Компилируемые языки

Языки: Pascal, C, C++, Erlang, Haskell, Rust, Go, Ada

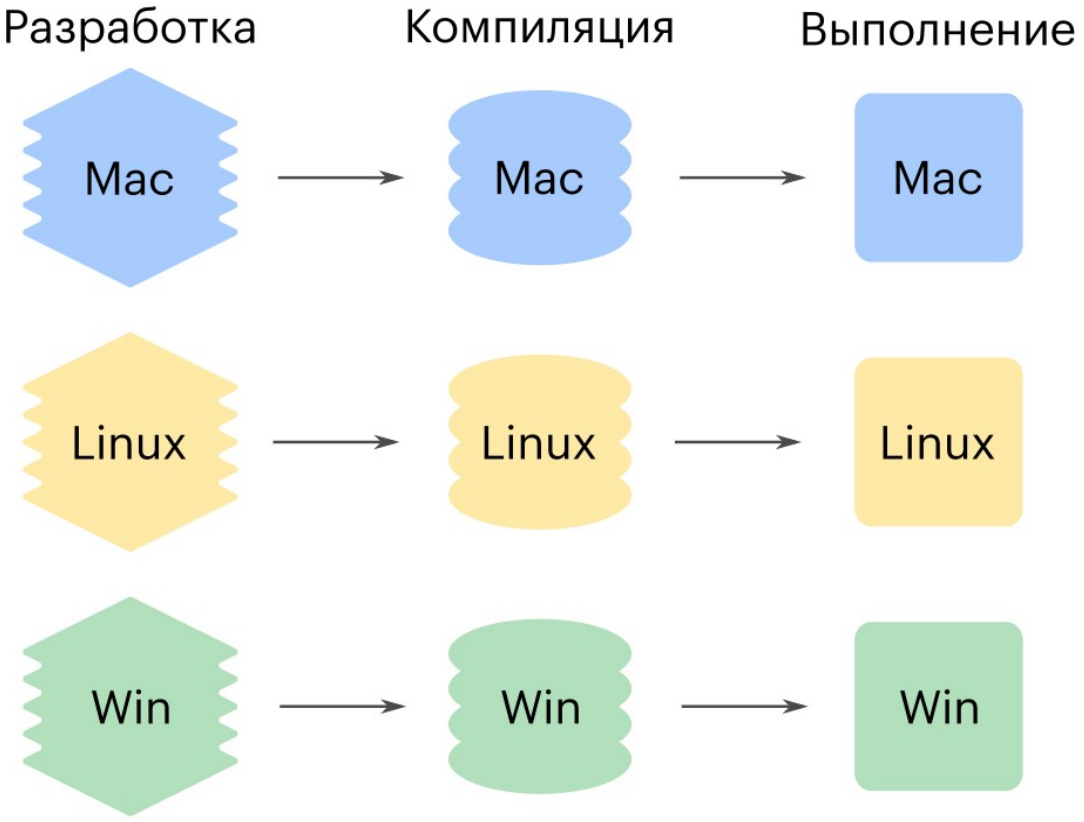
ПЛЮСЫ

- быстрое выполнение
- не требуют дополнительных программ для запуска

МИНУСЫ

- для каждой платформы необходимо своя компиляция
- сложный этап разработки

Программные продукты: операционные системы, nginx, Photoshop, StarCraft, JVM



Skillbox

Интерпретируемые языки

Языки: PHP, Perl, Ruby, Python, JavaScript

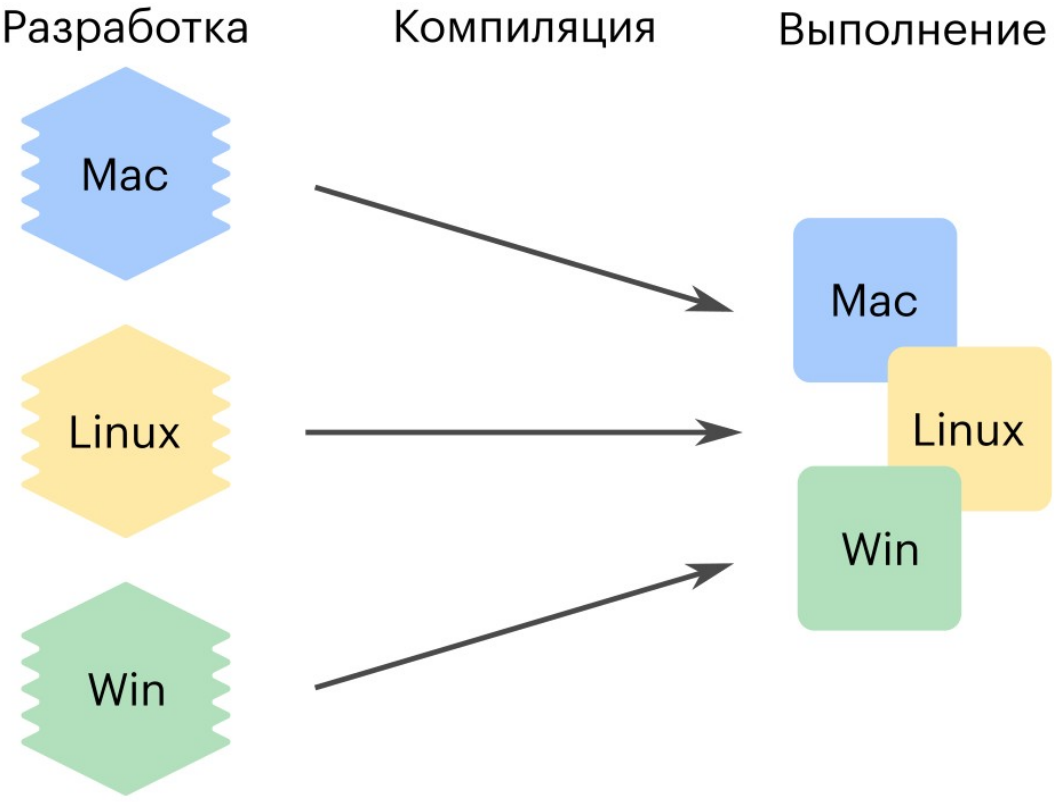
ПЛЮСЫ

- простая разработка – код пишется один раз
