Правила именования

Правила именования переменных, методов, классов

Желательно создавать составные осмысленные имена. В этом случае в одно слово можно втиснуть предложение, которое в доступной форме представит информацию о типе объекта, его назначении и особенностях использования.

При выборе имени желательно:

- выбирать конкретные слова, которые характеризуют объект;
- имена переменных и классов должны быть существительными;
- имена методов – глаголами;
- использовать суффиксы и префиксы для добавления дополнительной информации к имени;
- использовать форматирование имени.

Соглашения об именовании

1. Имена классов должны быть обязательно написаны в смешанном регистре, начиная с верхнего: Line, MyInterface.
2. Имена переменных и полей должны быть записаны в смешанном регистре, начиная с нижнего: line, borderColor.
3. Именованные константы (включая значения перечислений) должны быть записаны в верхнем регистре с нижним подчёркиванием в качестве разделителя: MAX_ITERATIONS, COLOR_RED, PI
4. Названия методов и функций должны быть глаголами и записанными в смешанном регистре и начинаться с нижнего: getName()
5. Все имена следует записывать по-английски: fileName, но не: imyaFayla
6. Слово compute может быть использовано в методах, вычисляющих что-либо: computeAverage()
7. Слово find может быть использовано в методах, осуществляющих какой-либо поиск: findMinElement()
8. Переменным, представляющим GUI, следует давать суффикс, соответствующий имени типа компонента.
9. Префикс is следует использовать только для булевых (логических) методов: isSet, isVisible, isFinished, isFound, isOpen

Имена методов

Выбрав хорошее имя, вы сообщаете гораздо больше информации, чем кажется на первый взгляд. По сути, правильное имя уже содержит комментарий. Ищите яркие запоминающие имена-синонимы.

Примеры синонимов:

- send (посылать) – deliver (доставлять), announce (извещать), route (направлять)
- find (искать) – search (искать), locate (обнаруживать)
- start (начинать) – launch (запускать), create (создавать), begin (начинать)
- make (создавать) – create (создавать), setup (устанавливать), build (строить), generate (генерировать)

Использование общих имен переменных

1. Переменные цикла: i, j, k, l, m, n.
2. Счетчики: n, count
3. Сумма: s
4. Корни уравнения: x1, x2, x3 и т.п.
5. Временная переменная tmp:

```
if (mas[i + 1] < mas[ i ]) {
    tmp = mas[i + 1];
    mas[i + 1] = mas[ i ];
    mas[ i ] = tmp;
}
```

Отступы

Отступы применяются для демонстрации логической структуры программы. Основной отступ на уровень – 8 пробелов, но чаще всего используется 4 пробела. Пример правильно форматированного кода:

```
// Программа вычисления суммы и факториала чисел от 1 до 5.
public static void main() {
    int sum = 0;
    int fact = 1;
    for(int i = 1; i <= 5; i++) {
        sum += i;
        fact *= i;
    }
    System.out.println("Сумма равна " + sum);
    System.out.println("Факториал равен " + fact);
}
```

Длинные строки и их разбиение

Строки программного кода не должны быть длинными, так как:

- их тяжело читать;
- препятствуют созданию вложенности.

При разбиении сложных выражений желательно первую строку заканчивать арифметическим или логическим оператором. Тогда не будет казаться, что просто забыли поставить скобку или точку с запятой:

```
customerBill = PreviousBalance(paymentHistory[customerID]) +
    LateCharge(paymentHistory[customerID]);
```

Использование пробелов

- Операторы следует отбивать пробелами с двух сторон.
- После зарезервированных ключевых слов следует ставить пробел.
- После запятых следует ставить пробелы. После точек с запятой в цикле for следует ставить пробелы.

Примеры:

- в объявлении методов после запятой, но не перед скобками:

```
testMethod(a, b, c);
```

Примеры нерекомендуемого использования:

```
testMethod(a,b,c);
testMethod( a, b, c );
```

- для выделения операторов:

```
a = b;
```

нерекомендуемое использование:

```
a=b;
```

- при форматировании циклов:

```
for (int i = 0; i < 10; ++i) ;
```

нерекомендуемое использование:

```
for (int i=0; i<10; ++i)
for(int i=0;i<10;++i)
```

Пустые строки

Пустые строки помогают разбивать код приложения на логические сегменты, что позволяет продемонстрировать организацию программы.

Несколькими строками могут отделяться классы и интерфейсы;

Одной пустой строкой отделяются друг от друга:

- методы;
- описание локальных переменных от первых операторов;
- блок операторов внутри метода для более удобного чтения;

Они используются для выделения комментариев.

Исследование показало, что оптимальное число пустых строк в программе составляет от 8% до 16%.

Длина метода и размер класса

Рекомендуемые длины:

- методов: от 20 до 50 строк;
- классов: не более 300 – 500 строк
- строк: не более 60 – 80 символов;
- количество аргументов не более 5 – 7;

Выделяйте сложный код в отдельный метод.