

# #1

# прогаем на мобилу Midlet Pascal



# Intro

Здравствуй, уважаемый читатель, перед тобой первый выпуск мануала, который послужит тебе хорошим путеводителем в программировании на MidletPascal: тебе предстоит знакомство с самой средой MidletPascal и ее особенностями – от простых примеров (арифметических действий, работы со строками, с кнопками, формами и т.д.) до создания полноценных и красивых игр. В этом номере мы постарались ответить на многие вопросы, терзающие начинающего программиста и максимально близко познакомить с особенностями MidletPascal.

И поэтому собрали материалы, которые вызовут у новичка живой интерес к этой среде и программированию в целом.



# Content

<b>1 Знакомьтесь, MidletPascal</b> .....	<b>03</b>
Введение	
<b>2 Меню или пишем шпоры на экзамен</b> .....	<b>08</b>
Создание менюшек	
<b>3 Работа с ресурсами: подключение и чтение</b> .....	<b>10</b>
Как правильно работать с ресурсами	
<b>4 Хранилище записей</b> .....	<b>12</b>
Сохранение информации в мобильном телефоне	
<b>5 Музыка в твоей мобиле</b> .....	<b>14</b>
Да, зазвучит песня моя!	
<b>6 Анимация в Midlet – приложениях</b> .....	<b>16</b>
Создание анимации	
<b>7 Игростройство: мобильные игры</b> .....	<b>20</b>
Создание простейшей игры	
<b>8 Загапанный дисплей</b> .....	<b>26</b>
Работаем с сенсорным экраном	
<b>9 Шифруем SMS</b> .....	<b>29</b>
Спрячь свои сообщения от чужого глаза	
<b>10 Танчики (Part 1)</b> .....	<b>34</b>
Создание игрового поля и танка	
<b>11 Танчики (Part 2)</b> .....	<b>39</b>
Заряжай снаряд или учим танк стрелять	
<b>12 Получите, распишитесь</b> .....	<b>43</b>
Исследование файлов компиляции	

# Знакомьтесь, MidletPascal

Мобильные телефоны давно перестали быть роскошью, и теперь такая игрушка лежит в кармане почти у каждого. С каждым днем приложения для мобильных устройств, приобретают все большую популярность. В связи с таким бурным развитием мобильных технологий большая часть разработчиков переметнула на рынок мобильных устройств. Уже создано достаточно много различных мобильных игр, программ, слайд-шоу, книг и многое другое, что без проблем можно загрузить на свой телефон. Но что делать если не получается найти необходимое приложение подходящее под определенный телефон, или есть необходимость в какой-нибудь уникальной программе? Правильно, написать самому!

Мобильные приложения пишутся на языке Java(J2ME). Java - интерпретируемый язык программирования, то есть программы на нём требуют дополнительного приложения - интерпретатора. Почти на каждом телефоне установлена собственная виртуальная Java-машина, которая переводит код Java-программы в понятные данному телефону инструкции. Поэтому одни и те же Java-программы удаётся запускать как на современных Windows Mobile коммуникаторах, так и на простеньких телефонах. Но что поделывать, если необходимо написать простое приложение, а выучить и использовать Яву нет желания или возможности? Знакомьтесь - MidletPascal! Уверен, что старый добрый Паскаль знаком многим еще со школы и на много роднее чем Java. Как и любое интересное, новаторское и полезное программное обеспечение MidletPascal заработал себе награды, представленные с права.

Скачать MidletPascal можно с официального сайта [www.midletpascal.com](http://www.midletpascal.com). Чем же так примечателен этот продукт, мы рассмотрим далее.

И так, что же это за зверь такой? MidletPascal – это язык программирования предназначенный для разработки мобильных приложений (мидлетов). После написания исходного кода на Паскале, компилятор Midlet-

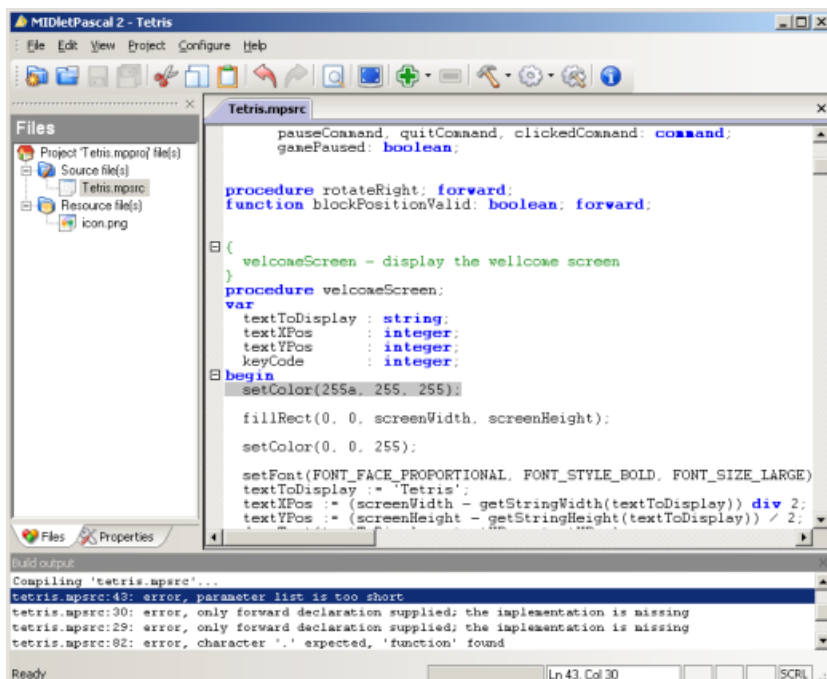
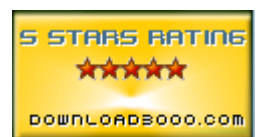


рис.1 среда разработки MidletPascal

Pascal транслирует его в код Java micro edition (J2ME). Программы, написанные на MidletPascal, могут исполняться любым мобильным телефоном с поддержкой платформы MIDP 1.0 и CLDC 1.0. MidletPascal содержит интуитивно понятную, и очень легкую в освоении среду разработки (IDE) для ОС Windows (98/2000/XP).



Если вы работали с Delphi или Kylix, то освоиться вам будет еще проще, так как сразу заметно сходство. Среда разработки содержит встроенный компилятор, инспектор кода Java и обеспечивает построение архива JAR. JAR файл — это Java архив (сокращение от англ. Java ARchive). Представляет собой обычный ZIP-архив, в котором содержится часть программы на языке Java. После компиляции создается мидлет очень маленького размера и эффективного времени исполнения. Это достигается за счет того, что MidletPascal создает непосредственно низкоуровневый Java-код, а не создает свой промежуточный код и не компонует его в архив JAR вместе с интерпретатором, как это делают аналогичные средства. Очень полезной вещью является встроенный редактор графических ресурсов. Не будем же мы оставлять иконку нашего мидлета стандартной, но найти подходящую не всегда получается, тут как раз и приходит на помощь этот редактор. К тому же будет очень красиво смотреться главное меню программы, где каждому пункту соответствует созданное своими руками определенное изображение.

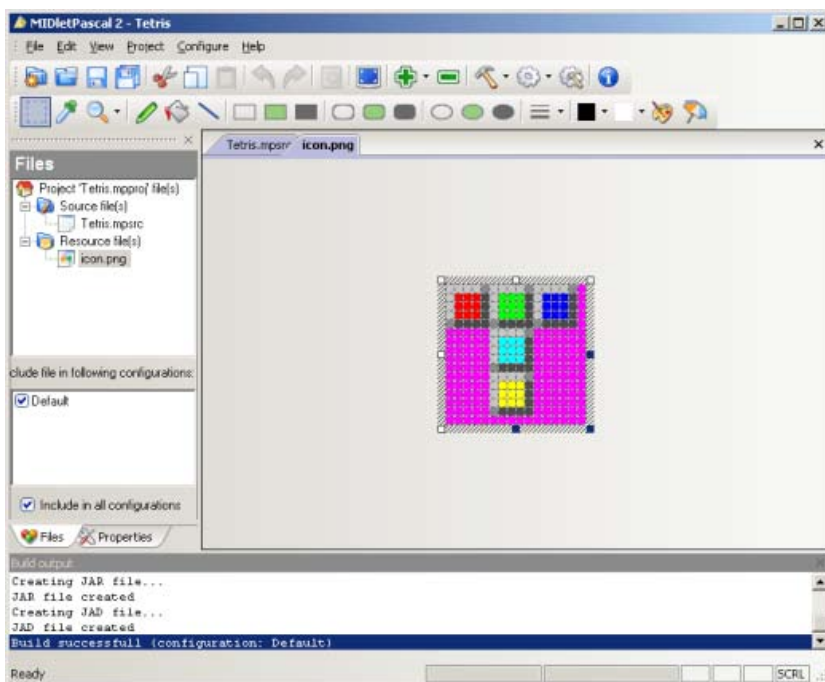


рис.2 встроенный редактор графических изображений

Приступим к более детальному рассмотрению возможностей MidletPascal. Начнем с того, что структура программы почти ни чем не отличается от программы написанной на обычном Паскале. Объявление констант, типов, переменных, процедур и функций происходит все так же, после соответствующих ключевых слов.

## code

```
program programName;
```

```
const
```

Объявления констант;

```
type
```

Объявления типов;

```
var
```

Объявления переменных;

Объявления процедур и функций;

```
begin
```

Операторы;

```
end.
```

## /code

При объявлении процедур и функций следует помнить, что они не могут быть вложенными, т.е. одна процедура или функция не может быть объявлена внутри другой процедуры или функции. При работе с функциями позволительна рекурсия.

**code**

```
program recursionSample;
var
  factorielOfFive: integer;

function factoriel(n: integer): integer;
begin
  if n = 1 then
    factoriel := 1;
  else
    factoriel := n * factoriel(n-1);
  end;

begin
  factorielOfFive := factoriel(5);
end.
```

**/code**

MidletPascal позволяет использовать опережающие объявления (forward declarations). Для этого, после объявления процедуры необходимо поставить ключевое слово forward.

**code**

```
procedure b(y:integer); forward; { опережающее объявление показывает компилятору,
                                что процедура 'b' будет описана где-то в коде программы }

procedure a (x: integer);
begin
  ...
  b(x); //хоть процедура b объявлена после процедуры a, ошибка не произойдет т.к. процедура
        // b объявлена выше как опережающая
  ...
end;

procedure b(y: integer);
begin
  ...
  a(y);
  ...
end;

begin
  a(5);
end.
```

**/code**

Операторы MidletPascal могут быть следующими:

- цикл for
- цикл while
- цикл repeat .. until

- оператор break
- оператор if .. then .. else
- оператор присвоения :=
- вызов процедуры или функции

Их использование ни чем не отличается от использования в обычном Паскале. Следует отметить отсутствие оператора case, из-за чего иногда приходится идти на ухищрение и многократно использовать операторы if .. then.

Теперь разберемся с типами данных. В MidletPascal их можно разделить на две категории: простые и комплексные. К комплексным типам относятся такие как: array (для объявления массивов) и record (комбинированный тип для создания собственного типа данных, являющихся совокупностью других типов). К простым типам данных относятся уже знакомые нам: boolean, char, integer, real, string. А вот простые типы данных, о которых следует поговорить отдельно: image, command, recordStore, http, resource. Начнем по порядку.

В MidletPascal графические изображения представлены типом image. В переменную данного типа можно загружать изображения из файла-ресурса (о них чуть позже) при помощи функции loadImage.

Тип command используется для обеспечения «кнопочного» функционала. Мидлеты J2ME не имеют таких кнопок, как приложения Windows. Вместо этого пользователь может воспользоваться так называемыми «командами».

При помощи переменной типа recordStore представляется возможность использовать хранилище телефона, которое можно ассоциировать с файлами на компьютере. Приложения могут сохранять некие данные в хранилище и впоследствии получать обратно сохранённые данные. Как и файлы, хранилища идентифицируются по имени. Их нельзя группировать внутри каталогов и каждый установленный мидлет имеет доступ только к своим собственным хранилищам. В отличие от файлов, где данные сохраняются последовательно, хранилище больше напоминает массив, где каждая запись имеет свой собственный индекс.

MidletPascal предоставляет возможность работы с протоколом HTTP. Переменной типа http идентифицируется HTTP-соединение у которого следующий жизненный цикл:

- открытие подключения к удалённому web-серверу
- установка метода запроса http
- добавление полей-заголовков запроса (необязательно)
- добавление данных в тело запроса (только в случае, если запрос имеет тип POST)
- отправка запроса и ожидание ответа от сервера
- получение желаемых полей-заголовков из ответа
- получение данных ответа
- закрытие соединения

Создавая мидлет, у нас есть возможность вместе с классами в архиве JAR разместить различные файлы ресурсов приложения. Изначально в созданном проекте уже есть один подключенный ресурсный файл – это иконка приложения находящаяся в папке «res» (в ней и должны находиться все используемые ресурсы). Все ресурсы, которые будут использоваться в программе, должны быть помещены в ту папку и подключены к проекту. Для этого выбираем в меню Project->Import new resource и выбираем соответствующий файл. Необходимо отметить, что MidletPascal может работать с графическими изображениями расширения которых png, и аудио-файлами с расширением wave, au, MP3, MIDI. С другими графическими и аудио файлами MidletPascal работать не может.

