Лекция 2.
Строки.
Арифметика.
Класс Math
Чтение с консоли

Строки

- Литералы строк заключаются в двойные кавычки: "Test", "Строка"
- Строки имеют тип String

```
String s = "Hello";
```

- Использование кавычек внутри литерала строки (пишется \"):

```
String s = "\"Hello\" world";   // "Hello" world
```

- Это называется **экранированием**

Специальные последовательности

- Литералы строк могут содержать специальные последовательности (есть и другие):
  - \n перевод строки (Enter)
  - \r возврат каретки
  - \t табуляция
  - \\ символ \
- Пример:

  "This is first line\nThis is second line"
- Результат печати:

  This is first line

  This is second line

Конкатенация строк

- В Java строки можно объединять друг с другом при помощи оператора +
- Это называется **конкатенацией** строк (добавление второй строки в конец первой)
- `String s1 = "Hello " + "world";  // Hello world`
- Кроме того, если к строке прибавлять значения других типов, то они будут преобразовываться в строки, и эти строки будут конкатенироваться
- `String s1 = "Hello " + 3;  // Hello 3`
- `String s2 = 3 + " tigers";  // 3 tigers`

Переводы строк

- В разных операционных системах в текстовых файлах перевод строки обозначается разным образом
- В UNIX-системах это просто \n
- В Windows – последовательность из двух символов \r\n
- Так как программы на Java исполняются на многих ОС, то есть команда для получения нужного перевода строки, в зависимости от системы, где выполняется программа:
- System.lineSeparator()

Пример использования переводов строк

- Не совсем верно – всегда используется UNIX-стиль
- `System.out.println("Строка 1\nСтрока 2");`

- Верно (на каждой платформе свой перевод строки):
- `System.out.println("Строка 1" + System.lineSeparator()`
  `+ "Строка 2");`

- Тоже верно, и этот вариант лучше:
- `System.out.println("Строка 1");`
  `System.out.println("Строка 2");`
- separator лучше избегать, т.к. код получается громоздкий

Задача

1. Вывести при помощи одного вызова System.out.println сразу несколько строк при помощи перевода строки
2. Вывести следующие строки:

   "30" августа 1903г.
   C:\Windows\System32\Drivers\etc\hosts
3. Посчитать некоторое выражение и распечатать его результат в виде строки. Вместо 4 – ваш результат:
   Результат = 4

Печать результатов

- Предположим, вы посчитали результат некоторого выражения:
- ```
  int x = 8;
  int y = x * 2;
  ```
- Дальше вы хотите распечатать пользователю **y**

- Вот так делать неправильно:
- ```
  System.out.println(y);
  ```
- Потому что пользователю выведется просто 16
- И ему будет не понятно что это вообще такое

Печать результатов

- ```
  int x = 8;
  int y = x * 2;
  ```
- Поэтому при выводе результатов важно выводить пользователю текст, чтобы он понимал что именно выводится
- ```
  System.out.println("y = " + y);
  ```
- Тут используется конкатенация, чтобы вывести и текст и значение переменной

- Если вычисляется что-то осмысленное, то и текст должен быть более осмысленным

Числа. Арифметика

Числа и арифметика

- В Java есть несколько разных типов для целых и вещественных чисел

Название типа	Диапазон значений	Размер в байтах
byte	-128..127	1
short	-32768..32767	2
int	-2147483648.. 2147483647	4
long	–9223372036854775808.. 9223372036854775807	8
float	±1.4e-45 .... 3.4e+38	4
double	±4.9e-324 .... 1.7e+308	8

- Для целых чисел чаще всего используют int, для вещественных чисел double

Литералы чисел

- Литерал числа считается вещественным, если в нем есть точка, либо символ экспоненты

```
1.0          // вещественное число 1.0 (double)
1            // целое число 1 (int)
1.           // вещественное число 1.0
0.4          // вещественное число 0.4
0.4d         // вещественное число 0.4
.1           // вещественное число 0.1
1e34         // 1 * 10 в степени 34
-1.43e-3     // -1.43 * 10 в степени -3
1.0f         // вещ. число 1.0, но float
1L           // long 1.0
```

Символ _ в литералах чисел

- Начиная с Java 7 в литералах чисел можно использовать символ _ между любыми цифрами для читаемости
- Эти варианты эквивалентны:
- 1000000
  1_000_000
- Это можно использовать и в вещественных числах
- Гипотетически, можно писать несколько символов _ подряд

