Платформа и язык программирования

Java for autotesters

Что такое Java?

- Java - это платформа и язык программирования

- Основные особенности:
  - Объектная ориентированность
  - Переносимость
  - Простота
  - Безопасность
  - Поддержка многопоточности

История

- Авторы: **Джеймс Гослинг (James Gosling)**, Патрик Ноутон (Patrick Naughton), Крис Варт (Chris Warth), Эд Франк (Ed Frank), Майк Шеридан (Mike Sheridan), etc.

- 1991 – начало работы, первая версия – Oak
- **1995 – официальный релиз Java 1.0**
- **1997 – Java 1.1**
- 1999 – Java 1.2 (Java2)
- 2000 – Java2 1.3
- 2001 – Java2 1.4
- **2004 – Java 5.0**
- 2006 – Java 6
- **2011 – Java 7**
- 2014 – Java 8 ?

Платформа Java

- **Платформа Java** отличается тем, что выполняется **поверх других платформ** и **не зависит от** конкретной **операционной системы**

- Платформа Java состоит из двух компонентов:
  - Виртуальная Java Машина (JVM)
  - Интерфейсы прикладного программирования Java (Java API)

- JVM – это основа платформы Java
- Существуют версии JVM для различных платформ

Платформа Java

Java Enterprise Edition

Java Standard Edition

CDC

CLDC

Java Micro Edition

Java Card

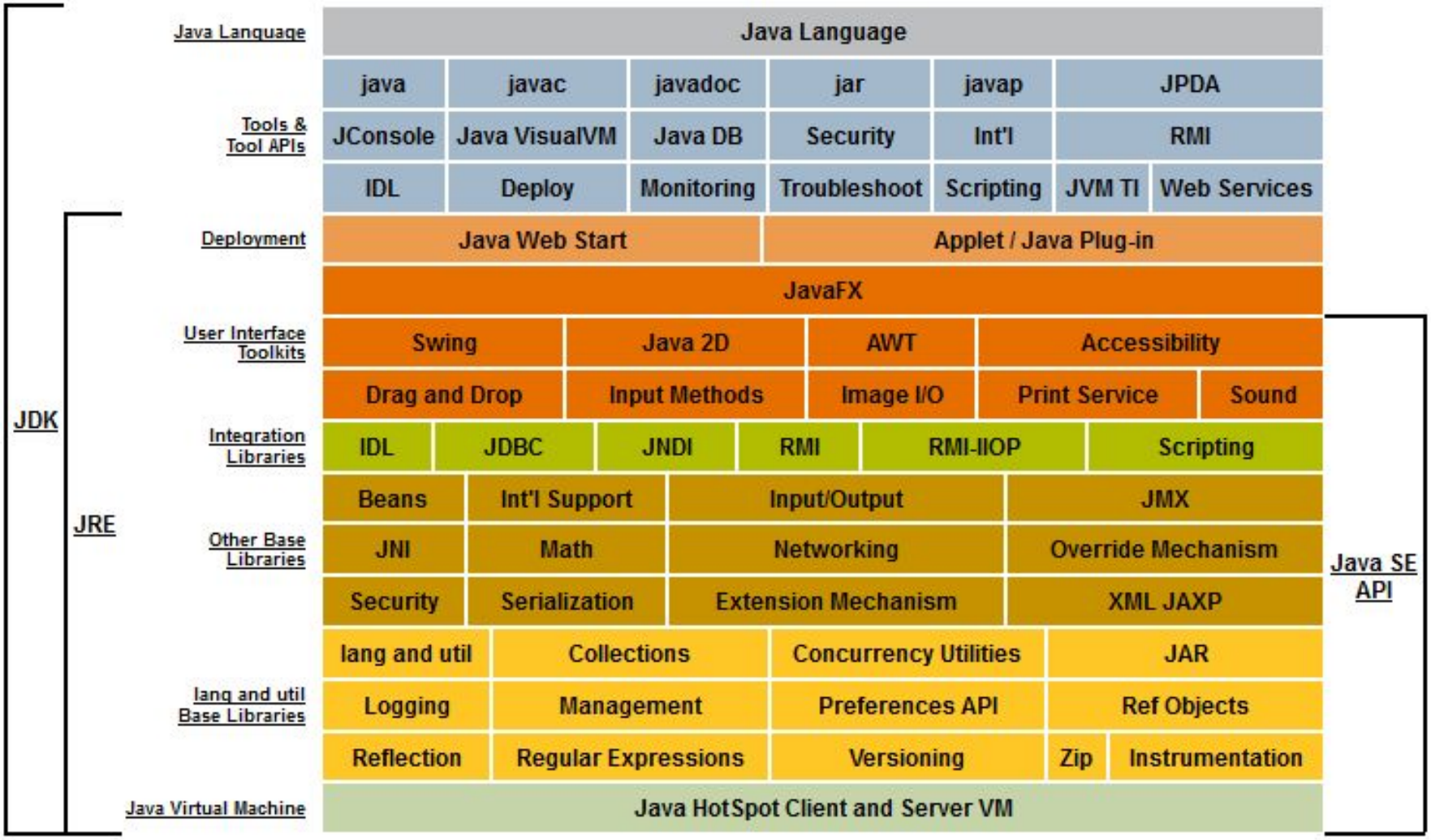
Язык Java

Java Virtual Machine

KVM

Card VM

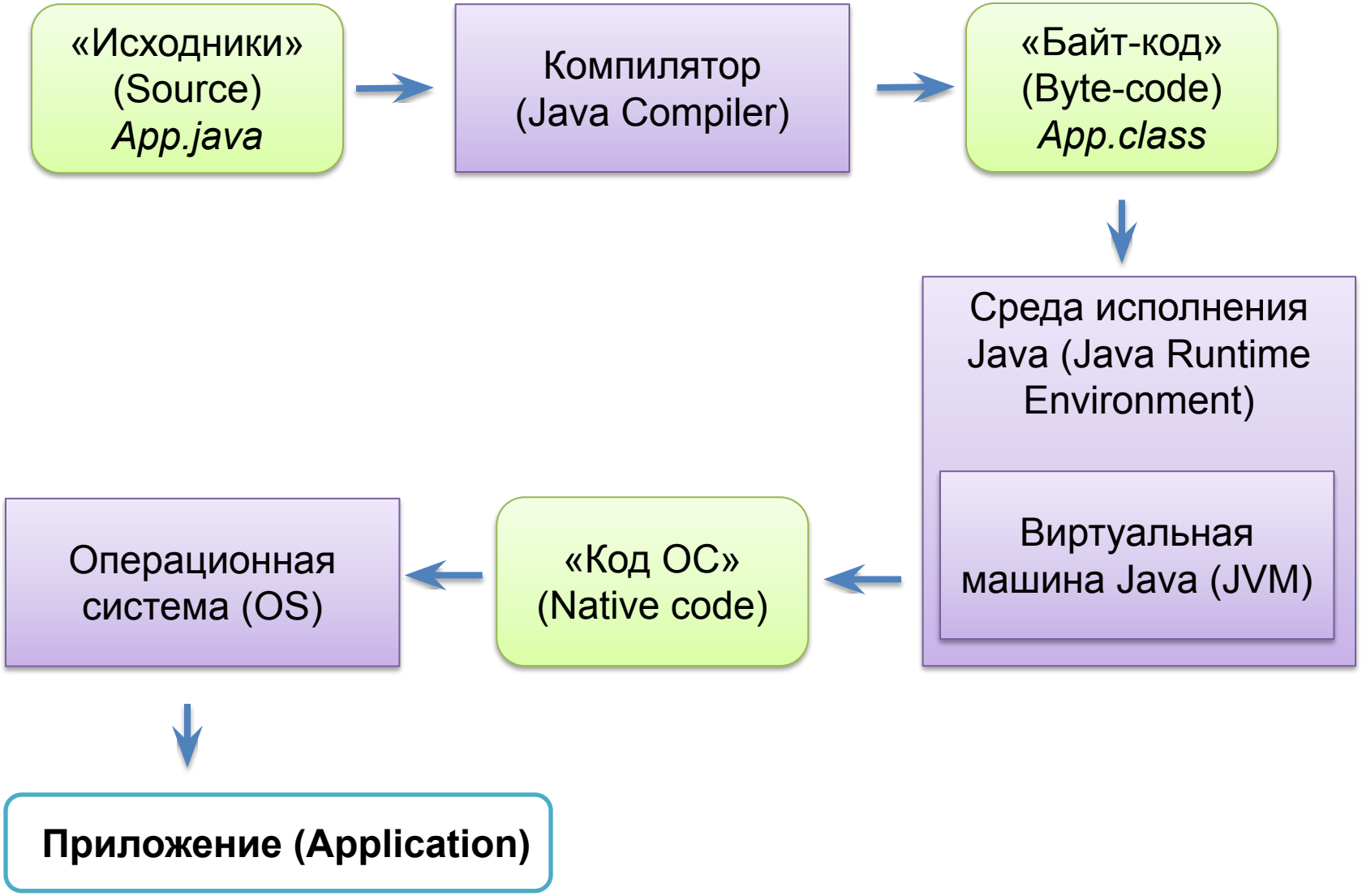
Платформа Java



Особенности платформы

Свойства	Достоинства	Недостатки
Виртуальная машина	Кроссплатформенность	Скорость
Отсутствие прямой работы с памятью	Уменьшение количества ошибок	Скорость
Сборка мусора	Уменьшение количества ошибок	Скорость
Байткод	Простота отладки, рефлексия	Проблема защиты кода, потеря скорости
JIT-компилятор	Увеличение скорости	
Стандартная библиотека	Ускорение разработки, уменьшение количества кода	

Процесс разработки программного обеспечения



Основы языка Java

- Hello, world!
- Переменные
- Операции
- Выражения и блоки
- Операторы управления
- Метод main
