Программирование на языке Java

Тема 21. Статические методы

Функции и библиотеки

Модульное программирование

- Организация программы в виде независимых модулей, которые выполняют совместную работу.
- Почему? Проще делиться и повторно использовать код для создания больших программ.
- Возможность независимо разрабатывать отдельные части больших программ.

Библиотека – набор функций. Обычно библиотека связана с какой-то предметной областью.

Алгоритм

Алгоритм – последовательность шагов для решения некоторой задачи.

Пример алгоритма изготовления печения:

- Смешайте сухие ингредиенты;
- Взбейте сахар и масло;
- Добавьте во взбитую массу яйца;
- Добавьте сухие ингредиенты;
- Разогрейте духовку до температуры 180 градусов;
- Установите таймер на 10 минут;
- Разложите печенье на противень;
- Выпекайте печенье;
- Смешайте ингредиенты для украшения
- …

Проблемы с алгоритмами

- Недостаток структурированности – много шагов.
- **Избыточность**: рассмотрим алгоритм изготовления двух порций печенья
  - Смешайте сухие ингредиенты;
  - Взбейте сахар и масло;
  - Добавьте во взбитую массу яйца;
  - Добавьте сухие ингредиенты;
  - Разогрейте духовку до температуры 180 градусов;
  - Установите таймер на 10 минут;
  - Разложите печенье на противень;
  - Выпекайте печенье;
  - Разогрейте духовку до температуры 180 градусов;
  - Установите таймер на 10 минут;
  - Разложите печенье на противень;
  - Выпекайте печенье;
  - Смешайте ингредиенты для украшения
  - …

Структурированный алгоритм

- Структурированный алгоритм разбивает решение задачи на **отдельные логические шаги**:

1. Приготовление теста:
   - Смешайте сухие ингредиенты;
   - Взбейте сахар и масло;
   - Добавьте во взбитую массу яйца;
   - Добавьте сухие ингредиенты.
2. Выпечка:
   - Разогрейте духовку до температуры 180 градусов;
   - Установите таймер на 10 минут;
   - Разложите печенье на противень;
   - Выпекайте печенье;
3. Украшение печенья:
   - Смешайте ингредиенты для украшения
   - …

Избавление от избыточности

Хорошо структурированный алгоритм может описывать повторяющиеся действия **без избыточности**.

1. Приготовление теста:
   - Смешайте сухие ингредиенты;
   - …
2. Выпекание:
   - Разогрейте духовку до температуры 180 градусов;
   - …
3. Выпекание:
   - Повторить шаг 2
4. Украшение печенья:
   - Смешайте ингредиенты для украшения
   - …

Статические методы

Статический метод – конструкция Java для создания вспомогательных алгоритмов, подпрограмм.

Во многих языках программирования статические методы называются функциями и процедурами.

Каждый статический метод — последовательность операторов, которые при вызове статического метода выполняются один за другим.

Слово **статические** отличает эти методы от **методов экземпляров** (будут рассмотрены позже в теме объектно-ориентированное программирование).

Статические методы

Метод может быть **многократно вызван** из разных частей программы.

Процедурная декомпозиция – представление разрабатываемой программы в виде совокупности вызывающих друг друга подпрограмм.

При вызове метода управление передается на соответствующий участок программного кода. После выполнения метода осуществляется возврат на оператор основной программы, следующий за вызовом метода.

Примеры методов

- Встроенные методы Java, например **`Math.random()`**, **`Math.abs()`**, **`Math.min()`**.
- Методы чтения данных **`in.nextInt()`**, **`in.nextDouble()`**
- Методы, определенные программистом, например метод **`main()`**.

Библиотека

- Создание статического метода это как добавление новой команды Java.
- Библиотека – набор функций (статических методов).

