**Задание 3**

Не удаляйте классы из Задания 2, они нам еще понадобятся!

На этом занятии мы познакомимся со строками и оболочками типов, модифицируем наши оконные классы, используя строки, а также создадим еще один оконный класс.

Распакуйте архив “Задание 3.zip“ в Ваш рабочий каталог. В Вашем проекте после этого должны появиться классы TestStringOperations и TestNumberOperations в каталоге test/java в пакете net.thumbtack.school.base и классы TestDesktop, TestPoint, TestRectButton, TestRoundButton, TestListBox и TestWindowFactory в каталоге test/java в пакете net.thumbtack.school.windows.v2

Создайте пакет net.thumbtack.school.base в каталоге main/java и в нем следующие классы и методы в них. Все методы будут иметь атрибут static. При написании методов постарайтесь ”не изобретать велосипед” : прежде чем писать свою реализацию метода, проверьте, нет ли в классах String, StringJoiner, StringBuilder метода, который решает эту задачу полностью или частично.

**StringOperations**

1. **public static int getSummaryLength(String[] strings)**

Возвращает суммарную длину строк, заданных массивом strings.

1. **public static String getFirstAndLastLetterString(String string)**

Возвращает двухсимвольную строку, состоящую из начального и конечного символов заданной строки.

1. **public static boolean isSameCharAtPosition(String string1, String string2, int index)**

Возвращает true, если обе строки в позиции index содержат один и тот же символ, иначе false.

1. **public static boolean isSameFirstCharPosition(String string1, String string2, char character)**

Возвращает true, если в обеих строках первый встреченный символ character находится в одной и той же позиции. Просмотр строк ведется от начала.

1. **public static boolean isSameLastCharPosition(String string1, String string2, char character)**

Возвращает true, если в обеих строках первый встреченный символ character находится в одной и той же позиции. Просмотр строк ведется от конца.

1. **public static boolean isSameFirstStringPosition(String string1, String string2, String str)**

Возвращает true, если в обеих строках первая встреченная подстрока str начинается в одной и той же позиции. Просмотр строк ведется от начала.

1. **public static boolean isSameLastStringPosition(String string1, String string2, String str)**

Возвращает true, если в обеих строках первая встреченная подстрока str начинается в одной и той же позиции. Просмотр строк ведется от конца.

1. **public static boolean isEqual(String string1, String string2)**

Возвращает true, если строки равны.

1. **public static boolean isEqualIgnoreCase(String string1, String string2)**

Возвращает true, если строки равны без учета регистра (например, строки “abc” и “aBC” в этом смысле равны).

1. **public static boolean isLess(String string1, String string2)**

Возвращает true, если строка string1 меньше строки string2.

1. **public static boolean isLessIgnoreCase(String string1, String string2)**

Возвращает true, если строка string1 меньше строки string2 без учета регистра (например, строка “abc” меньше строки “ABCd” в этом смысле).

1. **public static String concat(String string1, String string2)**

Возвращает строку, полученную путем сцепления двух строк.

1. **public static boolean isSamePrefix(String string1, String string2, String prefix)**

Возвращает true, если обе строки string1 и string2 начинаются с одной и той же подстроки prefix.

1. **public static boolean isSameSuffix(String string1, String string2, String suffix)**

Возвращает true, если обе строки string1 и string2 заканчиваются одной и той же подстрокой suffix.

1. **public static String getCommonPrefix(String string1, String string2)**

Возвращает самое длинное общее “начало” двух строк. Если у строк нет общего начала, возвращает пустую строку.

1. **public static String reverse(String string)**

Возвращает перевернутую строку.

1. **public static boolean isPalindrome(String string)**

Возвращает true, если строка является палиндромом, то есть читается слева направо так же, как и справа налево.

1. **public static boolean isPalindromeIgnoreCase(String string)**

Возвращает true, если строка является палиндромом, то есть читается слева направо так же, как и справа налево, без учета регистра.

1. **public static String getLongestPalindromeIgnoreCase(String[] strings)**

Возвращает самый длинный палиндром (без учета регистра) из массива заданных строк. Если в массиве нет палиндромов, возвращает пустую строку.

1. **public static boolean hasSameSubstring(String string1, String string2, int index, int length)**

Возвращает true, если обе строки содержат один и тот же фрагмент длиной length, начиная с позиции index.

1. **public static boolean isEqualAfterReplaceCharacters(String string1, char replaceInStr1, char replaceByInStr1, String string2, char replaceInStr2, char replaceByInStr2)**

Возвращает true, если после замены в string1 всех вхождений replaceInStr1 на replaceByInStr1 и замены в string2 всех вхождений replaceInStr2 на replaceByInStr2 полученные строки равны.

1. **public static boolean isEqualAfterReplaceStrings(String string1, String replaceInStr1, String replaceByInStr1, String string2, String replaceInStr2, String replaceByInStr2)**

Возвращает true, если после замены в string1 всех вхождений строки replceInStr1 на replaceByInStr1 и замены в string2 всех вхождений replceInStr2 на replaceByInStr2 полученные строки равны.

