0 Предварительные сведения

После выполнения каждого задания необходимо производить коммит изменений в репозиторий (если было, что коммитить). Нерабочий код коммитить крайне нежелательно. Некомпилируемый – тем более.

Все файлы классов должны лежать внутри созданного ранее пакета проекта или его подпакетов (и подпакетов подпакетов и т.д.).

Можно выполнять только те задания, которые нравятся.

Порядок выполнения заданий – произвольный, однако некоторые задания зависят от других.

Каждый участок программы необходимо тестировать на работоспособность.

1 Типы данных

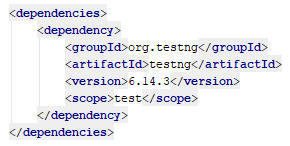
* 1. Создать класс Person (Человек) с полями firstName (имя) и lastName (фамилия) типа String, а также поле passportId (номер паспорта) типа int. Поля должны быть приватными, для доступа к ним необходимо добавить геттеры и сеттеры. Создать несколько объектов данного класса и заполнить все их поля.
  2. Создать 4 конструктора для класса Person – без параметров, с параметрами имени и фамилии, с параметром паспорта, а также со всеми тремя параметрами. Создать несколько объектов, задействовав все конструкторы.
  3. Создать класс Point (точка) с тремя публичными неизменяемыми (final) координатами-полями типа double: x, y и z. Создать единственный конструктор, который требует задания всех трёх координат. Создать несколько точек.
  4. Создать класс Points (точки) для работы с объектами класса Point. Все методы этого класса должны быть статическими. Внутри созданного класса реализовать четыре покоординатных операции с точками с помощью публичных статических методов sum(), subtract(), multiply(), divide() – сложение, вычитание, умножение и деление. Методы должны принимать на вход две точки и новую точку, полученную в результате операции. Проверить работоспособность методов. Добавить классу приватный конструктор без параметров с пустым телом, чтобы ни у кого не возникло желание создать экземпляр этого класса.
  5. Добавить классу Points метод enlarge(), принимающий на вход точку Point и число с плавающей точкой double и возвращающий новую точку, полученную произведением координат входной точки на входное число.
  6. Добавить классу Point метод length(), возвращающей длину радиус-вектора, соответствующего точке, у которой этом метод вызывается. Для нахождения квадратного корня можно использовать статический метод sqrt() класса java.lang.Math. Добавить классу Points статический метод length(), принимающий на вход точку Point, и возвращающий значение длины её радиуса-вектора с помощью делегирования полномочий методу length() входной точки (т.е. если на входе Point point, то нужно вызвать у неё написанный ранее метод point.length()).
  7. Добавить классу Points статические методы opposite() и inverse(), принимающие на вход точку и возвращающие новую точку. В первом случае её координаты должны иметь противоположный знак. Во втором – новые координаты должны быть получены из старых путём деления единицы на них.
  8. Добавить классу Points методы scalarProduct() и vectorProduct(), вычисляющие, соответственно, скалярное и векторное произведения векторов.
  9. Создать некоторый класс с одним полем и одним методом. Проверить все возможные варианты попарной сочетаемости модификаторов класса, метода и поля. Заполнить таблицы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Модификаторы объявления класса | | | | | | |
|  | public | protected | private | final | abstract | strictfp |
| public |  |  |  |  |  |  |
| protected |  |  |  |  |  |  |
| private |  |  |  |  |  |  |
| final |  |  |  |  |  |  |
| abstract |  |  |  |  |  |  |
| strictfp |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Модификаторы полей класса | | | | | | | |
|  | public | protected | private | static | final | transient | volatile |
| public |  |  |  |  |  |  |  |
| protected |  |  |  |  |  |  |  |
| private |  |  |  |  |  |  |  |
| static |  |  |  |  |  |  |  |
| final |  |  |  |  |  |  |  |
| transient |  |  |  |  |  |  |  |
| volatile |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Модификаторы методов класса | | | | | | | |  |  |
|  | public | protected | private | static | final | abstract | strictfp | synchronized | native |
| public |  |  |  |  |  |  |  |  |  |
| protected |  |  |  |  |  |  |  |  |  |
| private |  |  |  |  |  |  |  |  |  |
| static |  |  |  |  |  |  |  |  |  |
| final |  |  |  |  |  |  |  |  |  |
| abstract |  |  |  |  |  |  |  |  |  |
| strictfp |  |  |  |  |  |  |  |  |  |
| synchronized |  |  |  |  |  |  |  |  |  |
| native |  |  |  |  |  |  |  |  |  |