- Комментарии

Hello, world!

```java
/**
 * This is our first Java class.
 */
public class HelloWorld {

    public static void main(String[] args) {
        // Just printing of text "Hello, world!"
        System.out.println("Hello, world!");
    }
}
```

Терминология

- **Алфавит** - набор допустимых символов
  - Практически все символы Unicode
  - Реально используют только ANSI символы
- **Лексема (token)** - неделимый элемент языка
  - Отделяются пробелами и знаками препинания
- **Идентификатор** - лексема, обозначающая некоторый объект, имеющий смысл
  - Аналог слова в естественном языке
- **Ключевое слово** - идентификатор, зарезервированный языком программирования
  - Имеет некоторый специальный смысл

Переменные

- **Переменная** – объект программы, имеющий имя, занимающий некоторый участок памяти и хранящий некоторые данные

- У переменной есть
  - Имя
  - Тип
  - Область видимости

```
тип имя = инициализатор;
```

Типы данных

```
int count = 1;
int count;
```

- Тип данных переменной определяет:
  - какие могут храниться значения
  - какие можно выполнять операции

- Примитивные типы предопределены в языке и обозначается ключевым словом

Примитивные типы данных

Тип	Описание	По умолчанию
byte	8-битное знаковое целое число. Значения от -128 до 127 (включительно).	0
short	16-битное знаковое целое число. Значения от -32768 до 32767 (включительно).	0
int	32-битное знаковое целое число. Значения от -2147483648 до 2147483647 (включительно)	0
long	64-битное знаковое целое число. Значения от -9223372036854775808 до 9223372036854775807 (включительно)	0L
float	32-битное число с плавающей точкой, соответствующее IEEE 754.	0.0f
double	64-битное число с плавающей точкой двойной точности, соответствующее IEEE 754.	0.0
boolean	Логический тип данных с двумя возможными значениями: true и false	false
char	16-битный символ Unicode	‘\u0000’