Числа и арифметика

- Арифметические операторы: + - * / %
- % - остаток от целочисленного деления
  ```
  int r = 25 % 7;   // 4
  ```
- Приоритеты операторов – как в математике
- Можно ставить скобки в выражениях для повышения приоритета

- Пример: 5 + 5 * (3 – 2)

Важное замечание

- Если оба числа целые, то деление будет целочисленным. Если хотя бы одно число вещественное, то деление будет вещественным:

- 5 / 2 // 2, т.к. оба числа целые
  5.0 / 2 // 2.5, т.к. первое число – вещ.

Как поделить вещественно целые числа?

- int a = 5;
- int b = 2;
- double c = (double)a / b;

- (double)a – называется **преобразованием типа (приведение типа)**
- Выражение (double)a выдает значение переменной a как вещественное число

Задача

- Попрактиковаться с арифметическими операторами: сделать по 1 примеру для каждого оператора для целых чисел и для вещественных чисел, вывести результат в консоль
- Операторы: + - * / %
- Проверить как работает целочисленное и вещественное деление

Приоритеты операций

- Приоритеты операций как в математике:
  - Сложение и вычитание имеют равный приоритет, выполняются слева направо
  - Умножение и деление имеют равный приоритет, выполняются слева направо
  - Приоритет умножения и деления выше, чем у сложения и вычитания
  - Скобки повышают приоритет операций

Устная практика

- Чему равен x?
- $x = 3 + 5 * 2$

- Как записать на Java следующее выражение?
- $\frac{3x+5}{2y} + 3y$

Задача на дом «Вычисление выражений»

- Посчитайте на Java следующие выражения:
- $x = 3 - \frac{56 - 12}{44} * \frac{4}{2}$
- $y = \frac{2x}{33 - x}$
- $z = \frac{-x}{2y}$

- * При вычислении **x** у вас может возникнуть warning, его поправлять не нужно

Переполнение типов

- Тип имеет ограниченный диапазон значений, и если за него выйти, то происходит переполнение
- ```
  int x = 2147483647;
  System.out.println(x);
  // 2147483647
  ```
- ```
  x = x + 1;
  System.out.println(x);
  // -2147483648
  ```
- Почему получилось именно такое число?

Хранение чисел

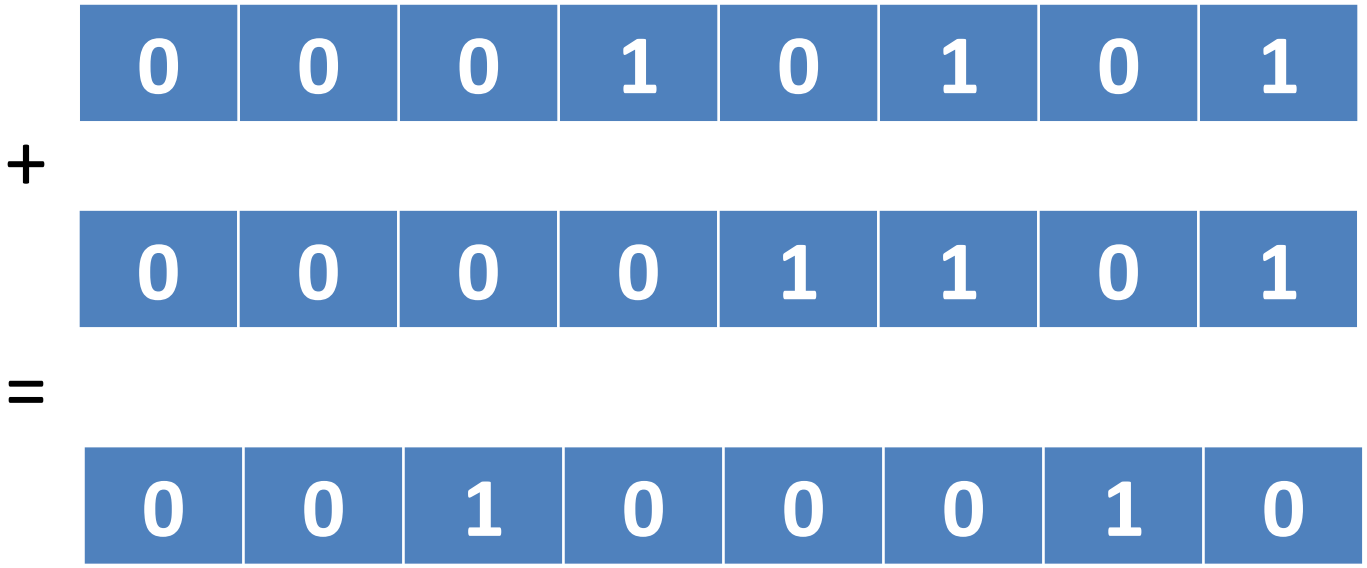
- Все числа хранятся в двоичной системе счисления, т.е. представляются последовательностью нулей и единиц
- Например, в переменной типа byte 8 бит. Рассмотрим, например, число 127 – максимальное значение типа byte