- нет компиляции

МИНУСЫ

- требуется интерпретатор
- работает медленнее компилируемого

Применятся: Skype, Slack (Electron) ML, DS, скрипты внутри другого ПО, сайты Instagram, FreeNAS (Django)



Java Virtual Machine языки

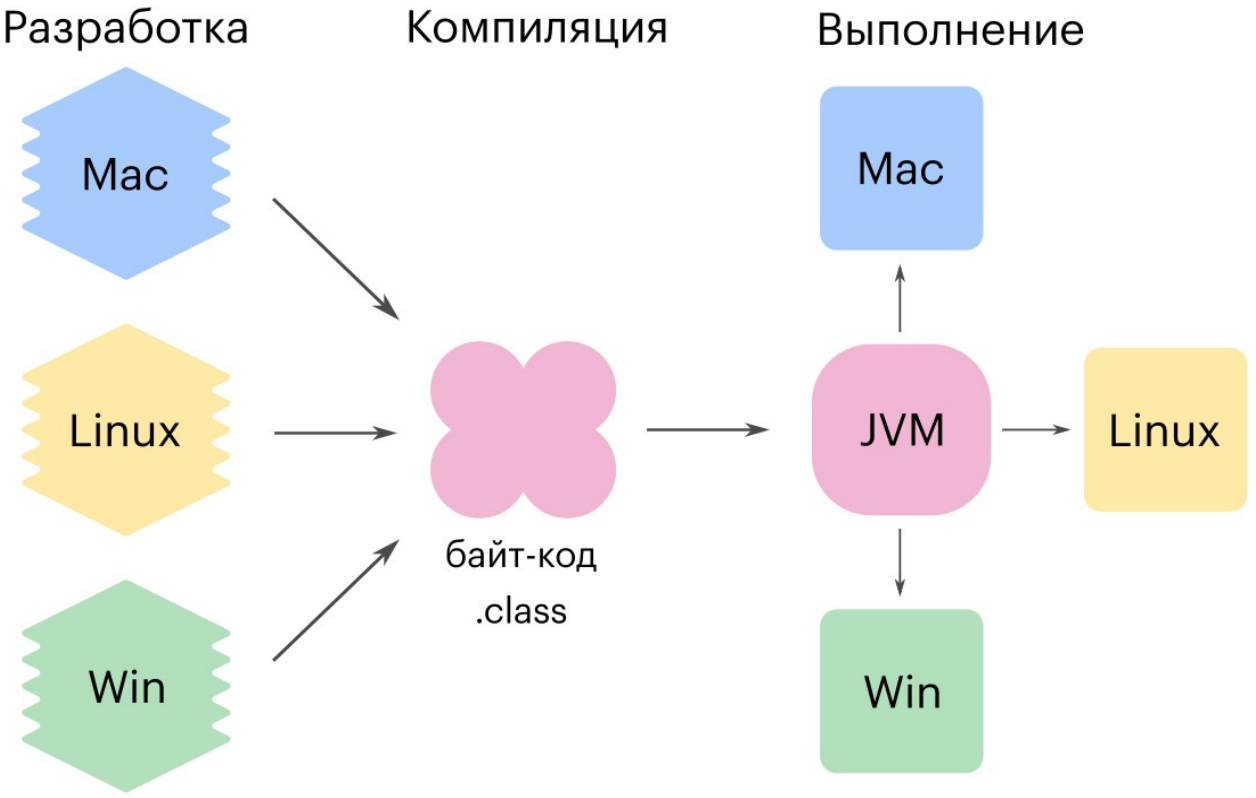
Языки: Java, Kotlin, Scala, Clojure, Groovy, JRuby

ПЛЮСЫ

- JVM
- кроссплатформенность
- JIT-компиляция

МИНУСЫ

- JVM



Применятся: Android, серверные приложения, электронная коммерция, IoT, Hadoop, IntelliJ IDEA, OpenOffice.