Ссылочные типы данных

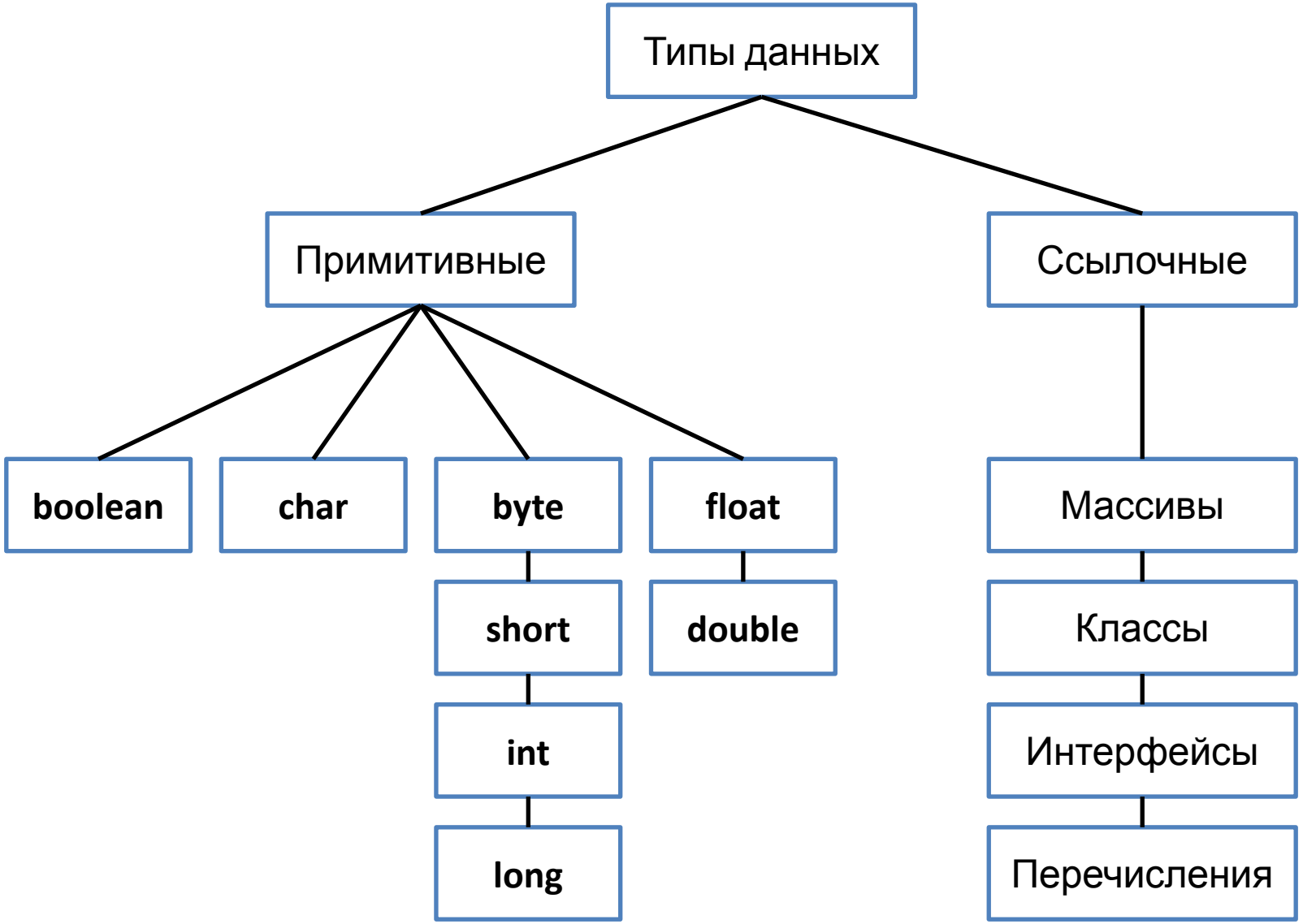
- Остальные типы являются ссылочными
- Значение ссылочной переменной
  - может быть **null**
  - может ссылаться на объект в куче
- Ссылка может быть на новый созданный объект (с помощью оператора **new**):

```
Object object = new Object();
```

- Или на уже существующий объект:

```
Object reference = object;
```

Типы данных


Типы данных
Примитивные
Ссылочные
boolean
char
byte
float
short
double
int
long
Массивы
Классы
Интерфейсы
Перечисления

Именование переменных

- Имя переменной – последовательность из латинских букв и цифр, начинающаяся с буквы
  - Могут также встречаться подчеркивания, знак $, буквы других алфавитов
- Чувствительно к регистру
- Не должно сов
  с ключевыми словами

- count
- gearRatio
- currentValue

Допустимо

- gr
- ses
- c_r

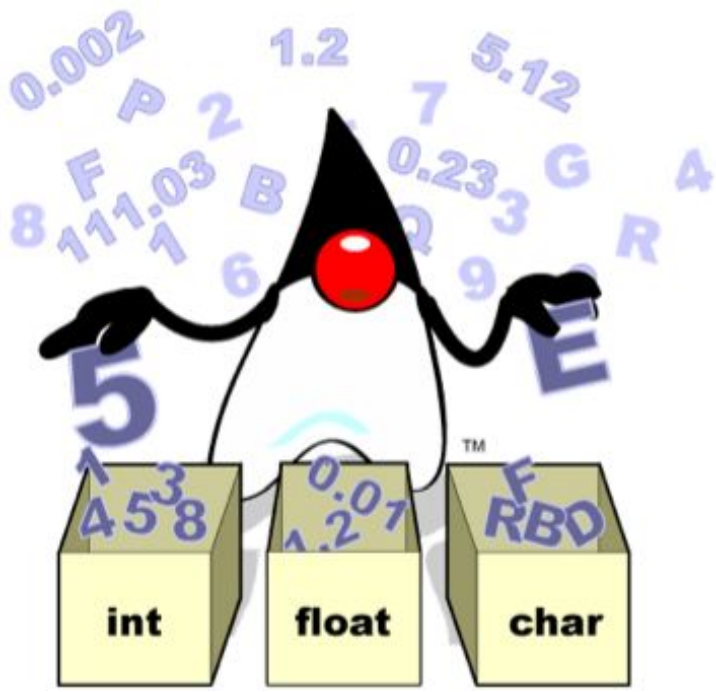
Допустимо, но непонятно

- 1a
- s*s
- d d

Не допустимо

Литералы (неименованные константы)

```
boolean result = true;
char capitalC = 'C';
int i = 100000;
long l = 1L;
int decVal = 26;
int octVal = 032;
int hexVal = 0x1a;
int binVal = 0b1010;
double d1 = 123.4;
double d2 = 1.234e2;
float f1  = 123.4f;
String s1 = "Hello, World!";
String s2 = "S\u00ED se\u00F1or";
```



Final переменные

Значение переменной примитивного типа, объявленной с модификатором final, нельзя изменить.

```
final int myConst = 1;
myConst = 2; // ОШИБКА
```

Если переменная ссылочного типа объявлена как final, то нельзя изменить значение ссылки, но можно изменить состояние объекта, на который ссылается переменная.

Преобразование типов

Автоматическое

- Два типа совместимы
- Целевой тип шире исходного
- Выполняется автоматически
- **byte → short → int → long → float → double**

Явное

Формат:

