Назначение лекции

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

Новые слова

**SQL –** Structured Query Language. Язык запросов к реляционной базе данных (состоящий из стандартных таблиц). Результат запроса – таблица.

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.

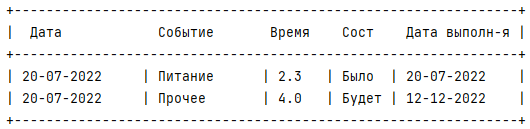
Рассмотрели реализацию режима ввода в части ввода количества времени на событие (вещественное число) и состояния события. В программе реализовано два состояния (статус): «было» и «будет». Для состояния «будет» понадобится дата этого события.

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:



Продолжение

Далее реализуем механизм запросов к нашим данным.

Любые данные, которые хранятся в структурированном виде, в виде стандартной базы данных MySQL, Oracle и пр., или же в файле типа нашего MyEvent не имеют никакой ценности и бесполезны до тех пор, пока отсутствуют средства обработки этих данных.

Для обработки данных стандартных баз данных давно уже реализованы специальные языки запросов, таких, как SQL. Это не язык программирования, в котором используются переменные, циклы и пр. Это язык который содержит команды обращения к базе данных. Потому в языке присутствуют названия таблиц, столбцов, условий выбора данных из таблиц, или же из нескольких таблиц. Результат работы запроса – это тоже таблица.

Наша база данных MyEvent имеет нестандартную структуру, поэтому использовать язык запросов невозможно. В реальности Java имеет стандартные средства по работе с различными базами данных. Но это отдельная тема.

Реализуем механизм обработки наших данных из файла. Для простоты ограничимся запросами из четырех вариантов:

1. Время затраченное на событие за период.
2. Максимальное время затраченное на событие за период.
3. Минимальное время затраченное на событие за период.
4. Какие события ждут нас после введенной даты.

***Время затраченное на событие за период.***

В этом режиме выполняется суммирование времени всех одинаковых событий за указанный период и выдается на экран.

***Максимальное время затраченное на событие за период.***

В этом режиме просматриваются все события и выдается на экран максимальное время на событие за указанный период.

***Минимальное время затраченное на событие за период.***

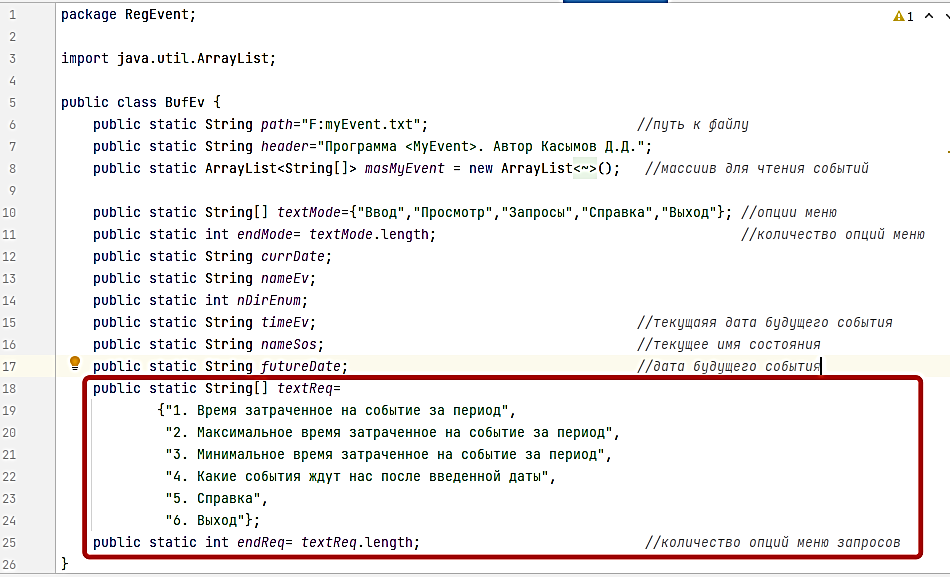
В этом режиме просматриваются все события и выдается на экран минимальное время на событие за указанный период.

