Configuring Vim for Java Development

Welcome back, my fellow Vimmers.

Posted on 1.11.2018

Welcome back, my fellow Vimmers.

Today I will show you how to convert (Neo)Vim into a Java IDE. We will be discussing multiple plugins, mappings, and external tools in this post.

Plugins

So in my `.vimrc` I have a plethora of different plugins directly related to Java development.

- Deoplete (https://web.archive.org/web/20180824150403/https://github.com/Shougo/deoplete.nvim)
- Javacomplete2 (https://web.archive.org/web/20180824150403/https://github.com/artur-shaik/vim-Javacomplete2)
- Ale (https://web.archive.org/web/20180824150403/https://github.com/w0rp/ale)
- Ultisnips (https://web.archive.org/web/20180824150403/https://github.com/SirVer/ultisnips)
- Tagbar (https://web.archive.org/web/20180824150403/https://github.com/majutsushi/tagbar)
- Ctrl-P (https://web.archive.org/web/20180824150403/https://github.com/kien/ctrlp.vim)
- NERD-TREE (https://web.archive.org/web/20180824150403/https://github.com/scrooloose/nerdtree)

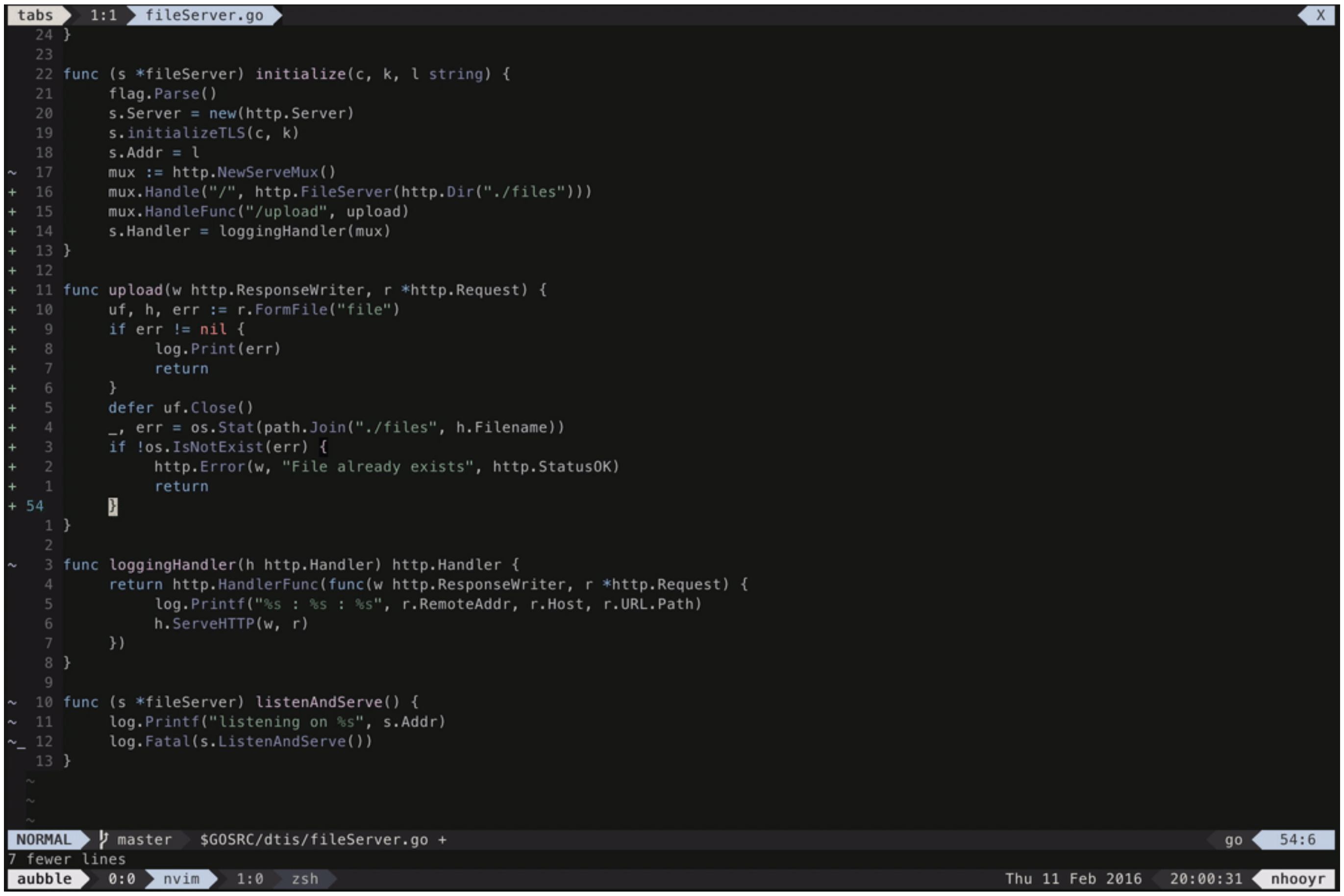
> *Also we will be assuming that you have vim-plug (https://web.archive.org/web/20180824150403/https://github.com/junegunn/vim-plug) installed as you plugin manager. If you don't, I recommend getting it. but if you don't want to make the switch you can find all of the installation instructions on the plugins GitHub.*