Skillbox

Байт-код

Main.java

```java
public class Main {

  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

javac Main.java

Main.class

```
// class version 58.0 (58)
// access flags 0x21
public class Main {

  // compiled from: Main.java

  // access flags 0x1
  public <init>()V
   L0
    LINENUMBER 1 L0
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
   L1
    LOCALVARIABLE this LMain; L0 L1 0
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 0x9
  public static main([Ljava/lang/String;)V
   L0
    LINENUMBER 4 L0
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    LDC "Hello world!"
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
   L1
    LINENUMBER 5 L1
    RETURN
   L2
    LOCALVARIABLE args [Ljava/lang/String; L0 L2 0
    MAXSTACK = 2
    MAXLOCALS = 1
}
```

java Main

```
% java Main
Hello world!
```

Skillbox

Примитивные типы

Skillbox

Примитивные типы

Тип		Минимальное значение		Максимальное значение		Значение по умолчанию
byte	8 bit	-2^7	-128	2^7-1	127	0
short	16 bit	-2^{15}	-32 768	$2^{15}-1$	32 767	0
int	32 bit	-2^{31}	-2 147 483 648	$2^{31}-1$	2 147 483 647	0
long	64 bit	-2^{63}	-9 223 372 036 854 775 808	$2^{63}-1$	9 223 372 036 854 775 807	0L
float	32 bit	-3.4E+38		3.4E+38		0.0f
double	64 bit	-1.7E+308		1.7E+308		0.0d
char	16 bit	'\u0000' = int 0		'\uffff' = int 65535		'\u0000'
boolean		-		-		false

Skillbox

Целочисленные примитивные типы

byte, short, char, int, long

Skillbox

Переполнение значений целочисленных переменных

```
byte b = 120;
for (int i = 0; i < 10; i++) {
  b++;
}
System.out.println(b);
```

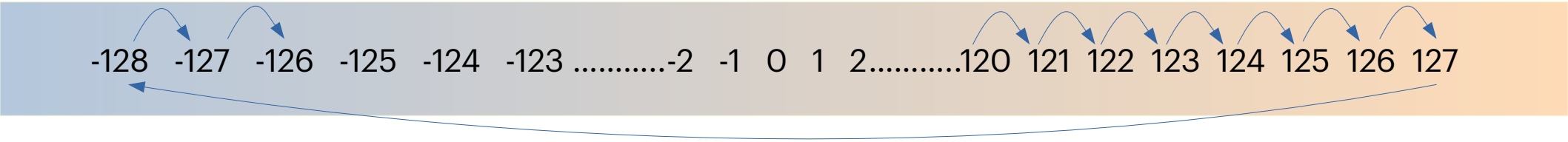
Какое значение будет выведено в консоль?

Переполнение значений целочисленных переменных

```
byte b = 120;
for (int i = 0; i < 10; i++) {
  b++;
}
System.out.println(b);
```

В консоль будет выведено значение **-126**

Аналогично работают переполнения **short, int, long, char**



Выбирайте подходящий тип для хранения данных

Skillbox

Переполнение значений целочисленных переменных

Если переполнение критично, что делать?

Java 7

```
int a = safeAdd(Integer.MAX_VALUE, right: 1);
```

```
static int safeAdd(int left, int right) {
  if (right > 0 ? left > Integer.MAX_VALUE - right
      : left < Integer.MIN_VALUE - right) {
    throw new ArithmeticException("Integer overflow");
  }
  return left + right;
}
```

Java 8

```
int a = Math.addExact(Integer.MAX_VALUE, 1);
```

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : integer overflow
  at java.base/java.lang.Math.addExact(Math.java:828)
  at examples.ArrayMaxDynamic.Max3.main(Max3.java:8)
```

Подробнее про переполнение и варианты предотвращения:
https://wiki.sei.cmu.edu/confluence/display/java/NUM00-J.+Detect+or+prevent+integer+overflow

Skillbox

Деление целочисленных переменных

```
long a = 17;
long b = 3;

long i = a / b;
```

i = 5

При делении целочисленных переменных результат не округляется.

Десятичная дробь отбрасывается.

Преобразования целочисленных типов данных

Явное

```
long a = 100;
int b = (int) a;
```

```
int a = 68;
char b = (char) a;
```

Явное преобразование требуется при сужении типа.

Неявное

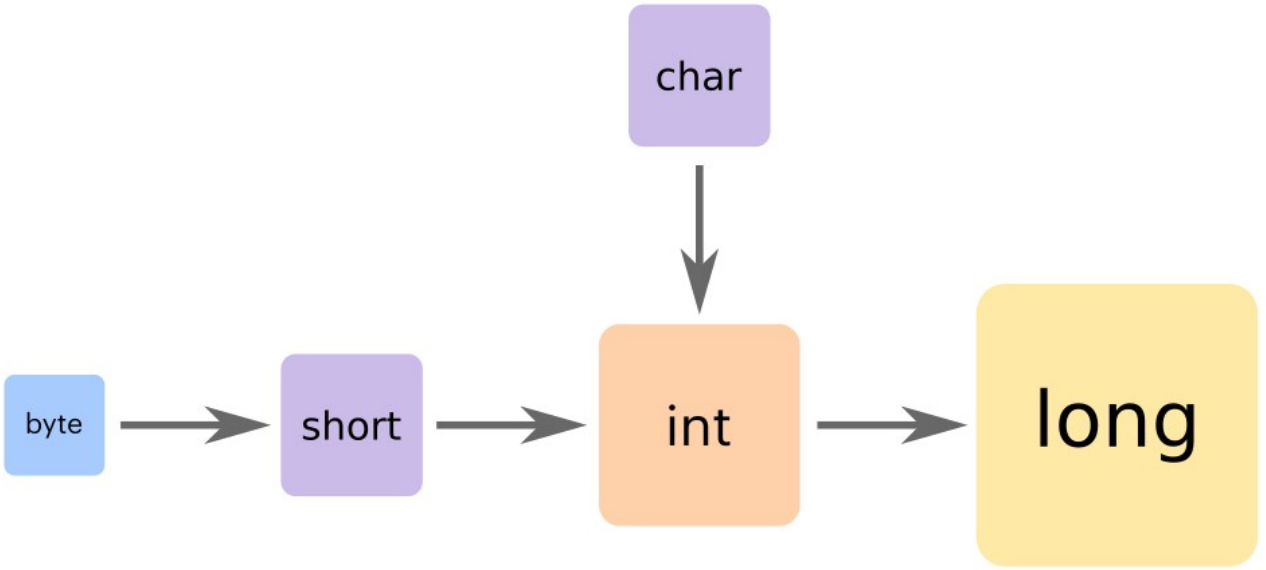
```
int a = 100;
long b = a;
```

```
char a = 100;
int b = a;
```

Неявное преобразование происходит при расширении типа.

