Программирование
1 семестр

ИТМО

Основы ООП

Основы ООП

ИТМО

- Процедурное программирование
  - Глобальные данные
  - Глобальные функции и процедуры
  - Данные передаются процедурам для обработки
  - Структурная и функциональная декомпозиция
  - Подходит для простых одиночных проектов

Основы ООП

ИТМО

- Объектно-ориентированное программирование
  - Объекты - программные модели реальных сущностей
  - Объект соединяет данные и код для обработки этих данных
  - Состояние объекта - его переменные (поля)
  - Поведение объекта - его функции (методы)
  - Объекты взаимодействуют, вызывая методы друг у друга
  - Объектно-ориентированное проектирование

Зачем ООП?

ИТМО

- Проще разрабатывать сложные программы и системы
- Проще поддерживать сложные системы
- Код более структурированный, модульный и понятный
- Проще вносить изменения в программы
- Проще тестировать код и искать ошибки

- Сложнее освоить базовые принципы

Площадь круга - процедурный подход

- $S = \pi R^2$
- массив для хранения радиусов
- функция для вычисления площади

Площадь круга - процедурный подход

ИТМО

```
class Main {
  static double circleArea(double radius) {
    return Math.PI * radius * radius;
  }
  public static void main(String... args) {
    double[] radii = { 1.0, 5.64, 2.5, 10.0 };
    for (double r : radii) {
      double area = circleArea(r);
      System.out.println("Радиус круга: " + r);
      System.out.println("Площадь: " + area);
    }
  }
}
```

Площадь прямоугольника

ИТМО

```
class Main {
  static double rectangleArea(double width, double height) {
    return width * height;
  }
  public static void main(String... args) {
    double[] widths = { 1.0, 5.64, 2.5, 10.0 };
    double[] heights = { 4.0, 5.64, 1.5, 9.0 };
    for (int i = 0; i < widths.length; i++) {
      double area = rectangleArea(widths[i], heights[i]);
      System.out.printf("Стороны: %s и %s%n", widths[i], heights[i]);
      System.out.println("Площадь: " + area);
    }
  }
}
```

Площадь прямоугольника

ИТМО

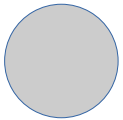
```
class Main {
  static double rectangleArea(double width, double height) {
    return width * height;
  }
  public static void main(String... args) {
    double[] widths = { 1.0, 5.64, 2.5, 10.0, 4.0 };
    double[] heights = { 4.0, 5.64, 1.5, 9.0 };
    for (int i = 0; i < widths.length; i++) {
      double area = rectangleArea(widths[i], heights[i]);
      System.out.printf("Стороны: %s и %s%n", widths[i], heights[i]);
      System.out.println("Площадь: " + area);
    }
  }
}
```

Проблемы процедурного подхода

- Параметров может быть очень много
  - Площадь 12-угольника + периметр
- Массивы - не связаны друг с другом
  - double[] side1 = { ... }; ... double[] side12 = { ... };
- Нужна единая структура данных (dict?, struct?) - class
  - Атрибуты объекта должны иметь свои имена и значения

Классы и объекты

- **Класс** - тип или шаблон или описание объекта
- Конкретный экземпляр (instance) класса - **объект**
- Круг - класс, вот этот синий круг с радиусом 3.5 - объект



- Кот - класс, соседская рыжая Мурка (5 лет) - объект

Классы и объекты

- **Класс** задает какие характеристики есть у объекта
  - с помощью переменных (полей)

```
class Cat
  name
  age
  color
  owner
```

```
объект класса Cat
  name  = Мурка
  age   = 5.1
  color = рыжий
  owner = сосед
```

- Каждый **объект** имеет свои значения характеристик
  - значения полей задают состояние объекта

Базовые принципы ООП

- **Абстракция**
- Инкапсуляция
- Наследование
- Полиморфизм

- Abstraction
- Фокус на существенных деталях при моделировании предметной области, игнорирование несущественных для данной реализации деталей объектов и классов

Принцип абстракции

- Реальные объекты имеют очень много характеристик
- Объекты и классы в программе должны фокусироваться на существенных для программы характеристиках
- Класс Cat:
  - домашний питомец: имя, владелец, спать, есть, мяукать, ...
  - зоомагазин: цвет, порода, цена, продать, сделать скидку, ...
  - ветклиника: возраст, владелец, диагноз, записать, лечить, ...

Класс Rectangle - описание объекта

ИТМО

- ключевое слово class
- + имя класса Rectangle
  (с заглавной буквы, camelCase)

```
class Rectangle {
    double width;
    double height;
}
```

- список полей класса
  (атрибутов, переменных)
- тип и имя

Объект Rectangle - экземпляр класса

ИТМО

```
class Main {
  static double rectangleArea(Rectangle rectangle) {
    return rectangle.width * rectangle.height;
  }
  public static void main(String... args) {
    Rectangle[] rectangles = { new Rectangle(), new Rectangle() };
      rectangles[0].width = 1.0; rectangles[0].height = 4.0;
      rectangles[1].width = 5.64; rectangles[1].height = 5.64;
    for (Rectangle r : rectangles) {
      System.out.printf("Стороны: %s и %s%n", r.width, r.height);
      System.out.println("Площадь: " + rectangleArea(r));
    }
  }
}
```

```
class Rectangle {
  double width;
  double height;
}
```

Жизненный цикл объекта

- Создание
  - выделение памяти в куче с помощью оператора new
- Инициализация
  - задание значений полей объекта (конструктор)
- Использование
  - получение и установка значений полей, вызов методов
- Недостижимость
  - не осталось ни одной ссылки на объект
- Удаление
  - удаляется сборщиком мусора с освобождением памяти

Создание объекта - new

- new - ключевое слово
  - выделяет память для объекта
  - инициализирует поля объекта
  - вызывает конструктор
  - возвращает ссылку на готовый объект

Действия с объектами

ИТМО

```
Rectangle r1;
r1 = new Rectangle()
r1.width = 7.0;
r1.height = 3.4;
a = r1.width * r1.height;
a = rectangleArea(r1);
```

- В стеке - ссылка r1
- В куче - новый объект (0,0)
- Поле width = 7.0
- Поле height = 3.4
- r1 - аргумент метода

null - не инициализированная ссылка

ИТМО

```
Rectangle r1;
~~r1 = new Rectangle()~~
r1.width = 7.0;
r1.height = 3.4;
a = r1.width * r1.height;
a = rectangleArea(r1);
```

- В стеке - ссылка r1 = null
- ~~В куче - новый объект (0,0)~~
- NullPointerException

