

# 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),  
[romanov.rvy@yandex.ru](mailto:romanov.rvy@yandex.ru)

# Литература

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

Романов Владимир Юрьевич ©2025

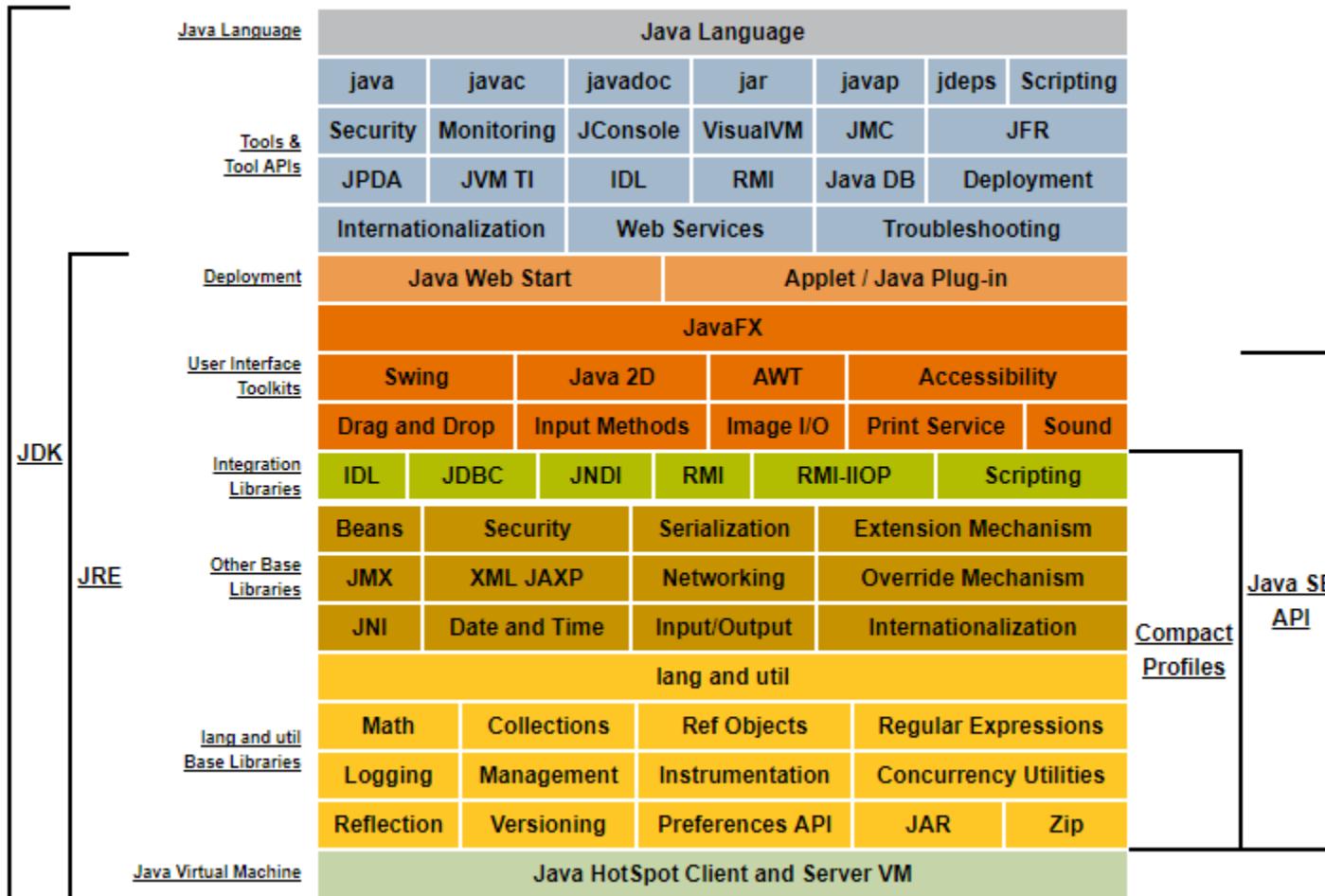
1. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley  
Java(TM) Programming Language, Java SE 8 Edition  
<https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf>
2. Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley  
The JavaTM Virtual Machine Specification, Java SE 8 Edition  
<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 ©2025

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



# Цели курса

---

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- Визуализация промежуточного представления компилятора с помощью *Graphic Editing Framework*
- Моделирование программ с помощью реализованной в среде Eclipse метамодели 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 ©2025

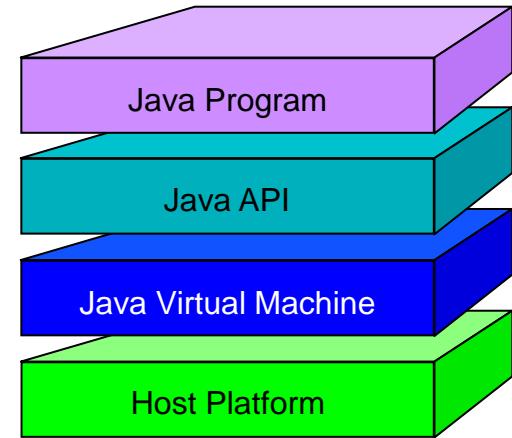
- Простота
- Язык высокого уровня
- Объектно-ориентированный
- Независим от архитектуры
- Переносим в исходных текстах и в двоичном коде
- Интерпретируемый и компилируемый
- Многопоточный

# Платформа Java

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

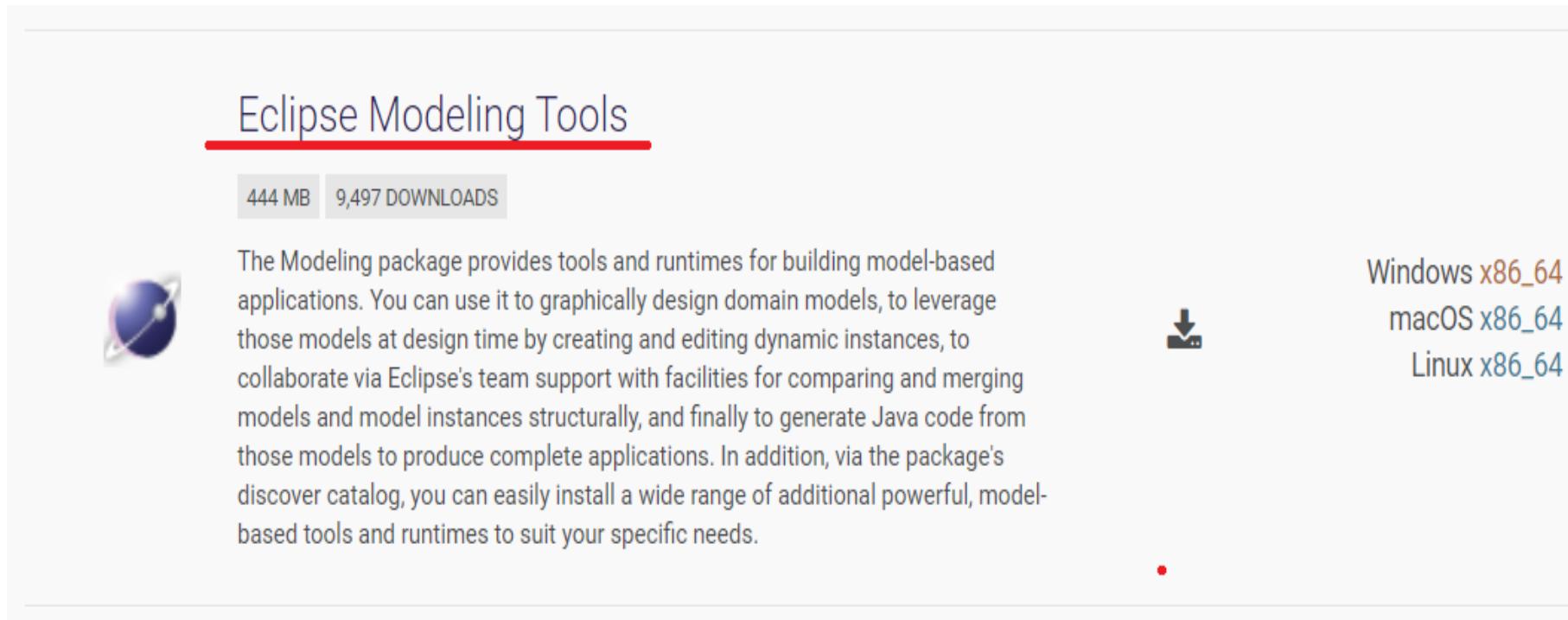


# Установка Eclipse. Версии среды

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

Романов Владимир Юрьевич ©2025

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



The screenshot shows a download page for the Eclipse Modeling Tools. At the top, the title 'Eclipse Modeling Tools' is underlined. Below it, a download button shows '444 MB' and '9,497 DOWNLOADS'. To the left is a small icon of a sphere with a grid. The main text describes the package's purpose: '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.' To the right, there are download links for 'Windows x86\_64', 'macOS x86\_64', and 'Linux x86\_64', each with a small download icon.

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 ©2025

<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 ©2025

<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

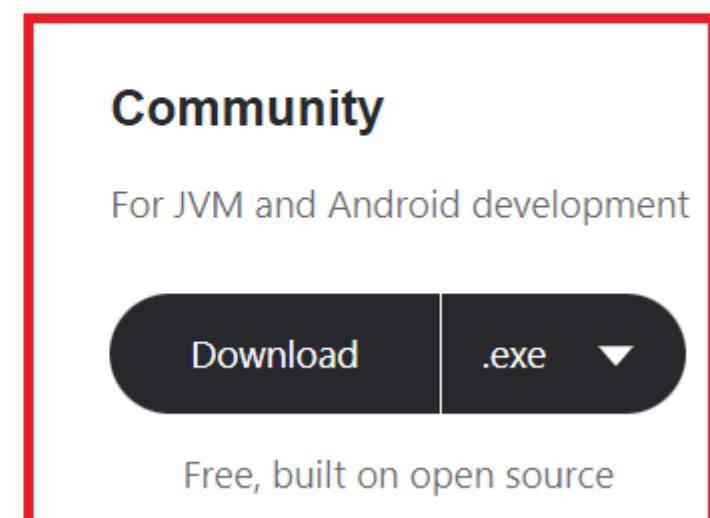
### Community

For JVM and Android development

Download

.exe

Free, built on open source



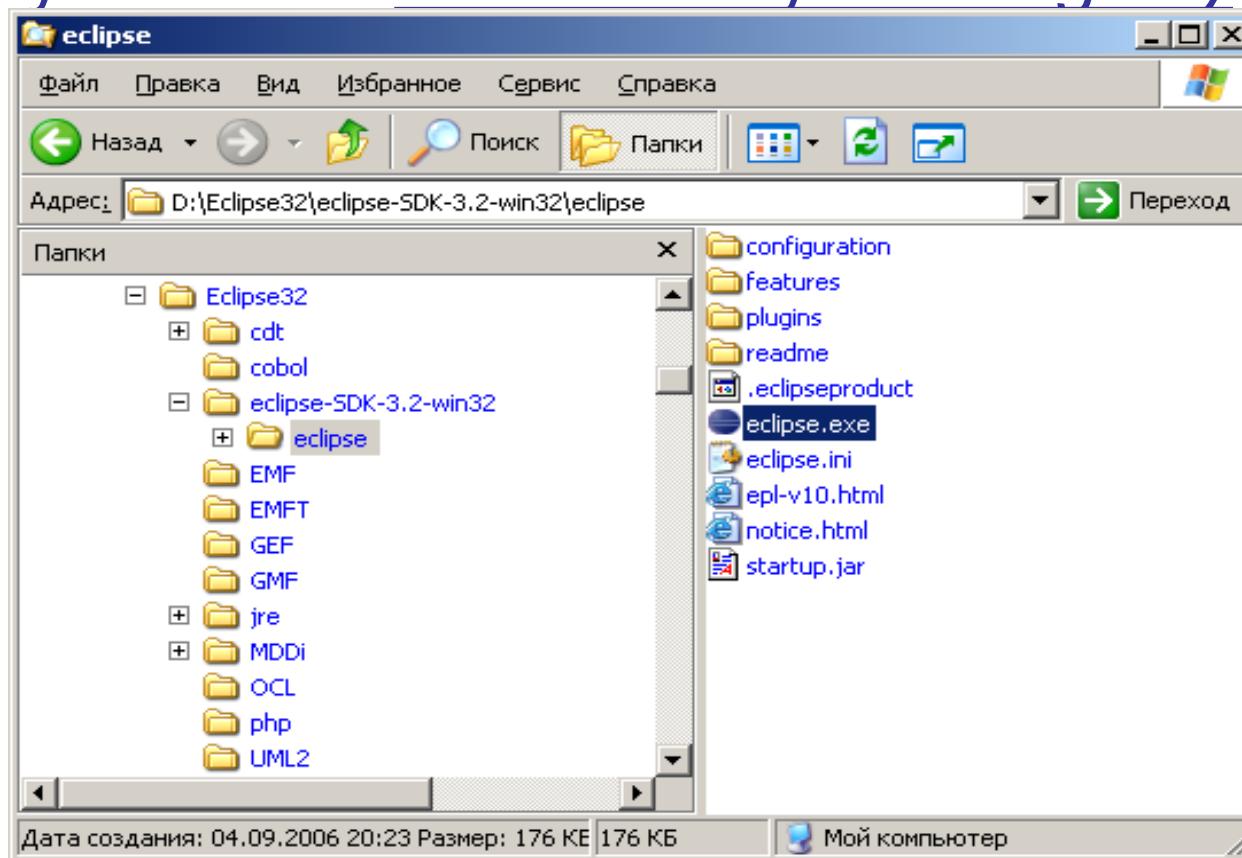
# Краткий обзор среды Eclipse для разработки на языке Java

# Запуск среды Eclipse на исполнение

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

Романов Владимир Юрьевич ©2025

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



# Обзор среды Eclipse

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

Романов Владимир Юрьевич ©2025

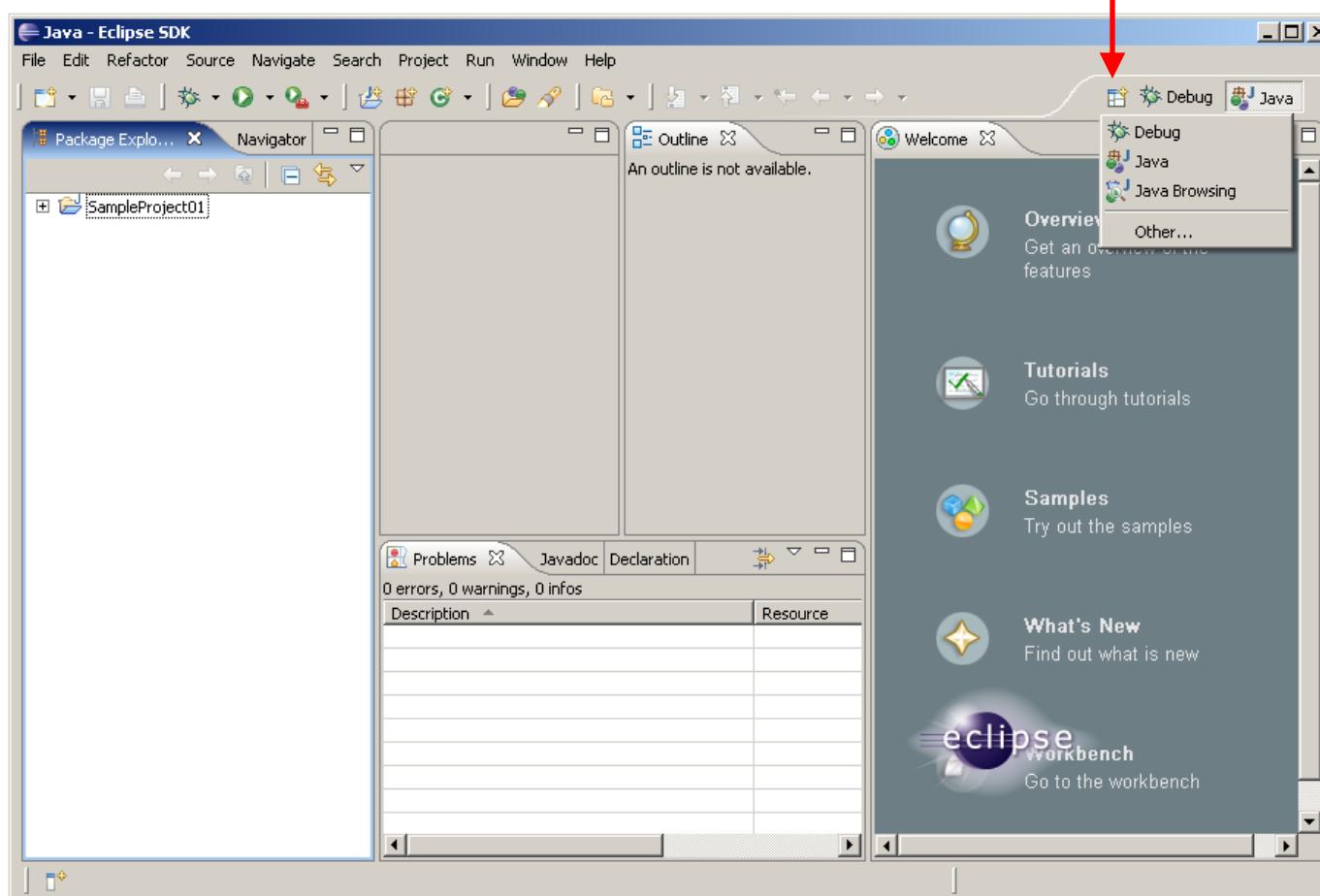
- Перспективы (Perspectives)
- Виды (Views)
- Редакторы (Editors)
- Работа с файлами
- Навигация по рабочему месту (Workbench)
- Консоль (Console)

# Обзор среды Eclipse. Перспективы (Java)

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

Романов Владимир Юрьевич ©2025

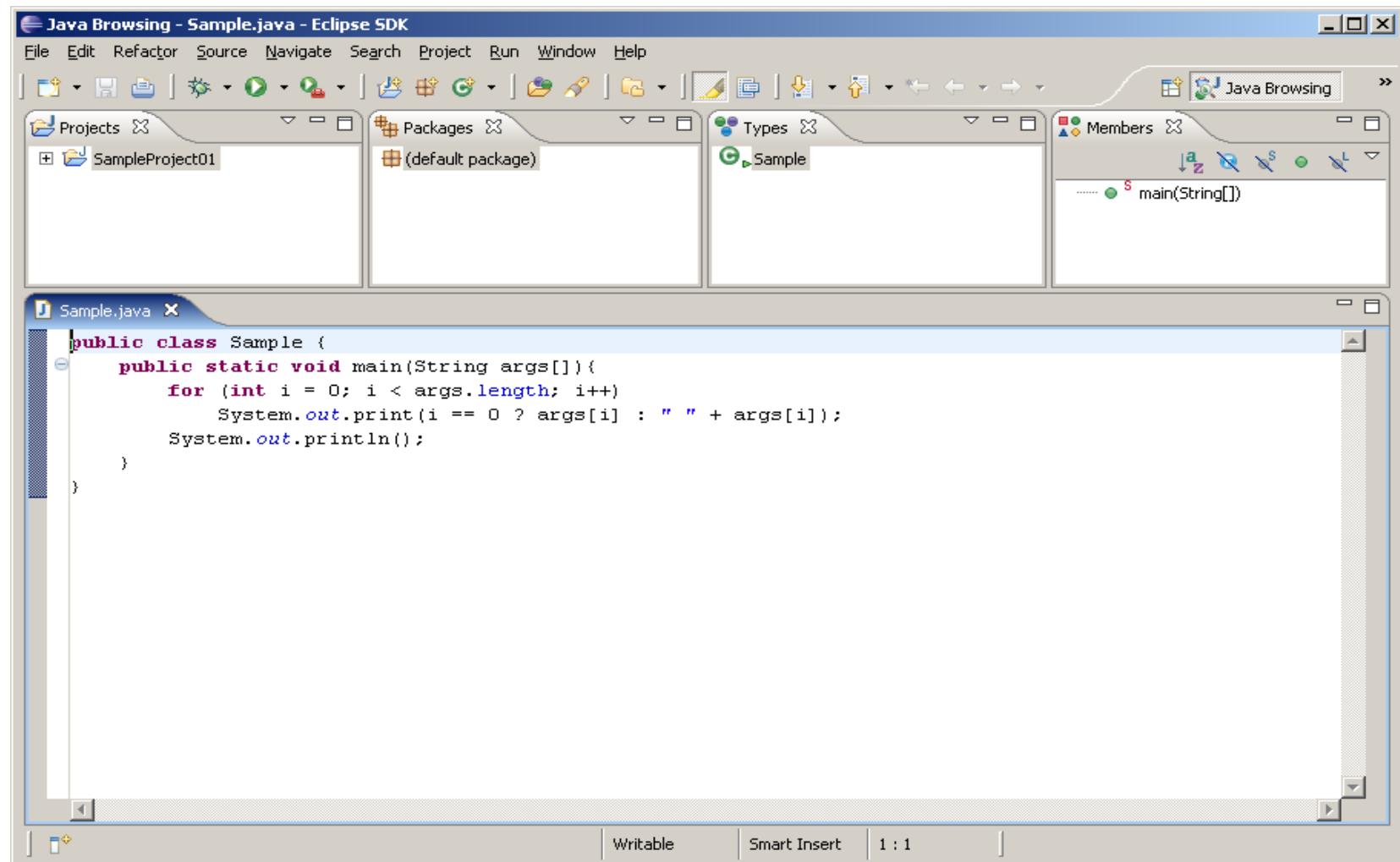
- На рабочем месте (workbench) может быть открыто несколько перспектив
- Различные перспективы могут быть выбраны из панели перспектив



# Обзор среды Eclipse. Перспективы (Java Browsing)

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

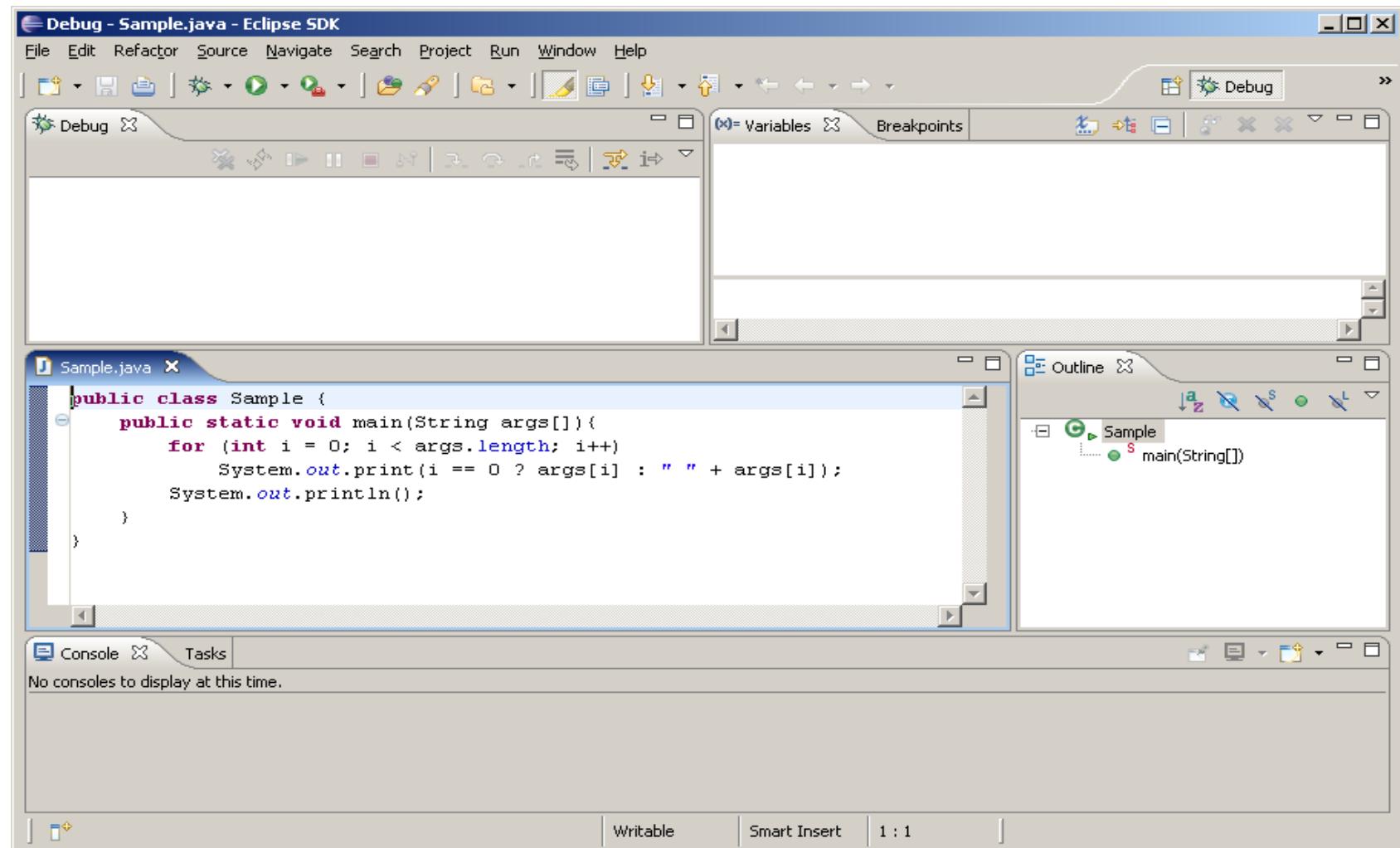
Романов Владимир Юрьевич ©2025



# Обзор среды Eclipse. Перспективы (Debug)

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

Романов Владимир Юрьевич ©2025

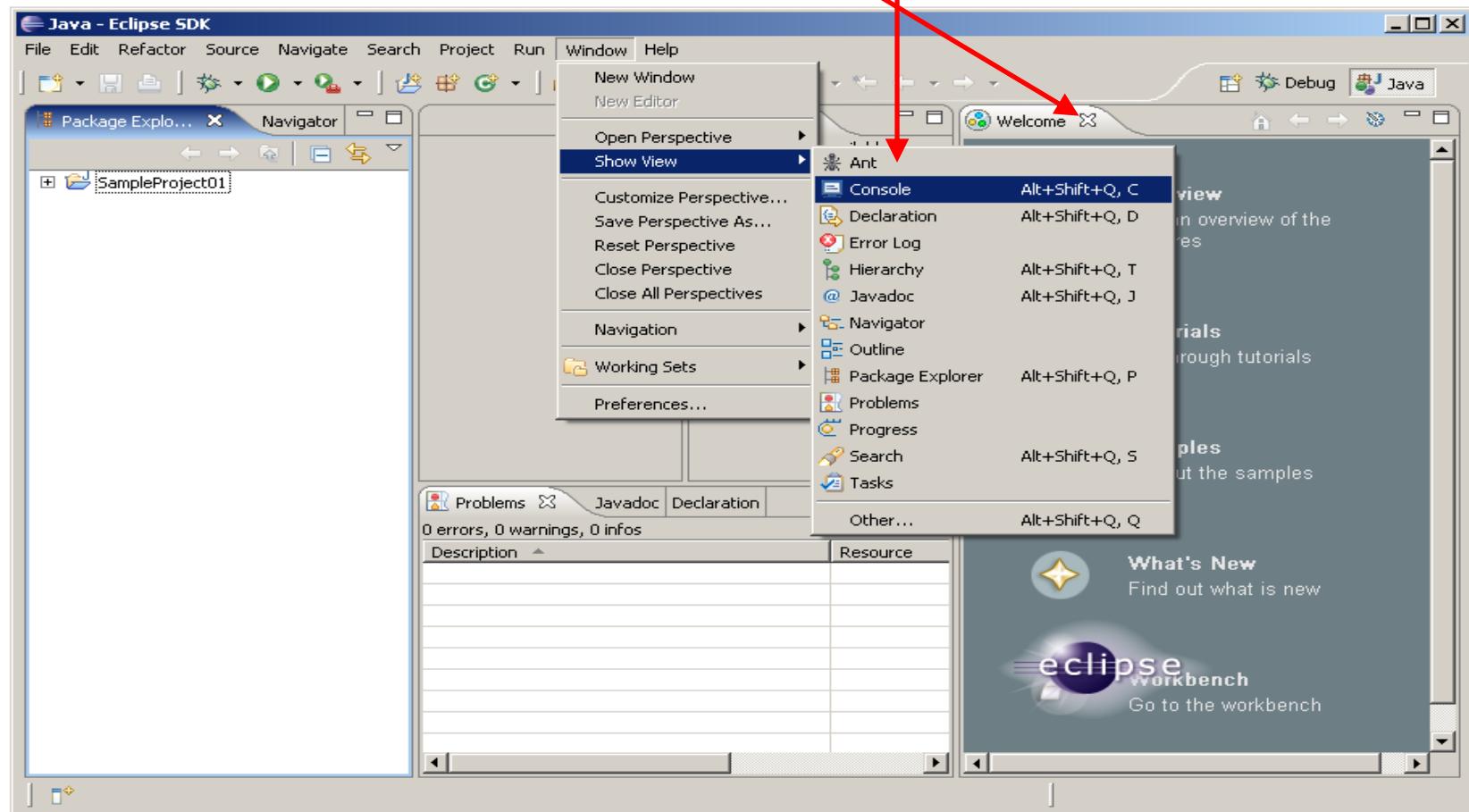


# Обзор среды Eclipse. Добавление видов к перспективам

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

Романов Владимир Юрьевич ©2025

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

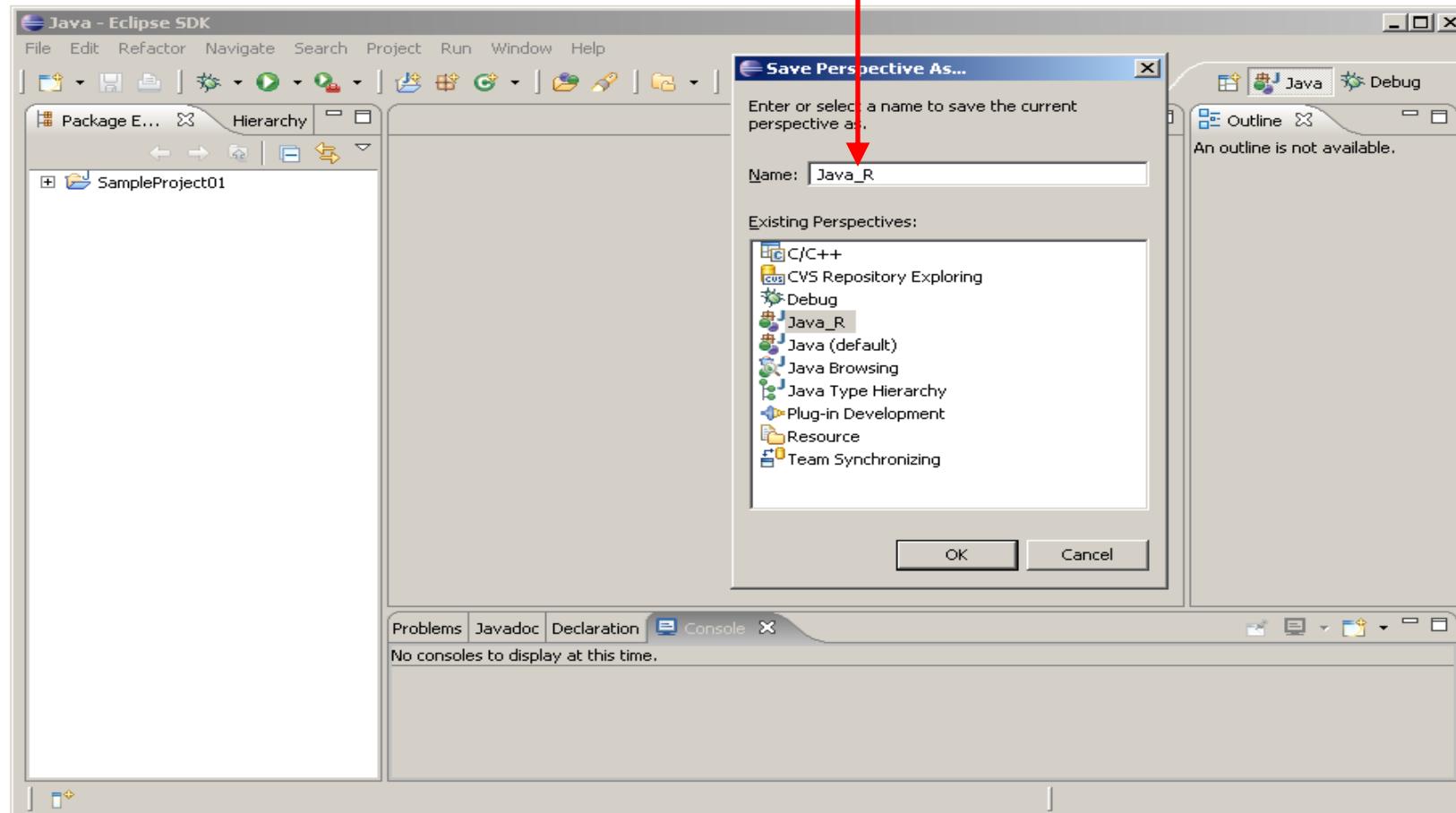


# Обзор среды Eclipse. Сохранение перспективы

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

Романов Владимир Юрьевич ©2025

- Сохранение перспективы под новым именем



# Обзор среды Eclipse.

## Виды

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

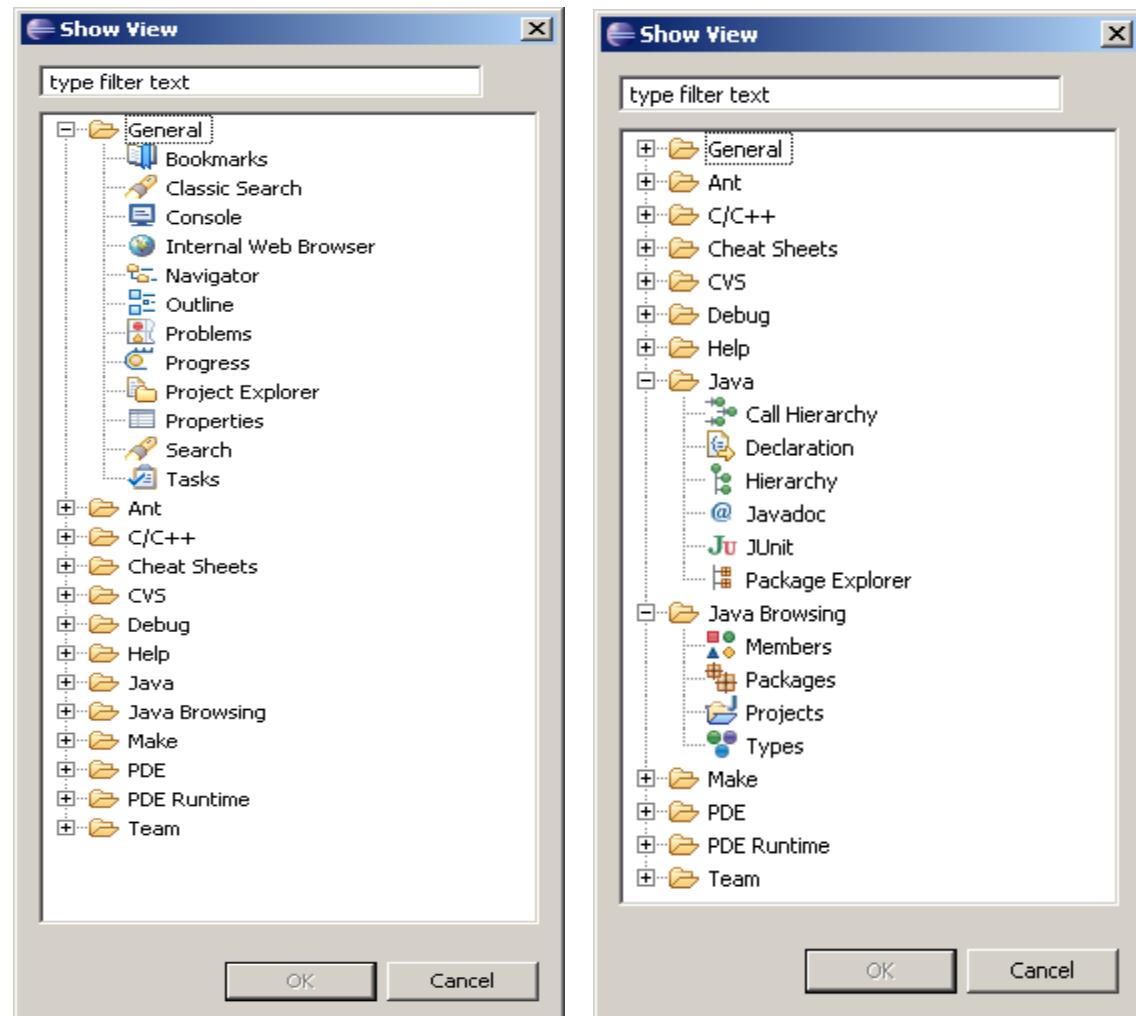
Романов Владимир Юрьевич ©2025

### Java Perspective

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

### Debug Perspective

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

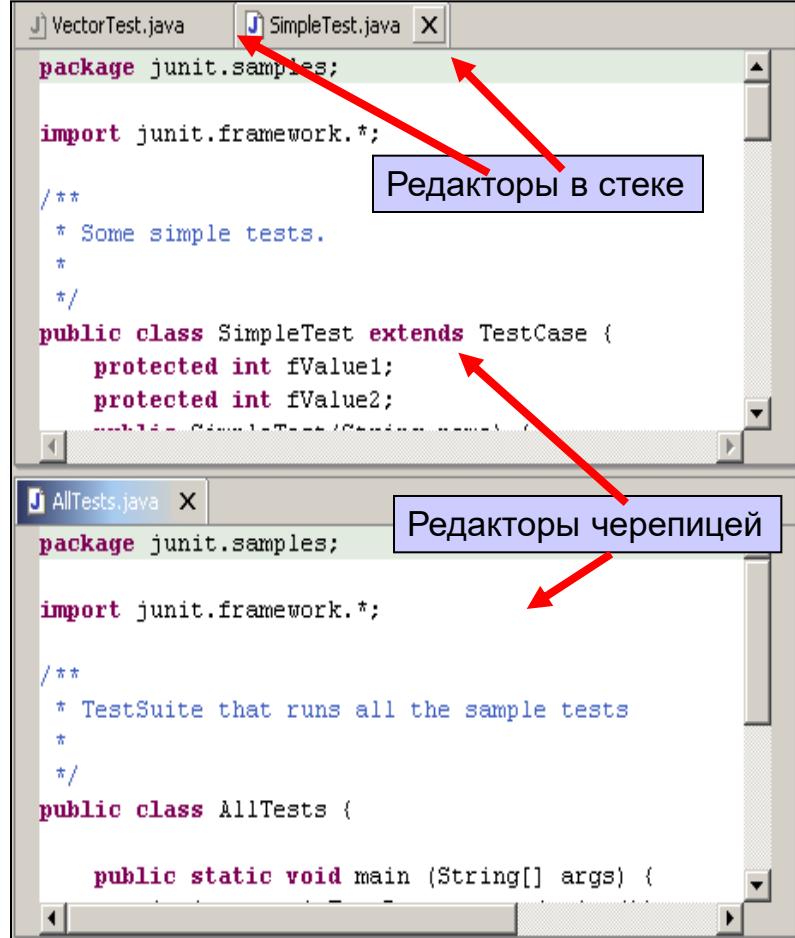


# Обзор среды Eclipse. Редакторы

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

Романов Владимир Юрьевич ©2025

- 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



SimpleTest.java

```
package junit.samples;

import junit.framework.*;

/**
 * Some simple tests.
 */
public class SimpleTest extends TestCase {
    protected int fValue1;
    protected int fValue2;
    ...
}
```