Skillbox

Схема автоматических неявных преобразование целочисленных типов данных



Примитивные типы с плавающей точкой

float, double

Skillbox

Точность расчета чисел с плавающей точкой

```
double a = 5.0 - 0.1;
double b = 4.8 + 0.1;
```

```
System.out.println(a == b); //false
```

a = 4.9

b = 4.899999999999995

```
double a = 5.0 / 0.0;
double b = a + 5.0;
```

```
System.out.println(a == b); // true
```

a = Infinity

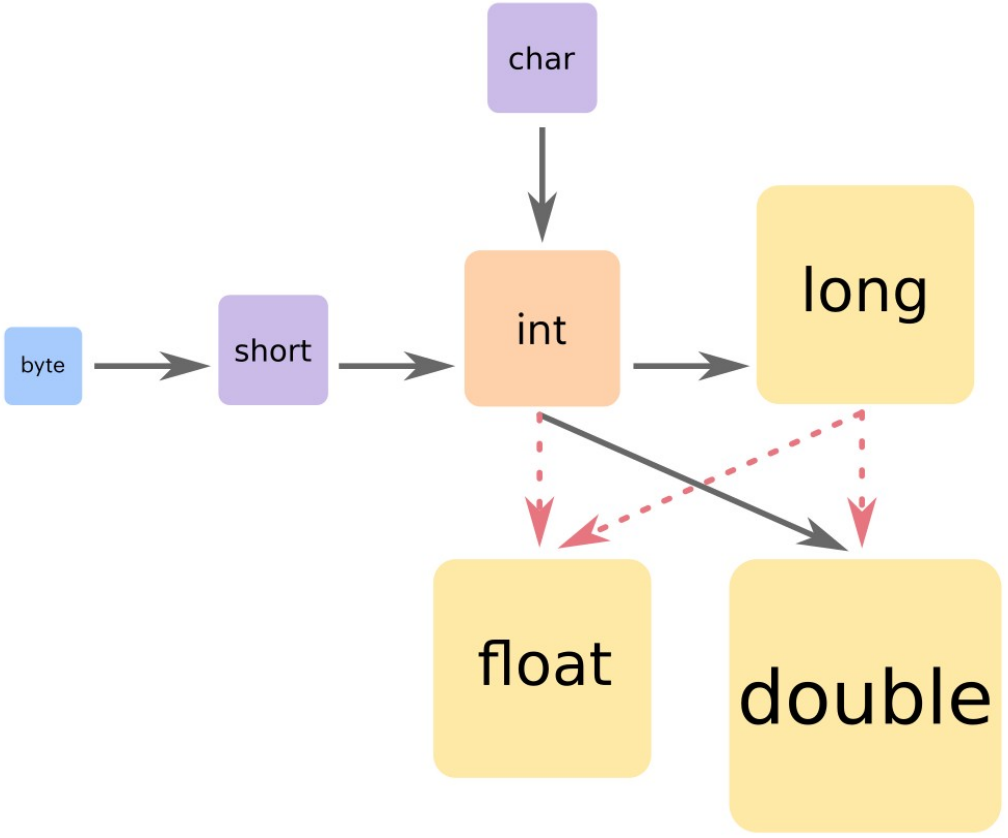
b = Infinity

Статья про причины:

https://habr.com/ru/post/219595/

https://habr.com/ru/post/112953/

Схема автоматических неявных преобразование числовых типов данных



без потери точности

с потерью точности

```
int a = 323453435;
float b = a;
System.out.println(b);
```

3.2345344E8 =
3.2345344*10^8 =
323453440

Skillbox

Точные вычисления

BigDecimal, JavaMoney

Skillbox

Точность расчета чисел с плавающей точкой - BigDecimal

Для точных и финансовых вычислений
используйте **java.math.BigDecimal**

```
BigDecimal a = new BigDecimal( val: "4.8");
BigDecimal b = a.add(new BigDecimal( val: "0.1"));

BigDecimal c = new BigDecimal( val: "5");
BigDecimal d = c.subtract(new BigDecimal( val: "0.1"));

System.out.println(b.compareTo(d)); //0 == равны
```

Skillbox

Точность расчета чисел с плавающей точкой - BigDecimal

+	add()	^	pow()
-	subtract()	max	max()
×	multiply()	min	min()
÷	divide()	√	sqrt()

Каждая операция возвращает новый объект BigDecimal

https://docs.oracle.com/javase/9/docs/api/java/math/BigDecimal.html

Skillbox

Точность расчета чисел с плавающей точкой - BigDecimal

Для создания BigDecimal – используйте строки

```
BigDecimal a = new BigDecimal( val: 4.85645);
BigDecimal b = new BigDecimal( val: "4.85645");

System.out.println(a);
System.out.println(b);
```

