowDemo();  
        try {  
            app.demoThrowMethod(-1);  
        } catch (Exception e) {  
            System.err.println(e.getMessage());  
        }  
    }  
}
```

# Исключения в языке Java.

## Предложение *throw*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Перехват исключения в порождающей ситуации методе

```
public class ThrowDemo {  
  
    public void demoThrowMethod(int n) {  
        try {  
            if (n < 0)  
                throw new Exception("n < 0");  
        } catch (Exception e) {  
            System.err.println(e.getMessage());  
        }  
    }  
  
    public static void main(String[] args) {  
        ThrowDemo app = new ThrowDemo();  
        app.demoThrowMethod(-1);  
    }  
}
```

# Исключения в языке Java.

## Предложение *throw*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
public void doMove() throws GameOver {  
    target.setPiece(piece);  
    changeCapturedColor();  
  
    Board board = target.getBoard();  
  
    List<Square> empties = board.getEmptySquares();  
    if (!empties.isEmpty()) return;  
  
    int enemies = piece.getEnemies().size();  
    int friends = piece.getFriends().size();  
  
    if (enemies == friends)  
        throw new GameOver(GameResult.DRAWN);  
}
```

# Утверждения в языке Java.

## Когда использовать утверждения

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Не рекомендуется использовать утверждения
  - Для проверки аргументов в публичных методах. Для этого требуются ***RuntimeException***, например ***IllegalArgumentException***
- Рекомендуется использовать утверждения для проверки
  - Внутренних инвариантов (значения, которые никогда не должны возникать). Например, вставить **`default: assert false`** в конец выбирающего предложения
  - Инварианты потоков управления. Например, вставить **`assert false`** в те части программы, которые никогда не должны быть достигнуты
  - Предусловия, постусловия и инварианты классов. Например, проверка аргументов скрытых методов

# Утверждения в языке Java.

## Использование утверждений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Две формы утверждений

```
assert <boolean expression> ;
assert <boolean expression> : <value expression> ;
```
- Если утверждение ложно
  - первая форма утверждения порождает исключение ***AssertionError*** без сообщений
  - вторая форма утверждения порождает исключение ***AssertionError*** с сообщением, определенным при вычислении второго выражения
- По умолчанию утверждения не работают (игнорируются). Для их включения необходимо в командной строке для компилятора **java** использовать ключ **enableassertions**
- Утверждения **включаются/выключаются** для **классов/пакетов**

# Утверждения в языке Java.

## Использование утверждений

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Две формы утверждений

```
assert <boolean expression> ;
assert <boolean expression> : <value expression> ;
```
- Если утверждение ложно
  - первая форма утверждения порождает исключение **AssertionError** без сообщений
  - вторая форма утверждения порождает исключение **AssertionError** с сообщением, определенным при вычислении второго выражения
- По умолчанию утверждения не работают (игнорируются). Для их включения необходимо в командной строке java использовать ключ **enableassertions**
- Утверждения включаются/выключаются для классов/пакетов

# Интерфейсы в языке Java



# Интерфейсы в языке Java.

## Объявление интерфейса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Интерфейс – именованный список объявлений методов
  - Методы только объявляются, а не реализуются
  - Интерфейс похож на абстрактный класс, но тем не менее отличается от него
- Объявляемые в Java типы либо классы, либо интерфейсы, либо перечисления
- Интерфейс можно трактовать как контракт – обязательство объектов реализовать некоторый набор услуг

```
package game.moves ;

interface Move {
    void doMove () ;
    void undoMove () ;
}
```

# Интерфейсы в языке Java.

## Иерархия интерфейсов (1)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Между интерфейсами возможно отношение **наследования**
- Интерфейс, расширяющий другой интерфейс, **наследует** все объявления методов интерфейса - предка
- Иерархия наследования интерфейсов независима от иерархии наследования классов

```
public interface IPutMove extends Move {  
    /**  
     * Вернуть клетку куда пошла фигура.  
     */  
    Square getTarget();  
}
```

```
public interface ITransferMove extends Move {  
    Square getSource();  
    Square getTarget();  
}
```

```
public interface ICaptureMove extends Move {  
    /**  
     * Вернуть клетки на которых стоят захваченные фигуры.  
     */  
    List<Square> getCaptured();  
}
```