***Какие события ждут нас после введенной даты.***

В этом режиме просматриваются все события и выдаются на экран все события в состоянии «Будут» за указанный период.

Заранее договоримся: если начало периода не указано, то считаем его равным нулю. Если конец периода не указан, то он равен 99990000. Это означает 9999 год.

Текст запросов сохраним в нашем классе BufEv:



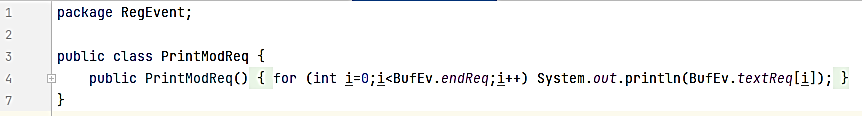
Мы помещаем эти данные в этот класс по двум причинам:

- количество запросов может менятся, и изменения вводятся в одном месте, а это удобно;

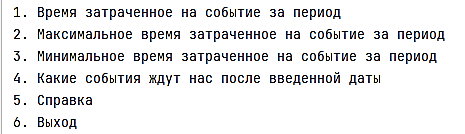
- эти запросы доступны для всех классов, т.к. модификатор static

Также вводим переменную *endReq,* причем не указываем ее равной 6, а через свойство *length.* В этом случае, если будет добавлен новый запрос, то их количество автоматически изменится.

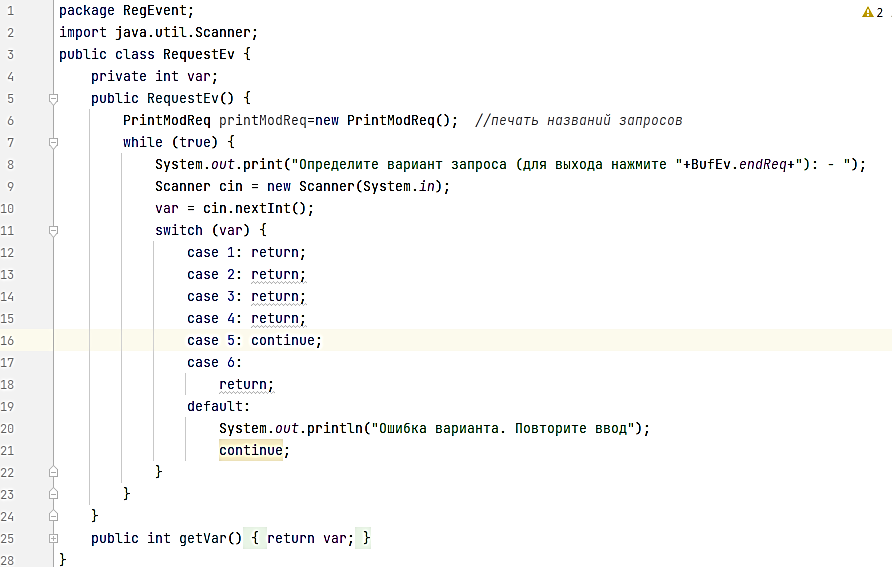
Создадим новый класс PrintModReq для вывода на экран списка названий запросов.



В цикле выдается список возможных запросов из класса BufEv в виде:



Далее откроем новый класс для ввода и контроля номера запроса:



private int var;

Локальная переменная *var*, которая будет содержать номер запроса. Т.к. она локальная, то должен быть геттер для чтения этого номера.

public int getVar() { return var;}

Метод (геттер) для чтения номера запроса.

public RequestEv() {

Конструктор класса

PrintModReq printModReq=new PrintModReq(); *//печать названий запросов*

Создание объекта для выдачи на экран наименований запросов. Рассмотрен выше.

while (true) {

Бесконечный цикл для ввода номера запроса.

System.*out*.print("Определите вариант запроса (для выхода нажмите "+BufEv.*endReq*+"): - ");

Выдача приглашения для ввода номера запроса.