Нам так же любезно предоставлены функции, реализующие возможность отправки SMS из своего мобильного приложения. Отправка сообщений в J2ME поддерживается не всеми устройствами. MIDletPascal пытается отправить сообщение используя Wireless Messaging

## API и Siemens API.

Может возникнуть ситуация, когда поставленную задачу будет просто не возможно решить предоставленными возможностями. MidletPascal позволяет подключать библиотеки сторонних разработчиков, написанных на Java. Подыскав библиотеку с необходимыми функциями мы вполне сможем справиться с поставленной задачей. Файл библиотеку необходимо поместить в папку Libs (находится там, куда был установлен MidletPascal) и подключить в коде в раздел uses. После этого можно смело пользоваться функциями этой библиотеки.

Используя графические возможности MidletPascal необходимо понимать различие между графическим дисплеем и формой. Дисплей (Canvas) может использоваться для отображения

графики или анимации на экране устройства. Форма же, может содержать элементы ввода (что то вроде Edit в Delphi), меню, поля для ввода текста, группы выбора, «команды» и многое другое. По умолчанию используется режим Canvas. Для отображения формы необходимо использовать функцию showForm, для отображения графического дисплея – showCanvas.

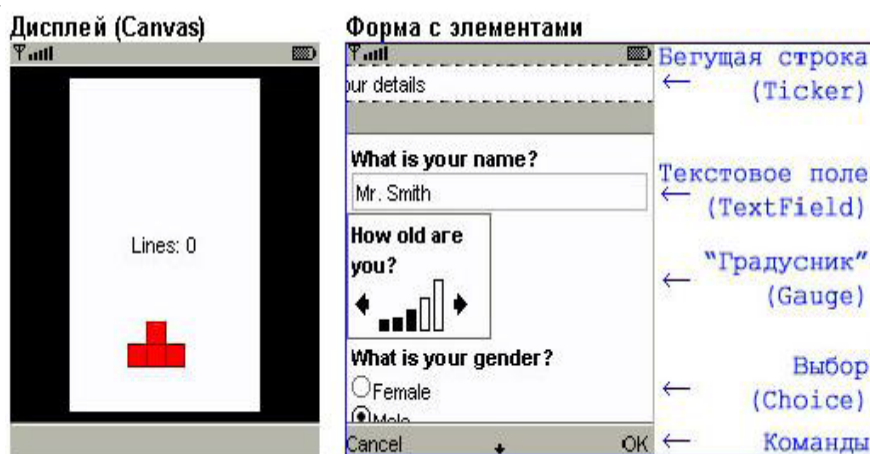


рис.3 пример отображения графического дисплея и формы

Ну вот мы потихоньку и разобрались, что же представляет из себя MidletPascal. Но вот остался один важный вопрос. Неужели для тестирования новоиспеченного мидлета придется каждый раз заливать его на телефон? Ответ - нет! Что бы так не мучиться существуют специальные эмуляторы, которые будут на компьютере эмулировать работу мобильного устройства и выполнять запущенный мидлет. Рассмотрим один из них.

Эмулятор MidpX J2ME Emulators Package при установке интегрируется в ОС Windows, проводник и браузер IE. С этого момента запуская JAD и JAR файлы, будет вызываться этот эмулятор и выполнять запущенный мидлет. MidpX эмулирует устройства с профилем MIDP 2.0 и конфигурацией CLDC 1.0. Следует отметить поддержку мультимедийных функций, кириллицы и интерфейсов (API) Nokia UI и Nokia Sound. Эмулятор позволяет сглаживать шрифты, управлять контрастностью и яркостью экрана и настраивать массу других параметров. Есть возможность замедления выполнения мидлетов.

### Схема работы MidpX:



1. Запуская мидлет, пользователь передает управление консольному конвертеру.

2. Консольный конвертер генерирует исполняемый EXE-файл эмулятора.

3. EXE-файл автоматически выполняется в системе, а на экране компьютера мы видим виртуальный телефон с запущенным Java-приложением.

Управление виртуальным телефоном сродни настоящему. Нажимать клавиши можно мышкой либо при помощи клавиатуры компьютера. В меню Options предусмотрен ползунок для регулировки громкости звука.

Надеюсь, я помог тебе разобраться в премудростях MidletPascal, который не такой уже и сложный – ведь пишем на родном Паскале.

**Автор статьи: Лопарев Роман aka Kastor**



# Меню или пишем шпоры на экзамен.

Странное название для статьи, не правда ли? Просто в этом материале я покажу как делать менюшки в приложениях для мобильного на МидлетПаскале, и попутно мы напишем шпоры на экзамены. Естественно шпоры будут в виде проги для мобилы, ведь на бумажке мелким почерком каждый и так умеет:)

Ну, давай начинать! Запускай MidletPascal и создавай новый проект, предварительно создав для него отдельную директорию. Всегда старайся держать исходники по папочкам, а то в полном хаосе вряд ли ты чего-нибудь найдёшь. Можешь переименовать прогу как твоей душе угодно, а я назвал её «menu», потому что в первую очередь - это пример, который научит тебя создавать менюшки для прог. Сразу объявляй восемь

переменных : пять из них сделай типа integer, а остальные command (надеюсь, ты знаешь что такое integer:), а про command я уже говорил в других статьях). Вот, что должно получиться :

**code**

```
var билет1, билет2, билет3, билет4, билет5 : integer;
show, clicked, exit : command;
```

**/code**

Для чего нам так много числовых переменных? Да они то и будут у нас пунктами меню! И названия соответствующие, ведь шпаргалку на экзамен пишем. Так, с переменными разобрались, идём дальше. Пиши в теле программы следующий кусок кода :

**code**

```
showMenu('Select a билет', CH_IMPLICIT);
билет1 := menuAppendString('Билет 1');
билет2 := menuAppendString('Билет 2');
билет3 := menuAppendString('Билет 3');
билет4 := menuAppendString('Билет 4');
билет5 := menuAppendString('Билет 5');
```

**/code**

Итак, первой строкой мы создаём меню. Когда на дисплее показано меню, то больше ни какие элементы формы не могут быть отображены на экране, кроме команд. Процедура имеет два параметра:

**title:string** - это название меню, т.е. текст, который будет показан перед меню. У меня это Select a билет, что в переводе на великий и могучий означает “выберете билет”.

**menuType:integer** - это тип меню. Может быть вот таким:

-**CH\_IMPLICIT** - это самое обыкновенное меню.

-**CH\_EXCLUSIVE** - при этом параметре, перед пунктом меню будет маленькая радио-кнопка, которая и осуществляет выбор.

-**CH\_MULTIPLE** - с этим параметром можно выбирать несколько пунктов меню

Нам много не надо, так что я поставил параметр menuType равным CH\_IMPLICIT. Этого будет достаточно. Но ты можешь поэкспериментировать с другими параметрами.

Дальше, я присваиваю каждой переменной результат функции menuAppendString. Эта функция вставляет в меню пункт, название которого указано в единственном параметре функции. У нас это “Билет 1”, “Билет 2” и т.д. Ну вот меню и создано. Но ничего не происходит. Надо создать команды, как это делать ты знаешь, но я всё же приведу пример:

**code**

```

show := createCommand('Показать', CM_SCREEN, 1);
addCommand(show);
repeat
delay(100);
clicked := getClickedCommand;
until clicked = show;

```

**/code**

Это всё ты уже должен знать сам, так что я объяснять не буду, а сразу пойду дальше. Теперь нам надо, чтоб при нажатии на определённый пункт меню вылазила определённая информация. Вот как можно это реализовать:

**code**

```

if menuGetSelectedIndex = билет1 then
begin
showTextVox('Билет 1', 'Выбран Билет 1 ', 2000, TF_ANY);
end;
if menuGetSelectedIndex = билет2 then
begin
showTextVox('Билет 2', 'Ты выбрал Билет 2 ', 2000, TF_ANY);
end;
if menuGetSelectedIndex = билет3 then
begin
showTextVox('Билет 3', 'Билет 3 выбран тобою', 2000, TF_ANY);
end;
if menuGetSelectedIndex = билет4 then
begin
showTextVox('Билет 4', 'Билет 4 рулит!!! ', 2000, TF_ANY);
end;
if menuGetSelectedIndex = билет5 then
begin
showTextVox('Билет 5', 'Билет 5.....!!!!!!.....!', 2000, TF_ANY);
end;

exit := createCommand('Выход', CM_SCREEN, 1);
addCommand(exit);
repeat
delay(100);
clicked := getClickedCommand;
until clicked = exit;

```

**/code**

Здесь всё очень просто : сначала идёт проверка, какой же пункт меню выбран? Проверка осуществлена с помощью функции menuGetSelectedIndex, которая возвращает имя (не название, не title) выбранного пункта меню. И если выбран первый пункт, то выводим первый текст, если выбран второй - то второй текст и т.д. Ну а потом мы создаём команду выхода, чтоб можно было нормально покинуть приложение.

Ну вот, прога готова! Тебе осталось только набить её своими билетами и смело идти на экзамен :) Да, кстати, прога не может возвратиться из выбранного билета обратно в меню. Это дело уже за тобой, сам доделывай, фантазируй. Ну а я вроде всё сказал, что хотел.

# Работа с ресурсами: подключение и чтение

Здорова, юный кодер! Сегодня мы с тобой опять будем кодить для своей мобилы. И опять будем писать на MidletPascal - чудо-юдо языке программирования для сотовых телефонов. И разговор у нас пойдёт не много не мало о ресурсах твоей проги. Почему? Постараюсь объяснить:

Во-первых, это просто необходимо, это основы работы с языком программирования.

Во-вторых, это способ засунуть в твоё приложение музыку, картинки и прочий хлам (в одной из следующих статей я покажу тебе, как сделать что-то вроде плеера, ну или просто фоновой музыки для проги).

В-третьих, это очень удобно.

Рассмотрим пример: тебе надо написать шпоры на экзамен, потому что учить лень, а сдавать надо. :) На экзамене 40 билетов. Просто так засунуть их в тело проги не хорошо. Потому что это не удобно - засоряется исходник; это долго - если в самом исходнике много лишних данных, то прога, естественно, работать будет медленнее; да и просто это не этично. Надо сделать 40 аккуратных файлов с ответами на экзаменационные вопросы, засунуть их как ресурсы в прогу и спокойно работать с ними!

По крайней мере так ты не запутаешься в своём проекте - всё будет разложено по полочкам. Где и как? В папке с проектом есть каталог под названием «res». Там по дефолту хранится картинка - иконка проекта. Очень удобно держать все ресурсы в этом каталоге, потому что он специально для этого сделан. Но просто поместить туда файл и работать с ним не прокнает. Надо сначала этот файл ресурсов подключить к нашему проекту.

Делается это очень просто: выбирай в главном меню «Project->Import resource file...». Появится обычный диалог выбора файла. Выбирай нужный и дави “Открыть”. Всё, ресурс в проекте, осталось тока написать прогу и скомпилировать её :)))

Можно конечно, сделать для своего удобства отдельные каталоги под ресурсы, но меня устраивает стандартная папка “res”. А как поступишь ты, решать тебе. Работа. Давай напишем маленький примерчик для работы с ресурсами.

Создавай текстовый файл “data.txt”, ну напиши там что-нибудь, например, “Purkin Zade RooleZZZ!!!”. Только напиши это не в одной строке, а в нескольких. Потом поймёшь почему. Подключай этот файл к проекту как ресурс, как это делать ты уже знаешь. Теперь в редакторе кода, измени сорец до вида:

```
code
```

```
program Resource; //Это название проги, пиши что хочешь
var data : resource; //наш ресурс
str : string; index :
integer;
begin
data := OpenResource('/data.txt'); //открытие ресурса
if (resourceAvailable(data)) then //проверка
begin
str := ReadLine(data); //читаем строку
CloseResource(data); //закрываем ресурс
end;
ShowForm; //создаём форму...
index := FormAddString('Text : ' + str); //выводим инфу из ресурса
Delay(10000); //задержка
end.
```

```
/code
```

Давай разбираться! Строкой `data := OpenResource('/data.txt');` я инициализирую переменную `data` и открываю ресурс. Функция открытия имеет только один параметр - имя файла-ресурса. Этой строкой `if (resourceAvailable(data)) then` я проверяю, открылся ли ресурс? Т.е. функция `ResourceAvailable(res: resource):boolean` вернёт `true` если ресурс, указанный в параметре, открыт нормально. Дальше я присваиваю переменной `str` функцию `ReadLine(res: resource):string`, которая при нормальной работе возвращает строку файла-ресурса. Потом надо закрыть файл, делается это функцией `CloseResource(res: resource)`.

Ну а дальше, тебе должно быть всё понятно и без моих слов. Пробуй! Прога работает, она выводит первую строку созданного тобой файла. Помнишь, я просил тебя создать файл из нескольких строк? Так вот, помни, что функция `ReadLine(res:resource):string` выводит только одну строку. Чтобы написать больше строк, добавь ещё одну строковую переменную, например `str2`, и после `str := ReadLine(data);` добавь `str2 := ReadLine(data);`, а после `index := FormAddString("Text :' + str);` добавь следующее: `index := FormAddString("Text_2 :' + str2);`. теперь будут читаться две строки, и так далее...