Редакторы в стеке

AllTests.java

```
package junit.samples;

import junit.framework.*;

/**
 * TestSuite that runs all the sample tests
 */
public class AllTests {

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

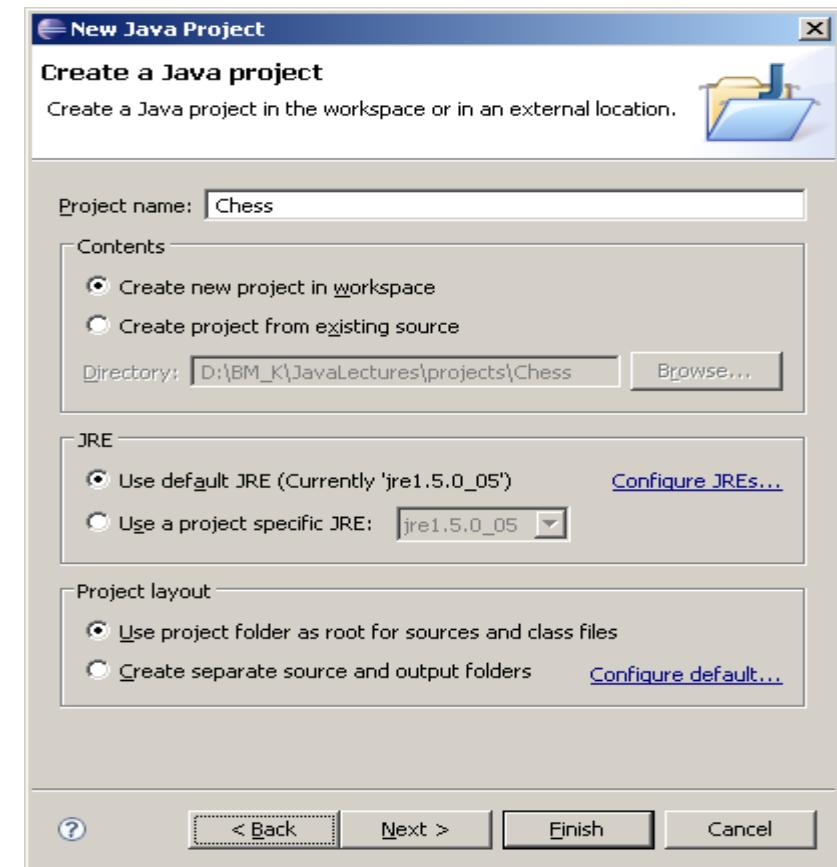
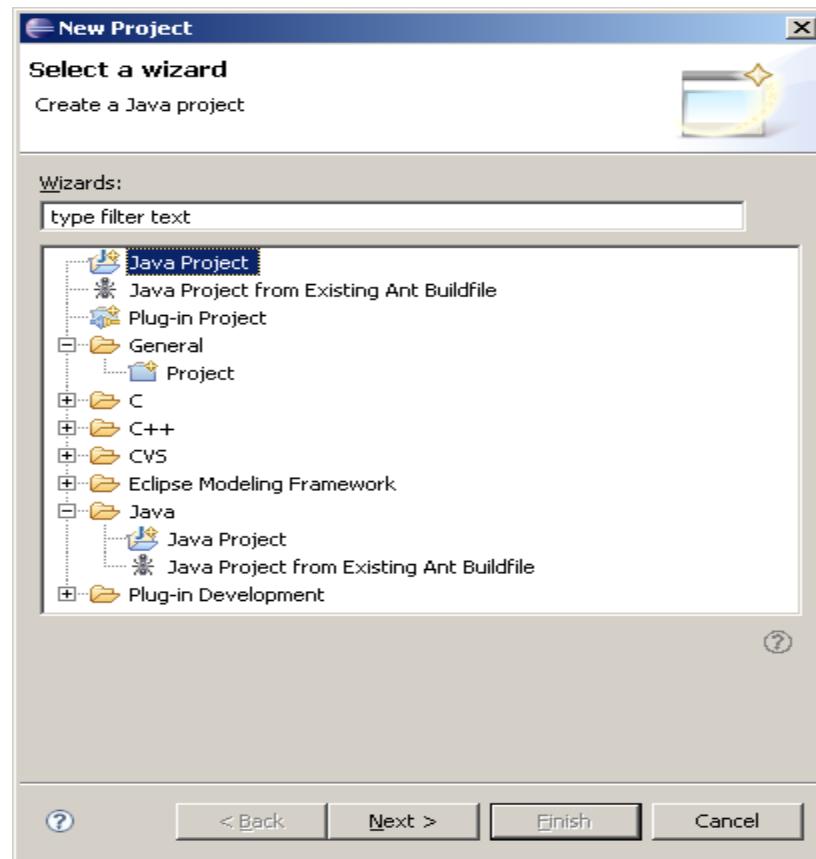
Редакторы черепицей

# Обзор среды Eclipse. Создание нового проекта

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

Романов Владимир Юрьевич ©2025

- File | New | Project

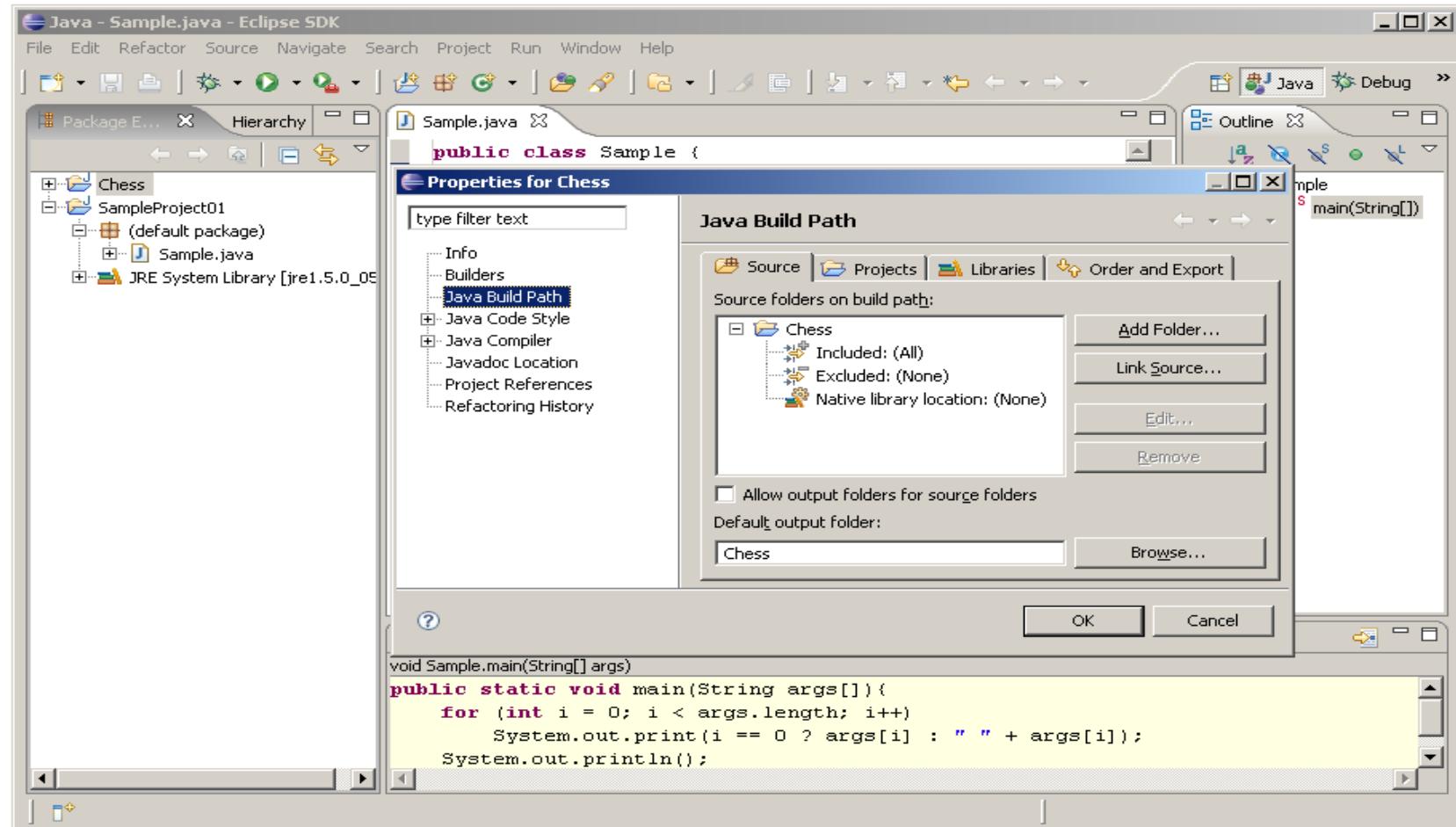


# Обзор среды Eclipse. Свойства

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

Романов Владимир Юрьевич ©2025

Все ресурсы имеют свойства.

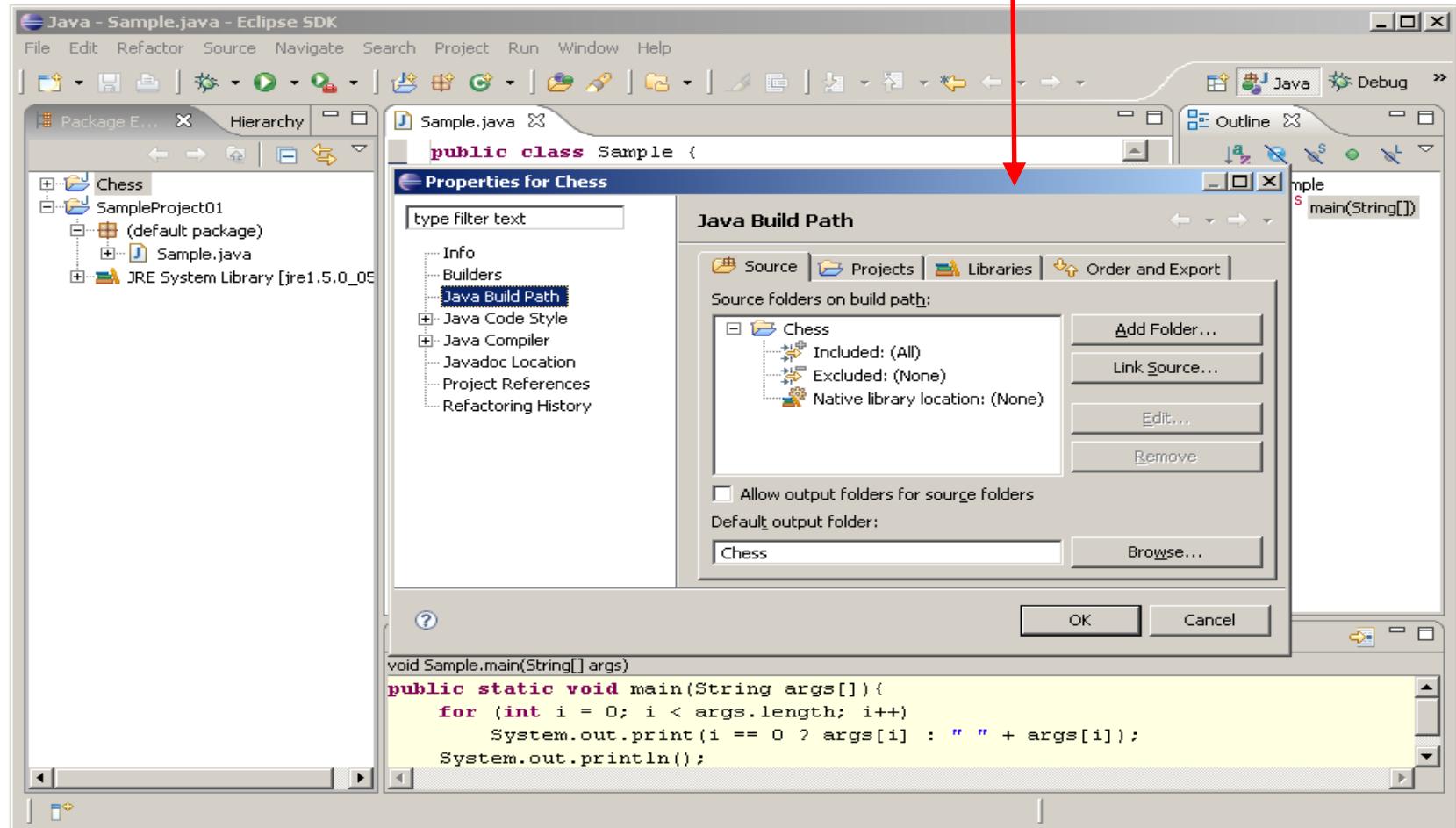


# Обзор среды Eclipse. Свойства

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

Романов Владимир Юрьевич ©2025

Все ресурсы имеют свойства. Свойства проекта *Chess*:

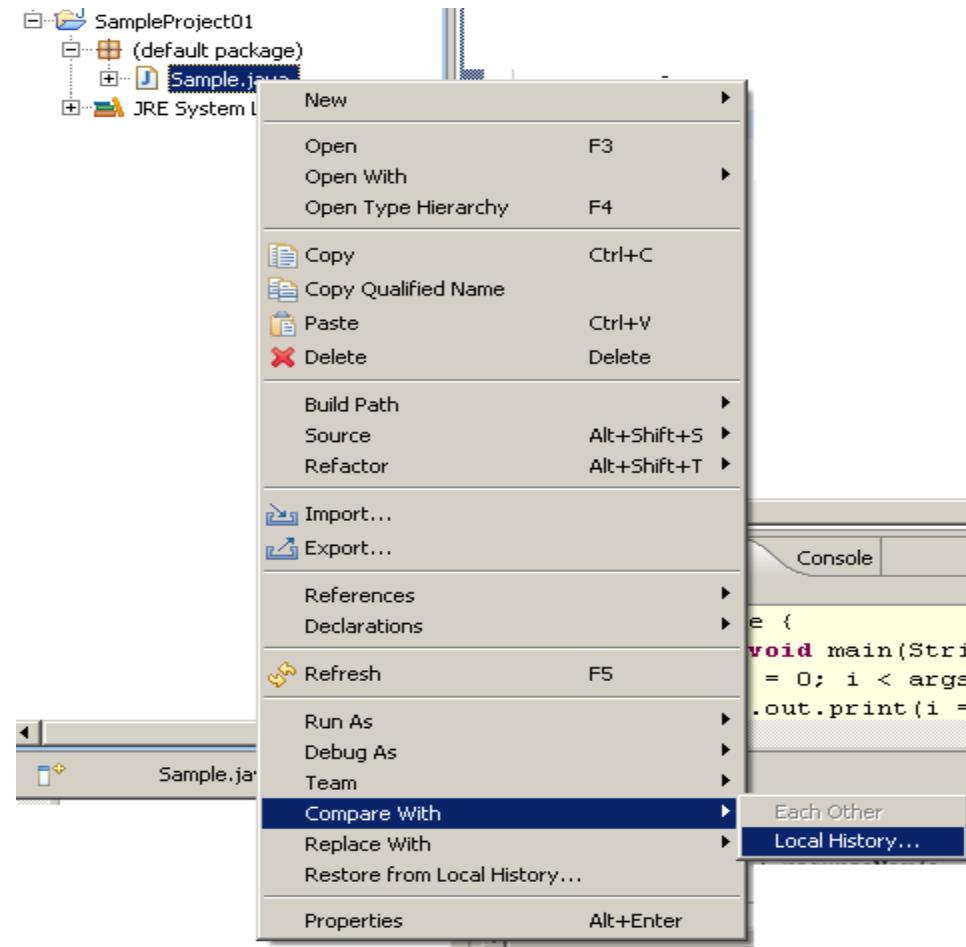


# Обзор среды Eclipse. Локальная история (1)

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

Романов Владимир Юрьевич ©2025

Для каждого файла хранится локальная история файла

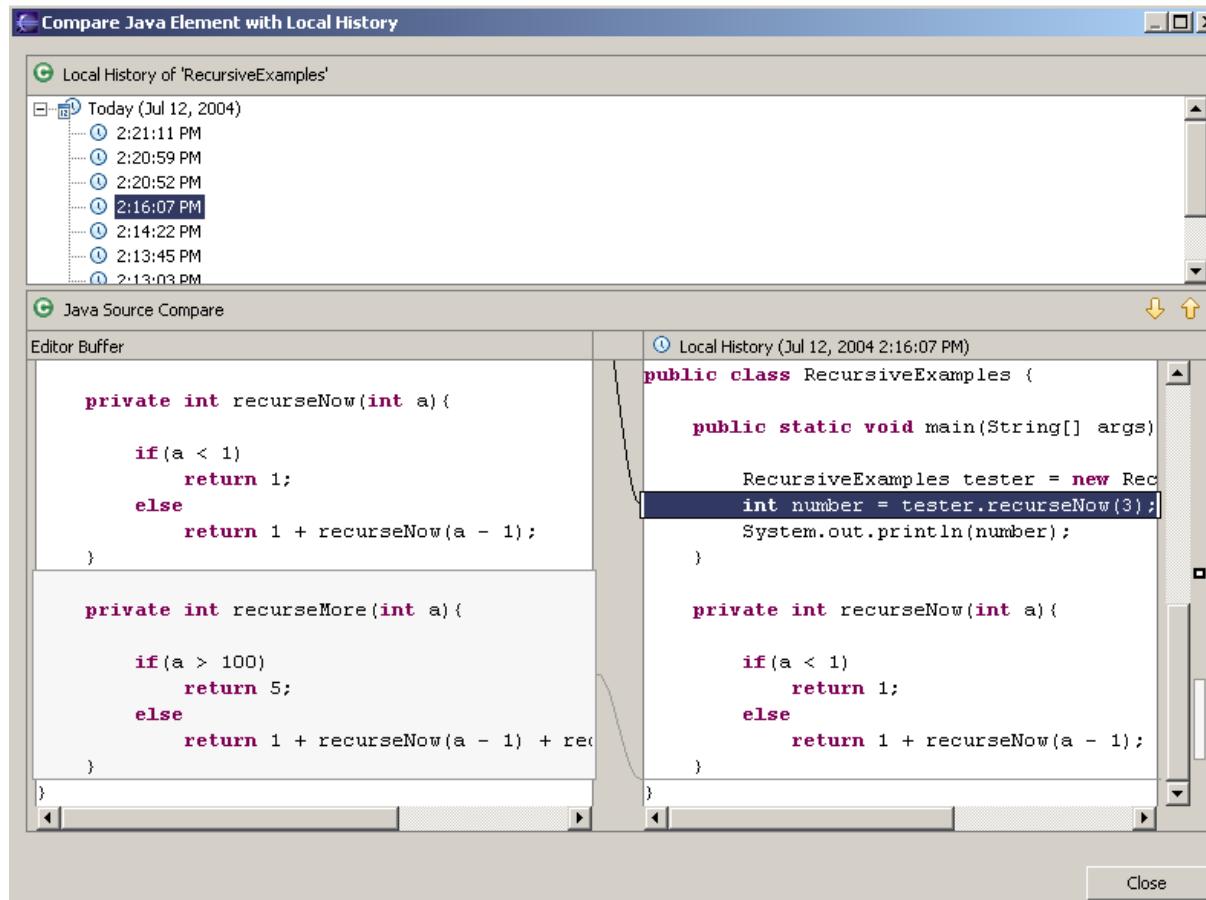


# Обзор среды Eclipse. Локальная история (2)

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

Романов Владимир Юрьевич ©2025

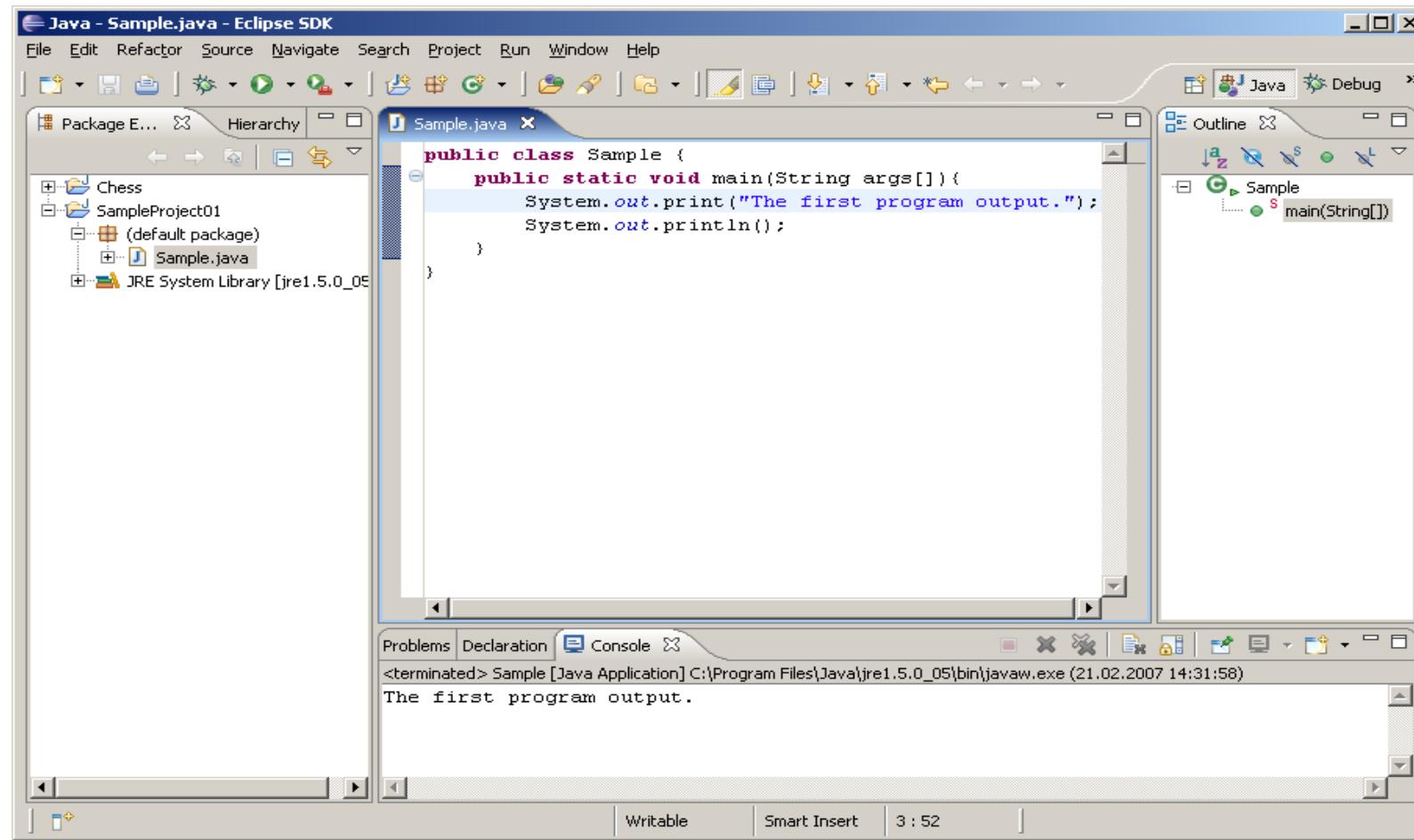
- Каждое использование команды *Save* сохраняет редакцию файла
- Редакции файлов можно сравнить



# Вывод программы на консоль

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

Романов Владимир Юрьевич ©2025



# Введение в язык Java

# Классы Java и их синтаксис

## Члены класса

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

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

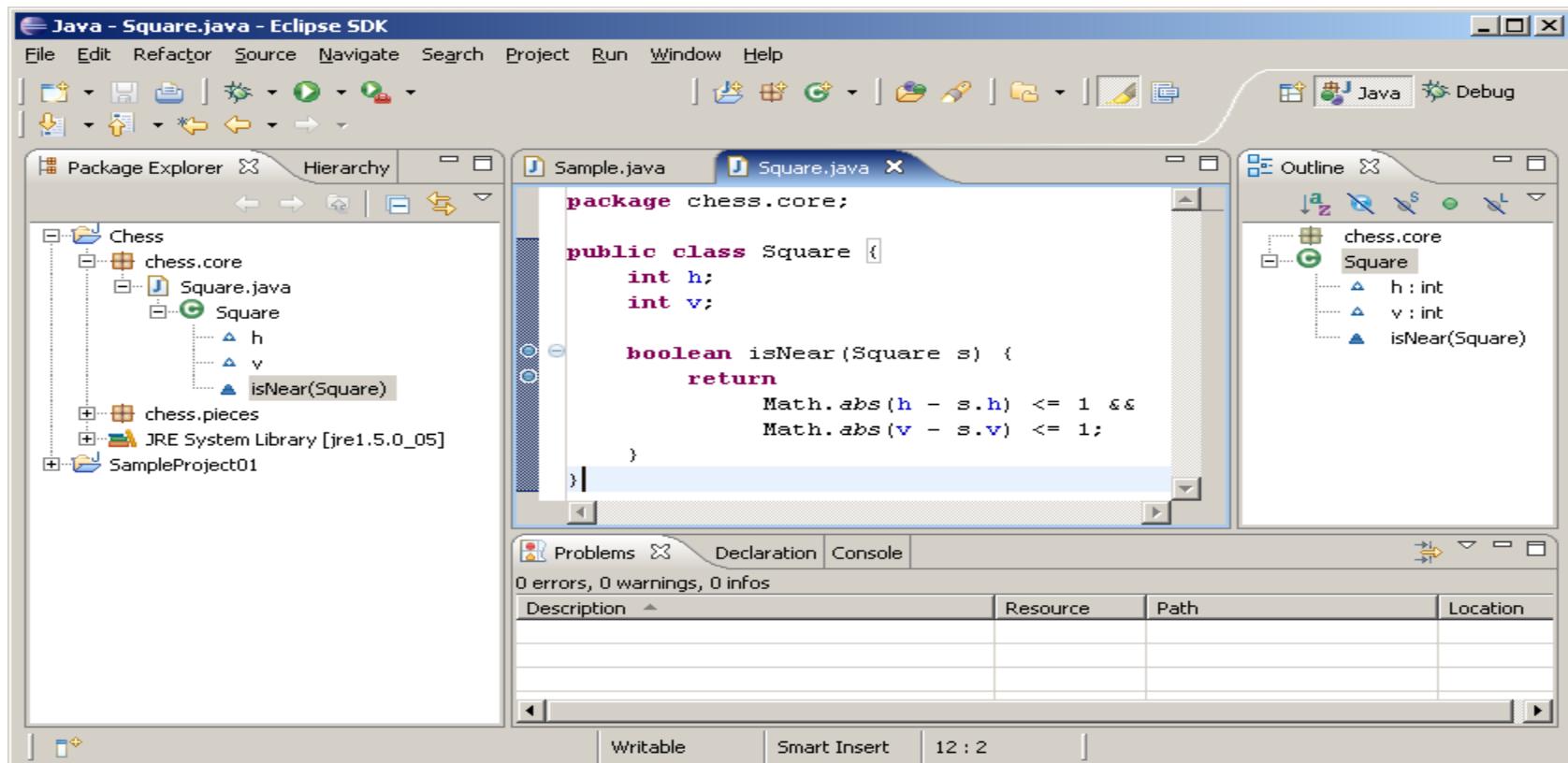
```
class Square {  
    int h;  
    int v;  
    boolean isNear(Square s) {  
        return  
            Math.abs(h - s.h) <= 1 &&  
            Math.abs(v - s.v) <= 1;  
    }  
}
```

# Классы Java и их синтаксис. Члены классов. Eclipse

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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; }  
}
```



# Классы Java и их синтаксис.

## Использование полей класса

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

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

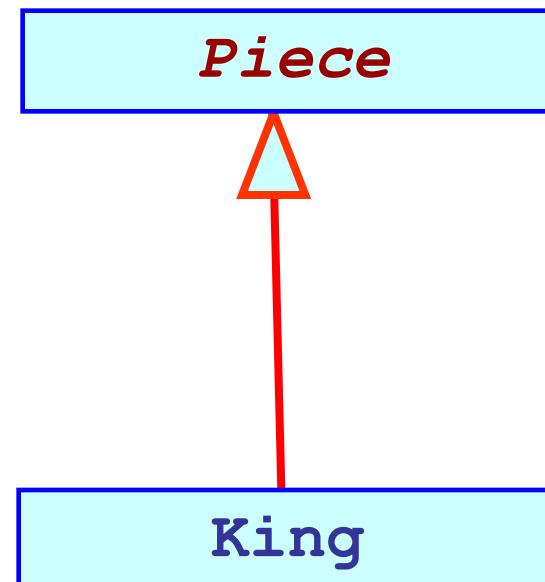
```
public class King extends Piece {  
    void move(Square s) {  
    }  
}
```

# Изображение наследования классов Java на UML-диаграммах

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

Романов Владимир Юрьевич ©2025

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

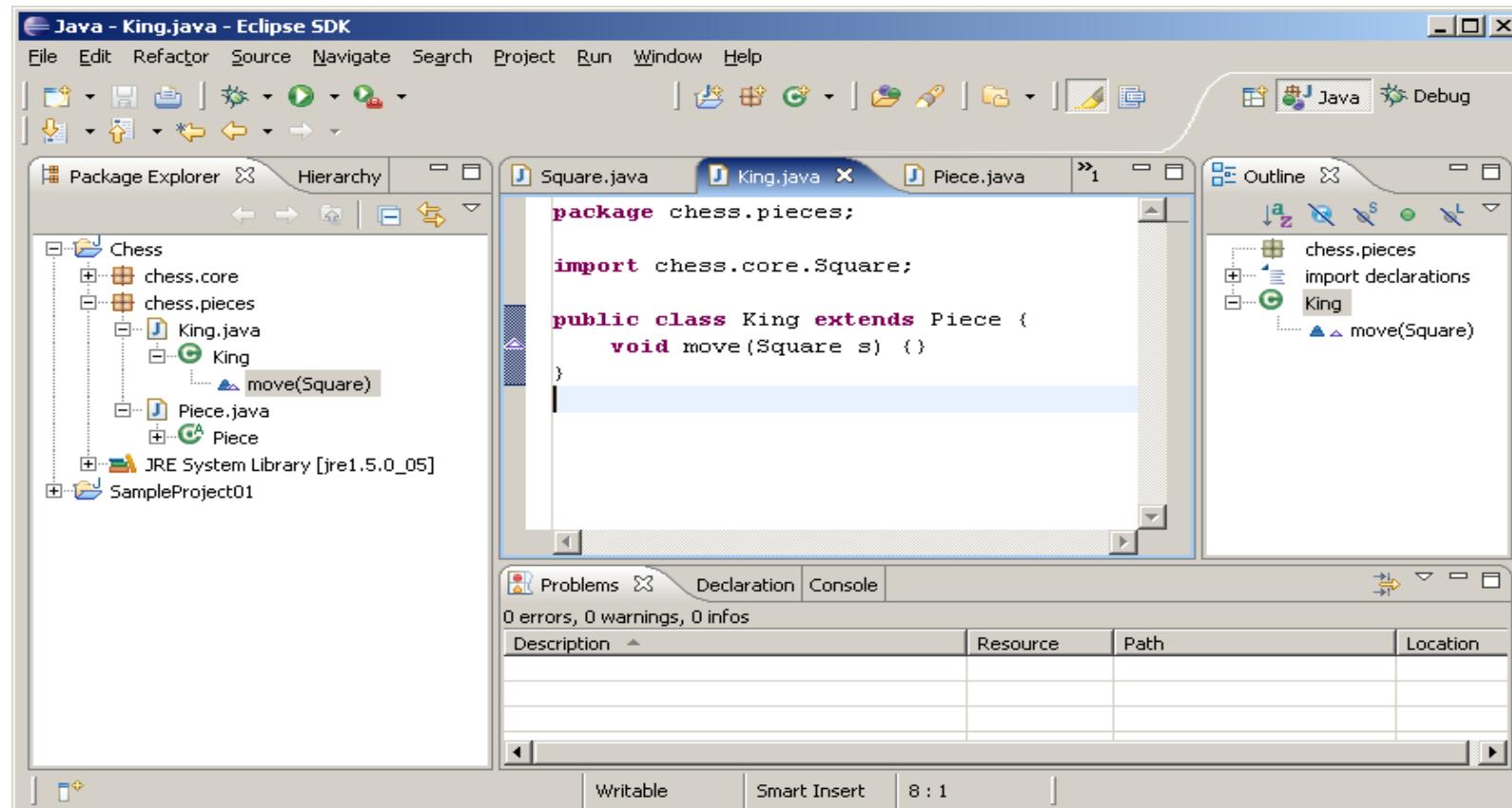


# Классы Java и их синтаксис. Наследование классов

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

```
// 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.
    }
}
```

Capture

Move



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

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

Романов Владимир Юрьевич ©2025

```
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 ©2025

```
// 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 ©2025

```
// 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 ©2025

```
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 ©2025

```
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 ©2025

```
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 ©2025

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

    Board getBoard(); // Ошибка!!!
}
```

Классы в уже отранслированных  
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 ©2025

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

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

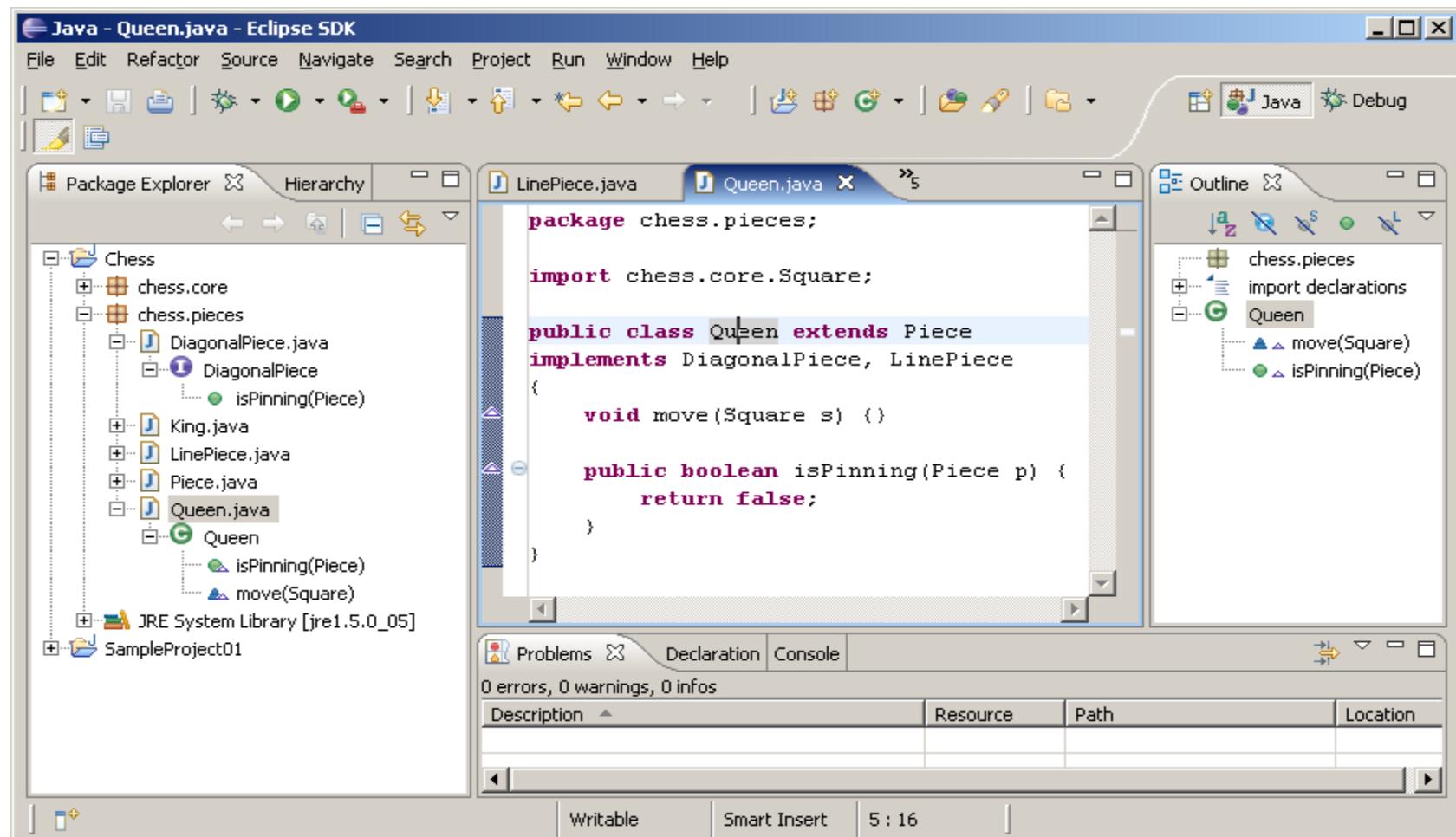
Для классов в уже  
отранслированных библиотеках  
используется умалчивающая  
реализация метода интерфейса

# Классы Java и их синтаксис. Реализация интерфейсов. Eclipse

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

Романов Владимир Юрьевич ©2025

- Классы могут реализовывать один и более *интерфейсов*



# Классы Java и их синтаксис.

## Группирование классов в пакеты

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

Романов Владимир Юрьевич ©2025

- 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;
    }
}
```

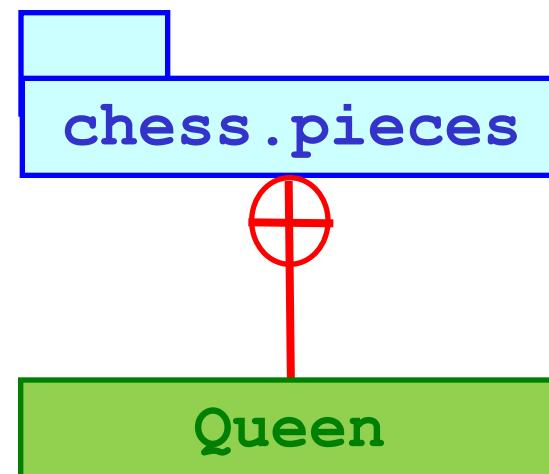
# Изображение пакетов на UML-диаграммах

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

Романов Владимир Юрьевич ©2025

```
package chess.pieces;

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



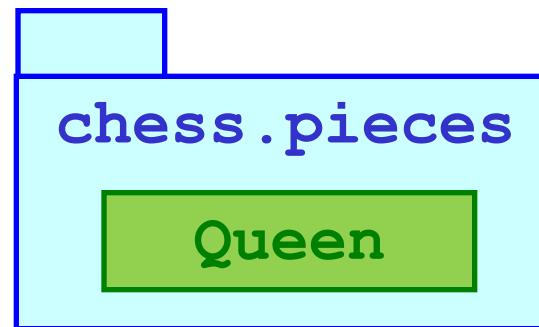
# Изображение пакетов на UML-диаграммах

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

Романов Владимир Юрьевич ©2025

```
package chess.pieces;