- (target-type) value

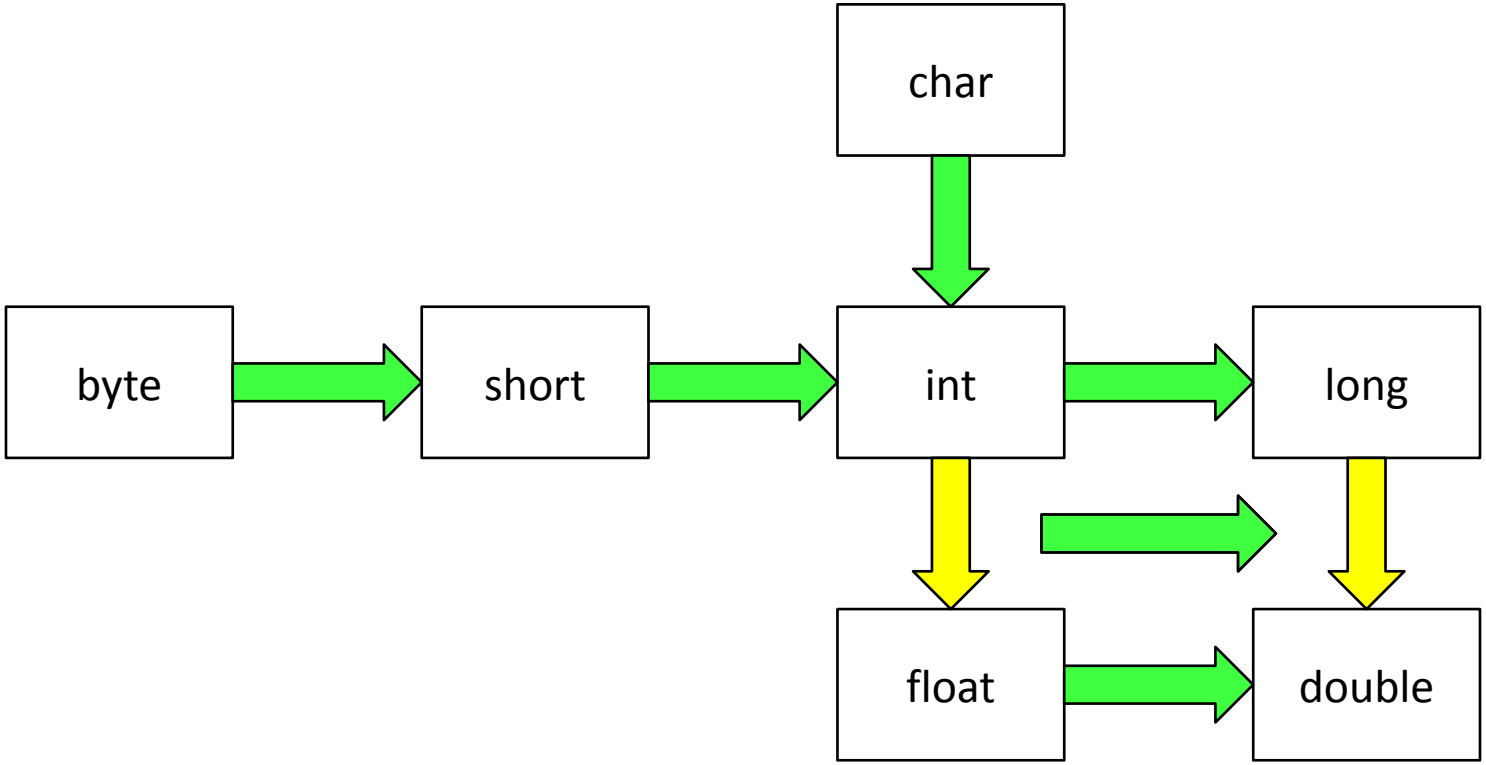
Пример:

- int a;
- byte b;
- b = (byte) a;

Усечение:

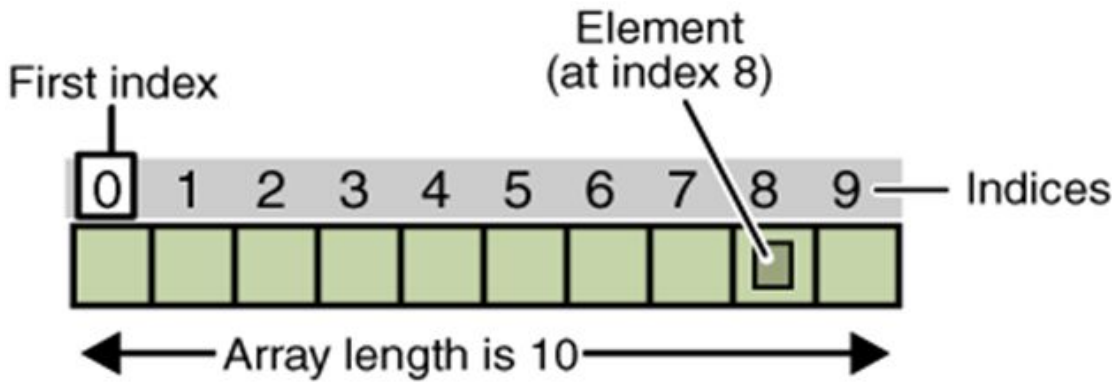
- int a;
- double b;
- a = (int) d;

Повышающие преобразования



Массивы

- *Массив* – это контейнерный объект, содержащий фиксированное количество значений одного типа
  - Длина массива устанавливается при его создании (во время выполнения)
  - После создания длину массива изменить нельзя
- *Элемент массива* – это одно из значений в массиве, к которому можно обратиться, используя его позицию внутри массива - *индекс элемента*
  - Нумерация элементов начинается с 0



Массивы

- Неинициализированная ссылка на массив

```
int[] array;
```

- Создание массива

```
int[] array = new int[10];
```

- Доступ к элементам

```
array[0] = 100;
System.out.println("1st element: " + array[0]);
```

Массивы

- Инициализация при создании

```
int[] array = {100, 200, 300, 400, 500, 600};
```

- Длина массива

```
int size = array.length;

for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}
```

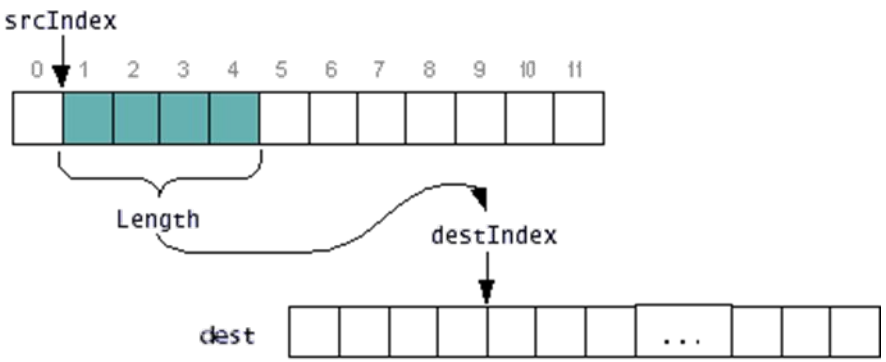
Массивы

- **Массивы могут быть многомерными**

```
String[][] names = {
        {"Mr. ", "Mrs. ", "Ms. "},
        {"Smith", "Jones"},
};
```

- **Копирование массивов – это стандартная операция**

```
public static void arraycopy(Object src,
                             int srcIndex,
                             Object dest,
                             int destIndex,
                             int length)
```



Перебор элементов массива

```
int[] array = {100, 200, 300, 400, 500, 600, 700};


for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}


for (int value : array) {
    System.out.println(value);
}
```

Операции

Приоритет операций	
постфиксные	`expr++  expr--`
унарные	`++expr  --expr  +expr  -expr  ~  !`
мультипликативные	`*  /  %`
аддитивные	`+  -`
сдвиговые	`<<  >>  >>>`
относительные	`<  >  <=  >=  instanceof`
сравнения	`==  !=`
побитовое И (AND)	`&`
побитовое исключающее ИЛИ (XOR)	`^`
побитовое включающее ИЛИ (OR)	`|`
логическое И (AND)	`&&`
логическое ИЛИ (OR)	`||`
тернарная	`? :`
присваивания	`=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=  >>>=`

Операторы ветвления и цикла

Операторы, позволяющие изменять последовательный порядок выполнения программ (создавая циклы и организовывая ветвления):