4.85644999999999971151964928139932453632354736328125
4.85645

Перед тем как попасть в конструктор BigDecimal, значение будет double

Skillbox

Точность расчета чисел с плавающей точкой - BigDecimal

Округление

Java 8

```
BigDecimal a = new BigDecimal( val: "45.34532")
    .setScale( newScale: 1, BigDecimal.ROUND_HALF_DOWN);

System.out.println(a);
```

45.3

Java 9+

```
BigDecimal a = new BigDecimal( val: "45.34532")
    .setScale( newScale: 1, RoundingMode.HALF_DOWN);

System.out.println(a);
```

45.3

С Java 9 объявлены устаревшими константы округления, заменены на enum RoundingMode

Точность расчета чисел с плавающей точкой - JavaMoney

Библиотека JavaMoney

- обмен валют
- финансовые операции
- расширенное форматирование
- API для работа с регионами в виде иерархии

https://javamoney.github.io/

Skillbox

Методы

Skillbox

Методы - определение

Метод это действие, при наименовании метода обычно используется глагол

```
add()
meow()
feed()
generateCarNumbers()
isBlocked()
fly()
```

Именуются методы с прописной маленькой буквы, каждое последующее слово с большой

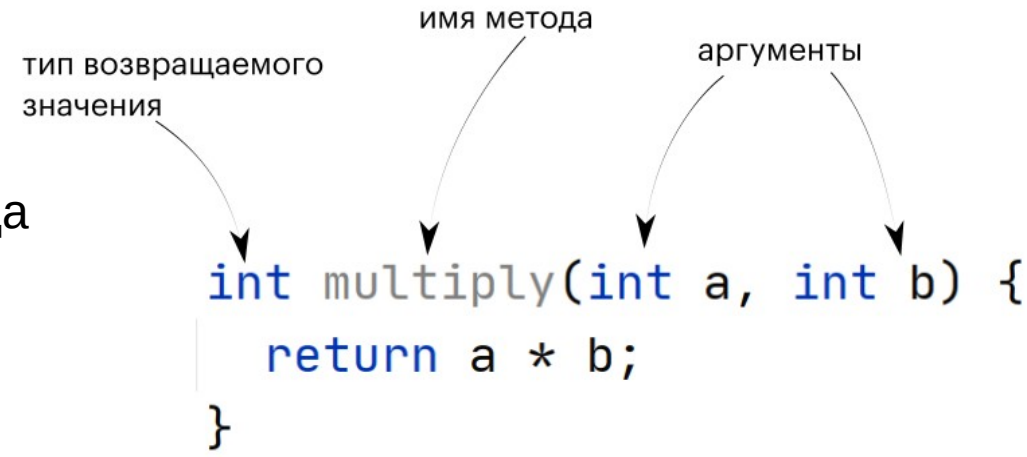
Метод располгается в классе, интерфейсе. Внутри метода объявить другой метод нельзя.

Методы - сигнатура

Сигнатура метода - набор свойств метода по которому возможно точно идентифицировать метод.

В состав сигнатуры метода входит:
- имя метода
- набор аргументов (параметров) метода
- порядок следования аргументов

В состав сигнатуры метода НЕ входит:
- возвращаемое значение
- выбрасываемые исключения
- модификатор доступа



Skillbox

Методы – виды методов

На основании комбинации сигнатур, можно выделить методы:

возврат	аргументы	пример
void	нет	`void repaint()` `void clear()`
void	есть	`void setName(String name)` `void add(Car car)`
есть	нет	`String getName()` `boolean isAlive()`
есть	есть	`int multiply(int a, int b)` `File getFile(String path)`

Skillbox

Методы – перегрузка методов

Методы могут иметь одинаковое имя в одном классе при выполнении условий - набор и/или порядок аргументов отличается

```
boolean remove(Car car);

Car remove(int number);
```

```
void add(Product newProduct)

int add(Product product)
```

```
void add(List<Product> products)

void add(Product product)
```

Методы имеют одинаковый тип аргументов, то что имена у аргументов разные - такой код не скомпилируется.

https://habr.com/ru/company/otus/blog/428307/

Skillbox

Стэк методов

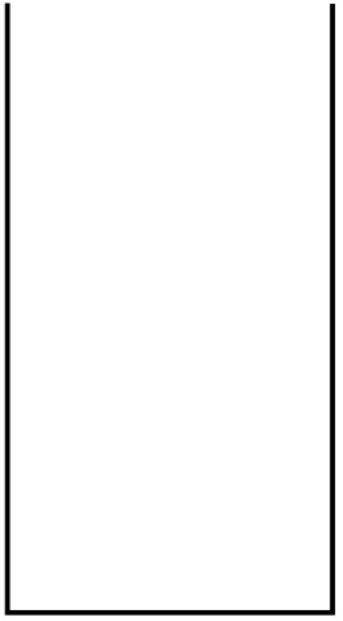
Skillbox

Стэк методов

```java
public static void main(String[] args) {
  int first = 4;
  int second = 15;

  int modulo = getModulo(first, second);
  int answer = multiply(modulo, first);

  System.out.println(answer);
}
```

```java
private static int multiply(int a, int b) {
  int result = a * b;
  return result;
}
```

```java
private static int getModulo(int a, int b) {
  int max = Math.max(a, b);
  int min = Math.min(a, b);
  return max % min;
}
```