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

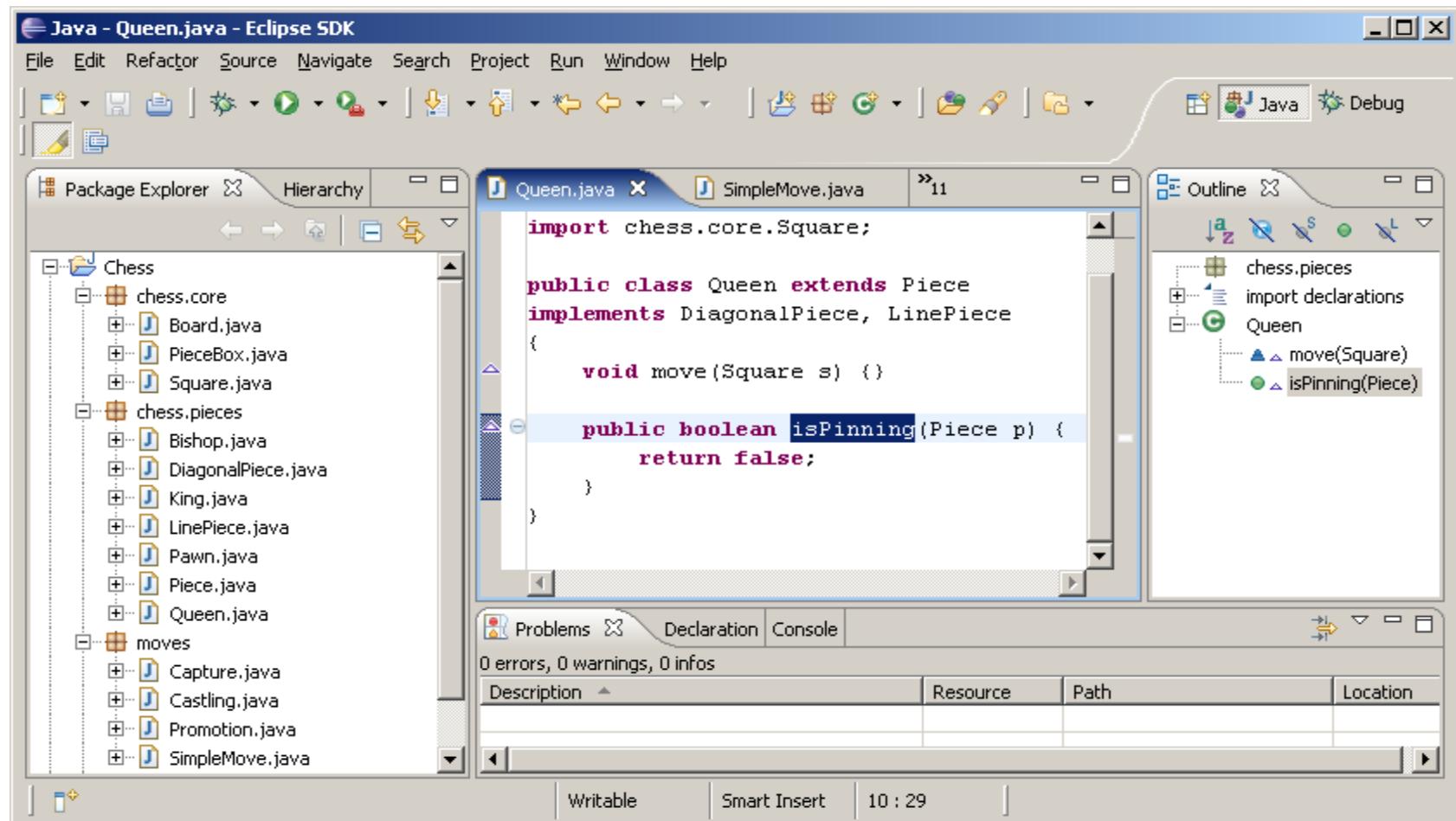


# Классы Java и их синтаксис. Группирование классов в пакеты

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

Романов Владимир Юрьевич ©2025

- Логически связанные классы группируются в пакеты



# Классы Java и их синтаксис.

## Модификаторы классов

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

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

*Piece*  
*{abstract}*

# Конструкторы классов

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

Романов Владимир Юрьевич ©2025

- 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();
```

# Умалчивающие конструкторы классов

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

Романов Владимир Юрьевич ©2025

- Умалчивающий 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 умалчивающий 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 ©2025

- *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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- *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 *умалчивающие 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 ©2025

- *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

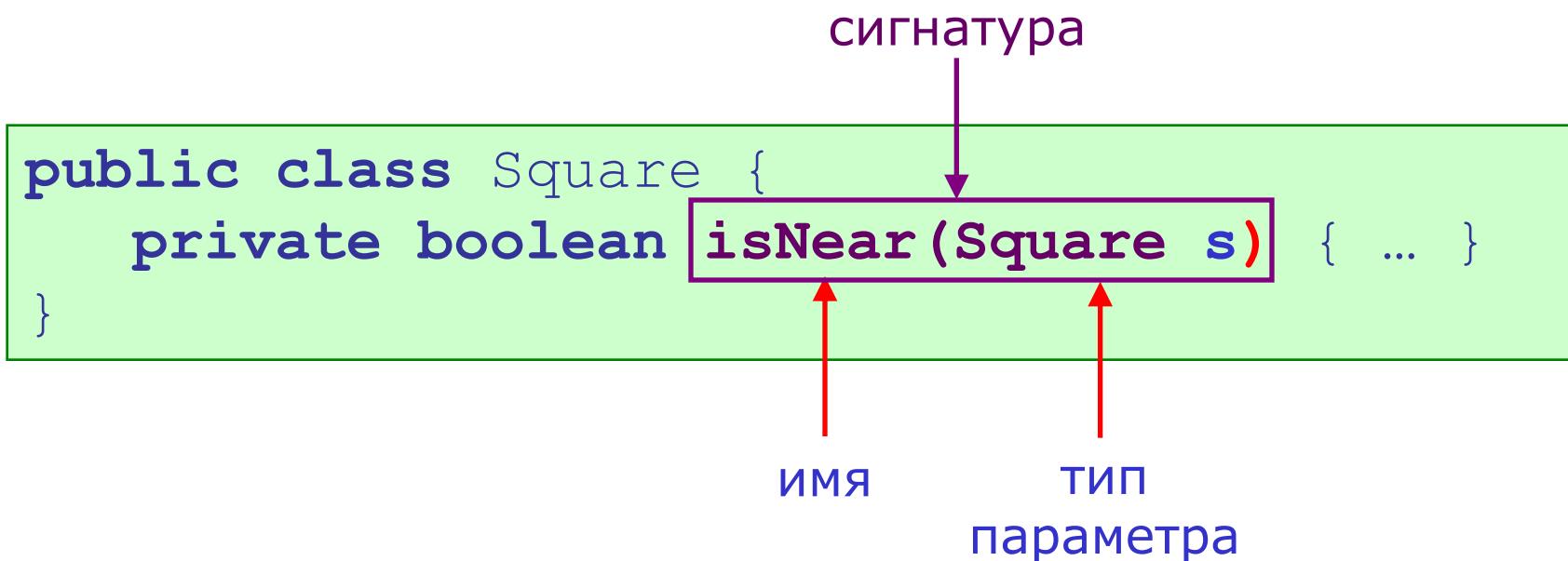
список  
параметров

# Сигнатура метода

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

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

## Вызов через оператор точки метода класса

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

Вызов через оператор точки  
метода экземпляра

метод определен в  
том же классе

# Перекрытие метода

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- 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 скрытия информации используется модификатор доступа *private*

# Статические члены

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- 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

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- Предложение *import* используется для того, чтобы 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 ©2025

- Предложение *import* используется для того, чтобы 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- **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 ©2025

- **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 ©2025

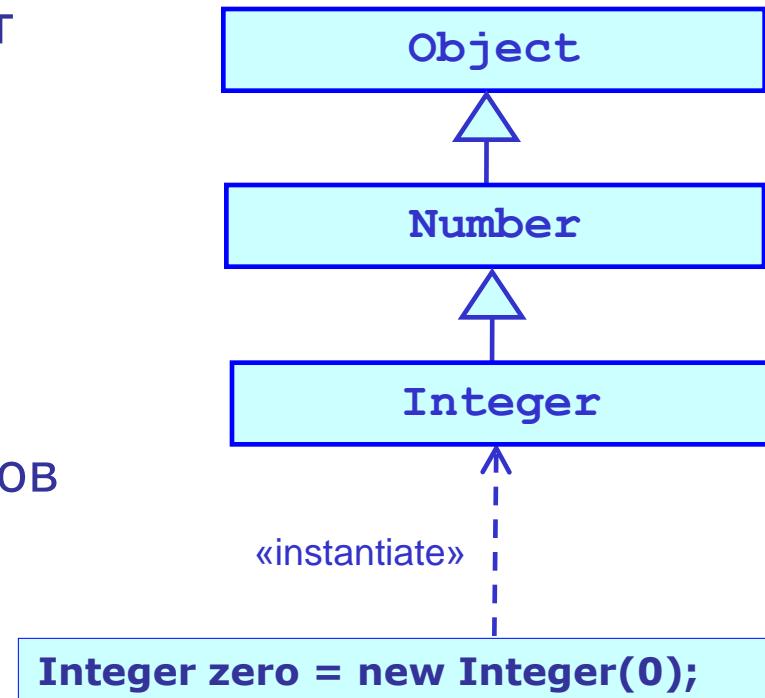
- Понимание наследования полей и методов
- Знакомство с иерархией классов
- Как подклассы специализируют классы
- Как выполняется поиск метода
- Как создаются и используются подклассы
- Понимание полиморфизма
- Рефакторинг в иерархии наследования

# Иерархии классов

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

Романов Владимир Юрьевич ©2025

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

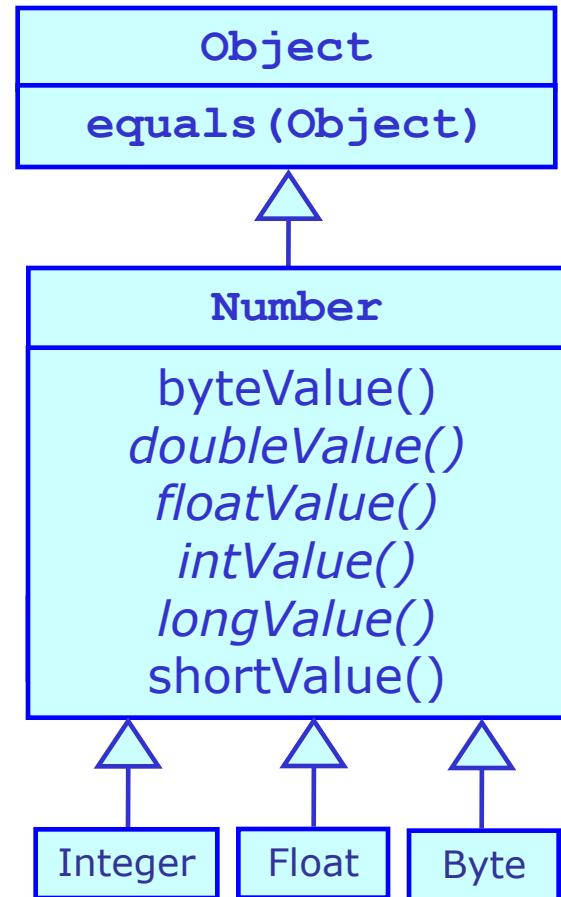


# Специализация и обобщение

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

Романов Владимир Юрьевич ©2025

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

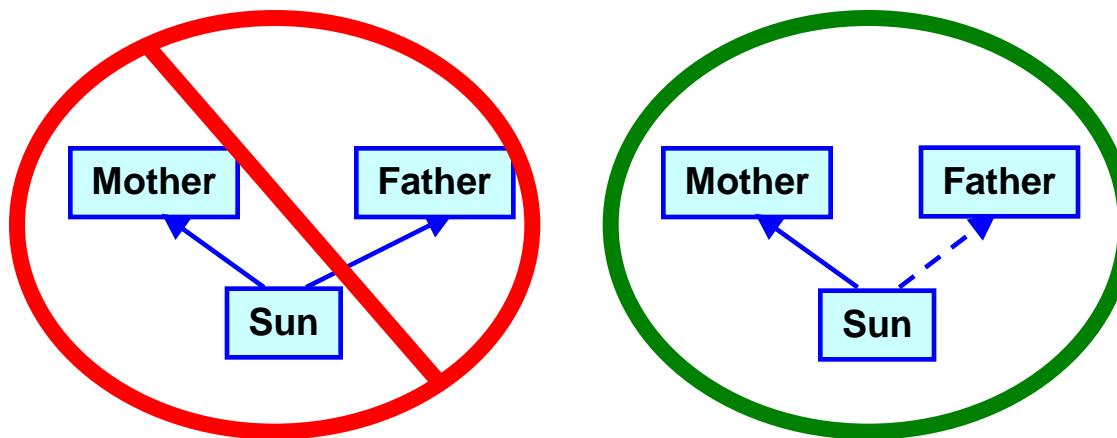


# Множественное наследование

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

Романов Владимир Юрьевич ©2025

- 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



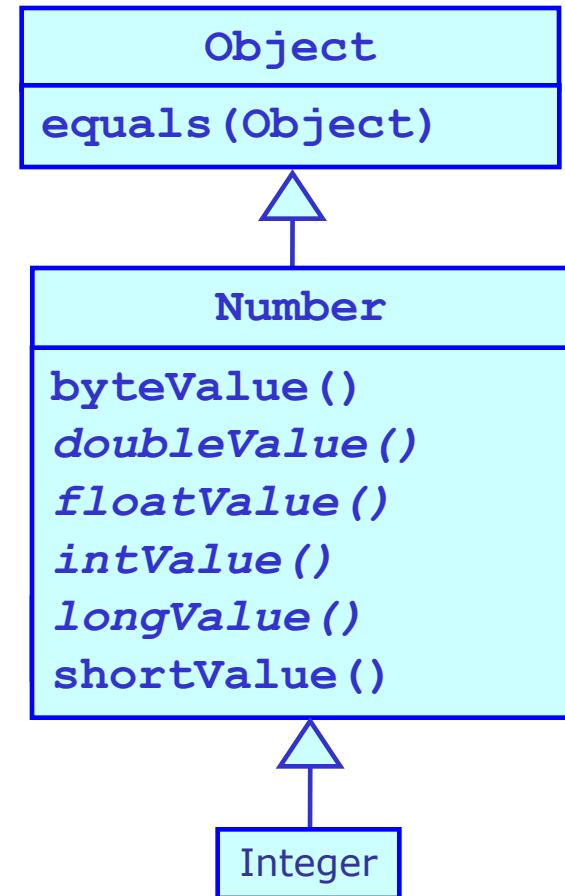
# Наследование полей и методов

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- Имеется возможность дополнить или изменить поведение суперкласса перекрытием в подклассе унаследованного метода
- Перекрывающие 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 ©2025

- 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 ©2025

- 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));  
}
```



# Наследование и конструкторы

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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

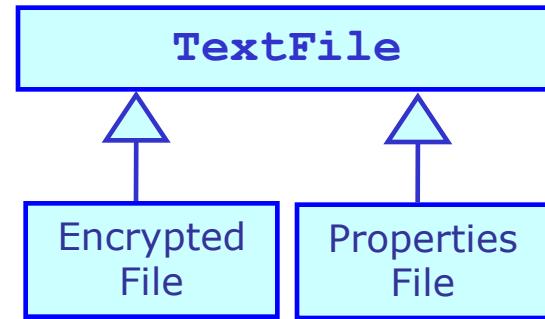
# Полиморфизм

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

```
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 ©2025

- 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 ©2025

```
package graph;

public class Node {
    static Color defaultLineColor, defaultColorFill;

    static {
        defaultLineColor = Color.Blue;
    }

    public Node() { }

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

Блок инициализации статических полей

Порядок расположения блоков инициализации не важен

# Вложенность классов

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

Романов Владимир Юрьевич ©2025

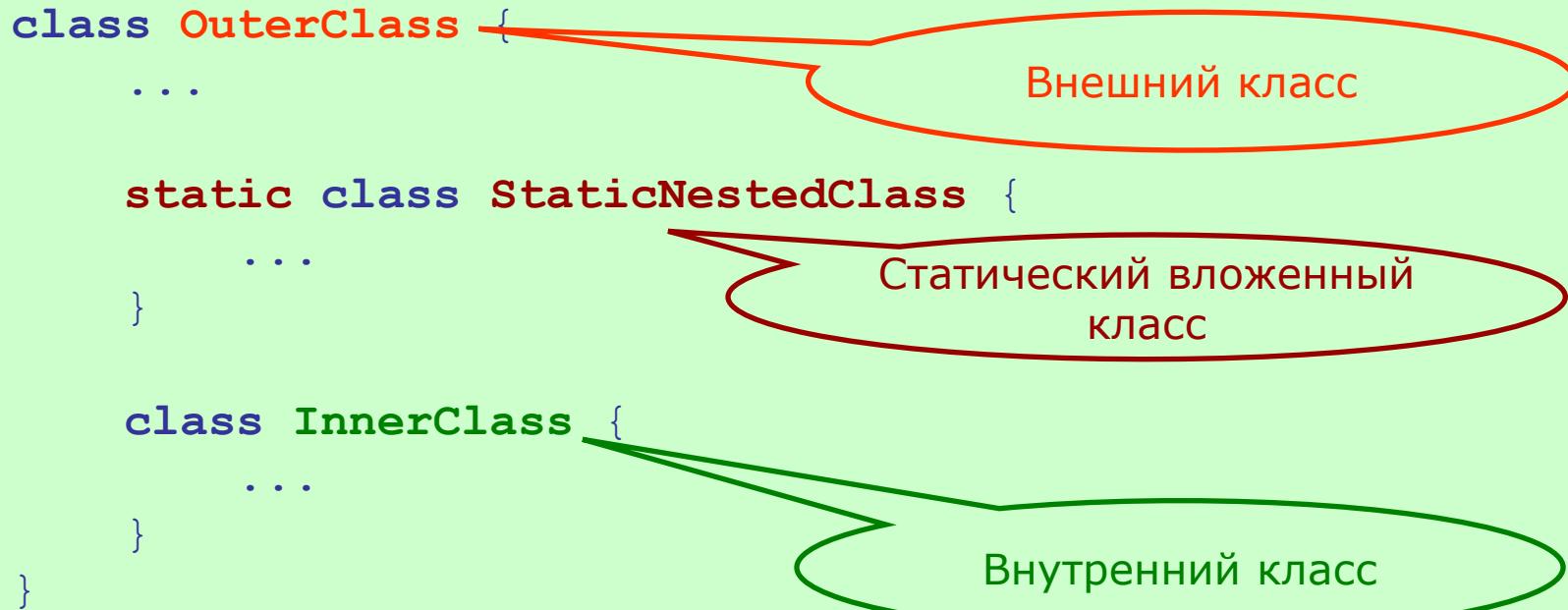
- Способ группирования классов используемых только в одном месте. Вспомогательные классы внутри
- Увеличение инкапсуляции – вложенный класс работает со скрытыми (*private*) полями и методами *внешнего* (*outer*) класса.
- Получаемый код проще понимать – вложенные классы обычно небольшие
- Получаемый 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 ©2025

```
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 ©2025

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



# Внутренние классы

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

Романов Владимир Юрьевич ©2025

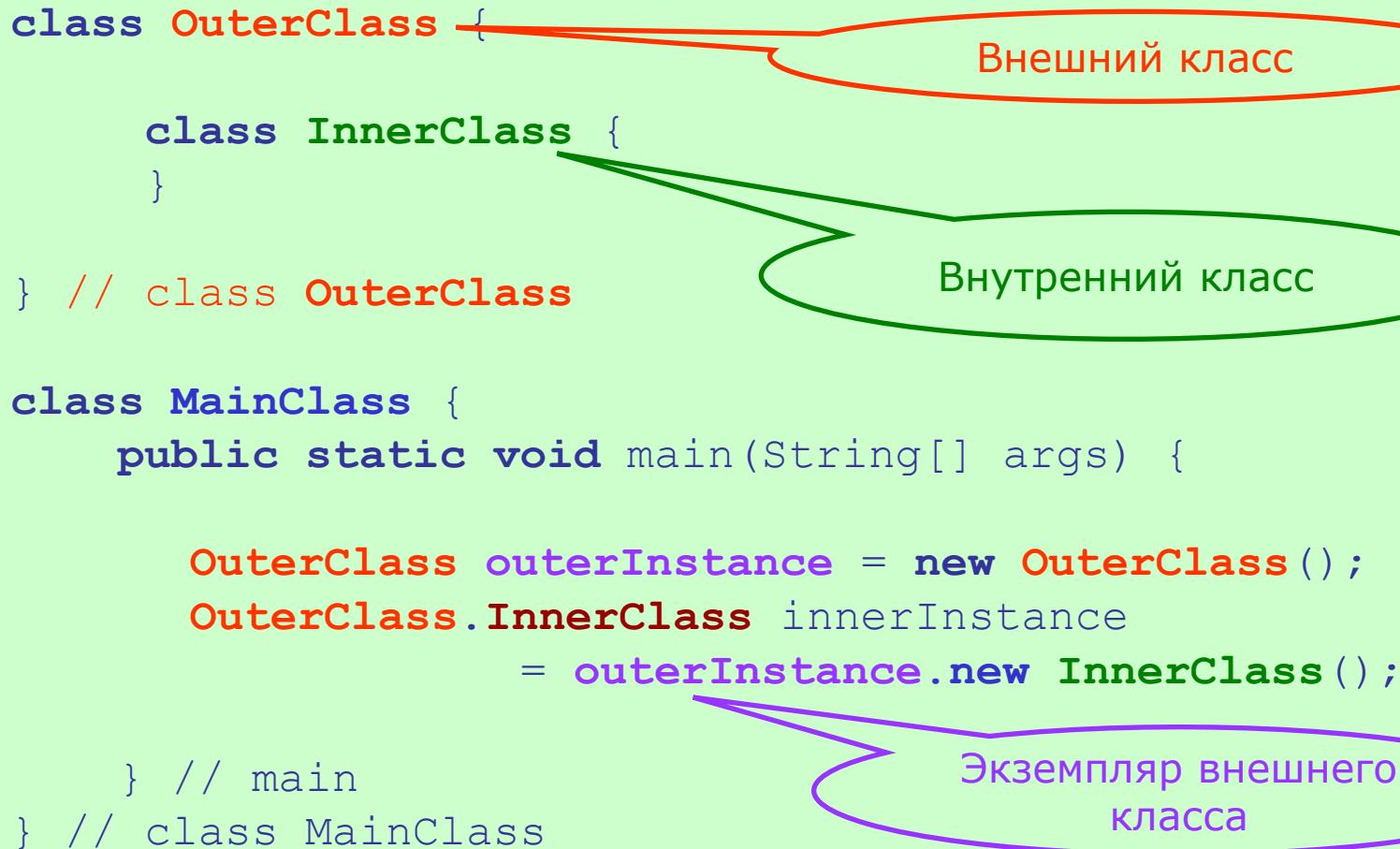
- Внутренние классы могут существовать только внутри экземпляров внешнего класса
- Внутренние классы имеют прямой доступ к полям и методам экземпляра класса
- Перед созданием экземпляра внутреннего класса необходимо создать экземпляр внешнего класса
- Для внутренних классов можно задавать видимости: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

# Доступ к внутренним классам

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

Романов Владимир Юрьевич ©2025

```
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
```



# Внутренние локальные классы

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

Романов Владимир Юрьевич ©2025

- *Внутренние локальные классы* могут объявляются 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 ©2025

- *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

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

Романов Владимир Юрьевич ©2025

- *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 ©2025

```
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 ©2025

```
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;  
    }  
}
```

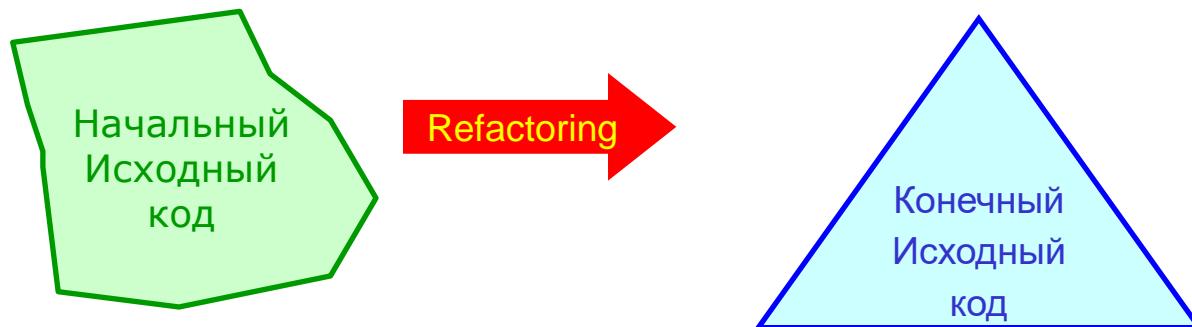
# Рефакторинг программ на языке Java

# Что такое рефакторинг

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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.

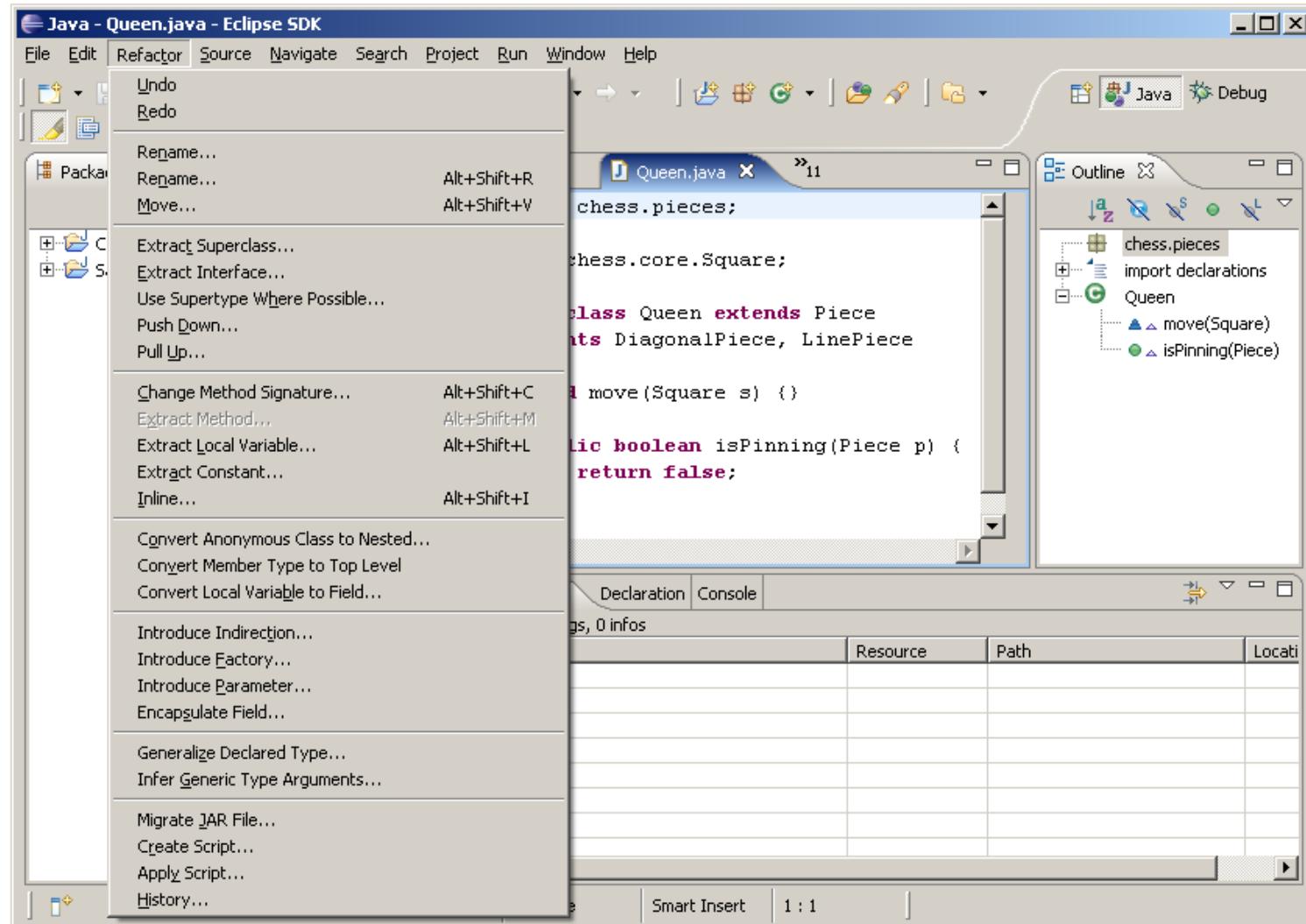
Романов Владимир Юрьевич ©2025

- 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 ©2025



# Рефакторинг в Eclipse.

## Переименование типа (1)

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

Романов Владимир Юрьевич ©2025

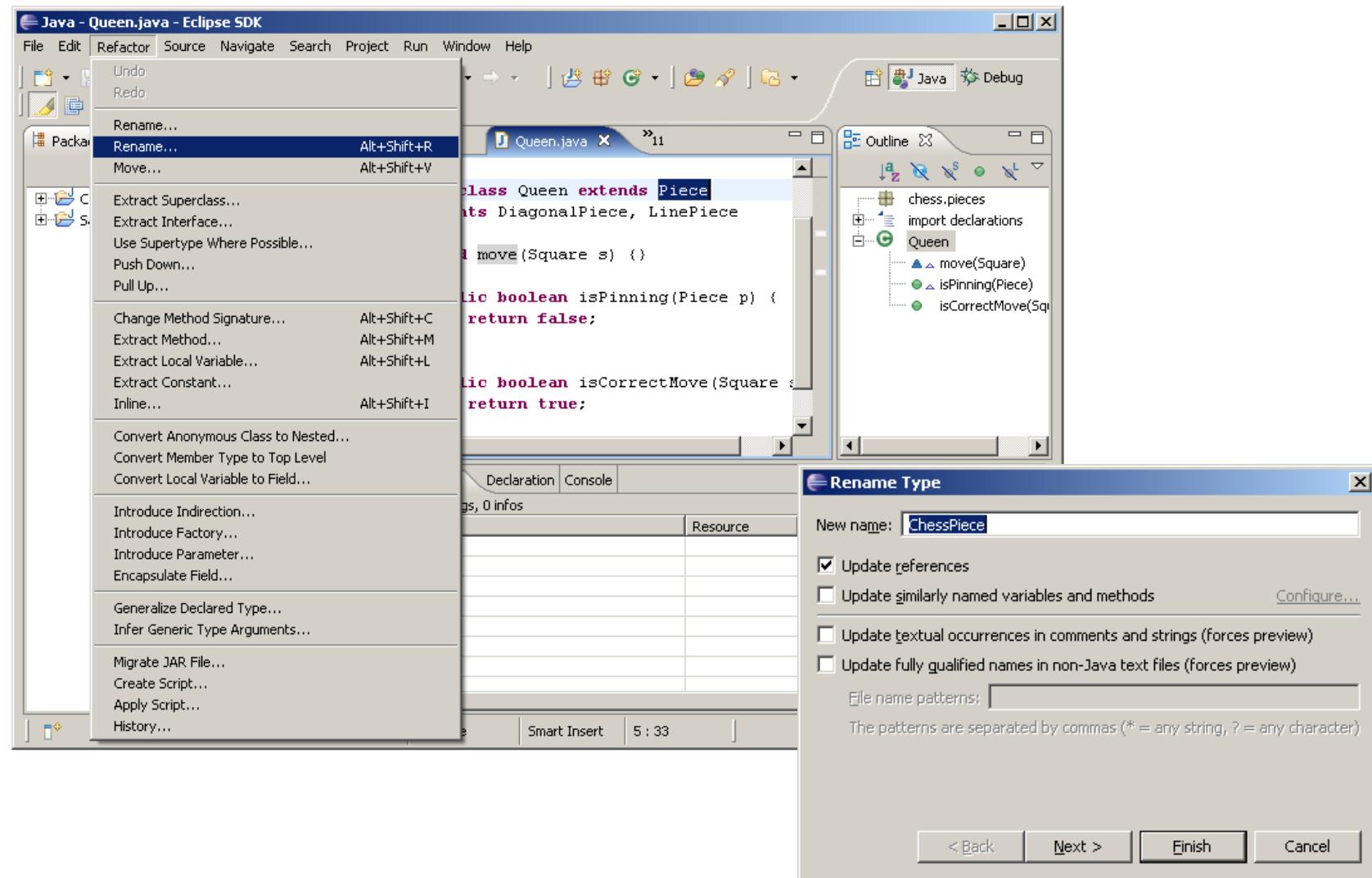
Среда Eclipse существенно упрощает  
проведение рефакторинга

- 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.

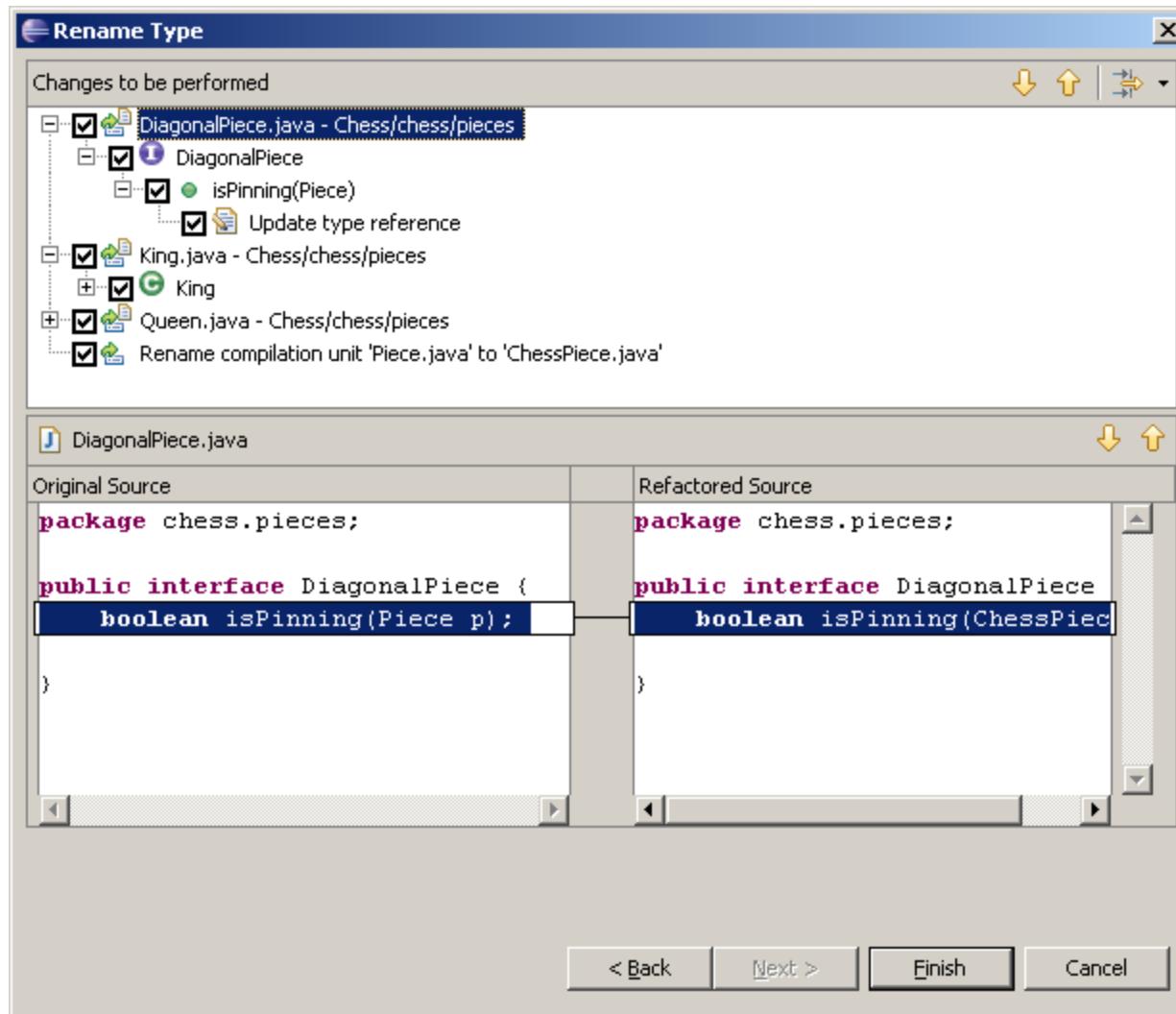
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Переименование типа (3)

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

Романов Владимир Юрьевич ©2025

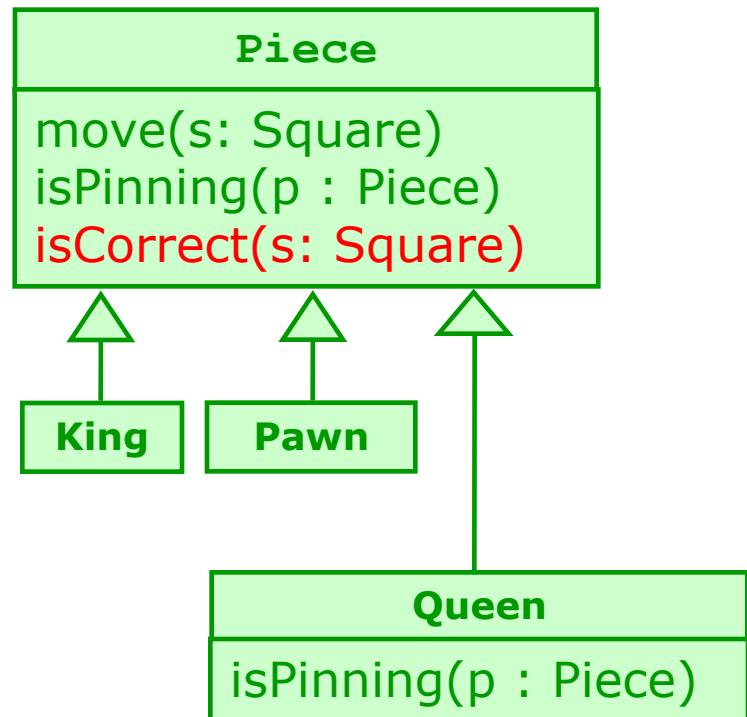
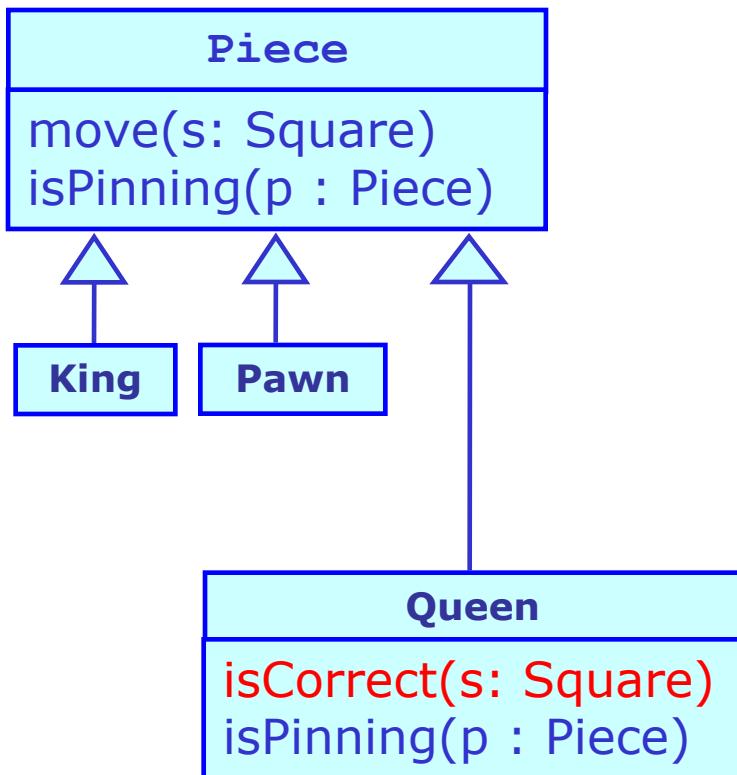


# Рефакторинг в Eclipse.

## Перемещение метода (1)

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

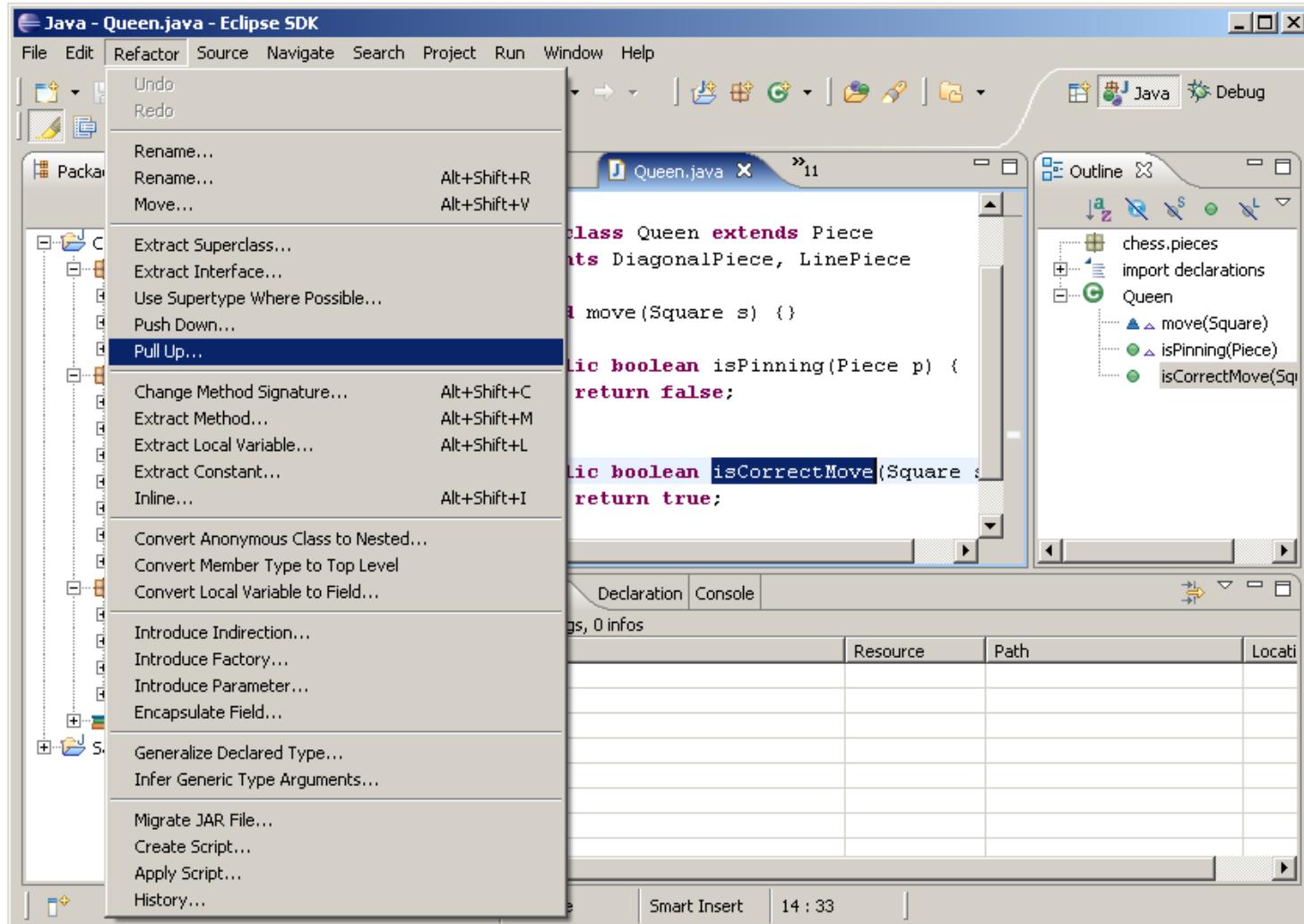
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Перемещение метода (2)