# Интерфейсы в языке Java.

## Реализация интерфейсов классом

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
package chess.moves;

public class Promotion implements ITransferMove , ICaptureMove {
    void doMove() {
        //...
    }
    void undoMove() {
        // ...
    }
    Square getSource() {
        // ...
    }
    Square getTarget() {
        // ...
    }
    List<Square> getCaptured() {
        // ...
    }
}
```

# Интерфейсы в языке Java.

## Реализация интерфейса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Методы объявленные в интерфейсе реализуются в классе, поддерживающем данный интерфейс

  
Определение

```
public interface ITransferMove extends Move {  
    Square getSource();  
    Square getTarget();  
}
```

```
public interface ICaptureMove extends Move {  
    List<Square> getCaptured();  
}
```

Определение  


Реализация  


```
package reversi.moves;  
  
public class Capture implements ITransferMove ,  
    ICaptureMove {  
    void doMove() { ... }  
    void undoMove() { ... }  
    Square getSource() { ... }  
    Square getTarget() { ... }  
    List<Square> getCaptured() { ... }  
}
```

# Интерфейсы в языке Java.

## Синтаксис реализации интерфейса

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Объявление суперкласса всегда предшествует объявлению интерфейсов реализуемых классом:

```
public class Directory extends Secure
    implements File {
    ...
}
```

- Если класс реализует несколько интерфейсов, то имена этих интерфейсов перечисляются через запятую:

```
public class Directory
    implements File, ISecure {
    ...
}
```

# Интерфейсы в языке Java.

## Типизация и интерфейсы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Типом переменной и параметра может быть интерфейс
- Переменной и параметру может быть присвоен только объект, реализующий этот интерфейс
- Переменная и параметр могут быть использованы только для вызова методов, определенных в интерфейсе
- Имя интерфейса не может быть в выражении ***new***