1. **public static boolean isPalindromeAfterRemovingSpacesIgnoreCase(String string)**

Возвращает true, если строка после выбрасывания из нее всех пробелов является палиндромом, без учета регистра.

1. **public static boolean isEqualAfterTrimming(String string1, String string2)**

Возвращает true, если две строки равны, если не принимать во внимание все пробелы в начале и конце каждой строки.

1. **public static String makeCsvStringFromInts(int[] array)**

Для заданного массива целых чисел создает текстовую строку, в которой числа разделены знаком “запятая” (т.н. формат CSV - comma separated values). Для пустого массива возвращается пустая строка.

1. **public static String makeCsvStringFromDoubles(double[] array)**

Для заданного массива вещественных чисел создает текстовую строку, в которой числа разделены знаком “запятая”, причем каждое число записывается с двумя знаками после точки. Для пустого массива возвращается пустая строка.

1. **public static StringBuilder makeCsvStringBuilderFromInts(int[] array)**

То же, что и в упражнении 25, но возвращает StringBuilder.

1. **public static StringBuilder makeCsvStringBuilderFromDoubles(double[] array)**

То же, что и в упражнении 26, но возвращает StringBuilder.

1. **public static StringBuilder removeCharacters(String string, int[] positions)**

Удаляет из строки символы, номера которых заданы в массиве positions. Предполагается, что будут передаваться только допустимые номера, упорядоченные по возрастанию. Номера позиций для удаления указаны для исходной строки. Возвращает полученный в результате StringBuilder.

1. **public static StringBuilder insertCharacters(String string, int[] positions, char[] characters)**

Вставляет в строку символы. Массивы positions и characters имеют одинаковую длину. В позицию positions[i] в исходной строке string вставляется символ characters[i]. Если в массиве positions один и тот же номер позиции повторяется несколько раз, это значит, что в указанную позицию вставляется несколько символов, в том порядке, в котором они перечислены в массиве characters. Предполагается, что будут передаваться только допустимые номера, упорядоченные по неубыванию. Возвращает полученный в результате StringBuilder.

**NumberOperations**

1. **public static Integer find(int[] array, int value)**

Ищет в массиве array первый элемент, значение которого равно value. Если такое значение найдено, возвращает его позицию в массиве array, в противном случае возвращает null.

1. **public static Integer find(double[] array, double value, double eps)**

Ищет в массиве array первый элемент, значение которого по модулю не отличается от value более чем на eps. Если такое значение найдено, возвращает его позицию в массиве array, в противном случае возвращает null.

1. **public static Double calculateDensity(double weight, double volume, double min, double max)**

Вычисляет плотность вещества по формуле weight / volume. Если получившееся значение не превышает max и не меньше min, возвращает полученное значение, в противном случае возвращает null.

1. **public static Integer find(BigInteger[] array, BigInteger value)**

Ищет в массиве array первый элемент, значение которого равно value. Если такое значение найдено, возвращает его позицию в массиве array, в противном случае возвращает null.

1. **public static BigDecimal calculateDensity(BigDecimal weight, BigDecimal volume, BigDecimal min, BigDecimal max)**

Вычисляет плотность вещества по формуле weight / volume. Если получившееся значение не превышает max и не меньше min, возвращает полученное значение, в противном случае возвращает null.

**Модификация классов окон из Задания 2**

Создайте пакет net.thumbtack.school.windows.v2 в каталоге main/java и скопируйте в него все классы из пакета net.thumbtack.school.windows.v1. Не забудьте изменить в каждом классе package!

Добавьте в классы RectButton и RoundButton в пакете net.thumbtack.school.windows.v2 дополнительное поле - текст в окне.

Модифицируйте конструкторы, добавив в каждый из них еще один параметр - String text.

Добавьте методы в каждый класс

1. **public String getText()**

Возвращает текст окна

1. **public void setText(String text)**

Устанавливает текст окна

Создайте новый класс в пакете net.thumbtack.school.windows.v2

**ListBox**

Прямоугольное окно, содержащее в себе список строк. Для ListBox определено 2 состояния - активен (изображается черным цветом) и пассивен (серым цветом). Предполагается, что всегда будут передаваться допустимые координаты, то есть при создании или изменении всегда будет выполняться : левая точка не правее правой, верхняя точка не ниже нижней.

Обращаем Ваше внимание, что ListBox должен иметь свою собственную копию массива, а не хранить в себе переданный массив, поскольку переданный массив может впоследствии быть изменен за пределами класса ListBox. Для создания копии массива можно воспользоваться методом System.arraycopy, который копирует массив. Сами строки при этом не копируются, так как необходимости в этом нет - строки иммутабельные. Новый массив после System.arraycopy будет содержать ссылки на те же строки.

1. **public ListBox(Point topLeft, Point bottomRight, boolean active, String[] lines)**

Создает ListBox по координатам углов - левого верхнего и правого нижнего, флагу активности и набору строк. Обращаем внимание на то, что обе точки входят в ListBox, так что если создать ListBox с topLeft.equals(bottomRight), то будет создан ListBox ширины и высоты 1. Параметр lines может быть null.

