Знакомство с наследованием классов при организации справки системы MyEvent.

Новые слова

**Наследование –** один из принципов объектно-ориентированного программирования (ООП), который дает возможность создавать новый класс, дочерний к родительскому классу. При этом сохраняются все свойства и методы родительского класса, но при этом можно вводить новые свойства и методы, а также заменять родительские.

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.

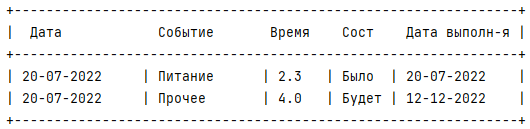
Рассмотрели реализацию режима ввода в части ввода количества времени на событие (вещественное число) и состояния события. В программе реализовано два состояния (статус): «было» и «будет». Для состояния «будет» понадобится дата этого события.

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:



Реализован механизм запросов к нашим данным.

Далее организуем небольшую справочную систему.

Продолжение

Любая современная программная система располагает справочной системой. Существуют готовые средства для создания таких систем, но мы организуем простую справку для демонстрации принципа наследования классов в Java.

Наследование – очень важный принцип ООП. Он дает возможность существенно сократить программный код, а также упростить структуру системы.

Наследование тесно связано с понятием абстрогирования. Существую абстрактные классы, которые невозможно использовать как обычно, а можно только объявлять наследников.

Абстрактный класс или любой другой родительский класс должен содержать самые общие свойства, а классы наследники уже уточняют эти свойства.

Например, для всего живого на Земле, а может и во вселенной, можно выделить абстрактный класс «Живые». У этого класса можно определить всеобщие свойства: рождаются, дышат, растут и развиваются, реагируют на изменение окружающей среды, питаются, размножаются, стареют, умирают.

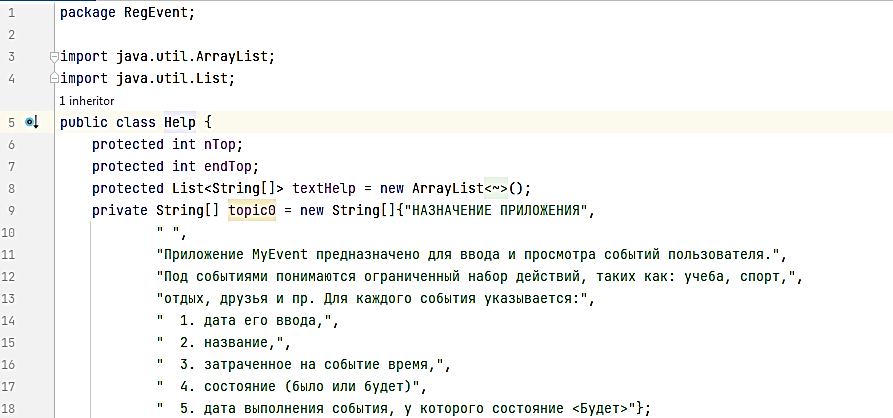
Ну а к наследникам этого класса можно отнести:

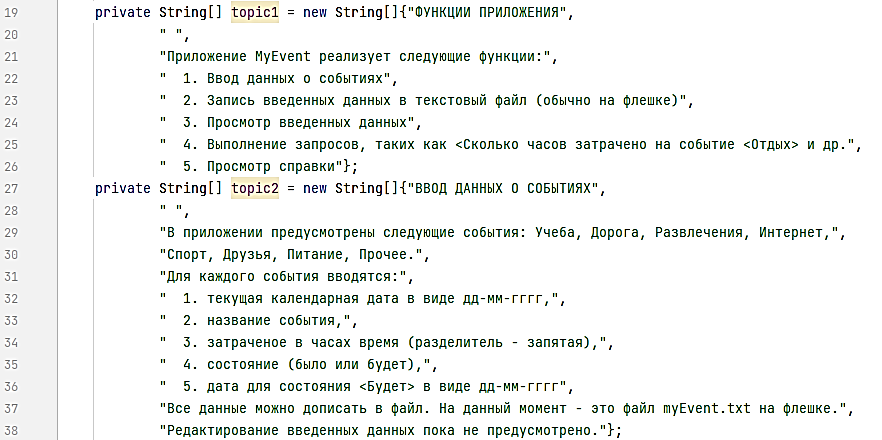
* человек,
* животные,
* птицы,
* рыбы,
* насекомые,
* растения,
* микроорганизмы (бактерии, вирусы, микробы)
* грибы (порой их относят к микроорганизмам, но мы не считаем, что грибы правильно относить к ним).