```
Move m1 = new Move(); // Ошибка
```

```
Move m2 = new Castling(); // Допустимо
```

# Интерфейсы в языке Java.

## Использование интерфейсов (2)

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Использование интерфейсов позволяет
  - Использовать полиморфизм независимо от иерархии классов
  - Осуществлять доступ к методам в отдельных независимых деревьях классов
  - Использовать в переменных и параметрах объекты не связанные иерархией наследования классов
- Классы, реализующие один и тот же интерфейс, понимают те же самые сообщения независимо от положения в иерархии классов

# Интерфейсы в языке Java.

## Соглашения об именах интерфейсов

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Суффикс "able" в именах интерфейсов
  - Cloneable, Serializable
- Существительное + Impl
  - Bank, BankImpl
  - BankAccount, BankAccountImpl
- Префикс I перед существительным
  - Bank, IBank
  - BankAccount, IBankAccountImpl



# Интерфейсы в языке Java.

## Использование интерфейсов (2)

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Программист может определить параметры метода как интерфейсы
  - Это ограничит использование этих параметров только типами, которые реализуют этот интерфейс
  - Более четкое указание программисту какие методы он может использовать
- Увеличивает повторное использование кода, использующего преимущественно типы - интерфейсы

# Вложенные классы и интерфейсы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Допускается вложенность интерфейсов в классы и другие интерфейсы
- Вложенность классов и интерфейсов – дополнительный способ структурирования программы. Вложенность возможна не только в пакеты.
- Вложенные интерфейсы могут иметь видимости как у полей и методов классов и интерфейсов
  - **public**
  - **protected**
  - **private**
  - **package**
- Вложенные классы имеют доступ к полям и методам охватывающих классов и интерфейсов.

# Коллекции в языке JAVA

# Коллекции языка Java.

## Что изучается

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Понимание основных понятий коллекций
- Оценка основных интерфейсов коллекций
  - интерфейсы
  - абстрактные типы
  - конкретные реализации
- Понять как "устаревшие" классы и интерфейсы связаны с новыми интерфейсами и классами