Stack

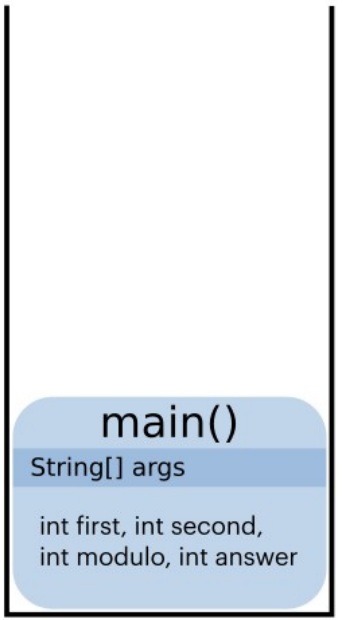
Skillbox

Стэк методов

```java
public static void main(String[] args) {
  int first = 4;
  int second = 15;

  int modulo = getModulo(first, second);
  int answer = multiply(modulo, first);

  System.out.println(answer);
}
```

```java
private static int multiply(int a, int b) {
  int result = a * b;
  return result;
}
```

```java
private static int getModulo(int a, int b) {
  int max = Math.max(a, b);
  int min = Math.min(a, b);
  return max % min;
}
```



Stack

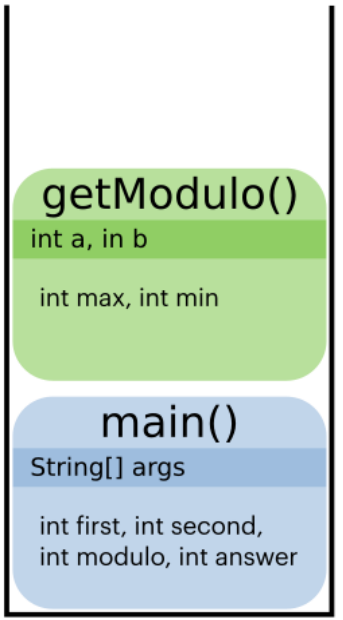
Skillbox

Стэк методов

```
public static void main(String[] args) {
  int first = 4;
  int second = 15;

  int modulo = getModulo(first, second);
  int answer = multiply(modulo, first);

  System.out.println(answer);
}
```

```
private static int multiply(int a, int b) {
  int result = a * b;
  return result;
}
```

```
private static int getModulo(int a, int b) {
  int max = Math.max(a, b);
  int min = Math.min(a, b);
  return max % min;
}
```



Stack

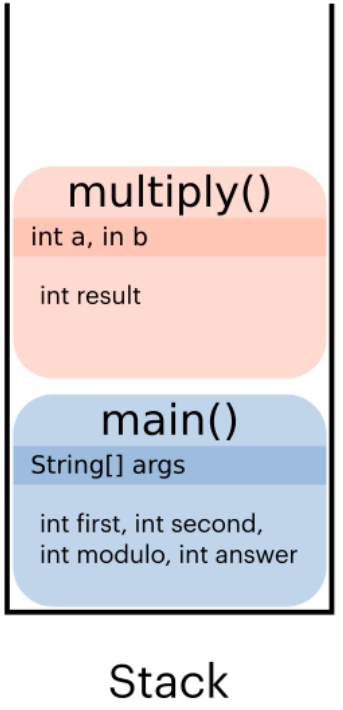
Skillbox

Стэк методов

```java
public static void main(String[] args) {
  int first = 4;
  int second = 15;

  int modulo = getModulo(first, second);
  int answer = multiply(modulo, first);

  System.out.println(answer);
}
```

```java
private static int multiply(int a, int b) {
  int result = a * b;
  return result;
}
```

```java
private static int getModulo(int a, int b) {
  int max = Math.max(a, b);
  int min = Math.min(a, b);
  return max % min;
}
```

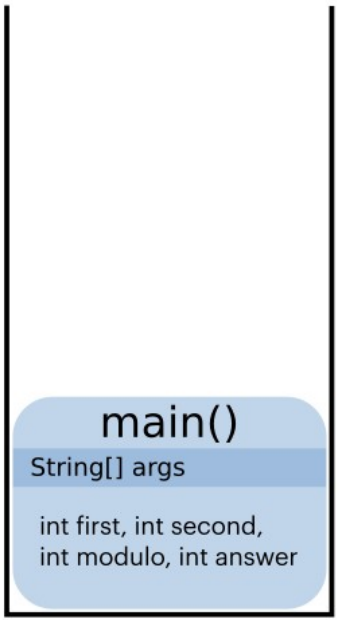


Stack

Skillbox

Стэк методов

```java
public static void main(String[] args) {
  int first = 4;
  int second = 15;

  int modulo = getModulo(first, second);
  int answer = multiply(modulo, first);

  System.out.println(answer);
}

private static int multiply(int a, int b) {
  int result = a * b;
  return result;
}

private static int getModulo(int a, int b) {
  int max = Math.max(a, b);
  int min = Math.min(a, b);
  return max % min;
}
```