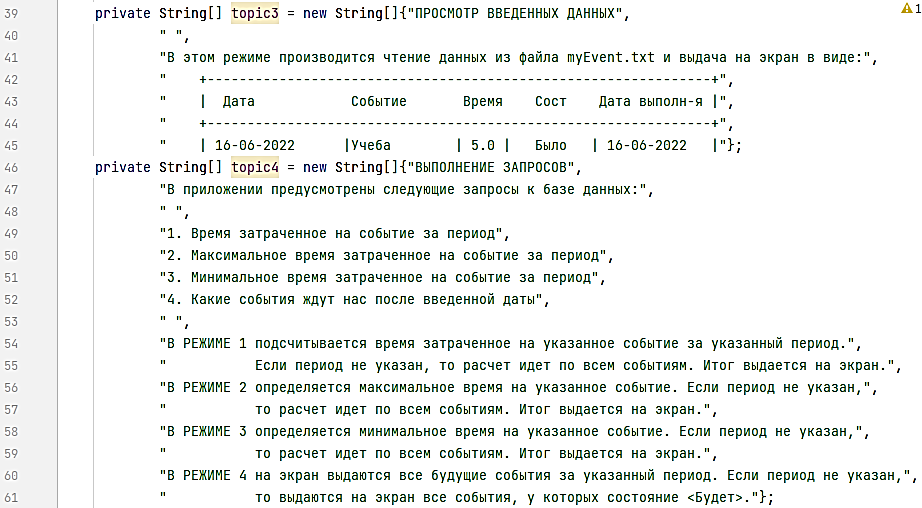
Дело программиста выделить абстрактные или родительские классы в своей системе.

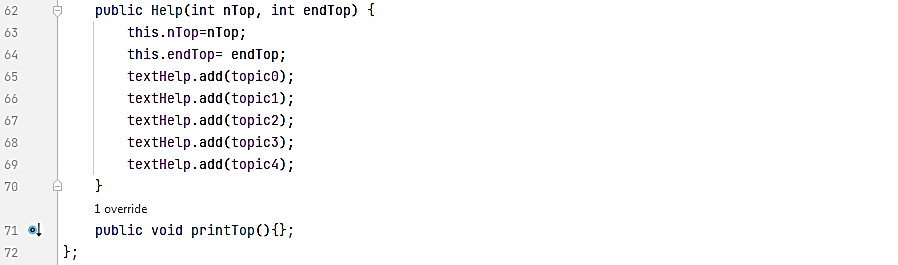
Выделим класс *Help*. Это не обстрактный класс, но в Java любой класс может быть родительским.

В этом классе определим текст справки, разделенный на топики (что бы можно было выдавать справку по топикам) и еще некоторые свойства:









Рассмотрим подробно код.

protected int nTop;  
protected int endTop;

Начальный и конечный топик справки. Модификатор protected делает эти переменные доступными в дочерних классах.

protected List<String[]> textHelp = new ArrayList<String[]>();

Объявление структуры textHelp , куда перепишем топики справки, что бы удобно было их использовать. ArrayList это структура динамического массива, а List это интерфейс для ArrayList. Этот интерфейс имеет удобные методы обращения к данным, которых нет в ArrayList

Сами топики организованы как обычные массивы строк. Каждый топик – отдельный массив.

public Help(int nTop, int endTop) {

Конструктор нашего класса *Help* с двумя параметрами: начальный и конечный топик.

this.nTop=nTop;  
this.endTop= endTop;

Переписываем эти параметры во внутренние переменные. Т.к. они одноименные используем *this*