Объявление статического метода

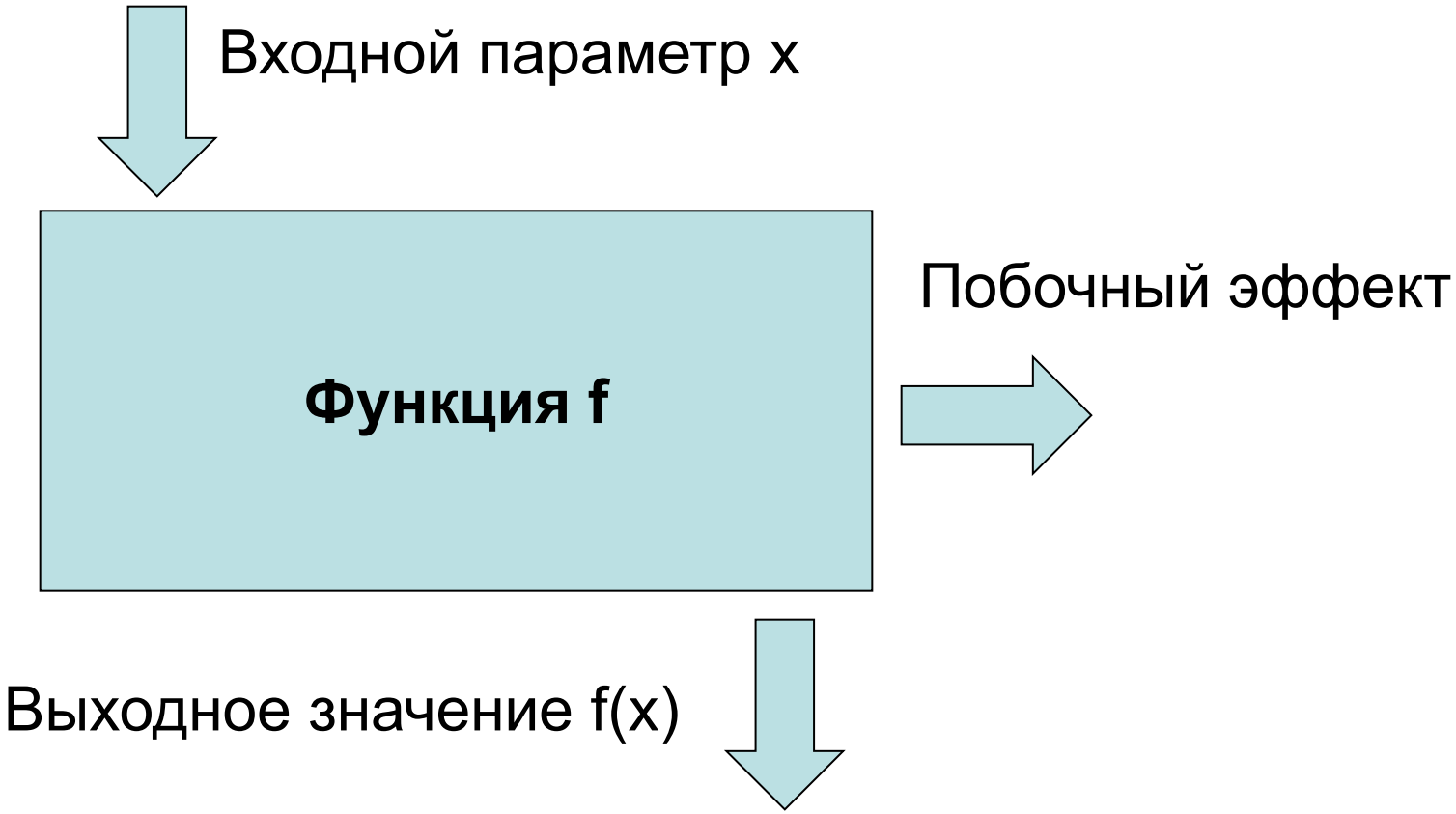
Метод содержит вычисление, которое определено как последовательность операторов.

Метод принимает **аргументы** (значения заданных типов данных, их может быть 0 и больше) и на основе этих аргументов:

- вычисляет **возвращаемое значение** определенного типа данных или
- вызывает **побочный эффект**, который зависит от аргументов (например, вывод значения).

Статический метод **`main()`** является примером метода второго типа (не возвращает значения).