Stack

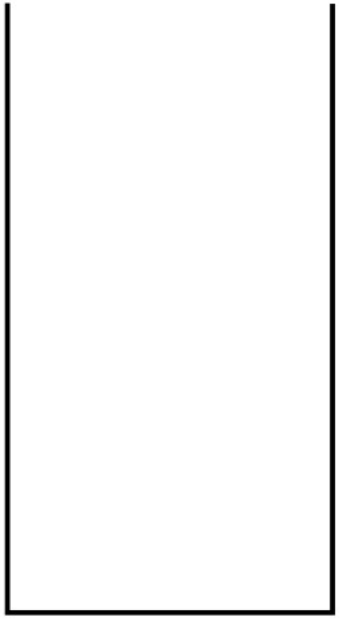
Skillbox

Стэк методов

```java
public static void main(String[] args) {
  int first = 4;
  int second = 15;

  int modulo = getModulo(first, second);
  int answer = multiply(modulo, first);

  System.out.println(answer);
}
```

```java
private static int multiply(int a, int b) {
  int result = a * b;
  return result;
}
```

```java
private static int getModulo(int a, int b) {
  int max = Math.max(a, b);
  int min = Math.min(a, b);
  return max % min;
}
```



Stack

Skillbox

Стэк методов

- Он заполняется и освобождается по мере вызова и завершения новых методов
- Переменные в стеке существуют до тех пор, пока выполняется метод в котором они были созданы
- Если память стека будет заполнена, Java бросит исключение java.lang.StackOverFlowError
- Доступ к этой области памяти осуществляется быстрее, чем к куче
- является потокобезопасным, поскольку для каждого потока создается свой отдельный стек

Heap (куча) и ссылочные переменные

Skillbox

Куча

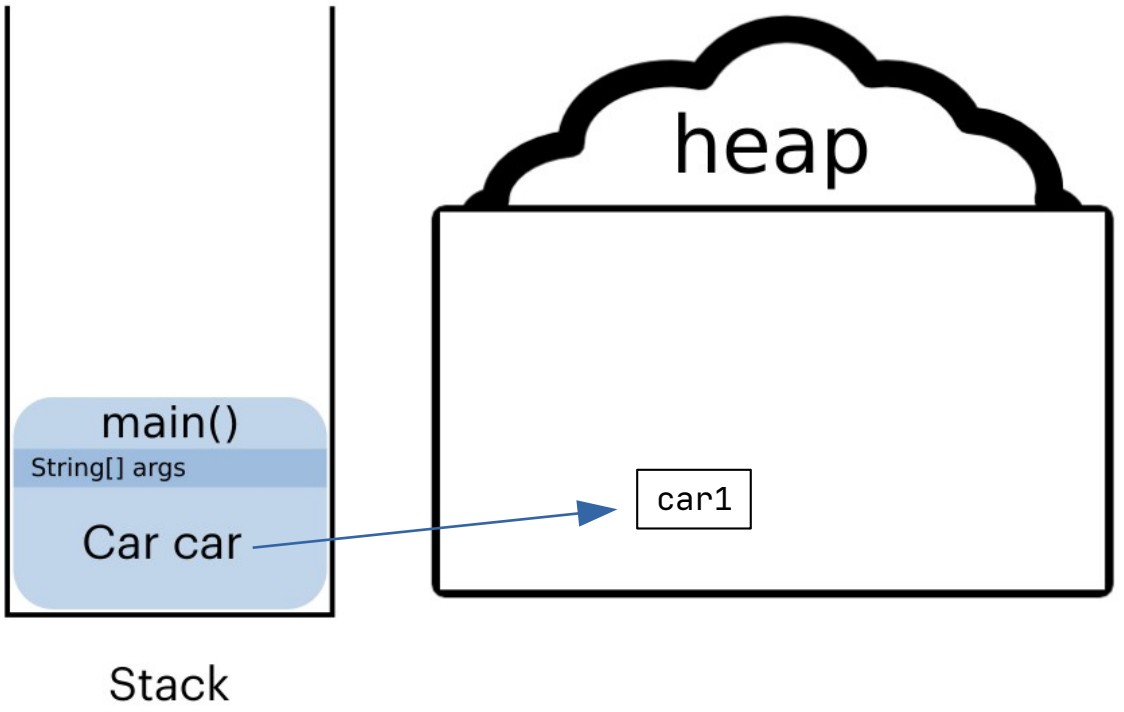
```
public class Main {

  public static void main(String[] args) {
    Car car = new Car();
    car.setName("Benefaro GT");
    car = new Car();
  }
}
```

```
public class Car {

  private String name;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Skillbox

Куча

