Структура программы на Java

У класса, с которого начинается программа, может быть любое имя, но у метода main, с которого начинает выполняться программа, всегда один и тот же вид:

```
public class House
{
    public static void main (String[] args)
    {
        КОД МЕТОДА
    }
}
```

Красным подчеркнута неизменяемая часть: объявление метода `main`

public: доступен для всех

static: метод можно вызывать без создания экземпляра класса

void: метод не возвращает значения

String[] args: аргументы командной строки, передаваемые в программу при ее запуске

Scanner

```
Scanner scanner = new Scanner(System.in);
int num = scanner.nextInt();
...
scanner.close()
```

Класс Scanner имеет следующие основные методы:

- **next()** считывает введенную строку
- **nextBoolean()** считывает введенное логическое значение
- **nextInt()** считывает введенное целое число типа int
- **nextFloat()** считывает введенное вещественное число типа float
- **nextDouble()** считывает введенное вещественное число типа double
- **nextLine()** считывает введенную всю строку до символа перехода на новую строку

Scanner

Если считываемый тип данных не соответствует используемому типу метода сканера, то возникнет ошибка времени выполнения программы, например, читаем целое число, а пользователь вводит дробное. Для того чтобы избежать подобных ошибок используются методы проверки считываемых типов:

- boolean **hasNextLine():** вернет true если следующим значением является строка, иначе — false;
- boolean **hasNextInt():** вернет true если значением является значение типа int;
- boolean **hasNextDouble():** вернетtrue если следующим значением является значение типа double;

Логические операторы

Операторы:

! — «отрицание», унарный оператор, меняет значение на противоположное (инвертирует: ложь превращает в истину, а истину — в ложь).

&& — логическое «и» («конъюнкция», «пересечение»), бинарная операция, возвращает истинное значение тогда и только тогда, когда оба операнда истины.

|| — логическое «или» («дизъюнкция», «объединение»), бинарная операция, возвращает истинное значение, когда хотя бы один из операндов истинный.

У логических операторов следующий приоритет: отрицание, конъюнкция, дизъюнкция.

Операторы сравнения

> — оператор «больше»

>= — оператор «больше или равно»

< — оператор «меньше»

<= — оператор «меньше или равно»

!= — оператор «не равно»

== — оператор эквивалентности (равенства)

if - else

Конструкция if-else позволяет выполнять один блок кода, если условие истинно, и другой блок, если оно ложно.

```
if (условие) {
  // Код, который выполняется, если условие истинно
}
else {
  // Код, который выполняется, если условие ложно
}
```

if - else

Пример:

```
int a = 10;
if (a > 0) {
   System.out.println("a положительное число");
}
else {
   System.out.println("a не положительное число");
}
```

if

Оператор if используется для выполнения блока кода, если заданное условие истинно.

```
int x = 10;

if (x > 5) {

   System.out.println("x больше 5");

}
```

if - else if - else

Эта конструкция используется для проверки нескольких условий последовательно.

Если первое условие ложно, проверяется следующее, и так далее.

```
int x = 10;

if (x > 10) {
   System.out.println("x больше 10");
} else if (x == 10) {
   System.out.println("x равно 10");
} else {
   System.out.println("x меньше 10");
}
```

switch/case

```
switch (выражение) {
  case значение1:
    // Код для значения 1
    break;
  case значение2:
    // Код для значения 2
    break;
  default:
    // Код выбора по умолчанию
}
```

После ключевого слова **switch** в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после операторов **case**. И если совпадение найдено, то будет выполняет соответствующий блок **case**.

switch/case

Пример:

```
int day = 1;
switch (day) {
  case 1:
    System.out.println("Понедельник");
    break;
  case 2:
    System.out.println("Вторник");
    break;
  default:
    System.out.println("Неизвестный день»);
}
```

Тернарный оператор

Тернарный оператор (или условный оператор) — это короткий способ записи конструкции if-else.

P ? v1 : v2

Где P — условие, v1 — значение, возвращаемое, если условие истинно, и v2 — значение, возвращаемое, если условие ложно.

