Лабораторная работа 3. Идентификаторы, переменные, типы данных.

Цель:

- создание приложений, использующих встроенные типы данных и литералы различных типов;
- создание приложений, демонстрирующих возможность работы с символами Unicode, использование символов Unicode в составе идентификаторов.

Задание 1. Использование встроенных типов

Требуется разработать программу расчета площади треугольника по формуле Герона.

1. В программе объявите три переменные вещественного типа float (стороны треугольника) и инициализируйте подходящими значениями (например: a1 = 15, a2 = 15, a3 = 15):
2. Вычислите полупериметр треугольника по формуле p = (a1+a2+a3)/2;
3. Вычислите площадь треугольника.

$$S = \sqrt{p(p-a1)(p-a2)(p-a3)}$$

Для расчета по формуле Герона требуется функция квадратного корня. Математические функции реализованы в виде статических методов класса **Math**, который расположен в пакете java.lang. Для того чтобы использовать статические методы класса необходимо перед именем метода указать, используя dot-натацию, имя класса, в котором описан данный метод. Результат расчета выведите на экран ("Площадь треугольника равна: " + s).

4. Постройте и протестируйте приложение.
5. Модифицируйте созданное приложение таким образом, чтобы оно осуществляло запрос каких-либо данных от пользователя.
6. Для считывания числа с клавиатуры, можно использовать класс Scanner, который обеспечивает чтение форматированного ввода и преобразования его в бинарную форму. Изучите следующую информацию об этом классе.

Общие сведения о классе Scanner

Класс **Scanner** определяет несколько конструкторов, и объект класса может быть создан для String, InputStream или File.

Например, следующая строка создает объект Scanner, читающий из стандартного ввода, которым по умолчанию является клавиатура:

Scanner con = new Scanner(System.in);

В этом случае вызывается конструктор класса Scanner вида Scanner (InputStream a), потому что System.in является объектом типа InputStream.

После создания объекта класса **Scanner**, он используется для чтения форматированного ввода. В общем случае, **Scanner** читает лексемы из некоторого лежащего в основе источника, который указывается при создании объекта **Scanner.** С точки зрения **Scanner** лексема – это порция ввода, отделенная набором разделителей, которыми по умолчанию являются пробелы. Лексема читается в соответствии с определенным регулярным выражением, задающим формат данных. Хотя **Scanner** позволяет определить специфический тип выражения, которому будет соответствовать следующая операция ввода, он включает множество предопределенных шаблонов, соответствующих примитивным типам, таким как int и double**,** а также строкам. Эти шаблоны реализованы в виде методов класса **Scanner.**

В общем случае для использования **Scanner** рекомендуется следовать следующей процедуре:

1) Определить, доступен ли специфический тип ввода, вызовом одного из методов hasNextX**,** где X— нужный тип данных.
2) Если ввод доступен, считать его одним из методов nextX.
3) Повторять процесс до завершения ввода.

Таким образом, класс **Scanner** определяет два набора методов, которые позволяют читать ввод. Первый набор – это методы hasNextX**,** определяющие, доступен ли указанный тип ввода. Например, вызов hasNexInt() возвращает **true** только в том случае, если следующая лексема, подлежащая чтению, является целым числом. И второй набор методов, если требуемые данные доступны, они считываются одним из методов nextX**.**

7. Доработайте программу, добавьте строку кода, позволяющую пользователю увидеть приглашение на ввод данных, например "Введите стороны треугольника: ".
8. Создайте объект класса **Scanner** с именем cin, для инициализации используйте вызов конструктора со следующими параметрами: new Scanner(System.in);
9. Определите, доступен ли специфический тип ввода вызовом метода hasNextFloat, и прочитайте данные методом nextX.
10. Добавьте инструкцию для импорта пакета java.util.Scanner;
11. Постройте и протестируйте приложение.

Задание 2. Реализация форматированного вывода

Пакет java.io включает класс **PrintStream**, у которого есть два метода форматирования, которые можно использовать вместо методов print и println. Это эквивалентные друг другу методы format и printf, имеющие следующую основную общую форму:

PrintStream printf(String ***fmtstring,*** Object ... ***args)***
PrintStream format(String ***fmtstring,*** Object ... ***args)***

Аргументы *args* записываются в стандартный вывод в формате, указанном параметром *fmtstring*, используя локальные установки по умолчанию.

Строка формата содержит открытый текст и ***спецификаторы формата***, которые являются специальными символами, форматирующими параметры Object... args.

Спецификаторы формата начинаются со знака процента (%) и заканчиваются конвертером. Конвертер – символ, указывающий тип параметра, который будет отформатирован. Между знаком процента (%) и конвертером можно указать дополнительные флаги и спецификаторы.

Поскольку объект **System.out** имеет тип **PrintStream,** можно вызывать методы, возвращающие объект типа **PrintStream** с объектом-потоком **System.out.** Поэтому методы printf() и format() могут служить в качестве замены println(), в случаях, когда необходимо выдавать на консоль форматированный вывод.

1. Реализуйте в программе из задания 1 возможность форматного вывода. Для этого добавьте, например, вывод в вещественном формате с указанием количества знаков после запятой (методы могут быть printf() или format()):
2. Добавьте возможность составного сообщения, например, на экране пользователь увидит: "Для сторон 15, 15 и 15 площадь равна: 22,5";
3. Протестируйте работу программы.

Задание 3. Использование символов Unicode

Для представления символов в Java используется Unicode, определяющий международный набор символов, который может представлять все символы, присутствующие во всех известных языках.

Unicode представляет собой унифицированные наборы символов, таких как латиница, греческий алфавит, арабский алфавит, кириллица, иврит, японские и тайские иероглифы и множество других.

Unicode представляет символы кодом из 2 байт, описывая, таким образом, 65.535 символов. Это позволяет поддерживать практически все распространенные языки мира. Первые 128 символов совпадают с набором ASCII.

Требуется некоторое специальное обозначение, чтобы иметь возможность задавать в программе любой символ Unicode. Стандартная конструкция задания символов, например ‘a’ представляет символ Unicode, используя только символы ASCII. Если в программу нужно вставить знак с кодом 6917, который не входит в ASCII, то необходимо его представить в шестнадцатеричном формате (1B05) и записать в виде: \u1B05 причем буква u должна быть прописной, а шестнадцатеричные цифры A, B, C, D, E, F можно использовать произвольно, как заглавные, так и строчные. Таким образом, можно закодировать все символы Unicode от \u0000 до \uFFFF. Буквы русского алфавита начинаются с \u0410 (только буква Ё имеет код \u0401) по \u044F (код буквы ё \u0451).

1. Реализуйте в программе предыдущего упражнения возможность использования Unicode.
2. Протестируйте работу программы.