# Коллекции языка Java.

## Что такое коллекция

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Коллекция – это объект который группирует несколько элементов в единую сущность
- Массивы языка Java также рассматриваются как коллекции
- Виды коллекций
  - *Set* (множество) – не может иметь повторяющиеся элементы. Например, книги в библиотеке
  - *List* (список) – упорядоченная коллекция, может содержать повторения. Например, список чисел
  - *Map* (карта) – объекты, которые отображают ключи на значения. Дублирование ключей недопустимо. Например, словарь, список свойств

# Коллекции языка Java.

## *Java Collection Framework*

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- **JCF** - унифицированная архитектура для представления и манипулирования коллекциями
- Состоит из трех частей
  - Интерфейсы
  - Реализации
  - Алгоритмы

# Коллекции языка Java.

## Интерфейсы, реализации, алгоритмы

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- **Интерфейсы** – абстрактные типы данных представляющие коллекции. Назначение коллекций:
  - позволить манипулирование коллекциями независимо от их деталей их представления
  - предоставить точки расширения для добавления новых типов коллекций и их реализаций
- **Реализации** – конкретная реализация интерфейса коллекции
- **Алгоритмы** – методы, выполняющие полезные вычисления над объектами, которые реализуют интерфейс коллекций. Например, поиск и сортировку.
  - Предоставляют повторно используемую функциональность посредством полиморфизма – один алгоритм для разных реализаций

# Коллекции языка Java.

## Достоинства использования JCF

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Уменьшает усилия по программированию
- Уменьшает усилия по изучению и использованию нового API
- Увеличивает скорость и надежность программы
- Допускает переносимость среди связанных API

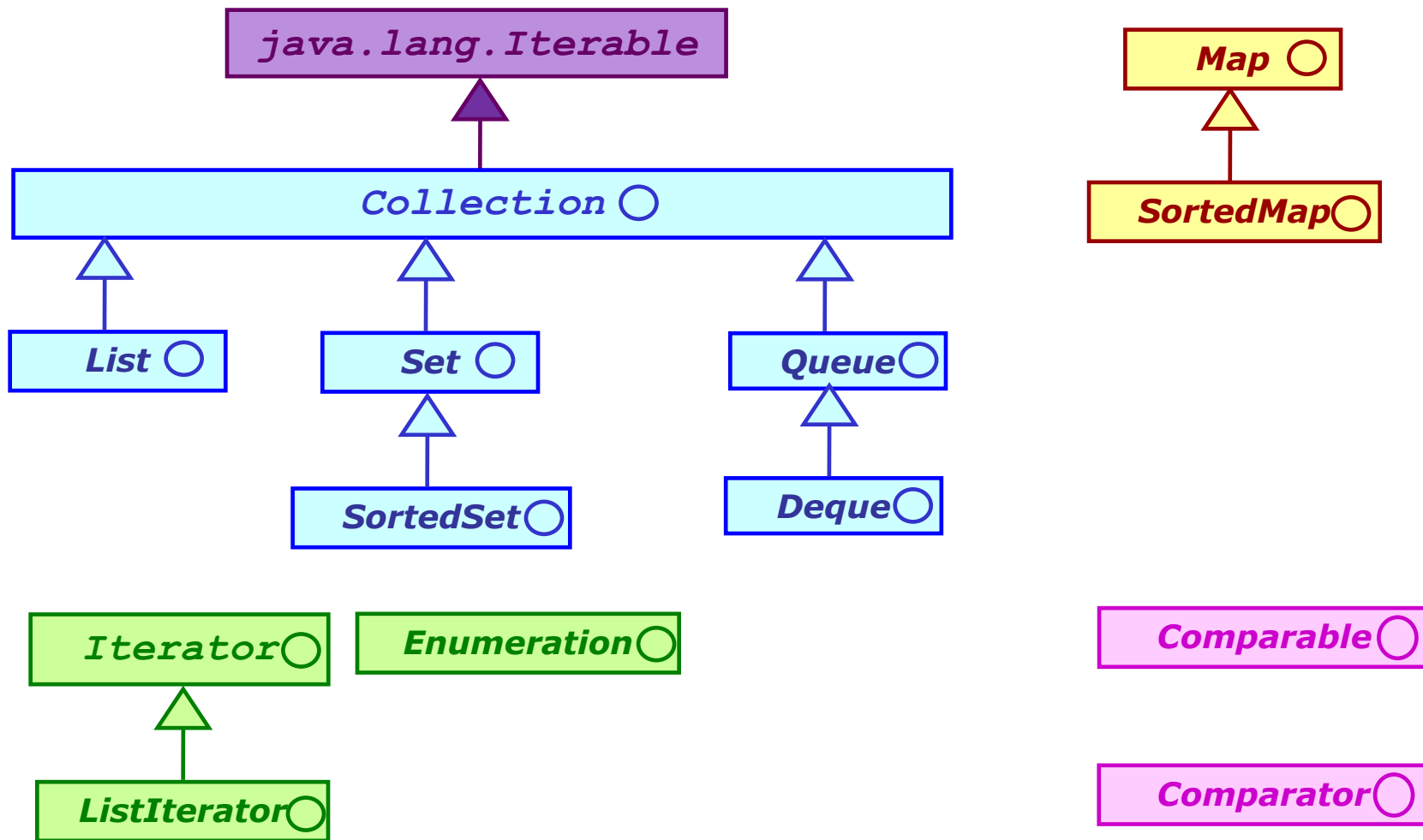


# Коллекции языка Java.

## Интерфейсы в коллекциях

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



# Коллекции языка Java.

## Интерфейс *Collection*

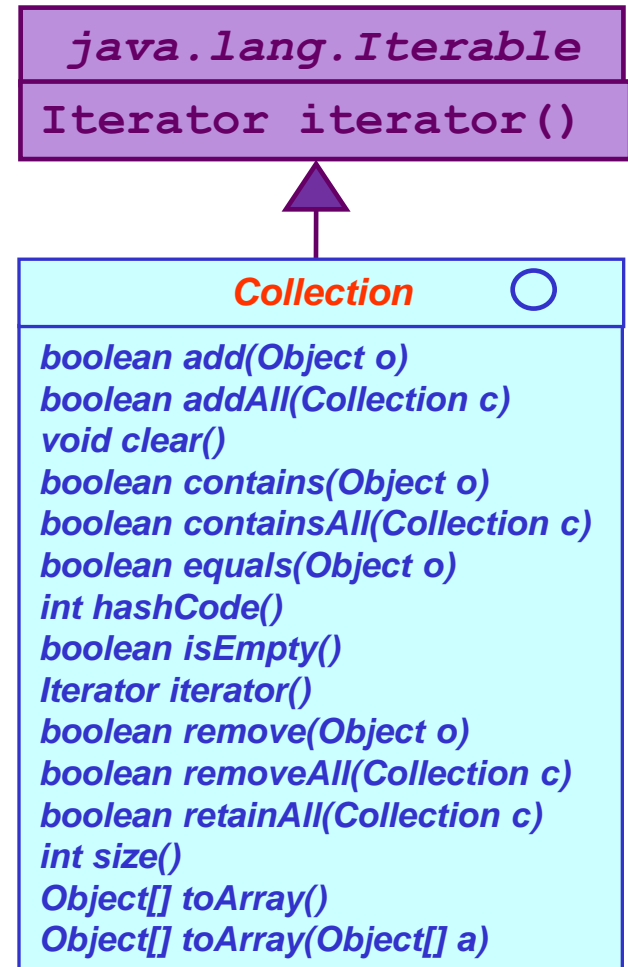
МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Этот универсальный интерфейс для изменения коллекций и прохода по элементам коллекций
- Проверки принадлежности элемента к коллекции
- Добавления элемента к коллекции
- Удаления элемента из коллекции