Функция (статический метод)



Определение статического метода

Каждый статический метод состоит из

сигнатуры –

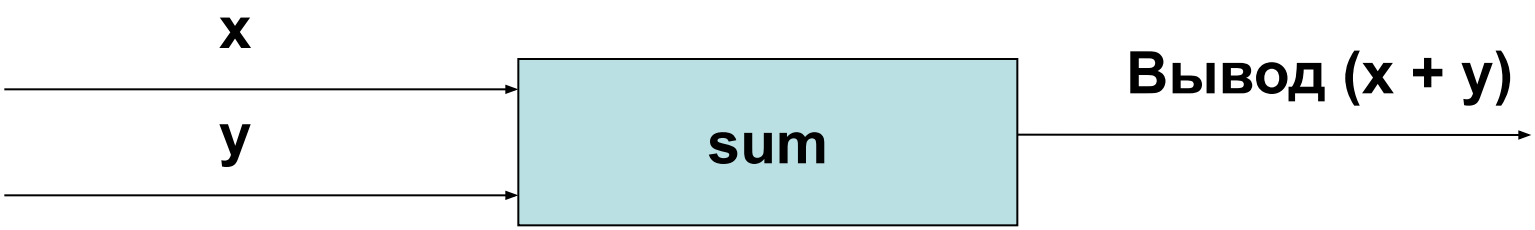
- ключевые слова **`public static`**,
- тип возвращаемого значения
- имя метода
- последовательность аргументов, каждый с объявленным типом в круглых скобках.

тела – последовательность операторов, заключенная в фигурные скобки.

Пример статического метода

Задача: составить метод, который вычисляет сумму двух значений выводит результат на экран.

Статический метод:




```
public static void sum (int x, int y) {
   int z = x + y;
   System.out.println(z);
}
```

Сигнатура метода

Тело метода

Описание метода, не возвращающего значение

Особенности:

- в сигнатуре ставится **ключевое слово `void`**, которое означает, что метод не возвращает значения

```
public static void sum (int x, int y)
```

- в сигнатуре описываются формальные параметры, они обозначаются именами с указанием типа параметра

```
public static void qq ( int a, float x )
```

Описание метода, не возвращающего значение

Особенности:

- можно объявлять и использовать **локальные переменные**

```
public static void test(int a, int b)
{
  float x, y;
...
}
```

локальные переменные

Локальные переменные недоступны в основной программе и других методах

Вызов статического метода

Вызов статического метода – это его имя, за которым в скобках следуют выражения, задающие значения аргументов, разделенные запятыми.

При вызове метода его переменные аргументов инициализируются значениями соответствующих выражений из вызова.

Программа

```
public static void sum(int x,int y)
{
  ...
}
public static void main(String[] args)
{
  int a, b, c = 0;
  a = in.nextInt();
  b = in.nextInt();
  sum (a,  b);
}
```

формальные параметры

фактические параметры

вызов метода

Использование статических методов

1. **Спроектировать (обдумать) алгоритм**
   - Посмотреть на структуру, отследить какие команды повторяются
   - Выделить основные части алгоритма
2. **Объявить методы (записать их в программе)**
   - Упорядочить выражения в группы, дать каждой группе имя.
3. **Вызвать методы**
   - Метод `main()` будет вызывать остальные методы для решения задачи.

Проектирование алгоритма

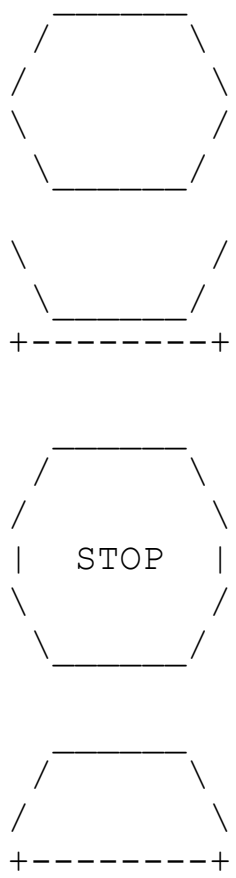
```
// Шаг 1: Приготовление теста
System.out.println("Смешайте сухие ингредиенты");
System.out.println("Взбейте сахар и масло");
System.out.println("Добавьте во взбитую массу яйца");
System.out.println("Добавьте сухие ингредиенты");

// Шаг 2а: Выпекание печенья (первый противень)
System.out.println("Разогрейте духовку");
System.out.println("Установите таймер на 10 минут");
System.out.println("Разложите печенье на противень");
System.out.println("Выпекайте печенье");

// Шаг 2б: Выпекание печенья (второй противень)
System.out.println("Разогрейте духовку ");
System.out.println("Установите таймер на 10 минут ");
System.out.println("Разложите печенье на противень");
System.out.println("Выпекайте печенье");


// Шаг 3: Украшение печенья
System.out.println("Смешайте ингредиенты для украшения");
System.out.println("Украсьте печенье");
```