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

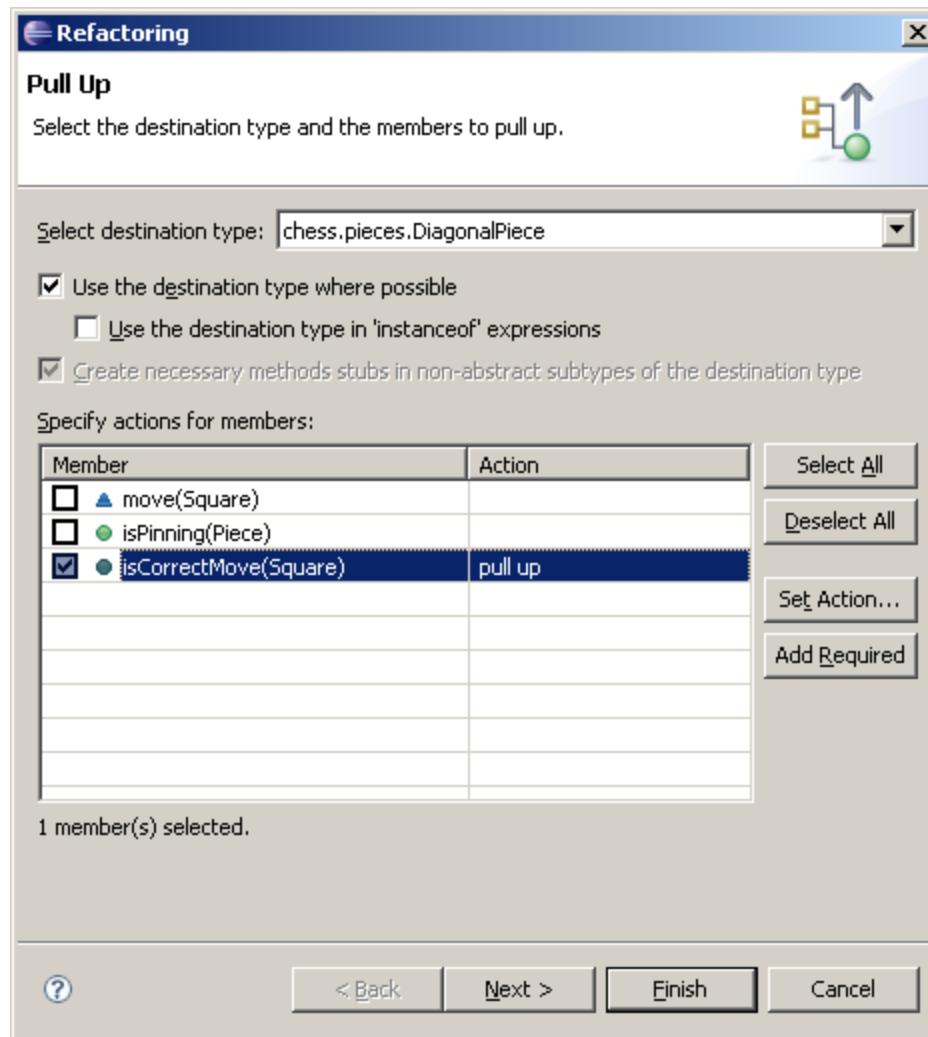
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Перемещение метода (3)

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

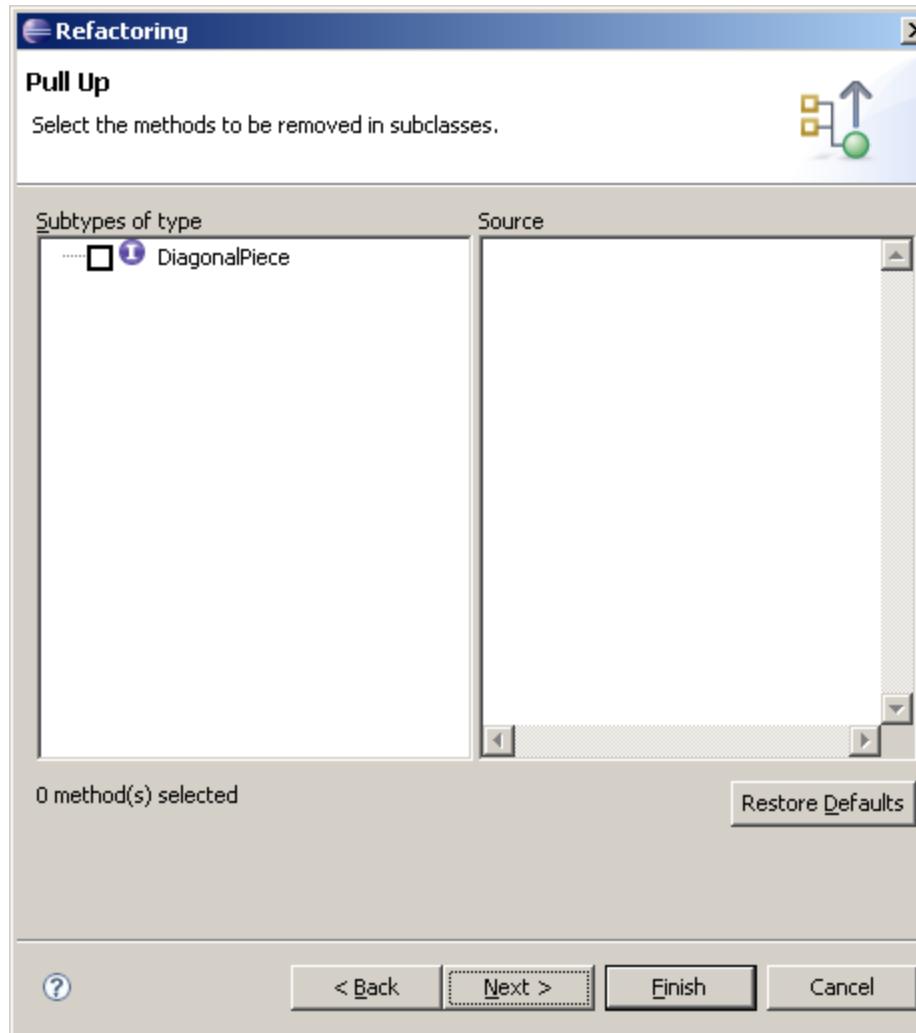
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Перемещение метода (4)

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

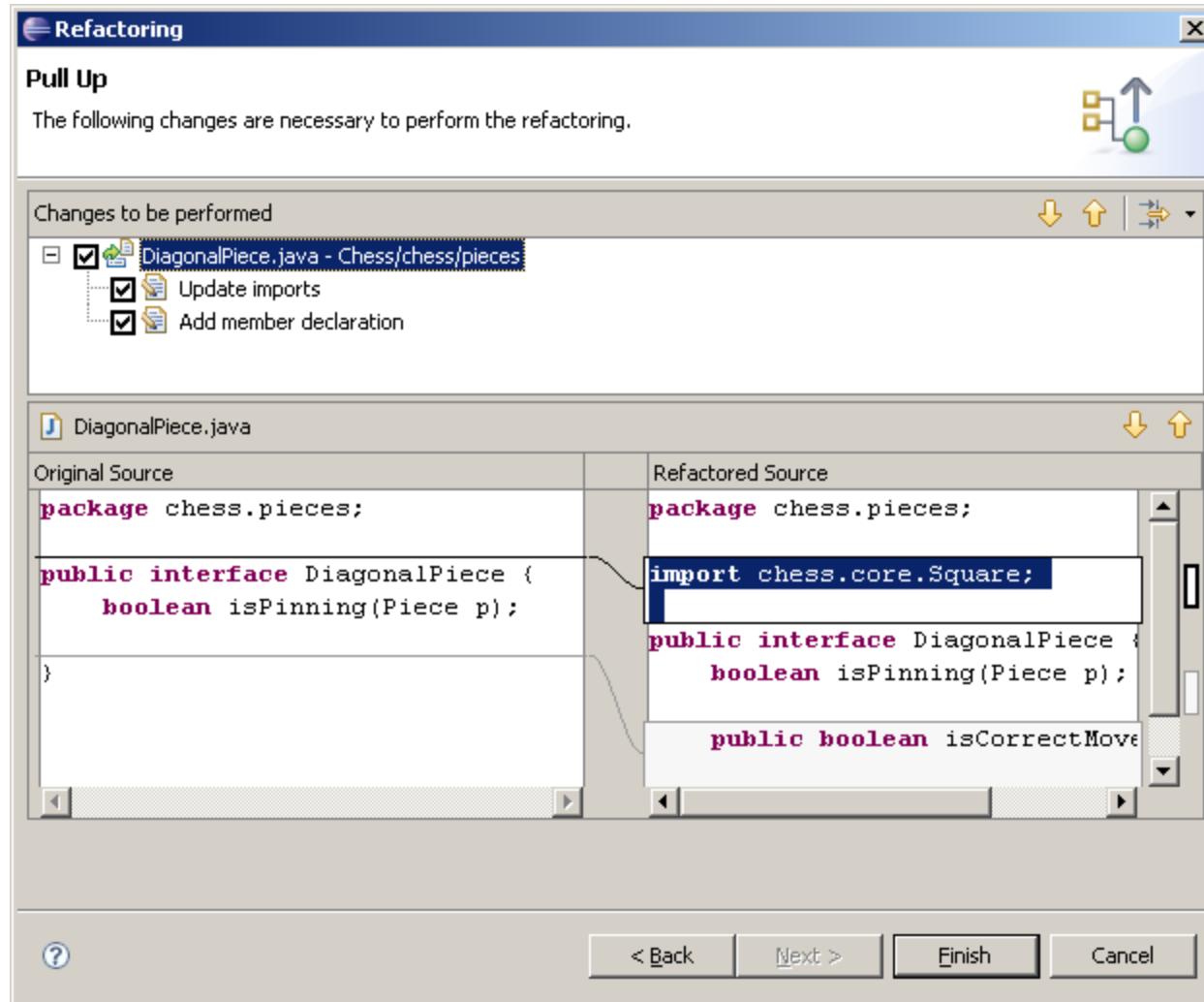
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Перемещение метода (5)

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

Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Использование полиморфизма (1)

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

Романов Владимир Юрьевич ©2025

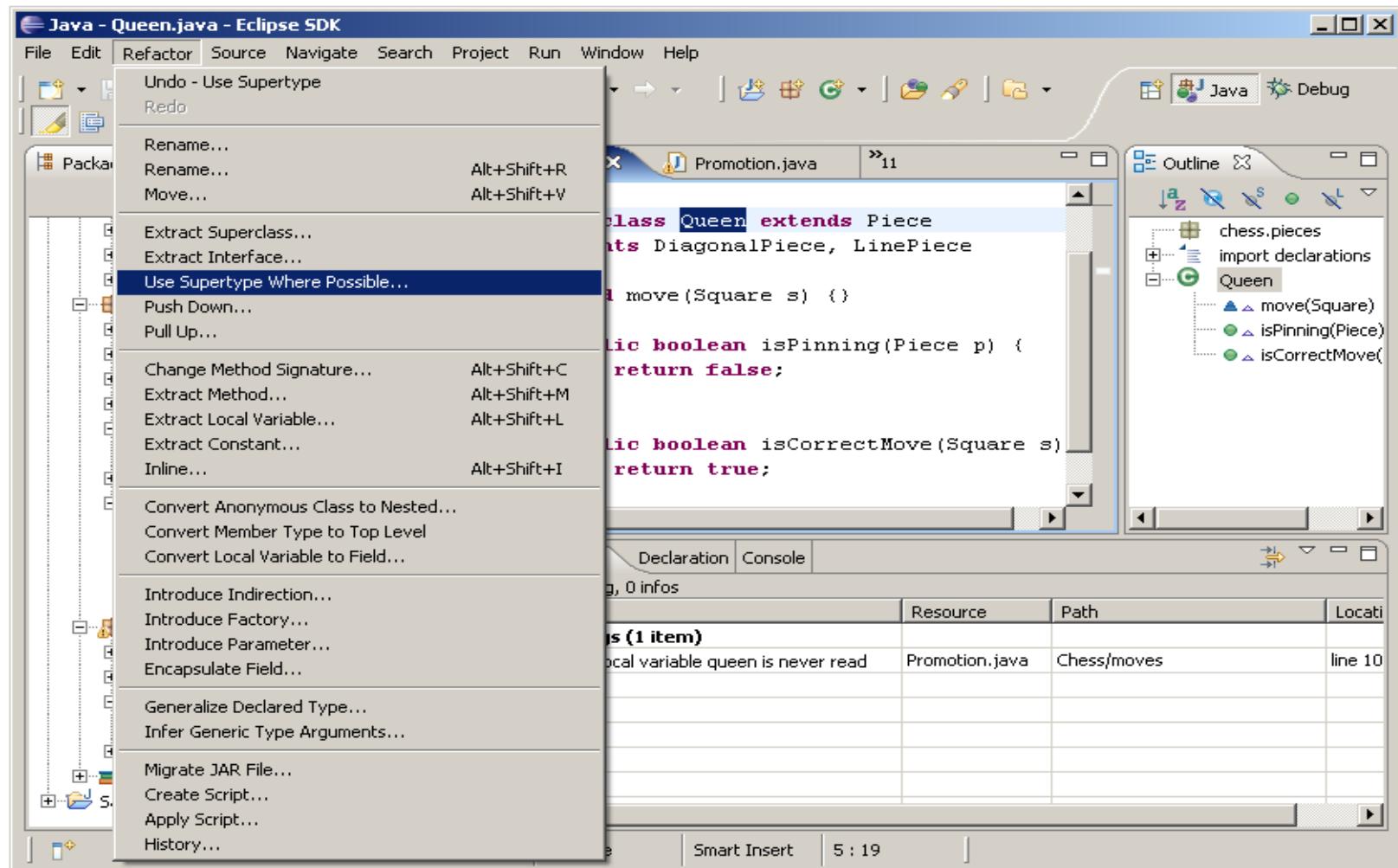
- Замена имени класса-потомка на имя класса-предка
- После этого программа работает со всеми потомками класса-предка

# Рефакторинг в Eclipse.

## Использование полиморфизма (2)

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

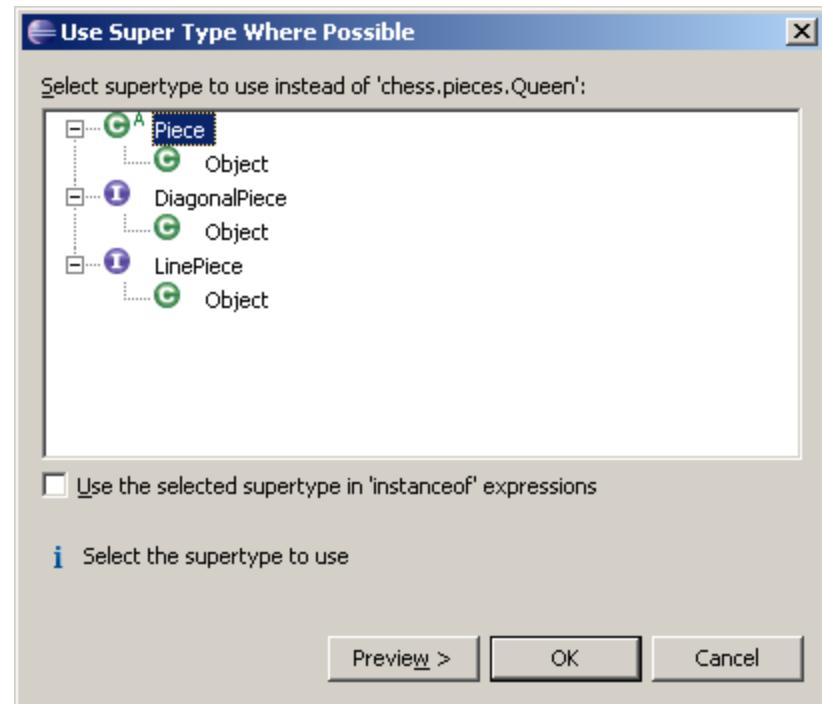
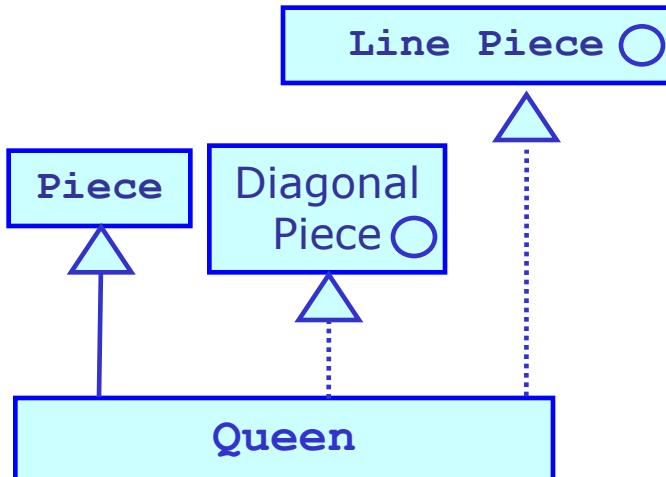
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Использование полиморфизма (3)

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

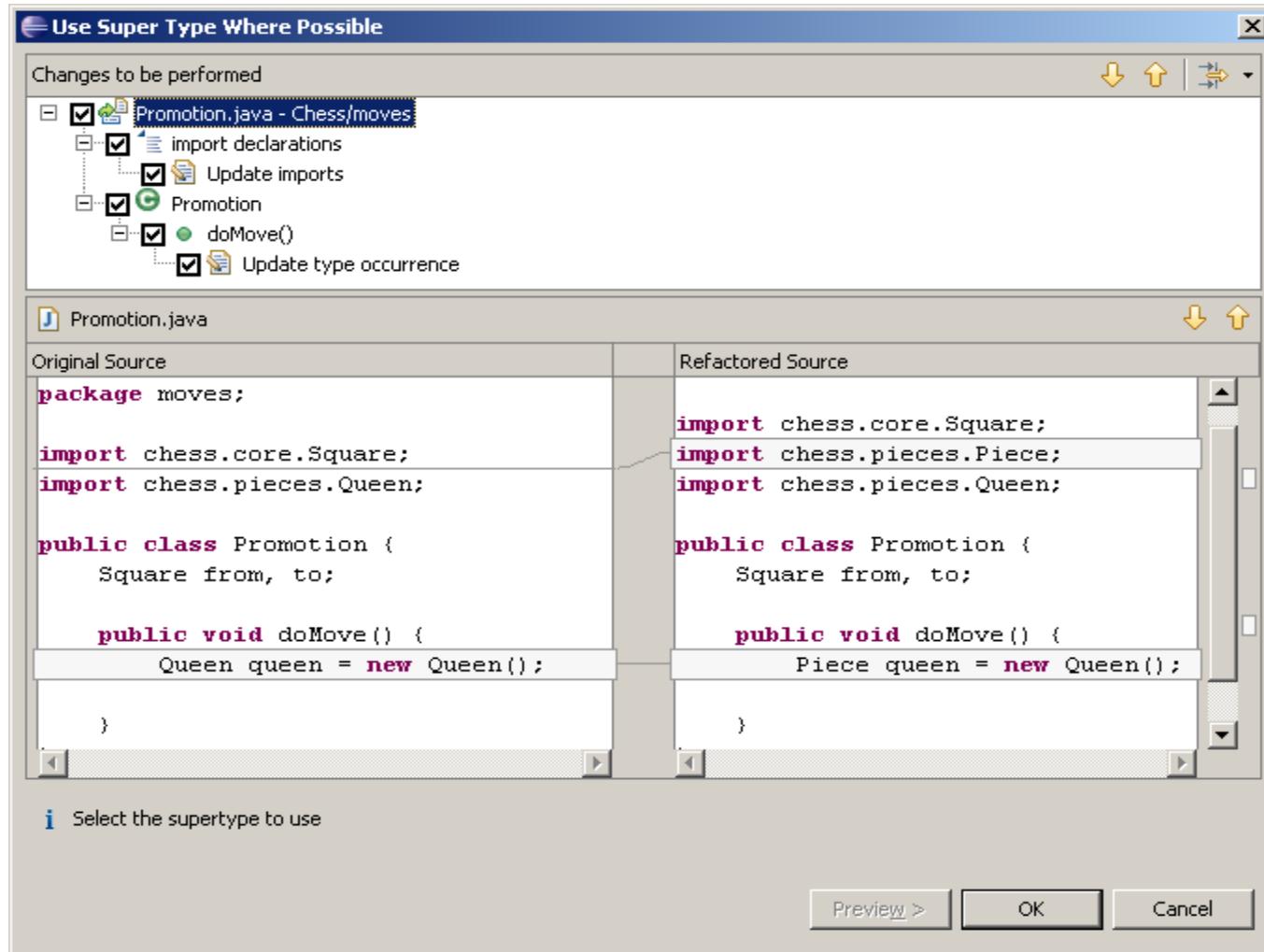
Романов Владимир Юрьевич ©2025



# Рефакторинг в Eclipse. Использование полиморфизма (4)

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

Романов Владимир Юрьевич ©2025

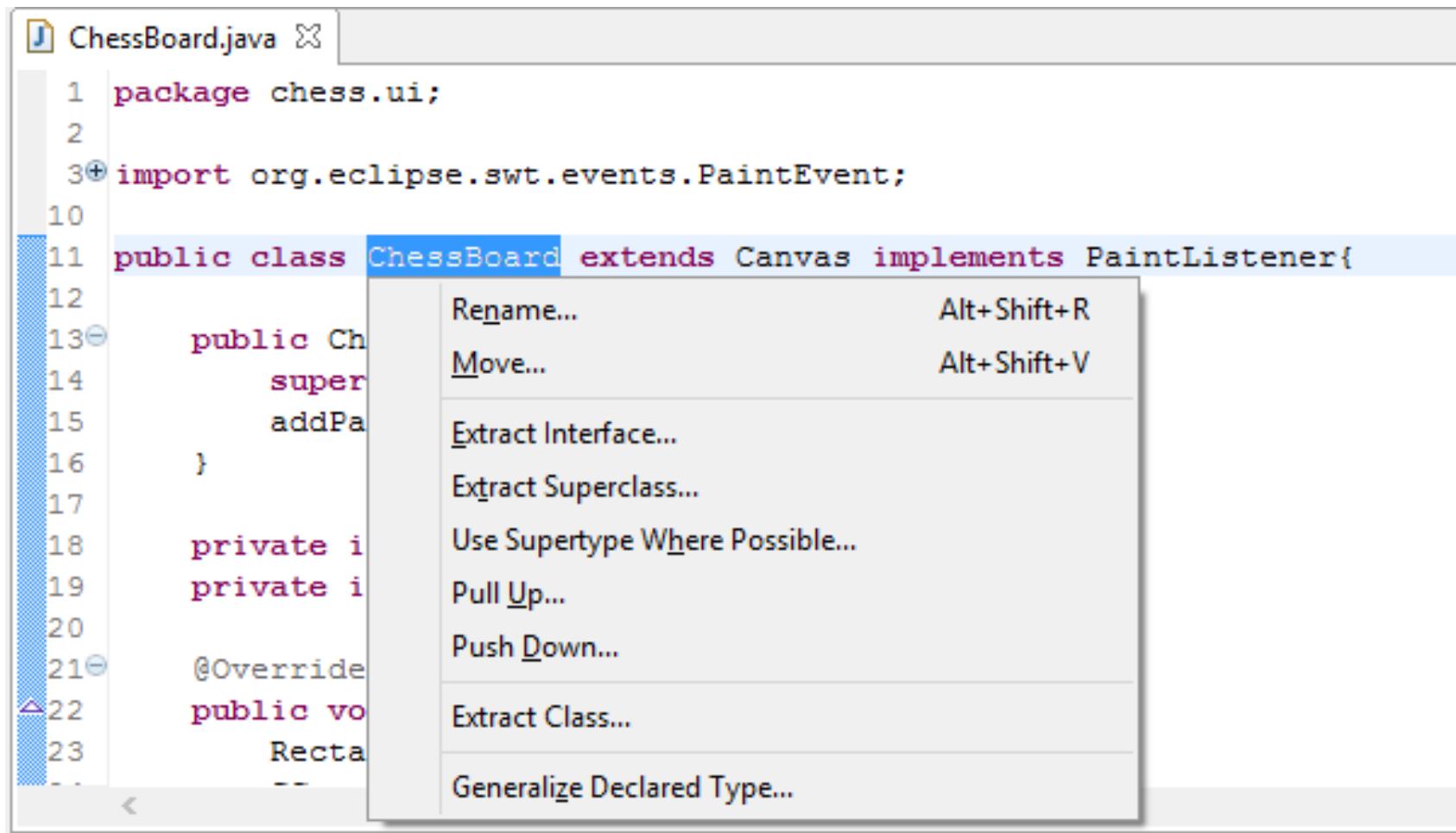


# Контексто-зависимый рефакторинг. Клавиши **Alt + Shift + T**

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

Романов Владимир Юрьевич ©2025

## Преобразования применимые к классу **ChessBoard**



The screenshot shows an IDE interface with a code editor and a context menu. The code editor displays a Java file named 'ChessBoard.java' with the following content:

```
1 package chess.ui;
2
3+ import org.eclipse.swt.events.PaintEvent;
10
11 public class ChessBoard extends Canvas implements PaintListener{
12
13     public Ch
14         super
15             addPa
16     }
17
18     private i
19     private i
20
21     @Override
22     public vo
23         Recta
--
```

The context menu is open over the line 'public class ChessBoard extends Canvas implements PaintListener{'. The menu items are:

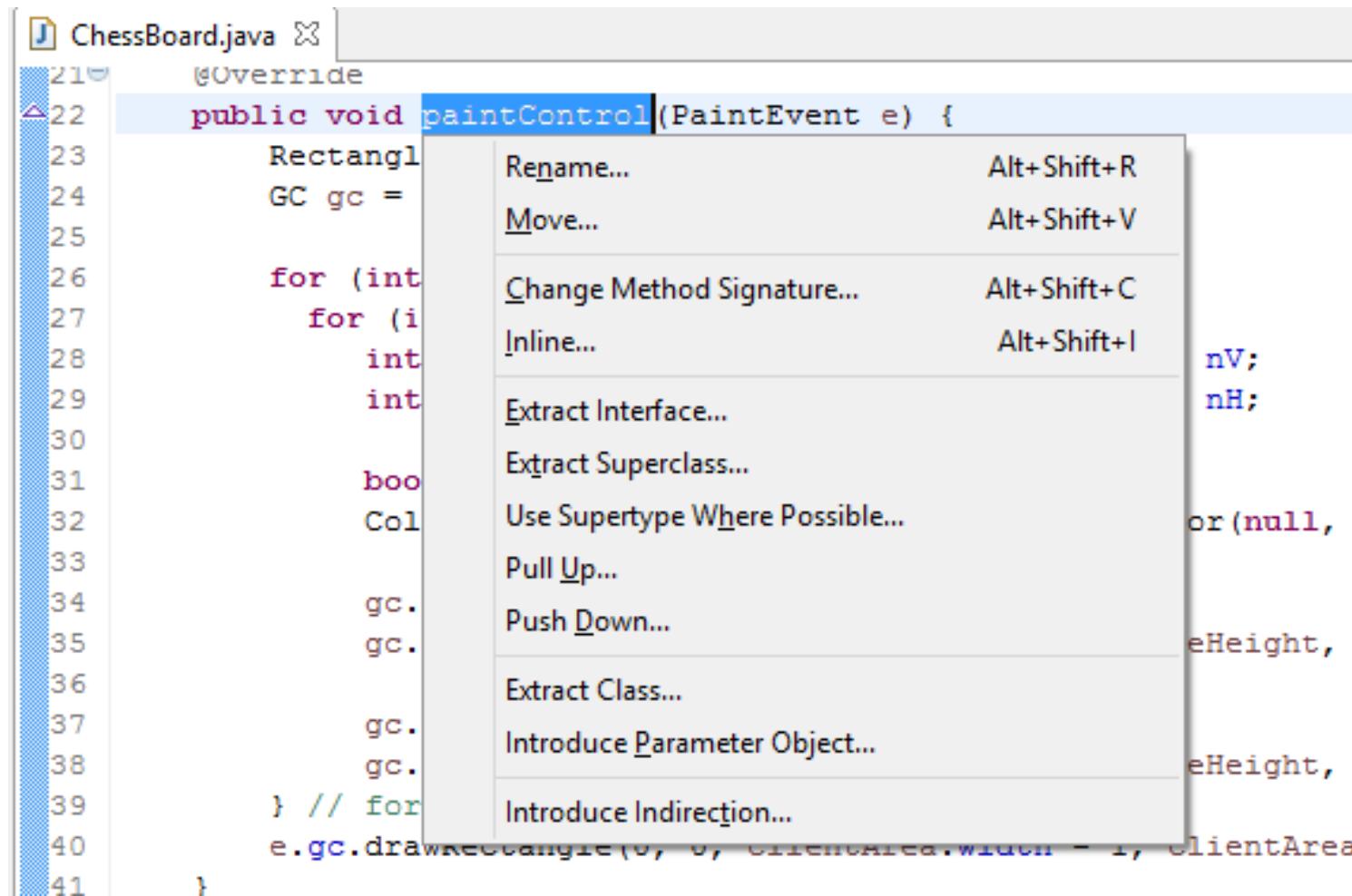
- Rename... (Alt+Shift+R)
- Move... (Alt+Shift+V)
- Extract Interface...
- Extract Superclass...
- Use Supertype Where Possible...
- Pull Up...
- Push Down...
- Extract Class...
- Generalize Declared Type...

# Контексто-зависимый рефакторинг. Клавиши **Alt + Shift + T**

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

Романов Владимир Юрьевич ©2025

## Преобразования применимые к методу `paintControl`

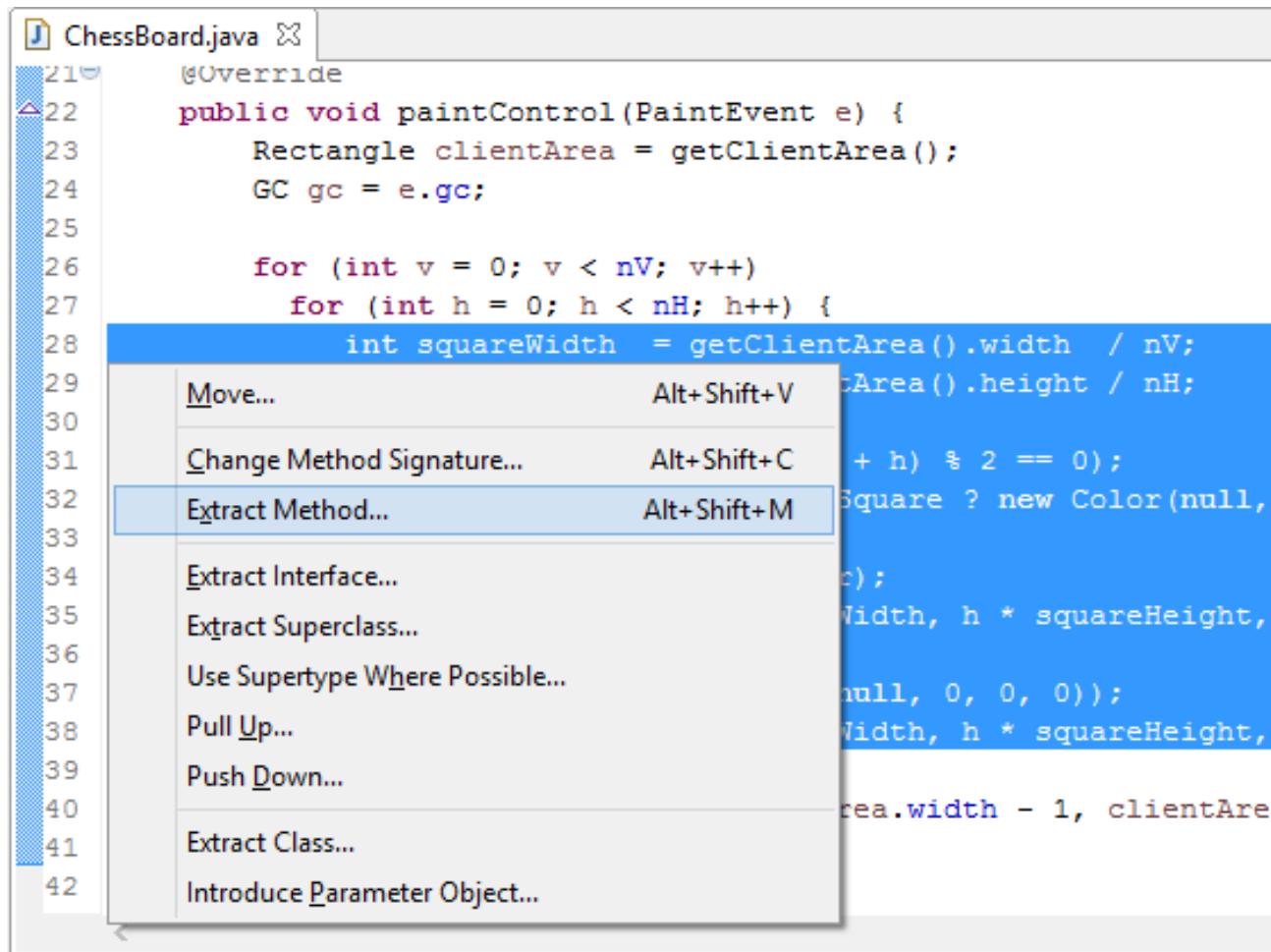


# Контексто-зависимый рефакторинг. Клавиши **Alt + Shift + T**

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

Романов Владимир Юрьевич ©2025

## 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 ©2025

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

# Исключения в языке Java.

## Определение

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

Романов Владимир Юрьевич ©2025

- Исключение

- Событие или условие которое нарушает нормальных 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 ©2025

- 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

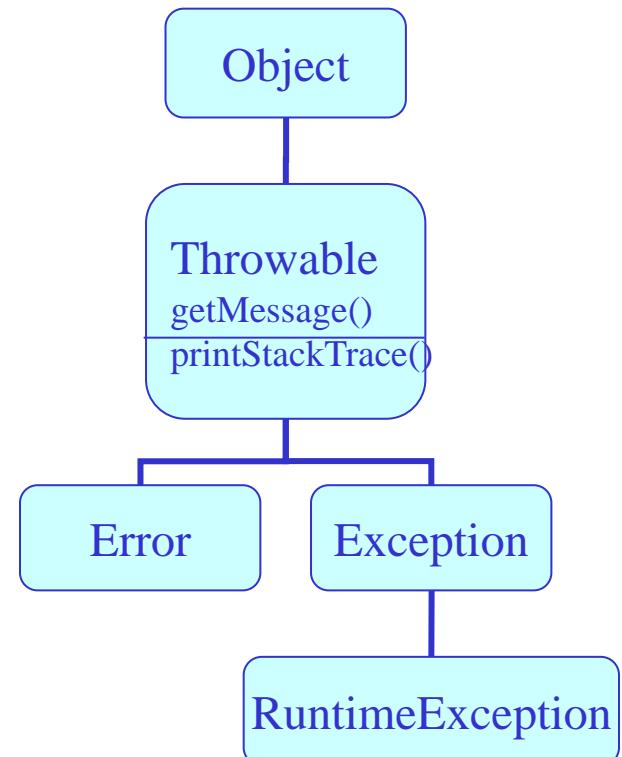
# Исключения в языке Java.

## Иерархия исключений

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

Романов Владимир Юрьевич ©2025

- **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**

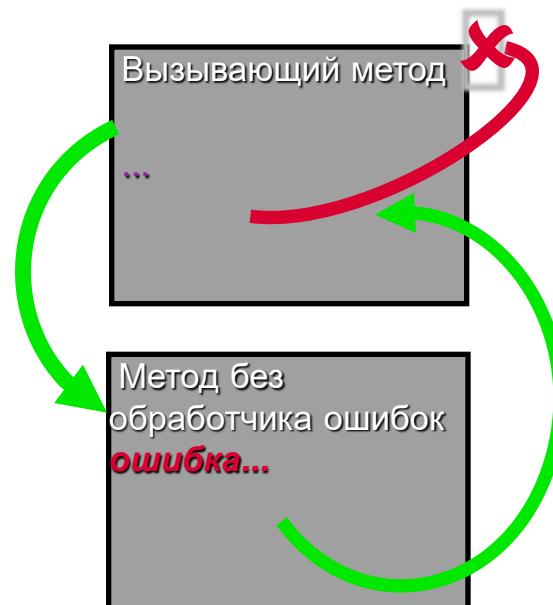
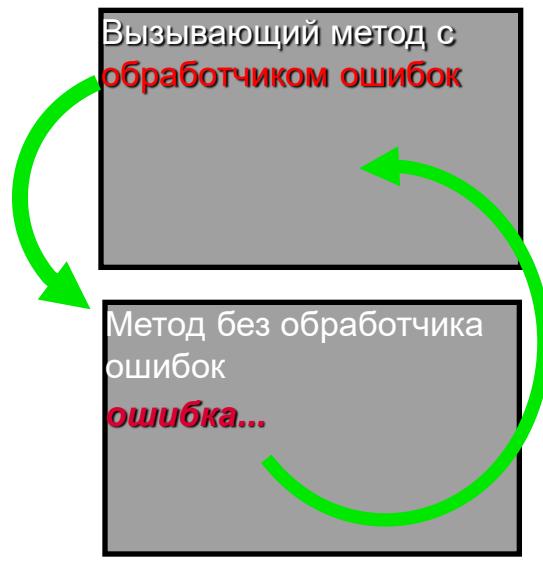


# Исключения в языке Java. Обработка исключений

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- ***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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- Перехват исключения в порождающем ситуацию 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 ©2025

```
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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

```
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 ©2025

- Методы объявленные в интерфейсе реализуются в классе, поддерживающим 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 независим от положения в иерархии классов

# Интерфейсы в языке Java.

## Соглашения об именах интерфейсов

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- 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 ©2025

- **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 ©2025

- **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 из 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.

