Назначение лекции

Знакомство с особенностями ввода даты по заданному формату.

Новые слова

**Исключение** – самостоятельная обработка аварийной ситуации в программе, для того что бы приложение аварийно не завершилось. Для этого место в программе, где может произойти авария отмечается блоком и обработчиком события. Этот обработчик выполняется в случае аварии.

Что было сделано

В предыдущех 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 определения состояния события. В программе реализовано два состояния: «было» и «будет». Для состояния «будет» понадобится дата этого события.

В этой лекции выполняется разбор кода в части ввода даты по заданому формату.

В отличии от многих языков программирования в Java отсутствует стандартный тип «дата». Но предлагается множество стандартных библиотек с классами по работе с датами.

Обычно эти классы позволяют выполнить ввод даты в любом требуемом формате и ряд методов преобразования в строку, число или в другой формат. С одной стороны это усложняет простейшие действия с датой, но с другой стороны дает множество вариантов для реализации.

По причине ограниченности времени не рассматриваются все библиотеки, а ограничиваемся самыми необходимыми.

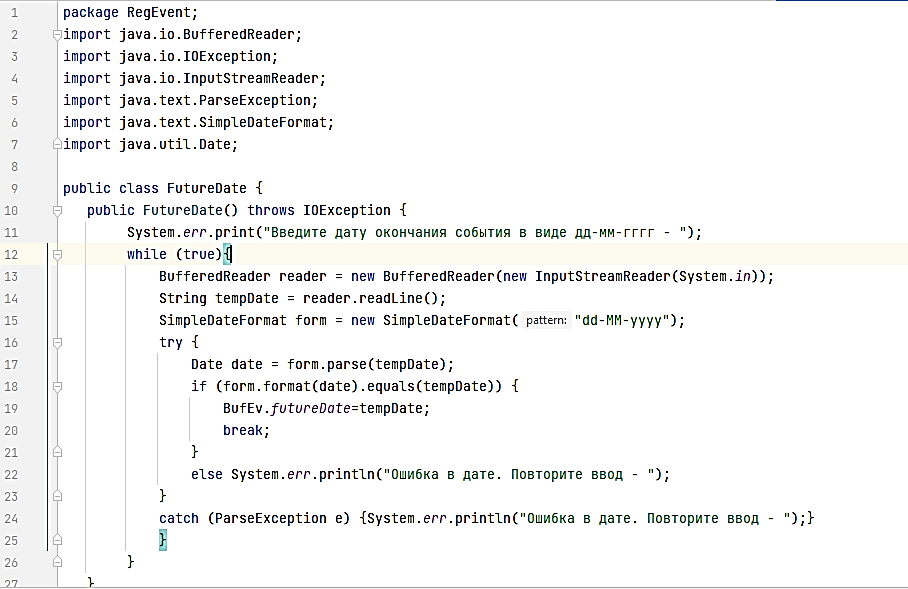
Необходимо также помнить, что некоторые библиотеки для работы с датами устарели, поэтому при реальной разработке потребуется это учесть. Хотя все библиотеки функциональны. В сети Интернет достаточно много информации на эту тему.

Итак, в процессе работы нашего приложения *MyEvent* реализуется режим «Ввод» для ввода данных о событии.

Если состояние события введено как «Будет», то требуется ввод даты, когда это событие наступит. Для состояния «Было» такой даты вводит не надо и она будет равна текущей.

Продолжение

Реализуем этот алгоритм. Для этого потребуется новый класс «будущая дата» FutureDate



Для работы констуктора этого класса потребуются заказать библиотеки:

import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Date;

Но их не требуется помнить, т.к. они импортируются автоматически. Далее будут комментарии для каждой библиотеки.

public FutureDate() throws IOException {

Этот конструктор в котором предполагается обработка исключений при вводе-выводе IOException. Исключение – это аварийная ситуация, которую можно перехватить и обработать по своему. При этом программа аварийно не завершается.

System.*err*.print("Введите дату окончания события в виде дд-мм-гггг - ");

Приглашение на ввод даты в конкретном формате.

while (true){

«Бесконечный» цикл для ввода даты. Завершается командой *break*

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

Ввод данных в буфер reader любых данных. Часто используется при вводе даты. При таком вводе требуется стандартный класс java.io.BufferedReader и java.io.InputStreamReader . Дата при этом представлена в служебном виде и использовать ее затруднительно.

String tempDate = reader.readLine();

Чтение из буфера данных и запись в строку tempDate Дата при этом будет записана как строка.

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

Определение формата даты. Он понадобится для контроля правильности введенной даты. Для формата требуется библиотека java.text.SimpleDateFormat

try {

Начало блока, где может произойти аварийное завершение программы (исключение).

Date date = form.parse(tempDate);

Переобразуем введенную дату в универсальный формат *date*. Это нам тоже понадобится для контроля данных. Для преобразования требуется библиотека java.util.Date

Далее требуется проверить правильность ввода даты. К сожалению из-за того что форматы даты в разных странах могут отличаться, простой проверки даты не найдено.

Воспользуемся некой “хитростью”, суть которой следующая: преобразуем дату во внутренний вид, а затем выполним обратное преобразование. Сравним дату с первоначальной и если они совпадут, то дата введена верно. Но даже в этом случае не все варианты можно охватить. Одним словом, контроль даты – в реальности, не совсем простой случай.