Ещё `MidletPascal` позволяет читать файл-ресурсов побайтно. Для этого надо завести переменную типа `integer` и использовать функцию `ReadByte(res: resource):integer`; Для удобства и уменьшения кода используй всевозможные циклы. Это упростит твою работу. Разбор полётов. Вот ты и научился работать с ресурсами. Ничего сложного здесь нет, они не кусаются 8). Теперь данные в твоей проге будут удобно скомпонованы, не будет свалки. Этого достаточно, чтобы переходить к следующим урокам. Надеюсь, тебе было интересно.

**Автор статьи: Васючков Андрей aka Soffrick**

# Хранилище записей.

Очень часто в приложении или игре возникает необходимость сохранить какую-либо информацию. К примеру, для игры это может быть таблица рекордов или текущий уровень, чтобы при последующем запуске игры, пользователь мог продолжить играть с прошлого достигнутого уровня; для приложения - это могут быть какие-то настройки и т.п. На компьютере в этих целях обычно пользуются записью в файл или в реестр. В мобильном телефоне нет реестра, да и доступа к файловой системе тоже во многих случаях нет, по-этому эти способы нам не подходят. Специально для этих целей в телефоне предусмотрена область памяти на встроенной флеш карте, названная Хранилищем Записей (Record Store или просто RMS).

В компьютере записываемая информация никак не защищена. Любые данные, записанные в реестр или в файл, могут быть открыты и изменены другой программой. В телефоне же это не так. Каждая игра или программа имеет своё Хранилище Записей и другая программа не может получить к нему доступ ни на чтение ни на изменение данных. По-этому вы можете смело хранить в Хранилище практически любую информацию и при этом не волноваться за её сохранность и безопасность. При удалении приложения из телефона его Хранилище Записей тоже удаляется :-)

Итак, вы усвоили, что каждое мобильное приложение имеет своё отдельное Хранилище Записей. Давайте разберемся подробнее, как же именно хранится там информация. Каждое Хранилище Записей состоит из одного или множества разделов, которые различаются по именам. То есть, к примеру, игра может иметь 2 раздела: score (в котором хранится таблица рекордов) и level (где хранится текущий уровень игры). Как вы понимаете, все названия разделов должны быть уникальными и состоять только из латинских букв или цифр. Максимальная длина имени раздела - 32 символа. Для того чтобы открыть раздел Хранилища служит функция `OpenRecordStore`. Эта функция принимает всего один аргумент - название раздела, и возвращает объект типа `RecordStore`. Если при попытке открытия раздела окажется, что раздела с таким именем нет, то ничего страшного не произойдет. Просто создастся новый пустой раздел с данным именем. Чтобы закрыть раздел Хранилища записей нужно вызвать процедуру `CloseRecordStore`. Чтобы удалить раздел в Хранилище Записей (и всю информацию в нем), используйте процедуру `DeleteRecordStore`.

Итак, движемся дальше. Каждый раздел в Хранилище Записей имеет собственно записи, которые выглядят в виде обычных текстовых строк. Каждая такая строка имеет свой номер. Это напоминает хранение строк в массиве. Тут тоже наблюдается отличие от компьютера, где программа может хранить информацию в бинарном или текстовом формате, тут же используется только текст. Вы можете узнать количество записей в вашем хранилище, воспользовавшись функцией `GetRecordStoreSize`. Для чтения записи из хранилища, используется функция `ReadRecordStoreEntry`, которая принимает 2 аргумента: объект типа `RecordStore` и номер записи, а возвращает строку текста, которая хранится под данным номером. Чтобы добавить новую запись в раздел хранилища, используют функцию `AddRecordStoreEntry`, которая возвращает номер, под которым была сохранена данная запись. Функция `GetRecordStoreNextId` возвращает следующий незанятый номер записи, который будет использован в случае добавления записи функцией `AddRecordStoreEntry`. Чтобы изменить строку, хранящуюся в разделе, используйте процедуру `ModifyRecordStoreEntry`. Ну и наконец, чтобы удалить запись из раздела нужно вызвать процедуру `DeleteRecordStoreEntry`.

Пример: Первая запись в хранилище всегда имеет номер 1. Вызов функции `AddRecordStoreEntry` добавит запись и присвоит ей номер 2. Следующей записи присвоится номер 3, и т.п. Если вы удалите запись под номером 1, и после этого добавите запись функцией `AddRecordStoreEntry`, новая запись будет сохранена под номером 4, а запись под номером 1 так и останется пустой (то есть, другими словами, дефрагментация записей в хранилище не делается). Вызов функции `GetRecordStoreNextId` вернет 5 (номер следующей незанятой записи), и `GetRecordStoreSize` вернет 3 (общее количество записей), потому что после удаления первой записи, в разделе осталось всего 3 из 4 записей.

Приведенный ниже пример кода подсчитывает и сохраняет в Хранилище Записей количество запусков программы (что может быть использовано, например, для ограничения количества запусков в демонстрационных версиях приложения):

**code**

```
var rs : RecordStore;
    countStr : string;
    countInt : integer;
    index : integer;
    nextId : integer;
begin
    // открываем раздел хранилища записей
    rs := OpenRecordStore('Count');
    nextId := GetRecordStoreNextId(rs);

    // если это первый запуск программы - добавляем запись в хранилище
    if nextId = 1 then
        index := AddRecordStoreEntry(rs, '0'); // устанавливаем счетчик в 0

    // читаем содержимое счетчика
    countStr := ReadRecordStoreEntry(rs, 1);

    // увеличиваем значение счетчика на 1
    countInt := StringToInteger(countStr) + 1;
    countStr := IntegerToString(countInt);

    // записываем измененное значение счетчика назад в хранилище
    ModifyRecordStoreEntry(rs, countStr, 1);

    // закрываем раздел хранилища записей
    CloseRecordStore(rs);

    // отображаем значение счетчика
    ShowForm;
    index := FormAddString('Ты уже запускал эту программу' + countStr + ' раз. ');
    AddCommand(CreateCommand('Выход', CM_EXIT, 1));
    repeat Delay(100) until GetClickedCommand <> EmptyCommand;
end.
```

**/code**

Если вы хотите чтобы программа не запускалась скажем после 10 запусков, просто добавьте:

```
if countInt > 9 then Halt;
```

**ПРИМЕЧАНИЕ:** MIDlet Pascal не умеет корректно сохранять записи на русском языке. Поэтому для хранения таких записей нужно воспользоваться специальной библиотекой или перекодировать строки текста в формат Win-1251, а после чтения их, перекодировать строки обратно в формат UTF-8. Также следует помнить, что операции с флеш памятью довольно медленные, что может привести к заметным задержкам в приложении при чтении/записи больших объемов информации.

# Музыка в твоей мобиле.

Итак, наступило время немного поработать с музыкой в твоей мобиле. Конечно, это не будет плеер с каталогизацией альбомов и прочими наворотами. Мы просто попробуем понять работу с музыкой на языке MidletPascal. Поняв этот материал, ты сможешь оснастить свою прогу фоновой музыкой. А если речь зайдёт за игры, то тут без музыки и звуковых эффектов никуда! Ну, какая же игра, даже мобильная, без музыки? Вот, я тоже так думаю, так что не будем терять время и начнём!

В одной из прошлых статей я рассказывал тебе про работу с ресурсами. Для понятия и реализации этого материала просто необходимо знать работу с файлами ресурсов. Если ты пропустил прошлую статью, то надо отыскать её и внимательно прочитать, иначе ничего не получится.

Для работы с музыкой и звуками в MidletPascal предусмотрено пять функций, которые мы с тобой сейчас и разберём:

```
function OpenPlayer(resource:string; mimetype:string):boolean;
```

Эта функция открывает указанный в первом параметре файл ресурса в аудио плеере. Функция вернёт false, если не сможет открыть файл. Ресурс не запустится на проигрывание, пока не будет выполнена функция 'startPlayer' (о ней немного позже). Второй параметр может принимать одно из следующих значений:

-audio/x-wav - для проигрывания wav-файлов

-audio/basic - для проигрывания au-файлов

-audio/mpeg - для проигрывания mp3-файлов

-audio/midi - для проигрывания MIDI-файлов

Внимательно следи за поддерживаемыми форматами музыки. А то не хорошо получится, если твоя прога работает с mp3, а телефон клиента поддерживает только wav. Так же разработчики Midlet Pascal предупреждают, что музыкальные функции работают только на телефонах MIDP-2.0, а на более ранних версиях телефонов приложения, использующие эти функции, приведут к краху.

```
function SetPlayerCount(loopCount:integer):boolean;
```

Эта функция устанавливает количество раз, которое музыка должна проиграться. Если значение 'loopCount' равно -1, то музыка будет проигрываться постоянно, т.е. повторяться. Функция должна быть использована после OpenPlayer, но перед startPlayer.

```
function StartPlayer:boolean;
```

Функция начинает проигрывать файл, открытый функцией OpenPlayer. Она возвратит false, если не сможет начать воспроизведение, а если всё в норме, то вернёт true.

```
function GetPlayerDuration:integer;
```

Возвращает продолжительность музыки (в миллисекундах), воспроизводимую плеером.

```
procedure StopPlayer;
```

Останавливает музыку.

Вот все инструменты, которые предлагает Midlet Pascal для работы с музыкой. Их, конечно, мало, но всё же они есть. Так что будем пользоваться тем, что есть. Вот небольшой пример, который проигрывает мелодию:

```
code
```

```
begin
```

```
  OpenPlayer('/simple.mid','audio/midi');
```

```
  SetPlayerCount(-1);
```

```
  StartPlayer;
```

```
  Delay(5000);
```

```
end.
```

```
/code
```

Здесь много не хватает, например, проверки правильной работы каждой функции. Но это уже твоё дело, моё дело - только показать тебе основы, а дальше сам разбирайся, улучшай и пробуй! Можно вывести форму, установить на ней в текстовом поле длину музыки, название файла и прочее, прочее... Пока хватит твоей фантазии.

Ну а пока разбирайся со всем прочитанным, желаю удачи! Остальное позже...

**Автор статьи: Васючков Андрей aka Soffrick**



# Анимация в Midlet – приложениях.

В этой статье, я хочу познакомить, многоуважаемого читателя, с реализацией анимации в Midlet-приложениях. Думаю, вы уже немного знакомы с тем, что такое «Midlet Pascal» и с чем его едят (см. статью «Знакомство с Midlet Pascal» - прим. ред.). Сразу оговорюсь, что особых затруднений для начинающего программиста нет, но есть некоторые нюансы, учитывать которые придется в любом случае (о них мы поговорим позднее), а если вы знакомы с Pascal, то для вас тем более, не возникнет проблем с освоением. Для начала подготовим себе необходимый инструментарий:

- Midlet Pascal, который можно взять с официального сайта разработчиков -
- <http://www.midletpascal.com>
- Эмулятор («Midp2Exe» - <http://home.kimo.com.tw/kwysshell/Project/Midp2Exe/index.html>), который непременно необходим для запуска приложений (использовать можно не только то, что я вам предложил, на сегодняшний день их предостаточно, так что выбирайте любой по вкусу).

Если с инструментарием разобрались, то идем дальше. Теперь следует разъяснить ситуацию, что же такое анимация. Так вот это набор кадров (картинок), которые с некоторым временным интервалом сменяют друг друга, таким образом можно анимировать любые объекты, но вопрос в основном стоит, в том, как реализовать данную задачу. Делается это очень просто - следует загрузить изображения в массив (получается что-то, наподобие контейнера) и оперировать ими через индексы самих картинок. Первое и последнее изображение, в массиве, должны быть одинаковыми, что необходимо для плавного перехода.



```
image_m[0]:=LoadImage('/image_1.png');  
image_m[1]:=LoadImage('/image_2.png');  
image_m[2]:=LoadImage('/image_3.png');  
image_m[3]:=LoadImage('/image_4.png');  
image_m[4]:=LoadImage('/image_5.png');  
image_m[5]:=LoadImage('/image_6.png');  
image_m[6]:=LoadImage('/image_1.png');  
image_bk:=LoadImage('/image_bk.png');
```

Следует помнить, что чем больше картинок вы загрузите, тем более плавной и четкой будет анимация. Также хочу напомнить, чтобы загрузить изображения необходимо выбрать: Project>Import resource file. Midlet предлагает примитивный редактор изображений, что-то вроде Paint'a, изображения можно потом поменять – они находятся в папке “res”, которая лежит в директории с вашим проектом. Но стоит помнить, что Midlet Pascal поддерживает только формат PNG (с альфа-каналом). Еще следует учитывать и цветопередачу, ведь не стоит забывать, что у каждого телефона она может быть разной.

Если вы захотите добавить, удалить или изменить изображения через Проводник, то делать это нужно при выключенном Midlet'е или закрытом проекте, т.к. изображения будут заменены на старые, которые уже загружены в Midlet.

В разделе “var” опишем переменные, которые в дальнейшем будем использовать.

```
var
  btn_start, btn_exit: command;
  image_m: array[0..6] of image;
  image_bk: image;
  i: integer;
```

Переменная “i” у нас будет играть роль счетчика, с помощью нее мы будем перебирать все картинки в массиве, т.е. выводить кадр соответствующий значению этой переменной. “image\_bk” послужит нам фоном, т.к. без его использования, анимации не выйдет - изображения просто будут накладываться друг на друга и получится что-то невразумительное. Переменные “btn\_start”, “btn\_exit” понадобятся нам для создания кнопок. Происходит это следующим путем:

```
btn_start:=CreateCommand('Start',CM_OK,1);
AddCommand(btn_start);
```

```
btn_exit:=CreateCommand('Exit',CM_EXIT,2);
AddCommand(btn_exit);
```

Таким образом, мы создадим две кнопки: одна запустит нашу анимацию, а при помощи другой кнопки мы закроем приложение.