Итоговый алгоритм

```
public static void main(String[] args) {
    makeBatter();
    bake();         // 1-ый противень
    bake();         // 2-ой противень
    decorate();
}

// Шаг 1: Приготовление теста
public static void makeBatter() {
    System.out.println("Смешайте сухие ингредиенты");
    System.out.println("Взбейте сахар и масло");
    System.out.println("Добавьте в массу яйца");
    System.out.println("Добавьте сухие ингредиенты");
}

// Шаг 2: Выпекание одного противня печенья
public static void bake() {
    System.out.println("Разогрейте духовку");
    System.out.println("Установите таймер на 10 минут");
    System.out.println("Разложите печенье на противень");
    System.out.println("Выпекайте печенье");
}

// Шаг 3: Украшение
public static void decorate() {
    System.out.println("Смешайте ингредиенты для украшения");
    System.out.println("Украсьте печенье");
}
```

Задача

Задача. Напишите программу, которая выводит на экран следующие фигуры.



Идея решения 1

Неструктурированная версия

- Создать метод **`main`**
- Скопировать ожидаемый вывод в программу, окружить оператором **`System.out.println`**

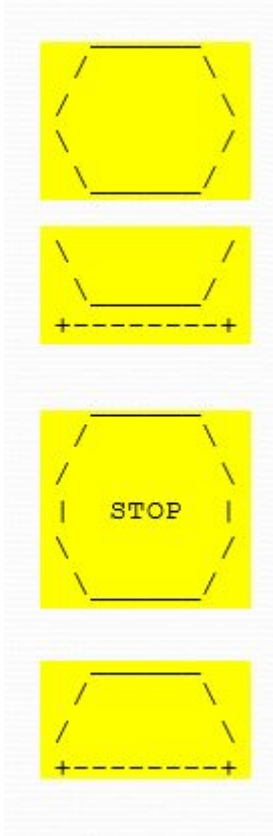
Вариант решения 1

```
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+--------+");
    }
}
```

Идея решения 2

Структурированная версия с избыточностью

– Выделить печать каждой фигуры в отдельный метод



Создадим методы:

- **`printEgg`**
- **`printTeaCup`**
- **`printStopSign`**
- **`printHat`**

Вариант решения 2

```
public class Figures2 {
    public static void main(String[] args) {
        printEgg();
        printTeaCup();
        printStopSign();
        printHat();
    }

    public static void printEgg() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
    }

    public static void printTeaCup() {
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println("+--------+");
        System.out.println();
    }
    ...
```

Вариант решения 2

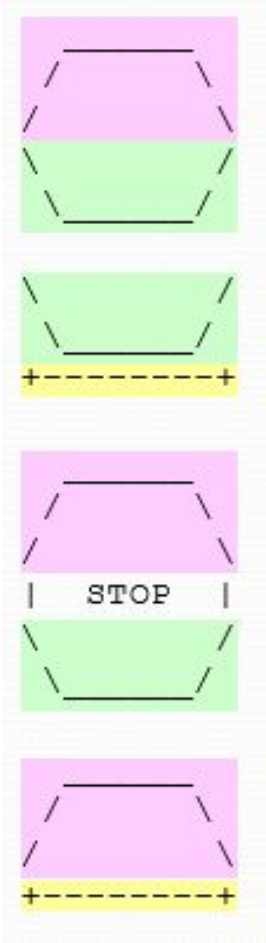
```
...

    public static void printStopSign() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
    }

    public static void printHat() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+---------+");
    }
}
```

Что плохо?