Объекты в памяти

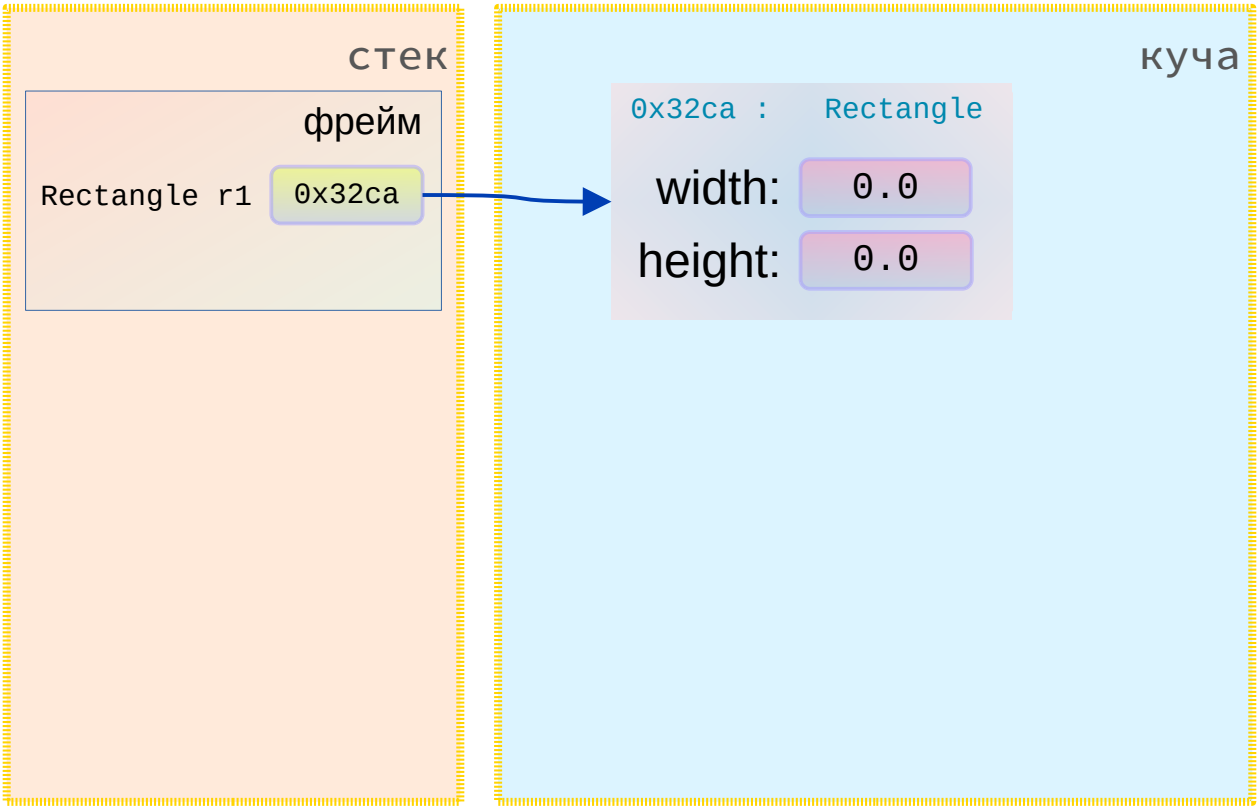
ИТМО

```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```

Объекты в памяти

ИТМО

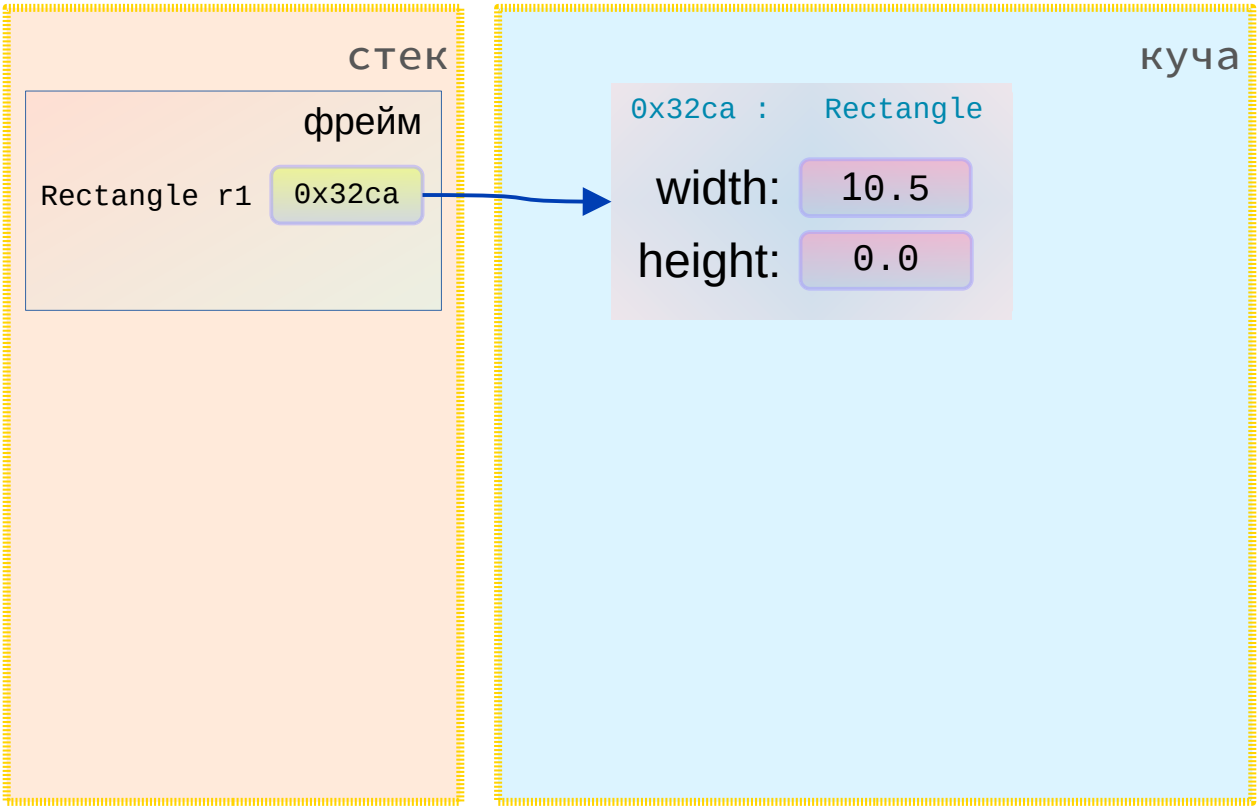
```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