Перейдем к созданию анимации, а создавать ее будем с помощью процедуры, описание которой должно быть после секции «var». Для смены изображений нужен цикл и в этом нам поможет функция:

```
Repeat
  //ваш код
Until условие выхода;
```

Эта конструкция запустит бесконечный цикл, до тех пор, пока не будет задействовано условие выхода из этого цикла. В первую очередь, необходимо определить область, которую будем использовать, (в данном случае используем всю, но следует учесть, что для разных эмуляторов, а также телефонов она будет различной!) - SetClip(0,0,200,200). Далее рисуем фон (происходит растяжение, если изображение меньше используемой области) и рисуем первое изображение (располагаем его в центре выделенной области), после этого увеличиваем счетчик. «Repaint» перерисует изображение фона и картинку, а далее небольшая пауза, это сделано, для того чтобы анимация была не слишком быстрой. В конце смотрим: если счетчик равен индексу последнего изображения, то обнуляем счетчик. Осталось проверить, нажата ли клавиша выхода.

#### code

```
procedure animate_image;
begin
  repeat
    SetClip(0,0,200,200);
    DrawImage(image_bk,0,0);
    DrawImage(image_m[i],70,75);
    repaint;
    Delay(170);
    i:=i+1;
    if i=6 then i:=0;
    get_btn:=GetClickedCommand;
    until Get_btn = btn_exit;
end;
```

#### /code

Чтобы задействовать процедуру мало просто вызвать ее, придется еще раз применить функцию повтора. Ниже приведенная конструкция необходима, для того чтобы анимация не сразу проигрывалась, а только при нажатии кнопки «Start».

```
repeat  
get_btn:=GetClickedCommand;  
if get_btn= btn_start then animate_image;  
until Get_btn = btn_exit;
```

Переписывай код в Листинге 1 (я его немного дополнил - анимированный объект будет передвигаться по экрану) и создавай изображения. Я привел пример самой простой анимации, если у тебя фантазии «полон двор», то думаю, теперь тебе не составит особого труда развить эту тему – стоит лишь добавить изображения для различных действий (прыжок, наклон, бросок и т.д.) и задать эти движения на каждую клавишу, то в итоге может получиться неплохая аркадная игрушка. Ведь создание приложений на Midlet Pascal далеко не самое сложное занятие.

## Листинг 1.

### code

```
program Midlet_anim;  
var  
  btn_start,btn_exit, Get_btn: command;  
  image_m: array[0..6] of image;  
  image_bk: image;  
  l,k:integer;  
  
procedure animate_image;  
begin  
  repeat  
    SetClip(0,0,200,200);  
    DrawImage(image_bk,0,0);  
    DrawImage(image_m[i],k,75);  
    repaint;  
    Delay(170);  
    i:=i+1;  
    k:=k+2;  
  
    if i=6 then i:=0;  
    if k>150 then k:=-20;  
    get_btn:=GetClickedCommand;  
    until Get_btn = btn_exit;  
end;  
  
begin  
  i:=0;  
  k:=-20;  
  
  image_m[0]:=LoadImage('/image_1.png');  
  image_m[1]:=LoadImage('/image_2.png');  
  image_m[2]:=LoadImage('/image_3.png');  
  image_m[3]:=LoadImage('/image_4.png');
```

```
image_m[4]:=LoadImage('/image_5.png');
image_m[5]:=LoadImage('/image_6.png');
image_m[6]:=LoadImage('/image_1.png');
image_bk:=LoadImage('/image_bk.png');

btn_start:=CreateCommand('Start',CM_OK,1);
btn_exit:=CreateCommand('Exit',CM_EXIT,2);
AddCommand(btn_start);
AddCommand(btn_exit);

repeat
get_btn:=GetClickedCommand;
if get_btn= btn_start then animate_image;
until Get_btn = btn_exit;
end.
```

**/code**

**Автор статьи: Jimmy Jonezz**

# Игростроение: мобильные игры.

Было бы неправильно с нашей стороны не упомянуть (хоть немного) о создании игр на MidletPascal'e. Причина тому, что игровая индустрия в мобильных телефонах явно находится на пике своего развития и роста и радуется многообразием/разнообразием игр, но всегда хочется создать что-то свое, пусть даже и не столь сложное, как хотелось бы.

Чтобы уместиться в формат журнала, я постараюсь кратко описать все шаги, для достижения поставленной цели – создание игры, а именно игры под названием «Крестики-Нолики». Создание нашей игры начинается с игрового поля, на котором мы будем потом ставить крестики нолики, добавим мозгов «компьютеру» и напишем несколько вспомогательных процедур и функций.

Начнем, пожалуй, с игрового поля, нарисуем решетку, состоящую из 9 клеток (по бокам я обрезал ее немного, чтобы наша решетка была не во весь экран, так будет выглядеть более эстетично). Порядок действий таков: делим экран по горизонтали пополам, берем одну часть и из нее вычитаем  $\frac{1}{4}$ , разделенной части экрана. Функция “div” позволяет брать целое число, без остатка при делении. DrawLine – вырисовывает линии, но не забудьте задать цвет для линий. Решетка получилась не совсем правильной, поэтому оптимизируйте по своим размерам экрана.

## code

```
procedure drawPlace;
var
  i,y:integer;
begin
  SetColor(0,0,0);
  //горизонтальные линии
  i:=(GetHeight div 2);
  y:=i div 4;
  DrawLine(20, i-y , GetWidth-20,i-y);
  DrawLine(20, i+(i div 2) - y, GetWidth-20, i+(i div 2)-y);

  //вертикальные линии
  i:=(GetWidth div 2);
  DrawLine(i -y, 20, i-y, GetHeight-20);
  DrawLine(i+(i div 2)-y, 20, i+(i div 2)-y, GetHeight-20);
end;
```

## /code

Опишем глобальные переменные, которые будут использоваться нами в игре, “p” – это массив, который будет содержать буквы «X» и «O» (далее я поясню для чего это сделано), поэтому он имеет символьный тип. Еще создадим три кнопки: выход, подтверждение и продолжение (в том случае если мы захотим сыграть еще).

```
var
  cmdOK, cmdContinue, cmdExit, clicked:command;
  p:array [1..9] of char;
  Ximg,Oimg:image;
```

Поле создано, а переменные описаны, пришло время, научиться рисовать крестики и нолики, на нашем поле. Сразу оговорюсь, что заполнять поле, т.е. решетку, мы будем буквами, и отображать это действие, будем рисованием изображений: крестика и нолика (потому что необходимо проверять, что поставлено в клетке или клетках, а также проверять, кто выиграл или проиграл, для этих действий мы и задействуем наш массив).

**code**

```

procedure DrawXO(c:char; pos:integer);
var
  x,x1,y,y1:integer;
  img:image;

begin
  if c='X' then img:=Ximg;
  if c='O' then img:=Oimg;
  x:=GetWidth; x1:=GetImageWidth(img)/2;
  y:=GetHeight; y1:=GetImageHeight(img)/2;
  //рисуем изображения крестика или нолика в клетках
  //каждой позиции соответствует одна клетка поля
  if pos=1 then DrawImage(img,(x div 4)-x1,(y div 4)-y1);
  if pos=2 then DrawImage(img,(x div 2)-x1,(y div 4)-y1);
  if pos=3 then DrawImage(img,(x -(x div 4)-x1),(y div 4)-y1);
  if pos=4 then DrawImage(img,(x div 4)-x1,(y div 2)-y1);
  if pos=5 then DrawImage(img,(x div 2)-x1,(y div 2)-y1);
  if pos=6 then DrawImage(img,(x -(x div 4)-x1),(y div 2)-y1);
  if pos=7 then DrawImage(img,(x div 4)-x1,(y -(y div 4)-y1));
  if pos=8 then DrawImage(img,(x div 2)-x1,(y -(y div 4)-y1));
  if pos=9 then DrawImage(img,(x -(x div 4)-x1),(y-(y div 4)-y1));
end;
//управляющая процедура
procedure DrawAll;
var
  i:integer;
begin
  for i:=1 to 9 do
    DrawXO(p[i],i);
  end;

```

**/code**

Создадим процедуру, которая пригодится для вывода сообщений: выиграл ли игрок или проиграл, а когда случилась и ничья. Особо мудрить не будем, а просто нарисуем горизонтальный прямоугольник, по середине экрана и выведем в нем соответствующую надпись. SetFont необходим для создания шрифта – зададим ему свойства: жирные большие системные буквы.

**code**

```

procedure ShowMess(s:string);
begin
  SetClip(0, 0, GetWidth, GetHeight);
  SetColor(0,0,0);
  FillRect(0,GetHeight div 2+5,GetWidth,GetStringHeight(s));

  SetColor(255,255,255);
  SetFont(FONT_FACE_SYSTEM,FONT_STYLE_BOLD,FONT_SIZE_LARGE);
  DrawText(s, (GetWidth - GetStringWidth(s))/2, GetHeight div 2);

  repaint;
end;

```

**/code**

Следующими двумя функциями мы проверим, заполнены ли все клетки и определим, кто выиграл, т.е. выведем сообщение о том, кто выиграл/проиграл или вообще сыграно вничью.

**code**

```
function CheckFull:boolean;
var
  i,all:integer;
begin
  CheckFull:=false;
  all:=0;
  for i:=1 to 9 do
    if p[i]<>' ' then all:=all+1;
    if all=9 then CheckFull:=true;
  end;

function CheckGameEnd:boolean;
begin
  CheckGameEnd:=false;

  //проверка, выиграл ли игрок
  if checkWin('X') then begin
    CheckGameEnd:=true;
    ShowMess('You WIN!');
  end else

  //проверка, выиграл ли компьютер
  if checkWin('O') then begin
    CheckGameEnd:=true;
    ShowMess('You lose...');
  end else
  //ничья?
  if CheckFull then begin
    CheckGameEnd:=true;
    ShowMess('Nobody win...!');
  end;
end;
```

**/code**

Подходим к основным частям, а именно к определению кто именно выиграл, и добавлению «мозгов» игре, пусть совсем и небольших (одному играть ведь не интересно). Ну, если с определением кто выиграл: «компьютер» или Игрок понятно, - просматриваем позиции по горизонтали, вертикали и диагонали, по всем трем клеткам, то с искусственным интеллектом посложней, а именно надо просматривать все варианты ходов, предугадать, куда будет поставлен следующий крестик/нолик, в общем, максимально усложнить игроку варианты установки крестика. Но в эти дебри нам идти не суждено, да и не надо, мы поступим проще – хаотичный ход компьютера будет сопровождаться поиском дух рядом стоящих ноликов. Так что рассчитывать на сложную игру не особо приходится. Но при желании ты можешь прислать мне свою версию игровых «мозгов», посмотрим выйдет ли это у тебя лучше?

## code

```
function checkWin(c:char):boolean;
begin
checkWin:=false;
//горизонталь
if (p[1]=c) and (p[2]=c) and (p[3]=c) then checkWin:=true;
if (p[4]=c) and (p[5]=c) and (p[6]=c) then checkWin:=true;
if (p[7]=c) and (p[8]=c) and (p[9]=c) then checkWin:=true;
//вертикаль
if (p[1]=c) and (p[4]=c) and (p[7]=c) then checkWin:=true;
if (p[2]=c) and (p[5]=c) and (p[8]=c) then checkWin:=true;
if (p[3]=c) and (p[6]=c) and (p[9]=c) then checkWin:=true;
//наскось :)
if (p[1]=c) and (p[5]=c) and (p[9]=c) then checkWin:=true;
if (p[3]=c) and (p[5]=c) and (p[7]=c) then checkWin:=true;
end;

//шаги «компьютера»
function Step(c:char;pos:integer):boolean;
begin
Step:=false;
if not ((pos<1) or (pos>9)) then
  if p[pos]=' ' then begin
    p[pos]:=c;
    Step:=true;
  end;
end;

//так создаются «мозги» для игры
procedure CompStep;
var
tmp:boolean;
begin
tmp:=false;
while not tmp do begin
randomize;

//смотри вертикальные варианты,
//например если есть «О О» то можно поставить еще один нолик
if not tmp then
if (p[1]='O') and (p[2]='O') and (p[3]=' ') then tmp:=Step('O',3) else

if (p[1]<>' ') and (p[2]<>' ') and (p[3]=' ') then tmp:=Step('O',3) else
if (p[1]<>' ') and (p[2]=' ') and (p[3]<>' ') then tmp:=Step('O',2) else
if (p[1]=' ') and (p[2]<>' ') and (p[3]<>' ') then tmp:=Step('O',1) else

if (p[4]<>' ') and (p[5]<>' ') and (p[6]=' ') then tmp:=Step('O',6) else
if (p[4]<>' ') and (p[5]=' ') and (p[6]<>' ') then tmp:=Step('O',5) else
if (p[4]=' ') and (p[5]<>' ') and (p[6]<>' ') then tmp:=Step('O',4) else

if (p[7]<>' ') and (p[8]<>' ') and (p[9]=' ') then tmp:=Step('O',9) else
if (p[7]<>' ') and (p[8]=' ') and (p[9]<>' ') then tmp:=Step('O',8) else
```



```
if (p[7]=' ') and (p[8]<>' ') and (p[9]<>' ') then tmp:=Step('O',7) else
```

```
//горизонтальные варианты
```

```
if (p[1]<>' ') and (p[4]<>' ') and (p[7]=' ') then tmp:=Step('O',7) else
```

```
if (p[1]<>' ') and (p[4]=' ') and (p[7]<>' ') then tmp:=Step('O',4) else
```

```
if (p[1]=' ') and (p[4]<>' ') and (p[7]<>' ') then tmp:=Step('O',1) else
```

```
if (p[2]<>' ') and (p[5]<>' ') and (p[8]=' ') then tmp:=Step('O',8) else
```

```
if (p[2]<>' ') and (p[5]=' ') and (p[8]<>' ') then tmp:=Step('O',5) else
```

```
if (p[2]=' ') and (p[5]<>' ') and (p[8]<>' ') then tmp:=Step('O',2) else
```

```
if (p[3]<>' ') and (p[6]<>' ') and (p[9]=' ') then tmp:=Step('O',9) else
```

```
if (p[3]<>' ') and (p[6]=' ') and (p[9]<>' ') then tmp:=Step('O',6) else
```

```
if (p[3]=' ') and (p[6]<>' ') and (p[9]<>' ') then tmp:=Step('O',3) else
```

```
//если ничего не подошло выборочно ставим нолик...
```

```
tmp:=Step('O',9-Random(9));
```

```
end;
```

```
end;
```

```
/code
```

