Продолжение знакомства с организацией простейших запросов программным способом (без SQL) к базе наших событий, которые хранятся в файле.

Новые слова

**Параметр –** данные которые передаются в подпрограмму и которые учитываются при выполнении алгоритма.

Что было сделано

За предыдущие работы 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.

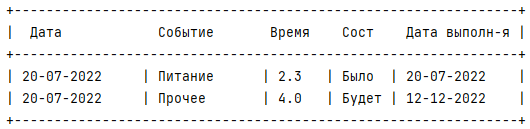
Рассмотрели реализацию режима ввода в части ввода количества времени на событие (вещественное число) и состояния события. В программе реализовано два состояния (статус): «было» и «будет». Для состояния «будет» понадобится дата этого события.

Разработан механизм ввода даты для будущего события.

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

Реализовано чтение, ранее записанных событий, из файла и запись в массив.

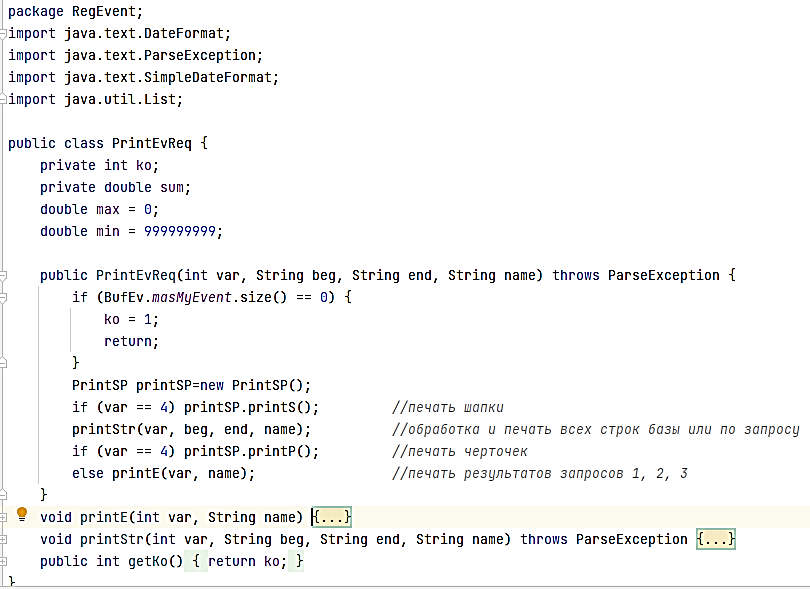
Далее подробно рассмотрели вывод на экран текстовых данных из массива в следующем виде:



Далее реализуем до конца механизм запросов к нашим данным.

Продолжение

В этой лекции выполним разбор механизма выполнения запросов. Это реализовано в классе PrintEvReq



В данном классе конструктор класса оснащен несколькими параметрами. Параметры в подпрограмме (а конструктор – это тоже подпрограмма) обычно используются в том случае, если ожидается использование различных вариантов. В нашем случае – это номер запроса, период и имя события.

Кроме конструктора, в классе есть внутренние методы printE, printStr. Вместо кода – три точки. Это означает наличие внутреннего кода. Что бы его открыть достаточно щелкнуть по нему мышкой.

public PrintEvReq(int var, String beg, String end, String name) throws ParseException {

Конструктор с параметрами. *var* –номер запроса, *beg* – начало периода, *end*-конец периода, *name*-имя события.

if (BufEv.*masMyEvent*.size() == 0) {  
 ko = 1;  
 return;  
}

На всякий случай проверяем: есть ли события в базе?

PrintSP printSP=new PrintSP();

Создаем объект для выдачи шапки и строки черточек

if (var == 4) printSP.printS(); *//печать шапки*

Для запроса 4 выдаем шапку. Для остальных запросов это не требуется.

printStr(var, beg, end, name);

Выдача на экран списка событий, с учетом номера запроса, периода и имени события. Сам код будет приведен ниже.