* 1. Добавить в файл pom.xml импорт библиотеки автоматического тестирования (например, testNG) между тегами <dependencies></dependencies>. Если там пусто, то блок будет выглядеть так:



Так выглядит импорт библиотеки testNG версии 6.14.3 в maven. В дальнейшем в блок dependencies можно добавлять и другие зависимости от сторонних библиотек. Выполнить реимпорт проекта. Перейти в класс Person, нажать Alt + Insert, выбрать Test. В появившемся окне выбрать библиотеку testNG и нажать OK. Будет создан файл теста класса в тестовой директории, но в том же пакете. Создать в этом классе публичный ничего не возвращающий метод тестирования геттеров и сеттеров: внутри метода создать несколько экземпляров класса Person, заполнить их с помощью методов set… и затем проверить, что они возвращают то же самое методами get… Для проверки необходимо использовать статический метод assertEquals класса org.testng.Assert. Первый параметр – фактическое значение (в данном случае вызов геттера), второй параметр – ожидаемое значение. Перед методом необходимо добавить аннотацию @Test из пакета org.testng.annotations.\*. Запустив тест, необходимо убедиться, что он прошёл успешно. Ради эксперимента можно поменять ожидаемое значение на неправильное, чтобы увидеть, что будет. Тест можно запустить с покрытием кода (with Coverage) – тогда можно будет проанализировать, какие классы, методы и строки были задействованы, а какие – нет. Тесты являются таким же кодом, как и код остальной программы, поэтому их тоже нужно коммитить.

* 1. Вместо метода, тестирующего геттеры и сеттеры класса Person, написать три отдельных тестовых метода для тестирования соответствующих методов каждого поля. Обязательно проверить работоспособность с передачей null вместо имени и фамилии, а также с вызовом геттера до задания значений полей. Покрыть тестами конструкторы.
  2. Покрыть тестами все методы класса Points (в новом классе-тесте). Для этого написать вспомогательный приватный метод этого класса equalsApproximately(), который будет сравнивать типы double с заданной точностью (модуль можно вычислить с помощью метода Math.abs()) и возвращать тип boolean. Заданную точность необходимо выделить в отдельное статическое публичное поле тестового класса и задать каким-либо приемлемым значением (например, 0.00005). Написать ещё один вспомогательный метод equalsApproximately, сравнивающий между собой уже два объекта Point – если их координаты равны между собой с заданной точностью (для сравнения координат необходимо использовать ранее написанный метод), то и точки считаются приблизительно равными. После написания этих двух методов для тестирования необходимо будет вызывать методы assertTrue() и assertFalse() класса Assert, в качестве аргументов передавать результат вызова приблизительного сравнения полученного значения и ожидаемого. Например:

Assert.*assertTrue*(equalsApproximately(1, 1.00000000000001));  
Assert.*assertFalse*(equalsApproximately(1, 0.5));

Assert.*assertFalse*(equalsApproximately(scalarProduct(x, y), 1));

Стоит отметить, что в классе Assert есть метод assertEquals(…) для типов float и double, в которых третьим параметром можно указать точность сравнения. Можно использовать и его.

* 1. Создать класс и дописать в него следующие методы:

**public static void** main(String[] args) {  
 Person person = **new** Person();  
 person.setName(**"Arkadiy"**);  
 **int** number = 5;  
 *checkInt*(number);  
 *checkPerson*(person);  
 System.***out***.println(number);  
 System.***out***.println(person.getName());  
}  
  
**private static void** checkInt(**int** number) {  
 number = 10;  
 System.***out***.println(number);  
}  
  
**private static void** checkPerson(Person person) {  
 person.setName(**"Oleg"**);  
 System.***out***.println(person.getName());  
}

Запустить. Объяснить вывод, полученный в консоли.

* 1. Изменить методы предыдущего класса следующим образом:

**public static void** main(String[] args) {  
 Person person = **new** Person();  
 person.setName(**"Arkadiy"**);  
 *checkAnotherPerson*(person);  
 System.***out***.println(person.getName());  
}  
  
**private static void** checkAnotherPerson(Person person) {  
 person = **new** Person();  
 person.setName(**"Ignat"**);  
 System.***out***.println(person.getName());  
}

Запустить. Объяснить вывод, полученный в консоли.