Scanner cin = new Scanner(System.*in*);

Потоковый ввод данных с клавиатуры.

var = cin.nextInt();

Чтение из потока номера запроса в виде целого числа.

switch (var) {

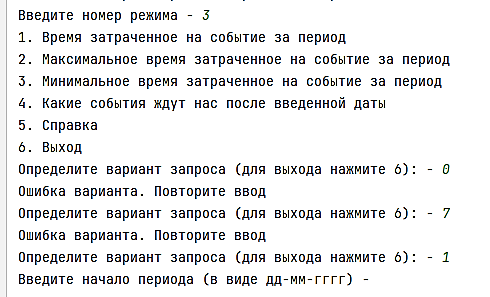
Анализ номера запроса.

case 1: return;  
case 2: return;  
case 3: return;  
case 4: return;  
case 5: continue;  
case 6: return;

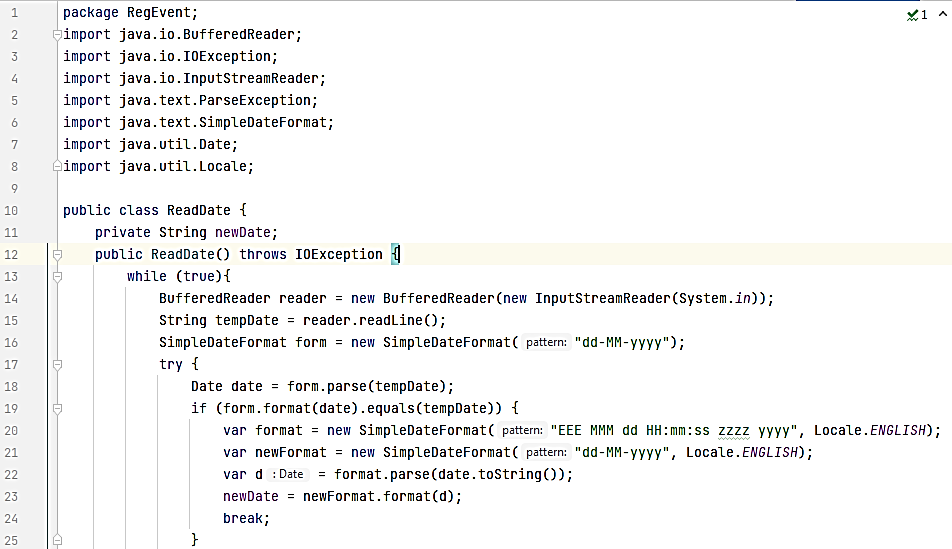
Для запросов с 1 по 4 просто возврат номера запроса. Для режима 5 (справка) – выдача справки и продолжение ввода номера запроса.

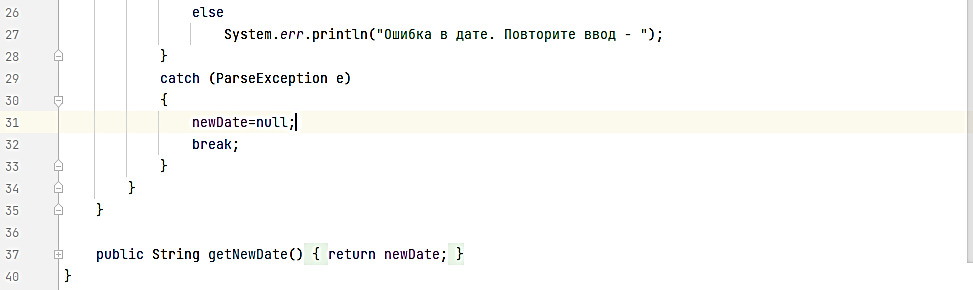
else {  
 System.*out*.println("Ошибка варианта. Повторите ввод");  
 continue;

Скриншот работы этой части программы:



Далее реализуем программу для выполнения запроса по его номеру. Но кроме номера запроса требутся начало периода, конец периода и название события. Название события уже ранее реализовано (ReadNameEv). Поэтому организуем ввод периода. Начало и конец периода это даты. Это может выполнить один класс ReadDate





Выясняется, что работа с датой не совсем тривиальная задача. В некоторых языках есть отдельный тип «date», но в языке Java для этого предлагаются ряд библиотек, которые отличаются друг от друга. Каждый программист должен выбрать свои приемы для работы с датой. Разберем код:

private String newDate;

Локальная переменная для введенной даты в строковом выражении.

public String getNewDate() {return newDate;}

Геттер для чтения введенной даты

public ReadDate() throws IOException {

Конструктор класса с возможным исключением при вводе даты.

while (true){

Бесконечный цикл для ввода даты.

BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));

Потоковый ввод даты через буфер. Пишут, что так лучше.

String tempDate = reader.readLine();

Чтение из потока буфера даты в строковом выражении. Но она вводится в неудобном виде: "EEE MMM dd HH:mm:ss zzzz yyyy" где даже секунды указаны, поэтому выполним преобразование для привычного вида.

SimpleDateFormat form = new SimpleDateFormat("dd-MM-yyyy");

Указываем формат привычного вида. Это потребуется для контроля правильности ввода даты.

try {

Начало блока возможного исключения

Date date = form.parse(tempDate);

Преобразование parse введенной даты в формат *Date*. Это тоже требуется для контроля.

if (form.format(date).equals(tempDate)) {

Это сам контроль, который выполняется «хитрым» способом. Преобразованная дата преобразуется в обратном порядке form.format(date) и результат сравнивается с исходной ).equals(tempDate). Если контроль не прошел, то выдается сообщение об ошибке:

else  
 System.*err*.println("Ошибка в дате. Повторите ввод - ");

Теперь нам требуется получить формат как в нашем файле. Для этого объявляем два формата: какой есть и какой надо:

var **format** = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzzz yyyy", Locale.*ENGLISH*);  
var **newFormat** = new SimpleDateFormat("dd-MM-yyyy", Locale.*ENGLISH*);

var d = format.parse(date.toString());

Преобразуем старый формат в промежуточный тип *Date*

newDate = newFormat.format(d);

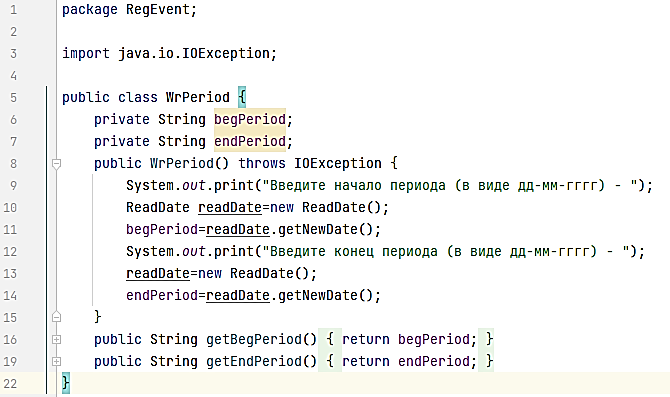
Преобразование из промежуточного типа в обычный формат в строковом виде.

Достаточно непростое преобразование и контроль, но проще пока сделать не получилось. Для большей ясности можно воспользоваться отладчиком, где на каждом шаге видно, как изменяется формат.

catch (ParseException e)  
{  
 newDate=null;  
 break;  
}

При возникновении исключения при вводе даты (в несовсем ясном случае) дату просто обнуляем и завершаем работу.

Класс для ввода даты реализован, следовательно можно разработать класс для ввода периода:



private String begPeriod;  
private String endPeriod;

Локальные переменные для начала и конца периода.

public String getBegPeriod() {return begPeriod;}  
public String getEndPeriod() {return endPeriod;}

Геттеры для чтения периода в строковом виде.

public WrPeriod() throws IOException {

Конструктор класса с возможным исключением.

System.*out*.print("Введите начало периода (в виде дд-мм-гггг) - ");

Приглашение для ввода начала периода.