Объекты в памяти

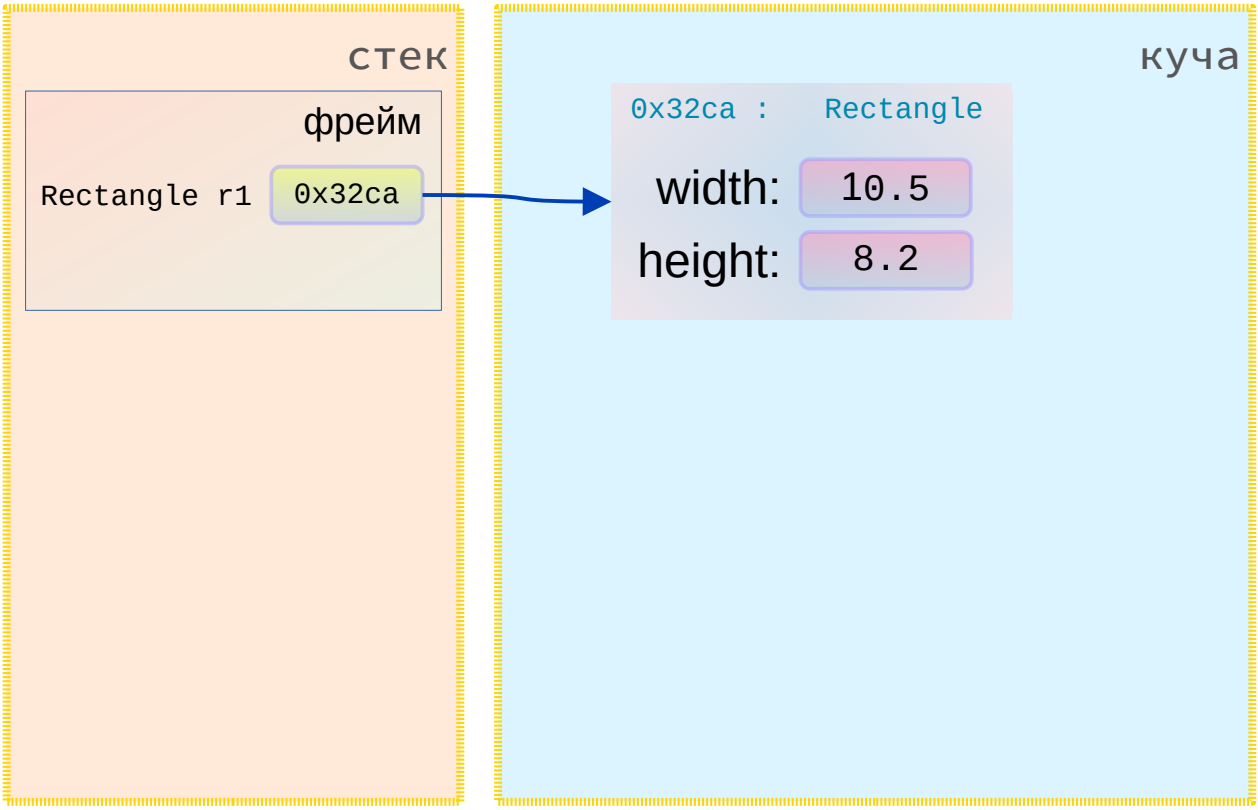
ИТМО

```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



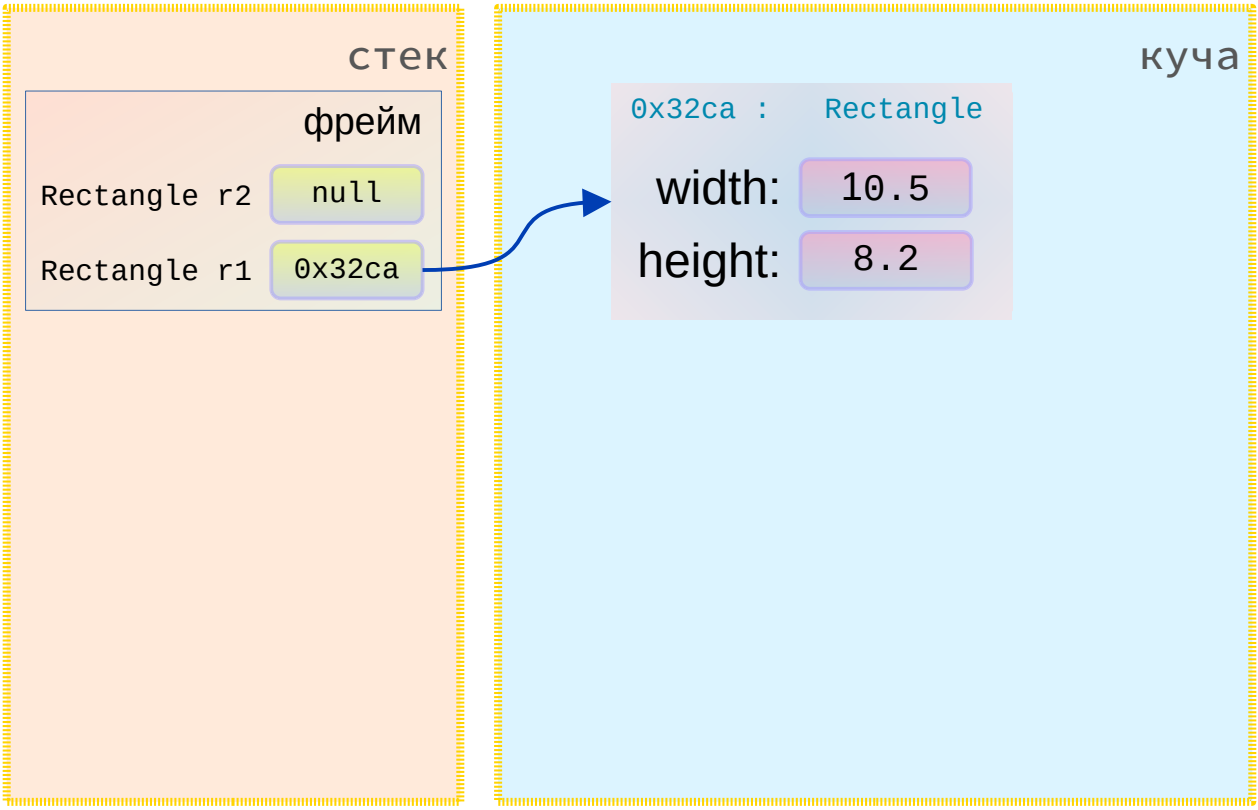
Объекты в памяти