* 1. Создать абстрактный класс Operation, имеющий абстрактный метод apply(double number), возвращающий double, и неабстрактный метод applyTriple(double number), реализация которого должна заключаться в применении метода apply() к входному числу три раза с возвращением полученного числа. Создать трёх неабстрактных наследников класса Operation: IdenticalOperation, SqrtOperation, TangentOperation. Реализация метода apply() в первом случае должна сводиться к возвращению переданного аргумента; во втором случае – возвращение корня входного аргумента с помощью метода Math.sqrt(); в третьем случае – возвращение тангенса входного аргумента с помощью метода Math.tan(). Создать три соответствующих тестирующих класса и покрыть методs apply() и applyTriple() автотестами. Среди тестов должны рассматриваться такие значения, как положительная и отрицательная бесконечности и NaN.
  2. Создать класс NamedPoint, являющийся наследником класса Point с полем String name. Поле не должно быть завершённым, доступ к нему должен осуществляться только через геттер и сеттер. В классе должно быть реализовано два конструктора: с тремя и четырьмя параметрами. Первый принимает только координаты точки, второй – координаты точки и её название. Добавить конструктор без параметров, вызывающий конструктор с четырьмя параметрами и передающий в него три нуля и строку «Origin». Покрыть класс тестами.
  3. Создать перечисление (enum) Gender, которое может принимать только два значения: MALE и FEMALE. Добавить поле gender типа Gender в класс Person, а также сеттер и геттер для него. По желанию добавить новую информацию в имеющиеся конструкторы или добавить новые конструкторы. Покрыть новые участки кода тестами (класс тестирования для Person уже был создан ранее).
  4. Создать интерфейс IntGenerator с методом nextInt() без аргументов и возвращающий значение int. Создать интерфейс Resettable с методом reset() без аргументов и ничего не возвращающий. Создать класс IntGeneratorImpl, реализующий интерфейс IntGenerator следующим образом: если у объекта данного класса метод вызывается впервые, он должен вернуть 0. Каждый следующий вызов должен возвращать число на 1 больше, чем предыдущий. Сделать класс NamedPoint реализующим интерфейс Resettable: метод reset() должен задавать полю name значение «Absent». Создать класс ResettableIntGenerator, реализующий оба вышеописанных интерфейса: при этом метод nextInt() должен быть реализован так, как и в классе IntGeneratorImpl, а метод reset() должен сбрасывать состояние объекта: после его вызова метод nextInt() должен работать так, как будто его раньше никогда не вызывали, т.е. вновь возвращать значения 0, 1, 2… Покрыть все методы тестами. Ссылки на объекты генератора должны быть типа интерфейса, а не класса.
  5. Создать класс, в котором создать девять перегруженных методов printType(…), каждый из которых имеет один аргумент: byte, char, short, int, long, float, double, boolean и Object. Задача каждого метода – вывести в консоль соответственно «byte», «char», «short», «int», «long», «float», «double», «boolean» - в случае примитивных типов. В последнем методе необходимо провести проверку на null: если аргумент null, то вывести «null», иначе – вывести возвращаемое значение метода getClass() у объекта-аргумента. Добиться того, чтобы были вызваны все методы: в некоторых случаях придётся прибегнуть к явному приведению типов. В последний метод передавать объекты различных классов, например, String, Integer, Person и т.п.
  6. Изучить классы-обёртки (кроме Void). Написать класс, в методах которого произвести различную упаковку/распаковку типов, автоупаковку и автораспаковку. Изучить, как взаимодействуют между собой и друг с другом классы обёртки и примитивы. Обязательно изучить оператор сравнения == значений примитивных типов и обёрток. Посмотреть, что будет, если ссылка обёртки – null. Создать переменную condition типа Boolean, после неё добавить блок кода с ветвлением if (condition) {…}. Проверить, что произойдёт, если переменная condition примет значение true, false или null. Заменить условие ветвление на Boolean.TRUE.equals(condition). Ещё раз проверить, что будет, если переменная condition примет значение true, false или null. Покрыть весь код тестами или выводами в консоль.