Осталось задействовать, то, что расписано нами в начале статьи, т.е. попробуем все это хозяйство объединить – создадим игровой процесс: загрузим изображения крестика и нолика (картинки предварительно нужно загрузить в ресурсы), инициализируем конец игры, отчистим массив (в том случае если мы играли и захотим продолжить игру), следующим шагом будет рисование решетки, компьютер делает ход, а дальше выжидаем ход игрока, с последующей проверкой того, кто выиграл, чтобы определить конец игры, иначе продолжаем сначала, а именно с шага компьютера.

```
code
```

```
procedure CreateGame;
```

```
var
```

```
  i:integer;
```

```
  UserStep:boolean;
```

```
  GameEnd:boolean;
```

```
begin
```

```
  //загружаем заранее чтобы не нагружать VM
```

```
  Ximg:=LoadImage('/Ximg.png');
```

```
  Oimg:=LoadImage('/Oimg.png');
```

```
  ShowCanvas;
```

```
  RemoveCommand(cmdOK);
```

```
  AddCommand(cmdExit);
```

```
  GameEnd:=false;
```

```
  For i:=1 to 9 do
```

```
    p[i]:='';
```

```
  //пока не конец игры
```

```
  while not GameEnd do begin
```

```
    SetColor(255,255,255);
```

```
    FillRect(0,0,GetWidth, GetHeight);
```

```
    drawPlace; //рисуем решетку
```

```

CompStep; //комп сделал ход
DrawAll; //рисует крестики и нолики
delay(300);
repaint;
delay(300);

GameEnd:=CheckGameEnd;
//если не конец игры, и игрок не нажал кнопку, то просто ждем его действий
if (not GameEnd) then
while not UserStep do
while (GetKeyPressed = KE_NONE) do begin
delay(100);
//если выход..
if GetClickedCommand=cmdExit then Halt;
//каждой кнопке соответствует одна клетка
if GetKeyPressed=KE_KEY1 then i:=1;
if GetKeyPressed=KE_KEY2 then i:=2;
if GetKeyPressed=KE_KEY3 then i:=3;
if GetKeyPressed=KE_KEY4 then i:=4;
if GetKeyPressed=KE_KEY5 then i:=5;
if GetKeyPressed=KE_KEY6 then i:=6;
if GetKeyPressed=KE_KEY7 then i:=7;
if GetKeyPressed=KE_KEY8 then i:=8;
if GetKeyPressed=KE_KEY9 then i:=9;

UserStep:=Step('X',i{Random(9)}); //шагаем и рисуем крестик игрока
i:=0;
end;
UserStep:=false;
DrawAll;
repaint;
GameEnd:=CheckGameEnd; //суммируем результат, чтобы не стереть предыдущее событие
end;
//создаем кнопку продолжения игры
AddCommand(cmdContinue);
repaint;

repeat
delay(100);
clicked := GetClickedCommand;
until clicked <> EmptyCommand;
//стартуем в новую игру и не забываем удалить кнопку продолжения
RemoveCommand(cmdContinue);
if clicked=cmdContinue then CreateGame;
end;

```

**/code**

Многое мы в этой статье не затронули: использование меню, переходы по разделам меню, вывод результатов – у кого, сколько выигрывает и т.д. Все это не столь важно, ведь самое главное понять принцип, взаимодействия модулей, легкое манипулирование ими, а в остальном вы думаю, останетесь, довольны, ведь теперь можно добавить еще одну игру в свою коллекцию игр (если такая имеется).

**Автор статьи: Jimmy Jonezz, Автор кода: Che**

# Залапанный дисплей

Сейчас среди производителей телефонов стало модно добавлять функцию сенсорного экрана. Хорошо это или плохо - не мне решать. Скорее это навязывание технологии ради того чтобы хоть чем-то переплюнуть конкурентов. И если раньше сенсорный экран был диковинкой и имелся только в ограниченных количествах моделей, то теперь я не удивлюсь, если через пару лет с сенсорным экраном будут выпускаться, чуть ли не все телефоны. Да и отношение к экрану меняется. Если раньше с сенсорным экраном нужно было работать специальным перышком, то теперь приветствуется нажатие пальцами. И как это сочетается с имиджем гламурного металлического блеска? :-D Ну да ладно, я отвлекся, давайте я Вам покажу как использовать функции данного чудо-экрана из самого чудесного языка для создания мидлетов - MIDlet Pascal'я.

Сам MP с сенсорными экранами работать не умеет, но у него есть отличная особенность – это расширяемость за счёт подключения дополнительных библиотек функций.

## Что нам потребуется?

В приложении к этой статье будет файл `Lib_sensor.class`. Это дополнительная библиотека

для MIDlet Pascal. Если Вы не знаете, как добавлять такие библиотеки в MP, я кратко опишу последовательность действий:

1. Скопируйте `Lib_sensor.class` файл в папку `Libs` программы MIDlet Pascal (у меня это `C:\Program Files\MIDlet Pascal\Libs`)

2. Вот, собственно, и всё. Для того чтобы теперь воспользоваться библиотекой в Вашей программе нужно сразу за строчкой `Program {Название}` добавить строчку: `Uses sensor;` (т.е. `Lib_` писать не надо).

## В чем тестировать?

Если в Вашем телефоне пока нет сенсорного экрана это ещё не повод быстрее бежать в магазин. Тестировать сенсорный экран можно и за компьютером, используя эмулятор телефона. Опишу как включать сенсорный экран на некоторых распространенных эмуляторах:

`Microemulator`, `Sjboy` и `NHAL` (он же `Midp2exe`, он же `MidpX`) - настройка не требуется, всё уже и так включено. `Wireless Tool Kit` (он же `WTK`) для включения сенсорного экрана откройте папку с `WTK` (у меня это `C:\WTK22\`), далее переходим в подпапку `wtklib\devices\DefaultColorPhone\` и открываем в блокноте файл `DefaultColorPhone.properties`. Там нужно изменить только одну строчку, вместо `touch_screen=false` пишем `touch_screen=true` и сохраняем изменения. Для других эмуляторов пакета `WTK` изменения делаются аналогично.

Итак, приступим. Для подключения библиотеки пишем:

```
Program Sensor_test;  
Uses sensor;
```

Всего в MP при этом добавляется 10 новых команд:

- `init` - Инициализация библиотеки. Надо вызывать перед началом работы с сенсорным экраном.
- `integer pointer_state` - Возвращает текущее состояние сенсорного экрана, например: 0 - ничего не происходит; 1 - есть нажатие на экран
- `integer pointer_pressed_x` - Возвращает X координату нажатой точки
- `integer pointer_pressed_y` - Возвращает Y координату нажатой точки
- `integer pointer_dragged_x` - Возвращает X координату точки при рисовании/перетаскивании пером
- `integer pointer_dragged_y` - Возвращает Y координату точки при рисовании/перетаскивании пером

`integer pointer_released_x` - Возвращает X координату точки в которой прекратили рисование/перетаскивание пером (убрали перо)

`integer pointer_released_y` - Возвращает Y координату точки в которой прекратили рисование/перетаскивание пером (убрали перо)

`integer has_pointer_events` - Возвращает 0 или 1 если произошло какое-либо событие типа нажатия

`integer has_pointer_motion_events` - Возвращает 0 или 1 если произошло какое-либо событие типа рисования/перетаскивания

Вот пример простейшей программы, позволяющей рисовать на сенсорном экране (каляки-маляки хайтек :- ) ):

#### code

```
Program PointerTest;
Uses sensor;
Var ox, oy, px, py, state: integer;
Begin
  sensor.init; // Инициализация библиотеки
  repeat // Бесконечный цикл
    state:=pointer_state; // Читаем состояние сенсора
    if state=1 then // Если нажали, то...
      begin
        if (ox=0) and (oy=0) then
          begin
            ox:=pointer_dragged_x; oy:=pointer_dragged_y;
          end;
          // Выясняем куда именно нажали (координаты)
          px:=pointer_dragged_x; py:=pointer_dragged_y;
          DrawLine(ox,oy,px,py); // Рисуем линию
          // Запоминаем предыдущую точку для рисования следующей линии
          ox:=px; oy:=py;
        end;
        // Если ничего не нажали - сбрасываем координаты в 0
        else begin ox:=0; oy:=0;
        end;
        Repaint; Delay(20); // Отрисовка линии и небольшая пауза
      until false;
End.
```

#### /code

Другой примерчик. Давайте нарисуем на экране пару кнопок и при нажатии на них будет что-то происходить:

#### code

```
Program PointerTest2;
Uses sensor;
Var xx, yy, state, key: integer;

Procedure DrawButtons; // Вспомогательная процедура, которая рисует на экране кнопки
begin
  SetColor(255,255,255); // Белый цвет
  if key=1 then SetColor(255,0,0); // Если была нажата кнопка 1 - ставим красный цвет
  if key=2 then SetColor(0,255,0); // Если кнопка 2 - зеленый
```

```
FillRect(0,0,GetWidth,GetHeight); // Закрашиваем весь экран выбранным цветом
SetColor(0,0,0); // Ставим черный цвет
DrawRect(10, 10, 100, 25); DrawText('Кнопка 1', 15, 15); // Рисуем первую кнопку
DrawRect(10, 40, 100, 25); DrawText('Кнопка 2', 15, 45); // Рисуем вторую кнопку
DrawRect(10, 70, 100, 25); DrawText('Выход', 15, 75); // Рисуем третью
Repaint; // Окончательный вывод всего нарисованного на дисплей
end;

Begin // Начало нашей программы
sensor.init; // Инициализация библиотеки
DrawButtons; // Рисуем кнопки на экране (см. процедуру выше)
repeat // Начинаем бесконечный цикл
    state:=pointer_state; // Читаем состояние сенсора
    if state=1 then // Если произошло нажатие, то...
        begin
            // Считываем координаты нажатия
            xx:=pointer_pressed_x; yy:=pointer_pressed_y;
            if (xx>9) and (xx<111) then // Проверяем сначала x координату...
                begin
                    if (yy>9) and (yy<36) then key:=1; // Нажали на кнопку 1
                    if (yy>39) and (yy<66) then key:=2; // Нажали на кнопку 2
                    if (yy>69) and (yy<96) then Halt; // Нажали на кнопку 3 (Выход)
                    DrawButtons; // Рисуем кнопки
                end;
            end;
        end;
    Delay(30); // Небольшая пауза - надо же и другим процессам дать поработать ;- )
until false;
End. // Конец программы
```

### /code

Как видите, в программе описана обработка нажатия на 3 кнопки: при нажатии на первые 2 происходит смена цвета фона экрана, при нажатии на третью - программа закрывается.

Ну вот, собственно, и всё, что я хотел показать в данном tutorialе.

Изучите внимательно данные исходники, разберитесь в них, и обязательно попробуйте написать что-нибудь подобное самостоятельно. К примеру, попробуйте написать простенький графический редактор с возможностью рисования пером на сенсорном экране, кнопками очистки экрана, выхода и изменения цвета пера.

**Автор статьи: Odd**

# Шифруем sms.

Привет, кодер! Я думаю ты уже знаешь что такое MidletPascal и что он из себя представляет. Если ты разбирал примеры из стандартного хелпа, то, думаю, тебе будет интересно замутить что ни будь посерьезнее. Сегодня мы с тобой научимся шифровать и отправлять sms-ки. И конечно же расшифровывать их. Сегодня уже умеют клонировать сим карты, лезит в чужих телефонах через bluetooth, или под предлогом «дай картинку посмотреть». Но даже если тебе не надо ничего скрывать, то все равно предлагаю тебе прочитать эту статью, так как здесь будет рассмотрено много интересных приемов программирования на Мидлет Паскале. Так что садись поудобнее и пристегивай ремни.

