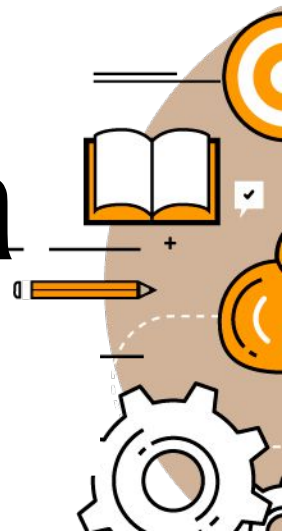




# Основные конструкции языка Java



# Виды скобок в Java и что они делают

- { }

- ( )

- [ ]



```
2
3 public class newApp {
4     int currentDepth=0;
5     public int dive(int howDeep){
6         currentDepth=currentDepth + howDeep;
7         if (currentDepth > 100){
8             System.out.println("Я маленькая рыбка "+
9                 " и не могу плавать глубже 100 метров");
10            currentDepth=currentDepth - howDeep;
11        }else{
12            System.out.println("Погружаюсь ещё на " + howDeep +
13                " метров");
14            System.out.println("Я на глубине " + currentDepth +
15                " метров");
16        }
17        return currentDepth;
18    }
19    public String say(String something){
20        return "Разве вы не знаете, что рыбы не говорят?";
21    }
22 }
23 }
24
```



```
3 public class newApp {
4     int currentDepth=0;
5     public int diveint howDeep(){
6         currentDepth=currentDepth + howDeep;
7         if (currentDepth > 100){
8             System.out.println("Я маленькая рыбка "+
9                 " и не могу плавать глубже 100 метров");
10            currentDepth=currentDepth - howDeep;
11        }else{
12            System.out.println("Погружаюсь ещё на " + howDeep +
13                " метров");
14            System.out.println("Я на глубине " + currentDepth +
15                " метров");
16        }
17        return currentDepth;
18    }
19    public String say(String something){
20        return "Разве вы не знаете, что рыбы не говорят?";
21    }
22 }
23
```

```
2
3 public class newApp {
4     int currentDepth=0;
5     public int dive(int howDeep){
6         currentDepth=currentDepth + howDeep;
7         if (currentDepth <> 100){
8             System.out.println("Я маленькая рыбка "+
9                 " и не могу плавать глубже 100 метров");
10            currentDepth=currentDepth - howDeep;
11        }else{
12            System.out.println("Погружаюсь ещё на " + howDeep +
13                " метров");
14            System.out.println("Я на глубине " + currentDepth +
15                " метров");
16        }
17        return currentDepth;
18    }
19    public String say(String something){
20        return "Разве вы не знаете, что рыбы не говорят?";
21    }
22 }
23
```



```
2
3 public class newApp {
4     int currentDepth=0;
5     public int dive(int howDeep){
6         currentDepth=currentDepth + howDeep;
7         if (currentDepth > 100){
8             System.out.println("Я маленькая рыбка "+
9                 " и не могу плавать глубже 100 метров");
10            currentDepth=currentDepth - howDeep;
11        }else{
12            System.out.println("Погружаюсь ещё на " * howDeep +
13                " метров");
14            System.out.println("Я на глубине " + currentDepth +
15                " метров");
16        }
17        return currentDepth;
18    }
19    public String say(String something){
20        return "Разве вы не знаете, что рыбы не говорят?";
21    }
22 }
23
```



Даны три переменные: A, B, C. Если их значения упорядочены по возрастанию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное (отрицательное). Вывести новые значения переменных A, B, C



# Тернарный оператор

Продвинутые программисты часто используют тернарный оператор `?:` вместо `if-else`. Тернарный оператор использует три операнда и записывается в форме:

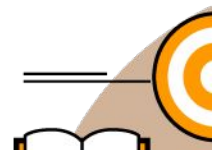
логическое\_условие ? выражение1 : выражение2

Если **логическое\_условие** истинно, т.е. возвращает **true**, то берётся (или вычисляется) первое выражение слева от двоеточия, если возвращается **false**, то берётся второе выражение справа от двоеточия





```
int largerNum;  
int lowNum = 9;  
int highNum = 27;  
  
if(lowNum < highNum) // если первое число меньше второго  
{  
    largerNum = highNum;  
} else { // иначе  
    largerNum = lowNum;  
}
```



```
int lowNum = 9;  
int highNum = 27;  
int largerNum = lowNum < highNum ? highNum : lowNum;
```