```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



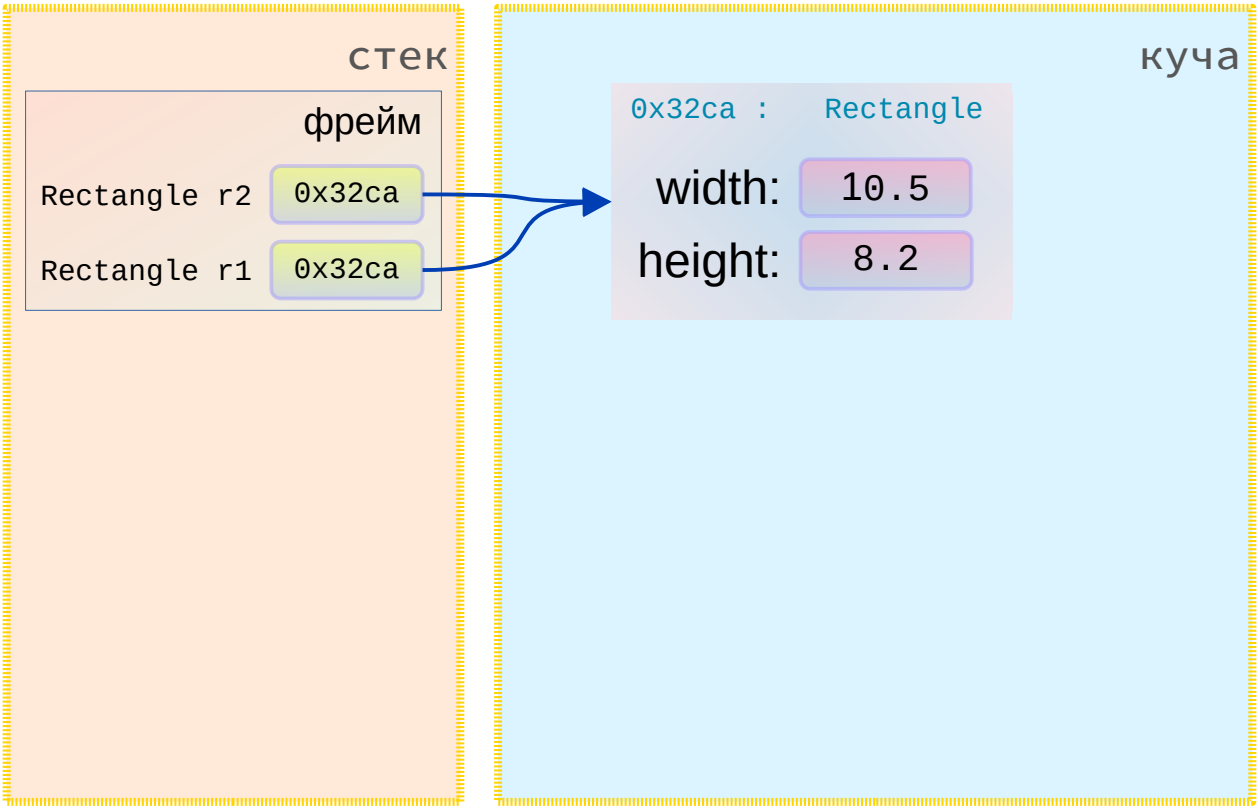
Объекты в памяти

```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



Объекты в памяти

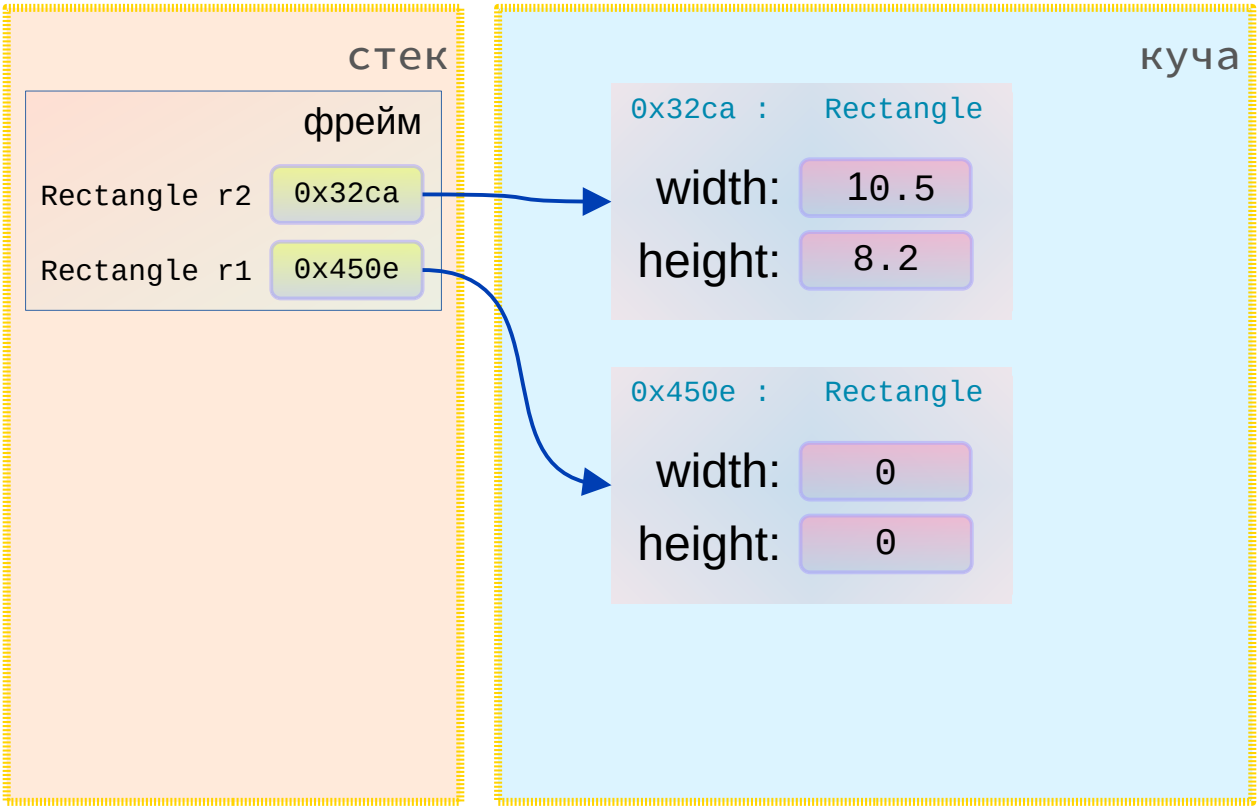
```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



Объекты в памяти

ИТМО

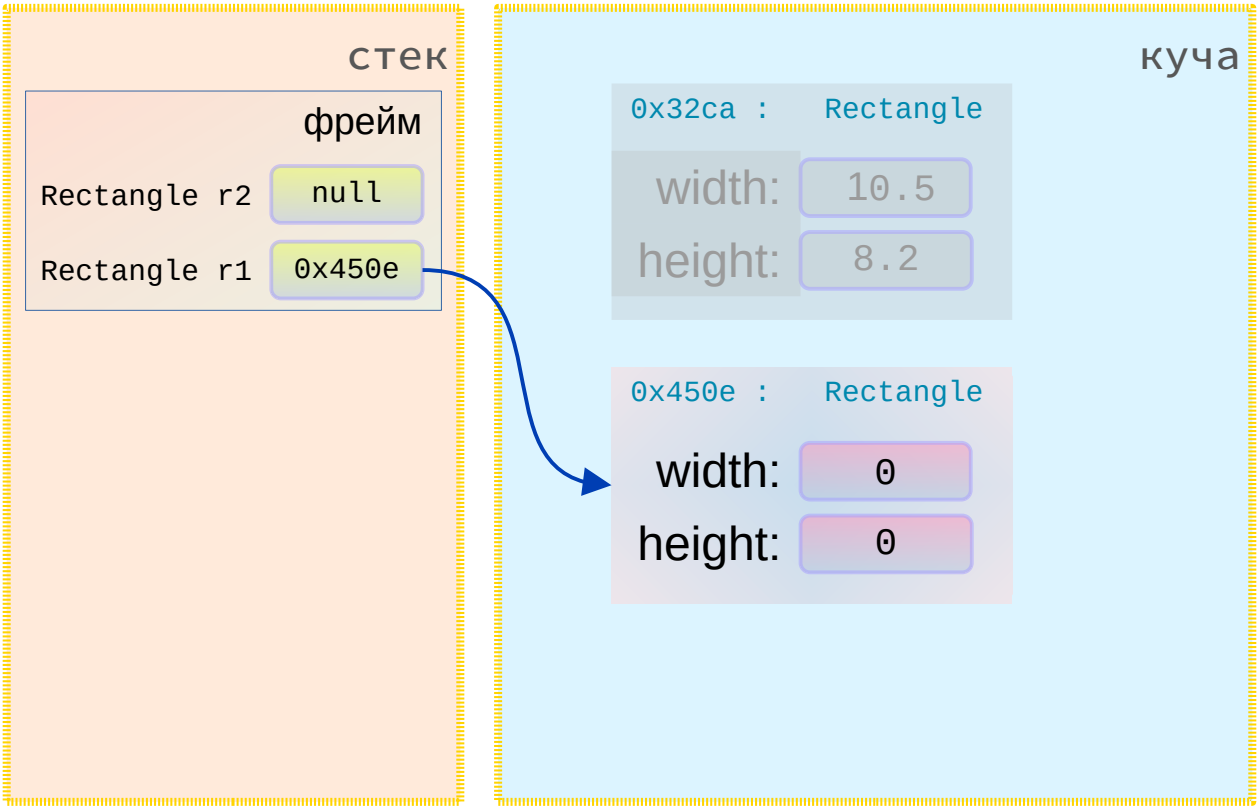
```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



Объекты в памяти

ИТМО