В двух словах о принципе работы. В нашей программе мы вводим номер телефона, куда будет отправлена смска, секретный ключ и конечно само сообщение. Мидлет зашифровывает сообщение в соответствии с ключом и отправляет на указанный номер. Чел, которому пришла зашифрованная мессага, вводит тот же ключ, потом само сообщение (хорошо если можно вырезать сообщение и вставить куда надо, а то придется набирать вручную) и получает расшифрованную смску. Не зная ключа, эта смска так и останется абракадаброй, навеки похоронив осмысленную информацию. Теперь о программе. При ее запуске будет выведено меню с четырьмя возможными вариантами выбора:

- 1)Написать сообщение;
- 2)Расшифровать;
- 3)информация;
- 4)Выход.

В связи с этим в коде программы мы создадим четыре функции, каждая из которых будет соответствовать определенному пункту меню. Основной код будет вызывать их в определенной последовательности. Теперь рассмотрим подробно каждую функцию.

Начнем с самой простой процедуры закрытия приложения.

```
procedure exit;  
begin  
halt;//команда halt просто закрывает приложение  
exit;
```

Теперь рассмотрим процедуру вывода информации на экран

## code

```
procedure info;//процедура вывода информации  
var  
  
exitCommand:command;//объявляем команду для создания кнопки выхода  
inform:integer;//эта переменная будет использоваться для создания надписи на экране  
  
begin  
clearForm;//очищаем форму  
exitCommand:=createCommand('Exit',CM_EXIT,1);//создаем кнопку выхода  
addCommand(exitCommand);//добавляем ее на форму  
showForm;//показываем саму форму  
inform:=FormAddString('Разработчик: Лопарев Роман aka Kastor');  
//создаем на форме не редактируемую подпись  
//далее запускаем цикл и ждем пока не будет нажата кнопка выхода
```

```

while (getClickedCommand<>exitCommand) do
begin
  delay(100);//ждем 0,1 секунды
end;
end;

```

**/code**

Как видишь здесь нет ничего сложного. Я постарался все подробно расписать в комментариях.

Теперь рассмотрим самое интересное. Кодировка будет производиться следующим образом. Получаем ASCII код первого символа отправляемого сообщения, прибавляем к нему первую цифру ключа, получаем символ соответствующий полученному ASCII коду, и заменяем наш символ закодированным. Далее берем второй символ и кодируем точно так же при помощи второй цифры ключа. Когда в процессе шифрования уже была использована последняя цифра ключа, то для операции берется снова первая, потом вторая и так далее. Такая ситуация непременно произойдет так как наверняка ключ будет меньше сообщения. Этот цикл длится пока все символы смски не будут закодированы.

**code**

```

procedure smsSend;//процедура шифрования и отправки сообщения
var

okCommand,exitCommand:command;//объявляем команды ОК и выхода
number,text,key,i,n,textsize,keysize,acode,label:integer;//переменные для создания элементов
//на форме
_text,_key,_number:string;//текстовые переменные
symbol:char;//будет использоваться для замещения символа на зашифрованный

begin
clearForm;//очищаем форму
okCommand:=createCommand('ОК',CM_OK,1);//создаем кнопку ОК
showForm;//показываем форму
addCommand(okCommand);//вставляем созданную кнопку ОК
number:=FormAddTextField('Введите номер телефона',15,TF_PHONENUMBER);
//вставляем текстовое поле для ввода телефонного номера
key:=FormAddTextField('Введите ключ',10,TF_NUMERIC);
//вставляем текстовое поле для ввода ключа
//пока не нажата ОК можем спокойно заполнять форму
while (getClickedCommand<>okCommand) do//цикл, как только нажмем кнопку ОК
begin
  delay(100);
end;

_number:=FormGetText(number);//извлекаем телефонный номер из поля ввода
_key:=FormGetText(key);//и ключ

clearForm;
showTextBox('Введите сообщение',200,TF_ANY);//выводим текстовое поле для ввода сообщения
okCommand:=createCommand('Send',CM_OK,1);//снова создаем кнопку ОК
addCommand(okCommand);//и вставляем, т.к. мы форму очистили
//снова пока не нажмем ОК можем спокойно вводить сообщение
while (getClickedCommand<>okCommand) do//как только будет нажата ОК
begin

```

```

delay(100);
end;

_text:=getTextBoxString;//получаем текст введенного сообщения
keysize:=length(_key);//длину ключа
textsize:=length(_text);//и длину текста

i:=0;n:=0;//обнуляем счетчики
while i<=textsize-1 do//цикл длится от 0 до кол-ва символом введенных в сообщении-1
begin
  if n>keysize-1 then n:=0;//если n>кол-ва цифр в ключе, то обнуляем его
  acode:=ord(getChar(_text,i));//получаем аски код i-го символа сообщения
  acode:=acode+StringToInteger(getChar(_key,n));//прибавляем к нему n-ую цифру ключа
  symbol:=chr(acode);//в переменную symbol записываем полученный переведенный
//обратно в символ аски код
  _text:=setChar(_text,symbol,i);//в введенном тексте заменяем i-й символ на зашифрованный

  n:=n+1;//инкрементируем n что бы он указывал на следующую цифру ключа
  i:=i+1;//тоже самое, указывает на следующий незашифрованный символ сообщения
end;

clearForm;
label:=FormAddString('Зашифрованное сообщение');//выводим простую надпись
label:=FormAddString(_text);//выводим "на показ" полученное зашифрованное смс которое
//будет отправлено

if not smsStartSend('sms://'+_number,_text) then halt;//если смс не передано подсистеме смс
//то закрыть приложение

while smsIsSending do//выполняется цикл с задержков в 0,5сек пока отправляется смс
delay(500);

if not smsWasSuccessfull then halt;//если смс не было успешно отправлено, то закрываем
//приложение

exitCommand:=createCommand('Exit',CM_EXIT,1);//создаем
addCommand(exitCommand);//и выводим кнопку выхода

while (getClickedCommand<>exitCommand) do//как только ее нажмем,то процедура сразу
//заканчивается
  delay(100);
end;

```

**/code**

Если тебе встретились неизвестные функции, то посмотри их описание в хелпе. Надеюсь тебе понятен код по комментариям, но некоторые вещи надо так рассмотреть подробнее. После того как был введен номер, ключ и само сообщение мы получаем длину сообщения и ключа. Эти значения будут использоваться в циклах для определения достигнута ли последняя цифра ключа и символа или нет. В цикле `while i<=textsize-1 do` и в `if n>keysize-1 then n:=0` мы вычитаем единицу так как функция `length` возвращает количество символов, например 10. А их нумерация начинается с нуля, то есть десятый символ будет под номером 9. Поэтому в цикле который длится от 0 до кол-ва символов мы вычитаем единицу. Если этого не сделать в цикле, где символы перебираются, то ничего страшного не произойдет, про-



сто в конце сообщения появится левый символ. Но если этого не сделать в цикле, где перебираются цифры ключа, то шифрование реально будет происходить через жо..пу. После того как символ зашифрован, мы в переменной `_text`, которая содержит наше первоначальное сообщение, меняем не зашифрованный символ на зашифрованный. Когда шифрование всего текста завершено, мы просто выводим его, что бы пользователь видел, что все-таки будет отправлено.

Теперь рассмотрим непосредственно отправку нашего мега шифра. Только имей ввиду, что смски будут отправляться только с тех мобилок, которые поддерживают отправку сообщений в J2ME. MidletPascal пытается отправить сообщение используя Wireless Messaging API и Siemens API. Функция `smsStartSend` выполняет отправку на переданный ей, в качестве параметра, номер. Возвращает `true` если смска была передана подсистеме смс телефона. Функцию `smsIsSending` (которая возвращает `true` если смс еще отправляется) мы используем в цикле, где проверяем если смс еще не отправилось, то ждем пол секунды. И последняя функция `smsWasSuccessfull` возвращает `true` если смс было успешно отправлено. Вроде разобрались.

Процедура декодирования пришедшего смс очень похожа на процедуру декодирования, поэтому ее описывать я не буду (ее код можешь посмотреть на выложенном мной архиве всего проекта). Расскажу только принцип ее работы. С начала очищаем форму, затем создаем команду (кнопку) ОК и поле для ввода ключа. В цикле ждем нажатия кнопки, и по ее нажатию выводим большое поле для ввода зашифрованного сообщения и команды ОК (как я уже говорил если есть возможность вырезать и вставить, то писать не придется). Опять в цикле ждем ее нажатия после которого декодируем сообщение и выводим его на показ. Раскодировка сообщения происходит так же как и кодировка, только от полученного ASCII кода символа теперь вычитаем цифру ключа.

Настало время рассмотреть основной код программы.

#### code

```

program smsCript;
var
  menu,m1,m2,m3,m4:integer;//переменные для создания меню
  okCommand,clicked:command;
...
//здесь идет описание наших четырех процедур
...
begin//основной код
  while true do//запускаем бесконечный цикл
  begin
    clearForm;
    okCommand:=createCommand('ОК',CM_OK,2);//создаем кнопку ОК
    showForm;

    showMenu('smsCript',CH_IMPLICIT);//создаем главное меню
    m1:=menuAppendStringImage('Написать сообщение',loadImage('/code.png'));//а к нему
    m2:=menuAppendStringImage('Расшифровать',loadImage('/decode.png'));//4-е строки выбора
    m3:=menuAppendStringImage('Информация',loadImage('/info.png'));//дальнейшего действия
    m4:=menuAppendStringImage('Выход',loadImage('/exit.png'));//с разными картинками для красоты
    addCommand(okCommand);//всавляем кнопку ОК

    repeat
      delay(100);
      clicked:=getClickedCommand;
    until clicked=okCommand;//по нажатию на ОК смотрим,
```

```
clearForm;
```

```
if menuGetSelectedIndex=m1 then smsSend;//какой из элементов меню  
if menuGetSelectedIndex=m2 then smsDecode;//был выбран,  
if menuGetSelectedIndex=m3 then info;//такую и соответствующую  
if menuGetSelectedIndex=m4 then exit;//функцию вызовем  
end;  
end.
```

```
/code
```

В основном коде запускаем бесконечный цикл. Создаем команду ОК, а так же меню с четырьмя пунктами выбора действия. Пункты меню содержат картинки (для красоты) которые необходимо сначала нарисовать, потом поместить их в папку res проекта, и подключить их зайдя в меню Project->Import Resource File, а создать свои картинки (как я и сделал) можно зайдя в Project->New Image Resource. Далее ждем нажатия ОК после чего в соответствии с выбранным пунктом меню вызываем соответствующую процедуру.

Вот мы и закончили создание программы для шифрования и расшифрования несчастных смсок. Теперь можешь быть спокоен, украв твое смс не зная ключа фиг кто узнает, где и когда ты встречаешься с агентом 007 для получения секретного задания.

Шифруйся на здоровье!

**Автор статьи: Лопарев Роман aka Kastor**

# Танчики. Создание игрового поля и танка

Приветствую тебя, кодер! Думаю, ты уже освоился в среде MidletPascal-я и готов к очередной порции материала. Пример получился достаточно большим, так что сразу приступим. Давай разберемся, что же мы сегодня будем делать. Уверен, ты хоть раз играл на Денди в игру, которая в народе именуется «Танчики». Вот что-то вроде этого мы и забабахаем сегодня. Пока наша задача состоит в создании игрового поля с разными препятствиями и, собственно, танка, который будет по нему передвигаться. Сразу скажу, что пример рассчитан на телефоны с экранами 240x320 пикселей (поэтому проверять его на эмуляторе NHAL Win 32 Emulator будет не очень удобно), но это не значит, что надо перемотать эту статью не читая. Все ниже описанное не сложно будет реализовать под свой телефон, подправив всего несколько значений под свой экран.

Давай разберемся, что будет представлять из себя игровое поле. Оно будет разбито на определенное количество клеток, в каждой из которых может находиться препятствие. А по пустым клеткам сможет передвигаться наш танк. Это можно реализовать при помощи записей. То есть каждая клетка будет являться объектом, который будет иметь следующие свойства: 1) Свои координаты; 2) Свое состояние. С координатами, думаю все понятно. Второе свойство определяет, находится ли в этой клетке препятствие. Если состояние клетки равно 0, то она пуста и по ней без проблем может проехать танк. Если она равна 1 (первый тип препятствия), или 2 (второй тип), то танк не сможет двигаться через эту клетку. Размеры игрового поля будут 10x15 клеток, сторона каждой из которых 20 пикселей. Небольшой ввод сделан, приступаем к кодированию. Для начала создадим тип «клетка». Для этого в самом начале прописываем раздел объявления комбинированных типов данных `type` и в нем пишем следующее:

```
program NewProject;
type
TField = record //создаем тип «клетка»
  x,y: integer; //координаты
  Condition: integer; //состояние
end;
```

После этого, в разделе объявления глобальных переменных `var` объявляем массив наших клеток следующим образом:

```
var
Field: array [1..15,1..15] of TField; //массив клеток
```

Думаю, ты заметил, что наше поле должно состоять из 10x15 клеток, а массив мы объявили 15x15. Это потому, что MidletPascal еще не умеет работать с массивами такого вида, поэтому и пришлось создать массив в виде квадрата. Теперь напишем процедуру, которая будет создавать пустое игровое поле. То есть всем 150-ти клеткам будут присвоены соответствующие координаты, а состояние их будет пока равно 0.