- **операторы ветвления** (if-then, if-then-else, switch)
- **операторы цикла** (for, while, do-while)
- **операторы перехода** (break, continue, return)

Оператор if-then

```
if (условие) {
    // операторы, выполняемые
    // если условие истинно
}
```

```
if (isMoving) {
    // Если велосипед движется
    // уменьшить его скорость
    currentSpeed--;
}
```

Оператор if-then-else

```
if (условие) {
    // операторы, выполняемые
    // если условие истинно
} else {
    // операторы, выполняемые
    // если условие ложно
}
```

```
if (isMoving) {
    currentSpeed--;
} else {
    System.err.println("The bicycle "
            + "has already stopped!");
}
```

Оператор if-then-else

```
if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
    grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else if (testscore >= 50) {
    grade = 'E';
} else {
    grade = 'F';
}
```

Оператор switch

```
switch (place) {
    case 1:
        medal = "Gold";
        break;
    case 2:
        medal = "Silver";
        break;
    case 3:
        medal = "Bronze";
        break;
    default:
        medal = "No medal";
        break;
}
```

Оператор ? : (Тернарная операция)

```
условие
      ? значение_если_истина
      : значение_если_ложь
```

```
int max = x > y ? x : y;
```

```
if (x > y) {
    max = x;
} else {
    max = y;
}
```

```
int a = x > ((y != 0) ? y : 1)
    ? (y - 2 > 0) ? y - 2 : - (y - 2) : 0;
```

Цикл while

```
while (условие) {
    // операторы тела цикла
}
```

```
int count = 0;
while (count < 10) {
    System.out.println("Count is: "
            + count);
    count++;
}
```

Цикл do-while

```
do {
    // операторы тела цикла
} while (условие);
```

```
int count = 0;
do {
    System.out.println("Count is: "
            + count);
    count++;
} while (count <= 10);
```

Цикл for

```
for (инициализация;
        условие;
        изменение) {
    // операторы тела цикла
}
```

```
for (int i = 0; i < 10; i++) {
    System.out.println("Count is: " + i);
}
```

Цикл for each

```
int[] numbers = {1, 2, 3, 4, 5,
                 6, 7, 8, 9, 10};

for (int item : numbers) {
    System.out.println("Count is: "
            + item);
}
```

Оператор break

```
int searchfor = 12;
int index = -1;

for (int i = 0; i < numbers.length; i++) {
    if (numbers[i] == searchfor) {
        index = i;
        break;
    }
}
```

Оператор continue

```
int numPs = 0;
for (int i = 0; i < max; i++) {
    // Игнорируем все не p
    if (searchMe.charAt(i) != 'p') {
        continue;
    }
    // Выполняем обработку p
    numPs++;
}
```

Метод main

- В Java выполнение приложений начинается с метода main

```
public static void main(String[] args)
```

- В приложении может быть несколько методов main (по одному на класс)
- Аргументы командной строки, указанные при запуске приложения доступны через переменную args.

Параметры командной строки

```
public static final void main(String[] args) {
    for (String arg : args) {
        System.out.println(arg);
    }
}
```

Простой консольный ввод-вывод

```
public static final void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    int i = scanner.nextInt();
    String text = scanner.next();

    System.out.println("Integer: " + i);
    System.out.println("String: " + text);

    System.err.println("This is error stream");

    scanner.close();
}
```

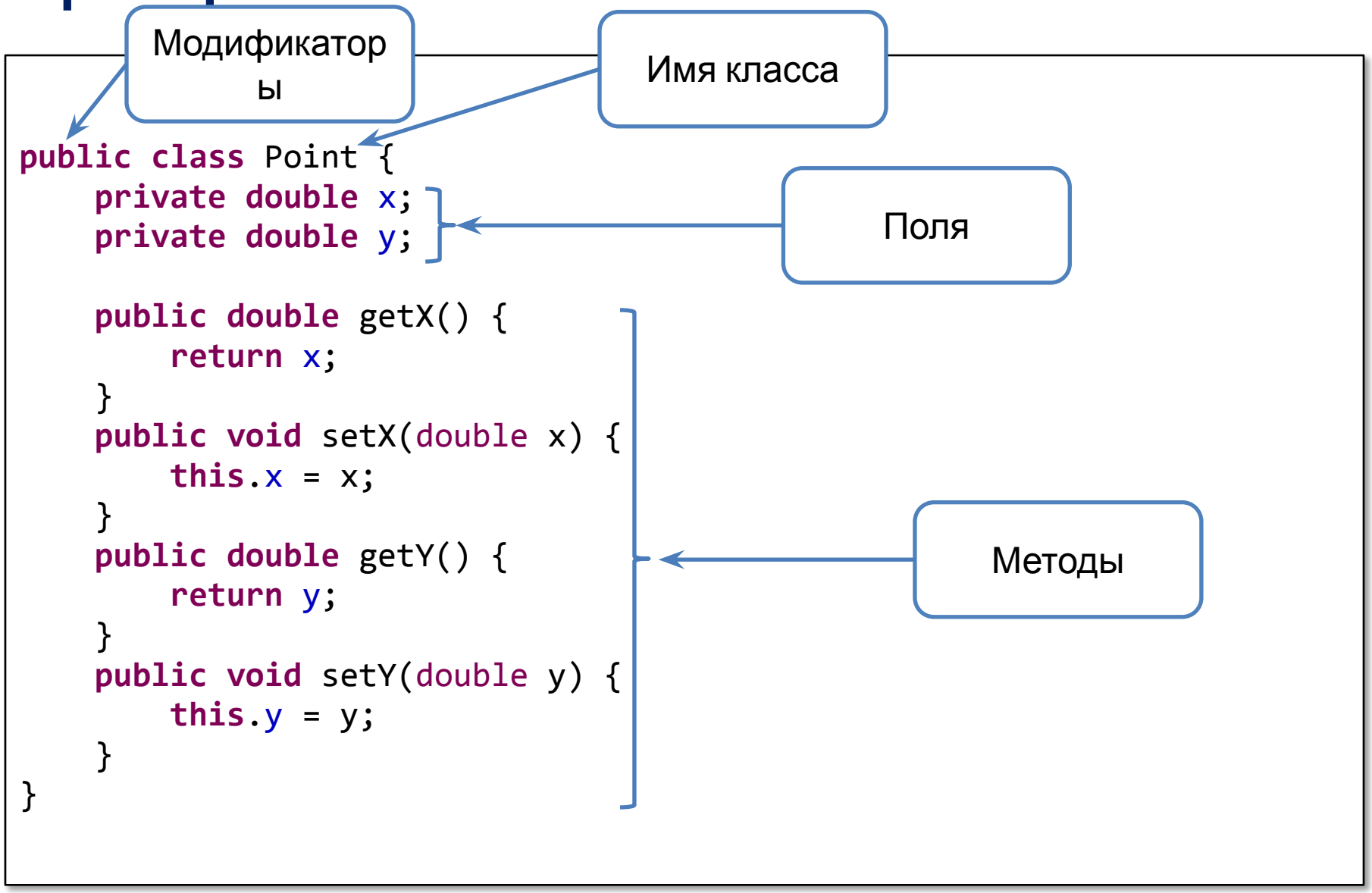
Комментарии

```
/* text */

/** documentation */

// text
```