0	1	1	1	1	1	1	1

- Причем в целых числах первый бит предназначен для хранения знака числа. Этот бит равен 0 для положительных чисел, 1 – для отрицательных

Операция сложения

- Сложение двух чисел выполняется побитово
- Допустим, мы складываем числа 21 и 13, получаем 34



Операция сложения

- Сложим 127 и 1, получается уже некоторое отрицательное число, т.к. первый бит получился 1. И это число -128
- Но почему это именно -128?

	0	1	1	1	1	1	1	1
+								
	0	0	0	0	0	0	0	1
=								
	1	0	0	0	0	0	0	0

Дополнительный код

- Так выглядит число 127

- Так выглядит число -127

- Почему -127 выглядит именно так?
- А выглядит так, чтобы если сложить 127 и -127, получился 0

- Первая единица выходит за границы 8 бит, и поэтому отбрасывается
- Такой способ хранения отрицательных чисел называется **дополнительным кодом**

Переполнение типов

- ```
  int x = 2147483647;
  System.out.println(x);
  // 2147483647
  ```
- ```
  x = x + 1;
  System.out.println(x);
  // -2147483648
  ```
- Почему получилось именно такое число?
- Потому что число в результате сложения происходит переполнение, в двоичном виде получилось число 10000…0, а оно соответствует самому маленькому отрицательному числу в типе

Приведение вещественных типов к целым

- `int c = (int)3.5;  // 3`
- При таком приведении отбрасывается дробная часть
- `int d = (int)-3.7; // -3`
- Не нужно путать с округлением

Преобразования числовых типов

- Преобразования числовых типов можно поделить на две группы:
  - **Безопасные** (без потери данных)
  - **Небезопасные** (с потерей данных)

Безопасные преобразования

- Без потери данных
- Сюда относятся:
  - Преобразование типа с меньшим диапазоном в тип с большим диапазоном
    - Например, short в int или float в double
  - Преобразования целых чисел в вещественные:
    - Например, long в float и double
- Безопасные преобразования можно выполнять автоматически (**неявно**)
- int x = 3;
  long y = x;

Небезопасные преобразования

- С возможной потерей данных
- Сюда относятся:
  - Преобразование типа с большим диапазоном в тип с меньшим диапазоном
    - Например, int в short или double в float
  - Преобразования вещественных чисел в целые:
    - double в int
- Небезопасные преобразования можно выполнять только **явно** при помощи преобразования типов
- long a = 3;
  int b = (int)a;

Пример

- ```
  float x = 3;
  double y = x;      // безопасно, неявное преобр-е
  ```

- ```
  long a = 3;
  int b = a;          // небезопасно, ошибка компиляции
  int b = (int) a;    // надо преобразовывать явно
  ```

Класс Math

- В Java имеется класс Math для выполнения более сложных арифметических операций, чем + - * / %

- Например:
  - возведение в степень
  - извлечение квадратного корня
  - тригонометрические функции
  - логарифмы
  - округление

Класс Math

- Квадратный корень:
  ```
  double a = Math.sqrt(4);  // 2
  ```
- Возведение в степень:
  ```
  double a = Math.pow(2, 3);  // 8
  ```
- Модуль числа:
  ```
  double m = Math.abs(-3.1);  // 3.1
  ```
- Логарифмы:
  ```
  Math.log(x)      // натуральный логарифм
  Math.log10(x)  // десятичный логарифм
  ```