2 Массивы

Для набора задач с массивами рекомендуется создать отдельный классы (или несколько классов). Когда требуется обеспечить покрытие тестами, не нужно рассматривать случаи с отрицательной размерностью или выходом за границы массива. Необходимо пользоваться конструкциями циклов.

* 1. Создать метод, принимающий на вход число – размерность массива. Необходимо вернуть новый массив чисел заданной размерности. Покрыть метод тестами
  2. Создать метод, генерирующий массив чисел по входной размерности. Массив должен быть заполнен единицами, кроме первого и последнего элемента – на их месте должны быть двойки. Метод должен возвращаться созданный массив. Покрыть метод тестами.
  3. Создать метод, генерирующий массив чисел по входной размерности. Заполнить массив последовательностью возрастающих нечётных чисел 1, 3, 5, … и вернуть его в качестве результата. Покрыть метод тестами.
  4. Создать метод, генерирующий массив чисел по входной размерности. Заполнить массив последовательностью убывающих чётных чисел, так, чтобы последним элементом была 2 (например, 10, 8, 6, 4, 2). Вернуть массив в качестве результата и покрыть метод тестами.
  5. Создать метод, возвращающий массив из первых *n* чисел Фибоначчи. Покрыть метод тестами.
  6. Создать метод, возвращающий массив входной размерности, каждый элемент которого равен квадрату своего индекса. Покрыть метод тестами.
  7. Создать метод, принимающий на вход *a*, *b* и *c* – дробные числа, соответствующие коэффициентам квадратного уравнения. Метод должен возвращать решение в виде массива: если решений нет, то массив должен быть пуст (0 элементов). Если решение одно, то массив должен иметь размерность 1 и в нём должно содержаться это единственное решение. Если решений два, то массив должен иметь размерность 2 и содержать в себе оба решения в порядке возрастания. Покрыть метод тестами (с заданной точностью). Не забывать, что *a* может быть равно 0.
  8. Создать метод, возвращающий массив входной размерности, содержащий в себе натуральные числа, начиная с 1, за исключением тех, что делятся на 3. Покрыть метод тестами.
  9. Создать метод, возвращающий массив входной размерности, представляющий собой арифметическую прогрессию, где первый элемент и разность являются также входными параметрами. Покрыть метод тестами.
  10. Создать метод, возвращающий массив входной размерности, представляющий собой геометрическую прогрессию, где первый элемент и знаменатель являются также выходными параметрами. Покрыть метод тестами с заданной точностью.
  11. Создать метод, возвращающий массив, содержащий все делители целого числа, являющегося входным аргументом метода. Покрыть метод тестами. Примечание: делитель числа не может быть больше, чем корень из этого числа.
  12. Создать метод, возвращающий массив, содержащий все положительные простые числа до входного числа включительно. Покрыть метод тестами: Примечание: число 1 не является ни простым, ни составным.
  13. Создать метод, возвращающий массив входной размерности, содержащий идущие друг за другом в порядке возрастания натуральные числа до середины массива. После середины массив должен быть заполнен так, чтобы выглядел симметрично. Например, для входного числа 7 результат будет таким: 1, 2, 3, 4, 3, 2, 1. Покрыть метод тестами.
  14. Создать метод, не возвращающий ничего, принимающий на вход массив чисел и меняющий в каждом элементе этого массива знак на противоположный. Покрыть метод тестами.
  15. Создать метод, принимающий на вход массив чисел и одно число. Метод должен вернуть значение типа boolean, содержится ли данное число в этом массиве. Покрыть метод тестами.
  16. Создать метод, принимающий на вход массив Integer[] и возвращающий значение типа boolean, содержится ли null среди элементов массива. Покрыть метод тестами.
  17. Создать метод, принимающий на вход массив чисел и возвращающий количество чётных чисел в массиве. Покрыть метод тестами.
  18. Создать метод, принимающий на вход массив целых чисел и возвращающий максимальный элемент в массиве. Если массив не содержит элементов, то вернуть null. Покрыть метод тестами.
  19. Создать метод, принимающий на вход массив чисел и возвращающий сумму всех чисел с чётными индексами. Покрыть метод тестами.
  20. Создать метод, принимающий на вход массив целых чисел и возвращающий значение типа boolean. Значение должно равняться true, если количество элементов, делящихся на первый элемент массива, больше количества элементов, делящихся на последний элемент массива. В остальных случаях метод должен вернуть false. Покрыть метод тестами.
  21. Создать метод, принимающий на вход массив чисел и возвращающий элемент, который встречается в массиве чаще всего. Если таких элементов несколько, возвращать тот, который раньше встречается. Покрыть метод тестами.