BE SURE TO RUN `:PlugInstall` WHEN WE ARE DONE!

Deoplete.nvim

So autocompletion in programming has been a hot topic recently. As many developers are claiming that we rely too much on autocompletion and don't learn as much about the actual syntax of the language. I disagree. I think that developers use autocompleting to be faster at what we do!

> ***Note**: this plugin only works on NeoVim (https://web.archive.org/web/20180824150403/https://neovim.io/). Or with Vim 8 and python support installed*



```
tabs  1:1  fileServer.go                                                  X
}

func (s *fileServer) initialize(c, k, l string) {
	flag.Parse()
	s.Server = new(http.Server)
	s.initializeTLS(c, k)
	s.Addr = l
	mux := http.NewServeMux()
	mux.Handle("/", http.FileServer(http.Dir("./files")))
	mux.HandleFunc("/upload", upload)
	s.Handler = loggingHandler(mux)
}

func upload(w http.ResponseWriter, r *http.Request) {
	uf, h, err := r.FormFile("file")
	if err != nil {
		log.Print(err)
		return
	}
	defer uf.Close()
	_, err = os.Stat(path.Join("./files", h.Filename))
	if !os.IsNotExist(err) {
		http.Error(w, "File already exists", http.StatusOK)
		return
	}
}

func loggingHandler(h http.Handler) http.Handler {
	return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
		log.Printf("%s : %s : %s", r.RemoteAddr, r.Host, r.URL.Path)
		h.ServeHTTP(w, r)
	})
}

func (s *fileServer) listenAndServe() {
	log.Printf("listening on %s", s.Addr)
	log.Fatal(s.ListenAndServe())
}
NORMAL  master  $GOSRC/dtis/fileServer.go +                            go  54:6
7 fewer lines
aubble  0:0  nvim  1:0  zsh                  Thu 11 Feb 2016  20:00:31  nhooyr
```

Installation

To install Deoplete add the following to your `.vimrc`

```
" Code completion
if has('nvim')
  Plug 'Shougo/deoplete.nvim', { 'do': ':UpdateRemotePlugins' }
else
  Plug 'Shougo/deoplete.nvim'
  Plug 'roxma/nvim-yarp'
  Plug 'roxma/vim-hug-neovim-rpc'
endif
```

Configuration

I recommend adding the following to your `.vimrc`:

```
" Don't forget to start deoplete let g:deoplete#enable_at_startup = 1 " Less spam let g:deoplete#auto_complete_start_length =
2
" Use smartcase
let g:deoplete#enable_smart_case = 1


" Setup completion sources
let g:deoplete#sources = {}

" IMPORTANT: PLEASE INSTALL JAVACOMPLETE2  AND ULTISNIPS OR DONT ADD THIS LINE!
" =====================================

let g:deoplete#sources.java = ['jc', 'javacomplete2', 'file', 'buffer', 'ultisnips']

" =====================================

""use TAB as the mapping
inoremap <silent><expr> <TAB>
    \ pumvisible() ?  "\<C-n>" :
    \ <SID>check_back_space() ? "\<TAB>" :
    \ deoplete#mappings#manual_complete()
function! s:check_back_space() abort "" {{{
    let col = col('.') - 1
    return !col || getline('.')[col - 1]  =~ '\s'
endfunction "" }}}
```

Now you're ready to go!

Vim-Javacomplete2

So now that we have a working completion plugin for vim let's get some Java syntax up in there.