Оператор `switch` иногда используется как альтернатива `if`. Значение переменной стоящей после оператора `switch` вычисляется, и программа переходит только к одному из `case` блоков, аргумент которого совпадает с результатом этого вычисления.



```
1 package lec6;
2 import java.util.Scanner;
3
4 public class newApp {
5     public static void main(String[] args) {
6         Scanner in = new Scanner(System.in);
7         System.out.println("Введите оценку");
8         int number = in.nextInt();
9
10        switch (number){
11            case 10:
12                System.out.println("Превосходная работа!");
13                break;
14            case 8:
15                System.out.println("Хорошая работа!");
16                break;
17            case 6:
18                System.out.println("Надо подтянуть знания!");
19                break;
20        }
21    }
22 }
23
```

- Не забудьте написать ключевое слово `break` в конце каждого `case` блока, чтобы после завершения выполнения его кода, произошёл выход из оператора `switch`. Если вы не напишете `break`, то напечатаются все четыре строки, не смотря на то, что значение переменной `yourGrade` имеет только одно значение.



- Попросите программу загадать число от 0 до 100. У вас будет семь попыток на угадывание. При каждой попытке вам будет выводиться сообщение - "Мало" или "Много". Если угадаете, уложившись в семь попыток, то выиграли. Если нет, то идёте кормить кота.
- Для генерации секретного числа

```
4  
5 public class NewApp {  
6     public static void main(String[] args) {  
7         Random random = new Random();  
8         int secret;  
9         // Генерируем число от 0 до 100  
10        secret = random.nextInt(100 + 1);  
11    }
```



Существует ещё одно ключевое слово для создания циклов -- `while`. В таких циклах не нужно точно знать, сколько раз будет повторяться действие, но необходимо задать условие окончания цикла. Давайте посмотрим, как можно поздравить участников игры с помощью цикла `while`, он закончится, когда `counter` (счётчик) станет равным `totalPlayers`.

```
int totalPlayers = players.length;
int counter=0;

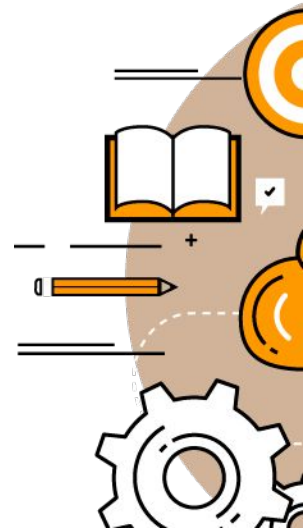
while (counter< totalPlayers){

    String thePlayer = players[counter];

    System.out.println("Поздравляем, "+ thePlayer + "!");

    counter++;

}
```



В следующем примере, после вывода элементов массива 0, 1 и 2, выполнение программы выйдет из цикла из-за инструкции `break` и продолжится со строки после закрывающей фигурной скобки.

Обратите внимание на двойной знак равенства в операторе `if`. Значение переменной `counter` *сравнивается* с числом 3. Одинарный знак равно означал бы, что значение 3 *присваивается* переменной `counter`. Если перепутать `==` с `=` в операторе `if`, то программа будет работать, но неправильно и такую ошибку иногда очень не легко найти:

```
int counter =0;
while (counter< totalPlayers){

    if (counter == 3){

        break; // Выпрыгиваем из цикла

    }

    String thePlayer = players[counter];
    System.out.println("Congratulations, "+thePlayer+ "!");
    counter++;

}
```



Оператор `continue` позволяет пропустить выполнение строк, стоящих после него и вернуться к началу цикла. Представим, что мы хотим поздравить всех, кроме Дэвида - `continue` вернёт выполнение программы к началу цикла.

```
while (counter < totalPlayers) {  
  
    counter++;  
    String thePlayer = players[counter];  
  
    if (thePlayer.equals("David")) {  
        continue;  
    }  
  
    System.out.println("Congratulations, "+ thePlayer+ "!");  
}
```

Есть ещё один тип оператора `while`, который начинается со слова `do`, например:

```
do {  
    // Здесь ваш код, что-нибудь считающий  
} while (counter < totalPlayers);
```

В таких циклах условие проверяется *после* выполнения кода, стоящего между фигурными скобками. Таким образом, этот код выполнится *хотя бы один раз*. Циклы, в которых `while` стоит вначале, могут не выполниться ни разу, если условие ложно (`false`).





# Домашняя работа

- *Дан массив размера  $N$ . Вывести его элементы в обратном порядке.*

