Graphical calculator for MIDP enabled devices(jxCalc)

Nagy Zoltan

January 6, 2008

Abstract

1 Inspiration/objectives

Today, almost every people owns a handheld cellphone, and nearly all of them is capable to run programs written for the j2me platform, this project's main objective is to provide an application for all of these devices. With the following expectations:

- formula editing
  it must have a comfortable editor interface, with the ability to go back anywhere
- scientific functions
  trigonometric/etc.
- graphical functions
  display functions
- programmable
  the calculator should provide a way to define functions, and use them freely in the formulas, with this extension - someone can 'tailor' the system to personal needs, and simplify a lot in calculations.
- easy to use
  maybe this is the most important part: it have to be as simple as possible - it have to function like a usual calculator, and it's usage should be painless as possible.

2 Using the calculator

to get started when someone starts the calculator, is as easy as taking in hand a usual calculator - with one exception, the device's keyboard is remapped in some cases, but the program is capable of displaying a picture which should be identical to the device keyboard, and displays the key mappings on the screen.
I've introduced two mode buttons to extended the availibe variations:

- 2nd functions
  i've put nearly every frequently used binary/unary operator to this map level
  on the screen this is mapped to the '*' button, and on the mapper it's logo is a laced **2**
- **alpha**
  this button covers the some symbols which can be used as variable names/parameters

the only thing which is not obvoius: the menu is accessible thru: ALPHA+[SELECT]

note: i've moved the backspace to the top-left button on the phone, because it's never was logical to put anywhere else(in my point of view ;) - but it can be moved anywhere as any other functionality, because the keyboard mapping are defined in a configuration file, which can be altered

3 Program design

because the program is built up from 95 classes not counting the parser's classes, i would like to give an overview of the whole system.
the program's main components are:

- Loader
  the function of this part is only to overseer the program's execution, and try to catch exception before they bubble up to the operating system - this is useful in debugging, and in it would give a chance for the program to recover from a program error or just to protect the user's work, and save the session.
- Utils
  this provides many basic functionality, which are not part of the midp profile.
- Menu
  this is a simple menu system, which was needed to easily configure the application
- Console
  this component provides the way to visualize the edited formulas in the editor, by rendering it to the screen
- Editor/VirtualEditor
  the real line-editor and formatting logic which sits behind the scene
- Real a floating point implementation written by Roar Lauritzsen
  availiable at: http://real-java.sourceforge.net/Real.html
- Parser with the help of javacc, the program has a real parser
- Input the keyboard input is wrapped thru a mapper which can call the bounded function.
- Plot this component makes possible to plot functions

3.1 Utilities

- Component
  i've recreated something like in the java standard edition, because on midp devices - redrawing everything - is not so cheap.

- FlowLayout
  it makes possible to layout display elements without i've to care where they would be, because partiotioning the screen by hand is very ungrateful
  FlowLayouts are used also in ScrollablePane's, which are required, but not so intresting things

- PriorityLayout
  this part extends the flowlayout functionality by using priorities, because in run time - the user should be able to toggle hiding of some parts, i want to introduce priorities to the components - to fill the screen the best way they can
  even throuh this has not been written, in theory this could address all my problems which i have with the screen partionings

- IntegerOption/EnumOption
  a small part to easily put some option into the menu, and make it configurable without the pain of having to write the n+1 part into the menu

3.2 Menu

a tree based menu-system can be build from the integer/enum options.

3.3 Console

The console renders the lines from the actual editor to the screen, and provides a way to scroll the lines, and shows the actual mode setting at the bottom.

3.4 Editor/VirtualEditor

My first implementation was to introduce breakable lines and some tricky classes to split the lines where they should. But i found this concept to be very fragile, because it's very hard to extend a system which is specialized by default - and when i wanted some more intresting features, i've decided to move forward to something else, and i've came up with a marker based design - which is a lot more precise, and dynamicly extendable, which i've find the right way to go.
I've groupped the markers into 4 different classes, based on there nature:

- static
  static markers are considered to be the properties of the document, for example: bold, italic..etc

- dynamic
  markers cleared and updated every time it's outdated, for example: bracehighlighting, cursor
  the update of these markers are dependant on the cursor and the editor itself.

- stable
  these markers are sensible only for editions in the context - it's updateable incrementally starting from the actual Cursor position, example: line wrapping, highlighting

- layout
  these are part of the stable marker set, but i've to handle them in another way, because on screen the console should somehow navigate between the lines - this should like some navigation beacon - which can help: "where to start?"

With these concept's the rendering of editor's content is simplifyed to marker-to-marker content part renders, and some interpretations of the markers.

I think that this concept is capable of drawing structured formulas as well, and support to render the entered formula into something like latex can do - and maybe it can be made editable as well.

3.5 Parser

The parser can read an entered formula and build up it's AST tree, then it's possible to execute directly the tree.
All the function are built into directly to the parser(for now) because i've wanted to usable as soon as possible.

3.6 Input

The FunctionMapper can read a keyboard configuration file, which describes where should be which functionality - because the parser 'reads' the lines - the functions are not special entites like in normal calculators
this is a part of the file:

```
      M_R     =    MOVE_RIGHT
      M_S     =    EXECUTE
@     M_S     =    MENU
!     M_S     =    RETURN

!     1       =    sin(
!@    1       =    asin(
!     2       =    cos(
```

the lines are build up from 3 parts

- mode from where the functions is accessible
  ! means 2nd is on
  @ means alpha is on

- button i've (re)named the buttons to overcame to insanity in the device manufacturers keycode world
  the keys for this have to be defined at /media/phone when someone adds the device

- command a descriptive name of the command - some classes are registered into the functionrouter, and place special keywords(and register themself as handler)
  for example: the *MOVE_RIGHT* is registered by the EditorMoveCommand etc

3.7 Media/device meta data

the program supports devices in a way that anyone can add make the application run on a new device, by creating a picture of it's screen, and giving some hints about the device's button coordinates
this is a part from the motorola-razr-v3 handset's configuration

```
# jxcalc
# keyboard mapping for the v3 handset

F_L = 13,8
F_C = 35,10
F_R = 57,8
F_X = 10,41
```

F_L gives the coordinates of the center of the top-left (Function-Left) button

3.8 Plot

currently just function plotting is supported (given in $y = f(x)$ style in the editor)
with this formula, the plotter initializes the parser, and resets the sequencer, and launches a thread to calculate all the values to display the (partial) result.
because the formula interpret is running on a double virtual machine, and these devices doesn't have much horsepower, incremental/tactial plotting can save time for the user, because of this observation i've detached the sequencer and the metric function from the plotter, so it's only provides a frame to these constructs to iteract with eachother.

4 Problems

the implementation of the markers have been not yet finished, and that's way i can't move forward to implement 'nicer' things.

5 Availability

the source can be downloaded from it's svn repository
https://demeter.teteny.bme.hu/svn/jxcalc

the project has a page on sf.net:
http://jxcalc.sf.net

the program's trac page can be found at:
http://demeter.teteny.bme.hu/trac/jxcalc