Installation

Install the plugin by adding the following to your `.vimrc`

```
" Java-completion
Plug 'junegunn/vim-javacomplete2'
```

OR if you have a fast machine

```
" Java-completion
Plug 'junegunn/vim-javacomplete2', {'for': 'java'} " Load only for java files
```

Configuring

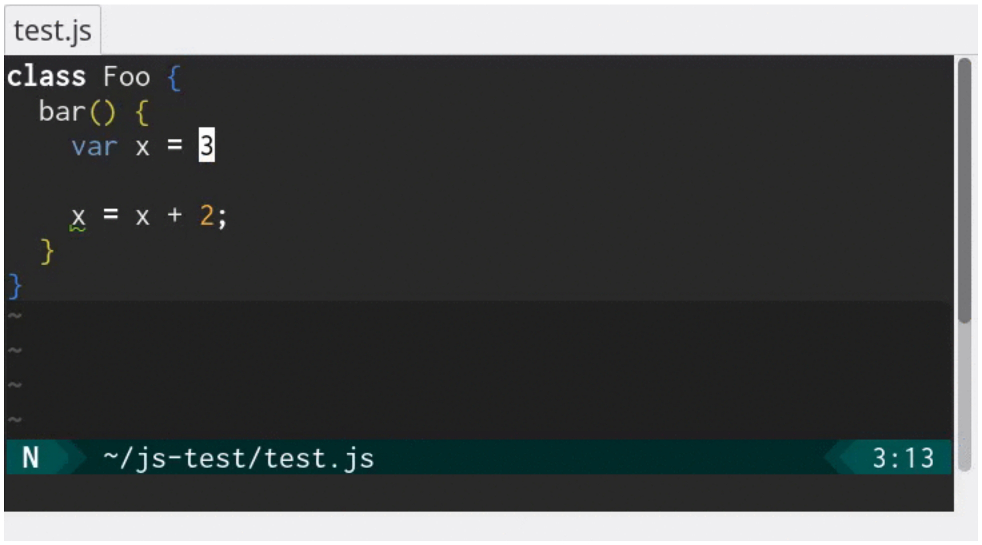
To get completion automatically when you open a Java file, add this to your `.vimrc`

```
" Java completion
autocmd FileType java setlocal omnifunc=javacomplete#Complete
autocmd FileType java JCEnable
```

ALE

Another great plugin I use (not just for Java) is ALE. Which stands for Asynchronous Linting Engine, which basically means that it tells you if you miswrote any code, and it's really fast.

You can use ale for other languages than just Java, it has a long list of support including python. C#, C++, Lua, Haskell, JavaScript (yes including node.js), a full list is available here (https://web.archive.org/web/20180824150403/https://github.com/w0rp/ale#supported-languages)



```
class Foo {
  bar() {
    var x = 3

    x = x + 2;
  }
}
```

Installation

So in order to magically fix all your code... add the following to your `.vimrc`

```
Plug 'w0rp-ale'
```

You're done.

Configuring

Add this and your brand new linter will work!

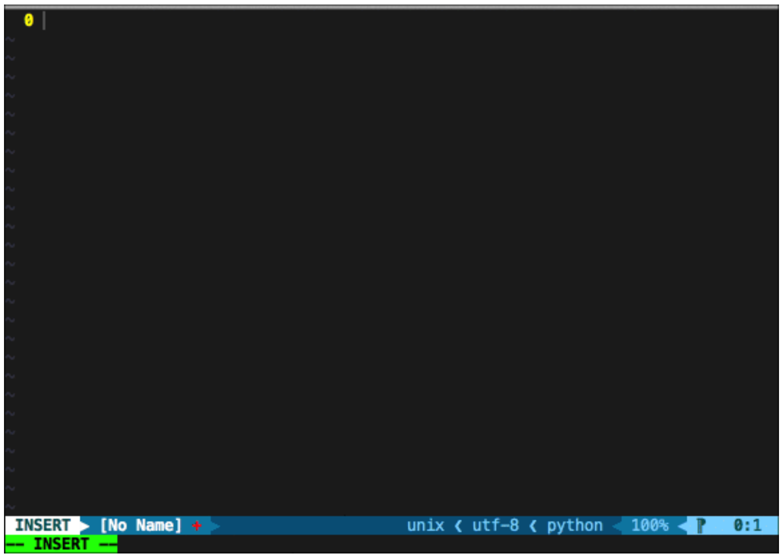
```
" Shorten error/warning flags
let g:ale_echo_msg_error_str = 'E'
let g:ale_echo_msg_warning_str = 'W'
" I have some custom icons for errors and warnings but feel free to change them.
let g:ale_sign_error = '✗✗'
let g:ale_sign_warning = '⚠⚠'

" Disable or enable loclist at the bottom of vim
" Comes down to personal preferance.
let g:ale_open_list = 0
let g:ale_loclist = 0


" Setup compilers for languages

let g:ale_linters = {
      \  'cs':['syntax', 'semantic', 'issues'],
      \  'python': ['pylint'],
      \  'java': ['javac']
      \ }
```

That's it. Now enjoy your nonerroneous code!