Класс Math

- Константы Пи и e:

```
double pi = Math.PI;
double e = Math.E;
// не создавайте для них свои переменные, используйте
// сами Math.PI и Math.E напрямую
```

- Преобразование между градусами и радианами:

```
double degree1 = 30;
double radians = Math.toRadians(degree1);
double degree2 = Math.toDegrees(radians);
```

Класс Math

- Тригонометрические функции (в радианах):

```
// сначала переводим угол в радианы
double angle = Math.toRadians(30);

double sin = Math.sin(angle);    // примерно 0.5
double cos = Math.cos(angle);  // примерно 0.866
double tan = Math.tan(angle);  // примерно 0.577
```

Класс Math

- Имеются обратные тригонометрические функции и гиперболические функции

Виды округления

- В классе Math есть 3 вида округления + есть округление при помощи приведения к целому типу

	(int)	Math.floor	Math.ceil	Math.round
Пояснение	Отбрасывает дробную часть	Округляет вниз по числовой оси	Округляет вверх по числовой оси	Округляет к ближайшему целому. Если 0.5, то вверх по числовой оси
1.1	1	1	2	1
1.5	1	1	2	2
1.9	1	1	2	2
-1.1	-1	-2	-1	-1
-1.5	-1	-2	-1	-1
-1.9	-1	-2	-1	-2

Задача

- Написать программу, которая вычисляет и печатает площадь круга с заданным радиусом.
- Значение радиуса задать самим в тексте программы

- Использовать класс Math: тригонометрические функции, возведение в степень, квадратный корень, константа Пи и др.

Задача на дом «Окружность»

- Написать программу, которая вычисляет и печатает:
  - Площадь круга и длину окружности с заданным радиусом. Значение радиуса задать самим в тексте программы
  - Посчитать радиус окружности с заданной площадью круга. Значение площади задать самим в тексте программы
  - * Посчитать площадь сектора с заданными радиусом и углом в градусах.
    Значения радиуса и угла задать в тексте программы
- Использовать класс Math: возведение в степень, квадратный корень, константа Пи и др.
- Имена переменных должны быть хорошими, не ориентируемся на математические обозначения

Четность, кратность

- int r = x % y;
- Когда остаток от деления x на y равен 0, то говорят, что **x кратно y**
- То есть делится нацело

- Число является **четным**, если оно кратно 2
  (то есть остаток от деления на 2 равен 0)
- Иначе – число является **нечетным** (остаток равен 1)

Использование деления и остатка

- Пусть x, y – два некоторых целых числа
- ```
  int a = x / y;
  int r = x % y;
  ```

- Тогда r лежит в интервале от 0 до y – 1 включительно
- Это свойство позволяет «зациклить» номера
- Примеры: круг; положение квартиры на площадке
  ```

Получение цифр числа

- Как получить последнюю цифру числа?
- Просто взять остаток от деления на 10
- ```
  int x = 327;
  int lastDigit = x % 10;   // 7
  ```

- Как получить предпоследнюю цифру числа?
- Поделить на 10, потом взять остаток от деления на 10
- ```
  int x = 327;
  int digit = x / 10 % 10;   // 2
  ```

Получение цифр числа

- Как получить n-ю с конца цифру числа?
- Поделить на 10^{n-1}, затем взять остаток от деления на 10
- ```
  int x = 327;
  int n = 3;
  int digit = (x / (int)Math.pow(10, n - 1)) % 10;   // 3
  ```
  ```

Задача на курс «Квартиры»

- Есть дом с известным количеством этажей и подъездов. Все подъезды одинаковые, на каждом этаже в подъезде 4 квартиры.
- Считаем, что номера квартир на лестничной площадке распределяются так:

- То есть можно сказать, что квартира 1 – ближняя слева, квартира 2 – дальняя слева, квартира 3 – дальняя справа, квартира 4 – ближняя справа
- Прочитать с консоли количество этажей, подъездов и номер квартиры. По введенному номеру квартиры выдать номер подъезда и этажа, где находится эта квартира, а также положение квартиры на лестничной площадке
- Выдать сообщение, если квартиры с таким номером нет в доме

Комментарии

Комментарии

- В коде можно писать **комментарии**
- Это текст внутри программы, который отбрасывается компилятором
- Используются для следующих целей:
  - Поясняющий текст для улучшения понимания кода другими людьми
  - Временное отключение части кода
  - TODO – заметки, что потом надо будет что-то сделать
  - Документирование кода