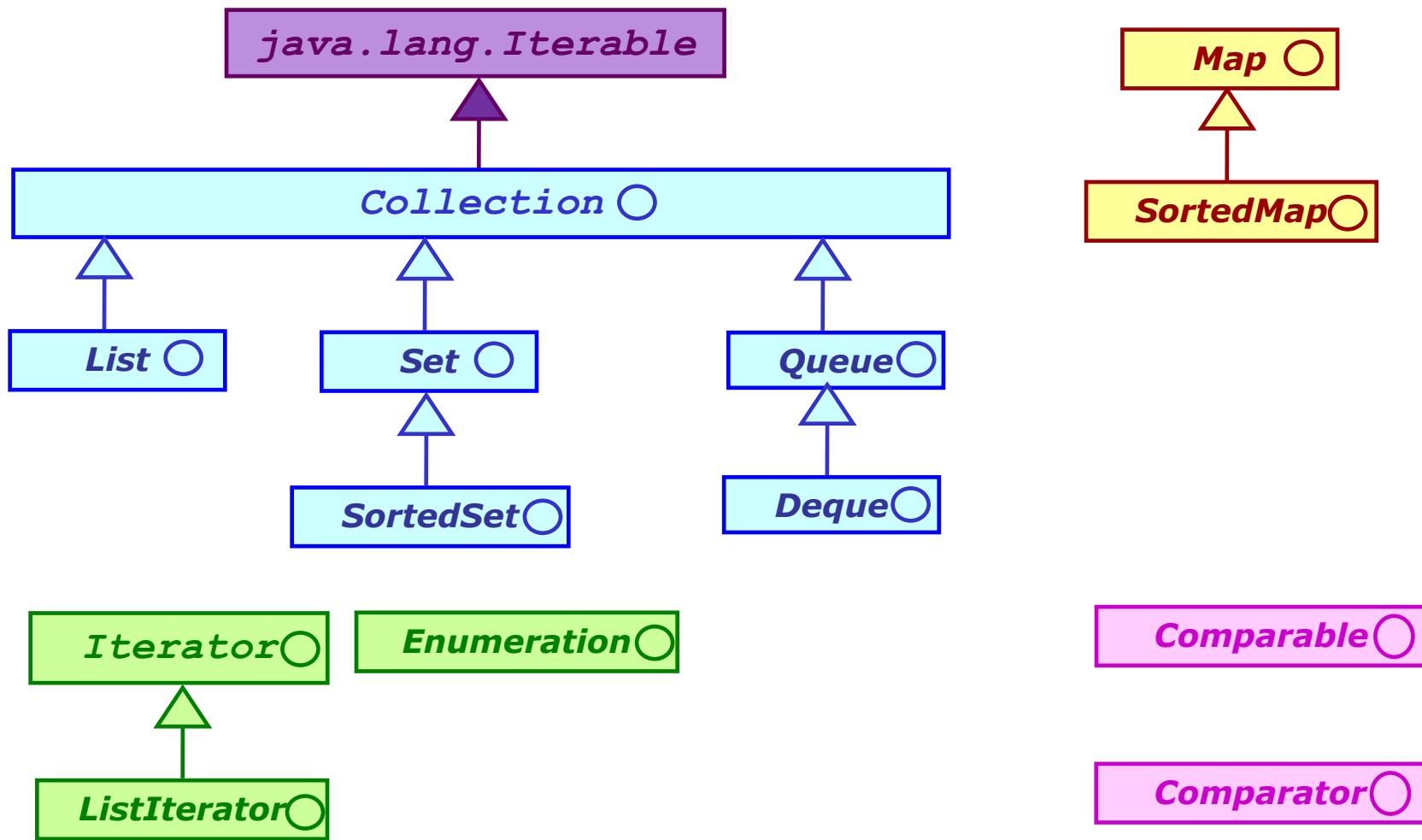
Романов Владимир Юрьевич ©2025

- 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 ©2025



# Коллекции языка Java.

## Интерфейс *Collection*

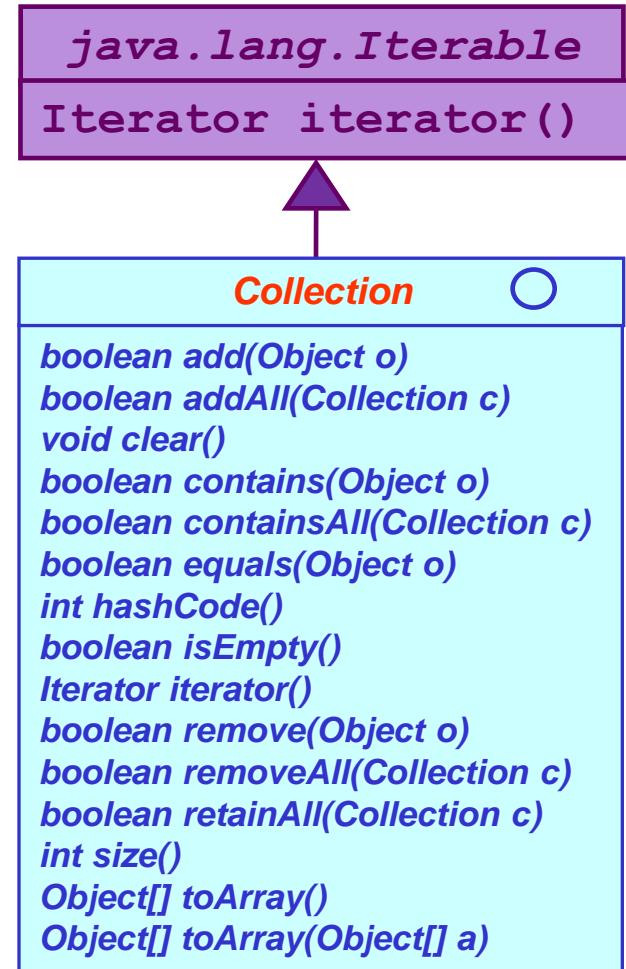
МГУ им. М.В.Ломоносова. Факультет ВМК.

Романов Владимир Юрьевич ©2025

- 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;

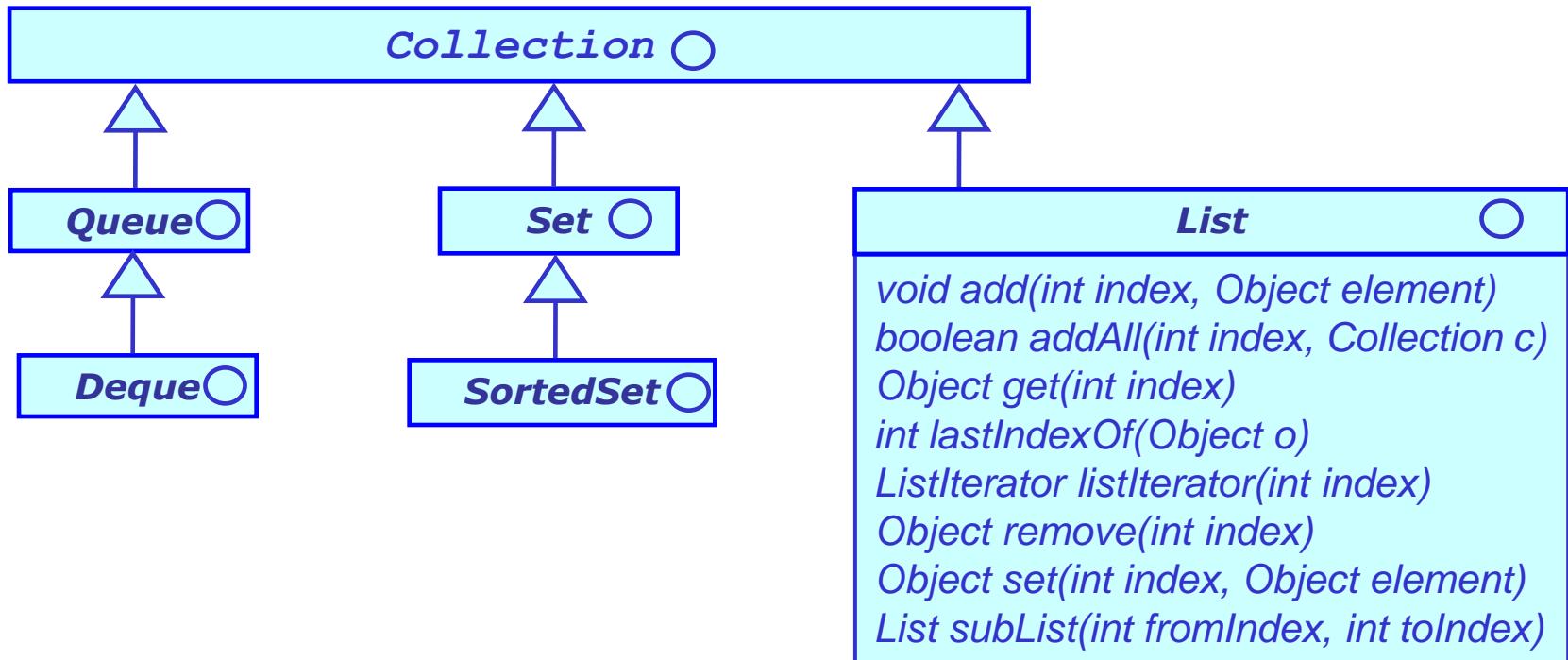
for(Object o : c) {
    System.out.println(o);
}
```



# Коллекции языка Java. Интерфейс списка

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

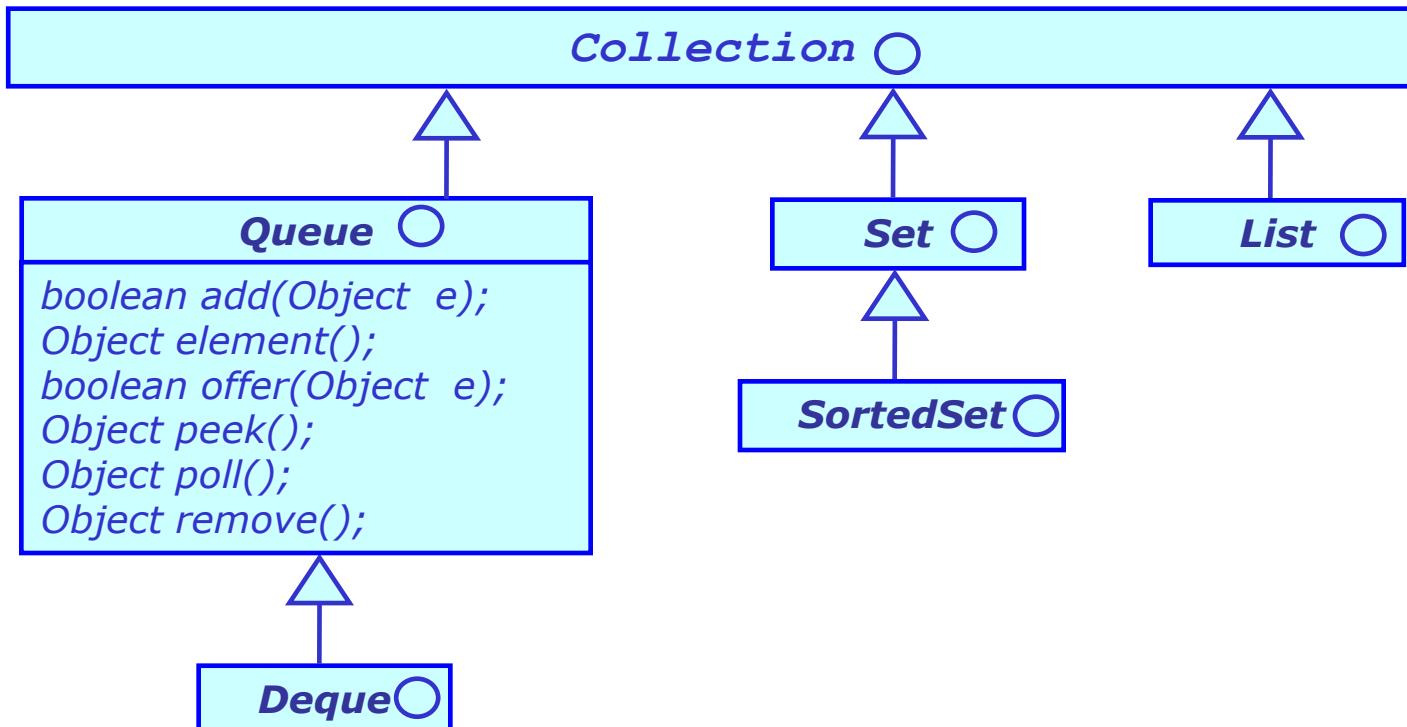
Романов Владимир Юрьевич ©2025



# Коллекции языка Java. Интерфейс очереди Queue

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

Романов Владимир Юрьевич ©2025



# Коллекции языка Java.

## Интерфейс очереди Queue

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

Романов Владимир Юрьевич ©2025

Действия по добавлению, удалению и проверки элемента в очереди.

Различие – что делается если действие выполнить невозможно

### Queue

```
boolean add(Object e);
Object element();
boolean offer(Object e);
Object peek();
Object poll();
Object remove();
```

#### Порождение ситуации

**Вставка** *boolean add(Object e)*

**Удаление** *Object remove()*

**Проверка** *Object element()*

#### Возвращается значение или null

*boolean offer(Object e)*

*Object poll()*

*Object peek()*

# Коллекции языка Java.

## Интерфейс очереди Queue

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

Романов Владимир Юрьевич ©2025

### **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 ©2025

### **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 ©2025

| <b>Deque</b>   | ○ |
|--|---|
| <i>boolean addFirst(Object e);<br/>Object elementFirst();<br/>boolean offerFirst (Object e);<br/>Object peekFirst();<br/>Object pollFirst();<br/>Object removeFirst();</i> |   |

|                 |  |   |
|-----------------|--|---|
| <b>Вставка</b>  | <b>Ситуация</b><br><i>boolean addFirst(Object e)</i> | <b>Значение или null</b><br><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 ©2025

| <b>Deque</b>   | ○ |
|--|---|
| <i>boolean addLast(Object e);<br/>Object elementLast();<br/>boolean offerLast (Object e);<br/>Object peekLast();<br/>Object pollLast();<br/>Object removeLast();</i> |   |

|                 |                 |                                  |
|-----------------|-----------------|----------------------------------|
| <b>Вставка</b>  | <b>Ситуация</b> | <i>boolean addLast(Object e)</i> |
| <b>Удаление</b> |                 | <i>Object removeLast()</i>       |
| <b>Проверка</b> |                 | <i>Object elementLast()</i>      |

|                          |                                    |
|--------------------------|------------------------------------|
| <b>Значение или null</b> | <i>boolean offerLast(Object e)</i> |
|                          | <i>Object pollLast()</i>           |
|                          | <i>Object peekLast()</i>           |

# Коллекции языка Java.

## Интерфейс *Map*

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

Романов Владимир Юрьевич ©2025

- 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

| <b>Map</b>                                 |  |
|--|---|
| <i>void clear()</i>                        |   |
| <i>boolean containsKey(Object key)</i>     |   |
| <i>boolean containsValue(Object value)</i> |   |
| <i>Set entrySet()</i>                      |   |
| <i>boolean equals(Object o)</i>            |   |
| <i>Object get(Object key)</i>              |   |
| <i>int hashCode()</i>                      |   |
| <i>boolean isEmpty()</i>                   |   |
| <i>Set keySet()</i>                        |   |
| <i>Object put(Object key)</i>              |   |
| <i>void putAll(Map t)</i>                  |   |
| <i>Object remove(Object key)</i>           |   |
| <i>int size()</i>                          |   |
| <i>Collection values()</i>                 |   |

# Сравнение объектов.

## Упорядочивание объектов

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

Романов Владимир Юрьевич ©2025

- 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 ©2025

- Интерфейс *Comparable*
  - Реализуют классы, объекты которых способны сравнивать 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

|                                |   |
|--------------------------------|---|
| <b><i>Comparable</i></b>       | ○ |
| <i>int compareTo(Object o)</i> |   |

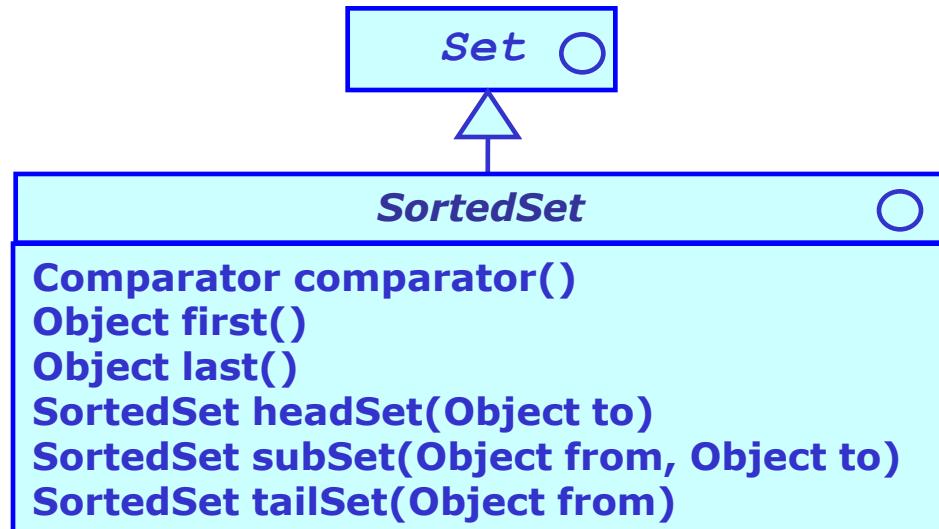
|  |   |
|--|---|
| <b><i>Comparator</i></b>                 | ○ |
| <i>int compare(Object o1, Object o2)</i> |   |

# Сортированные коллекции.

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

Романов Владимир Юрьевич ©2025

- *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

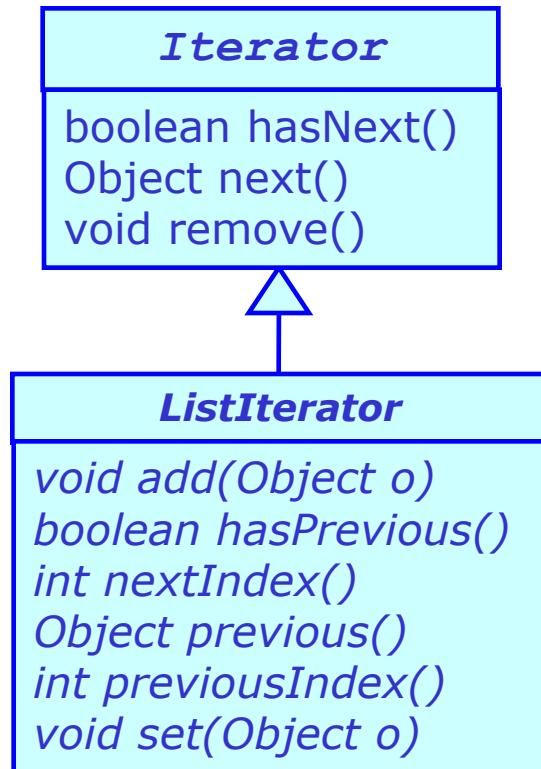


# Итераторы.

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

Романов Владимир Юрьевич ©2025

- 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



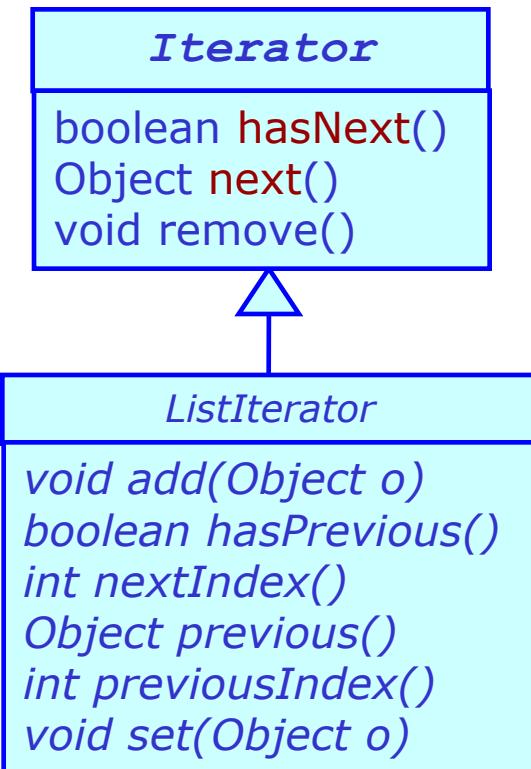
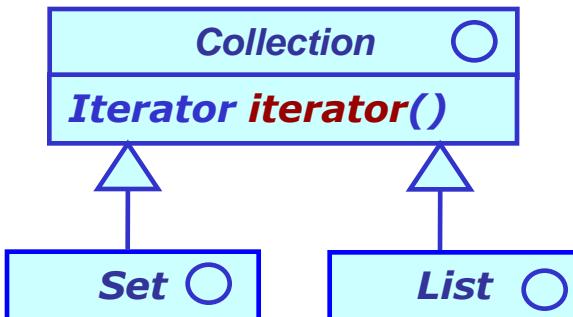
# Шаблон кода итератора.

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

Романов Владимир Юрьевич ©2025

```
Collection c;

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

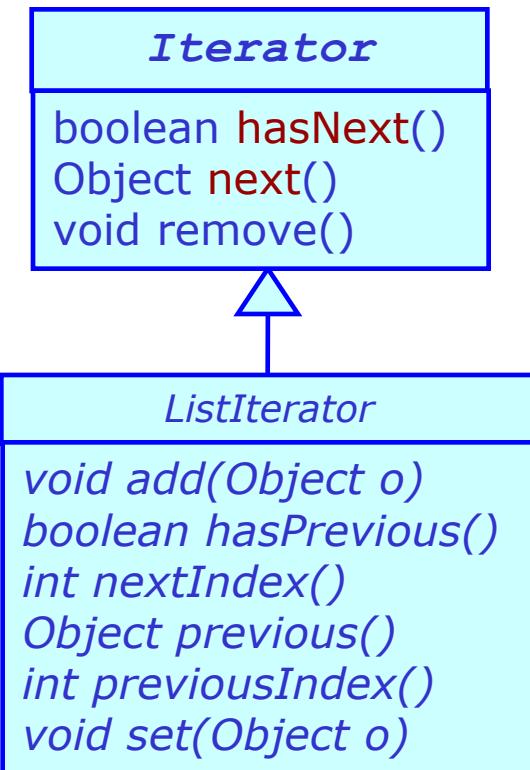
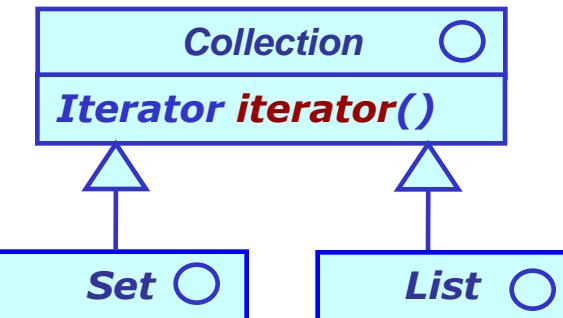


# Итератор встроенный в язык Java.

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

Романов Владимир Юрьевич ©2025

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



# Интерфейсы и реализации.

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

Романов Владимир Юрьевич ©2025

| Интерфейсы | Реализации  |            |         |                  |                       |
|------------|-------------|------------|---------|------------------|-----------------------|
|            | Хэш таблица | Массив     | Дерево  | Связанный список | Устаревшие            |
| Set        | HashSet     |            | TreeSet |                  |                       |
| List       |             | ArrayList  |         | LinkedList       | Vector, Stack         |
| Map        | HashMap     |            | TreeMap |                  | HashTable, Properties |
| Queue      |             | ArrayDeque |         | LinkedList       |                       |
| Deque      |             | ArrayDeque |         | LinkedList       |                       |

Рекомендация: при кодировании использовать интерфейсы, а не реализацию

# Пример. Использование карты.

## Частота слов в командной строке

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

Романов Владимир Юрьевич ©2025

```
public class MapExample {  
    public static void main(String args[]) {  
        Map map = new HashMap();  
        Integer ONE = new Integer(1);  
  
        for (String key : args) {  
            Integer frequency = (Integer) map.get(key);  
            if (frequency == null) {  
                frequency = ONE;  
            } else {  
                int value = frequency.intValue();  
                frequency = new Integer(value + 1);  
            }  
            map.put(key, frequency);  
        }  
        System.out.println(map);  
        Map sortedMap = new TreeMap(map);  
        System.out.println(sortedMap);  
    }  
}
```

# Пример. Использование карты в Java8.

## Частота слов в командной строке

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

Романов Владимир Юрьевич ©2025

```
public class MapExample {  
    public static void main(String args[]) {  
        Map<String, Integer> map = new HashMap<>();  
  
        for (String key : args) {  
            int frequency = !map.containsKey(key) ? 1 : map.get(key) + 1;  
            map.put(key, frequency);  
        }  
  
        System.out.format("HashMap: %s %n%n", map);  
  
        Map<String, Integer> sortedMap = new TreeMap<>(map);  
        System.out.format("TreeMap: %s %n%n", sortedMap);  
    }  
}
```

# Пример. Использование карты в Java8. Частота слов в командной строке

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

Романов Владимир Юрьевич ©2025

```
System.out.format("Keys:  %s %n%n", sortedMap.keySet());
System.out.format("Values:  %s %n%n", sortedMap.values());
System.out.format("EntrySet: %s %n%n", sortedMap.entrySet());

for (Entry<String, Integer> entry : sortedMap.entrySet()) {
    System.out.format("Город: %10s Частома=%d %n",
                      entry.getKey(),
                      entry.getValue());
}
```

# Пример. Использование карты.

## Частота слов в командной строке

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

Романов Владимир Юрьевич ©2025

### Ввод

**Москва Тула Москва Петербург Москва Тула**

### Вывод

**HashMap: {Петербург=1, Тула=2, Москва=3}**

**TreeMap: {Москва=3, Петербург=1, Тула=2}**

**Keys: [Москва, Петербург, Тула]**

**Values: [3, 1, 2]**

**EntrySet: [Москва=3, Петербург=1, Тула=2]**

**Город: Москва Частота=3**

**Город: Петербург Частота=1**

**Город: Тула Частота=2**

# Сравнение реализаций. Множества и карты

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

Романов Владимир Юрьевич ©2025

- **Set / Map**
  - **HashSet / HashMap**
    - Очень быстрая, без упорядочения
    - Выбирается начальная плотность (*initial capacity*) и коэффициент загрузки (*load factor*) для улучшения представления
  - **TreeSet / TreeMap**
    - Хранит сбалансированное дерево, хорошо для сортированных вставок
    - Нет параметров настройки
  - **HashTable**
    - Синхронизирована
    - Рекомендуется использовать через интерфейс **Map**

# Сравнение реализаций. Списки

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

Романов Владимир Юрьевич ©2025

- **List**
  - **ArrayList**
    - Очень быстрый
    - Можно использовать «native» метод **System.arraycopy**
  - **LinkedList**
    - Хорошо использовать для меняющихся коллекций или для вставки в начало списка (для очередей Queue и Deque)
  - **Vector**
    - Синхронизированный
    - Рекомендуется использовать через интерфейс **List**

# Сравнение реализаций. Списки

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

Романов Владимир Юрьевич ©2025

- **Queue и Deque**

- **LinkedList**

- Хорошо использовать для меняющихся коллекций или для вставки в начало списка (для очередей Queue и Deque)

- **DeckArray**

- Очень быстрый
    - Если не надо уделять элементы из середины
    - Можно использовать «native» метод **System.arraycopy**

# Сравнение реализаций. Устаревшие коллекции

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

Романов Владимир Юрьевич ©2025

- Классы устаревший коллекций по прежнему доступны, но их реализации были изменены.
  - **java.util.Vector**
    - Расширяемый индексируемый список
  - **java.util.Stack**
    - Расширяет вектор операциями *push* и *pop*
  - **java.util.BitSet**
    - Расширяемое множество «флагов» True/False
  - **java.util.Dictionary**
    - Этот класс заменен на `java.util.Map`
  - **java.util.Hashtable**
    - Эффективное хранение данных без сортировки
  - **java.util.Properties**
    - Хранит пары ключ-значение. Ключ есть имя свойства.

# Коллекции языка Java. Интерфейсы в коллекциях. Comparable

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

Романов Владимир Юрьевич ©2025

```
class Point implements Comparable<Point> {  
    public int x, y;  
  
    static public Point p0 = new Point(0, 0);  
  
    public Point(int x, int y)  
    { this.x = x; this.y = y; }  
  
    public int compareTo(Point p) {  
        double d = p0.distance(p) - p0.distance(this);  
        return d == 0 ? 0 : (d > 0 ? -1 : 1);  
    }  
  
    private double distance(Point p) {  
        int dx = x - p.x; int dy = y - p.y;  
        return Math.sqrt(dx*dx + dy*dy);  
    }  
}
```

Comparable

# Коллекции языка Java. Интерфейсы в коллекциях. Comparable

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

Романов Владимир Юрьевич ©2025

```
public class ComparableExample {  
    public static void main(String args[]) {  
        SortedSet<Point> points = new TreeSet<>();  
  
        points.add( new Point(10, 9) );  
        points.add( new Point( 2, 2) );  
        points.add( new Point( 1, 3) );  
  
        for (Point p : points)  
            System.out.format("(%2s, %2s) %n", p.x, p.y);  
    }  
}
```

**Comparable**○

# Коллекции языка Java.

## Интерфейсы в коллекциях. Comparable

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

Романов Владимир Юрьевич ©2025

( 2, 2)  
( 1, 3)  
(10, 9)

*Comparable* 

# Коллекции языка Java.

## Интерфейсы в коллекциях. Comparator

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

Романов Владимир Юрьевич ©2025

```
class Person {  
    public String name;  
    public int age;  
    public Point location;  
  
    public Person(String name, int age, Point location) {  
        this.name = name;  
        this.age = age;  
        this.location = location;  
    }  
  
    public String toString() {  
        return String.format("name=%7s age=%7s, location=(%2s, %2s)%n",  
                            name, age, location.x, location.y);  
    }  
}
```

Comparator

# Коллекции языка Java.

## Интерфейсы в коллекциях. Comparator

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

Романов Владимир Юрьевич ©2025

```
public class ComparatorExample {  
    public static void main(String args[]) {  
        List<Person> persons = new ArrayList<>();  
        persons.add( new Person("Bill", 19, new Point(40, 50) ) );  
        persons.add( new Person("Alex", 9, new Point(60, 40) ) );  
        persons.add( new Person("John", 3, new Point(10, 90) ) );  
  
        persons.sort ( new Comparator<Person>() {  
            public int compare(Person p1, Person p2) {  
                return p1.name.compareTo(p2.name);  
            }  
        });  
  
        System.out.println();  
        for (Person p : persons)  
            System.out.print("by name: " + p);  
    }  
}
```

Comparator

# Коллекции языка Java.

## Интерфейсы в коллекциях. Comparator

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

Романов Владимир Юрьевич ©2025

```
persons.sort( new Comparator<Person>() {  
    @Override  
    public int compare(Person p1, Person p2) {  
        int d = p1.age - p2.age;  
        return d == 0 ? 0 : (d > 0 ? 1 : -1);  
    }  
});  
  
System.out.println();  
for (Person p : persons)  
    System.out.print("by age: " + p);
```

Comparator

# Коллекции языка Java.

## Интерфейсы в коллекциях. Comparator

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

Романов Владимир Юрьевич ©2025

```
persons.sort( new Comparator<Person>() {  
    @Override  
    public int compare(Person p1, Person p2) {  
        return p1.location.compareTo(p2.location);  
    }  
});
```

Comparator

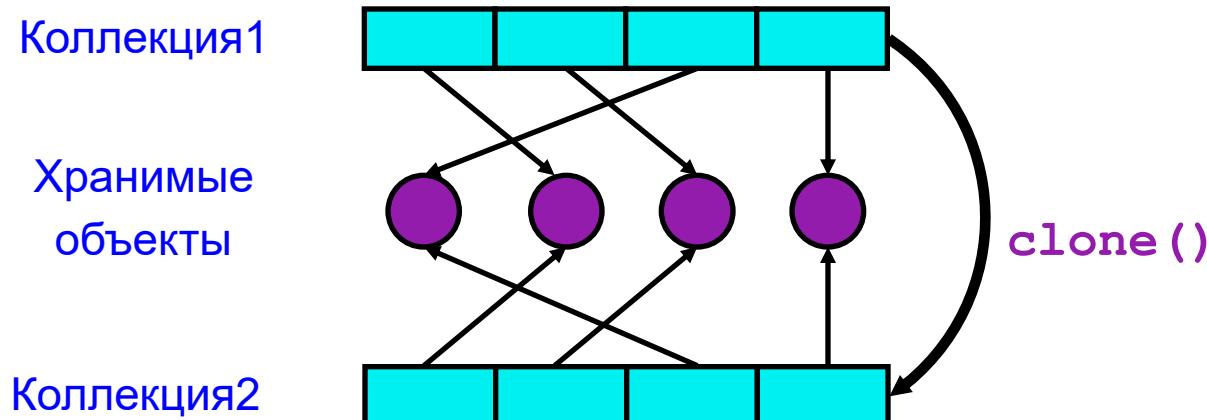
```
System.out.println();  
for (Person p : persons)  
    System.out.print("by location: " + p);
```

# Клонирование коллекций

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

Романов Владимир Юрьевич ©2025

- Для большинства коллекций можно сделать копию, не создавая копию для хранимых объектов («неглубокое» копирование)



# Алгоритмы. Класс *Collections*

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

Романов Владимир Юрьевич ©2025

- *java.util.Collections* содержит только статические методы для работы с коллекциями или для создания коллекций. Он содержит:
  - Полиморфные алгоритмы для работы с коллекциями, например:
    - **binarySearch**
    - **copy**
    - **min и max**
    - **replace**
    - **reverse**
    - **rotate**
    - **shuffle**
    - **sort**
    - **swap**
  - «Обертки» – возвращают новые коллекции на основе имеющихся
    - **Синхронизированные коллекции**
    - **Не модифицируемые коллекции**

# Алгоритмы. Класс *Collections*. Добавление строк и сортировка

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

Романов Владимир Юрьевич ©2025

```
List<String> strings = new ArrayList<>();  
  
Collections.addAll(strings, "Moscow", "London", "Berlin");  
  
System.out.println("До: " + strings);  
Collections.sort(strings);  
System.out.println("После: " + strings);
```

До: [Moscow, London, Berlin]

После: [Berlin, London, Moscow]

# Алгоритмы. Класс *Collections*. Добавление массива строк и сортировка

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

Романов Владимир Юрьевич ©2025

```
List<String> strings = new ArrayList<>();
```

```
String[] cities = { "Moscow", "London", "Berlin" };
```

```
Collections.addAll(strings, cities);
```

```
System.out.println("До: " + strings);
```

```
    Collections.sort(strings);
```

```
System.out.println("После: " + strings);
```

До: [Moscow, London, Berlin]

После: [Berlin, London, Moscow]

# Алгоритмы. Класс *Collections*. Сортировка без учета регистра

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

Романов Владимир Юрьевич ©2025

```
Comparator<String> comparator = new Comparator<String>(){
    public int compare(String s1, String s2) {
        return s1.compareToIgnoreCase(s2);
    }
};

List<String> strings = new ArrayList<>();

String[] cities = { "Moscow", "berlin", "London", "moscow", "Berlin" };

Collections.addAll(strings, cities);

System.out.println("До: " + strings);
Collections.sort(strings, comparator);
System.out.println("После: " + strings);
```

До: [Moscow, berlin, London, moscow, Berlin]  
После: [berlin, Berlin, London, Moscow, moscow]

# Алгоритмы. Класс *Collections*.

## Java8 – переходим к ламбда-выражениям

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

Романов Владимир Юрьевич ©2025

```
Comparator<String> comparator = new Comparator<String>(){
    public int compare(String s1, String s2) {
        return s1.compareToIgnoreCase(s2);
    }
};

List<String> strings = new ArrayList<>();

String[] cities = { "Moscow", "berlin", "London", "moscow", "Berlin" };

Collections.addAll(strings, cities);

System.out.println("До: " + strings);
Collections.sort(strings, comparator);
System.out.println("После: " + strings);
```

До: [Moscow, berlin, London, moscow, Berlin]  
После: [berlin, Berlin, London, Moscow, moscow]

# Алгоритмы. Класс *Collections*

## Внешний сортировщик – лямбда выражение

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

Романов Владимир Юрьевич ©2025

```
Comparator<String> comparator = (a, b) -> a.compareToIgnoreCase(b);
```

```
List<String> strings = new ArrayList<>();
```

```
String[] cities = { "Moscow", "berlin", "London", "moscow", "Berlin" };
```

```
Collections.addAll(strings, cities);
```

```
System.out.println("До: " + strings);
```

```
    Collections.sort(strings, comparator);
```

```
System.out.println("После: " + strings);
```

До: [Moscow, berlin, London, moscow, Berlin]

После: [berlin, Berlin, London, Moscow, moscow]

# Алгоритмы. Класс *Collections*

## Лямбда выражение в месте вызова сортировки

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

Романов Владимир Юрьевич ©2025

```
List<String> strings = new ArrayList<>();  
  
String[] cities = { "Rome", "berlin", "Bern", "moscow", "Minsk" };  
  
Collections.addAll(strings, cities);  
  
System.out.println("До: " + strings);  
Collections.sort(strings, (s1, s2) -> s1.length() - s2.length() );  
System.out.println("После: " + strings);
```

До: [Rome, berlin, Bern, moscow, Minsk]  
После: [berlin, moscow, Minsk, Rome, Bern]

# Алгоритмы. Класс *Arrays*

## Лямбда выражение в месте вызова сортировки

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

Романов Владимир Юрьевич ©2025

```
public static void main(String[] args) {  
    List<String> strings = Arrays.asList(args);  
  
    strings.forEach(str -> System.out.format(" %s", str));  
}
```

'Moscow' 'London' 'Rome'

# Алгоритмы. Класс *Collections*

## Случайное расположение элементов коллекции

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

Романов Владимир Юрьевич ©2025

```
// Запрашиваем все правильные ходы на доске
// фигурами заданного цвета.
List<Move> correctMoves = getCorrectMoves(board, color);

// Случайным образом переставим ходы в списке
// чтобы игра не повторялась.
Collections.shuffle(correctMoves);

// В переменной brain хранится ссылка на класс реализующий
// интерфейс Comparator.
// Каждая программа-игрок сортирует ходы по своему.
correctMoves.sort(brain);

// После сортировке первый ход – лучший ход.
Move bestMove = correctMoves.get(0);
```

# Родовые типы в языке Java

# Родовые типы языка Java.

## 1. Необходимость родовых типов

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

Романов Владимир Юрьевич ©2025

- Безопасность типов. Контроль типов должен выполняться на этапе компиляции.

```
Map m = new HashMap();
m.put("key", "blarg");
String s = (String) m.get("key");
```

- Удаление излишних преобразований типов. Текст программы должен быть более читаемый и содержать меньше ошибок.

```
Map m = new HashMap();
m.put("key", 1);
String s = (String) m.get("key");
```

- Код должен быть более эффективным по времени выполнения.

# Родовые типы языка Java.

## 2. Пример использования родовых типов

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

Романов Владимир Юрьевич ©2025

- Объявление родового типа-интерфейса **Мар**

```
public interface Map<K, V> {  
    public void put(K key, V value);  
    public V get(K key);  
}
```

- Использование родового типа-интерфейса **Мар**

```
Map<String, String> m = new HashMap<String, String>();  
m.put("key", "blarg");  
m.put("key", 1); // Ошибка компиляции.  
String s = m.get("key");
```

# Родовые типы языка Java.

## 3. Не ковариантность родовых типов

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

Романов Владимир Юрьевич ©2025

- Ковариантность массивов языка Java

```
Integer[] intArray = new Integer[10];  
Number[] numberArray = intArray;
```

- Не ковариантность родовых типов

```
List<Integer> intList = new ArrayList<Integer>();  
List<Number> numberList = intList; // Ошибка компиляции
```

- Number есть супертип для Integer.

Но List<Number> не есть супертип List<Integer>  
Object супертип для List<Integer>

# Родовые типы языка Java.

## 4. Wildcard (Неизвестный тип)

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

Романов Владимир Юрьевич ©2025

- Метод *printList* не работает с типом *List<String>*

```
void printList(List I) {  
    for (Object o : I)  
        System.out.println(o);  
}
```

- Метод *printList* работает только с типом *List<Object>*

```
void printList(List<Object> I) {  
    for (Object o : I)  
        System.out.println(o);  
}
```

- Метод *printList* работает со списком элементов любого типа: *List<Object>*, *List<Integer>*, ...

```
void printList(List<?> I) {  
    for (Object o : I)  
        System.out.println(o);  
}
```

# Родовые типы языка Java.

## 5. Родовые методы

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

Романов Владимир Юрьевич ©2025

- Родовые методы не обязательно должны быть у родовых типов.
- Родовые методы также имеют **placeholders**.

```
public <T> T ifThenElse(boolean b, T first, T second) {  
    return b ? first : second;  
}
```

- Пример: метод ifThenElse работает со всем типами, если у 2-го и 3-го параметров типы одинаковые.

```
String s = ifThenElse(b, "a", "b");  
Integer i = ifThenElse(b, new Integer(1), new Integer(2));
```

- Родовые методы приемлемы для статических методов, когда не используются типы заданные для родового класса.
- Когда тип родового класса применяется только к методу. В этом случае упрощается сигнатура родового класса.