UltiSnips

Another tool I like to use is Ultisnips, a snippet manager worthy of your time. For those of you who don't know what a snippet is: Wikipedia - Snippet (https://web.archive.org/web/20180824150403/https://en.wikipedia.org/wiki/Snippet_(programming))

> ***Note**: This one takes a bit longer to set up. But can be extremely rewarding.*



Installation

Install the plugin by adding the following to your `.vimrc`

```
" Snippet manager
Plug 'SirVer/ultisnips'
```

Configuring

So as a mentioned above this one takes a lot longer to setup, so grab your popcorn ladies and gentlemen...

`.vimrc` config

```
" Trigger configuration. Do not use <tab> if you use https://github.com/Valloric/YouCompleteMe.

" Since we are already using Deoplete, and using tab with both doesn't work nice use <c-j> instead
let g:UltiSnipsExpandTrigger="<c-j>"
let g:UltiSnipsJumpForwardTrigger="<c-b>"
let g:UltiSnipsJumpBackwardTrigger="<c-z>"

" If you want :UltiSnipsEdit to split your window.
let g:UltiSnipsEditSplit="vertical"

let g:UltiSnipsSnippetDirectories = ['~/.vim/UltiSnips', 'UltiSnips']
let g:UltiSnipsSnippetsDir="~/.vim/UltiSnips"
```

Alright now that we have set up Vim to use snippets. Let's add some to our Ultisnips directory.

```
cd ~/.vim/UltiSnips/
```

Now that we are here, add some snippets files depending on what programming language you want.

```
touch ./<programming language here>.snippets
```

Since this a Java guide, I will be using Java, but you get the point.

Adding snippets

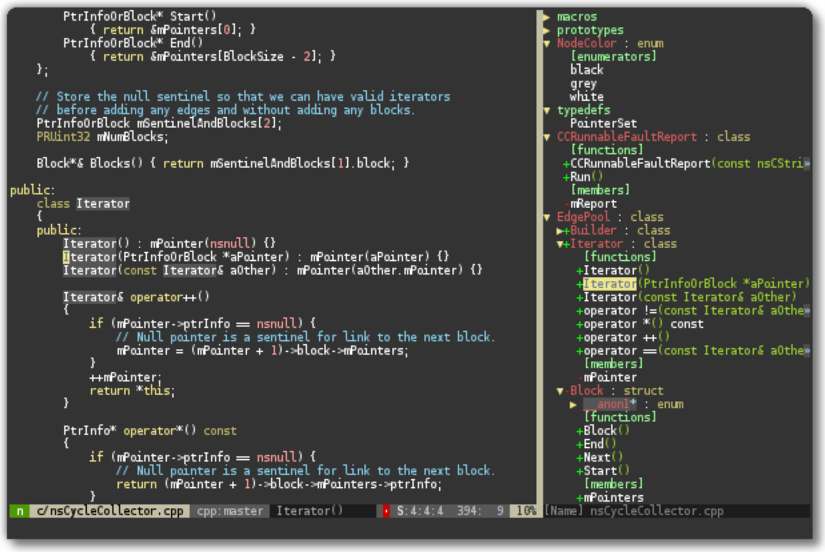
Now it is time to add some snippets. For example, my `java.snippets` might look something like:

```
priority 10


# System.out.println();
snippet sout "System.out.println();" bA
System.out.println($0);
endsnippet

snippet br "break" bA
break;
endsnippet

snippet cs "case" b
case $1:
    $2
$0
endsnippet

snippet ca "catch" b
catch (${1:Exception} ${2:e})`!p nl(snip)`{
    $0
}
endsnippet

# Main method for Java Class
snippet main "public static void main()" bA
public static void main (String[] args){
    $0
}
endsnippet

# if statement
snippet if "If this then that" bA
if ($1){
    $2
}
$3
endsnippet

# Else if statement
snippet elif "else if this then that" bA
else if ($1){
    $2
}
$3
endsnippet

snippet for "for(int i; i < imax; i++)" bA
for ($1; $2; $3){
    $4
}

endsnippet
```

This allows me to type `sout` and get `System.out.println();`, or `br` turns to `break;`, or `ca` to `catch (e)`. There are a lot more snippets in this file which I won't get into detail for your sake. But if you're interested you can read more here (https://web.archive.org/web/20180824150403/https://github.com/SirVer/ultisnips/blob/master/doc/UltiSnips.tx