Однострочные комментарии

- Действуют от // до конца строки

- ```
  // Это комментарий
  int i = 4; // это тоже комментарий, после кода

  // Следующая строка – не является кодом и не выполнится
  // System.out.println(i);
  // это пример временного отключения части кода
  ```
  ```

Многострочные комментарии

- Действуют от /* до */

```
int a = 4; /* первая строка комментария
Вторая строка комментария */
```

- Запрещено использовать вложенные многострочные комментарии /* /* */ */

TODO комментарии

- С точки зрения языка – это обычные комментарии
- Но среды разработки их по-особому трактуют:
  - Выделяют их цветом
  - В IDEA есть специальный раздел с такими комментариями

- ```
  //TODO: потом реализовать чтение с консоли
  int width = 4;
  int height = 3;
  System.out.println(width * height);
  ```
  ```

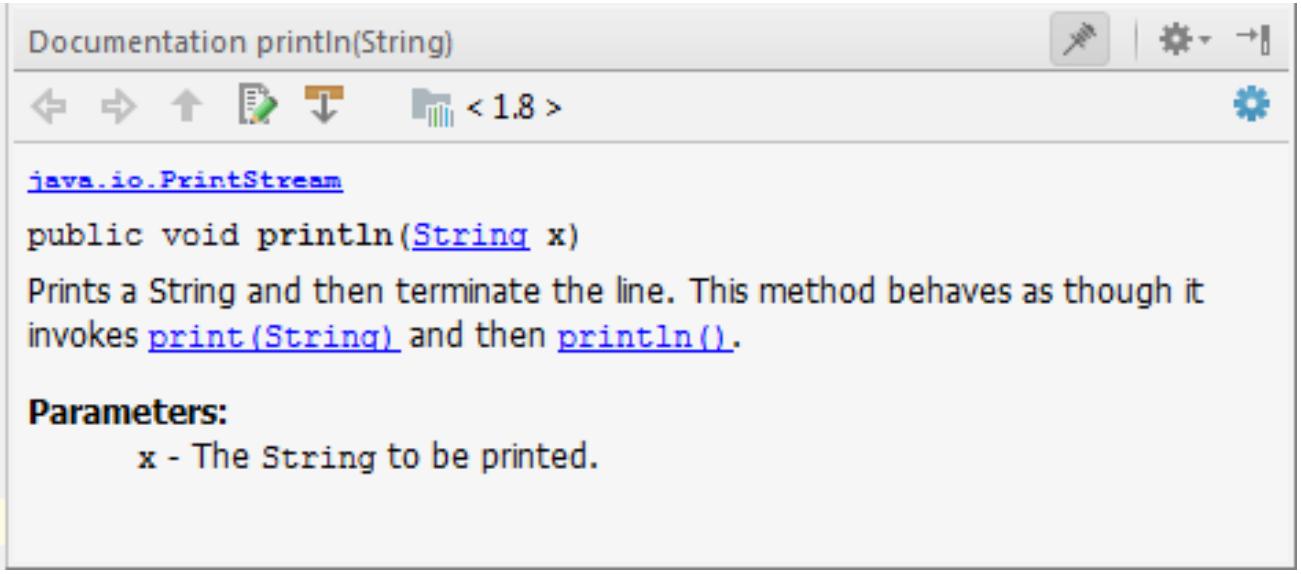
Документирующие комментарии

- Используются для написания документации прямо в коде
- Они многострочные, начинаются с /** и заканчиваются на */

```
/**
 * Prints a String and then terminate the line.  This method behaves as
 * though it invokes <code>{@link #print(String)}</code> and then
 * <code>{@link #println()}</code>.
 *
 * @param x  The <code>String</code> to be printed.
 */
public void println(String x) {
    synchronized (this) {
        print(x);
        newLine();
    }
}
```

Документирующие комментарии

- Если в IDEA поставить курсор на вызов функции, и нажать Cltr+Q, то откроется окно с документацией, взятой из комментариев



Когда нужны комментарии?