## code

```
procedure CreateField; //процедура создания поля (пустого)
var
i,j,x,y: integer; //перем. для промежуточных значений
begin
x:=0; y:=0; //обнуляем координаты
for i:=1 to 15 do //от одного до 15 (по вертикали)
  begin
```

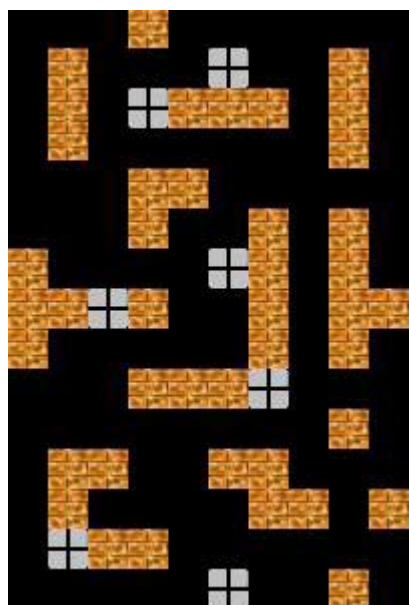
```

for j:=1 to 10 do           //от 1 до 10 (по горизонтали)
begin
  Field[j,i].x:=x;         //координаты
  Field[j,i].y:=y;         //координаты
  x:=x+20;                 //увеличиваем x
end;
x:=0;
y:=y+20;                   //как прошли все по горизонтали, увел y на 20 чтоб перейти
                           //на следующую строку
end;
end;

```

**/code**

Я надеюсь, что ты понял, как работает эта процедура, и мы можем двигаться дальше. Пустое поле мы создали, но на нем ведь должны находиться препятствия и соответствующей клетке мы должны установить свойство Condition(Состояние) в определенное значение. Как будут располагаться препятствия на поле зависит от твоей фантазии. Я предлагаю такой вариант:



			1					
	1				2			1
	1		2	1	1	1		1
	1							1
			1	1				
			1			1	1	
1					2	1		1
1	1	2	1			1	1	1
1						1	1	
			1	1	1	2		
							1	
	1	1			1	1		
	1					1	1	1
	2	1	1					
					2			1

Слева показано изображение, которое будет использоваться в игре. Справа показано игровое поле, состоящее из клеток. На нем изображено, какой клетке нужно свойству Condition(Состояние) присвоить какое значение. Для этого мы сделаем процедуру, которая будет устанавливать состояние каждой клетки в соответствии с этими рисунками.

**code**

```

Procedure CreateZone1; //процедура создания игровой зоны
begin
  Field[4,1].Condition:=1;
  Field[2,2].Condition:=1;
  ...
  Field[2,14].Condition:=2;
  Field[6,15].Condition:=2;
end;

```

**/code**


Это часть процедуры. Все остальное допиши сам в соответствии с рисунком. Только будь внимателен, если присвоить не той клетке не то значение, то твой танк при движении или сможет проходить сквозь стены или упираться в пустоту. С полем разобрались.

А дальше у нас идет создание танка. Он будет управляться следующим образом: нажимая 2,4,6,8 задаем направление его движения, а нажимая на 5 – передвигаем танк в указанном направлении. Давай подумаем, какими свойствами должен обладать наш танк. Они

будут следующими: 1)координаты; 2)количество жизней; 3)направление движения; 4)его положение (определяет, в какой клетке поля находится танк); 5)количество ходов (понадобится в будущем, при создании полноценной игры). Реализуем наш танк так же, как и клетку. Для этого в разделе type, после объявления клетки пишем следующее:

```
TPanzers = record
  x,y: integer;
  Condition: integer;           //жизни
  Direction: integer;         //направление
  j,i: integer;               //его положение
  Course: integer;           //количество ходов
end;
```

Для нашего танка необходимо подготовить следующие изображения:

Справа от изображения находится его название. Именно так их надо назвать. В зависимости от свойства Direction мы будем загружать нужное изображение. Переходим к рассмотрению основного кода программы. Для начала отредактируем раздел объявления глобальных переменных до следующего вида:

```
var
  i,j,a,x,y,key,dir:integer;           //потом понадобятся
  Field: array [1..15,1..15] of TField; //массив клеток
  Picture: array [0..3] of image;     //массив для изображений танка
  Panzers: TPanzers;                 //объявляем наш танк
```

Записал? Отлично! Далее, в начале основного кода, необходимо выполнить подготовительные действия:

#### code

```
begin
  showCanvas;           //вызов графического экрана
  CreateField;         //создаем пустое поле

  for dir:=0 to 3 do //загрузка изображений в массив
  begin
    Picture[dir]:=loadImage('/tank'+IntegerToString(dir)+'png');
  end;

  a:=20;               //сторона клетки
  Panzers.x:=a*4;      //устанавливаем свойства танка по умолчанию
  Panzers.y:=a*14;
  Panzers.Condition:=5;
  Panzers.Direction:=0;
  Panzers.j:=5;
  Panzers.i:=15;
  Panzers.Course:=7;

  CreateZone1;        //вызываем процедуру создания игровой зоны
```

```

SetColor(130,130,130);      //устанавливаем цвет серым
FillRect(0,0,GetWidth,GetHeight); //и закрашиваем им весь экран

drawImage(loadImage('/zone.png'),0,0); //выводим изображение с игровым полем

drawImage(Picture[2],a*4,a*14); //выводим изображение нашего танка
repaint;                      //перерисовываем экран
...

```

### /code

Я постарался хорошо прокомментировать код и надеюсь, что тебе здесь все понятно. Шуруем дальше. Пора реализовать цикл, в котором будет идти проверка на нажатую клавишу и в зависимости от этого выполнять соответствующие действия. Давай пока обработаем клавиши изменения направления (2,4,6,8). Нажимая 2 танк должен смотреть вверх, для этого надо его свойство Direction установить в 0. Нажимая 6 (танк смотрит вправо) устанавливаем Direction в 1 и так далее:

```

...
repeat
  key:=GetKeyClicked;      //получаем код последней нажатой клавиши

  if key=KE_KEY2 then      //если нажата была клавиша 2, то
  begin
    Panzers.Direction:=0;  //устанавливаем направление танка вверх
  end;
...

```

Остальные три направления доделай сам по аналогии. Вот мы потихоньку и подоברались к самому интересному. Настало время обработать клавишу 5 (движение). Что же должно происходить после ее нажатия? Во-первых, должна происходить проверка свойства Direction, то есть куда направлен танк. Допустим Panzers.Direction=0, то есть он смотрит вверх. Теперь надо проверить, не в самом ли верху он стоит. Если Panzers.y>0, значит не в самом и вверх можно сделать, по крайней мере, одно движение. Но и его нельзя будет сделать, если вверху будет препятствие, поэтому мы проверяем:

Field[Panzers.j,Panzers.i-1].Condition=0. То есть, проверяем свойство Condition(состояние) клетки поля, которая находится на одну позицию выше, чем танк. Если движению ничего не мешает, значит, мы можем его передвинуть на одну клетку вверх:

### code

```

...
if key=KE_KEY5 then //если нажато движение
begin
  if Panzers.Direction=0 then //если направлен вверх
  begin
    //если танк не возле самой верхней клетки и верхняя клетка не препятствие
    if Panzers.y>0 then
      if Field[Panzers.j,Panzers.i-1].Condition=0 then
        begin //то
          Moved(Panzers.Direction); //процедура плавного перемещения
          Field[Panzers.j,Panzers.i].Condition:=0; //состояние клетки уст. в ноль
          Panzers.i:=Panzers.i-1; //уст. позицию танка на одну вверх
          Field[Panzers.j,Panzers.i].Condition:=Panzers.Condition; //уст. сост. клетки куда
        //переместился танк = состоянию танка

```

```

end;
end;
if Panzers.Direction=2 then //если направлен вниз
...
drawImage(Picture[Panzers.Direction],Panzers.x,Panzers.y); //вывод на экран танка
repaint; //перерисовка
until key=KE_KEY0;

```

**/code**

Если движению ничего не мешает, то передвигаем танк, куда надо при помощи процедуры Moved, которую мы напишем чуть позже. После передвижения, устанавливаем состояние клетки, на которой стоял танк в 0. Раз он там стоял, значит, препятствий там нет и можно обнулить состояние этой клетки. Далее позицию танка переносим на одну клетку вверх. Теперь устанавливаем состояние клетки, куда переместился танк равной состоянию танка (это может потом понадобится при создании противников, которые будут стрелять по танку игрока). При установленном другом направлении все делается по аналогии, поэтому эти действия допиши сам. Если ты разобрался с передвижением танка вверх, то тебе не составит труда реализовать его передвижение в другие стороны. После обработки нажатий клавиш идет вывод на экран изображения танка в определенном месте. Теперь разберем процедуру Moved, благодаря которой танк будет плавно переезжать с места на место. Ей передается только один параметр – направление танка. Эта процедура затирает изображение танка и рисует его, например, при движении вверх, на один пиксель выше. Этот цикл повторяется пока танк полностью не войдет в новую клетку (то есть 20 раз). Вот код процедуры:

**code**

```

Procedure Moved(Direct:integer); //процедура плавного передвижения
var
i:integer;
begin
SetColor(0,0,0);

if Direct=0 then //если направлен вверх
begin
for i:=1 to 20 do
begin
FillRect(Panzers.x,Panzers.y,20,20); //затираем предыдущее изображение
Panzers.y:=Panzers.y-1; //на 1 пиксель сдвигаем наш танк вверх
drawImage(Picture[Panzers.Direction],Panzers.x,Panzers.y); //выводим его на экран
repaint; //перерисовка
delay(10); //задержка в 1/100 секунды, что танк ехал медленнее
end;
end;
if Direct=1 then
...
end;

```

**/code**

Передвижение в остальных направлениях доделай сам по аналогии. Это не сложно. Да, и не забудь поместить все используемые изображения в папку res и подключить их к проекту.

Если ты все правильно сделал, то нажимай F9 и тестируй демо-версию настоящей игры, написанной собственными руками. Поздравляю! Сегодня мы с тобой разобрались в основах создания несложной игрушки. Надеюсь, сегодня ты узнал что-то новое для себя, и мой материал будет тебе полезен. До встречи!

**Автор статьи: Лопарев Роман aka Kastor**

## Танчики. Заряжай снаряд или учим танк стрелять

Ну, вот мы снова встретились, мой дорогой друг! В прошлый раз мы с тобой создали игровое поле с препятствиям и танк. Но что же это за танк, который не умеет стрелять? Вот этим вопросом мы и займемся сегодня. Я надеюсь, ты разобрался с моей предыдущей статьей и помнишь, как устроено игровое поле, это нам сегодня понадобится. Приступаем!

С начала, давай представим себе, что будет происходить при выстреле. Нажимая на клавишу 3 (можешь взять любую) из танка на максимально определенное расстояние и в определенном направлении будет вылетать «шлейф» снаряда. Максимально определенное расстояние необходимо для большего интереса. Танк, стоящий в углу и стреляющий на все поле, это не прикольно. При выстреле, мы проверяем, есть ли на пути в максимально определенное расстояние препятствие. Если да, то проверяем, какое. Если первого типа (пробиваемое), то воспроизводим на его месте взрыв и после этого присваиваем клетке, где было это препятствие состояние 0. То есть теперь через эту клетку можно будет ездить. Если же препятствие второго типа (не пробиваемое), взрыв все равно рисуем на его месте, но после этого там же обратно прорисовываем эту бетонную стену и состояние клетки не изменяем. Если на пути снаряда препятствий нет, необходимо проверить, не вылетит ли он за пределы экрана. Если так оно и есть, то взрыв воспроизводим на границе зоны. А когда нам и препятствия не мешают, и в размеры поля укладываемся, то взрыв рисуется на максимально определенном расстоянии от танка. Это в принципе и все, что нам необходимо реализовать сегодня. Если что-то не понятно, то оно прояснится чуть позже, во время разбора кода.

Обработку нажатия клавиши 3 надо реализовать в цикле `repeat until` основного блока кода программы, сразу после обработки нажатия клавиши 5 (движение). Рассмотрим случай, когда при выстреле танк направлен вверх:

### code

```

...
if key=KE_KEY3 then
  begin
    setColor(160,90,0);           //устанавливаем цвет выстрела
    if Panzers.Direction=0 then //если направлен вверх
      begin
        for i:=1 to 6 do          //запускаем цикл проверки на 6 клеток
          begin
            if Panzers.i-i<1 then
              begin
                fire(Panzers.x,0,1,0); //если снаряд улетает за верхнюю границу, то взрыв
                //рисуем в пределах верхней границы

                break;
              end;
            if Field[Panzers.j,Panzers.i-i].Condition=1 then //если препятствие 1-го типа
              begin
                fire(Panzers.x,Panzers.y-(i*20),1,0);
                Field[Panzers.j,Panzers.i-i].Condition:=0;
                break;
              end;
            if Field[Panzers.j,Panzers.i-i].Condition=2 then //если препятствие 2-го типа
              begin
                fire(Panzers.x,Panzers.y-(i*20),2,0);
                drawImage(block,Panzers.x,Panzers.y-(i*20));
                break;
              end;
          end;
        end;
      end;
    end;
  end;

```



```

    if i=6 then fire(Panzers.x,Panzers.y-(i*20),1,0); //если выстрел на макс. опред. расст.
    end;
end;

if Panzers.Direction=2 then //если направлен вниз
...

