**Аргументы функций**

В идеальном случае количество аргументов функции равно нулю (нуль-арная функция). Далее следуют функции с одним аргументом (унарные) и с двумя аргументами (бинарные). Функций с тремя аргументами (тернарных) следует по возможности избегать. Необходимость функций с большим количеством аргументов (полиарных) должна быть подкреплена очень вескими доводами — и все равно такие функции лучше не использовать.

Аргументы усложняют функции и лишают их значительной части концептуальной мощи. Именно по этой причине я почти полностью избавился от них в этом примере. Возьмем хотя бы переменную StringBuffer. Ее можно было бы передать в аргументе (вместо того, чтобы делать ее переменной экземпляра), но тогда читателям кода пришлось бы интерпретировать ее каждый раз, когда она встречается в коде.

Аргументы создают еще больше проблем с точки зрения тестирования. Только представьте, как трудно составить все тестовые сценарии, проверяющие правильность работы кода со всеми комбинациями аргументов. Если аргументов нет — задача тривиальна. При одном аргументе все обходится без особых сложностей. С двумя аргументами ситуация усложняется. Если же аргументов больше двух, задача тестирования всех возможных комбинаций выглядит все более устрашающе. Выходные аргументы запутывают ситуацию еще быстрее, чем входные. Читая код функции, мы обычно предполагаем, что функция получает информацию в аргументах, и выдает ее в возвращаемом значении. Как правило, никто не ожидает, что функция будет возвращать информацию в аргументах. Таким образом, выходные аргументы часто заставляют нас браться за чтение функции заново.

**Стандартные унарные формы**

Существует два очень распространенных случая вызова функции с одним аргументом. Первая — проверка некоторого условия, связанного с аргументом, как в вызове boolean fileExists("MyFile"). Вторая — обработка аргумента, его преобразование и возвращение. Например, вызов InputStream fileOpen("MyFile") преобразует имя файла в формате String в возвращаемое значение InputStream.

Несколько менее распространенным, но все равно очень полезным частным случаем функции с одним аргументом является ***событие***. В этой форме имеется входной аргумент, а выходного аргумента нет. Предполагается, что программа интерпретирует вызов функции как событие и использует аргумент для изменения состояния системы, например,

void passwordAttemptFailedNtimes(int attempts).

Старайтесь избегать унарных функций, не относящихся к этим формам.

Преобразования, в которых вместо возвращаемого значения используется выходной аргумент, сбивают читателя с толку. Если функция преобразует свой входной аргумент, то результат должен передаваться в возвращаемом значении.

**Аргументы-флаги**

Аргументы-флаги уродливы. Передача логического значения функции — воистину ужасная привычка. Она немедленно усложняет сигнатуру метода, громко провозглашая, что функция выполняет более одной операции. При истинном значении флага выполняется одна операция, а при ложном — другая!

**Бинарные функции**

Функцию с двумя аргументами понять сложнее, чем унарную функцию. Например, вызов writeField(name) выглядит более доступно, чем writeField(outputStream, name)1 . Хотя смысл обеих форм понятен, первая форма просто проскальзывает под нашим взглядом, моментально раскрывая свой смысл. Во второй форме приходится сделать непродолжительную паузу, пока вы не поймете, что первый параметр должен игнорироваться. И конечно, это в конечном итоге создает проблемы, потому что никакие части кода игнорироваться не должны. Именно в проигнорированных частях чаще всего скрываются ошибки. Бинарные функции не являются абсолютным злом, и вам почти наверняка придется писать их. Тем не менее следует помнить, что за их использование приходится расплачиваться, а вам стоит воспользоваться всеми доступными средствами для их преобразования в унарные. Например, можно сделать метод writeField членом класса outStream, чтобы использовать запись outputStream. writeField(name). Другой вариант — преобразование outputStream в поле текущего класса, чтобы переменную не приходилось передавать при вызове. Также можно создать новый класс FieldWriter, который получает outputStream в конструкторе и содержит метод write.

**Тернарные функции**

Разобраться в функции с тремя аргументами значительно сложнее, чем в бинарной функции. Проблемы соблюдения порядка аргументов, приостановки чтения и игнорирования увеличиваются более чем вдвое. Я рекомендую очень хорошо подумать, прежде чем создавать тернарную функцию. Для примера возьмем стандартную перегруженную версию assertEquals с тремя аргументами: assertEquals(message, expected, actual). Сколько раз вы читали значение message и думали, что перед вами expected? Я сталкивался с этой конкретной тернарной функцией и задерживался на ней много раз. Более того, ***каждый раз, когда я ее вижу, мне приходится делать новый заход и вспоминать о необходимости игнорировать message.***

***Объекты как аргументы***

Если функция должна получать более двух или трех аргументов, весьма вероятно, что некоторые из этих аргументов стоит упаковать в отдельном классе. Рассмотрим следующие два объявления: Circle makeCircle(double x, double y, double radius); Circle makeCircle(Point center, double radius); Сокращение количества аргументов посредством создания объектов может показаться жульничеством, но это не так. Если переменные передаются совместно как единое целое (как переменные x и y в этом примере), то, скорее всего, вместе они образуют концепцию, заслуживающую собственного имени.

**Выходные аргументы**

Аргументы естественным образом интерпретируются как входные данные функции. Каждый, кто занимался программированием более нескольких лет, наверняка сталкивался с необходимостью дополнительной проверки аргументов, которые на самом деле оказывались выходными, а не входными. Пример: appendFooter(s); Присоединяет ли эта функция s в качестве завершающего блока к чему-то другому? Или она присоединяет какой-то завершающий блок к s? Является ли s входным или выходным аргументом? Конечно, можно посмотреть на сигнатуру функции и получить ответ: public void appendFooter(StringBuffer report) Вопрос снимается, но только после проверки объявления. Все, что заставляет обращаться к сигнатуре функции, нарушает естественный ритм чтения кода. Подобных «повторных заходов» следует избегать.