TagBar

Another tool I like using is TagBar a tool that shows you the methods variables and more in a buffer to the right of your file



```
        PtrInfoOrBlock* Start()
            { return &mPointers[0]; }
        PtrInfoOrBlock* End()
            { return &mPointers[BlockSize - 2]; }
    };

    // Store the null sentinel so that we can have valid iterators
    // before adding any edges and without adding any blocks.
    PtrInfoOrBlock mSentinelAndBlocks[2];
    PRUint32 mNumBlocks;

    Block*& Blocks() { return mSentinelAndBlocks[1].block; }

public:
    class Iterator
    {
    public:
        Iterator() : mPointer(nsnull) {}
        Iterator(PtrInfoOrBlock *aPointer) : mPointer(aPointer) {}
        Iterator(const Iterator& aOther) : mPointer(aOther.mPointer) {}

        Iterator& operator++()
        {
            if (mPointer->ptrInfo == nsnull) {
                // Null pointer is a sentinel for link to the next block.
                mPointer = (mPointer + 1)->block->mPointers;
            }
            ++mPointer;
            return *this;
        }

        PtrInfo* operator*() const
        {
            if (mPointer->ptrInfo == nsnull) {
                // Null pointer is a sentinel for link to the next block.
                return (mPointer + 1)->block->mPointers->ptrInfo;
            }
 n  c/nsCycleCollector.cpp  cpp:master  Iterator()          S:4:4:4  394:  9  10%
```

```
▶ macros
▶ prototypes
▼ NodeColor : enum
    [enumerators]
    black
    grey
    white
▼ typedefs
    PointerSet
▼ CCRunnableFaultReport : class
    [functions]
   +CCRunnableFaultReport(const nsCStri»
   +Run()
    [members]
   -mReport
▼ EdgePool : class
  ▶+Builder : class
  ▼+Iterator : class
      [functions]
     +Iterator()
     +Iterator(PtrInfoOrBlock *aPointer)
     +Iterator(const Iterator& aOther)
     +operator !=(const Iterator& aOthe»
     +operator *() const
     +operator ++()
     +operator ==(const Iterator& aOthe»
      [members]
     -mPointer
  ▼-Block : struct
     ▶ __anon1* : enum
      [functions]
     +Block()
     +End()
     +Next()
     +Start()
      [members]
     +mPointers
[Name] nsCycleCollector.cpp
```

Installation

You can install TagBar by adding the following to your `.vimrc`

```
Plug 'majutsushi/tagbar'
```

Configuring

To open TagBar do `:TagbarToggle`

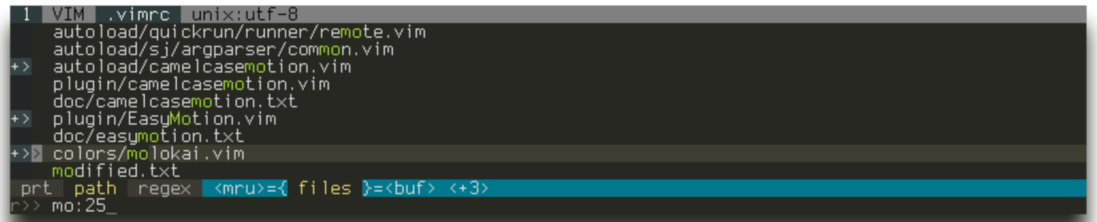
or add the following to your `.vimrc`

```
" Ctrl-b to open Tagbar
map <C-b> :TagbarToggle<CR>
```

Now you can use Ctrl-b to open Tagbar.

CtrlP.vim

Ctrlp is a great fuzzy finder which can be configured to use faster tools than grep. It can search buffers, files, and more!



```
 1  VIM  .vimrc  unix:utf-8
     autoload/quickrun/runner/remote.vim
     autoload/sj/argparser/common.vim
+>   autoload/camelcasemotion.vim
     plugin/camelcasemotion.vim
     doc/camelcasemotion.txt
+>   plugin/EasyMotion.vim
     doc/easymotion.txt
+>>  colors/molokai.vim
     modified.txt
 prt  path  regex  <mru>={ files }=<buf> <+3>
r>> mo:25_
```

Installation

You can install Ctrl-P by adding the following to your `.vimrc`

```
Plug 'ctrlpvim/ctrlp.vim'
```