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

Это та самая «хитрая» проверка. Выполняется обратное преобразование form.format(date) и сравнивается (equals) с исходной tempDate

BufEv.*futureDate*=tempDate;  
break;

Если проверка прошла, то записываем дату, как строку в переменную *futureDate* из класса *BufEv* и цикл while (true){ завершен

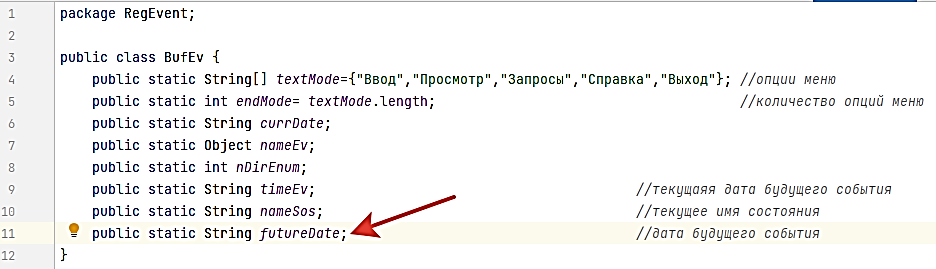
else System.*err*.println("Ошибка в дате. Повторите ввод - ");

Если «хитрая» проверка не прошла, то выдаем сообщение и цикл повторяется

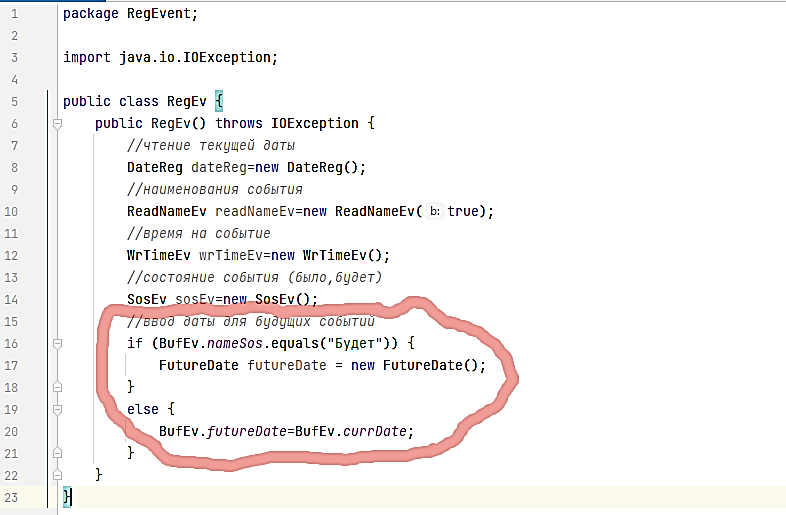
catch (ParseException e) {System.*err*.println("Ошибка в дате. Повторите ввод - ");}

Вариант, когда в блоке try { сама Java нашла ошибку в дате, выполняется эта печать и цикл также повторяется.

Далее потребуется дополнить класс *BufEv*:



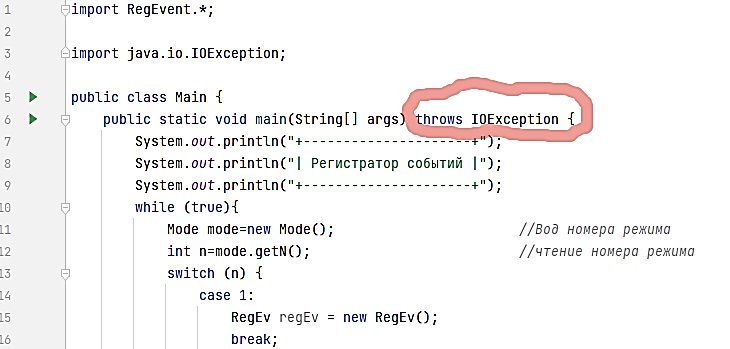
Также надо учесть, что дата события необходима лишь для состояния «Будет». Для состояния «Было» вводить дату смысла нет, но что бы она не пустовала сделаем ее равной текущей дате. Введем изменения в класс *RegEv*:



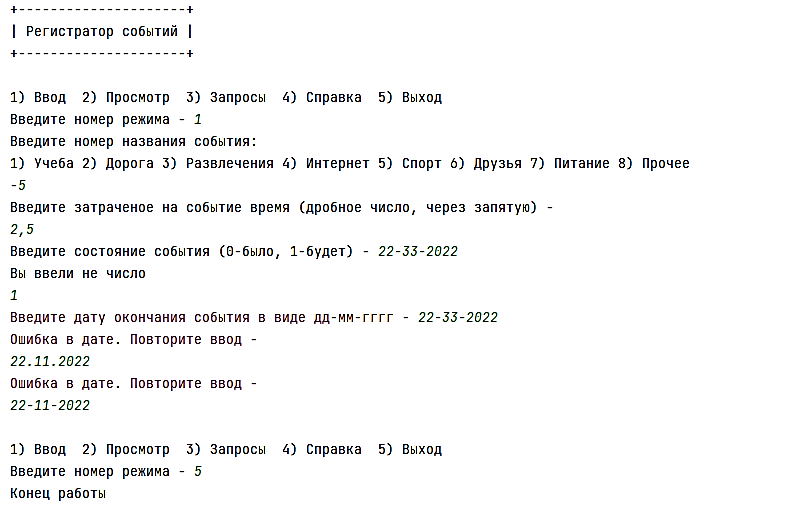
Очевидно, что класс *FutureEv* используется лишь для состояния «Будет».

Также следует обратить внимание, что в конструкторе пришлось добавить обработку исключения throws IOException т.к. она присутствует в классе *FutureEv*

Кроме того обработка исключения для даты потянется и для класса *Main*



Приведем скриншот тестового прогона приложения:



Конец лекции