- Комментарии пишут достаточно редко
  - Когда в коде принято неочевидное решение, которое, казалось бы, можно упростить, или является неверным, но по некоторой причине «так надо»
  - Временное отключение кода
  - TODO комментарии
  - Документирующие комментарии

- Комментарии не пишутся:
  - Чтобы пояснить имя переменной/функции/класса – имена должны и так быть понятными

Задача

- Прокомментировать текст какой-либо сделанной вами программы всеми типами комментариев
- Обязательно попробуйте TODO комментарии

Чтение с консоли

Чтение с консоли

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

    System.out.print("Введите число: ");
    double a = scanner.nextDouble();

    System.out.print("Введите второе число: ");
    double b = scanner.nextDouble();

    System.out.println("a - b = " + (a - b));
}
```

Импорт классов

- Пример из предыдущего слайда не скомпилируется, IDEA выделит красным слово Scanner
- Это произойдет потому что в Java нет класса с именем Scanner, но есть класс с именем java.util.Scanner
- Использовать длинные имена неудобно, поэтому есть понятие **импорт класса**
- Когда класс импортирован, то можно обращаться к нему по короткому имени, например, просто Scanner

Импорт классов в IDEA

- IDEA позволяет автоматически импортировать класс
- Для этого нужно:
  1. Поставить курсор на имя класса, выделенное красным
  2. Нажать Alt + Enter
  3. В выпадающем списке выбрать **Import class**
  4. Если там несколько вариантов, то надо выбрать нужный. Для Scanner там надо выбрать java.util.Scanner

Чтение с консоли

- Для чтения с консоли используется тип Scanner
- Ключевое слово new создает новый объект типа Scanner
- В скобках передаем ему System.in – источник чтения с консоли
- Далее пользуемся сканнером – читаем при помощи него числа и строки с консоли

- ```
  int a = scanner.nextInt();              // прочитать целое число
  double b = scanner.nextDouble();  // прочитать double
  String s = scanner.nextLine();        // прочитать строку
  ```
  ```

Задание локали

- double a = scanner.nextDouble();
  // вводим в консоль 3.5
  // программа падает, почему?
- Сканнер использует региональные настройки ОС, поэтому по умолчанию в качестве разделителя дробной части в double будет запятая
- Если введем 3,5, то всё будет хорошо

Задание локали

- Сканнеру при желании можно задать региональные настройки:
- Scanner scanner = new Scanner(System.*in*);
  scanner.useLocale(Locale.US);

- Теперь сканнер будет использовать настройки США (например, десятичная точка вместо запятой)

Приглашение для ввода

- Очень важно при чтении с консоли выводить пользователю приглашение для ввода
- Это некоторый текст, по которому пользователь поймет, что именно ему нужно ввести
  - И вообще поймет сам факт, что нужно что-то ввести

- Предположим, просто написан такой код:
  `double a = scanner.nextDouble();`
- Когда программа дойдет до этой строки, она приостановится
- Пользователь никак не поймет, что от него что-то требуется

Приглашение для ввода

- Поэтому правильно делать, например, так:
- ```
  System.out.print("Введите число: ");
  double a = scanner.nextDouble();
  ```

- Теперь программа напечатает пользователю сообщение до начала ввода с консоли, и пользователь поймет, что нужно ввести данные
- И по сообщению поймет что именно нужно ввести

- Естественно, сообщение должно быть более информативным
  ```

Задача

- Написать программу, читающую с консоли длину и ширину прямоугольника, и печатающую его площадь
- * Посчитайте и периметр
- Считать, что длина и ширина – вещественные числа

Задача на дом «Приветствие»

- Написать программу, которая просит ввести ваше имя, а затем выводит в консоль приветствие. Для чтения использовать nextLine() Scanner’а

- Пример:

Введите ваше имя: // это печатает программа

Павел // это ввожу я

Привет, Павел! // это печатает программа

Печать в консоль

Печать в консоль

- System.out.println(аргумент);
- Печатает переданный аргумент, а потом печатает перевод строки (Enter)

- System.out.print(аргумент);
- Печатает переданный аргумент, но не вставляет перевод строки

- System.out.println();
- Печатает перевод строки

Печать в консоль

- System.out.printf(formatString, arg1…);
- Принимает **строку форматирования** первым аргументом, а затем через запятую указываются аргументы, которые будут вставлены в эту строку

- Пример:
- ```
  int a = 7 * 5;
  System.out.printf("Result = %d", a);    // Result = 35
  ```
  ```