```
Rectangle r1;
r1 = new Rectangle();
r1.width = 10.5;
r1.height = 8.2;
Rectangle r2;
r2 = r1;
r1 = new Rectangle();
r2 = null;
```



Сборка мусора

ИТМО

- Идентификация неиспользуемых объектов
  - подсчет ссылок на объект
  - сканирование кучи и поиск недостижимых объектов
  - циклические зависимости и утечки памяти
- Освобождение памяти
  - участки памяти помечаются как свободные
- Сборщик мусора периодически запускаеися автоматически

Объекты в качестве полей

ИТМО

- Полем объекта может быть ссылка на другой объект

```
class Rectangle {
  double width;
  double height;
  Point corner;
}
```

```
class Point {
  double x;
  double y;
}
```

```
Rectangle r1 = new Rectangle();
r1.width = 5.0;
r1.height = 7.0;
r1.corner = new Point();
r1.corner.x = 30.0;
r1.corner.y = 22.5;
```

Объекты в качестве полей

ИТМО

- Такое отношение между классами называется **композиция**

```
class Rectangle {
  double width;
  double height;
  Point corner;
}
```

```
class Point {
  double x;
  double y;
}
```

```
Point p1 = new Point();
p1.x = 30.0;
p1.y = 22.5;
Rectangle r1 = new Rectangle();
r1.width = 5.0;
r1.height = 7.0;
r1.corner = p1;
```

Массив объектов

ИТМО

- Объекты могут быть элементами массива

```
class Rectangle {
  double width;
  double height;
}
```

```
Rectangle r1 = new Rectangle();
r1.width = 5.0;
r1.height = 7.0;
Rectangle r2 = new Rectangle();
r2.width = 9.64;
r2.height = Math.PI;
Rectangle[] array = { r1, r2 };
```

Поля объекта и поля класса

ИТМО

- Поле объекта (экземпляра) - instance field
  - Свое различное значение у каждого объекта
  - Получает значение при инициализации объекта
  - Не зависит и не влияет на поля других объектов
  - Значение хранится в куче внутри объекта
- Статическое поле (поле класса) - **static** field
  - Одно общее значение на весь класс
  - Получает значение в момент загрузки класса в память
  - Доступ к полю через класс (нагляднее) или через объект
  - Значение хранится в особом разделе кучи - Metaspace

Статические поля

ИТМО

```
class Rectangle {
  double width;
  double height;
  static int numOfSides = 4;
}
```

```
class Person {
  int id;
  String name;
  ...
  static int maxID = 0;
}
```

```
Rectangle r1 = new Rectangle();
int x = r1.numOfSides; // 4
x = Rectangle.numOfSides; // 4
```

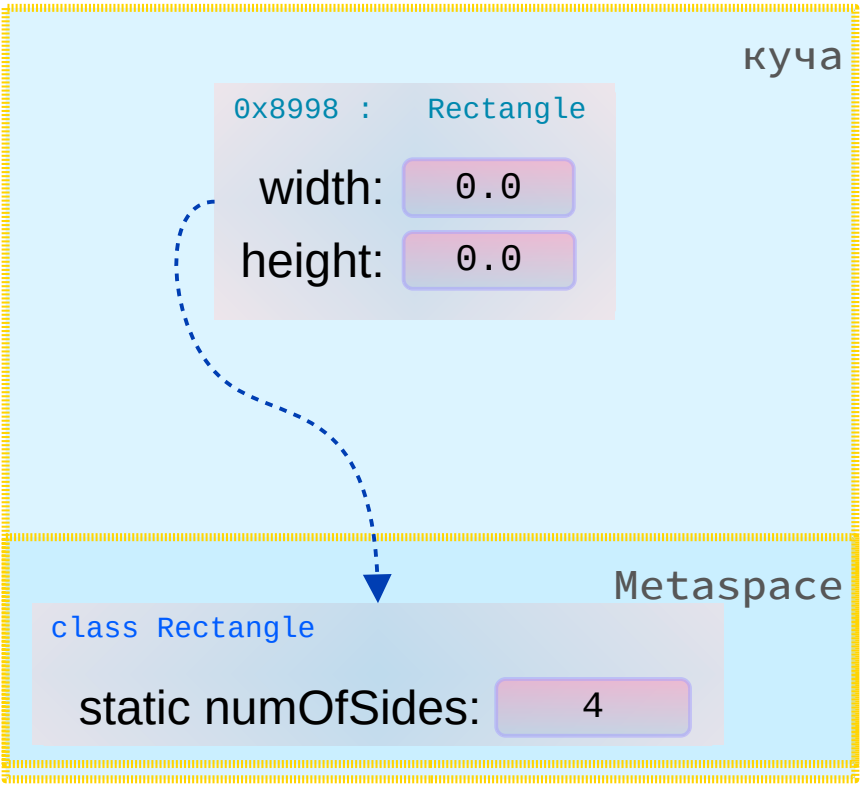
```
Person p1 = new Person();
p1.id = Person.maxID++; // 1
Person p2 = new Person();
p2.id = p2.maxID++; // 2
```

Статические поля

ИТМО

```
class Rectangle {
  double width;
  double height;
  static int numOfSides = 4;
}
```

```
Rectangle r1 = new Rectangle();
int x = r1.numOfSides; // 4
x = Rectangle.numOfSides;
```



Статические поля и константы

ИТМО

```
class Rectangle {
  double width;
  double height;
  static final int NUM_OF_SIDES = 4;
}
```

```
class Person {
  int id;
  String name;
  ...
  static int maxID = 0;
}
```

```
Rectangle r1 = new Rectangle();
int x = r1.NUM_OF_SIDES; // 4
x = Rectangle.NUM_OF_SIDES; // 4

double result = Math.PI / Math.E;
```

```
Person p1 = new Person();
p1.id = Person.maxID++; // 1
Person p2 = new Person();
p2.id = p2.maxID++; // 2
```

Методы

- Методы можно связать с классом
  - Метод rectangleArea нужен только прямоугольнику
  - Метод circleArea считает площадь только круга
- Методы определяют поведение объектов класса
- Методы - множество операций над объектами

Класс Rectangle + метод для объектов класса

ИТМО

- метод area() считает площадь
- значения полей передавать не нужно, они есть внутри объекта