Классы и объекты

- Класс – описание нового ссылочного типа
  - Содержит описание полей и методов
- Классы позволяют организовать приложение как набор взаимодействующих объектов
- Объект (экземпляр) – переменная нового ссылочного типа (класса)
  - Поля характеризуют состояние объекта
  - Методы обеспечивают поведение объекта
  - Каждый объект уникален

Пример класса

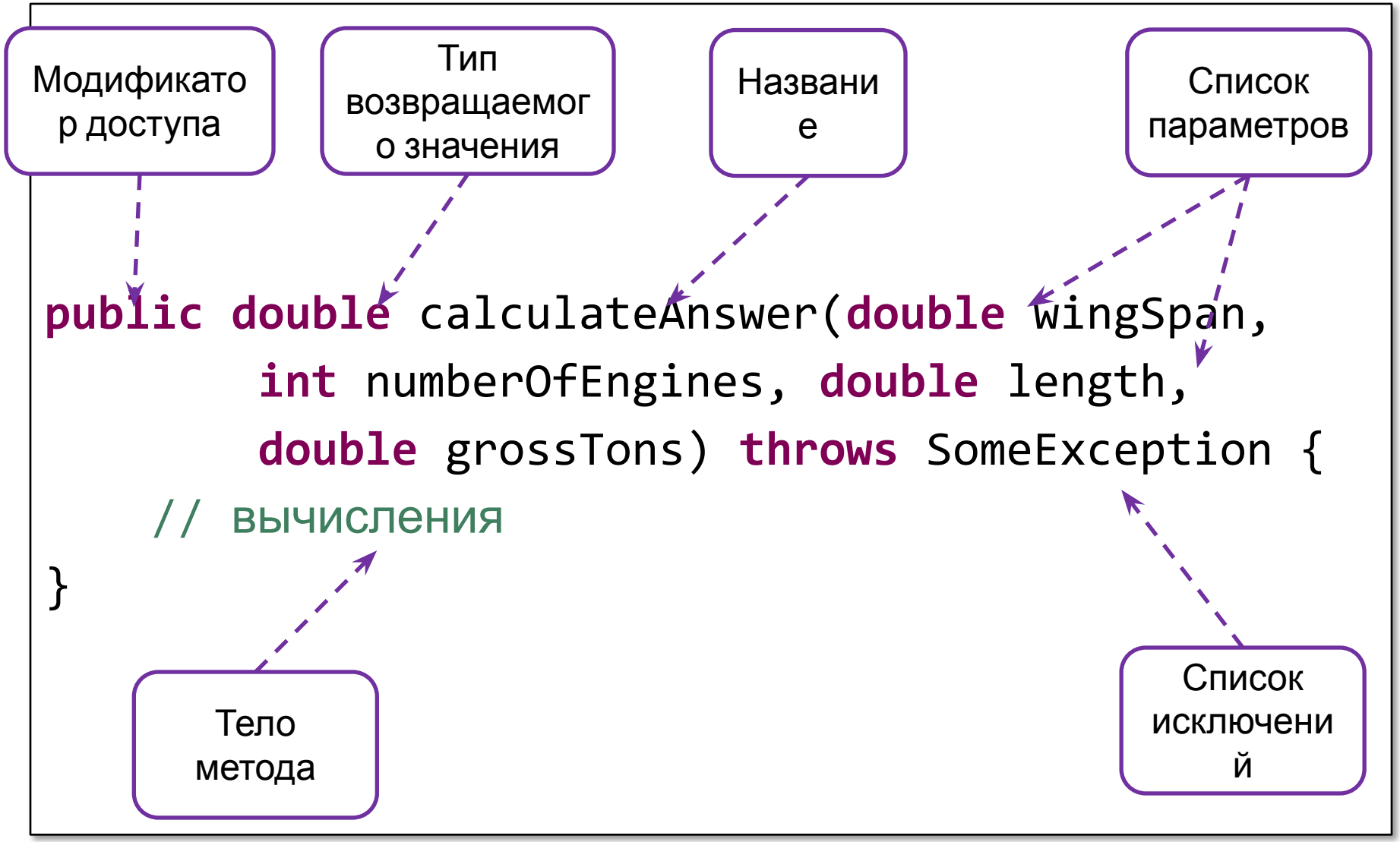



```
public class Point {
    private double x;
    private double y;

    public double getX() {
        return x;
    }
    public void setX(double x) {
        this.x = x;
    }
    public double getY() {
        return y;
    }
    public void setY(double y) {
        this.y = y;
    }
}
```

Пример использования класса

```
Point point = new Point();

point.setX(0.0);

point.setY(1.0);

point.x = 1.0; // Ошибка

System.out.println("(" + point.getX() + ", "
        + point.getY() + ")");
```

Методы


Модификатор доступа
Тип возвращаемого значения
Название
Список параметров
public double calculateAnswer(double wingSpan,
int numberOfEngines, double length,
double grossTons) throws SomeException {
// вычисления
}
Тело метода
Список исключений

