Назначение лекции

Знакомство с особенностями чтения текстовых данных из файла.

Новые слова

**ArrayList** – структура хранения данных подобная массиву, но количество элементов не фиксировано.

**Отладчик –** специальный режим системы программирования, который дает возможность выполнять приложение по «шагам» и просматривать онлайн знчения переменных, выражений, массивов и пр.

Что было сделано

В предыдущей работе наше приложение MyEvent научилось выдавать список доступных режимов работы и выполнять контроль правильности номера режима. Если номер был введен не верно, то ввод повторялся.

Далее рассмотрели реализацию режима «Ввод» в части ввода текущей даты и названия события.

Рассмотрели реализацию режима ввода в части ввода количества времени на событие (вещественное число) и состояния события. В программе реализовано два состояния: «было» и «будет». Для состояния «будет» понадобится дата этого события.

Разработан механизм ввода даты для будущего события.

Разработали механизм записи данных о событии в файл в текстовом виде.

Продолжение

В этой лекции реализуем чтение, ранее записанных событий, из файла и запись этих данных в массив.

Результат записи в файл предыдущей работы:

Программа <MyEvent>. Автор Касымов Д.Д.

20-07-2022

Питание

2.3

Было

20-07-2022

20-07-2022

Прочее

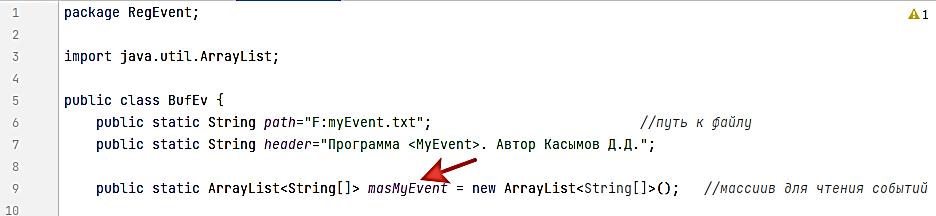
4.0

Будет

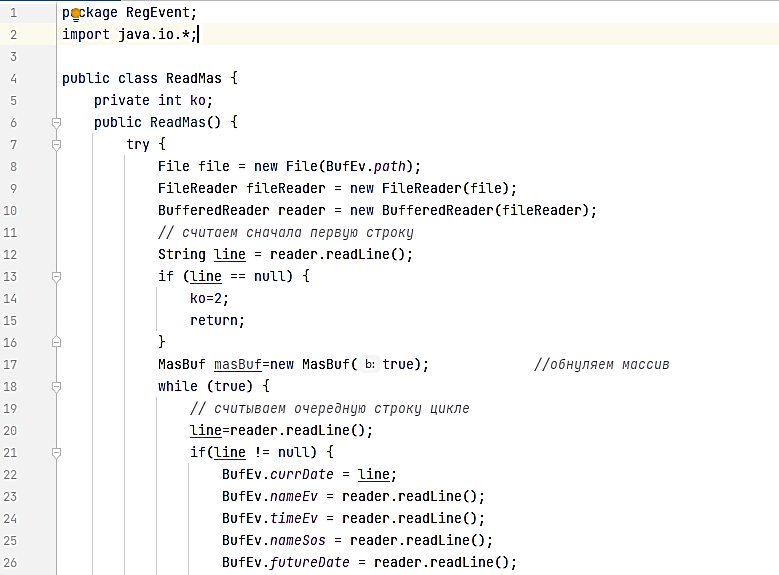
12-12-2022

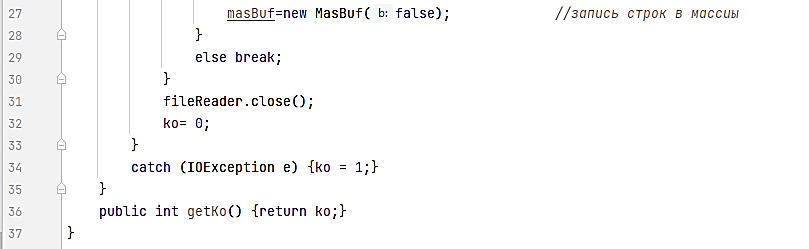
Чтение реализуем в два этапа: первый – чтение в специальный массив (список), второй – выдача списка на экран.