```
class Rectangle {
  double width;
  double height;

  double area() {
    return width * height;
  }
}
```

Класс - собственный тип данных

ИТМО

- Класс - ссылочный тип
- В объекте хранятся конкретные значения полей
- Переменная хранит ссылку на объект в памяти
- Методы задают набор операций

```
class Rectangle {
  double width;
  double height;

  double area() {
    return width * height;
  }
}
```

```
Rectangle rect1 = new Rectangle();
rect1.width = 10.0;
rect1.height = 3.5;
System.out.println(rect1.area());
```

- width: 10.0
- height: 3.5

Классы и методы

ИТМО

```
class Main {
  public static void main(String... args) {
    Rectangle[] rectangles = {
      new Rectangle(),
      new Rectangle() };
    rectangles[0].width = 1.0;
    rectangles[0].height = 4.0;
    rectangles[1].width = 5.64;
    rectangles[1].height = 5.64;
    for (Rectangle r : rectangles) {
      System.out.printf("Стороны: %s и %s%n", r.width, r.height);
      System.out.println("Площадь: " + r.area());
    }
  }
}
```

```
class Rectangle {
  double width;
  double height;

  double area() {
    return width * height;
  }
}
```

Ссылка this (метод vs функция)

ИТМО

```
class Rectangle {
  double       width;
  double       height;

  double area(                    ) {
    return       width *       height;
  }
}
```

```
for (Rectangle r : rectangles) {

  r.area();

}
```

Ссылка this (метод vs функция)

ИТМО

```
class Rectangle {
  double this.width;
  double this.height;

  double area( Rectangle this ) {
    return this.width * this.height;
  }
}
```

```
for (Rectangle r : rectangles) {
  r.area();
}
```

```
Rectangle.area( r );
```

Ссылка this (метод vs функция)

ИТМО

```
class Rectangle {
  double this.width;
  double this.height;

  double area( Rectangle this ) {
    return this.width * this.height;
  }
}
```

```
for (Rectangle r : rectangles) {

  r.area();

}
```

```
Rectangle.area( r );
```

Ссылка this и статические методы

- this - ссылка на текущий объект, у которого вызван метод
- Неявно передается первым параметром во все методы
  - (кроме статических)
- Статические методы (методы класса)
  - Не получают ссылку this
  - Можно вызывать по имени класса (без объекта)
  - Нельзя вызывать из нестатических методов
  - Не могут менять состояние объектов

Статический контекст

ИТМО

- Статические поля и методы относятся к классу
- Они не требуют наличия объекта (объекта может не быть)
- Правила
  - static может обращаться только к static
  - static **не может** обращаться к non-static
  - non-static может обращаться к static
  - non-static может обращаться к non-static

объект точно есть!

Примеры статических методов (Java API)

ИТМО

- Math - все методы статические - класс-утилита
  - математические функции не зависят от состояния
- LocalTime
  - static LocalTime now()
    - метод-фабрика, создает объект и возвращает его
    - на момент вызова метода объекта еще нет - static
  - toString()
    - метод возвращает стороковое представление объекта

```
LocalTime current = LocalTime.now();
System.out.println(current.toString());
```

Пакеты (packages)

ИТМО

- Пакет - логическое объединение классов
- Пакет указывается в первой строк файла (**package** *name*)
- Пакет образует пространство имен - в одном пакете не может быть классов с одним именем
- Пакет соответствует каталогу с файлами *.class

```
ru
└itmo
  └mypackage
    └MyClass.class
```

```
package ru.itmo.mypackage;
class MyClass { }

// ru.itmo.mypackage.MyClass - полностью квалифицированное имя класса
```

Имя пакета

ИТМО

- Именованные пакеты
  - Полный вариант (для больших проектов)
    - домен организации в обратном порядке
    - se.itmo.ru → ru.itmo.se.mypackage
  - Сокращенный вариант (для небольших проектов)
    - одно слово
    - pokemons, moves, client, server
  - Безымянный пакет (простой вариант)
    - если имя пакета не указано

Импорт

ИТМО

- Импорт добавляет имя класса из другого пакета в текущее пространство имен
- Импорт ничего не подключает, не увеличивает размер кода
- Импорт можно не писать, если использовать полные имена
- Импорт не нужен для классов своего пакета
- Импорт не нужен для пакета java.lang

Импорт

ИТМО

```
package mypackage;
import java.lang.*;

import java.util.Arrays;
import java.time.*;
import static java.lang.Math.*;
...
myTime =
  LocalTime.now();

Arrays.sort(myArray);

y = sin(x) * cos(x);
```

```
package mypackage;
import java.lang.*;




...
myTime =
  java.time.LocalTime.now();

java.util.Arrays.sort(myArray);

y = Math.sin(x) * Math.cos(x);
```

Исходный код классов

ИТМО

- В одном файле - один public класс
- Имя файла = имя public класса + .java
  - Если нет public классов - любое имя
- Первая инструкция в файле - package
- Далее - import
- Далее - декларации классов

Базовые принципы ООП

- Абстракция
- **Инкапсуляция**
- Наследование
- Полиморфизм

- Encapsulation
- Объединение данных и кода для их обработки в единую структуру - класс, с реализацией контроля доступа к внутренним элементам класса через строго определенный интерфейс взаимодействия.

Базовые принципы ООП

- Абстракция
- **Инкапсуляция**
  - **Сокрытие информации**
- Наследование
- Полиморфизм

- Information hiding
- Сокрытие деталей реализации внутренних элементов класса, открыт только интерфейс взаимодействия
  - возможность менять реализацию класса без воздействия на остальные элементы системы

Модификаторы для полей, методов, классов

ИТМО

• private	• Доступ только своим (внутри класса)
• Здесь могла бы быть ваша реклама	• Доступ классам того же пакета
• protected	• Доступ классам пакета и наследникам
• public	• Доступ всем (почти)
• final (поле)	• Нельзя присвоить новое значение
• static (поле)	• Значение поля - общее для класса
• static (метод)	• Метод связан не с объектом, а с классом

Модификаторы для полей, методов, классов

ИТМО

• private	• Доступ только своим (внутри класса)
• **Здесь нет никакого модификатора**	• Доступ классам того же пакета
• protected	• Доступ классам пакета и наследникам
• public	• Доступ всем (почти)
• final (поле)	• Нельзя присвоить новое значение
• static (поле)	• Значение поля - общее для класса
• static (метод)	• Метод связан не с объектом, а с классом

Модификаторы для полей, методов, классов

ИТМО

• private	• Доступ только своим (внутри класса)
• (package-private)	• Доступ классам того же пакета
• protected	• Доступ классам пакета и наследникам
• public	• Доступ всем (почти)
• final (поле)	• Нельзя присвоить новое значение
• static (поле)	• Значение поля - общее для класса
• static (метод)	• Метод связан не с объектом, а с классом

Модификаторы для полей, методов, классов

ИТМО

• private	• Доступ только своим (внутри класса)
•	• Доступ классам того же пакета
• protected	• Доступ классам пакета и наследникам
• public	• Доступ всем (почти)
• final (поле)	• Нельзя присвоить новое значение
• static (поле)	• Значение поля - общее для класса
• static (метод)	• Метод связан не с объектом, а с классом

Публичные и приватные поля и методы

ИТМО