Примеры:

```
int a = 5;
String result = (a > 0) ? "Положительное число" : "Отрицательное число";
System.out.println(result);
```

```
int x = 3;
int y = 2;
int z = x < y ?  (x + y) : (x - y);
System.out.println(z);
```

Задания

Задание 1: Определение четного или нечетного числа

Напишите программу, которая запрашивает у пользователя целое число и определяет, является ли оно четным или нечетным. Выведите соответствующее сообщение.

Подсказка: Используйте оператор остатка от деления %.

Задание 2: Вводятся 2 вещественных числа. Найти максимальное среди них. (nextDouble())

Задания

Задание 2: Оценка баллов

Создайте программу, которая запрашивает у пользователя оценку (в виде целого числа от 0 до 100) и выводит соответствующее описание:

"Неудовлетворительно" для баллов от 0 до 59

```
if (score >= 0 && score <= 59)
```

"Удовлетворительно" для баллов от 60 до 74

"Хорошо" для баллов от 75 до 89

"Отлично" для баллов от 90 до 100

"Ошибка: вне диапазона" для неверного ввода

while

```
while(Логическое выражение)
{
   // Код
}
```

Цикл с предусловием

Выполняется следующим образом (по шагам):

1. Вычисляем логическое условие (условие входа в цикл), следующее в скобках за while;
2. Если логическое условие истинно, то выполняются операторы в теле цикла, после выполнения последнего оператора в теле цикла, переходим на шаг 1;
3. Если логическое условие ложно, то переходим к первому оператору за пределами цикла while.

while

```
int number = 3;
int result = 1;
int power = 1;
while (power <= 10) {
   result = result * number;
   System.out.println(number + " в степени " + power + " = " + result);
   power++;
}
```

while

Бесконечный цикл

```
while(true) {

}
```

Данные управляющие команды чаще всего находят применение в бесконечном цикле. Его так называют, потому что логическое условие выхода никогда не выполняется.

В этом случае и пригодится применение команды **break** для организации выхода из него. Этот вид цикла имеет место при ожидании внешних условий, которые формируются за пределами логики тела цикла.

while

```
int number = 3;
  int result = 1;
  int power = 1;
  while(true) {
     result = result * number;
     System.out.println(number + " в степени " + power + " = " + result);
     power++;
     if (power>10)
        break;
  }
```

do while

Цикл с постусловием

Выполняется следующим образом (шаги):

1. Выполняется тело цикла (сразу после ключевого слова do);
2. Вычисляем логическое условие, следующее в скобках за while;
3. Если логическое условие истинно, то переходим на шаг 1;
4. Если логическое условие ложно, то переходим к первому оператору за пределами цикла while.

do {

}**while** (Логическое выражение);

do while

```
int number = 3;
  int result = number;
  int power = 1;
  do {
     System.out.println(number + " в степени " + power + " = " + result);
     power++;
     result = result * number;
  }while (result < 10000);
```

while и do while

Собственно, единственное отличие цикла **do-while** от цикла **while** как раз и состоит в том, что *тело цикла* в цикле **do-while** выполняется как минимум один раз.

Цикл **do-while** обычно используют именно тогда, когда нет смысла проверять условие, если тело цикла не выполнилось. Например, в *теле цикла* проходят какие-нибудь вычисления, и их результаты используются в *условии*.

continue

Оператор **continue** используется для пропуска текущей итерации цикла и перехода к следующей итерации. ***continue*** – прекращает выполнение тела текущего цикла и осуществляет переход к логическому выражению оператора **while**. Если вычисленное выражение будет истинно – выполнение цикла будет продолжено.

Он часто применяется в циклах **for**, **while** и **do while** для управления потоком выполнения программы.

```
 Scanner scanner = new Scanner(System.in);
 int number;
System.out.println("Введите числа (введите 0 для выхода):");
 while (true) {
   number = scanner.nextInt();
   if (number == 0) {
     break; // Выход из цикла, если введено 0
   }
   if (number < 0) {
     continue; // Пропускаем отрицательные числа
   }
   System.out.println("Вы ввели: " + number);
 }
 scanner.close();
}
```

break

break – немедленно прекращает выполнение текущего цикла и осуществляет переход к первой команде за его пределами. Таким образом, выполнение текущего цикла прерывается.