Методы

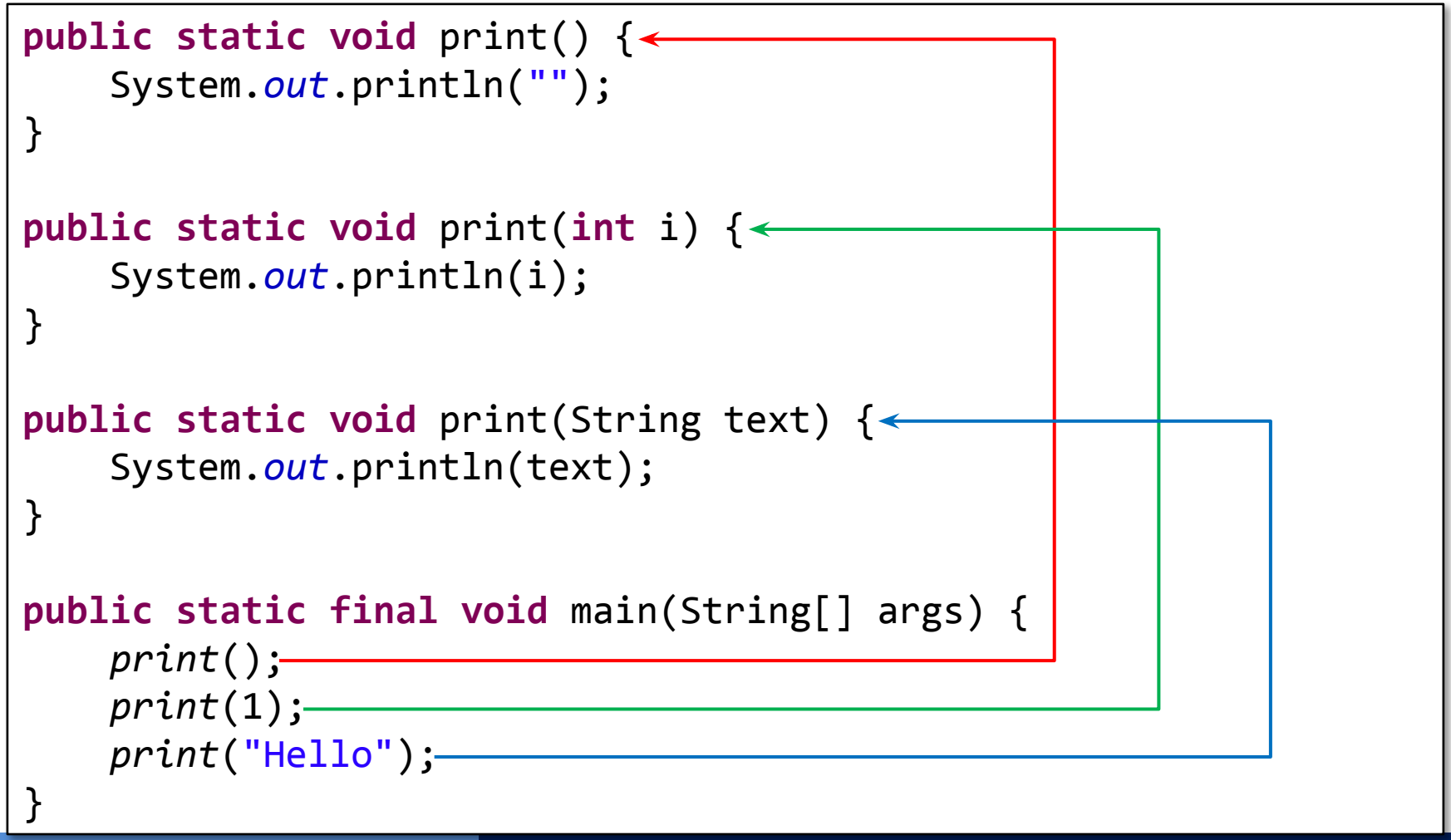
- Название и список параметров составляют **сигнатуру метода**

- Примеры названий
  - run
  - runFast
  - getBackground
  - getFinalData
  - compareTo
  - isEmpty

Перегрузка (Overloading) методов

- Могут существовать методы с одинаковым названием, но разной сигнатурой

```
public static void print() {
    System.out.println("");
}

public static void print(int i) {
    System.out.println(i);
}

public static void print(String text) {
    System.out.println(text);
}

public static final void main(String[] args) {
    print();
    print(1);
    print("Hello");
}
```



Конструкторы

- Конструктор по умолчанию

```
public Point() {
    x = 0;
    y = 0;
}
```

- Конструктор с параметрами

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

Передача параметров в методы

- Параметры в методы всегда передаются **по значению**
  - Изменения в методе примитивных параметров не влияют на вызывающий код
  - Изменения объекта, ссылка на который передается в метод, будут доступны в вызывающем коде
  - Если изменить ссылочный параметр (настроить ссылку на другой объект) , то это не скажется на вызывающем коде

Методы с переменным числом параметров

- **Определение метода**

```
public double sum(double... values) {
    double sum = 0.0;
    for (double v : values) {
        sum += v;
    }
    return sum;
}
```

- **Вызов метода**

```
sum();
sum(1.0);
sum(1.0, 2.0);
sum(1.0, 2.0, 3.0);
```

Оператор return

- В каждом методе, возвращающем некоторое значение, должен быть хотя бы один оператор возврата:

```
return выражение;
```

- Если типом возвращаемого значения является `void`, то может присутствовать такой оператор возврата:

```
return;
```

Завершение работы метода

- Метод возвращает управление в код, которые его вызвал, если
  - выполнены все операторы метода
  - достигнут оператор `return`
  - брошено исключение

Создание объектов

- **Объявление переменной**
- **Инстанцирование (создание объекта)**
- **Инициализация**

```
Point originOne = new Point(23, 94);

Rectangle rectOne = new Rectangle(originOne,
                                  100, 200);

Rectangle rectTwo = new Rectangle(50, 100);
```

Использование объектов

- Обращение к полям и методам
  - `objectReference.fieldName`
  - `objectReference.methodName(arguments);`
  - `objectReference.methodName();`

- Когда объект становится не нужным (на него нет больше ссылок) он может быть собран сборщиком мусора (garbage collector)

Сборщик мусора

- Не нужно явно удалять объекты
- Память, занятая неиспользуемыми объектами, освобождается сборщиком мусора (garbage collector)
- Сбор мусора происходит в фоновом режиме
  - Нельзя точно сказать, когда именно будет выполняться сборка
  - Можно лишь рекомендовать запустить сборку мусора (System.gc() или Runtime.getRuntime().gc())

Ключевое слово this

- **this** – это ссылка на объект, для которого вызывается метод
- **this** определен только для нестатических методов

```
public Rectangle(int width, int height) {
    this(0, 0, width, height);
}

public Rectangle(int x, int y,
                 int width, int height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
}
```

Спецификатор static

- Статические поля

```
static int x;
MyClass.x = 5;
```

- Статические методы

```
static int method() { ... }
MyClass.method();
```

- Константы

```
static final int WIDTH = 800;
System.out.println(MyClass.WIDTH);
```

- Статический блок инициализации

```
static {
    // операторы инициализации
}
```

Спецификаторы

	abstract	static	final
Класс	Нельзя инстанцировать Можно наследоваться	Только у вложенных классов	У класса не может иметь наследников
Метод	Не имеет тела Реализуется в наследниках	Общий для всех экземпляров класса	Нельзя переопределять (override) в наследниках
Поле	Не применим	Общее для всех экземпляров класса	Константа В случае ссылок можно изменять объект, на который она ссылается

Пакеты

- *Пакет* – это группа связанных типов с защитой доступа и управлением пространством имен

- Основные задачи:
  - Логическая структура проекта
  - Избежание конфликтов имен

- Правило:
  - название пакета совпадает с названием папки
  - структуры папок и пакетов совпадают