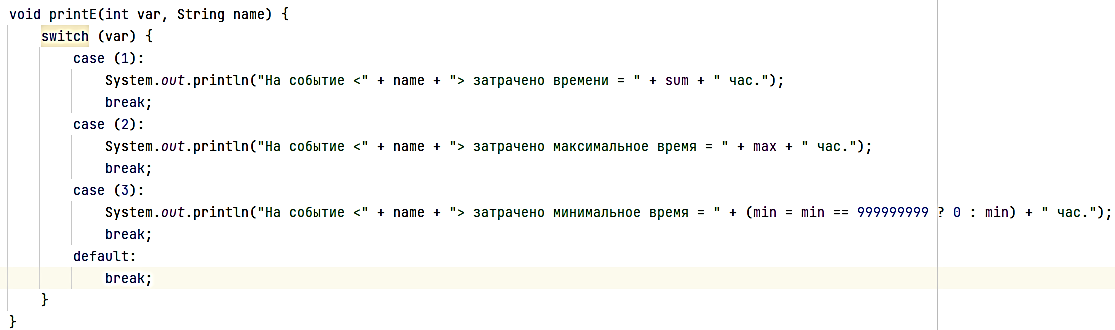
if (var == 4) printSP.printP();

Для запроса 4 выдаем строку черточек (для красоты). Для остальных запросов это не требуется.

else printE(var, name);

Выдача результатов запросов 1, 2, 3

Теперь разберем внутренние методы.



void printE(int var, String name) {

Название метода с параметрами: номер запроса и его имя.

switch (var) {

Переключатель по номеру запрса.

case (1):  
 System.*out*.println("На событие <" + name + "> затрачено времени = " + sum + " час.");

Печать затраченного времени по первому запросу.

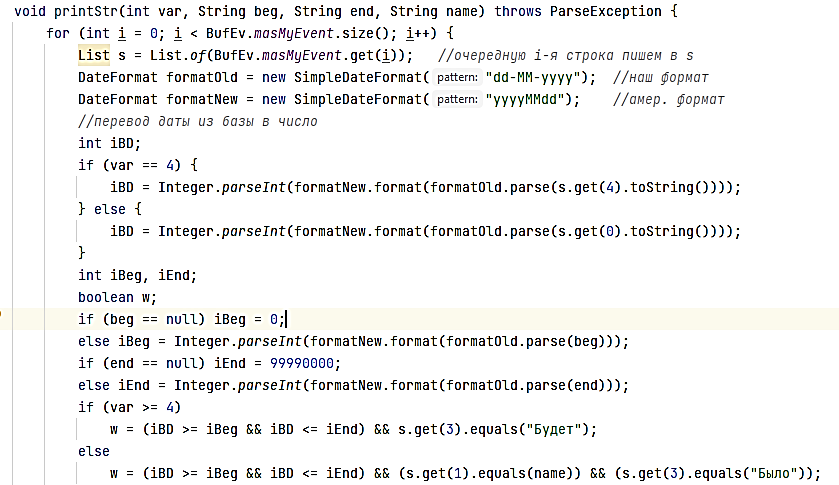
case (2):  
 System.*out*.println("На событие <" + name + "> затрачено максимальное время = " + max + " час.");

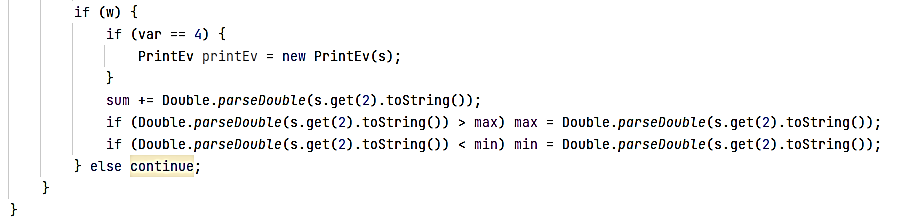
Печать максимального времени по второму запросу.

case (3):  
 System.*out*.println("На событие <" + name + "> затрачено минимальное время = " + (min = min == 999999999 ? 0 : min) + " час.");

Печать минимального времени по третьему запросу. При это учитывается ограничение 999999999. В этом случае выдается ноль.