В начале впишем наш список в класс *BufEv*:



Для чтения данных из файла в массив *masMyEvent* откроем новый класс *ReadMas*





Выполним разбор каждой строки кода:

private int ko;

Внутренняя переменная с кодом ответа. Для ее просмотра из других классов вводится метод *GetKo.*

try {

Блок для перехвата исключения при чтения файла.

File file = new File(BufEv.*path*);

Создание объекта *file* для указанием пути и имени файла

FileReader fileReader = new FileReader(file);

Открытие файла *fileReader* для чтения

BufferedReader reader = new BufferedReader(fileReader);

Создание буфера *reader* для чтения файла. При этом «головка» чтения устанавливается на первую запись файла.

String line = reader.readLine();

Чтение первого элемента файла в строку *line*

if (line == null)

Проверка элемента на «пусто». Если пусто, то ошибка с кодом 2 и завершение работы объекта.

MasBuf masBuf=new MasBuf(true); *//обнуляем массив*

Создание объекта *masBuf* класса *MasBuf* с параметром *true* для подготовки массива *masMyEvent* для чтения файла. Код класса *MasBuf* опишем далее.

while (true) {

Бесконечный цикл для по-строчного чтения данных из файла *MyEvent* в подготовленный массив. После каждого чтения «головка» чтения автоматически перемещается на следующий элемент.

line=reader.readLine();

Чтение очередной строки из файла строку *line*

if(line != null) {

Если строка не пуста, то записываем ее:

BufEv.*currDate* = line;

Запись даты события

ufEv.*nameEv* = reader.readLine();

Чтение имени события

BufEv.*timeEv* = reader.readLine();

Чтение количества времени для события.

BufEv.*nameSos* = reader.readLine();

Чтение состояния события

BufEv.*futureDate* = reader.readLine();

Чтение даты будущего события

masBuf=new MasBuf(false); *//запись строк в массиы*

Запись всех прочитанных данных в массив *masMyEvent*

else break;

Если очередная строка пуста, то следовательно это конец данных и выходим из бесконечного цикла.

fileReader.close();  
ko= 0;

После цикла закрываем наш файл.

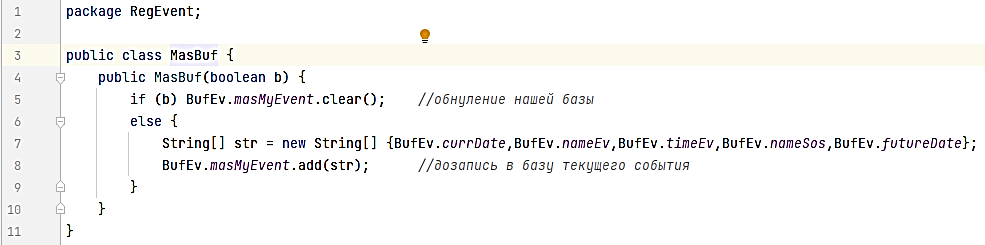
catch (IOException e) {ko = 1;}

Если при чтении произошло исключение, то *ко=1*

public int getKo() {return ko;}

Метод для получения кода ответа из другого класса.

Теперь рассмотрим как устроен класс *MasBuf* для записи очередного события в массив:



public MasBuf(boolean b) {

Конструктор класса с параметром *b* логического типа. Если истина, то обнулить массив.