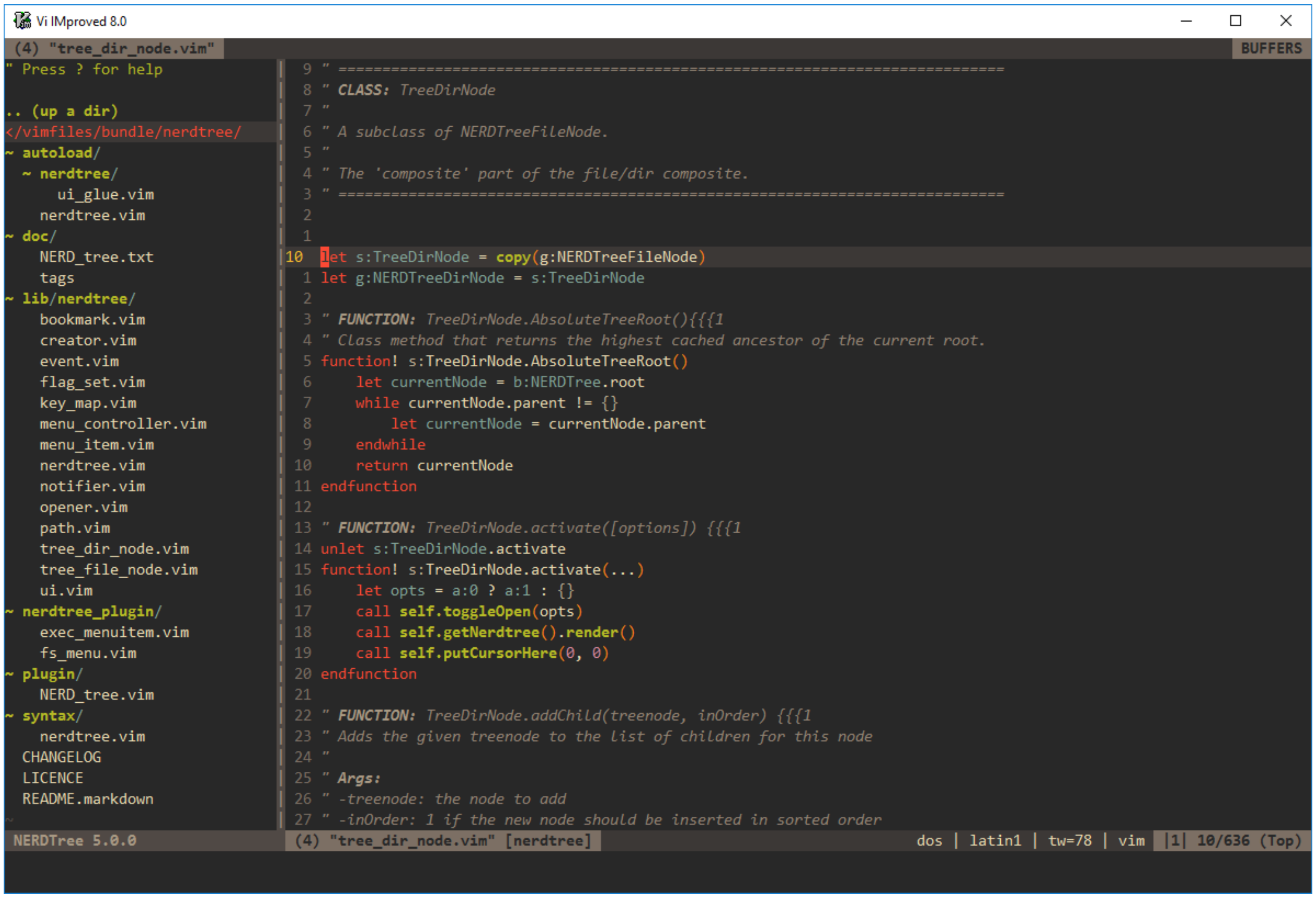
Configuring

Map Ctrl-p to open Ctrl-p.

```
let g:ctrlp_map = '<c-p>'
let g:ctrlp_cmd = 'CtrlP'

" use current git repo/file director with ctrl p
let g:ctrlp_working_path_mode = 'ra'
```