Разберем следующий, наиболее сложный метод printStr где и выполняются все основные действия.





void printStr(int var, String beg, String end, String name) throws ParseException {

Объявление внутреннего метода printSt с параметрами и возможным исключением при преобразовании даты, которое не будем учитывать.

for (int i = 0; i < BufEv.*masMyEvent*.size(); i++) {

Цикл по всем событиям (они находятся в массиве *masMyEvent* )

List s = List.*of*(BufEv.*masMyEvent*.get(i));

i-е событие переписываем в структуру *List*, т.к. структура *masMyEvent* это двухмерный массив вообщем. Нам неудобно будет с ним работать.

DateFormat formatOld = new SimpleDateFormat("dd-MM-yyyy"); *//наш формат*DateFormat formatNew = new SimpleDateFormat("yyyyMMdd"); *//амер. формат*

Готовим два формата дат, для того что бы можно было их сравнивать на “больше” – “меньше”. Есть множество готовых функций для сравнения, но работающие для стандартных баз данных. В нашем случае сравнивать будем вручную. Для этого достаточно дату “перевернуть”: вначале год, потом месяц и затем день.

**int iBD;  
if (var == 4) {iBD = Integer.*parseInt*(formatNew.format(formatOld.parse(s.get(4).toString())));}**

**else {iBD = Integer.*parseInt*(formatNew.format(formatOld.parse(s.get(0).toString())));}**

Отдельно преобразуем дату события и дату будущего события (статус “Будет”). Это приходится делать т.к. они в запросе используются по разному. Т.е. при сравнении надо будет использовать либо одну, либо другую, в зависимости от запроса. Поэтому, для простоты, записываем ее в одну переменную *iBD*

if (beg == null) iBeg = 0;  
else iBeg = Integer.*parseInt*(formatNew.format(formatOld.parse(beg)));

Также преобразуем начало периода из формальных параметров или устанавливаем в ноль.

if (end == null) iEnd = 99990000;  
else iEnd = Integer.*parseInt*(formatNew.format(formatOld.parse(end)));

Преобразуем конец периода из формальных параметров или устанавливаем в 99990000.

**if (var >= 4)  
 w = (iBD >= iBeg && iBD <= iEnd) && s.get(3).equals("Будет");  
else  
 w = (iBD >= iBeg && iBD <= iEnd) && (s.get(1).equals(name)) && (s.get(3).equals("Было"));**

Готовим условие для сравнения, отдельно для двух состояний событий. В одном случае при сравнении используется дата будущего события, а в другом дата ввода.

if (w) {

Условие по переменной *w*

if (var == 4) {  
 PrintEv printEv = new PrintEv(s);  
}

Для запроса 4 (выдача будущих событий) создаем объект PrintEv конструктор которого выдает событие (s) на экран. Для остальных запосов это не требуется.

sum += Double.*parseDouble*(s.get(2).toString());

Подсчет суммы времени на событие.