if (b) BufEv.*masMyEvent*.clear(); *//обнуление нашей базы*

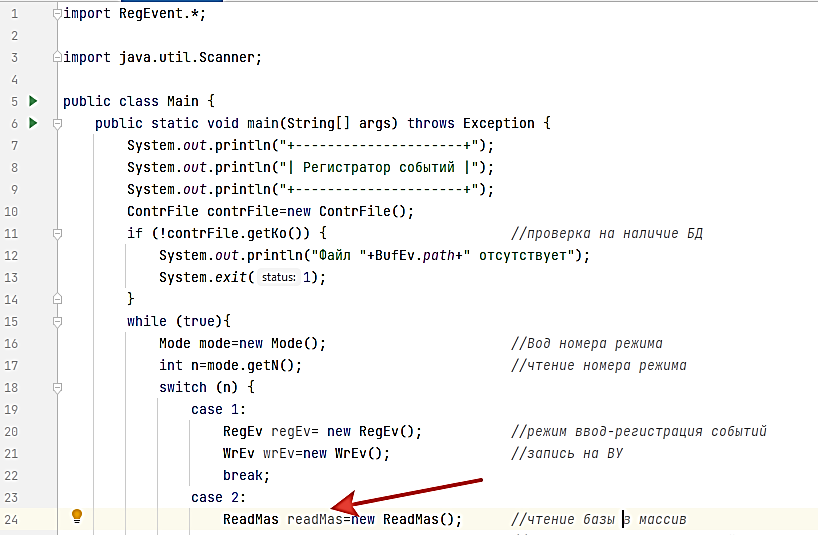
Обнуление массива при *b=true*

else {  
 String[] str = new String[]

{BufEv.*currDate*,BufEv.*nameEv*,BufEv.*timeEv*,BufEv.*nameSos*,BufEv.*futureDate*};  
 BufEv.*masMyEvent*.add(str); *//дозапись в базу текущего события*}

Иначе добавление (*add*) в массив *masMyEvent* очередного события.

Далее добавим код в класс *Main*:



На этом первый этап режима «Вывод» закончен и на следующем занятии приступим ко второму этапу: выдаче массива *masMyEvent* на экран.

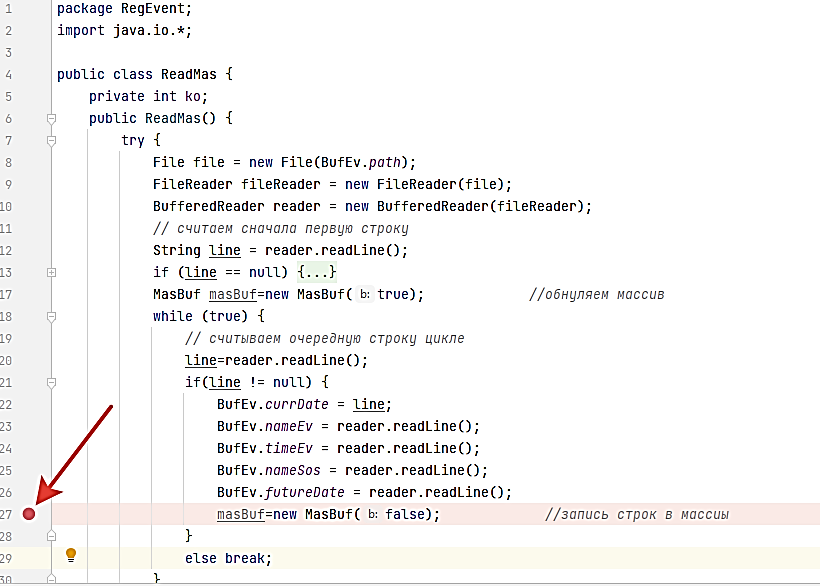
# Отладчик

Пока попробуем поработать с отладчиком. Отладчик, это специальный режим работы системы программирования, который позволяет выполнять программу по “шагам”, и на каждом шаге просматривать значения переменных, выражений, массивов и пр. Это облегчает процесс отладки.

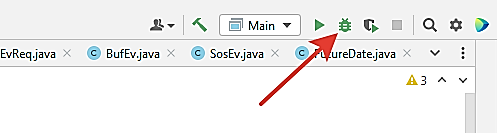
Для отладки требуется установить точку останова (одну или несколько) и запустить программу в режиме отладки. Система выполнит останов в точке останова. В этот момент можно просмотреть требуемые значения и продолжить работу по шагам или до конца.

Рассмотрим этот процесс более подробно.