  22. Создать метод, принимающий на вход массив чисел и ещё одно число. Метод должен возвращать индекс первого элемента, равного входному числу. Покрыть метод тестами.
  23. Создать метод, принимающий на вход массив чисел и не возвращающий ничего. Метод должен поменять первый максимальный и первый минимальный элементы массива. Покрыть метод тестами.
  24. Создать метод, принимающий на вход массив целых чисел и не возвращающий ничего. Метод должен применить к каждому числу операцию побитового отрицания (и перезаписать его). Покрыть метод тестами, при этом необходимо в том числе рассмотреть случай применения метода дважды к одному массиву.
  25. Создать метод, принимающий на вход массив целых чисел и возвращающий массив целых чисел. Новый массив должен быть составлен путём применения к старому массиву операции побитового отрицания. Старый массив не должен быть изменён. Покрыть метод тестами, в том числе рассмотреть случай двукратного применения метода.
  26. Создать метод, принимающий на вход массив целых чисел и возвращающий массив целых чисел. У входного массива необходимо просуммировать попарно элементы (1 и 2, 3 и 4, 5 и 6 и т.д.) и записать в возвращаемый массив результаты. Для этого надо предварительно вычислить необходимую длину нового массива. Если входной массив имеет нечётную длину, считается, что он складывается с нулём. Покрыть метод тестами.
  27. Создать метод, принимающий на вход массив целых чисел и возвращающий массив boolean такой же длины. В массиве должны быть значения true на тех местах, где во входном массиве число чётное, и false в противном случае. Покрыть метод тестами.
  28. Создать метод, принимающий на вход число типа long. Метод должен разбить число на 2 числа типа int: первое содержит битовые значения старших 32 разрядов входного числа, второе – младших 32 разрядов. Метод должен вернуть массив этих двух чисел. Создать метод, принимающих на вход два числа типа int и возвращающих число типа long с помощью битового «склеивания» двух чисел. Покрыть методы тестами.
  29. Создать метод, принимающий на вход размерность массива и индекс начала заполнения. Метод должен создать массив и заполнить его натуральными числами, идущими друг за другом, т.е. 1, 2, 3, 4, … Но метод должен начать заполнять с индекса, который задаётся вторым аргументом метода. Если массив заканчивается, то заполнение должно продолжиться до начального индекса исключительно. Например, для параметров «5, 3» заполнение должно быть таким: 3, 4, 5, 1, 2. Метод нужно реализовать с использованием операции остатка от деления. Покрыть метод тестами.
  30. Создать метод, принимающий на вход число. Метод должен создавать двумерный массив чисел, первая размерность которого – входное число. Каждый элемент-массив двумерного массива должен быть массивом, размерность которого на 1 ниже размерности предыдущего элемента. Исходная размерность равна входному числу. Весь массив должен быть заполнен неповторяющимися идущими друг за другом натуральными числами. Например, если входное число 3 – то результат должен быть таким: {{1, 2, 3}, {4, 5}, {6}}. Покрыть метод тестами.
  31. Создать класс Matrix, инкапсулирующий в себе прямоугольный двумерный массив дробных чисел. В классе должны быть методы доступа к элементу (getAt(n, m), setAt(n, m)), конструктор с неизменяемыми входным размерностями и геттеры этих размерностей. Создать класс Matrices, в котором должны быть реализованы статические методы для работы с матрицами, возвращающие новую матрицу, не меняя содержимое старых: сложение матриц, умножение матриц и умножение матрицы на число. В случае, если матрицы имеют несовместимые размерности, возвращать null. При желании можно написать вычисление определителя матрицы. Покрыть классы тестами.
  32. Создать метод, принимающий на вход массив чисел и не возвращающий ничего. Метод должен проверить, содержится ли в массиве NaN: если нет, то отсортировать массив с помощью делегирования поведения методу java.util.Arrays.sort(…). В противном случае ничего не делать. Покрыть метод тестами.
  33. Создать метод, принимающий на вход массив строк. Метод должен использовать цикл for-each, чтобы обойти весь массив и вывести в консоль все значения строк.
  34. Создать метод, принимающий на вход массив чисел. Метод должен найти произведение всех чисел, не равных нулю, NaN и бесконечностям и вернуть его. При реализации необходимо использовать цикл for-each.
  35. Создать метод, принимающий на вход массив целых чисел. Метод должен для каждого элемента массива вывести его значение в шестнадцатеричном виде с помощью вызова метода toHexString() соответствующего класса-обёртки. Цикл в методе должен быть реализован с помощью использования конструкции языка Arrays.stream(входной массив).forEach(лямбда-выражение).