Идея решения 3

Структурированная версия без избыточности

– Выделить избыточность в выводе, создать методы без повторений



Создадим методы:

- **`printEggTop`** – используется в 3 фигурах
- **`printEggBottom`** – используется в 3 фигурах
- **`printLine`** – используется в 2 фигурах

Вариант решения 3

```
public class Figures3 {
    public static void main(String[] args) {
        printEgg();
        printTeaCup();
        printStopSign();
        printHat();
    }

    // Рисует верхнюю часть фигуры Яйцо
    public static void printEggTop() {
        System.out.println("    _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
    }

    // Рисует нижнюю часть фигуры Яйцо
    public static void printEggBottom() {
        System.out.println("\\         /");
        System.out.println(" \\_______/");
    }

    // Рисует фигуру Яйцо
    public static void printEgg() {
        printEggTop();
        printEggBottom();
        System.out.println();
    }

    ...
```

Вариант решения 3

```
...
    // Рисует фигуру Чашка
    public static void printTeaCup() {
        printEggBottom();
        printLine();
        System.out.println();
    }
    // Рисует знак Стоп
    public static void printStopSign() {
        printEggTop();
        System.out.println("|  STOP  |");
        printEggBottom();
        System.out.println();
    }
    // Рисует фигуру Шляпа
    public static void printHat() {
        printEggTop();
        printLine();
    }
    // Рисует линию
    public static void printLine() {
        System.out.println("+--------+");
    }
}
```

Процедурная декомпозиция (структурирование)

Правило. Когда вы можете четко разделить задачи в своей программе – сделайте это.

Глобальные и локальные переменные

Глобальные переменные описываются в классе (вне методов).

Локальные переменные создаются в теле метода, они существуют только в течение времени выполнения метода, определяются при его вызове и «исчезают» после завершения работы метода.

Область видимости

Областью видимости переменной называется та часть программы, которая может обращаться к этой переменной по имени.

В Java: область видимости переменных, объявленных в блоке, ограничивается строками этого блока. Такие переменные называют **локальными.**

Хороший стиль программирования – использование преимущественно локальных переменных.

Пример. Область видимости

```
public static void printSum(int x, int y)
{
 int sum = x + y;
 print(sum);
}
```

Область видимости переменных x, y, sum

```
public static void main(String[] args) {
   int x, sum;
   x = in.nextInt();
   sum = in.nextInt();
   printSum( x, sum );
}
```

Область видимости переменных x, sum

Формальные и фактические параметры

Список **формальных параметров** указывается в сигнатуре метода.

```
public static void sum (int x, int y)
```

Каждый такой параметр является локальным (т.е. к нему можно обращаться только в пределах данного метода).

Фактические параметры – параметры, которые передаются методу при обращении к нему.

```
int a = in.nextInt(), b = in.nextInt();
sum (a, b);
sum (1, 10);
```

Внимание! Количество и типы формальных и фактических параметров **должны совпадать**.

Что неправильно?

Какое имя лучше дать методу?

```
public static void sum (int x, int y, int z) {
  int u = x * y * z;
  System.out.printf("%d*%d*%d=%d", x, y, z, u);
 }
public static void main (String[] args) {
  sum (1,2,3);
 }
```

Метод, возвращающий значение

Метод, возвращающий значение – это вспомогательный алгоритм, результатом работы которого является некоторое значение.

Примеры:

- **вычисление модуля числа,** $\sqrt{x}$
- **расчет значений по сложным формулам**
- **ответ на вопрос (простое число или нет?)**

Зачем?

- **для выполнения одинаковых расчетов в различных местах программы**
- **для создания общедоступных библиотек методов**

Метод, возвращающий значение

Задача: составить метод, который вычисляет и возвращает наибольшее из двух значений

Метод:

формальные параметры

```
public static int max ( int a, int b )
{
  if ( a > b ) return a ;
  else         return b ;
}
```

return - вернуть результат

Метод, возвращающий значение

Особенности:

- в сигнатуре указывается **тип результата**

```
public static int max ( int a, int b )
```

- В сигнатуре описываются формальные параметры, они обозначаются именами и типами

```
public static float qq ( int a, float x )
```

Метод, возвращающий значение

Особенности:

- Метод возвращает единственное значение, но может содержать несколько операторов возврата.
- Java-метод может возвратить только одно значение – того типа, который объявлен в сигнатуре метода.
- Управление возвращается в вызывающую программу как только в методе достигается первый оператор **`return`**.