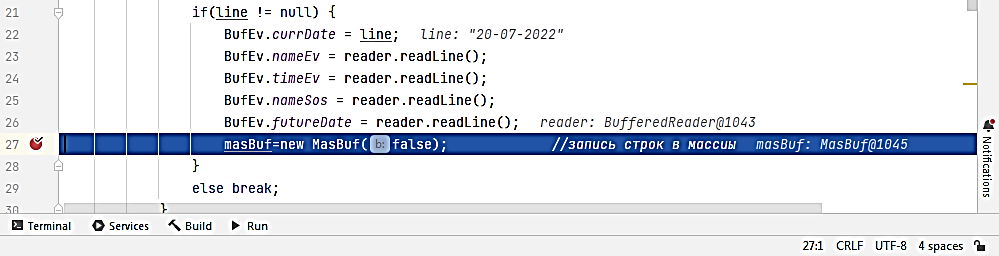
Откроем класс *ReadMas* и установим точку останова. Для этого на левом поле щелкнуть мышкой в нужной строке:



При запуске программы в режие отладки:



Программа остановится в этом месте, если выберем режим работы 2 (Вывод):

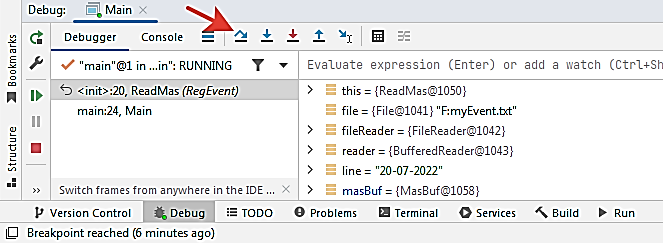


После этого система будет ждать действий программиста. Действия могут быть разными: завершение отладки, выполнить шаг без захода в подпрограмму, выполнить шаг с заходом в подпрограмму и пр.

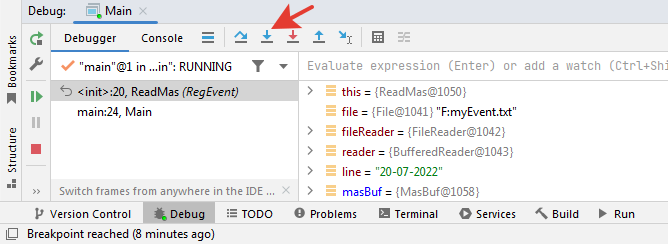
Завершение отладки:



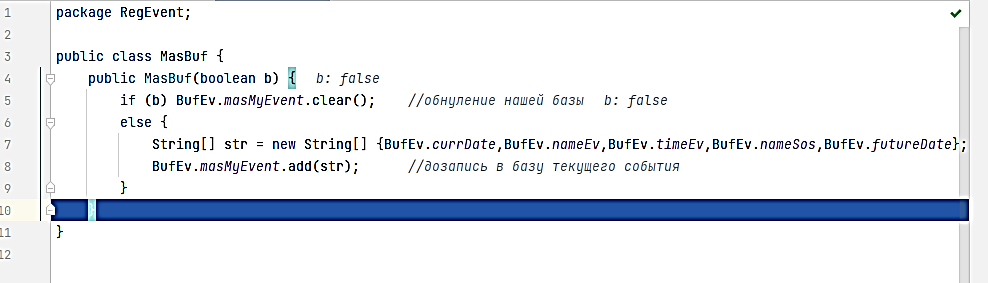
Выполнить шаг без захода в подпрограмму:



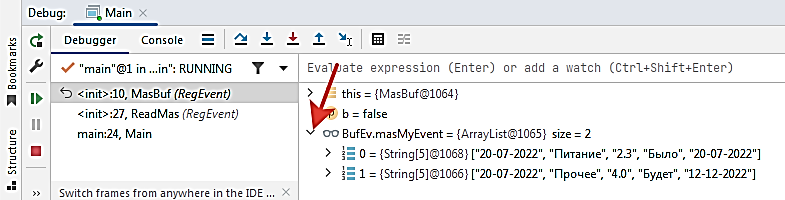
Выполнить шаг с заходом в подпрограмму:



Выполним шаг с заходом в подпрограмму. В этом случае мы окажемся в классе *MasBuf* и пройдем его по шагам, нажимая ту же кнопку:



После двойного прохода можно просмотреть состав массива *masMyEvent*:



Из скриншота видно, что нулевой элемент массива содержит событие «Питание», а первый – «Прочее». Размерность массива = 2. Вспомним, что индексация массива начинается с нуля.

Конец лекции