**if (Double.*parseDouble*(s.get(2).toString()) > max) max = Double.*parseDouble*(s.get(2).toString());**

Определение события с максимальным временем

**if (Double.*parseDouble*(s.get(2).toString()) < min) min = Double.*parseDouble*(s.get(2).toString());**

Определение события с минимальным временем

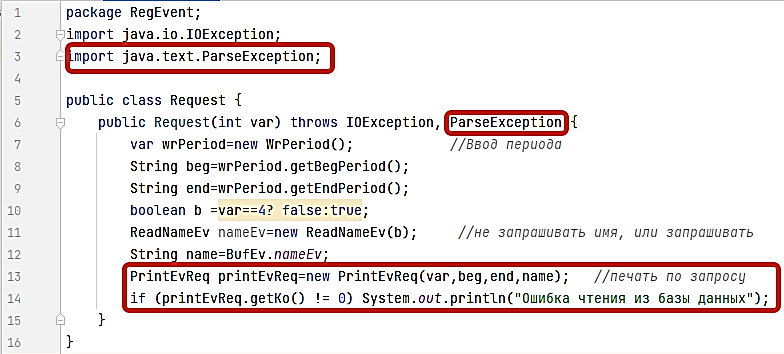
} else continue;

Если условие w=false, то переходим к следующему событию

public int getKo() {return ko;}

Возврат кода ответа.

Далее вернемся к классу из занятия 12 Request и немного изменим его:



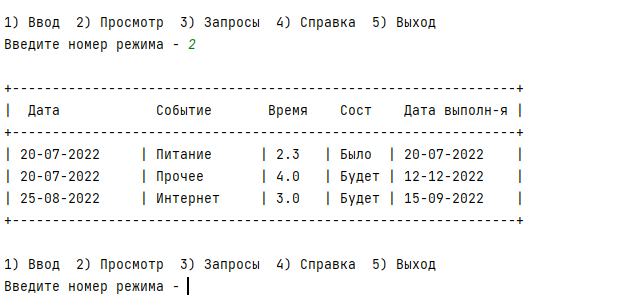
intEvReq printEvReq=new PrintEvReq(var,beg,end,name);

Создание объекта printEvReq для выполнения запроса и печати результатов. Необходимые параметры передаем.

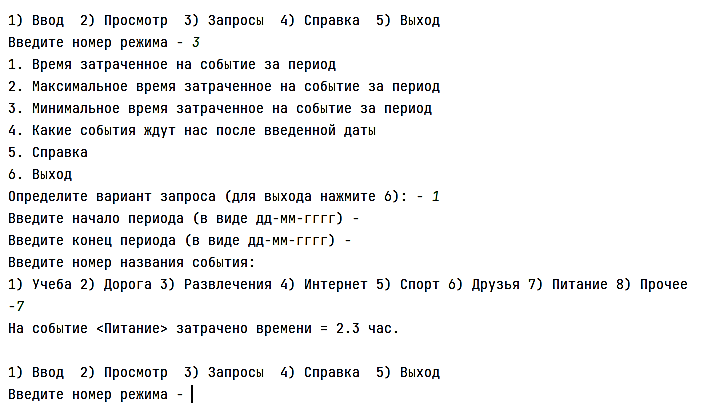
if (printEvReq.getKo() != 0) System.*out*.println("Ошибка чтения из базы данных");

Сообщение об ошибке.

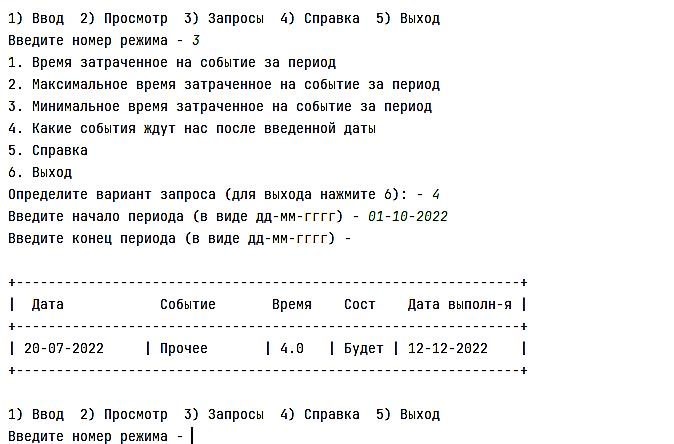
Пример 1 с результатом работы программы.



Пример 2 с результатом работы программы.



Пример 3 с результатом работы программы.



Конец лекции