Программа

```
public static int max ( int a, int b )
{
  ...
}
public static void main(String[] args)
{
 int a, b, c;
 a = in.nextInt ();
 b = in.nextInt();
 c = max ( a, b );
 System.out.printf("max = %d", c);
}
```

формальные параметры

фактические параметры

вызов метода

Логические методы

Задача: **составить метод, который определяет, верно ли, что заданное число – простое.**

Особенности:

- ответ – **логическое** значение: **`true`** (да) или **`false`** (нет)
- результат метода можно использовать как логическую величину **в условиях** (**`if, while`**)

Алгоритм: считаем число делителей в интервале от **2** до **N-1**, если оно не равно нулю – число составное.

```
int count = 0;
for (int i = 2; i < N; i ++)
  if ( N % i == 0 ) count ++;
if ( count == 0 )
     // число N простое
else // число N составное
```

Как улучшить?

Логические методы

```
public static boole
{
int count = 0, i;
for (i = 2; i*i <= N; i++)
  if (N % i == 0) count ++;
return (count == 0);
}
```

перебор только до $\sqrt{N}$

Как улучшить?

```
if (count == 0) return true;
else            return false;
```

```
public static void main(String[] args)
{
  int N;
  N = in.nextInt();
 if ( isPrime(N) )
     System.out.printf ("%d - простое число",N);
  else System.out.println ("%d - составное число",N);
}
```

вызов метода

Стиль и использование методов

- Тщательно структурируйте ваш код
- Избегайте избыточности кода
- Следуйте соглашениям по именованию методов
- Используйте комментарии для описания поведения кода

Задача 1

Что будет выведено на экран при запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        int j = i * i * i;
        return j;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

```
1 - 1
2 - 8
3 - 27
4 - 64
5 - 125
```

Задача 2

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        int i = i * i * i;
        return i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

Ошибка! Попытка объявить уже объявленную переменную `i`

Задача 3

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        i = i * i * i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

Ошибка! Отсутствует возвращаемое значение

Задача 4

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        i = i * i * i;
        return i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

```
1 - 1
2 - 8
3 - 27
4 - 64
5 - 125
```

Задача 5

Что будет выведено на экран при компиляции и запуске программы?

```
public class Cubes {

    public static int cube(int i) {
        return i * i * i;
    }

    public static void main(String[] args) {
        int N = 5;
        for(int i = 1; i<= N; i++)
            System.out.printf("%d - %d\n", i, cube(i));
    }

}
```

```
1 - 1
2 - 8
3 - 27
4 - 64
5 - 125
```

Задания

1. Написать метод, который возвращает сумму всех чисел от 1 до N и привести пример его использования.

Пример:

```
Введите число:
100
Ответ: сумма чисел от 1 до 100 = 5050
```

2. Написать метод, который принимает в качестве параметров два целых числа и возвращает наибольшее значение модуля числа.

Пример:

```
Введите число: 4
Введите число: -5
Ответ: 5
```

Задания

3. Написать метод, который принимает в качестве параметра четыре значения целого типа `x1, y1, x2, y2` и возвращает расстояние между точками `(x1,y1)` и `(x2,y2)`.

Пример:

```
Введите координаты:
0 3 4 0
расстояние между точками (0, 3) и (4, 0) равно 5
```

4. Написать метод, который принимает в качестве параметра два значения типа: мантиссу и порядок и возвращает десятичную запись этого числа.

Пример:

```
Введите мантиссу: 6,23
Введите порядок: 5
Ответ: 623000.0
```

Программирование на языке Java

Тема 22. Перегрузка методов

Перегрузка методов

Сигнатура метода – совокупность его имени и набора формальных параметров.

Java позволяет создавать несколько методов с одинаковыми именами, но разными сигнатурами.