NERD TREE

NERD TREE is an advance netrw replacement for Vim, I am not saying that there is anything wrong with netrw. However, I do think many would agree with me when I say that NERD TREE is more *user freindly*.



```
" Press ? for help

.. (up a dir)
</vimfiles/bundle/nerdtree/
~ autoload/
  ~ nerdtree/
      ui_glue.vim
    nerdtree.vim
~ doc/
    NERD_tree.txt
    tags
~ lib/nerdtree/
    bookmark.vim
    creator.vim
    event.vim
    flag_set.vim
    key_map.vim
    menu_controller.vim
    menu_item.vim
    nerdtree.vim
    notifier.vim
    opener.vim
    path.vim
    tree_dir_node.vim
    tree_file_node.vim
    ui.vim
~ nerdtree_plugin/
    exec_menuitem.vim
    fs_menu.vim
~ plugin/
    NERD_tree.vim
~ syntax/
    nerdtree.vim
  CHANGELOG
  LICENCE
  README.markdown
```

```
" ============================================================================
" CLASS: TreeDirNode
"
" A subclass of NERDTreeFileNode.
"
" The 'composite' part of the file/dir composite.
" ============================================================================


let s:TreeDirNode = copy(g:NERDTreeFileNode)
let g:NERDTreeDirNode = s:TreeDirNode

" FUNCTION: TreeDirNode.AbsoluteTreeRoot(){{{1
" Class method that returns the highest cached ancestor of the current root.
function! s:TreeDirNode.AbsoluteTreeRoot()
    let currentNode = b:NERDTree.root
    while currentNode.parent != {}
        let currentNode = currentNode.parent
    endwhile
    return currentNode
endfunction

" FUNCTION: TreeDirNode.activate([options]) {{{1
unlet s:TreeDirNode.activate
function! s:TreeDirNode.activate(...)
    let opts = a:0 ? a:1 : {}
    call self.toggleOpen(opts)
    call self.getNerdtree().render()
    call self.putCursorHere(0, 0)
endfunction

" FUNCTION: TreeDirNode.addChild(treenode, inOrder) {{{1
" Adds the given treenode to the list of children for this node
"
" Args:
" -treenode: the node to add
" -inOrder: 1 if the new node should be inserted in sorted order
```

```
NERDTree 5.0.0    (4) "tree_dir_node.vim" [nerdtree]    dos | latin1 | tw=78 | vim  |1| 10/636 (Top)
```

Some things you can do with NERD TREE includes:

- Manages files
  - Move
  - Rename
  - Delete
  - Create
- Check git statuses (i.e. install nerdtree-git-plugin)
- Change working directory

Installation

You can install NERD TREE by adding the following to your `.vimrc`

```
Plug 'scrooloose/nerdtree'
```

Configuring

Add this to your `.vimrc`

```
" Open when no files were speficied on vim launch
autocmd StdinReadPre * let s:std_in=1
autocmd VimEnter * if argc() == 0 && !exists("s:std_in") | NERDTree | endif

" Toggle nerdtree
map <C-n> :NERDTreeToggle<CR>
```

Now you can use your new file tree.

Mappings

Now we will go over some shortcuts you can add to your `.vimrc`

Quick Compile

Auto compile java with leader-m

```
" Easy compile java in vim
autocmd FileType java set makeprg=javac\ %
set errorformat=%A%f:%l:\ %m,%-Z%p^,%-C.%#
```

Loc List (ALE Output)

Open and close ALEs output window with leader-e leader-w

```
"Loc List
map <leader>e :lopen<CR>
map <leader>w :lclose<CR>
```

External Tools

Use RipGrep (RG) with Ctrlp

1. Make sure RipGrep (https://web.archive.org/web/20180824150403/https://github.com/BurntSushi/ripgrep) is installed

2. Add the following to your `.vimrc`

```
function! CtrlPCommand()
  let c = 0
  let wincount = winnr('$')
  " Don't open it here if current buffer is not writable (e.g. NERDTree)
  while !empty(getbufvar(+expand("<abuf>"), "&buftype")) && c < wincount
    exec 'wincmd w'
    let c = c + 1
  endwhile
  exec 'CtrlP'
endfunction
let g:ctrlp_cmd = 'call CtrlPCommand()'

"RipGrep
if executable('rg')
  set grepprg=rg\ --color=never
  let g:ctrlp_user_command = 'rg %s --files --color=never --glob ""'
  let g:ctrlp_use_caching = 0
endif
let g:ctrlp_custom_ignore = {
      \ 'dir':  '',
      \ 'file': '\.so$\|\.dat$|\.DS_Store$|\.meta|\.zip|\.rar|\.ipa|\.apk',
      \ }
```

3. Profit!

Java compilers and command-line tools

1. Install the OpenJDK for your Java version these tend to perform better with Vim plugins and Linux.

2. I have heard good things about JavaKit as an ALE compiler. Let me know what you think.

Conclusion

Now that you have finished setting up your own *VimJ IDE*. You can now write some Java code in the coziness of your favorite text editor!

tl;dr

Wow, you're finally done! How does it feel?

Oh, wait a minute... you didn't actually read this now did you?

Whats this you want me to make a tl;dr?