```
Collection c;

for(Object o : c) {
    System.out.println(o);
}
```

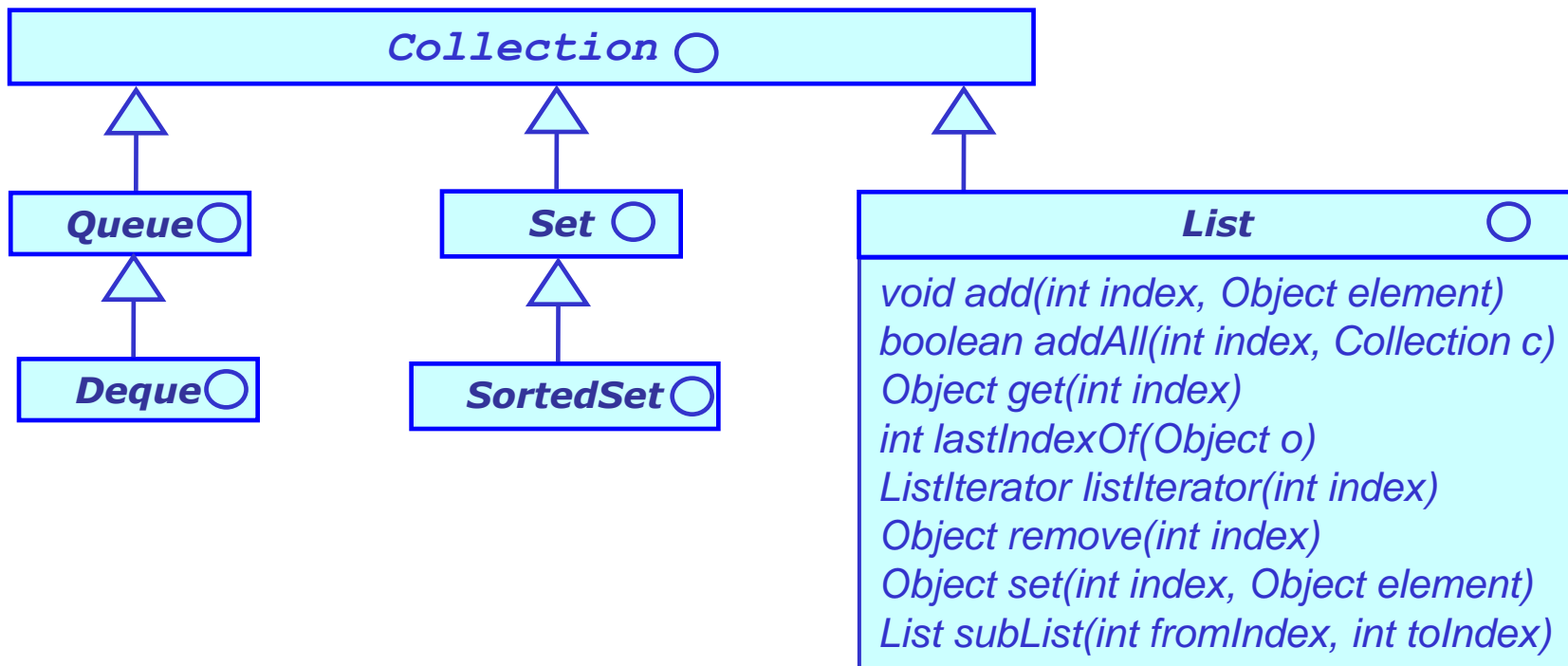


# Коллекции языка Java.

## Интерфейс списка

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

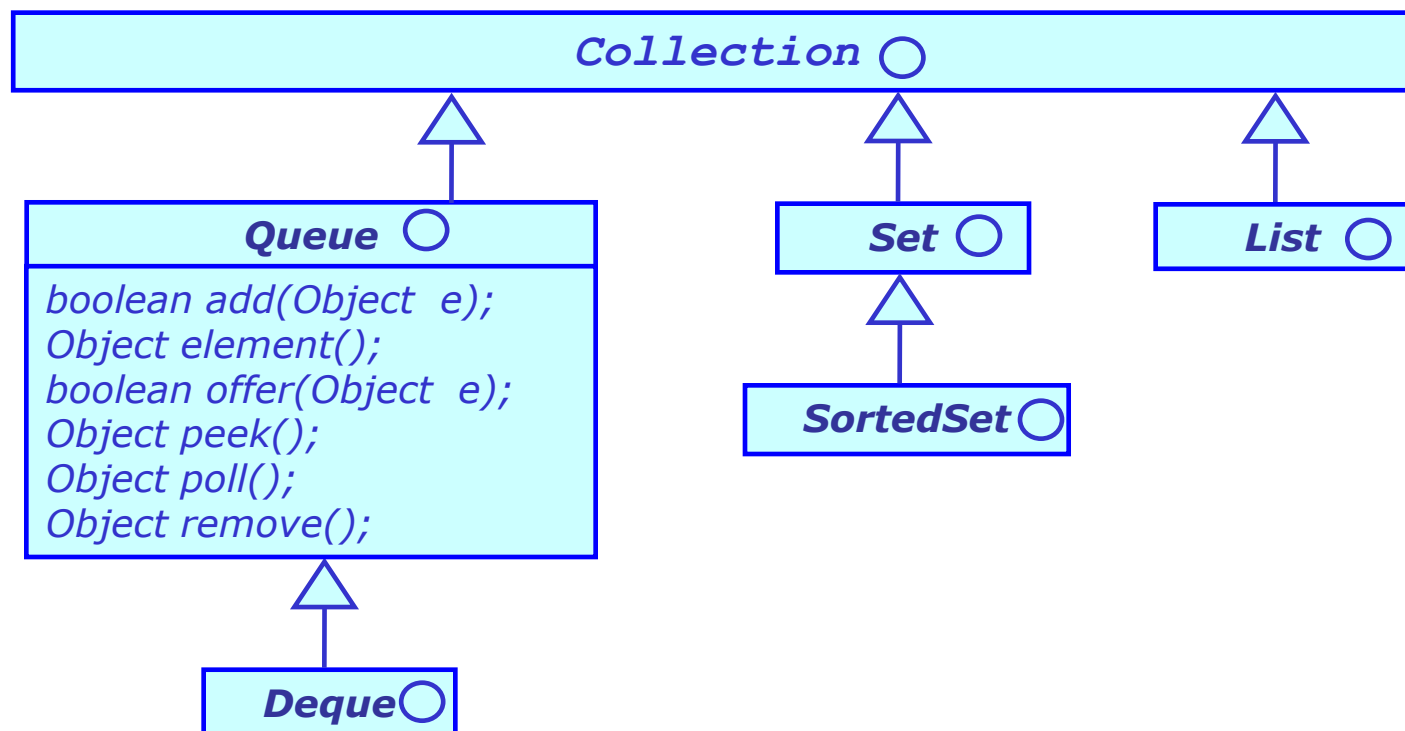


# Коллекции языка Java.

## Интерфейс очереди Queue

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024



# Коллекции языка Java.

## Интерфейс очереди Queue

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

Действия по добавлению, удалению и проверки элемента в очереди.

Различие – что делается если действие выполнить невозможно

<b>Queue</b> ○
<i>boolean add(Object e);</i>
<i>Object element();</i>
<i>boolean offer(Object e);</i>
<i>Object peek();</i>
<i>Object poll();</i>
<i>Object remove();</i>

	<b>Порождение ситуации</b>	<b>Возвращается значение или null</b>
<b>Вставка</b>	<i>boolean add(Object e)</i>	<i>boolean offer(Object e)</i>
<b>Удаление</b>	<i>Object remove()</i>	<i>Object poll()</i>
<b>Проверка</b>	<i>Object element()</i>	<i>Object peek()</i>

# Коллекции языка Java.

## Интерфейс очереди Queue

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

### **boolean add(E e)**

Вставляет указанный элемент в эту очередь, если это возможно сделать немедленно, не нарушая ограничения емкости, возвращая значение **true** в случае успеха и вызывая исключение ***IllegalStateException***, если в настоящее время нет свободного места.

### **E element()**

Извлекает, но не удаляет заголовок этой очереди. Этот метод отличается от ***peek*** только тем, что выдает исключение, если эта очередь пуста.

Возвращает: заголовок этой очереди

Порождает ситуацию ***IllegalStateException*** если очередь пуста.

### **E remove()**

Возвращает, удаляя, элемент в заголовке очереди.

Порождает ситуацию ***IllegalStateException*** если очередь пуста.

# Коллекции языка Java.

## Интерфейс очереди Queue

---

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