ReadDate readDate=new ReadDate();  
begPeriod=readDate.getNewDate();

Ввод начала периода через класс ReadDate

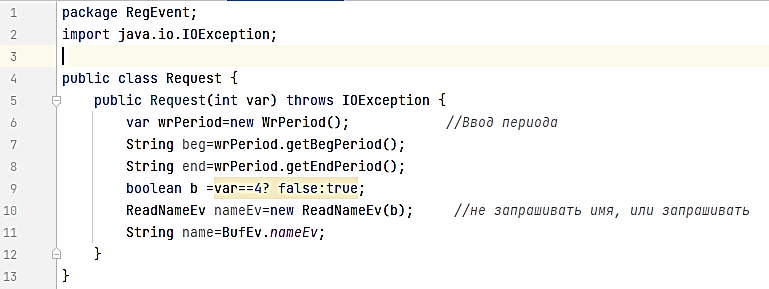
System.*out*.print("Введите конец периода (в виде дд-мм-гггг) - ");

Приглашение для ввода конца периода.

readDate=new ReadDate();  
endPeriod=readDate.getNewDate();

Ввод конца периода через класс ReadDate

Завершим эту часть программы работы с запросами, вводом исходных данных для запроса: номер запроса, период и имя события. Все это реализовано в классе Request



public Request(int var) throws IOException {

Конструктор класса с возможным исключением, которое не будем учитывать.

var wrPeriod=new WrPeriod();

Ввод периода

String beg=wrPeriod.getBegPeriod();  
String end=wrPeriod.getEndPeriod();

Через геттеры класса WrPeriod чтение начала и конца периода.

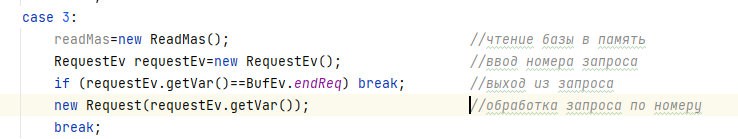
boolean b =var==4? false:true;

Определение по номеру запроса, нужно ли вводить имя. Если номер=4, то оно не требуется и будет равно *null*

ReadNameEv nameEv=new ReadNameEv(b); *//не запрашивать имя, или запрашивать*String name=BufEv.*nameEv*;

Ввод и чтение имени события. Для запроса 4 имя отсутствует.

На этом первый этап реализации запросов завершен. Осталось добавить код в класс *Main*:



readMas=new ReadMas(); *//чтение базы в память*

На всякий случай читаем данные из файла в массив

RequestEv requestEv=new RequestEv(); *//ввод номера запроса*

Организуем ввод номера запроса.

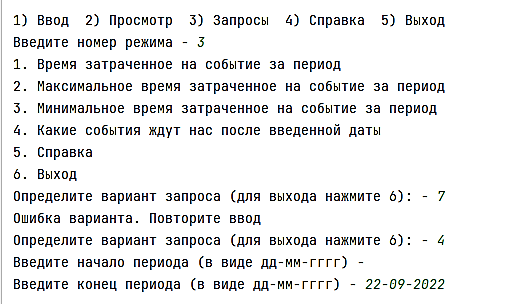
if (requestEv.getVar()==BufEv.*endReq*) break;

Если вместо запроса ввели конец, то заканчиваем.

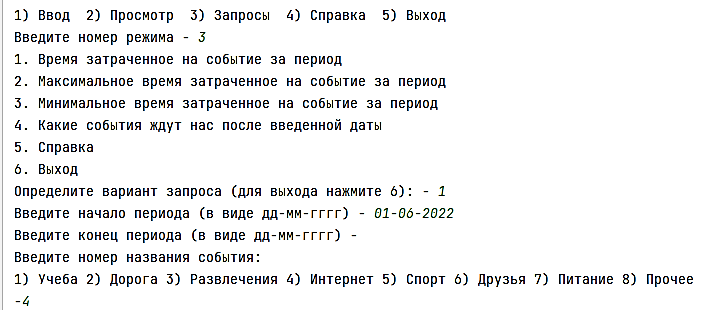
new Request(requestEv.getVar()); *//обработка запроса по номеру*

Обработка запроса. Пока это ввод периода и имени события.

Пример 1 работы программы:



Пример 2 работы программы:



Конец лекции