for

Цикл **for** обычно используется, когда нужно повторить какую-либо операцию определенное количество раз.

```
for (инициализация; условие; итерация) {

  // тело цикла

}
```

for

Цикл **for** используется для перебора элементов массива по индексам

```
int[] numbers = {1, 2, 3, 4, 5};

// Используем цикл for для перебора массива

for (int i = 0; i < numbers.length; i++) {

  System.out.println("Число: " + numbers[i]);

}
```

for

```
// Используем цикл for, чтобы вывести числа от 1 до 5

for (int i = 1; i <= 5; i++) {

   System.out.println("Число: " + i);

}
```

```
//вывод чисел от 10 до 1 в обратном порядке.

for (int i = 10; i >= 1; i--) {

      System.out.println(i);

}
```

for each

for-each (или "улучшенный цикл for") предназначен для упрощенного перебора элементов коллекций и массивов, не требуя явного указания индексов.

```
int[] numbers = {1, 2, 3, 4, 5};

for (int number : numbers) {

   System.out.println("Число: " + number);

}
```

Задания

Задача 1: Сумма чисел от 1 до 100

Создайте программу, которая вычисляет сумму всех чисел от 1 до 100.

Задача 2: Факториал числа

Напишите программу, которая вычисляет факториал заданного числа.

Задача 3: Четные и нечетные числа

Напишите программу, которая разделяет числа от 1 до 20 на четные и нечетные.

Первоначальная настройка Git

Указать имя пользователя — git config --global user.name "Adelya Enikeeva"
Задаёт имя пользователя, от которого будут идти коммиты. Если имя состоит из одного слова, кавычки можно не ставить.

Указать электронную почту — git config --global user.email adisen24@yandex.ru
Обратите внимание, она должна совпадать с той, на которую зарегистрирован аккаунт в Гитхабе.

Посмотреть настройки — git config --list. Параметры можно посмотреть и в конфигурационном файле, но этот способ быстрее.

Git

git init

Эта команда инициализирует новый локальный репозиторий в текущей директории, где и будет в дальнейшем храниться вся информация об истории коммитов, тегах — о ходе разработки проекта:

Пример использования:

```
mkdir my_project
cd my_project
git init
```

Git

git add

Команда **git add** добавляет изменение из рабочего каталога в раздел проиндексированных файлов.

Пример использования:

```
# Добавляет конкретный файл
git add hello.txt

# Добавляет все изменения в текущей директории
git add .
```

Git

git commit

Команда **git commit** фиксирует изменения в репозитории. Каждый коммит может содержать сообщение, описывающее сделанные изменения.

Пример использования:

```
git commit -m "Добавлена новая функция"
```

Git

git push

Команда **git push** отправляет локальные изменения в удаленный репозиторий (например, на GitHub). При этом необходимо быть авторизованным.

Пример использования:

```
# Связать удалённый и локальный репозитории
git remote add origin https://github.com/yourusername/my_project.git
# Отправляет все изменения с локального репозитория в удалённый
git push -u origin master
```

Git

git log

Команда **git log** показывает историю коммитов.

Пример использования:

```
git log
```

Git

.gitignore

Это файл, который указывает Git, какие файлы или папки следует игнорировать.

Это полезно для исключения временных файлов, файлов конфигурации и других ненужных для отслеживания элементов.

Пример использования:

```
echo "*.class" > .gitignore

git add .gitignore
```

Демонстрация использования

Шаг 1. Инициализация репозитория:

```
mkdir test_project

cd test_project

git init
```

Шаг 2. Создание файла проекта:

```
echo "print(Hello World!) " > hello.java

git add hello.java
```

Демонстрация использования

Шаг 3. Коммит изменений:

```
git commit -m "Добавлен файл hello.txt"
```

Шаг 4. Создание файла .gitignore:

```
echo "*.txt" > .gitignore

git add .gitignore

git commit -m "Добавлен .gitignore для игнорирования файлов .txt»
```

Демонстрация использования

Шаг 5. Связать удалённый и локальный репозитории :

git remote add origin https://github.com/yourusername/hello_project.git

Шаг 6. Отправка изменений в удаленный репозиторий:

git push -u origin master

Шаг 7. Просмотр истории коммитов:

git log