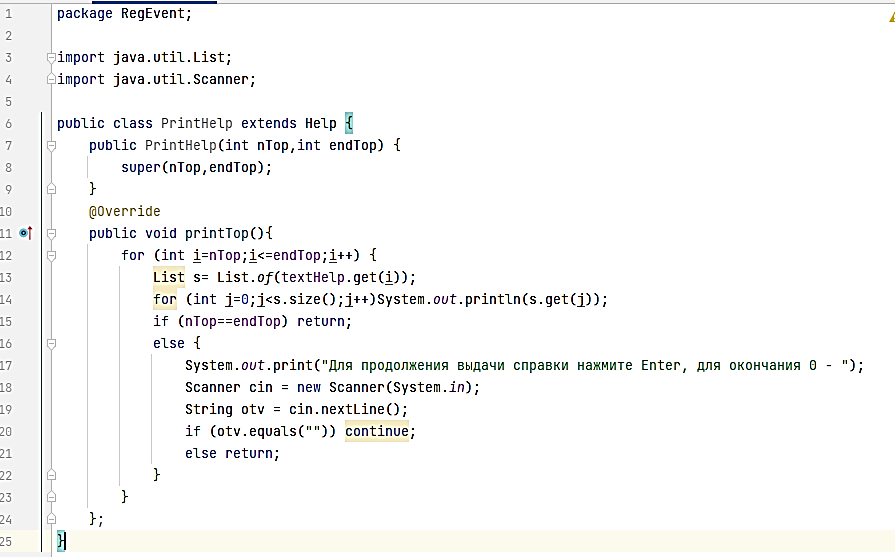
textHelp.add(topic0);  
textHelp.add(topic1);  
textHelp.add(topic2);  
textHelp.add(topic3);  
textHelp.add(topic4);

Запись в структуру textHelp всех топиков.

public void printTop(){};

Пустой метод, который будет перегружен в дочернем классе, потому что заранее неизвестно какой топик печатать.

Откроем новый класс *PrintHelp*, наследник класса *Help:*



public class PrintHelp extends Help {

Указание (extends) что PrintHelp является наследником Help

public PrintHelp(int nTop,int endTop) {

Конструктор класса PrintHelp

super(nTop,endTop);

Указание, что используется конструктор родительского класса.

@Override

Указание что метод заменяет (перегружает) родительский.

public void printTop(){

Название метода

for (int i=nTop;i<=endTop;i++) {

Цикл от начального топика до последнего.

List s= List.*of*(textHelp.get(i));

Выборка i-го топика в структуру s

for (int j=0;j<s.size();j++)System.*out*.println(s.get(j));

Выдача j-ой строки из топика s

if (nTop==endTop) return;

Если указан лишь один топик для печати, то завершаем работу справки.

else {

Если топиков больше, чем один

**System.*out*.print("Для продолжения выдачи справки нажмите Enter, для окончания 0 - ");**

Приглашение для продолжения выдачи справки

Scanner cin = new Scanner(System.*in*);  
String otv = cin.nextLine();

Ввод ответа

if (otv.equals("")) continue;

Если ввели *Enter* (можно это сделать и так), то продолжим выдачу топиков.

else return;

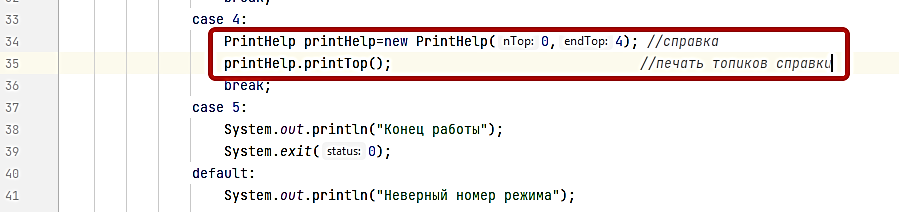
Иначе завершаем выдачу.

Далее вспомним, что справка выдается в двух местах:

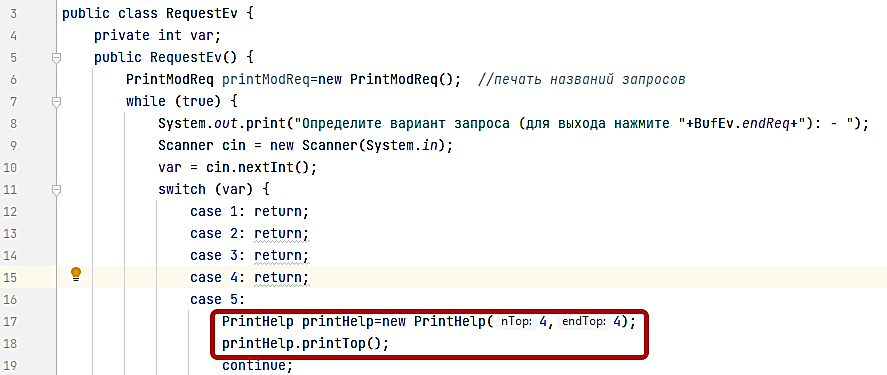
1. Основное меню
2. Меню запросов

В первом случае могут потребоваться все топики, а во втором, лишь один.

Изменим код класса *Main* без пояснения*:*



Изменим в меню запросов:



Это изменение задает выдачу 4-го топика и работа по выполнению запроса продолжается.

Конец лекции