3 Строки

Так же, как и для работы с массивами, для большинства этих задач следует создать отдельный класс или классы.

* 1. Создать метод, принимающий на вход строку String, и выводящий в консоль каждый символ входной строки на отдельной строке консоли. Реализовать метод с помощью цикла for, а также с использованием методов строк length() и charAt(). Нумерация символов, как и в массиве, с нуля.
  2. Создать метод, принимающий на вход строку String. У строки необходимо получить массив байт с помощью метода getBytes(), затем требуется пробежаться по всему массиву с помощью цикла for-each и вывести каждый байт с новой строки. Проверить, как отличаются выводы, если строка содержит латинские и/или русские символы.
  3. Создать метод, не принимающий на вход ничего, и не возвращающий ничего. Создать в методе произвольную строку с помощью строкового литерала. Создать вторую строку с помощью вызова конструктора класса String() с одним параметром путём передачи в конструктор первую строку. Метод должен вывести в консоль значения сравнений двух строк сначала с помощью оператора ==, а затем – с помощью метода equals(), вызванной у одной из строк. Запустить и сделать выводы.
  4. Создать метод, принимающий на вход строку. Метод должен вернуть значение типа boolean, является ли строка палиндромом. Покрыть метод тестами.
  5. Создать метод, принимающий на вход две строки. Метод должен сравнить содержание строк и вернуть true только в том случае, если строки различаются и, более того, различаются только регистром. В противном случае вернуть false. Использовать методы equals() и equalsIgnoreCase(). Подразумевается, что вместо любой из строк (или обеих сразу) может быть null. Покрыть метод тестами, включая передачу null.
  6. Создать метод для изучения экранирования символов. Тело метода:

**int** i = 0;  
System.***out***.println(**"Символ\t№"** + ++i);  
System.***out***.println(**"Символ\b№"** + ++i);  
System.***out***.println(**"Символ\n№"** + ++i);  
System.***out***.println(**"Символ\r№"** + ++i);  
System.***out***.println(**"Символ\'№"** + ++i);  
System.***out***.println(**"Символ\"№"** + ++i);  
System.***out***.println(**"Символ\\№"** + ++i);

Запустить метод и сделать вывод о том, для чего нужен символ экранирования \. Удалить символ экранирования (в последней строке оставить один обратный слеш) и посмотреть, что произойдёт.