```
class Main {
  public static void main(String... args) {
    Rectangle[] rectangles = {
      new Rectangle(),
      new Rectangle() };
    rectangles[0].width = - 1.0;
    rectangles[0].height = 0.0;
    rectangles[1].width = 5.64;
    rectangles[1].height = 5.64;
    for (Rectangle r : rectangles) {
      System.out.printf("Стороны: %s и %s%n", r.width, r.height);
      System.out.println("Площадь: " + r.area());
    }
  }
}
```

Некорректные значения

```
class Rectangle {
  double width;
  double height;

  double area() {
    return width * height;
  }
}
```

Публичные и приватные поля и методы

ИТМО

- внутреннее закрытое состояние
  - детали реализации не имеют значения для внешних пользователей класса
  - реализация можно менять как угодно, пока не затронут внешний интерфейс

```
class Rectangle {
  private double width;
  private double height;

  public double area() {
    return width * height;
  }
}
```

- внешний открытый интерфейс
  - фиксируется в начале разработки и далее не должен меняться
  - обеспечивает внешним пользователям возможность управлять объектами класса в контролируемых пределах

Доступ к полям - getter / setter

ИТМО

- Стандартная практика
  - поля закрытые (private)
  - методы открытые (public)
    - T getField() - getter (accessor)
    - void setField(T value) - setter (mutator)
    - boolean isActive() - is-method (predicate) для полей типа boolean
- Геттер может выводить значение в определенном формате
- Сеттер может проверять корректность значения, менять его формат или запретить изменение значения

Доступ к полям - getter / setter

ИТМО

```
class Rectangle {
  private double width;
  private double height;
  public double getWidth() { return width; }
  public double getHeight() { return height; }
  public void setWidth(double width) {
    if (isSideValid(width) { this.width = width; }
  }
  public void setHeight(double height) {
    if (isSideValid(height) { this.height = height; }
  }
  private boolean isSideValid(double side) {
    return side > 0;
  }
}
```

Доступ к полям - getter / setter

ИТМО

```
class Rectangle {
  private double width;
  private double height;
  public double getWidth() { return width; }
  public double getHeight() { return height; }
  public void setSides(double width, double height) {
    if (width > 0 && height > 0) {
      this.width = width; this.height = height;
    } else { System.err.println("Стороны должны быть > 0!"); }
  }
  public double area() { return width * height; }
  public String descr() { return "Прямоугольник со сторонами " +
    width + " и " + height; }
}
```

Публичные и приватные поля и методы

ИТМО

```
class Main {
  public static void main(String... args) {
    Rectangle[] rectangles = { new Rectangle(),
                               new Rectangle() };
    rectangles[0].setSides(1.0, 4.0);
    rectangles[1].setSides(5.64, 5.64);
    for (Rectangle r : rectangles) {
      System.out.println(r.descr()+"\nПлощадь: " + r.area());
    }
  }
}
```

Конструкторы

ИТМО

- Конструктор - специальный метод для инициализации полей объекта
- Имя совпадает с именем класса
- Нет возвращаемого типа
- Вызывается оператором new

```
class Rectangle {
  private double width;
  private double height;

  public Rectangle(double w,
                   double h) {
    width = w;
    height = h;
  }
}
```

Конструкторы

```
class Main {
  public static void main(String... args) {
    Rectangle[] rectangles = { new Rectangle(),
                               new Rectangle() };
    rectangles[0].setSides(1.0, 4.0);
    rectangles[1].setSides(5.64, 5.64);
    for (Rectangle r : rectangles) {
      System.out.println(r.descr()+"\nПлощадь: " + r.area());
    }
  }
}
```

Вызов конструктора

Программа перестала работать - почему?

ИТМО

```
class Main {
  public static void main(String... args) {
    Rectangle[] rectangles = { new Rectangle(),
                               new Rectangle() };
    rectangles[0].setSides(1.0, 4.0);
    rectangles[1].setSides(5.64, 5.64);
    for (Rectangle r : rectangles) {
      System.out.println(r.descr()+"\nПлощадь: " + r.area());
    }
  }
}
```

Ошибка!

Конструктор по умолчанию

- Если в классе не определен конструктор, компилятор автоматически добавит пустой конструктор без параметров
- Если определен любой конструктор, то конструктор по умолчанию добавляться не будет

```
class Rectangle {
  private double width;
  private double height;
  // если явного конструктора нет

}
```

```
  public Rectangle() {
  // Конструктор по умолчанию
  }
```

Неявный конструктор по умолчанию

ИТМО

```
class Rectangle {
  private double width;
  private double height;

  public Rectangle(double w,
                   double h) {
    width = w;
    height = h;
  }
}
```

```
class Rectangle {
  private double width;
  private double height;
  // если явного конструктора нет

}
```

```
  public Rectangle() {
  // Конструктор по умолчанию
  }
```

Вызов конструктра с параметрами

ИТМО

```
class Main {
  public static void main(String... args) {
    Rectangle[] rectangles = {
      new Rectangle(1.0, 4.0),
      new Rectangle(5.64, 5.64) };
    for (Rectangle r : rectangles) {
      System.out.println(r.descr()+"\nПлощадь: " + r.area());
    }
  }
}
```

Убираем лишнее

ИТМО

```
class Rectangle {
  private double width, height;

  public double area() {
    return width * height;
  }
  public String descr() {
    return "Rectangle: "+width+"x"+height;
  }
  public Rectangle(double w, double h) {
    width = w; height = h;
  }
}
```

Перегрузка конструкторов

ИТМО

```
class Rectangle {
  private double width, height;

  public double area() {
    return width * height;
  }
  public String descr() {
    return "Rectangle: "+width+"x"+height;
  }
  public Rectangle(double w, double h) {
    width = w; height = h;
  }
  public Rectangle() { this(1.0, 1.0); }
}
```

вызов конструктора внутри другого конструктора

Еще про ссылку this

ИТМО

- Внутри класса ссылка this ссылается на текущий объект (объект, у которого был вызван метод или конструктор)
- С помощью this можно обращаться
  - к полям класса this.width
  - к методам класса this.area()
  - к конструкторам this(0, 0)

Перегрузка методов и конструкторов

ИТМО

- Перегрузка (overloading) методов или конструкторов - если у них одинаковые имена, но разный набор параметров (разный тип, разное количество, разный порядок)
- Компилятор выбирает метод (или конструктор), который подходит по типу, количеству и порядку параметров с учетом возможного расширяющего приведения типа