```

### /code

Давай разбираться в коде поподробнее. Я уже говорил, что его надо вписать в основной цикл после обработки нажатия клавиши 5. В начале мы устанавливаем цвет шлейфа, который будет виден при выстреле. Далее идет проверка, если танк направлен вверх, то запускается цикл от 1 до максимально определенного расстояния выстрела, то есть до 6. В этом цикле с начала проверяется не находится ли танк в притык к границе поля, в данном случае верхней. Если да, то вызываем процедуру прорисовки взрыва. Эту процедуру мы напишем чуть позже, у нее будет четыре параметра. Первые два это координаты, где будет взрыв. Третий это вид взрыва, если встречаем препятствие второго типа, то передаем сюда число 2. Во всех остальных случаях 1-цу (потом объясню подробнее). А четвертый параметр это направление танка, что бы знать, в какую сторону шлейф рисовать. Если танк не уткнулся в границу, то проверяется, нет ли препятствия первого типа на его пути на расстоянии 1-й клетки. Если да, то вызывается процедура прорисовки взрыва в этой клетке, после чего ее состояние обнуляется. Далее идет выход из цикла при помощи break и при следующем выстреле весь цикл пойдет с начала. Но если нам такое препятствие не повстречалось, то проверяем, нет ли на пути препятствия второго типа. Если да, делаем все что и при встрече первого препятствия, только после взрыва изображение этого блока прорисовываем на его месте и состояние клетки не меняем. Если и такое препятствие нам не повстречалось, следует проверить, не достигнул ли наш снаряд своей максимальной дальности полета. Пока что нет. И цикл повторяется снова, но на этот раз он проверяет все тоже самое только на расстоянии в две клетки от танка и так далее. Если на расстоянии шести клеток он ничего не нашел, вступает в силу последняя проверка в цикле, после чего происходит взрыв в клетке, которая находится на расстоянии в шесть шагов от танка. Стрельба в остальные направления реализуется аналогично. Теперь о воспроизведении взрыва. Следует подготовить следующие изображения и назвать их так как показано ниже:



Я думаю, ты уже понял, прокручивая одно изображение за другим мы добьемся красивого бабах. Но такой бабах не подойдет при взрыве несшибаемого блока т. к. тот должен все время оставаться на месте. Для этого создаем следующие изображения:



Как ты заметил, здесь при взрыве на заднем плане всегда остается та самая стена. В разделе глобальных переменных надо объявить переменные для использования этих изображений. Предварительно их надо подключить к проекту и проследить, что бы разрешение изображений было маленькими буквами, иначе на телефоне мидлет может кроме белого экрана ничего не показать. Приведем раздел var к следующему виду:

### code

```

var
i,j,a,x,y,key,dir:integer;
Field: array [1..15,1..15] of TField; //массив клеток
Picture: array [0..3] of image;

```

```

Boom: array [1..4] of image; //изображения взрыва
Block: image; //изображение непробиваемого препятствия
BoomBlock: array [1..4] of image; //изображения взрыва непробиваемого препятствия
Panzers: TPanzers;

```

**/code**

В самом начале кода, где происходят подготовительные операции необходимо дописать следующее:

**code**

```

...
for dir:=1 to 4 do //загрузка изображений взрыва
begin
  Boom[dir]:=loadImage('/boom'+IntegerToString(dir)+'.png');
  BoomBlock[dir]:=loadImage('/boomblock'+IntegerToString(dir)+'.png');
end;

Block:=loadImage('/block1.png');
...

```

**/code**

Здесь в массивы boom и boomblock загружаются изображения простого взрыва и взрыва непробиваемого препятствия. А в переменную block – изображение самого препятствия. Теперь настало время разобраться с процедурой прорисовки взрыва, которая называется fire. Она выполняет две функции: в соответствии с направлением танка рисует шлейф снаряда, и непосредственно прорисовывает в нужном месте нужный взрыв в зависимости от находящегося там препятствия. Внутри нее объявлена всего одна переменная, которую мы будем использовать как счетчик при прорисовке взрыва. Первым делом рисуем шлейф. Для этого определяем направление танка по последнему переданному параметру. Из четырех направлений, разберем одно, когда танк смотрит вверх. Остальные ты сможешь доделать по аналогии:

**code**

```

procedure fire(x,y,cond,direct:integer); //процедура взрыва
var
i:integer;
begin
if direct=0 then //если направлен вверх
begin
drawLine(Panzers.x+10,Panzers.y,x+10,y+20);
repaint;
delay(30);
setColor(0,0,0);
drawLine(Panzers.x+10,Panzers.y,x+10,y+20);
repaint;
end;
if direct=1 then //если направлен вправо
...

```

**/code**

При помощи процедуры drawLine рисуем шлейф (простую линию). Первые две координаты это откуда начать, вторые – где закончить линию. Здесь достаточно представить, что линия должна идти с середины верхней части танка и до нижней середины клетки, кото-

рую надо взорвать. Поэтому передаются такие координаты. В остальных направлениях координаты рассчитываются так же, это не сложно. Линию провели, экран перерисовали, задержку сделали. Теперь цвет меняем на черный, проводим линию с теми же координатами и перерисовываем экран (то есть, удаляем наш шлейф). После этого необходимо реализовать сам взрыв. Для начала проверяем тип препятствия, который передается в третьем параметре. Если он единица, то взрыв будет первого вида, если 2 – то второго, где на фоне бетонная стена. Выглядит последний кусок процедуры вот так:

**code**

```
...
if cond=1 then
begin
for i:=1 to 4 do //поочередно выводятся изображения взрыва
begin
drawImage(Boom[i],x,y);
repaint;
delay(70);
end;
SetColor(0,0,0); //устанавливаем цвет в черный
FillRect(x,y,20,20); //и закрашиваем им клетку где был взрыв
end
else
begin
for i:=1 to 4 do //поочередно выводятся изображения взрыва
begin
drawImage(Boomblock[i],x,y);
repaint;
delay(70);
end;
end;
end;
```

**/code**

В зависимости от препятствия в цикле воспроизводится на экран тот или иной взрыв. При первом взрыве содержимое клетки затирается черным цветом, так как там уже все взорвалось, а при втором, наоборот, на месте клетки снова прорисовывается несокрушимое препятствие.

Теперь наш танк готов к бою! {Если что-то осталось не понятным, ты всегда сможешь изучить мой исходник выложенный на сайте}. Могу посоветовать еще при передвижении и выстреле проигрывать звуковые файлы скрежета гусениц и разрыва снаряда. Это придаст игре большего эффекта. Удачной компиляции!

**Автор статьи: Лопарев Роман aka Kastor**

# Получите, распишитесь.

Вот через несколько часов, а то и дней тяжелой работы, умственного напряжения и стучания пальцами по клавиатуре мы нажимаем на F7 и, затаив дыхание, через секунду блестящими глазами, в появившемся окошке, читаем эту замечательную надпись: «Build finished successfully!». Мигом заливаем полученные файлы на телефон, устанавливаем и бежим хвастаться друзьям своим новым «супер» приложением часто не задумываясь о том, что же это за магические файлы с расширением JAR и JAD созданные при компиляции. При создании несложных программ данная информация не обязательна, но ведь знания лишними не бывают. После компиляции мы получаем JAR и JAD файлы, о которых пойдет речь дальше.

Начнем с файла с расширением JAD, который является файлом-«дескриптором». В нем находится вспомогательная информация, необходимая Java-машине для корректного запуска мидлета (некоторые телефоны могут обходиться и без него). То есть это просто файл, в котором описывается что и как, описывается размер JAR файла, имя автора и т.д. Создай новый проект и компилируй его. Теперь зайди в папку bin этого проекта и при помощи любого текстового редактора открой файл с расширением JAD. Ты должен увидеть следующее:

```
MIDlet-1: NewProject, /icon.png, FW
MIDlet-Jar-Size: 2914
MIDlet-Jar-URL: NewProject.jar
MIDlet-Name: NewProject
MIDlet-Vendor: MIDletPascal
MIDlet-Icon: /icon.png
MIDlet-Version: 1.0.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Рассмотрим здесь самые интересные строки. **MIDlet-Jar-Size: 2914** – показывает получившийся размер JAR файла в байтах (это значение должно быть точным до последнего байта). Следующая строка **MIDlet-Jar-URL: NewProject.jar** – это ссылка на JAR файл. Если тут просто имя файла, то предполагается, что JAR лежит в той же папке, что и JAD. «Что же это за ссылка?», - поинтересуешься ты. При загрузке мидлета с WAP-портала телефон сперва скачивает JAD файл, в нем смотрит и показывает владельцу телефона свойства JAR (имя, размер), а затем с его согласия скачивает мидлет по этой самой ссылке. Едем дальше.

**MIDlet-Name: NewProject** – собственно, название мидлета.

**MIDlet-Vendor: MIDletPascal** – разработчик.

**MIDlet-Version: 1.0.0** – версия твоего приложения.

Далее **MicroEdition-Configuration: CLDC-1.0**

и **MicroEdition-Profile: MIDP-1.0** – означает, что для работы программы необходимо мобильное устройство с поддержкой платформы CLDC 1.0 и MIDP 1.0.

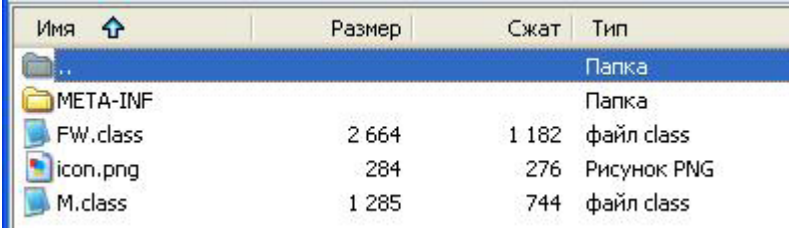
Теперь переключись в MidletPascal и найди в левой части, что то похожее на объектный инспектор в Delphi, на вкладке Properties поменяй следующие значения: в поле MIDlet name измени NewProject на SuperMidlet; в поле MIDlet vendor затри MIDletPascal и напиши Microsoft ; MIDlet version меняем с 1.0.0 на 2.0.0 и компилируем наш проект. Снова смотрим наш JAD и видим следующее:

```
MIDlet-1: SuperMidlet, /icon.png, FW
MIDlet-Jar-Size: 2971
MIDlet-Jar-URL: NewProject.jar
MIDlet-Name: SuperMidlet
```

MIDlet-Vendor: Microsoft:)  
MIDlet-Icon: /icon.png  
MIDlet-Version: 2.0.0  
MicroEdition-Configuration: CLDC-1.0  
MicroEdition-Profile: MIDP-1.0

Теперь, при установке нашего приложения или в его свойствах будет высвечиваться, что разработчиком SuperMidlet-а версии 2.0.0 является Microsoft .

Так, с JAD разобрались, переходим к JAR. Для начала определение. JAR файл – это Java архив (сокращено от англ. Java ARchive). Представляет собой обычный ZIP-архив, в котором содержится часть программы на языке Java. Чтобы JAR файл был исполняемым, он должен содержать файл MANIFEST.MF в каталоге META-INF. В какой то мере JAR это обычный ZIP-архив, потому что компания Sun приняла популярный формат архивирования файлов Zip как основу для Java-архивов. Она расширила использование формата zip различными конвенциями, обеспечив возможность упаковки классов Java в архив. С дополнением файла манифеста JAR среда исполнения Java-программ может легко найти и непосредственно выполнить main-класс Java-приложения, содержащегося в jar-файле. Выходит, что если это обычный архив, то и открыть его можно каким ни будь архиватором. Открой при помощи WinRAR наш JAR файл, полученный минуту назад. Ты увидишь следующее:



Имя	Размер	Сжат	Тип
..			Папка
META-INF			Папка
FW.class	2 664	1 182	файл class
icon.png	284	276	Рисунок PNG
M.class	1 285	744	файл class

В JAR архиве находятся два класса необходимые для запуска мидлета. Если бы во время создания приложения ты подключил бы, какие ни будь новые библиотеки (для использования новых функций), то их классы тоже бы находились здесь. Так же здесь находится иконка нашего приложения, которая является подключенным по умолчанию ресурсом приложения, находящаяся в папке res проекта. Если тебе довелось использовать какие ни будь другие ресурсы (новые картинки или звуковые файлы), то после компиляции они тоже будут здесь находиться. Зайдя в папку META-INF, которая так же находится в JAR-архиве, мы увидим файл манифеста. Открой его. Ничего не напоминает? Правильно, это же полная копия JAD-файла за исключением полей MIDlet-Jar-Size и MIDlet-Jar-URL.

Подведем итог. JAR-архив – это достаточно хорошее решение для хранения и передачи данных (то есть самого мидлета). Это хорошее хранилище файлов, а так же скорость загрузки повышается за счет того, что все файлы отдельных классов и ресурсы находятся в одном архиве. Вот мы и разобрались с этими, уже не столь загадочными файлами, полученными после компиляции.

P.S. Строение этих файлов я изучал вместе с MetallFox который часто выдвигал интересные идеи и находил новые решения, за что ему большой респект

**Автор статьи: Лопарев Роман aka Kastor.**

**От Soffrick'a: надеюсь, Вам понравилось ;)**