* 1. Создать метод, принимающий на вход две строки. Метод должен вернуть индекс символа первой строки, с которого начинается первое вхождение второй строки в первую. Например, для аргументов «abcabcabc» и «ca» результат будет 2. Если такого вхождения нет, вернуть -1. Использовать один из методов indexOf(). Покрыть метод тестами.
  2. Создать метод, принимающий на вход две строки и возвращающий индекс первого вхождения второй строки во вторую половину первой строки. Если такого вхождения нет, вернуть -1. Использовать один из методов indexOf(), который начинает вести поиск с заданной символа. Покрыть метод тестами.
  3. Создать метод, принимающий на вход две строки и возвращающий индекс последнего вхождения второй строки в первую половину первой строки. Если такого вхождения нет, вернуть -1. Использовать один из методов lastIndexOf(), который начинает вести поиск с заданной символа в обратную сторону. Покрыть метод тестами.
  4. Создать метод, принимающий на вход три аргумента: массив строк, строку-префикс и строку-постфикс. Метод должен вернуть количество строк, содержащихся в массиве и начинающихся со строки-префикса и заканчивающихся строкой-постфиксом. Использовать методы startWith() и endsWith(). Покрыть метод тестами.
  5. Создать метод, принимающий на вход три аргумента: массив строк, строку-префикс и строку-постфикс. Метод должен вернуть количество строк, содержащихся в массиве и начинающихся со строки-префикса и заканчивающихся строкой-постфиксом, игнорируя все пробельные символы в начале и в конце строки. Использовать дополнительно метод trim(). Покрыть метод тестами.
  6. Создать метод, принимающий на вход три строки: метод должен вернуть новую строку, полученную путём замены в первой строки каждое вхождение второй строки на третью строку. Если при этом после замены в результирующей строке вновь остались вхождения второй строки, их нужно заменить повторно до тех пор, пока вхождений не останется (но не более 100 попыток). Например, для строк «ороророро», «оро», «ро» должно получиться «роррро». Использовать методы contains() и replaceAll(). Покрыть метод тестами.
  7. Создать метод, принимающий на вход строку и два целых числа: *from* и *to*. Если первое число отрицательно, его надо заменить на ноль. Если второе число больше длины строки, то его надо заменить на длину строки. Если второе число меньше, либо равно первому, то нужно вернуть пустую строку. В остальных случаях нужно вернуть подстроку первой строки, начинающуюся символом с индексом *from* включительно, и заканчивая индексом *to*, не включая его. Использовать методы substring(). Покрыть метод тестами.
  8. Переопределить в классе Person метод toString() так, чтобы он возвращал имя и фамилию, записанные через пробел. Покрыть метод тестами.
  9. Переопределить метод toString() в классе Point, чтобы он возвращал запись точки в формате [x, y, z]. Покрыть метод тестами.
  10. Переопределить метод toString() в классе NamedPoint: если значение поля name – null, то вернуть результат родительского метода toString() с помощью ключевого слова super. Иначе вернуть строку в формате: имя точки и, через пробел, результат вызова этого же метода у родительского класса. Покрыть метод тестами.
  11. Переопределить метод toString() в классе Matrix. Результат должен выводиться в виде прямоугольной матрицы, где элементы в строке разделены запятыми, а сами строки отделены друг от друга точкой с запятой и переносом строки. Покрыть метод тестами.
  12. Создать метод, принимающий на вход объект класса Object. Метод должен выводить в консоль строку: «Описание объекта: » (оканчивается пробелом), после чего идёт результат выполнения метода toString() у входного объекта. При этом запрещено явно вызывать метод toString(). Запустив метод несколько раз, передавая туда разные объекты, в том числе объекты классов Person, Point, NamedPoint и Matrix.
  13. Создать метод, принимающий на вход строку, разбивающий её на массив строк по символу пробела, и заменяющий строку в массиве на такую же, но первый символ должен быть написан верхним регистром (если это буква). Использовать методы split(), toUpperCase(), substring() и конкатенацию. Примечание: если в строке идут подряд два символа пробела, в массиве должна остаться пустая строка без каких-либо замен (так как она как раз между символами пробела). Метод должен вернуть полученный массив. Покрыть метод тестами.
  14. Создать метод, принимающий на вход массив строк и объединяющий её в одну строку с перечислением через запятую и пробел. Метод должен возвращать полученную строку. Использовать статический метод join класса String, первый элемент которого – разделитель, а второй – массив строк. Покрыть метод тестами.
  15. Создать метод, принимающий на вход строку. Эту строку следует использовать для создания объекта StringBuilder, чтобы произвести следующие модификации: заменить каждый чётный символ на номер этого символа, затем перевернуть строку, сформировать объект String методом toString() и вернуть в качестве результата работы метода. Следует использовать методы replace() и revert(). Покрыть метод тестами.
  16. Создать метод, принимающий на вход целое число. Метод должен с использованием объекта класса StringBuilder сформировать строку, состоящую из перечисления через пробел всех чисел от 0 (включительно) до входного (исключительно). Полученную строку метод должен вернуть. Покрыть метод тестами. Вывести в консоль результат работы метода при входном параметре 10000.
  17. Вывести в консоль значение кодировки по умолчанию (зависит от системных настроек) с помощью метода java.nio.charset.Charset.defaultCharset().
  18. Создать метод, принимающий на вход строку String и два объекта типа Charset. Метод должен вернуть новую строку, переводящую входную строку из первой кодировки во вторую. Для этого у входной строки можно вызвать метод getBytes(), в который необходимо передать первую кодировку, а затем создать новую строку String с помощью конструктора: первый параметр – массив полученный ранее байт, второй параметр – вторая кодировка. Проверить работу метода через вывод в консоль. В качестве кодировок использовать те, что объявлены в классе java.nio.charset.StandardCharsets.