# Родовые типы языка Java.

## 6. Ограниченные типы. Ограничение сверху

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

Романов Владимир Юрьевич ©2025

- Параметр типа **V** не ограничен

```
public class Matrix<V> { ... }
```

- Параметр типа **V** ограничен типом **Number**

```
public class Matrix<V extends Number> { ... }
```

Ограничение сверху.  
**V** класс **Number**  
и его потомки

# Родовые типы языка Java.

## 6. Ограниченные типы. Ограничение снизу

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

Романов Владимир Юрьевич ©2025

- Параметр типа **V** ограничен типом **Number**

```
public class A<V super Number> { ... }
```

Ограничение снизу.  
**V** класс **Number**  
и его предки

# Java 8.

Владимир Юрьевич Романов  
Московский Государственный Университет им. М.В.Ломоносова  
Факультет Вычислительной Математики и Кибернетики  
vromanov@cs.msu.su,  
vladimir.romanov@gmail.com  
<http://master.cmc.msu.ru>

# Java 8. Lambda - выражения

Владимир Юрьевич Романов  
Московский Государственный Университет им. М.В.Ломоносова  
Факультет Вычислительной Математики и Кибернетики  
vromanov@cs.msu.su,  
vladimir.romanov@gmail.com  
<http://master.cmc.msu.ru>

# Назначение Lambda-выражений

- Трактовать блоки кода как объекты (данные)
- Это не только полезно, но и необходимо
- Lambda ( $\lambda$ )-выражение – это просто блок кода с параметрами, которые необходимо передать в блок (анонимная функция).

# Обычные функции

---

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

Романов Владимир Юрьевич ©2025

- Могут получать объекты
- Могут создавать объекты
- Могут возвращать объекты

# Функции высшего порядка (появились в Java 8)

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

Романов Владимир Юрьевич ©2025

- Могут получать функции
- Могут создавать функции
- Могут возвращать функции

## Пример: Lambda – выражение вместо анонимного класса

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

Романов Владимир Юрьевич ©2025

```
public class RunnableTest {  
  
    public static void main(String[] a) {  
  
        // Anonymous Runnable  
        Runnable r = new Runnable() {  
            @Override  
            public void run(){  
                System.out.println("Hello");  
            }  
        };  
        r.run();  
    }  
}
```

```
public class RunnableTest {  
  
    public static void main(String[] a){  
  
        // Lambda Runnable  
        Runnable r = () ->  
            System.out.println("Hello");  
  
        r.run();  
    }  
}
```

```
Runnable r = new Runnable
{
    @Override
    public void run() {
        System.out.println("Hello");
    }
}
```

# Пример: Lambda – выражение вместо анонимного класса

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

Романов Владимир Юрьевич ©2025

```
final String name = "John";
Button button = new Button();

button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        System.out.println("hi " + name);
    }
});
```

```
final String name = "John";
Button button = new Button();

button.addActionListener( event -> System.out.println("hi " + name) );
```

# Пример: Lambda – выражение в операторе сравнения при сортировке

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

Романов Владимир Юрьевич ©2025

```
List<Person> personList = Person.createShortList();  
  
Collections.sort(personList,  
    new Comparator<Person>(){  
        public int compare(Person p1, Person p2){  
            return p1.getName().compareTo(p2.getName());  
        }  
    }  
);
```

```
List<Person> personList = Person.createShortList();  
  
Collections.sort(personList,  
    (Person p1, Person p2) ->  
        p1.getSurName().compareTo(p2.getSurName()));  
);
```

# Пример: Lambda – выражение в операторе сравнения при сортировке

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

Романов Владимир Юрьевич ©2025

```
List<Person> personList = Person.createShortList();  
  
Collections.sort(personList,  
    new Comparator<Person>(){  
        public int compare(Person p1, Person p2){  
            return p1.getName().compareTo(p2.getName());  
        }  
    }  
);
```

```
List<Person> personList = Person.createShortList();  
  
Collections.sort(personList,  
    (p1, p2) -> p1.getSurName().compareTo(p2.getSurName()));  
);
```

# Пример: Lambda – выражение при работе с коллекциями

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

Романов Владимир Юрьевич ©2025

```
List<String> features = Arrays.asList("A", "B", "C", "D");  
  
// Внешний итератор.  
for (String feature : features)  
{ System.out.println(feature); }
```

```
List<String> features = Arrays.asList("A", "B", "C", "D");  
  
// Внутренний итератор.  
features.forEach(n -> System.out.println(n));  
  
// Ссылка на метод.  
features.forEach(System.out::println);
```

# Назначение Lambda-выражений

---

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

Романов Владимир Юрьевич ©2025

- Лямбда – выражение **SAM**-типа:  
(**S**ingle **A**bstract **M**ethod) – интерфейс с  
**единственным** *абстрактным* методом
- Другое название **SAM**-типа –  
*функциональный интерфейс*
- В Java8 могут быть интерфейсы с **НЕ**  
абстрактными методами (*default* – методы)

# Функциональный интерфейс. Пример

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

Романов Владимир Юрьевич ©2025

```
package java.util.function;

@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);

    default Consumer<T> andThen(Consumer<? super T> after) {
        // ...
    }
}
```

- **@FunctionalInterface** – аннотация для синтаксического контроля: только один абстрактный метод.
- **default** – может быть неограниченное число умалчиваемых методов

# Синтаксис Lambda-выражений

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

Романов Владимир Юрьевич ©2025

Параметры

Стрелка

Тело

(int x, int y) -> x + y

() -> 100

(String s) -> {  
System.out.println(s);  
}

# Синтаксис Lambda-выражений

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

Романов Владимир Юрьевич ©2025

Параметры

Стрелка

Тело

(x, y)

->

x + y

()

->

100

s

-> { System.out.println(s) ; }

# Функциональные интерфейсы. java.util.function

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

Романов Владимир Юрьевич ©2025

| Имя интерфейса    | Аргумент | Возвращает |
|-------------------|----------|------------|
| Predicate<T>      | T        | Boolean    |
| Consumer<T>       | T        | Void       |
| Function<T,R>     | T        | R          |
| Supplier<T>       | Нет      | T          |
| UnaryOperator<T>  | T        | T          |
| BinaryOperator<T> | (T, T)   | T          |

# Функциональный интерфейс. Примеры из пакета `java.util.function`

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

Романов Владимир Юрьевич ©2025

- `Consumer<T>` – выполняет действия на типом Т.  
Результат не возвращается
- `Supplier<T>` – возвращает результат типа Т.  
Ввод не требуется
- `Predicate<T>` – получает экземпляр типа Т как параметр. Возвращает логическое значение
- `Function<T, R>` – получает экземпляр типа Т как параметр. Возвращает результат типа R

# Пример: Lambda – выражение для функционального программирования

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

Романов Владимир Юрьевич ©2025

```
package java.util.function;

interface Predicate<T> {
    boolean test(T t);
    //...
}
```

```
// Функциональный интерфейс
interface Predicate<Person> {
    boolean test(Person p);
}
```

```
// Функциональный интерфейс
interface CheckPerson {
    boolean test(Person p);
}
```

# Пример: Lambda – выражение для функционального программирования

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

Романов Владимир Юрьевич ©2025

```
// Объявление
public static
void printWithPredicate( List<Person> roster, Predicate<Person> tester) {
    for (Person p : roster) {
        if (tester.test(p)) {
            p.printPerson();
        }
    }
}
```

```
// Вызов
printWithPredicate( roster,
    p -> p.getGender() == Person.Sex.MALE &&
        p.getAge() >= 18 &&
        p.getAge() <= 25
);
```

# Умалчивающие методы интерфейсов. Новые методы для коллекций

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

Романов Владимир Юрьевич ©2025

```
interface Collection {  
    //...  
  
    default void forEach (Consumer<? super T> action) {  
  
        for (T t : this)  
            action.accept(t);  
  
    }  
}
```

```
@FunctionalInterface  
interface Consumer<T> {  
    //...  
    void accept(T t);  
}
```

# Умалчивающие методы интерфейсов. Новые методы для коллекций

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

Романов Владимир Юрьевич ©2025

```
List<String> features = Arrays.asList("A", "B", "C", "D");
```

```
// Внутренний итератор.  
features.forEach (n -> System.out.println(n));
```

```
// Внутренний итератор со ссылкой на метод.  
features.forEach (System.out::println);
```

# Java 8. Stream - потоки

Владимир Юрьевич Романов  
Московский Государственный Университет им. М.В.Ломоносова  
Факультет Вычислительной Математики и Кибернетики  
vromanov@cs.msu.su,  
vladimir.romanov@gmail.com  
<http://master.cmc.msu.ru>

# Определение потоков

---

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

Романов Владимир Юрьевич ©2025

**Последовательность элементов** из **источника** которые поддерживают **составные операции**

**Последовательность элементов:**

поток предоставляет интерфейс для упорядоченного множества значений для заданного типа элементов.

Однако потоки не хранят значения, а вычисляют их по требованию.

**Источник:**

потоки потребляют информацию из источников предоставляющих данные: коллекций, массивов, ресурсов ввода и вывода.

**Составные операции:**

Потоки поддерживают операции из функциональных языков программирования: **filter, map, reduce, find, match, sorted, ...**

# Отличие операций потоков от операций коллекций

---

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

Романов Владимир Юрьевич ©2025

## **Конвейер (Pipelining):**

Многие операции потока возвращают сами потоки.  
Это позволяет объединять операции в форме большого конвейера.

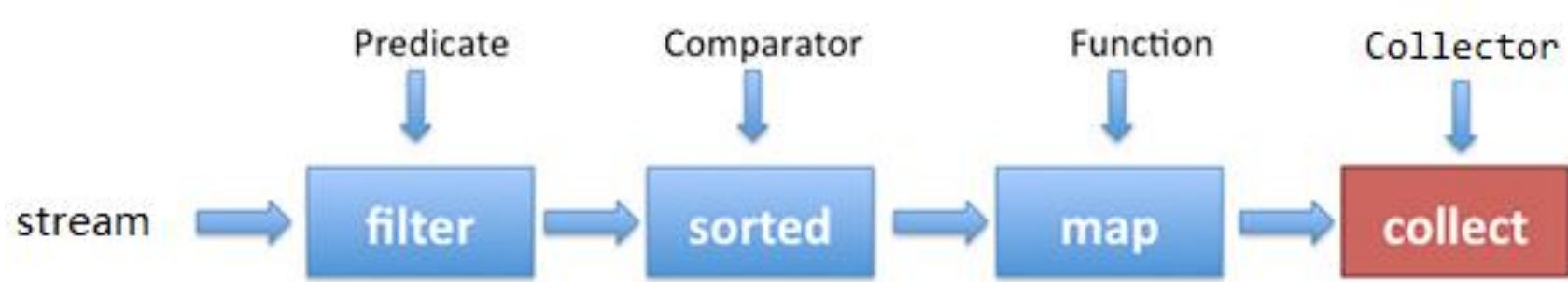
## **Внутренние итерации:**

В отличие от коллекций, которые проходят явно (*внешняя итерация*), в потоковых операциях итераций скрыта.

# Обработка данных в потоке

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

Романов Владимир Юрьевич ©2025



# Интерфейс Stream. Преобразования конвейера

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

Романов Владимир Юрьевич ©2025

```
public interface Stream<T> extends BaseStream<T, Stream<T>>
{
    static <T> Stream<T> of(T... values);

    Stream<T> filter(Predicate<? super T> predicate);
    Stream<T> distinct();
    Stream<T> sorted();
    Stream<T> sorted(Comparator<? super T> comparator);
    Stream<T> peek(Consumer<? super T> action);
    <R> Stream<R> map(Function<? super T,? extends R> mapper);
    IntStream mapToInt(ToIntFunction<? super T> mapper);

    //...
}
```

# Интерфейс Stream. Окончание конвейера

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

Романов Владимир Юрьевич ©2025

```
public interface Stream<T> extends BaseStream<T, Stream<T>>
{
    //...

    void forEach(Consumer<? super T> action);
    Object[] toArray();
    Optional<T> min(Comparator<? super T> comparator);
    long count();
    boolean allMatch(Predicate<? super T> predicate);
    boolean anyMatch(Predicate<? super T> predicate);
    <R,A> R collect(Collector<? super T,A,R> collector);
}
```

# Интерфейс BaseStream

```
interface BaseStream<T,S extends BaseStream<T,S>> {  
    boolean isParallel();  
    Iterator<T> iterator();  
    S parallel();  
    S sequential();  
    S unordered();  
    S onClose(Runnable closeHandler);  
}
```

# Java8. Потоки

## Порождение потоков из коллекций

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

Романов Владимир Юрьевич ©2025

```
interface Collection {  
    //...  
  
    default Stream<E> stream();  
  
    //...  
}
```

```
List<String> collection = Arrays.asList("A", "B", "C", "D");  
  
// sequential version  
Stream stream = collection.stream();  
  
// parallel version  
Stream parallelStream = collection.parallelStream();
```

# Java8. Потоки

## Порождение потоков

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

Романов Владимир Юрьевич ©2025

```
Stream<Integer> numbersFromValues = Stream.of (1, 2, 3, 4);
```

```
int[] numbers = {1, 2, 3, 4};
```

```
IntStream numbersFromArray = Arrays.stream(numbers);
```

```
List<String> list = Arrays.asList("Monkey", "Lion", "Giraffe");
```

```
Stream<String> streamFromList = list.stream();
```

```
Set<String> set = new HashSet<>(list);
```

```
Stream<String> streamFromSet = set.stream();
```

```
Stream<String> lines =
```

```
    Files.lines(Paths.get("File.txt"), Charset.defaultCharset());
```

```
long numberOfLines =
```

```
    Files
```

```
        .lines(Paths.get("File.txt"), Charset.defaultCharset())
```

```
        .count();
```

# Java8. Потоки

## Порождение потоков

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

Романов Владимир Юрьевич ©2025

```
Stream<Integer> numbers = Stream.iterate (0, n -> n + 10);  
numbers  
    .limit(5)  
    .forEach( System.out::println ); // 0, 10, 20, 30, 40
```

```
IntStream oneTwoThree = IntStream.of(1, 2, 3);
```

```
IntStream positiveSingleDigits = IntStream.rangeClosed(1, 9);
```

```
IntStream powersOfTwo = IntStream.iterate(1, i -> i * 2);
```

```
IntStream randomInts = new Random().ints();
```

```
IntStream chars = "ABC".chars();
```

# Пример: Обработка списка из строк

```
List<String> collection = Arrays.asList(  
    "alfred", "Adam", "Alex", "David", "Catrin", "Anita", "ann" );
```

```
List<String> result = collection .stream()  
    .filter( e -> e.startsWith("A") || e.startsWith("a") )  
    .sorted( String.CASE_INSENSITIVE_ORDER )  
    .peek( System.out::println )  
    .sorted()  
    .map(e -> "Name is: " + e)  
    .collect( Collectors.toList() );  
  
System.out.println(result);
```

# Пример: Обработка списка из строк

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

Романов Владимир Юрьевич ©2025

```
Adam
Alex
alfred
Anita
ann
```

```
[Name is: Adam, Name is: Alex, Name is: Anita, Name is: alfred, Name is: ann]
```

# Пример: редукция

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

Романов Владимир Юрьевич ©2025

```
int sum = Stream.of(1, 2, 3)
    .reduce (0, (acc, element) -> acc + element);
```

T **reduce**(T identity, BinaryOperator<T> accumulator);

Optional<T> **reduce**(BinaryOperator<T> accumulator)

# Пример: Сборка

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

Романов Владимира Юрьевич ©2025

```
List<String> collected = Stream.of("a", "b", "c")
    .collect( Collectors.toList() );
```

```
<R, A> R collect(Collector<? super T, A, R> collector);
```

## **interface Collector <T, A, R>**

**T** – Тип входных параметров для операции редукции.

**A** – Тип для аккумуляции операции редукции

**R** – Результирующий тип операции редукции

# Пример: Подсчет количества фигур противника атакующих заданную фигуру

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

Романов Владимир Юрьевич ©2025

```
public long enemyAttacks(Piece piece) {  
    Board board = piece.square.getBoard();  
  
    PieceColor myColor = piece.getColor();  
    PieceColor enemyColor = Board.getOpponentColor(myColor);  
  
    List<Square> squares = board.getSquares();  
    long n = squares  
        .stream()  
        .filter(s -> !s.isEmpty())  
        .map(s -> s.getPiece())  
        .filter(p -> p.getColor() == enemyColor)  
        .filter(p -> attacksSquare(p, square))  
        .count();  
  
    return n;  
}
```

# Пример: Лямбда выражение при сравнении ходов. Выбор ходов с наибольшим шагом

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

Романов Владимир Юрьевич ©2025

```
/**  
 * Приоритет у хода делающего больший шаг к цели -  
 * противоположному углу доски.  
 */  
Comparator<ITransferMove> maxStep =  
    (move1, move2) -> {  
        // Направление шага.  
        int dir = move1.getPiece().isWhite() ? 1 : -1;  
  
        int step1 = move1.getSource().shift( move1.getTarget() );  
        int step2 = move2.getSource().shift( move2.getTarget() );  
  
        return dir * (step2 - step1);  
   };
```

# Пример: Лямбда выражение при сравнении ходов. Выбор ходов с удаленной от цели клетки

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

Романов Владимир Юрьевич ©2025

```
/**  
 * Приоритет у хода с более дальней от цели клетки  
 * (ход отстающими фигурами).  
 */  
Comparator<ITransferMove> fromBack =  
    (move1, move2) -> {  
        Square goal = Halma.getPieceGoal(move1.getPiece());  
  
        // Расстояние до клетки - цели.  
        int distance1 = goal.distance(move1.getSource());  
        int distance2 = goal.distance(move2.getSource());  
  
        return Math.abs(distance2) - Math.abs(distance1);  
   };
```

# Пример: Использование умалчивающих методов интерфейса **Comparator**

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

Романов Владимир Юрьевич ©2025

```
protected Comparator <ITransferMove> getComparator() {  
    return maxStep.thenComparing(fromBack);  
}
```

Использование  
умалчивающего метода  
**thenComparing** для  
задания порядка  
сравнения

# Пример: Лямбда выражение при сравнении ходов. Приоритет у хода с большим весом

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

Романов Владимир Юрьевич ©2025

```
/**  
 * Приоритет у хода с большим весом  
 */  
final Comparator<? super Move> brain =  
    (m1, m2) -> getMoveWeight (m2) - getMoveWeight (m1);  
};
```

```
private int getMoveWeight (Move move) {  
    IPutMove putMove = (IPutMove) move;  
  
    Square target = putMove.getTarget();  
  
    if (isCorner(target))  
        return 1000; // Встали в угол.  
  
    if (isBorder(target))  
        return 900; // Встали на край доски.
```

# Пример: Применение сортировки для выбора лучшего хода

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

Романов Владимир Юрьевич ©2025

```
List<Move> correctMoves = getCorrectMoves(board, color);

// Случайным образом переставим ходы
// чтобы игра не повторялась.
Collections.shuffle(correctMoves);

// Ищем лучший ход
correctMoves.sort( brain );

// Выбираем лучший ход
Move bestMove = correctMoves.get(0);
```

# Аннотации в языке Java

# Аннотации

- *Аннотации* предоставляют информацию о программе и хранится с исполняемым кодом программы
- Не влияют на выполнение программы
- Виды аннотации
  - Информация для компилятора - может использоваться для обнаружения ошибок и подавления предупреждений
  - Для обработки во время компиляции и сопровождения – например для генерации XML-файлов
  - Для обработки во время выполнения программы

# Объявление типа аннотации - информация об авторе

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

Романов Владимир Юрьевич ©2025

```
import java.lang.annotation.*; // импорт @Documented

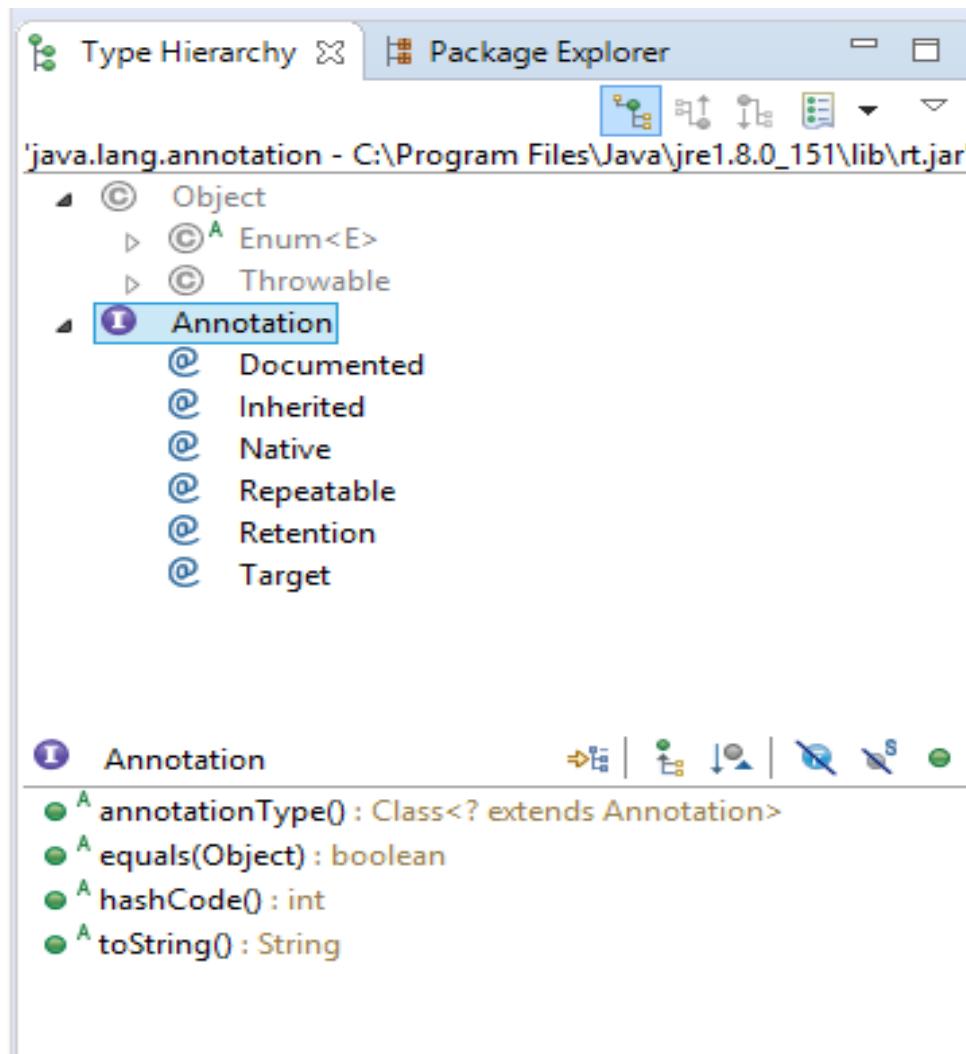
/**
 * Аннотация об авторе будет хранится
 * в оттранслированном классе (байткоде).
 */
@Documented
@interface Doc {
    String author = "";
    int version = 1;
}
```

Объявление

# Базовый интерфейс Annotation и его потомки

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

Романов Владимир Юрьевич ©2025



# Использование аннотации - информации об авторе

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

Романов Владимир Юрьевич ©2025

```
@Doc ( —
author = "John Doe",
version = 2
)
public class DocumentedClass {

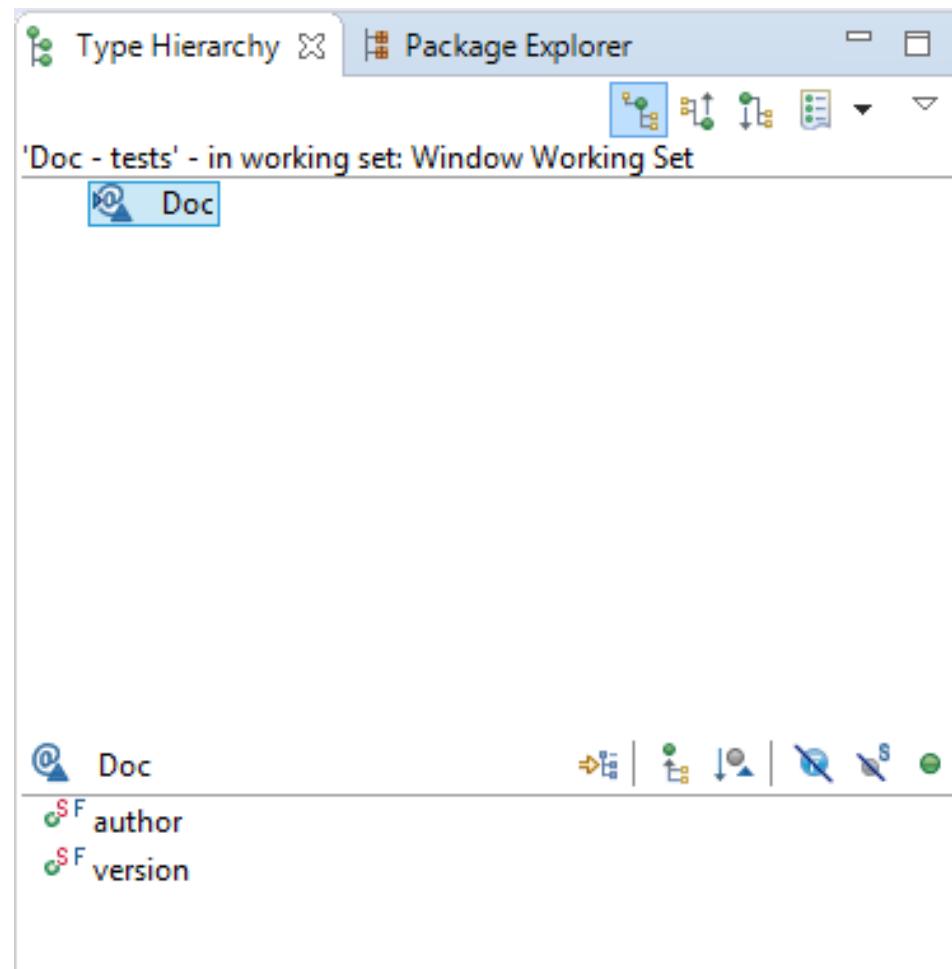
// ...
}
```

Использование  
аннотации

# Аннотация @Doc

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

Романов Владимир Юрьевич ©2025



# Объявление аннотации **Documented**

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

Романов Владимир Юрьевич ©2025

```
@Documented
@Retention(value=RUNTIME)
@Target(value=ANNOTATION_TYPE)
public @interface Documented {  
}
```

# Аннотация **Retention**

---

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

Романов Владимир Юрьевич ©2025

@Retention

Как хранится аннотация

RetentionPolicy.SOURCE

RetentionPolicy.CLASS

RetentionPolicy.RUNTIME

# Аннотация `@Documented`

---

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

Романов Владимир Юрьевич ©2025

**@Documented**

Будет ли использоваться инструментом  
**JavaDoc**

# Аннотация `@Target`

---

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

Романов Владимир Юрьевич ©2025

## `@Target`

К какому элементу программы на Java может применяться

`ElementType.ANNOTATION_TYPE`

`ElementType.CONSTRUCTOR`

`ElementType.FIELD`

`ElementType.LOCAL_VARIABLE`

`ElementType.METHOD`

`ElementType.PACKAGE`

`ElementType.PARAMETER`

`ElementType.TYPE`

# Аннотация **@Inherited**

---

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

Романов Владимир Юрьевич ©2025

## **@Inherited**

Может ли аннотация унаследована от  
суперкласса

# Аннотация **@Repeatable**

---

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

Романов Владимир Юрьевич ©2025

## **@ Repeatable**

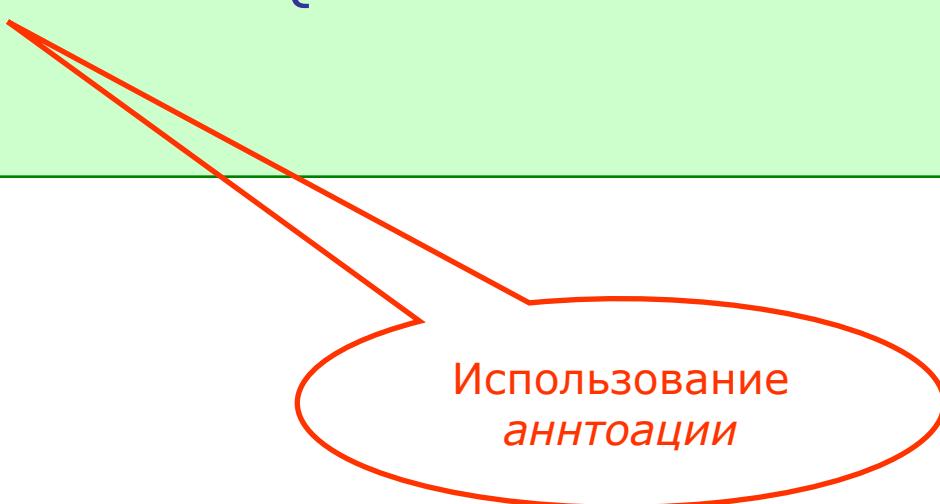
Может ли аннотация применяться многократно

# Объявление аннотации *Documented*

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

Романов Владимир Юрьевич ©2025

```
@Documented  
@Retention(value=RUNTIME)  
@Target(value=ANNOTATION_TYPE) public  
@interface Documented {  
}
```



Использование  
аннотации

# Аннотации для компилятора.

## Устаревшие методы

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

Романов Владимир Юрьевич ©2025

```
public class OldClass {  
    /**  
     * @deprecated  
     * Этот метод был написан 20 лет назад  
     */  
    @Deprecated  
    static void deprecatedMethod() {}  
}
```

Устаревший  
метод

```
class User {  
    public void main(String[] args) {  
        OldClass.deprecatedMethod();  
    }  
}
```

Предупреждение  
О старом методе

# Аннотации для компилятора. Подавление лишней диагностики

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

Романов Владимир Юрьевич ©2025

```
class User {  
    @SuppressWarnings("deprecation")  
    public void main(String[] args) {  
        OldClass.deprecatedMethod();  
    }  
}
```

Подавление  
диагностики-  
предупреждения

# Аннотации для компилятора.

## Перекрытие метода

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

Романов Владимир Юрьевич ©2025

```
@Override  
int overriddenMethod() {  
}
```

Метод  
перекрывает  
метод  
суперкласса

# Аннотации для компилятора. Проверка указателей на *null*

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

Романов Владимир Юрьевич ©2025

```
@NotNull String str;
```



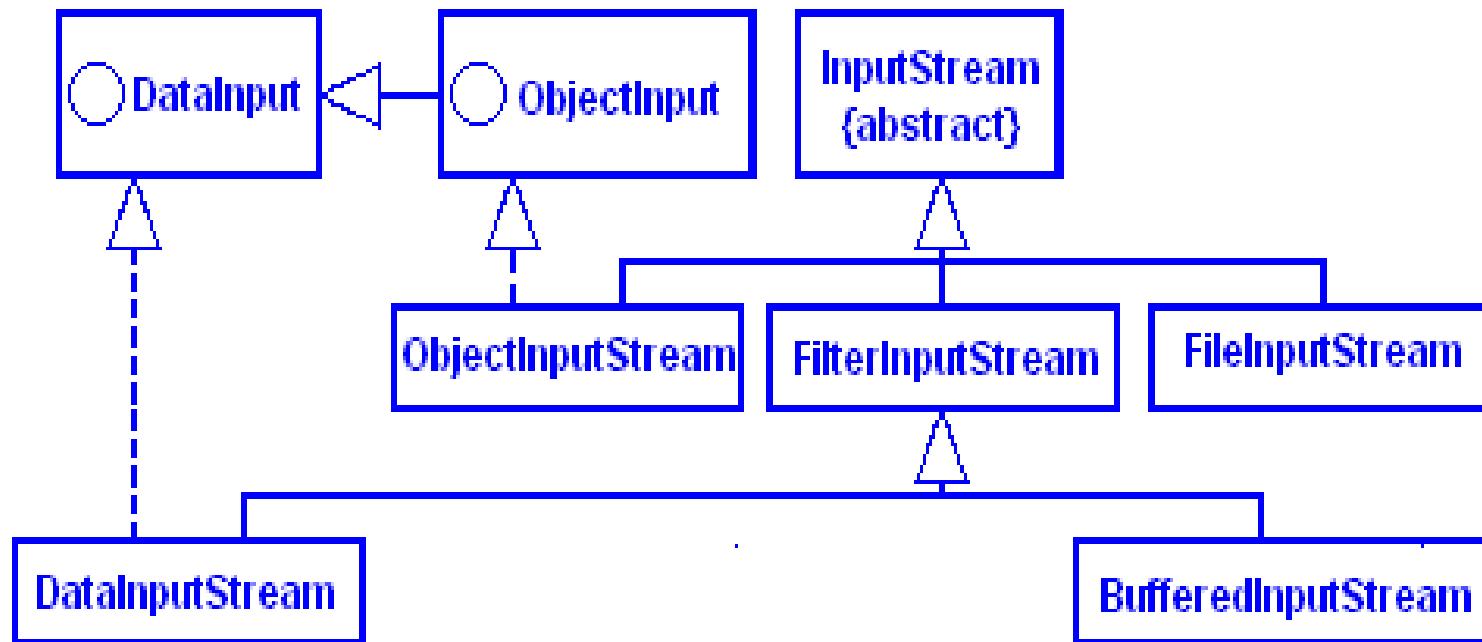
Переменной str  
нельзя  
присваивать  
значение **null**

# Программирование ввода-вывода в языке Java

# Потоки ввода/вывода языка Java. Классы и интерфейсы потоков ввода

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

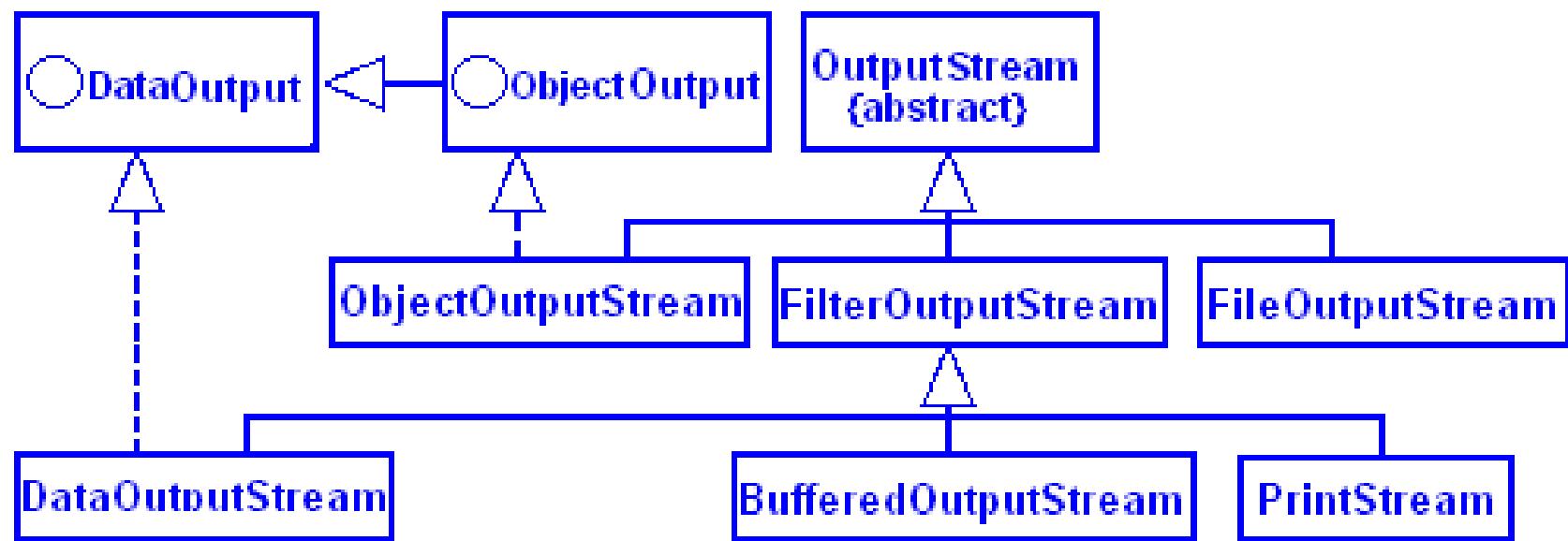
Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java. Классы и интерфейсы потоков вывода

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

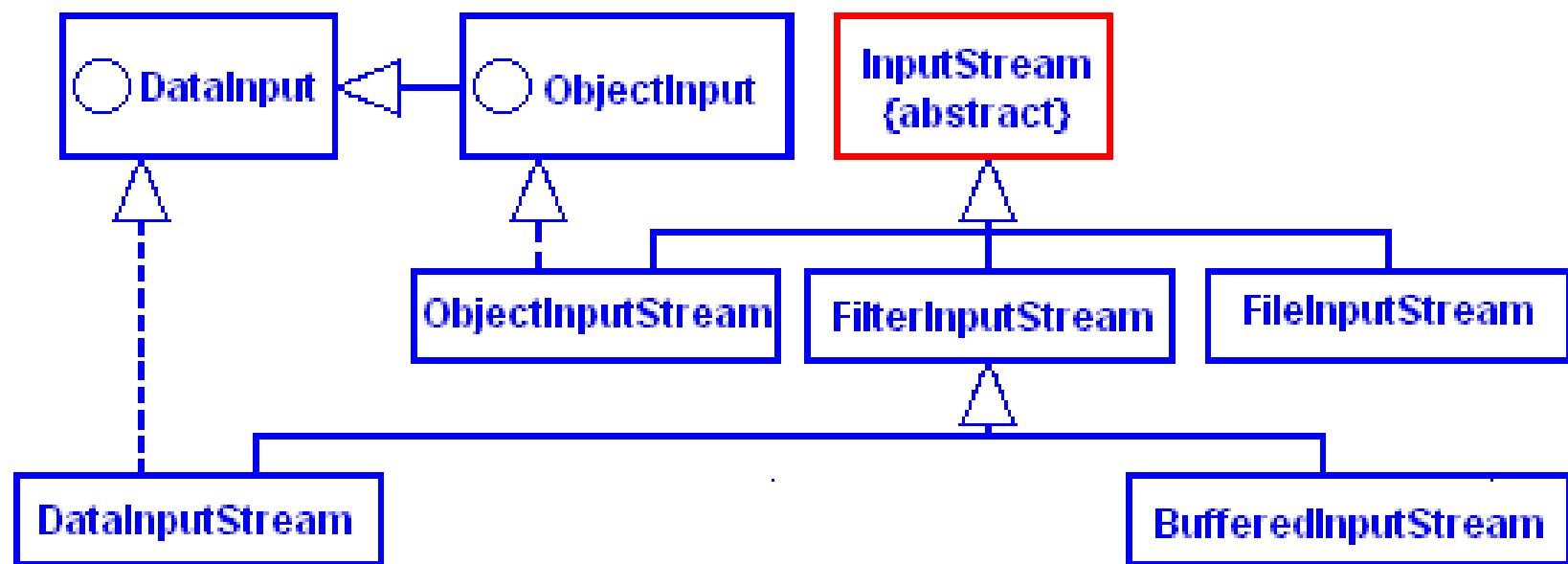
Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java. Класс InputStream.

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

Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java.

## Класс *InputStream*. Абстрактный метод *read*.

---

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

Романов Владимир Юрьевич ©2025

- Метод *read()* – должен быть переопределен в классах-потомках
- Считывает 1 байт
- Возвращает байт как целое число
- Если ввод закончен, то возвращает -1
- Если при вводе ошибка, то порождает ситуацию *IOException*
- *IOException* – потомок класса *Exception*

# Потоки ввода/вывода языка Java.

## Класс *InputStream*. Методы чтения.

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

Романов Владимир Юрьевич ©2025

- Реализуются с помощью абстрактного метода *read()*
- `int read(byte[] b)`  
Чтение нескольких байтов в буферный массив *b*.
- `int read(byte[] b, int off, int len)`  
Чтение нескольких байтов в буферный массив начиная с позиции в массиве *off*.
- `skip(long n)`  
Пропустить *n* байтов из потока ввода.