### **boolean offer(E e)**

Вставляет описанный элемент в очередь, если возможно сделать это немедленно без нарушения ограничений по плотности.

Возвращает значение **true** если вставка прошла успешно и **false** иначе.

### **E peek()**

Возвращает, не удаляя, элемент в заголовке очереди.

Возвращает **null** если очередь пуста.

### **E poll()**

Возвращает, удаляя, элемент в заголовке очереди.

Возвращает **null** если очередь пуста.

# Коллекции языка Java.

## Интерфейс очереди Deque (1)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

<b>Deque</b> ○
<pre>boolean addFirst(Object e); Object elementFirst (); boolean offerFirst (Object e); Object peekFirst (); Object pollFirst (); Object removeFirst ();</pre>

	<b>Ситуация</b>	<b>Значение или null</b>
<b>Вставка</b>	<i>boolean addFirst(Object e)</i>	<i>boolean offerFirst(Object e)</i>
<b>Удаление</b>	<i>Object removeFirst()</i>	<i>Object pollFirst()</i>
<b>Проверка</b>	<i>Object elementFirst()</i>	<i>Object peekFirst()</i>



# Коллекции языка Java.

## Интерфейс очереди Deque (2)

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

<b>Deque</b> ○
<pre>boolean addLast(Object e); Object elementLast (); boolean offerLast (Object e); Object peekLast (); Object pollLast (); Object removeLast ();</pre>

	<b>Ситуация</b>	<b>Значение или null</b>
<b>Вставка</b>	<i>boolean addLast(Object e)</i>	<i>boolean offerLast(Object e)</i>
<b>Удаление</b>	<i>Object removeLast()</i>	<i>Object pollLast()</i>
<b>Проверка</b>	<i>Object elementLast()</i>	<i>Object peekLast()</i>

# Коллекции языка Java.

## Интерфейс *Map*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Карта отображает ключи на значения
- Добавление/удаление пары ключ-значение
- Взять значение для данного ключа
- Проверить наличие элемента в карте
- Можно рассматривать карту как:
  - Множество ключей
  - Множество пар ключ-значение
  - Коллекцию значений