```
byte  byteX  = 1;
short shortX = 2;
long  longX  = 3;
```

```
method(byteX);
method(shortX);
method(longX);
```

```
void method(byte x) { ... }

void method(int x) { ... }
```

Перегрузка методов и конструкторов

ИТМО

- Перегрузка (overloading) методов или конструкторов - если у них одинаковые имена, но разный набор параметров (разный тип, разное количество, разный порядок)
- Компилятор выбирает метод (или конструктор), который подходит по типу, количеству и порядку параметров с учетом возможного расширяющего приведения типа

```
byte  byteX  = 1;
short shortX = 2;
long  longX  = 3;
```

```
method(byteX);
method(shortX);
method(longX);
```

```
void method(byte x) { ... }

void method(int x) { ... }
```

UML-диаграммы классов

ИТМО

Rectangle
- width : double - height : double
+ Rectangle(w:double, h:double) + area() : double + descr() : String

```
+   public
-   private
~   (package-private)
#   protected
```

```
class Rectangle {

  private double width, height;



  public double area() {

    return width * height;

  }

  public String descr() {

    return "Rectangle: "+width+"x"+height;

  }

  public Rectangle(double w, double h) {

    width = w; height = h;

  }

}
```

Блоки инициализации

ИТМО

```
class Test {
  int value;
  String color;
  public Test() {
    color = "White";
    System.out.println(color);
  }
  public Test(int i) {
    color = "White";
    System.out.println(color);
    value = i;
  }
}
```

```
class Test {
  int value;
  String color;
  public Test() {
  }
  public Test(int i) {
    value = i;
  }
  { color = "White";
    System.out.println(color);
  }
}
```

instance initializer

Блоки инициализации

ИТМО

```
class Test {
  int value;
  String color;
  public Test() {
    color = "White";
    System.out.println(color);
  }
  public Test(int i) {
    color = "White";
    System.out.println(color);
    value = i;
  }
}
```

```
class Test {
  int value;
  String color = "White";
  public Test() {
  }
  public Test(int i) {
    value = i;
  }
  { System.out.println(color); }
}
```

instance initializer

Блоки инициализации

```
class Test {
  static int[] array = {1,2,3};
}
```

```
class Test {

  static final int SIZE=10000;
  static int[] array;

  static {
    array = new int[SIZE];
    for (var i=0; i<SIZE; i++) {
      array[i] = i*i;
    }
  }
}
```

static initializer

Блоки инициализации - порядок выполнения

ИТМО

- При загрузке класса
  - инициализация статических переменных
  - выполнение статических блоков инициализации
- При создании объекта
  - инициализация переменных экземпляра
  - выполнение блоков инициализации экземпляра
  - выполнение кода конструктора

Конструктор private?

ИТМО

- Что будет, если конструктор будет private?

Конструктор private?

- Что будет, если конструктор будет private?
  - Контроль процесса создания объектов
  - Для создания объектов - метод-фабрика
- Объект только один или ограниченное количество
- Дополнительные проверки при создании объекта

Неизменяемые объекты

ИТМО

- Поля - private final
- Будет ли объект неизменяемым?

```
class Rectangle {
  private final double width;
  private final double height;

  public Rectangle(
    double w, double h) {
    width = w; height = h;
    origin = new Point(10,10);
  }
}
```

Неизменяемые объекты

ИТМО

- Поля - private final
- Будет ли объект неизменяемым?

```
class Point {
  private double x;
  private double y;
  public Point(double x,
               double y) {
    this.x = x;
    this.y = y; }
  public void reset() {
    x = 0; y = 0; }
}
```

```
class Rectangle {
  private final double width;
  private final double height;

  public Rectangle(
    double w, double h) {
    width = w; height = h;
    origin = new Point(10,10);
  }

  private final Point origin;
  public Point getOrigin() {
    return origin;
  }
}
```

Неизменяемые объекты

ИТМО

- Поля - private final
- Будет ли объект неизменяемым?
- Нет, если в цепочке не всё - final

```
class Point {
  private double x;
  private double y;
  public Point(double x,
               double y) {
    this.x = x;
    this.y = y; }
  public void reset() {
    x = 0; y = 0; }
}
```

```
class Rectangle {
  private final double width;
  private final double height;

  public Rectangle(
    double w, double h) {
    width = w; height = h;
    origin = new Point(10,10);
  }

  private final Point origin;
  public Point getOrigin() {
    return origin;
  }
}
```

```
Rectangle myRect = new Rectangle(300,200);
myRect.getOrigin().reset();
```

Неизменяемые объекты

ИТМО

- Поля - private final
- Будет ли объект неизменяемым?
- Вот теперь - да!

```
class Point {
  private final double x;
  private final double y;
  public Point(double x,
               double y) {
    this.x = x;
    this.y = y; }
  public void reset() {
    x = 0; y = 0; }
}
```

```
class Rectangle {
  private final double width;
  private final double height;

  public Rectangle(
    double w, double h) {
    width = w; height = h;
    origin = new Point(10,10);
  }

  private final Point origin;
  public Point getOrigin() {
    return origin;
  }
}
```

```
Rectangle myRect = new Rectangle(300,200);
myRect.getOrigin().reset();
```

Enum

ИТМО

- Тип - перечисление
- Список именованных констант
- Набор значений небольшой и не меняется

```
enum Month {
   JANUARY, FEBRUARY,
   MARCH, APRIL, MAY,
   JUNE, JULY, AUGUST,
   SEPTEMBER, OCTOBER,
   NOVEMBER, DECEMBER
}
```

```
Month current = Month.OCTOBER;

System.out.println(current)
// OCTOBER - не String
```

Enum - продвинутое перечисление

ИТМО

```
enum Month {
   JANUARY, FEBRUARY,
   MARCH, APRIL, MAY,
   JUNE, JULY, AUGUST,
   SEPTEMBER, OCTOBER,
   NOVEMBER, DECEMBER
}
```

```
Month current = Month.OCTOBER;

current.ordinal() // 9

current.name()    // "OCTOBER" (String)

Month.values()
/* { JANUARY, FEBRUARY, MARCH, APRIL,
MAY, JUNE, JULY, AUGUST, SEPTEMBER,
OCTOBER, NOVEMBER, DECEMBER } */

Month.valueOf("MAY") // Month.MAY
```

Enum - еще более продвинутое перечисление

ИТМО

```
enum Month {
  JANUARY(31),FEBRUARY(28),MARCH(31),APRIL(30),MAY(31),JUNE(30),
  JULY(31),AUGUST(31),SEPTEMBER(30),OCTOBER(31),NOVEMBER(30),DECEMBER(31);

  private int days;

  public Month(int days) {
    this.days = days;
  }

  public numberOfDays() {
    return days;
  {
}
```

Enum - еще более продвинутое перечисление

ИТМО

```
enum Month {
  JANUARY(31),FEBRUARY(28),MARCH(31),APRIL(30),MAY(31),JUNE(30),
  JULY(31),AUGUST(31),SEPTEMBER(30),OCTOBER(31),NOVEMBER(30),DECEMBER(31);

  private int days;

  public Month(int days) {
    this.days = days;
  }

  public numberOfDays() {
    return days;
  {
}
```

```
Month current = Month.OCTOBER;

current.numberOfDays() // 31
```

Вопросы?