Shame on YOU. You should know by now just how important it is to know everything that is in your vimrc! But fine, if you insist.

```
call plug#begin('~/.vim/bundle')

Plug 'Shougo/deoplete.nvim'
Plug 'artur-shaik/vim-javacomplete2' " , { 'for': 'java'}
Plug 'majutsushi/tagbar', { 'on': 'TagbarToggle' }
Plug 'SirVer/ultisnips'
Plug 'scrooloose/nerdtree', { 'on': ['NERDTreeToggle', 'NERDTreeFind'] }
Plug 'w0rp/ale'
Plug 'ctrlpvim/ctrlp.vim'

call plug#end()
"Ctrlp Settings {{{

let g:ctrlp_map = '<c-p>'

let g:ctrlp_cmd = 'ctrlp'
let g:ctrlp_dont_split = 'nerd'
let g:ctrlp_working_path_mode = 'rw'
set wildignore+=*/.git/*,*/tmp/*,*.swp/*,*/node_modules/*,*/temp/*,*/Builds/*,*/ProjectSettings/*
" Set no max file limit
let g:ctrlp_max_files = 0
" Search from current directory instead of project root


function! CtrlPCommand()
  let c = 0
  let wincount = winnr('$')
  " Don't open it here if current buffer is not writable (e.g. NERDTree)
  while !empty(getbufvar(+expand("<abuf>"), "&buftype")) && c < wincount
    exec 'wincmd w'
    let c = c + 1
  endwhile
  exec 'CtrlP'
endfunction
let g:ctrlp_cmd = 'call CtrlPCommand()'

"RipGrep
if executable('rg')
  set grepprg=rg\ --color=never
  let g:ctrlp_user_command = 'rg %s --files --color=never --glob ""'
  let g:ctrlp_use_caching = 0
endif
let g:ctrlp_custom_ignore = {
      \ 'dir':  '',
      \ 'file': '\.so$\|\.dat$|\.DS_Store$|\.meta|\.zip|\.rar|\.ipa|\.apk',
      \ }
" }}}
"Ale Settings {{{

let g:ale_echo_msg_error_str = 'E'
let g:ale_echo_msg_warning_str = 'W'
let g:ale_sign_error = '✗✗'
let g:ale_sign_warning = '⚠⚠'
let g:ale_open_list = 0
let g:ale_loclist = 0
"g:ale_javascript_eslint_use_global = 1
let g:ale_linters = {
      \  'cs':['syntax', 'semantic', 'issues'],
      \  'python': ['pylint'],
      \  'java': ['javac']
      \ }
" }}}
" Deoplete {{{

let g:deoplete#enable_at_startup = 1

let g:deoplete#auto_complete_start_length = 2
let g:deoplete#sources = {}
let g:deoplete#sources._=['buffer', 'ultisnips', 'file', 'dictionary']
let g:deoplete#sources.javascript = ['tern', 'omni', 'file', 'buffer', 'ultisnips']

" Use smartcase.
let g:deoplete#enable_smart_case = 1


"set completeopt-=preview

""use TAB as the mapping
inoremap <silent><expr> <TAB>
      \ pumvisible() ?  "\<C-n>" :
      \ <SID>check_back_space() ? "\<TAB>" :
      \ deoplete#mappings#manual_complete()
function! s:check_back_space() abort "" {{{
  let col = col('.') - 1
  return !col || getline('.')[col - 1]  =~ '\s'
endfunction "" }}}
" }}}
" UltiSnips {{{

" Trigger configuration. Do not use <tab> if you use https://github.com/Valloric/YouCompleteMe.
let g:UltiSnipsExpandTrigger="<c-j>"
let g:UltiSnipsJumpForwardTrigger="<c-b>"
let g:UltiSnipsJumpBackwardTrigger="<c-z>"

" If you want :UltiSnipsEdit to split your window.
let g:UltiSnipsEditSplit="vertical"

let g:UltiSnipsSnippetDirectories = ['~/.vim/UltiSnips', 'UltiSnips']
let g:UltiSnipsSnippetsDir="~/.vim/UltiSnips"

" }}}
" Java {{{

" Easy compile java in vim
autocmd FileType java set makeprg=javac\ %
set errorformat=%A%f:%l:\ %m,%-Z%p^,%-C.%#
" Java completion
autocmd FileType java setlocal omnifunc=javacomplete#Complete
autocmd FileType java JCEnable
" }}}
```

java (/web/20180824150403/http://www.averywagar.com/tag/java) **vim** (/web/20180824150403/http://www.averywagar.com/tag/vim) **dev** (/web/20180824150403/http://www.averywagar.com/tag/dev) **code** (/web/20180824150403/http://www.averywagar.com/tag/code)

OLDER POSTS →

● (https://web.archive.org/web/20180824150403/https://twitter.com/ajmwagar)

● (https://web.archive.org/web/20180824150403/https://www.facebook.com/ajmwagar)

● (https://web.archive.org/web/20180824150403/https://github.com/ajmwagar)