Printf

- ```
  int x = 35;
  int y = 34;
  System.out.printf("X = %d, Y = %d", x, y);    // X = 35, Y = 34
  ```

- Строка форматирования – строка, некоторые части которой имеют особый смысл
- Такие части начинаются с символа % и называются **спецификаторами формата**
- Аргументы, которые мы передаем в printf, подставляются вместо этих спецификаторов
- Например, вместо первого %d вставится значение x = 35, вместо второго %d – значение y = 34
  ```

Зачем нужен printf?

- Он позволяет формировать сложные строки из нескольких строк и переменных в читаемом виде

- Пример:
  - **Конкатенация:**
  - System.out.println("X = " + x + ", y = " + y);

  - **Printf:**
  - System.out.printf("X = %f, y = %f", x, y);

Зачем нужен printf?

- Он позволяет вывести данные в разном формате
- Например, у вещественного числа может быть много знаков после запятой
- double x = 33.333333;
- Можно указать сколько знаков вывести, нужно ли вывести в экспоненциальная форме, нужно ли добавлять пробелы для форматирования и т.д.
-
```
System.out.printf("%f", x);     // 33.333333
System.out.printf("%e", x);     // 3.333333e+01
System.out.printf("%.2f", x);   // 33.33
```

Спецификаторы

- Для каждого типа данных используется свой спецификатор, начинающийся с %

Спецификатор	Тип	Пример
%d	Десятичное целое число	159
%x	Шестнадцатеричное целое число	9f
%o	Восьмеричное целое число	237
%f	Вещественное число с точкой	15.9
%e	Вещественное число в экспоненциальной форме	1.59e+01
%s	Строка	Hello
%%	Символ процента	%
%n	Перевод строки	

Примеры использования printf

- Примеры:
- ```
  int result = 50;
  System.out.printf("Result = %d%%", result);
  // Result = 50%
  ```
- ```
  String s = "Hello";
  System.out.printf("Result = %s.", s);
  // Result = Hello.
  ```
- ```
  System.out.printf("First line%nSecond line");
  // First line
  // Second line
  ```
  ```

Дополнительные параметры формата

- Кроме самих спецификаторов формата, можно указать дополнительные флаги, которые влияют на форматирование

- Например:
- ```
  double x = 33.333333;
  System.out.printf("%f", x);      // 33.333333
  System.out.printf("%.2f", x);  // 33.33
  ```

- Флаг .n, где n – целое число, показывает сколько знаков нужно оставить после запятой
  ```

Дополнительные параметры формата

- Можно указать число до десятичной точки, тогда это будет ширина выводимого значения
- Если значение будет занимать меньшее число символов, то перед ним добавятся пробелы

- Например:
- ```
  double x = 33.333333;
  System.out.printf("%10.2f", x);     // _ _ _ _ _ 33.33
  // имеется в виду, что перед числом 5 пробелов
  ```
  ```

Дополнительные параметры формата

- ```
  double x = 33.333333;
  System.out.printf("%10.2f", x);     // _ _ _ _ _ 33.33
  // имеется в виду, что перед числом 5 пробелов
  ```

- Эта опция удобна, чтобы ровно выводить данные разной длины
- ```
  int age = 33;
  System.out.printf("%-10s = %d", "Возраст", age);
  ```
- Минус означает, что пробелы будут справа, а не слева

Дополнительные параметры формата

```
int age = 22;
int weight = 66;
double temperature = 36.66;
String name = "Петр";
System.out.printf("%-11s = %d%n", "Возраст", age);
System.out.printf("%-11s = %d%n", "Вес", weight);
System.out.printf("%-11s = %f%n", "Температура", temperature);
System.out.printf("%-11s = %s%n", "Имя", name);
```

- Возраст = 22
- Вес = 66
- Температура = 36,660000
- Имя = Петр

String.format

- Если хочется просто сформировать строку, но не печатать ее, то можно воспользоваться командой String.format
- Она имеет те же аргументы, что и System.out.printf

- Пример:
- ```
  int x = 44;
  String result = String.format("Result = %d.", x);
  ```
  ```

Задача на дом «Прямоугольник»

- В задаче про площадь прямоугольника вывести длину, ширину и площадь прямоугольника при помощи одного printf
- * Вывести там же и периметр