Пакеты

- Объявление пакета

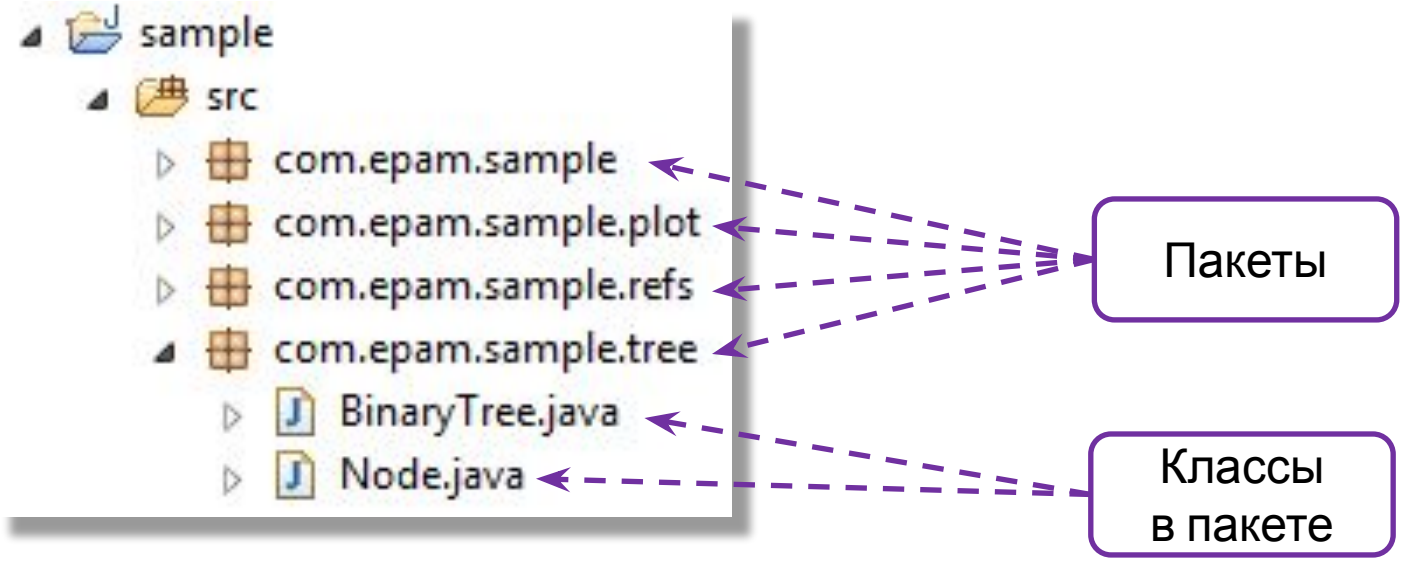
```
package com.epam.weather;
```

- Импорт пакета

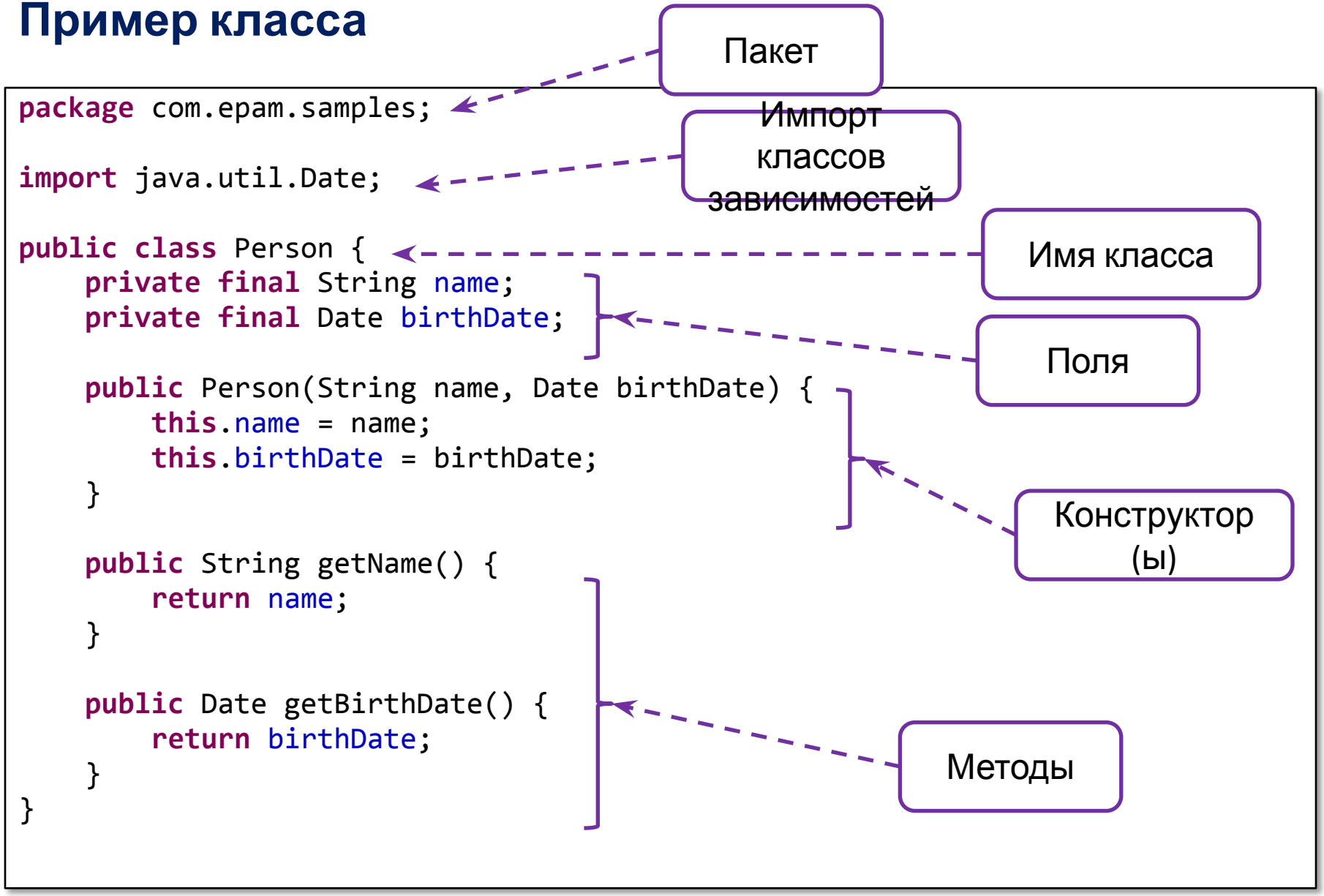
```
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;
import java.io.StringWriter;
```

- Правило для имен: доменное имя наоборот
  - URL: www.epam.com
  - Пакет: com.epam

Пакеты


sample
src
com.epam.sample
com.epam.sample.plot
com.epam.sample.refs
com.epam.sample.tree
BinaryTree.java
Node.java
Пакеты
Классы
в пакете

Пример класса




```
package com.epam.samples;

import java.util.Date;

public class Person {
    private final String name;
    private final Date birthDate;

    public Person(String name, Date birthDate) {
        this.name = name;
        this.birthDate = birthDate;
    }

    public String getName() {
        return name;
    }

    public Date getBirthDate() {
        return birthDate;
    }
}
```

СПАСИБО ЗА ВНИМАНИЕ!

ВОПРОСЫ?

Платформа и язык программирования

Author: Alexander Lotsmanov
E-mail: Alexander_Lotsmanov@epam.com