# Потоки ввода/вывода языка Java.

## Класс *InputStream*. Маркировка потока ввода

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

Романов Владимир Юрьевич ©2025

- *boolean markSupported()*  
Это свойство экземпляра класса. Поддерживает ли этот экземпляр класса методы *mark* и *reset*.
- *mark(int readlimit)*  
Помечает текущую позицию потока ввода.
- *void reset()*  
Возвращает позицию ввода в позицию заданную методом *mark*.

# Потоки ввода/вывода языка Java.

## Класс *InputStream*. Другие методы

---

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

Романов Владимир Юрьевич ©2025

- *int available()*  
Возвращает количество байтов которое может быть считано или пропущено.
- *void close()*  
Закрывает поток ввода и освобождает связанные ресурсы.

# Потоки ввода/вывода языка Java.

## Класс *ByteArrayInputStream*. Назначение и методы

- Является потомком класса *InputStream*
- Используется для чтения из байтового массива.
- *ByteArrayInputStream(byte[] buf)*  
*ByteArrayInputStream(byte[] buf, int offset, int length)*  
Конструкторы байтового потока
- *protected byte[] buf*  
Массив байтов предоставленных при создании потока.
- *protected int count*  
Номер следующего за допустимым символа.
- *protected int mark*  
Помеченная позиция.
- *protected int pos*  
Индекс следующего для чтения символа.

# Потоки ввода/вывода языка Java. Класс *ByteArrayInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
byte[] bytes = {1, -1, 0};

ByteArrayInputStream in = new ByteArrayInputStream(bytes);

int readedInt = in.read(); // readedInt = 1

System.out.println("first element read is: " + readedInt);

readedInt = in.read();
// readedInt = 255.
// Однако (byte) readedInt даст значение -1

System.out.println("second element read is: " + readedInt);

readedInt = in.read(); // readedInt = 0

System.out.println("third element read is: " + readedInt);
```

# Потоки ввода/вывода языка Java.

## Класс *StringInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *InputStream*
- Используется для чтения из строки.
- *StringInputStream(String s)*  
Конструкторы строкового потока

# Потоки ввода/вывода языка Java.

## Класс *SequenceInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *InputStream*
- Используется для чтения из нескольких потоков.
- *SequenceInputStream(Enumeration e)*  
*SequenceInputStream(InputStream s1, InputStream s2)*  
Конструкторы последовательности потоков

# Потоки ввода/вывода языка Java. Класс *SequenceInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
FileInputStream inFile1 = null;
FileInputStream inFile2 = null;
SequenceInputStream sequenceStream = null;
try {
    inFile1 = new FileInputStream("file1.txt");
    inFile2 = new FileInputStream("file2.txt");
    sequenceStream = new
        SequenceInputStream(inFile1, inFile2);

    int readedByte = sequenceStream.read();

    while(readedByte != -1) {
        readedByte = sequenceStream.read();
    }
}
```

# Потоки ввода/вывода языка Java. Класс *SequenceInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
}

catch (IOException e) {
    System.out.println("IOException: " +
                        e.toString());
}

finally {
    try{ sequenceStream.close(); }
    catch(IOException e){};
}
```

# Потоки ввода/вывода языка Java.

## Класс *FileInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *InputStream*
- Используется для чтения из файла.
- *FileInputStream(String fileName) throws FileNotFoundException*  
Конструкторы файлового потока.

# Потоки ввода/вывода языка Java.

## Класс *FileInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
byte[] bytesReaded = new byte[10];
String fileName = "d:\\test.txt";

try {
    FileInputStream inFile = new FileInputStream(fileName);
    System.out.println("Файл открыт для чтения");

    // Узнать, сколько байт готово к считыванию
    int bytesAvailable = inFile.available();
    System.out.println("Готово к считыванию: " + bytesAvailable +
                       " байт");

    // Считать в массив
    int count = inFile.read(bytesReaded, 0, bytesAvailable);
    System.out.println("Считано: " + count + " байт");

    for (I = 0; I < count; i++)
        System.out.print(bytesReaded[i] + ", ");
}
```

# Потоки ввода/вывода языка Java.

## Класс *FileInputStream*. Завершение примера

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

Романов Владимир Юрьевич ©2025

```
System.out.println();
inFile.close();

System.out.println("Входной поток закрыт");

}

catch (FileNotFoundException e) {
    System.out.println("Невозможно произвести запись в файл: " +
                       fileName);
}

catch (IOException e) {
    System.out.println("Ошибка ввода/вывода: " + e.toString());
}
```

# Потоки ввода/вывода языка Java.

## Класс *FilterInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *InputStream*
- Используется как класс-адаптер для чтения из файла.
- *protected FilterInputStream(InputStream is)*  
Конструкторы фильтрованного потока.
- Предок для классов:  
*BufferedInputStream*  
*LineNumberInputStream*  
*PushBackInputStream*  
*DataInputStream*

# Потоки ввода/вывода языка Java.

## Класс *BufferedInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *FilterInputStream*
- Используется для буферизации данных перед их чтением из надстраиваемого потока данных.
- *BufferedInputStream(InputStream in)*  
*BufferedInputStream(InputStream in, int size)*  
Конструкторы буферизованного потока.
- Реализуют маркировку потока ввода

# Потоки ввода/вывода языка Java. Класс *BufferedInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
// Определить время считывания без буферизации
timeStart = System.currentTimeMillis();
inStream = new FileInputStream(fileName);

while(inStream.read() != -1) {}

time = System.currentTimeMillis() - timeStart;
inStream.close();

System.out.println("Direct read time: " + (time) + "ms");

// Определить время считывания с буферизацией
timeStart = System.currentTimeMillis();

    inStream = new FileInputStream(fileName);
inStream = new BufferedInputStream(inStream);

while(inStream.read() != -1)

time = System.currentTimeMillis() - timeStart;
```

# Потоки ввода/вывода языка Java.

## Класс *BufferedInputStream*. Продолжение примера

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

Романов Владимир Юрьевич ©2025

```
inStream.close();
System.out.println("Buffered read time: " + (time) + "ms");

}

catch (IOException e) {
System.out.println("IOException: " + e.toString());
e.printStackTrace();
}
```

# Потоки ввода/вывода языка Java.

## Класс *LineNumberInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *FilterInputStream*
- При чтении подсчитывает количество считанных строк.
- *LineNumberStream(InputStream in)*  
Конструктор потока.
- *int getLineNumber()*  
Возвращает количество считанных строк.
- *void setLineNumber(int lineNumber)*  
Выполняет переход на указанную строку

# Потоки ввода/вывода языка Java.

## Класс *PushBackInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

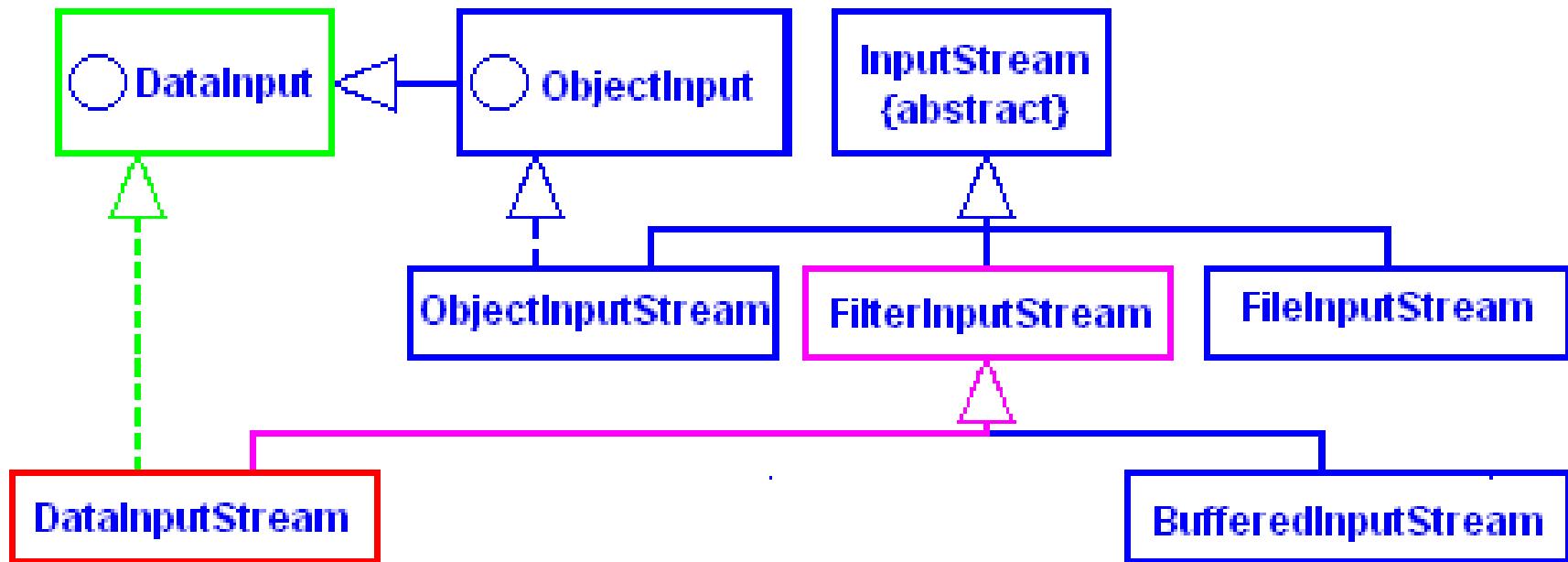
- Является потомком класса *FilterInputStream*
- Позволяет вернуть один символ в поток ввода с помощью функции *unread()*.
- Является буферизованным потоком ввода
- *PushbackInputStream(InputStream in)*  
*PushbackInputStream(InputStream in, int size)*  
Конструкторы потока ввода
- *void unread(byte[] b)*  
*void unread(byte[] b, int off, int len)*  
*void unread(int b)*  
Функции записи в поток ввода.

# Потоки ввода/вывода языка Java.

## Класс *DataInputStream*. UML диаграмма

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

Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java.

## Класс *DataInputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *FilterInputStream*
- Используется для чтения не только байтов, но и примитивных типов данных языка Java
- Реализует интерфейс *DataInput*

# Потоки ввода/вывода языка Java.

## Интерфейс *DataInput*. Методы

---

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

Романов Владимир Юрьевич ©2025

- *boolean readBoolean()*
- *byte readByte()*
- *char readChar()*
- *double readDouble()*
- *float readFloat()*
- *void readFully(byte[] b)*
- *void readFully(byte[] b, int off, int len)*
- *int readInt()*
- *String readLine()*
- *long readLong()*
- *Short readShort()*
- *int readUnsignedByte()*
- *int readUnsignedShort()*
- *String readUTF()*
- *int skipBytes(int n)*

# Потоки ввода/вывода языка Java. Класс *DataInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
try {
    InputStream in = new FileInputStream("c:/test.txt");
    DataInputStream inData = new DataInputStream(in);

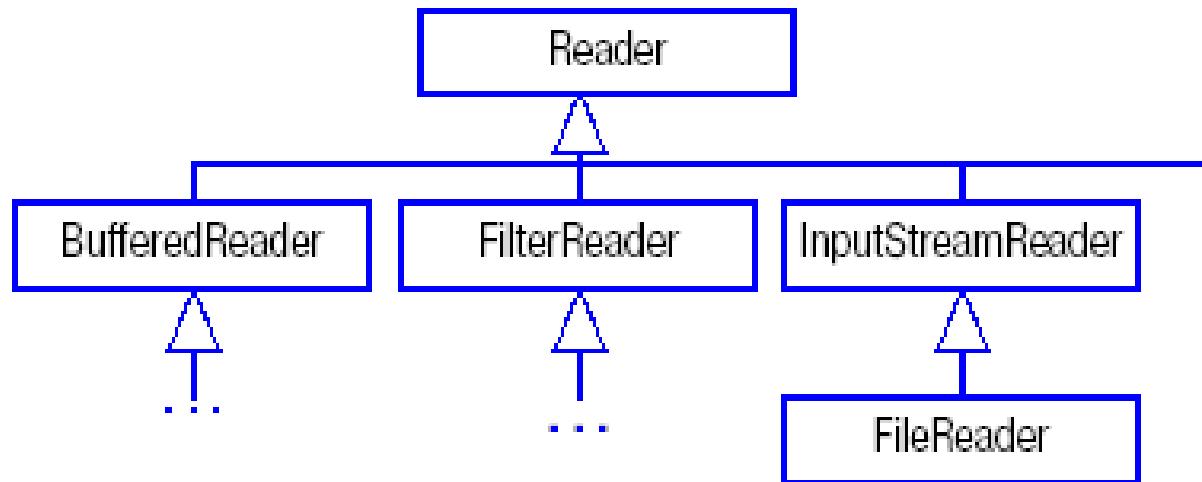
    System.out.println("readByte: " + inData.readByte());
    System.out.println("readInt: " + inData.readInt());
    System.out.println("readLong: " + inData.readLong());
    System.out.println("readDouble: " + inData.readDouble());

    inData.close();
}
catch (Exception e) {
    System.out.println("Impossible IOException occurs: " +
                       e.toString());
    e.printStackTrace();
}
```

# Потоки ввода/вывода языка Java. Класс – читатели. UML диаграмма

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

Романов Владимир Юрьевич ©2025



- Работают не с массивом байтов *byte*, а с массивом символов *char*
- Используются для чтения символов в кодировке UNICODE

# Потоки ввода/вывода языка Java.

## Таблица соответствия потоков ввода и читателей (1)

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

Романов Владимир Юрьевич ©2025

| <b>Байтовый поток</b>          | <b>Символьный поток</b>  |
|--------------------------------|--------------------------|
| <i>InputStream</i>             | <i>Reader</i>            |
| <i>ByteArrayInputStream</i>    | <i>CharArrayReader</i>   |
| ---                            | <i>InputStreamReader</i> |
| <i>FileInputStream</i>         | <i>FileReader</i>        |
| <i>FilterInputStream</i>       | <i>FilterReader</i>      |
| <i>BufferedInputStream</i>     | <i>BufferedReader</i>    |
| <i>DataInputStream</i>         | ---                      |
| <i>ObjectInputStream</i>       | ---                      |
| <i>PipedInputStream</i>        | <i>PipedReader</i>       |
| <i>StringBufferInputStream</i> | <i>StringReader</i>      |
| <i>LineNumberInputStream</i>   | <i>LineNumberReader</i>  |
| <i>PushBackInputStream</i>     | <i>PushBackReader</i>    |
| <i>SequenceInputStream</i>     | ---                      |

# Потоки ввода/вывода языка Java. Класс-читатели. Пример

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

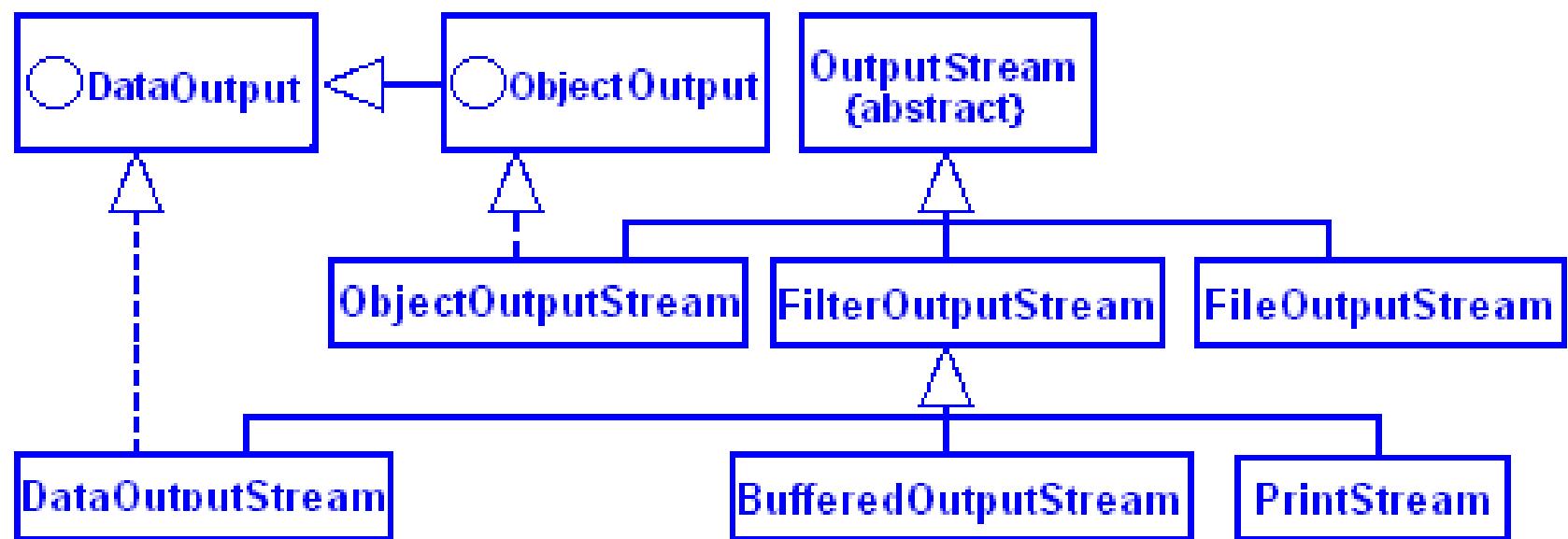
Романов Владимир Юрьевич ©2025

```
try {  
    // Считываем результат  
    FileReader fr = new FileReader(fileName);  
    BufferedReader br = new BufferedReader(fr);  
  
    String s = null;  
    int count = 0;  
    System.out.println("Read data from file: " + fileName);  
  
    // Считывать данные, отображая на экран  
    while( (s = br.readLine()) != null)  
        System.out.println("row " + (++count) + " read:" + s);  
  
    br.close();  
}  
catch(Exception e) { e.printStackTrace(); }
```

# Потоки ввода/вывода языка Java. Классы и интерфейсы потоков вывода

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

Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java.

## Класс *OutputStream*. Абстрактный метод *write*.

---

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

Романов Владимир Юрьевич ©2025

- Метод *write(int byte)* – должен быть переопределен в классах-потомках
- Записывает 1 байт в поток вывода, остальные 24 разряда целого числа игнорирует
- Если при выводе ошибка, то порождает ситуацию *IOException*
- *IOException* – потомок класса *Exception*

# Потоки ввода/вывода языка Java. Класс *OutputStream*. Методы записи.

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

Романов Владимир Юрьевич ©2025

- Реализуются с помощью абстрактного метода *write()*
- `void write(byte[] b)`  
Запись нескольких байтов из буферного массива *b*.
- `void write(byte[] b, int off, int len)`  
Запись нескольких байтов из буферного массива начиная с позиции в массиве *off*.

# Потоки ввода/вывода языка Java.

## Класс *OutputStream*. Другие методы класса

- ***flash()***

Стирает поток вывода и при буферизованном выводе и записывает информацию из буфера.

- ***void close()***

Закрывает поток вывода и освобождает связанные ресурсы.

# Потоки ввода/вывода языка Java.

## Класс *ByteArrayOutputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *OutputStream*
- Используется для записи в байтовый массив.
- *ByteArrayOutputStream()*  
*ByteArrayOutputStream(int bufferSize)*  
Конструкторы байтового потока
- *protected byte[] buf*  
Массив байтов для хранения данных потока  
предоставленных при создании потока.
- *protected int count*  
Номер следующего за допустимым символа.

# Потоки ввода/вывода языка Java.

## Класс *ByteArrayOutputStream*. Конвертирование данных потока

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

Романов Владимир Юрьевич ©2025

- **byte[] toByteArray()**

Создает новый массив и возвращает его как результат метода

- **String toString()**

Конвертирует содержимое буфера в строку, транслируя байты в символы в соответствии с умалчиваемой для платформы *кодировкой символов*

- **String toString(String enc)**

Конвертирует содержимое буфера в строку, транслируя байты в символы в соответствии с заданной параметром *enc* кодировкой символов

# Потоки ввода/вывода языка Java.

## Класс *ByteArrayOutputStream*. Методы

---

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

Романов Владимир Юрьевич ©2025

- **close()**  
Действие по закрытию потока в унаследованном этим классом методе эффекта не имеет.
- **void reset()**  
Обнуляет счетчик байтов и уничтожает накопленные выводимые в поток данные.
- **int size()**  
Возвращает текущий размер буфера.

# Потоки ввода/вывода языка Java. Класс *ByteInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
ByteArrayOutputStream out = new ByteArrayOutputStream();

out.write(10);
out.write(11);

byte[] bytes = out.toByteArray();
```

# Потоки ввода/вывода языка Java.

## Класс *FileOutputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *OutputStream*
- Используется для записи в файл.
- *FileOutputStream(String fileName) throws FileNotFoundException*

*FileOutputStream(String name, boolean append)*  
*FileNotFoundException*

Конструктор файлового потока вывода.

# Потоки ввода/вывода языка Java. Класс *FileOutputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
byte[] bytesToWrite = {1, 2, 3};  
String fileName = "d:\\test.txt";  
  
try {  
    // Создать выходной поток  
    FileOutputStream outFile = new  
                           FileOutputStream(fileName);  
    System.out.println("Файл открыт для записи");  
  
    // Записать массив  
    outFile.write(bytesToWrite);  
    System.out.println("Записано: " + bytesToWrite.length  
                      + " байт");  
  
    // По окончании использования поток должен быть закрыт  
    outFile.close();  
    System.out.println("Выходной поток закрыт");
```

# Потоки ввода/вывода языка Java.

## Класс *FileOutputStream*. Завершение примера

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

Романов Владимир Юрьевич ©2025

```
    }
    catch (FileNotFoundException e) {
        System.out.println("Невозможно произвести чтение из" +
                           " файла: " + fileName);
    }
    catch (IOException e) {
        System.out.println("Ошибка ввода/вывода: " +
                           e.toString());
    }
}
```

# Потоки ввода/вывода языка Java.

## Класс *FileOutputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *OutputStream*
- Используется для записи в файл.
- *FileOutputStream(String fileName) throws FileNotFoundException*

*FileOutputStream(String name, boolean append)*  
*FileNotFoundException*

Конструктор файлового потока вывода.

# Потоки ввода/вывода языка Java.

## Классы PipedInputStream и PipedOutputStream

### Назначение и методы

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

Романов Владимир Юрьевич ©2025

- Используются в программе только в паре. Рекомендуется использовать ввод и вывод в отдельных потоках управления (threads)
- **PipedInputStream** является потомком класса *InputStream*  
**PipedOutputStream** является потомком класса *OutputStream*
- **PipedInputStream()**  
**PipedOutputStream()**  
Создают еще не подсоединеные потоки ввода и вывода.
- **PipedInputStream(PipedOutputStream src)**  
**PipedOutputStream(PipedInputStream snk)**  
Создают пару соединенных потоков

# Потоки ввода/вывода языка Java.

## Класс PipedInputStream. Методы

---

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

Романов Владимир Юрьевич ©2025

- **Int available()**

Возвращает количество байтов которое можно скачать из потока ввода без блокировки.

- **void close()**

Закрывает поток ввода и освобождает связанные с ним ресурсы.

- **void connect(PipedOutputStream src)**

Присоединяет поток ввода к потоку вывода *src*.

- **int read()**

Читает следующий байт из потока ввода.

- **int read(byte[] b, int off, int len)**

Читает *len* байтов из потока ввода в массив байтов.

# Потоки ввода/вывода языка Java.

## Класс PipedOutputStream. Методы

- **void close()**

Закрывает поток вывода и освобождает связанные с ним ресурсы.

- **void connect(PipedInputStream snk)**

Присоединяет поток вывода к получателю.

- **void flush()**

Стирает поток вывода и выводи накопленные в буфере данные.

- **void write(byte[] b, int off, int len)**

Пишет **len** байтов из описанного байтового массива в поток вывода.

- **void write(int b)**

Пишет байт в поток вывода

# Потоки ввода/вывода языка Java.

## Классы PipedInputStream и PipedOutputStream.

### Пример

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

Романов Владимир Юрьевич ©2025

```
try {
    int countRead = 0;
    byte[] toRead = new byte[100];
    PipedInputStream pipeIn = new PipedInputStream();
    PipedOutputStream pipeOut = new PipedOutputStream(pipeIn);

    // Считывать в массив, пока он полностью не будет заполнен
    while(countRead<toRead.length) {
        // Записать в поток некоторое количество байт
        for(int i=0; i<(Math.random()*10); i++)
            pipeOut.write((byte)(Math.random()*127));

        // Считать из потока доступные данные,
        // добавить их к уже считанным.
        int willRead = pipeIn.available();
        if(willRead+countRead>toRead.length)

            // Нужно считать только до предела массива
            willRead = toRead.length-countRead;
            countRead += pipeIn.read(toRead, countRead, willRead);
    }
}
```

# Потоки ввода/вывода языка Java.

## Классы PipedInputStream и

## PipedOutputStream. Продолжение примера

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

Романов Владимир Юрьевич ©2025

```
    } catch (IOException e) {  
        System.out.println ("Impossible IOException occur: ") ;  
        e.printStackTrace () ;  
    }
```

# Потоки ввода/вывода языка Java.

## Класс *FilterOutputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *OutputStream*
- Используется как класс-адаптер для записи в файл.
- *protected FilterOutputStream(OutputStream os)*  
Конструкторы фильтрованного потока вывода.
- Предок для классов:
  - *BufferedOutputStream*
  - *DataOutputStream*
  - *PrintStream*
  - *DeflaterOutputStream*

# Потоки ввода/вывода языка Java.

## Класс *BufferedOutputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *FilterOutputStream*
- Используется для буферизации данных перед их записью в надстраиваемый поток данных.
- *BufferedOutputStream(OutputStream in)*  
*BufferedOutputStream(OutputStream in, int size)*  
Конструкторы буферизованного потока вывода.

# Потоки ввода/вывода языка Java. Класс *BufferedOutputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
try {
    String fileName = "d:\\file1";
    InputStream inStream = null;
    OutputStream outStream = null;

    //Записать в файл некоторое количество байт
    long timeStart = System.currentTimeMillis();

    outStream = new FileOutputStream(fileName);
    outStream = new BufferedOutputStream(outStream);

    for(int i = 1000000; --i >= 0;)
        outStream.write(i);

    long time = System.currentTimeMillis() - timeStart;
    System.out.println("Writing time: " + time + " millisec");
    outStream.close();
```

# Потоки ввода/вывода языка Java. Класс *BufferedOutputStream*. Продолжение

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

Романов Владимир Юрьевич ©2025

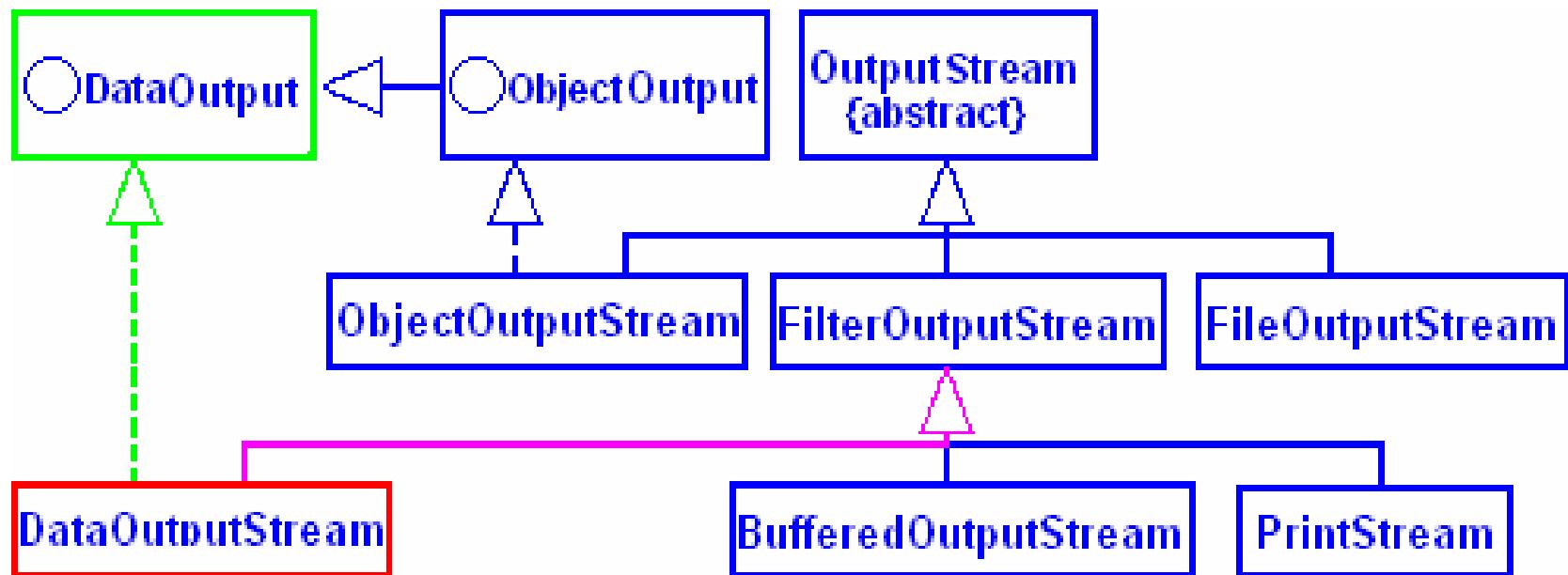
```
    }
    catch (IOException e) {
        System.out.println("IOException: " + e.toString());
        e.printStackTrace();
    }
```

# Потоки ввода/вывода языка Java.

## Класс *DataOutputStream*. UML диаграмма

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

Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java.

## Класс *DataOutputStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Является потомком класса *FilterOutputStream*
- Используется для записи не только байтов, но и примитивных типов данных языка Java
- Реализует интерфейс *DataOutput*

# Потоки ввода/вывода языка Java.

## Интерфейс *DataOutput*. Методы

---

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

Романов Владимир Юрьевич ©2025

- `void write(byte[] b)`
- `void write(byte[] b, int off, int len)`
- `void write(int b)`
- `void writeBoolean(boolean v)`
- `void writeByte(int v)`
- `void writeBytes(String s)`
- `void writeChar(int v)`
- `void writeChars(String s)`
- `void writeDouble(double v)`
- `void writeFloat(float v)`
- `void writeInt(int v)`
- `void writeLong(long v)`
- `void writeShort(int v)`
- `void writeUTF(String str)`

# Потоки ввода/вывода языка Java.

## Класс *DataOutputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
try {
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    DataOutputStream outData = new DataOutputStream(out);

    outData.writeByte(128);

    // этот метод принимает аргумент int, но записывает
    // лишь младший байт
    outData.writeInt(128);
    outData.writeLong(128);
    outData.writeDouble(128);
    outData.close();
}
catch (Exception e) {
    System.out.println("Impossible IOException occurs: " +
        e.toString());
    e.printStackTrace();
}
```

# Потоки ввода/вывода языка Java.

## Класс *PrintStream*. Назначение и методы

- Является потомком класса *FilterOutputStream*
- Для форматированного вывода данных в текстовый файл.
- `PrintStream format(Locale l, String format, Object... args)`  
`PrintStream format(String format, Object... args)`  
Форматированный вывод в поток данных
- `PrintStream printf(Locale l, String format, Object... args)`  
`PrintStream printf(String format, Object... args)`  
Форматированный вывод в поток данных

# Потоки ввода/вывода языка Java.

## Класс *PrintStream*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- *void println()*
- *void println(boolean x)*
- *void println(char x)*
- *void println(char[] x)*
- *void println(double x)*
- *void println(float x)*
- *void println(int x)*
- *void println(long x)*
- *void println(Object x)*
- *void println(String x)*

# Потоки ввода/вывода языка Java.

## Таблица соответствия потоков вывода и писателей

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

Романов Владимир Юрьевич ©2025

| <b>Байтовый поток</b>          | <b>Символьный поток</b>   |
|--------------------------------|---------------------------|
| • <i>OutputStream</i>          | <i>Writer</i>             |
| • <i>ByteArrayOutputStream</i> | <i>CharArrayWriter</i>    |
| • ---                          | <i>OutputStreamWriter</i> |
| • <i>FileOutputStream</i>      | <i>FileWriter</i>         |
| • <i>FilterOutputStream</i>    | <i>FilterWriter</i>       |
| • <i>BufferedOutputStream</i>  | <i>BufferedWriter</i>     |
| • <i>PrintStream</i>           | <i>PrintWriter</i>        |
| • <i>DataOutputStream</i>      | ---                       |
| • <i>ObjectOutputStream</i>    | ---                       |
| • <i>PipedOutputStream</i>     | <i>PipedWriter</i>        |
| • ---                          | <i>StringWriter</i>       |

# Потоки ввода/вывода языка Java. Класс *FileWriter*. Пример

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

Романов Владимир Юрьевич ©2025

```
String fileName = "d:\\file.txt";

//Строка, которая будет записана в файл
String data = "Some data to be written and read.\n";

try{
    FileWriter fw = new FileWriter(fileName);
    BufferedWriter bw = new BufferedWriter(fw);

    System.out.println("Write some data to file: " + fileName);

    // Несколько раз записать строку
    for(int i = (int)(Math.random()*10); --i >= 0; )
        bw.write(data);

    bw.close();
}
catch(Exception e) {
    e.printStackTrace();
}
```

# Потоки ввода/вывода языка Java.

## Класс *java.util.Scanner*. Назначение и методы

---

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

Романов Владимир Юрьевич ©2025

- Использует поток как параметр
- Используется для чтения структурированного текста.
- Позволяет распознавать примитивные типы данных и строки
- **Scanner(InputStream source)**  
**Scanner(String source)**  
Конструкторы для распознавания потока ввода и строки

# Потоки ввода/вывода языка Java.

## Класс *java.util.Scanner*. Методы просмотра вперед

---

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

Романов Владимир Юрьевич ©2025

- *boolean*      *hasNextBoolean()*
- *boolean*      *hasNextByte()*
- *boolean*      *hasNextDouble()*
- *boolean*      *hasNextFloat()*
- *boolean*      *hasNextInt()*

# Потоки ввода/вывода языка Java.

## Класс *java.util.Scanner*. Методы чтения данных

---

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

Романов Владимир Юрьевич ©2025

- *String*            *next()*
- *boolean*        *nextBoolean()*
- *byte*            *nextByte()*
- *double*          *nextDouble()*
- *float*            *nextFloat()*
- *int*              *nextInt()*
- *String*          *nextLine()*
- *long*            *nextLong()*
- *Short*           *nextShort()*

# Потоки ввода/вывода языка Java.

## Класс *Scanner*. Пример

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

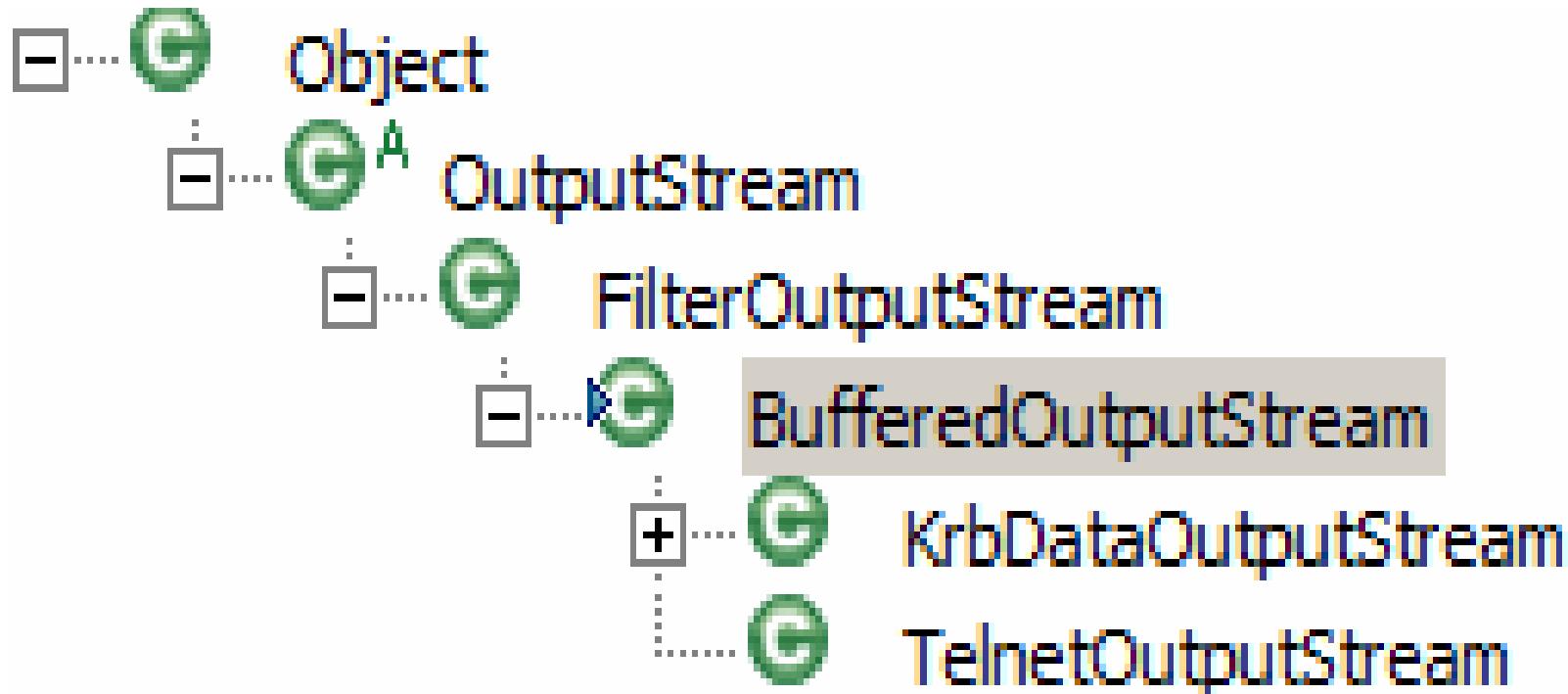
Романов Владимир Юрьевич ©2025

```
Scanner sc = new Scanner( new FileReader("numbers.txt") ) ;  
  
while (sc.hasNextLong()) {  
    long aLong = sc.nextLong();  
}  
  
sc.close();
```

# Потоки ввода/вывода языка Java. Класс *BufferedOutputStream*. Пример

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

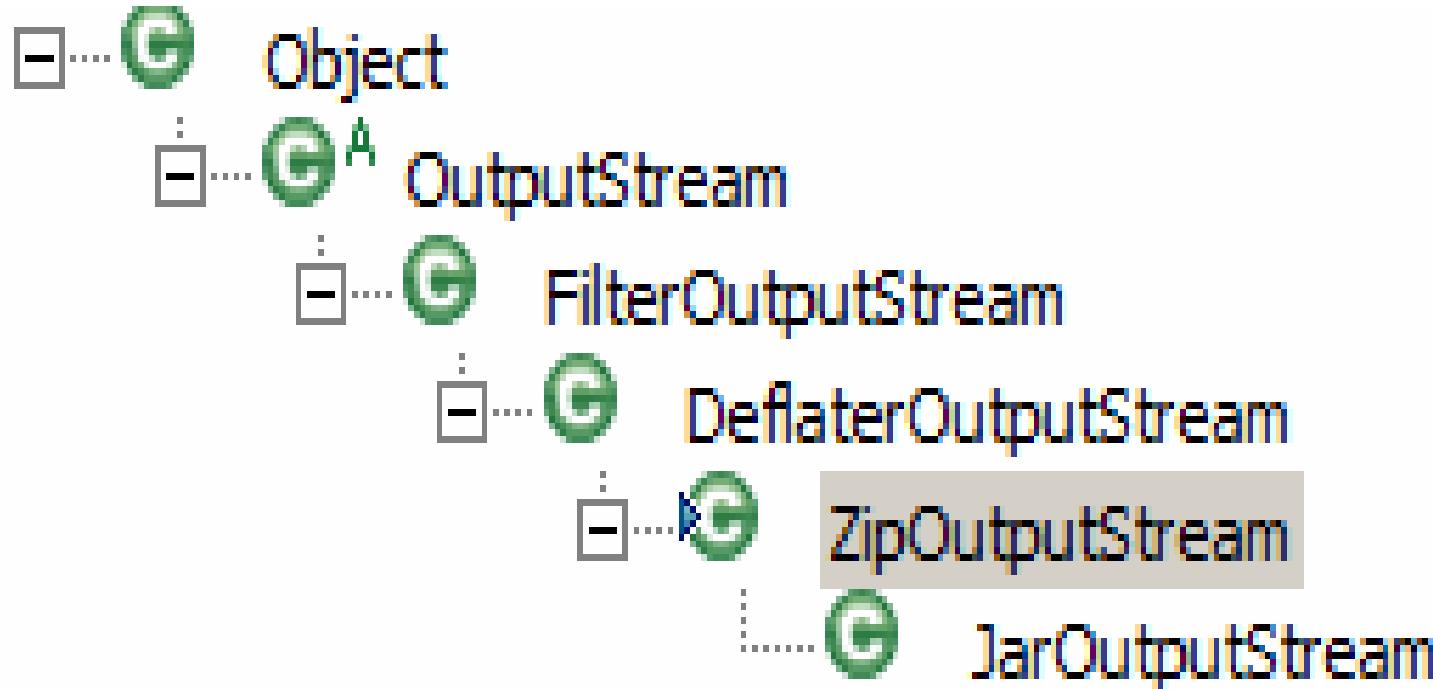
Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java. Класс ZipOutputStream. Пример

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

Романов Владимир Юрьевич ©2025



# Потоки ввода/вывода языка Java. Класс *ZipInputStream*. Пример

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

Романов Владимир Юрьевич ©2025

```
FileOutputStream fo;
BufferedOutputStream bo;
ZipOutputStream zo;

try {
    fo = new FileOutputStream("demo.zip");
    bo = new BufferedOutputStream(fo);
    zo = new ZipOutputStream(bo);

    ZipEntry ze = new ZipEntry("ZippedFile1.txt");
    zo.putNextEntry(ze);
```

Создаем новый элемент  
в zip-архиве.