```
Map ○  
void clear()  
boolean containsKey(Object key)  
boolean containsValue(Object value)  
Set entrySet()  
boolean equals(Object o)  
Object get(Object key)  
int hashCode()  
boolean isEmpty()  
Set keySet()  
Object put(Object key)  
void putAll(Map t)  
Object remove(Object key)  
int size()  
Collection values()
```

# Сравнение объектов.

## Упорядочивание объектов

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Для сортировки объектов в коллекциях, необходим способ упорядочивания объектов
  - Объект **A** идет перед объектом **B**
  - Объект **B** идет перед объектом **A**
  - Объект **A** и объект **B** равны
- Существует два способа упорядочения объектов при сортировке:
  - Интерфейс **Comparable**
  - Интерфейс **Comparator**

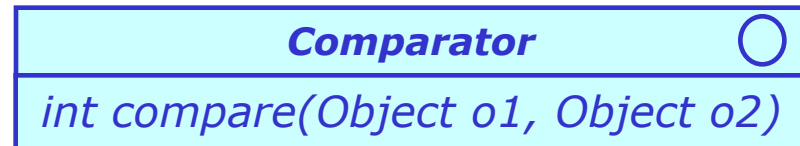
# Сравнение объектов.

## Интерфейсы *Comparable* и *Comparator*

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Интерфейс *Comparable*
  - Реализуют классы, объекты которых способны сравнить себя с другими объектами сами
  - Такие классы называются классами с естественным упорядочиванием
- Интерфейс *Comparator*
  - Реализует класс, назначение которого сравнивать один объект с другим
  - Два объекта могут сравниваться по разному



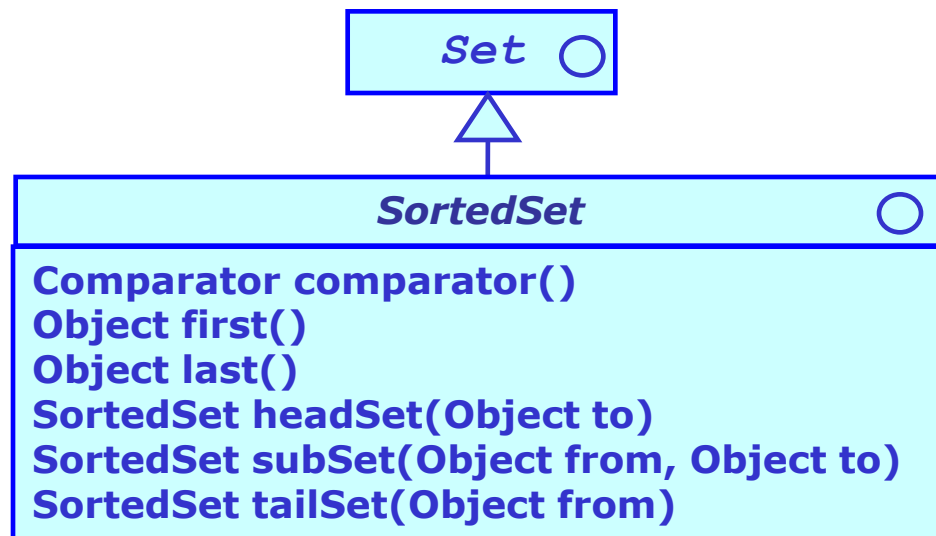