```
public class Main {

  public static void main(String[] args) {
    Car car = new Car();
    car.setName("Benefaro GT");
    car = new Car();
  }
}
```



Эта область памяти используется для объектов и классов. Новые объекты всегда создаются в куче, а ссылки на них хранятся в стеке.

Эти объекты имеют глобальный доступ и могут быть получены из любого места программы.

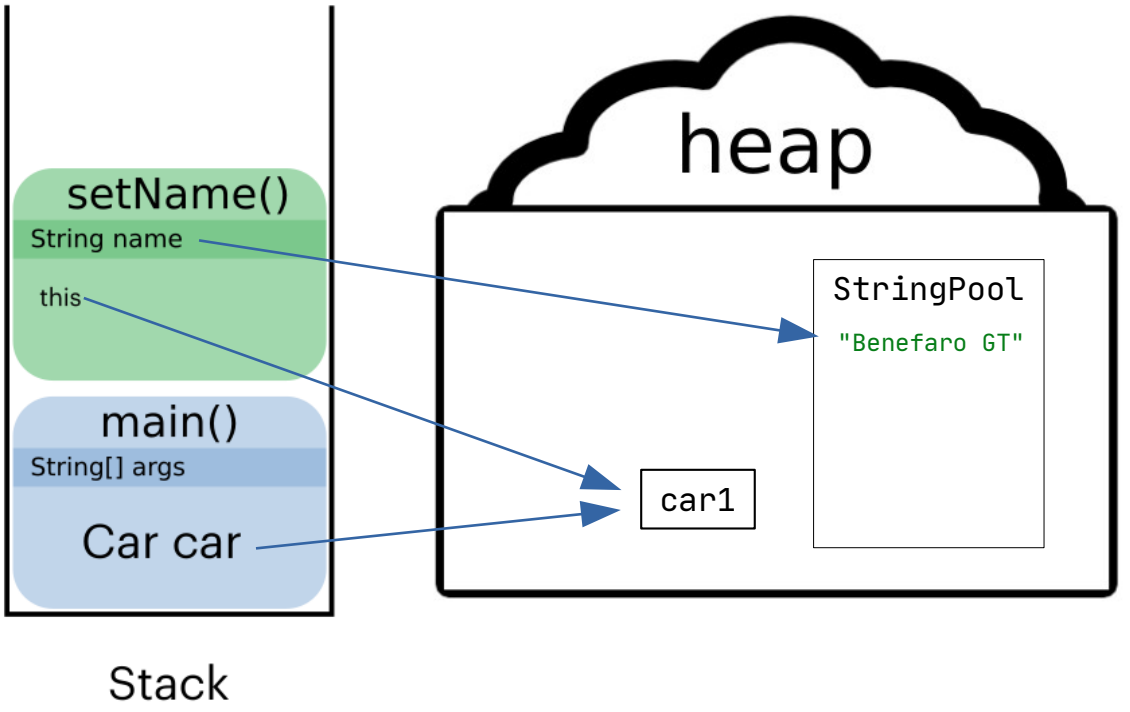
Skillbox

Куча

```
public class Main {

  public static void main(String[] args) {
    Car car = new Car();
    car.setName("Benefaro GT");
    car = new Car();
  }
}
```

В методы передаются ссылки на объекты.

Внутри объекта есть ссылка на сам объект - this

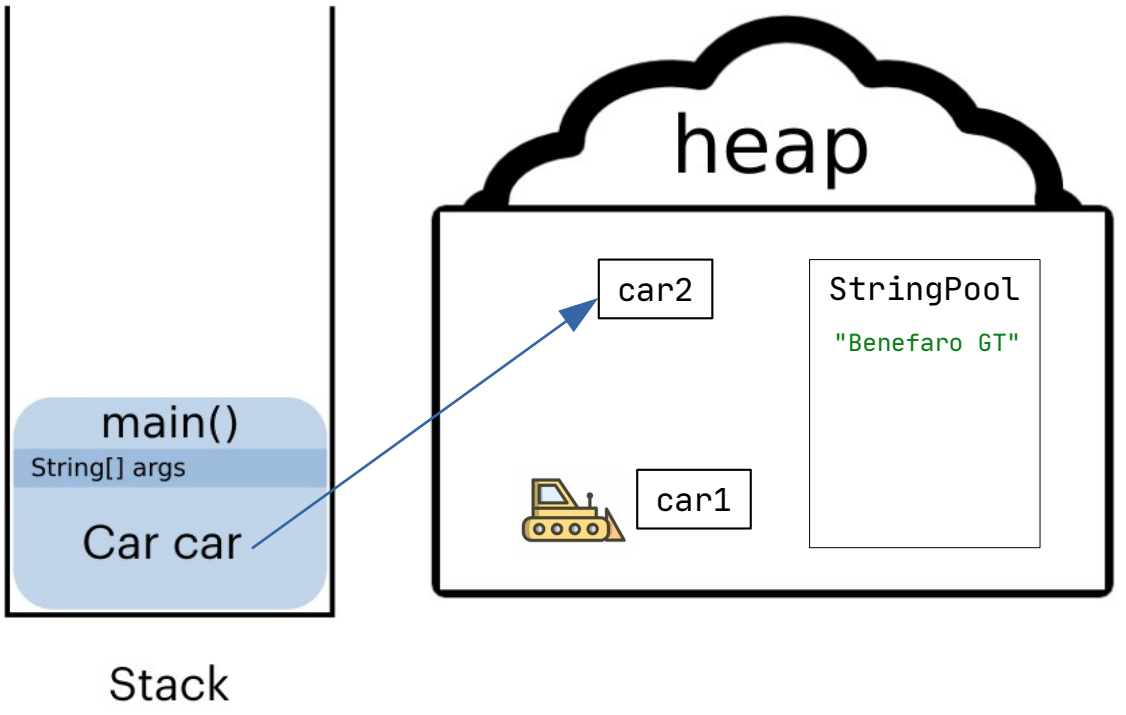


Куча

```
public class Main {

  public static void main(String[] args) {
    Car car = new Car();
    car.setName("Benefaro GT");
    car = new Car();
  }
}
```

Объект доступен в программе до тех пор пока существует хотя бы одна ссылка на него.

Объекты на которые нет ссылок – уничтожаются сборщиком мусора (Garbage Collector) если в Heap заканчивается место для новых объектов.



Skillbox

Подробнее про переполнение и варианты предотвращения:
https://wiki.sei.cmu.edu/confluence/display/java/NUM00-J.+Detect+or+prevent+integer+overflow

Расчеты с использованием double, float. Причины неточности таких расчетов:
https://habr.com/ru/post/219595/
https://habr.com/ru/post/112953/

Перегрузка методов:
https://habr.com/ru/company/otus/blog/428307/

Heap и stack:
https://javadevblog.com/chto-takoe-heap-i-stack-pamyat-v-java.html