# Потоки ввода/вывода языка Java. Класс ZipOutputStream. Пример

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

Романов Владимир Юрьевич ©2025

```
FileOutputStream fo;
BufferedOutputStream bo;
ZipOutputStream zo;

try {
    fo = new FileOutputStream("demo.zip");
    bo = new BufferedOutputStream(fo);
    zo = new ZipOutputStream(bo);

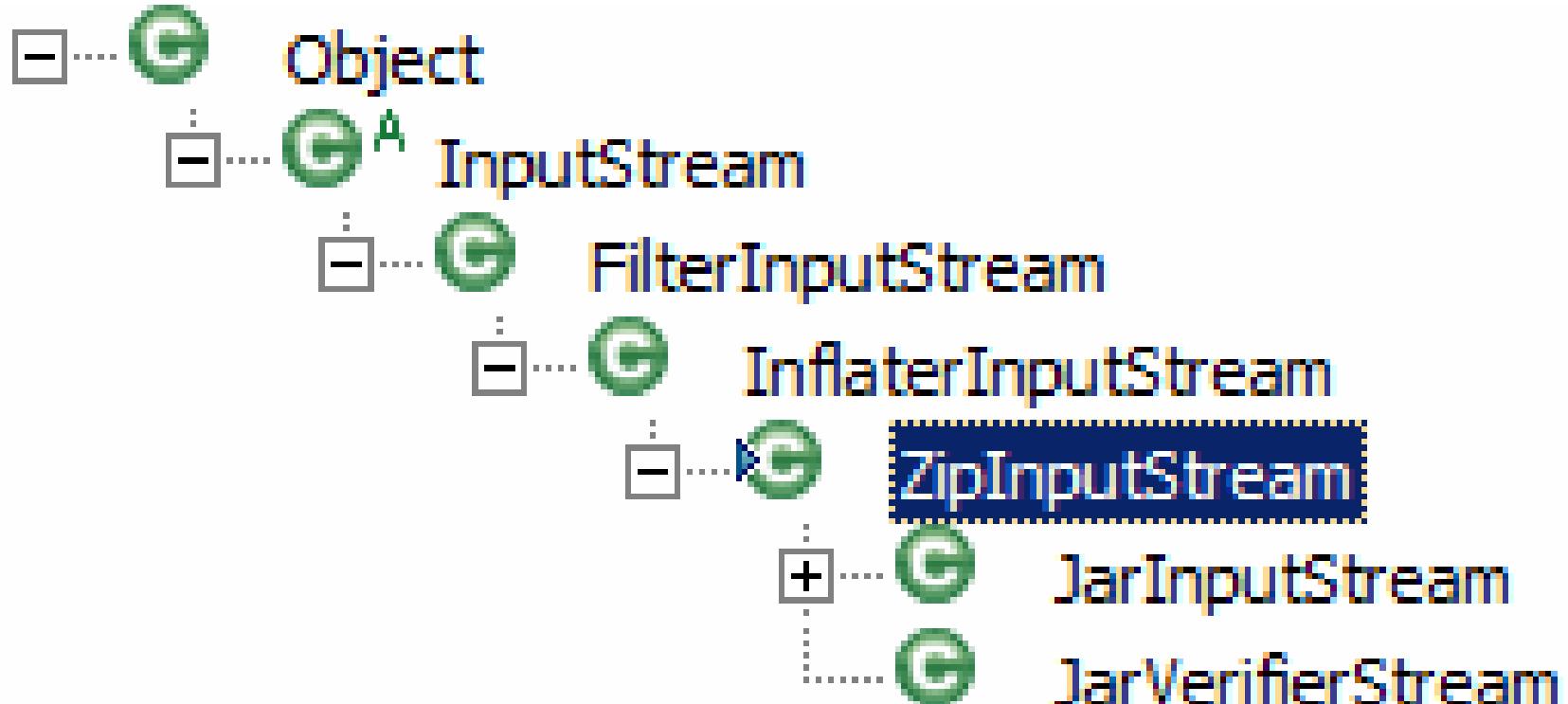
    ZipEntry ze = new ZipEntry("ZippedFile1.txt");
    zo.putNextEntry(ze);
```

Создаем  
новый  
элемент в zip-  
архиве.

# Потоки ввода/вывода языка Java. Класс ZipInputStream. Пример

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

Романов Владимир Юрьевич ©2025



# Интернационализация программ в языке Java

# Интернационализация программ на Java. Файлы текстовых ресурсов

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

Романов Владимир Юрьевич ©2025

- Файл **StringResources\_ru.properties**

**yes=Да**

**no=Нет**

- Файл **StringResources\_en.properties**

**yes=Yes**

**no>No**

- Файл **StringResources.properties**

**yes=Yes**

**no>No**

# Интернационализация программ на Java.

## Использование умалчивающего файла ресурсов

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

Романов Владимир Юрьевич ©2025

```
ResourceBundle bundle =  
    ResourceBundle.getBundle("StringResources"  
);  
  
String title = bundle.getString("yes");  
  
Button yesButton = new Button(title);
```

Текст для ключа **yes** из  
умалчивающего файла  
**StringResources.properties**

# Интернационализация программ на Java.

## Использование файла ресурсов

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

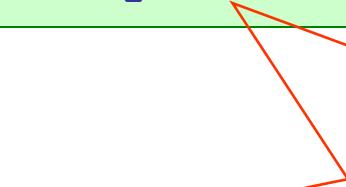
Романов Владимир Юрьевич ©2025

```
Locale locale= new Locale("ru");

ResourceBundle bundle =
    ResourceBundle.getBundle("StringResources", locale);

String title = bundle.getString("yes");

Button yesButton = new Button(title);
```



Текст для ключа **yes** из файла  
**StringResources\_ru.properties**

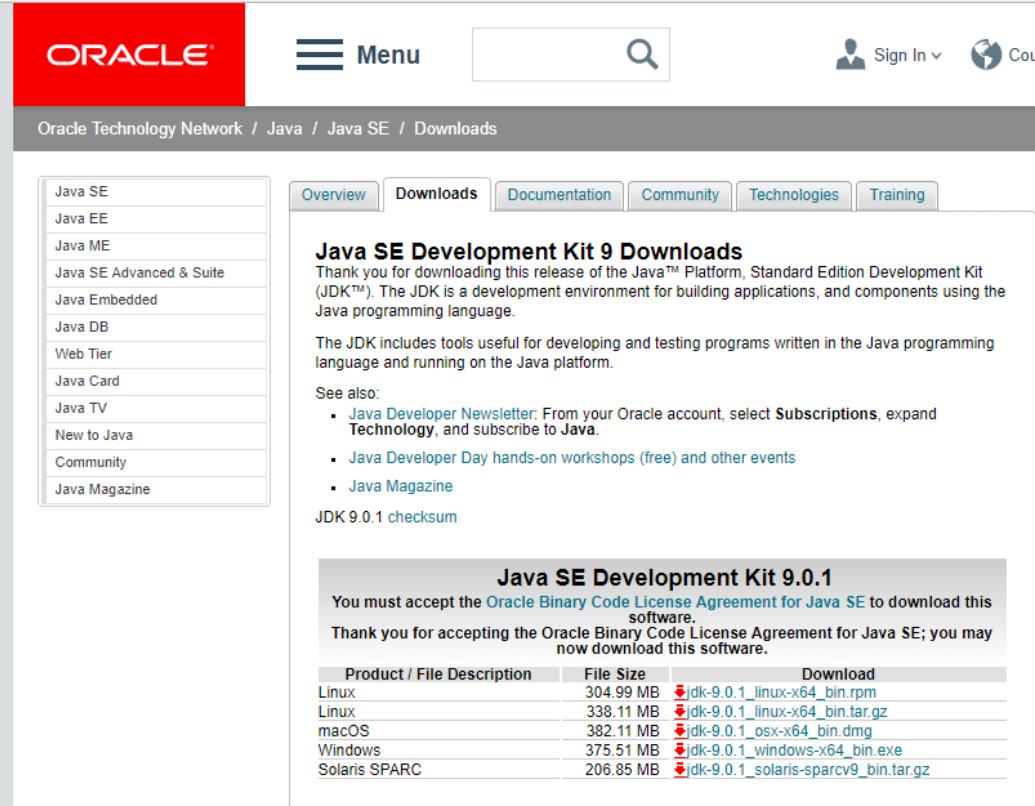
# Java 9

# Загрузка JDK9

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

Романов Владимир Юрьевич ©2025

<http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html>



The screenshot shows the Oracle Java SE Development Kit 9 Downloads page. The page has a red header with the Oracle logo. The navigation bar includes 'Menu', a search bar, and 'Sign In' and 'Courses' buttons. The main menu on the left lists categories like Java SE, Java EE, Java ME, etc. The 'Downloads' tab is selected in the top navigation bar. The main content area is titled 'Java SE Development Kit 9 Downloads' and includes a brief description of the JDK. It lists 'See also:' items: Java Developer Newsletter, Java Developer Day, and Java Magazine. Below this is a link to the 'JDK 9.0.1 checksum'. A large callout box at the bottom left states: 'Java SE Development Kit 9.0.1' and 'You must accept the Oracle Binary Code License Agreement for Java SE to download this software.' It also says 'Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.' A table at the bottom shows download links for various operating systems: Linux, Linux, macOS, Windows, and Solaris SPARC. Each row includes a file size and a download link.

| Product / File Description | File Size | Download   |
|----------------------------|-----------|--|
| Linux                      | 304.99 MB | <a href="#">jdk-9.0.1_linux-x64_bin.rpm</a>          |
| Linux                      | 338.11 MB | <a href="#">jdk-9.0.1_linux-x64_bin.tar.gz</a>       |
| macOS                      | 382.11 MB | <a href="#">jdk-9.0.1_osx-x64_bin.dmg</a>            |
| Windows                    | 375.51 MB | <a href="#">jdk-9.0.1_windows-x64_bin.exe</a>        |
| Solaris SPARC              | 206.85 MB | <a href="#">jdk-9.0.1_solaris-sparcv9_bin.tar.gz</a> |

# Установка Eclipse

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

Романов Владимир Юрьевич ©2025

[http://www.eclipse.org/downloads  
/packages/release](http://www.eclipse.org/downloads/packages/release)

The screenshot shows the Eclipse website's download section for the Oxygen 1A release. At the top, there's a navigation bar with links for 'GETTING STARTED', 'MEMBERS', 'PROJECTS', and 'MORE'. On the right, there's a 'Google Custom Search' bar and a large orange 'DOWNLOAD' button. Below the navigation, a breadcrumb trail shows 'HOME / DOWNLOADS / PACKAGES / ECLIPSE OXYGEN 1A PACKAGES'. The main content area features a large heading 'Eclipse Oxygen 1A Packages' and a list of download options. Each option includes a small icon, the package name, its size, and the number of downloads. For the 'Eclipse IDE for Java Developers', the details are: 'Eclipse IDE for Java Developers', '180 MB - Downloaded 443,392 Times'. On the right, there's a column with download links for Windows, Mac, and Linux in both 32-bit and 64-bit architectures.

## RELEASES

Oxygen Packages  
Photon Packages  
Neon Packages  
Mars Packages

## Eclipse Oxygen 1A Packages



### Eclipse IDE for Java Developers

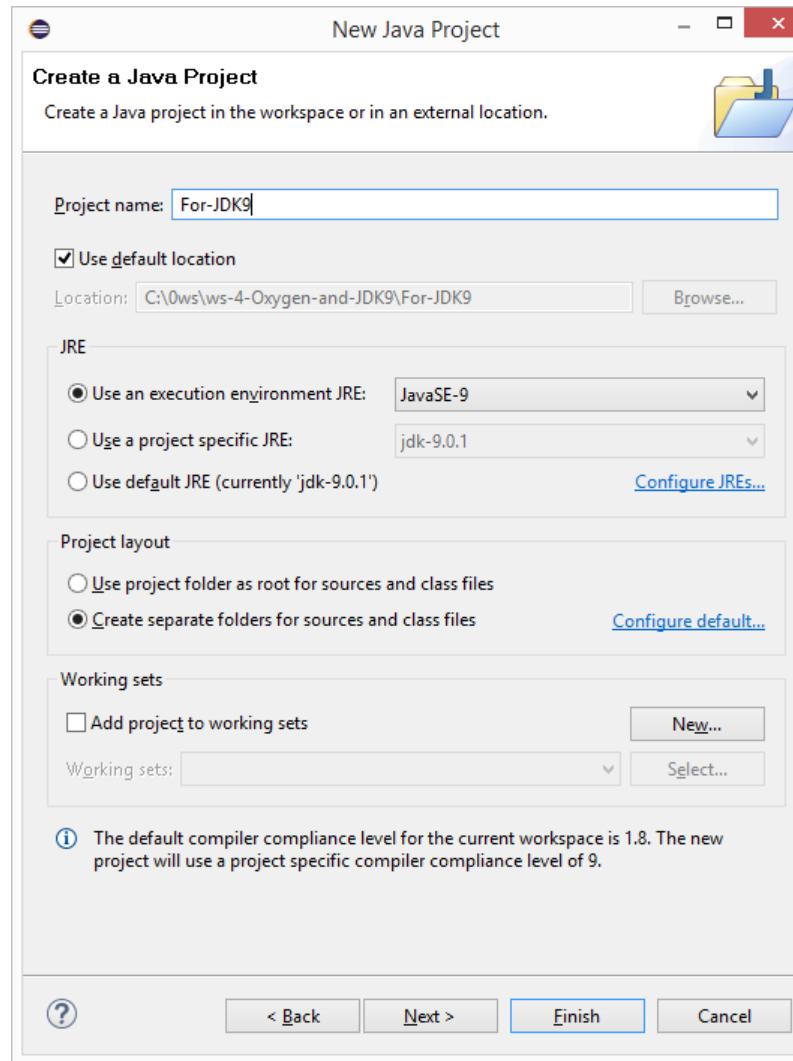
180 MB - Downloaded 443,392 Times

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

# Создание проекта в Eclipse

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

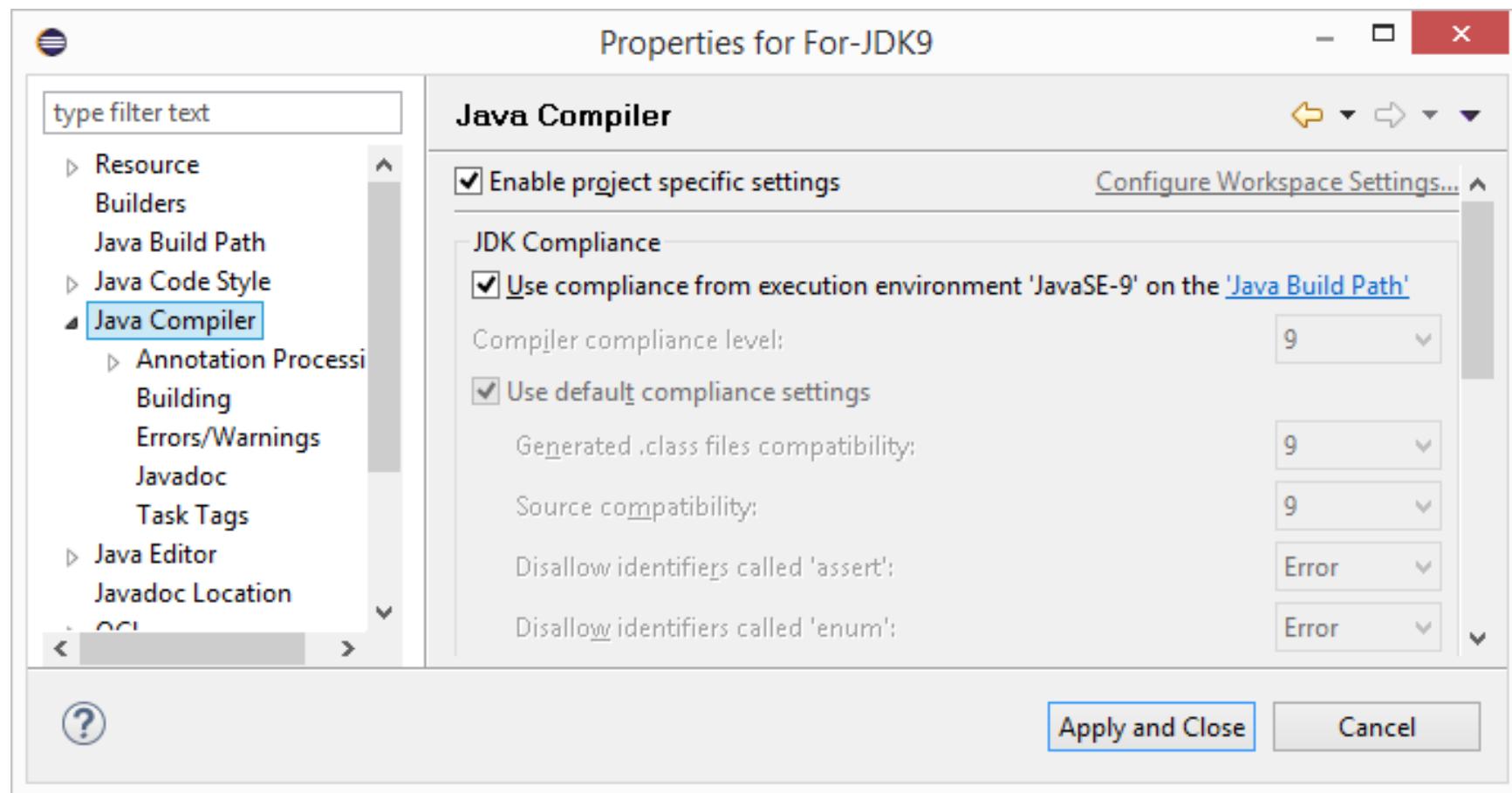
Романов Владимир Юрьевич ©2025



# Совместимость кода в Eclipse

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

Романов Владимир Юрьевич ©2025



# Модули в Java 9

# Java 9. Назначение модулей (1)

- Если набор пакетов достаточно согласован, пакеты могут быть сгруппированы в **модуль**.
- Модуль классифицирует некоторые или все его пакеты как **экспортированные**, что означает, что их типы могут быть доступны из кода вне модуля.
- Если пакет **НЕ экспортируется** модулем, тогда только код внутри модуля может обращаться к его типам.

# Java 9. Назначение модулей (2)

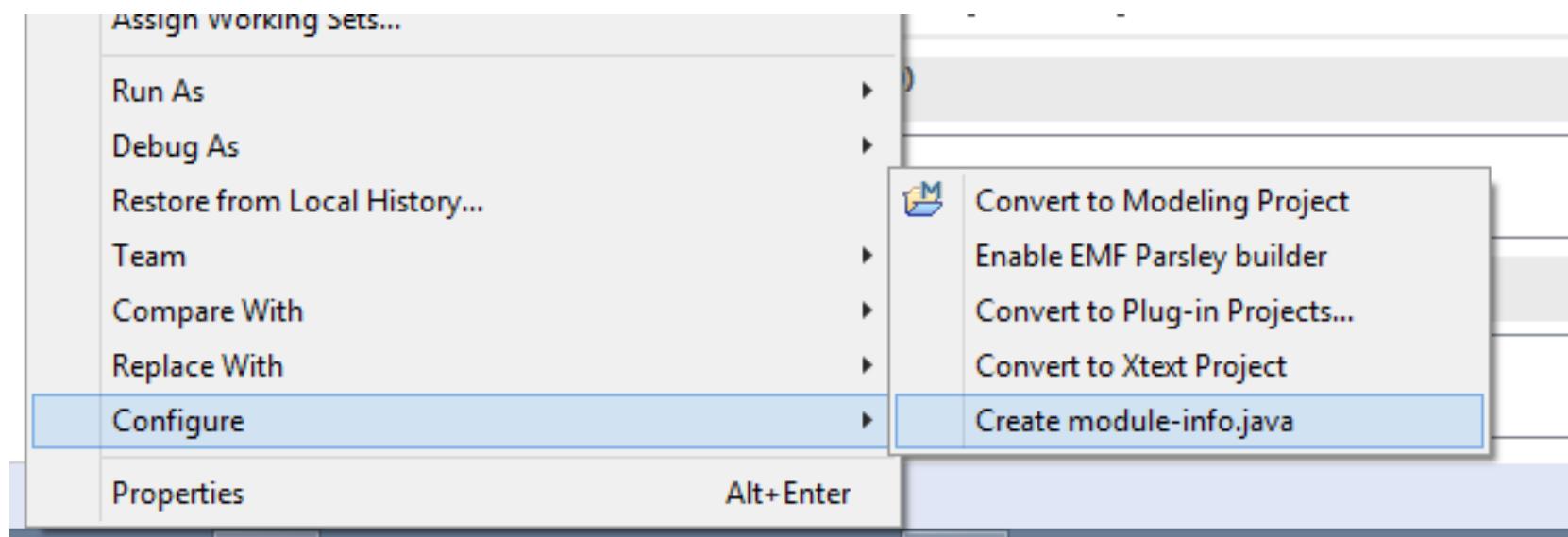
- Кроме того, если код в модуле хочет получить доступ к пакетам, экспортированным другим модулем, то первый модуль должен **ЯВНО зависеть** от второго модуля.
- Таким образом модуль:
  - контролирует, как его пакеты используют другие модули (определяя **зависимости**)
  - контролирует как другие модули используют его пакеты (указывая **экспорт** этих пакетов)

# Создание файла - модуля

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

Романов Владимир Юрьевич ©2025

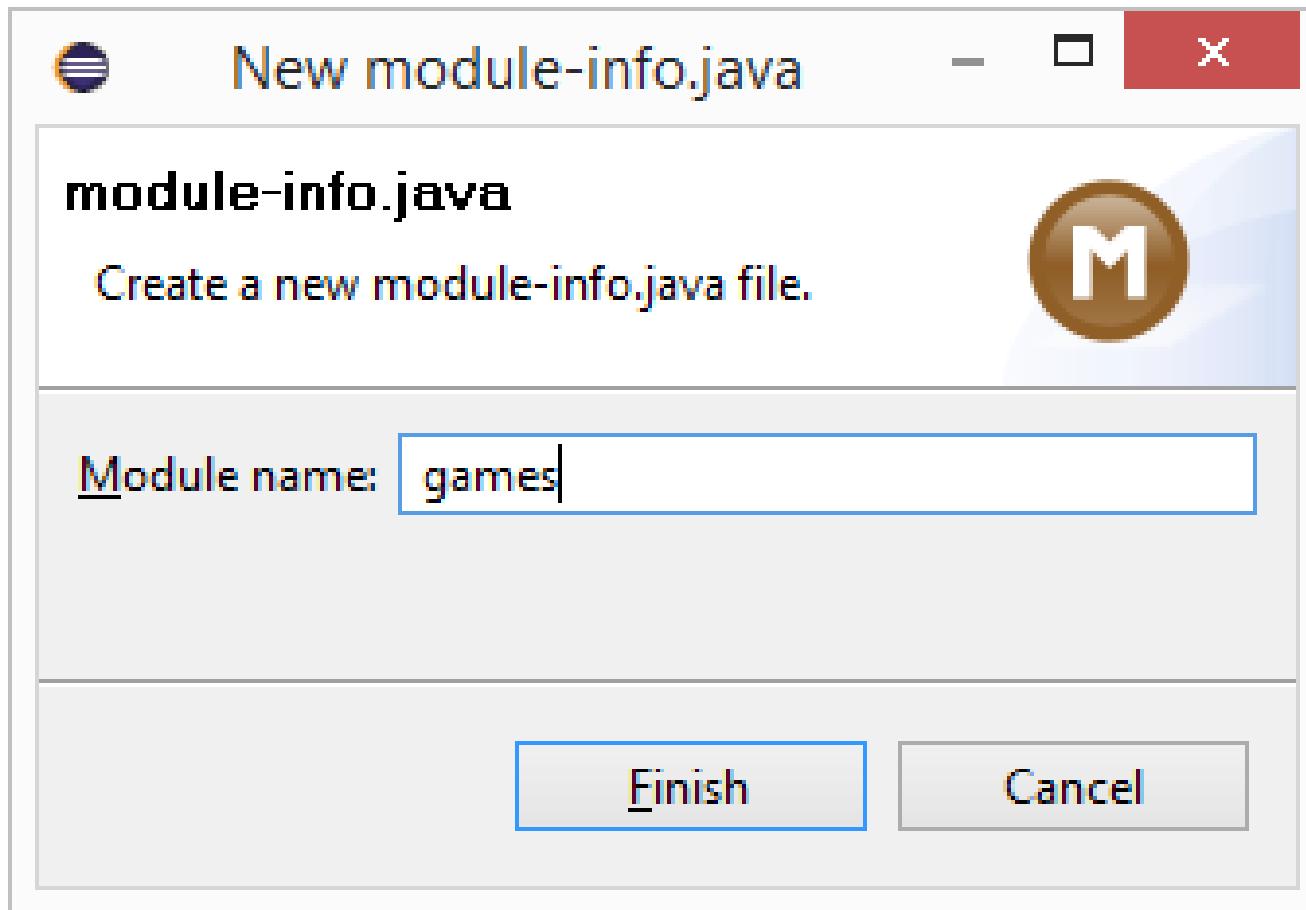
## Меню проекта Java:



# Создание файла - модуля

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

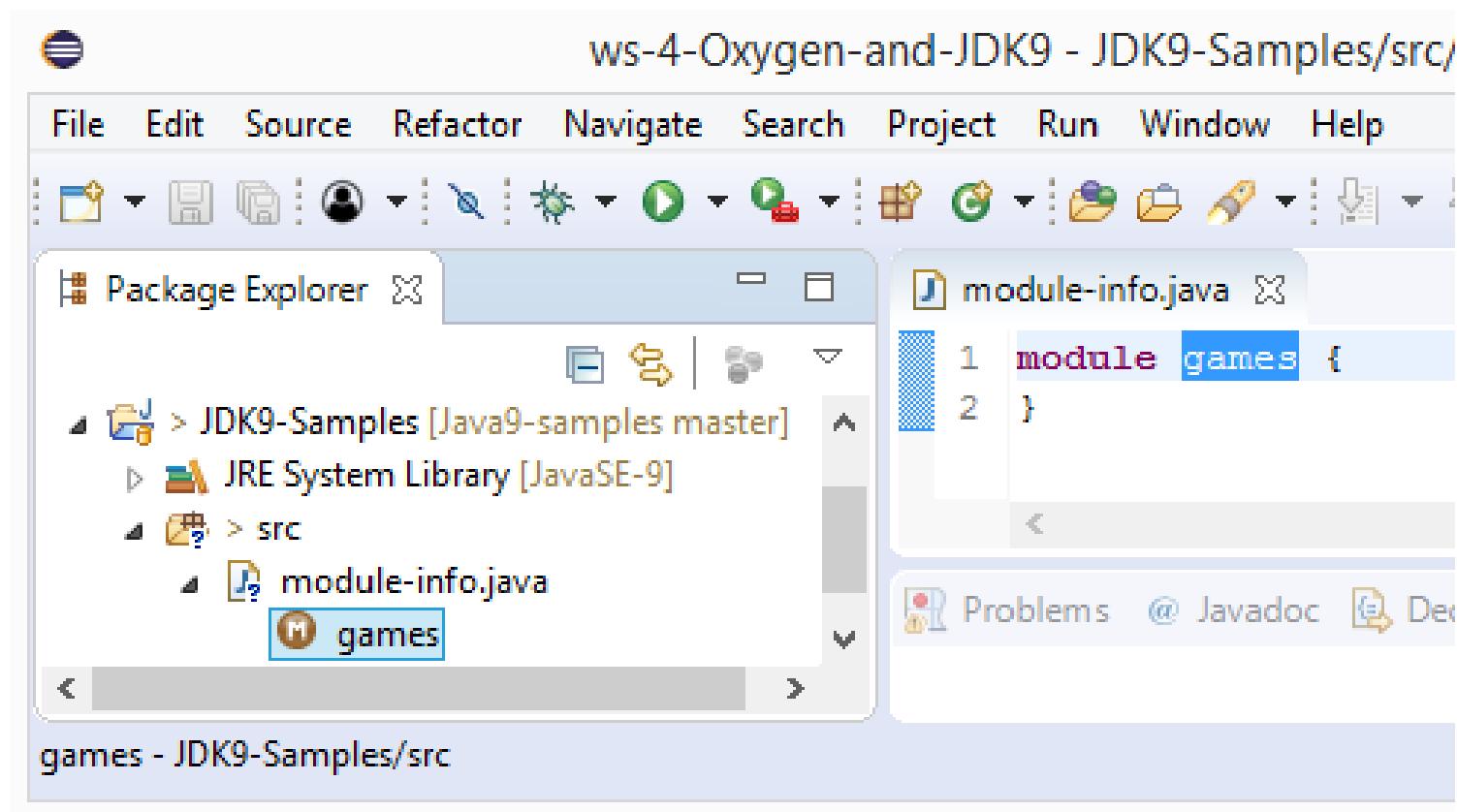
Романов Владимир Юрьевич ©2025



# Создание файла - модуля

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

Романов Владимир Юрьевич ©2025



# Java 9. Синтаксис описания модулей

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

Романов Владимир Юрьевич ©2025

```
ModuleDeclaration =  
  { Annotation }  
  [ "open" ]  
  "module" Identifier { .Identifier }  
  "{"  
    { ModuleDirective }  
  "}"
```

До **Java 9** через механизм **Reflection** были доступны и приватные методы класса.

Теперь доступ к коду в время выполнения только для **открытых (open)** модулей

# Java 9. Синтаксис описания модулей

## Файл module-info.java

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

Романов Владимир Юрьевич ©2025

```
ModuleDirective =  
    "requires" { RequiresModifier } ModuleName ";" "  
    "exports" PackageName  
        [ "to" ModuleName { "," " ModuleName } ] ";" "  
    "opens"    PackageName  
        [ "to" ModuleName { "," " ModuleName } ] ";" "  
    "uses"    TypeName ";" "  
    "provides" TypeName  
        "with" TypeName { "," " TypeName } ";" "  
  
RequiresModifier =  
    "transitive" | "static"
```

# Java 9. Пример модулей. Открытый модуль

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

Романов Владимир Юрьевич ©2025

```
open
module com.example.foo {
    requires com.example.foo.http;
    requires java.logging;
    requires transitive com.example.foo.network;

    exports com.example.foo.bar;
    exports com.example.foo.internal
        to com.example.foo.probe;

    uses com.example.foo.spi.Intf;

    provides com.example.foo.spi.Intf
        with com.example.fooImpl;
}
```

# Java 9. Пример модулей. Открытый модуль

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

Романов Владимир Юрьевич ©2025

```
open  
module com.example.foo {
```

модуль открыт во время  
выполнения для всех  
модулей

```
}
```

# Java 9. Пример модулей. Открытость модуля

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

Романов Владимир Юрьевич ©2025

```
module com.example.foo {  
    requires com.example.foo.http;  
    requires java.logging;  
    requires transitive com.example.foo.network;  
  
    exports com.example.foo.bar;  
    exports com.example.foo.internal  
        to com.example.foo.probe;  
  
    opens com.example.foo.quux;  
    opens com.example.foo.internal  
        to com.example.foo.network, com.example.foo.probe;  
}
```

# Java 9. Пример модулей. Открытость модуля

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

Романов Владимир Юрьевич ©2025

```
module com.example.foo {
```

МОДУЛЬ ОТКРЫВАЕТ ПАКЕТ ВО  
ВРЕМЯ ВЫПОЛНЕНИЯ ДЛЯ  
**ВСЕХ** МОДУЛЕЙ

```
    opens com.example.foo.куих;
```

```
    opens com.example.foo.internal  
        to com.example.foo.network, com.example.foo.probe;
```

```
}
```

МОДУЛЬ ОТКРЫВАЕТ ПАКЕТ ВО  
ВРЕМЯ ВЫПОЛНЕНИЯ ДЛЯ  
**ОПРЕДЕЛЕННЫХ** МОДУЛЕЙ

# Java 9. Пример модулей. Транзитивность требований

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

Романов Владимир Юрьевич ©2025

```
open
module com.example.foo {
    // ...
    requires transitive com.example.foo.network;

    // ...
}
```

- Любой модуль, который требует модуль **com.example.foo**, должен явно требовать и модуль **com.example.foo.network**
- ключевое слово **static** после слова **requires** означает **обязательность** зависимости во время компиляции, и **возможность** во время выполнения

# Java 9. Пример модулей

## Экспорт и открытие

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

Романов Владимир Юрьевич ©2025

```
module com.example.foo {  
    // ...  
    exports com.example.foo.bar;  
    exports com.example.foo.internal  
        to com.example.foo.probe;  
  
    opens com.example.foo.quux;  
    opens com.example.foo.internal  
        to com.example.foo.network,  
        com.example.foo.probe;  
    // ...  
}
```

- Для других модулей код **пакета** доступен
  - **exports** – во время компиляции и выполнения
  - **opens** – только во время выполнения
- **to** в конкретных модулях (друзьях модуля)

# Изменения в языке Java 9

# Java 9. Приватные методы интерфейса

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

Романов Владимир Юрьевич ©2025

```
public interface PrivateInterfaceMethods {  
    private int getName() {  
        return 1;  
    }  
  
    private static void displayCardDetails() {  
    }  
}
```

# Java 9. Методы-фабрики

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

Романов Владимир Юрьевич ©2025

```
List<?> immutableList = List.of();
```

```
List<String> immutableList2
    = List.of("one", "two", "three");
```

```
Map<?, ?> emptyImmutableMap = Map.of();
```

```
Map<Integer, String> nonemptyImmutableMap
    = Map.of(1, "one",
              2, "two",
              3, "three");
```

# Изменения в языке Java 10

# Java 10. Новый тип переменных *var*

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

Романов Владимир Юрьевич ©2025

```
static public void main(String[] args) {  
    var s = "Java";  
  
    name = 3; // Ошибка  
  
    var stringList = List.of("1", "2", "3");  
  
    var mixList = List.of( 2, 3.0);  
  
}  
  
// В лямбда выражениях  
(var firstName, var lastName) -> firstName + lastName
```

# Изменения в языке Java 11

# Java 11. Новые методы типа *String*

```
Path path = Files.writeString(  
    Files.createTempFile("helloworld", ".txt"),  
    "Hi, my name is!");
```

```
String s = Files.readString(path);
```

```
Stream<String> lines = "Jav\nna".lines();
```

```
"Java ".strip();
```

```
"Java".isBlank();
```

# Изменения в языке Java 12

# Java 12. Предложение `switch`. Старый

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

Романов Владимир Юрьевич ©2025

```
switch (day) {  
    case MONDAY:  
    case FRIDAY:  
    case SUNDAY:  
        numLetters = 6;  
        break;  
    case TUESDAY:  
        numLetters = 7;  
        break;  
    case THURSDAY:  
    case SATURDAY:  
        numLetters = 8;  
        break;  
    case WEDNESDAY:  
        numLetters = 9;  
        break;  
    default:  
        throw new IllegalStateException("Huh? " + day);  
}
```

# Java 12. Предложение `switch`. Новый

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

Романов Владимир Юрьевич ©2025

```
int numLetters = switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> 6;  
    case TUESDAY -> 7;  
    case THURSDAY, SATURDAY -> 8;  
    case WEDNESDAY -> 9;  
    default -> throw new IllegalStateException("Huh? " + day);  
};
```

# Изменения в языке Java 13

# Java 13. Блоки текста

```
String html =  
    """  
        <html>  
            <body>  
                <p>Hello, world</p>  
            </body>  
        </html>  
    """;
```

# Изменения в языке Java 14

# Java 14. Records

## Было

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

Романов Владимир Юрьевич ©2025

```
final class Point
{
    public final int x;
    public final int y;

    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
}
```

# Java 14. Records (записи)

## Стало

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

Романов Владимир Юрьевич ©2025

```
record Point(int x, int y) { }
```

- Создается **private final** поле для его компонентов **x, y**
- **public** метод доступа для чтения для компонента с тем же именем и типом компонента методы **Point::x()** и **Point::y()**.
- **public** конструктор, сигнатура которого получена из списка компонентов записи.
- Конструктор инициализирует каждое *private* поле с помощью соответствующего аргумента.
- Реализации методов **equals()** и **hashCode ()**, которые указывают, что две записи равны, если они одного типа и их соответствующие компоненты записи равны
- Реализация метода **toString()**, который включает строковое представление всех компонентов записи с их именами.

# Java 14. Преобразование типов после *instanceof*

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

Романов Владимир Юрьевич ©2025

```
if (obj instanceof String) {  
    String s = (String) obj;  
  
    System.out.println(s.contains("hello"));  
}
```

# Java 14. Преобразование типов после *instanceof*

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

Романов Владимир Юрьевич ©2025

```
if (obj instanceof String s) {  
    System.out.println(s.contains("hello"));  
}
```

# Изменения в языке Java 15

# Java 15. Изолированные классы и интерфейсы

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

Романов Владимир Юрьевич ©2025

```
package geometry;

public sealed class Shape
    permits Circle, Rectangle, Square {...}
```

- *sealed (изолированный, запечатанный)* класс или интерфейс можно наследовать или реализовывать **только** тем классам и интерфейсам, которым это разрешили (**permits**).
- Классы, описанные в **permits** должны находиться рядом с суперклассами:
  - В **том же модуле** (если суперкласс находится в **именованном модуле**), или
  - в **том же пакете** (если суперкласс не находится в **именованном модуле**).

# Java 15. Изолированные классы и интерфейсы

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

Романов Владимир Юрьевич ©2025

```
package geometry;

public final class Circle extends Shape {
```

# Java 15. Изолированные классы и интерфейсы

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

Романов Владимир Юрьевич ©2025

```
package geometry;

public non-sealed class Square extends Shape {
```

# Java 15. Изолированные классы и интерфейсы

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

Романов Владимир Юрьевич ©2025

```
package geometry;

public sealed class Rectangle extends Shape
    permits RoundRectangle
{}
```

```
package geometry;

public final class RoundRectangle extends Rectangle {}
```

# Изменения в языке Java 16

# Java 16. Pattern Matching for switch Expressions and Statements

## БЫЛО

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

Романов Владимир Юрьевич ©2025

```
public static double getPerimeter(Shape shape)
    throws IllegalArgumentException
{
    if (shape instanceof Rectangle r) {
        return 2 * r.length() + 2 * r.width();
    }
    else
        if (shape instanceof Circle c) {
            return 2 * c.radius() * Math.PI;
        }
        else {
            throw new IllegalArgumentException("Unrecognized shape");
        }
}
```

# Java 16. Pattern Matching for switch Expressions and Statements

## Стало

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

Романов Владимир Юрьевич ©2025

```
public static double getPerimeter(Shape shape)
    throws IllegalArgumentException
{
    return
        switch (shape) {
            case null -> System.out.println("null!");
            case Rectangle r -> 2 * r.length() + 2 * r.width();
            case Circle c  -> 2 * c.radius() * Math.PI;
            default          ->
                throw new IllegalArgumentException("Unrecognized shape");
        };
}
```