1. **public ListBox(int xLeft, int yTop, int width, int height, boolean active, String[] lines)**

Создает ListBox по координатам левого верхнего угла, ширине, высоте, флагу активности и набору строк. Параметр lines может быть null.

1. **public ListBox(Point topLeft, Point bottomRight, String[] lines))**

Создает активный ListBox по координатам углов - левого верхнего и правого нижнего и набору строк. Параметр lines может быть null.

1. **public ListBox(int xLeft, int yTop, int width, int height, String[] lines))**

Создает активный ListBox по координатам левого верхнего угла, ширине и высоте и набору строк. Параметр lines может быть null.

1. **public Point getTopLeft()**

Возвращает левую верхнюю точку ListBox.

1. **public Point getBottomRight()**

Возвращает правую нижнюю точку ListBox.

1. **public boolean isActive()**

Возвращает true, если ListBox активен, иначе false

1. **public void setTopLeft(Point topLeft)**

Устанавливает левую верхнюю точку ListBox.

1. **public void setBottomRight(Point bottomRight)**

Устанавливает правую нижнюю точку ListBox.

1. **public void setActive(boolean active)**

Устанавливает состояние активности ListBox.

1. **public int getWidth()**

Возвращает ширину ListBox.

1. **public int getHeight()**

Возвращает высоту ListBox.

1. **public String[] getLines()**

Возвращает копию набора строк ListBox.

1. **public void setLines(String[] lines)**

Устанавливает набор строк ListBox.

1. **public String[] getLinesSlice(int from, int to)**

Возвращает набор строк ListBox, начиная со строки “from” и до строки (“to”-1) включительно . Если в ListBox строк меньше, чем “to”, возвращает строки от “from” и до конца. Гарантируется, что “from” < “to”. Если массив строк равен null, возвращает null.

1. **public String getLine(int index)**

Возвращает строку с номером index. Если строки с таким номером нет или массив строк равен null, возвращает null.

1. **public void setLine(int index, String line)**

Заменяет строку с номером index. Если строки с таким номером нет или массив строк равен null, ничего не делает.

1. **public Integer findLine(String line)**

Ищет первую совпадающую с line строку в массиве строк ListBox. Если строка найдена, возвращает ее индекс, в противном случае возвращает null.

1. **public void reverseLineOrder()**

Переворачивает массив строк ListBox., то есть делает 0-ю строку - последней, первую - предпоследней и т.д. Если массив строк равен null, не делает ничего.

1. **public void reverseLines()**

Переворачивает каждую строку в массиве строк ListBox.Если массив строк равен null, не делает ничего.

1. **public void duplicateLines()**

Заменяет массив строк на новый массив, вместо каждой строки вставлены две копии ее. Если массив строк равен null, не делает ничего.

1. **public void removeOddLines()**

Заменяет массив строк на новый массив, в котором каждая нечетная исходная строка удалена. Если массив строк равен null или содержит только одну строку, не делает ничего.

1. **public boolean isSortedDescendant()**

Возвращает true, если массив строк строго упорядочен по убыванию, иначе false. Если массив строк равен null, возвращает true.

1. **public void moveTo(int x, int y)**

Передвигает ListBox так, чтобы левый верхний угол его оказался в точке (x, y).

1. **public void moveTo(Point point)**

Передвигает ListBox так, чтобы левый верхний угол его оказался в точке point.

1. **public void moveRel(int dx, int dy)**

Передвигает ListBox на (dx, dy).

1. **public void resize(double ratio)**

Изменяет ширину и длину ListBox в ratio раз при сохранении координат левой верхней точки. Дробная часть вычисленной длины или ширины отбрасывается. Если при таком изменении длина или ширина какой-то из сторон окажется меньше 1, то она принимается равной 1.

1. **public boolean isInside(int x, int y)**

Определяет, лежит ли точка (x, y) внутри ListBox. Если точка лежит на стороне, считается, что она лежит внутри.

1. **public boolean isInside(Point point)**

Определяет, лежит ли точка point внутри ListBox. Если точка лежит на стороне, считается, что она лежит внутри.

1. **public boolean isIntersects(ListBox listBox)**

Определяет, пересекается ли ListBox с другим ListBox. Считается, что ListBox’ы пересекаются, если у них есть хоть одна общая точка.

1. **public boolean isInside(ListBox listBox)**

Определяет, лежит ли ListBox целиком внутри текущего ListBox.

1. **public boolean isFullyVisibleOnDesktop(Desktop desktop)**

Определяет, верно ли, что весь ListBox находится в пределах Desktop.

1. **методы equals и hashCode.**

Не пишите эти методы сами, используйте средства IDEA.

Проверьте работу тестов в консольном окне, запишите все классы на сервер (не забудьте изменить текст сообщения в git commit!) и убедитесь, что на сервере все тесты также проходят успешно (см. Занятие 1, п.15-19)