Создание метода с тем же именем, но с другим набором параметров называется **перегрузкой**.

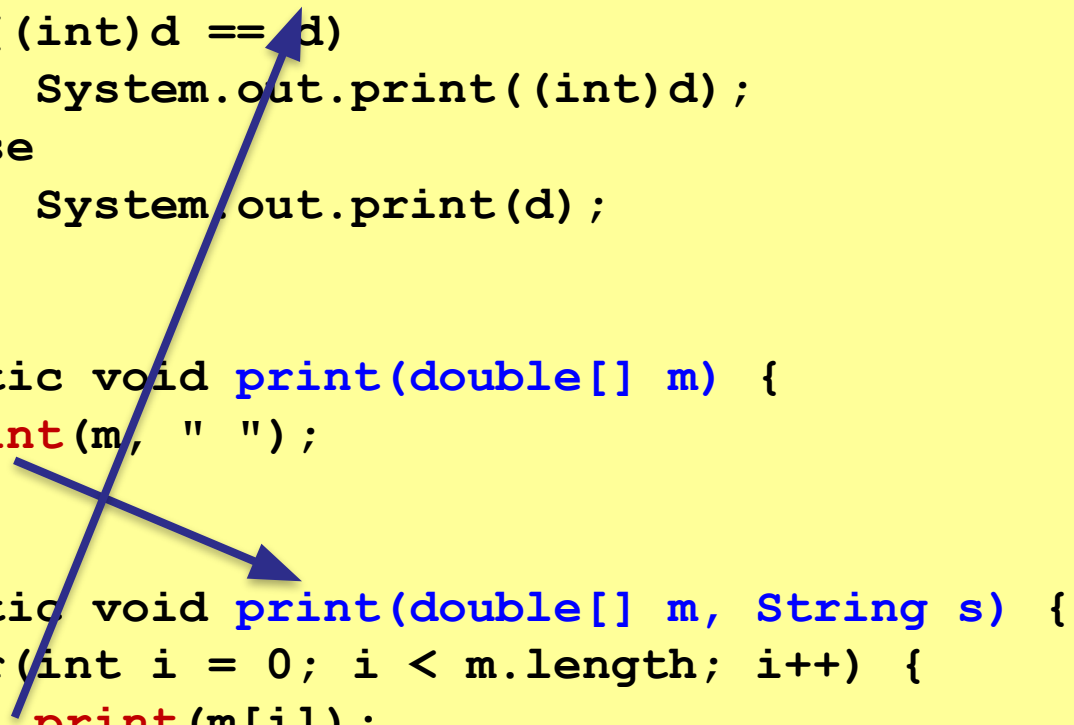
Какой из перегруженных методов должен выполняться при вызове, Java определяет на основе фактических параметров.

Перегрузка методов. Пример – 1

```
pubilc void print(double a) {
 System.out.println(a);
}
pubilc void print(String a) {
  System.out.println(a);
}
pubilc void print(int[] a) {
  for (int i=0; i<a.length; i++) {
    System.out.printf("%d ",a[i]);
  }
  System.out.println("");
}
...
int a = 5;
int [] m = {1, 2, 8, 3}
String s = "Мир";
print (a) // работает исходный метод
print (a + s); // 5 мир, работает первая перегрузка
print (m); // 1 2 8 3
print (m + a); // ошибка
```

Перегрузка методов. Пример – 2

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

public static void print(double d) {
        if((int)d == d)
            System.out.print((int)d);
        else
            System.out.print(d);
}

public static void print(double[] m) {
        print(m, " ");
}

public static void print(double[] m, String s) {
        for(int i = 0; i < m.length; i++) {
            print(m[i]);
            System.out.print(s);
        }
}
```



Перегрузка методов. Пример – 2

```
public static void main(String[] args) {
        double[] a = {1.0, 2.71, 3.14, 15, -5, 92, 0.5};
        double p = 3.0;
        int k = 13;
        print(p);
        print();
        print(a);
        print();
        print(a,", ");
        print();
        print(k);
    }
```

```
3
1 2.71 3.14 15 -5 92 0.5
1, 2.71, 3.14, 15, -5, 92, 0.5,
13
```