# Сортированные коллекции.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- **SortedSet** (Сортированное Множество) – это множество (**Set**) с встроенным и автоматически поддерживаемым упорядочением
  - Существуют методы для использования этого порядка
- **SortedMap** (Сортированная Карта) – это карта (**Map**) имеет аналогичные свойства, основанные на упорядочении ключей карты

- Порядок может определяться как с помощью естественного порядка, так и с помощью класса реализующего интерфейс `Comparator`

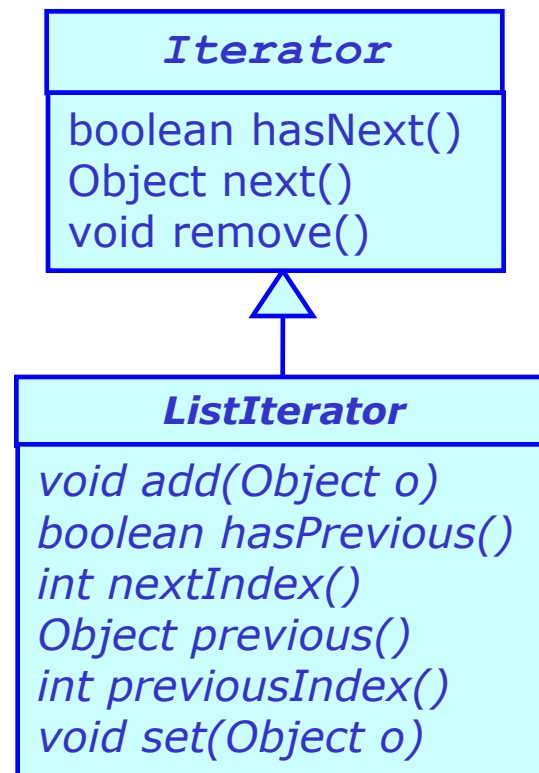


# Итераторы.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

- Итератор предоставляет удобный способ перебора элементов коллекции
- *ListIterator* – добавляет методы представляющие последовательность элементов этой коллекции
- Методы *add* и *remove* позволяют изменять коллекцию во время ее прохода
- Итераторы для сортированных коллекций учитывают порядок заданный при сортировке



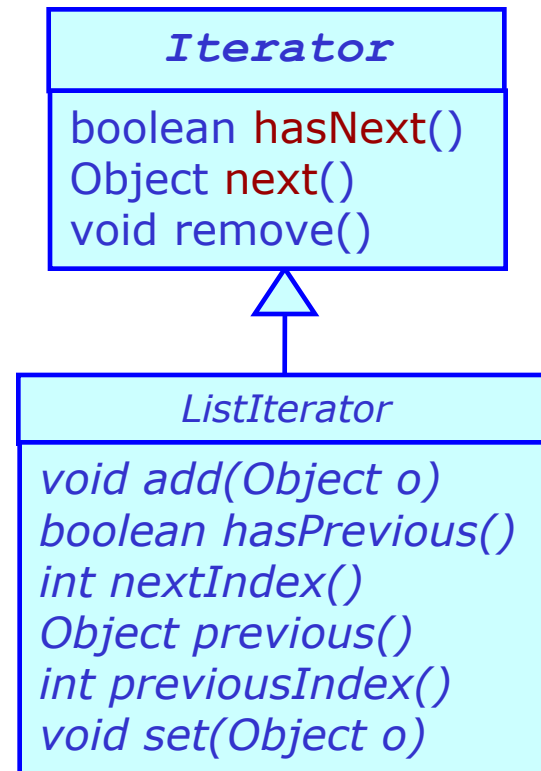
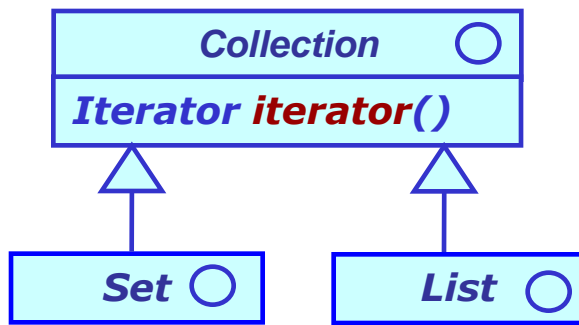
# Шаблон кода итератора.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
Collection c;

Iterator i = c.iterator();
while (i.hasNext()) {
    Object o = i.next();
    System.out.println(o);
}
```



# Итератор встроенный в язык Java.

МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2024

```
Collection c;
//...
for(Object o : c) {
    // process this object
}
```

