Domain Class

```
class Book implements Comparable {

  static withTable = 't_book'
  static transients = ['year']
  static embedded = ['price']

  String title
  Integer year
  Money price
  Date releaseDate = new Date() // default value

  Author mainAuthor
  Author coAuthor
  Map editions       // b.editions = ['first':e]
  Set publisher      // b.addToPublisher(p)

  static belongsTo = Author
  static hasMany = [editions: Edition, publisher: Publisher]

  static constraints = {
    title(size:2..5, blank:false, unique:true)
    releaseDate(min:new Date(), nullable:false)

  }

  String toString() {
    return "${id}: ${name}"
  }

  public int compareTo(obj) {
    releaseDate.compareTo(obj.releaseDate)
  }
}

class Author {
  String name
  SortedSet books
  List coBooks      // def b = a.coBooks[0]
  static hasMany =  [books:Book, coBooks:Book]
  static mappedBy = [books:'mainAuthor', coBooks:'coAuthor']
}

class Publisher {
  ...
  static mapping = {
    table 'publisher'
    cache usage:'read-only', include:'non-lazy'
    sort 'releaseDate' // default sort
    order 'desc'
    tablePerHierarchy false
    version false
    id generator:'hili',
       param:[table:'hi_value',column:'next_val',max_lo:100]
//  id composite:['name','city']
    books lazy:false, cache:'read-write'
    city column:'city_name', index:'city_idx'
    notes type:'text', length: 1000000
    photo type:'blob', length: 50000
  }
}
```

Domain Class Validation

blank	login(blank: false)	
creditcard	cardNo(creditcard:true)	
email	contactEmail(email:true)	
inList	name(inList:['A','B','C'])	S,DB
matches	name(matches:'[A-Za-z]+')	RE
max	percentage(max:100)	N,DB
maxSize	children(maxSize:10)	S,C,DB
min	price(min:0.5)	N,DB
minSize	children(minSize:5)	S,C,DB
notEqual	login(notEqual:'root')	
nullable	name(nullable:true)	
range	age(range:1..150)	N,DB
scale	price(scale:2)	N,DB
size	children(size:5..10)	S,C,DB
unique	login(unique:true) login(unique:'scope') login(unique:['group','dept'])	
url	homePage(url:true)	

validator	even(validator: { it % 2 == 0 })
	
	pwd(validator: { val, obj ->
	  obj.properties['pwd2'] == val
	})
	
	// class.login.validator.error
	login(validator:{it.length != 0})
	
	// class.login.*custom.error*
	login(validator:{
	  if (!it.endsWith('a')) {
	  return ['custom.error', arg]
	  }
	})	

N – Number, S – String, C – Collection
DB – influences schema, RE- Regular Expression

Domain Class Operations

new	def obj = new Domain(name:'abc')
get(id)	def obj = Domain.get(id)
getAll(...)	def list = Domain.getAll(2,1,3) def list = Domain.getAll([2,1,3]) def list = Domain.getAll()
list(...)	def c = Domain.list() def c = Domain.list(max:10,offset:5) def c = Domain.list(sort:'col',order:'desc')
listOrderBy*	def c = Book.listOrderByName() def c = Book.listOrderById(max: 5)
find(...)	def b = Book.find('Book b where b.id = 1') def b = Book.find('from Book as b where b.title = :title', [title:'ABC']) // find by example def b = Book.find(new Book(title:'ABC'))
findAll(...)	def c = Book.findAll('from Book b where b.title like ? order by b.title', ['ABC'], [max: 5]) // Ignore sort and order params
findBy*	
findAllBy*	

save(), delete()	obj.save()
	obj.save(flush:true)
	obj.save(false) // no validation
	obj.delete()
	obj.delete(flush:true)

findAllWhere()	def c = Book.findAllWhere(title: 'ABC')
attach()	
discard()	

hasErrors(),
errors
	def b = new Book(title:'ABC')
	b.validate()
	if (b.hasErrors()) {
	  b.errors.allErrors.each {
	  println it
	  }
	}

ExecuteQuery(...)	def c = Book.executeQuery(hql, params, [offset: 5])
ExecuteUpdate(...)	Book.executeUpdate(hql, params)
properties	obj.properties = params obj.save()
ident()	
merge()	b = b.merge()
refresh()	b.refresh()
addTo*	author.addToBooks(b)
removeFrom*	author.removeFromBooks(b)
count()	Author.count()
countBy*	Book.countByAuthor(author)
lock()	b.lock()

exists(id)	if (Author.exists(1)) {
	  ...
	}

withTransaction	Author.withTransaction { status ->
	  ...
	  // rollback
	  status.setRollbackOnly()
	}

Hibernate Criteria

```
def results = Book.withCriteria {

  def now = new Date()

  between('releaseDate', now - 7, now)
  like('title', '%Groovy%')

  or { // and, not
    eq('publisher', publisher1) // ne, lt, le, gt, ge
    eq('publisher', publisher2)
  }

  fetchMode('reviews', org.hibernate.FetchMode.EAGER)

  maxResults(10)
  order('releaseDate', 'desc')

}

def criteria = Book.createCriteria()
def results = criteria.list {
  // Same as above
}
```

Commands

```
bootstrap                   init
bug-report                  install-plugin
clean                       install-templates
compile                     list-plugins
console                     package
create-app                  package-plugin
create-controller           plugin-info
create-domain-class         release-plugin
create-filters              run-app
create-integration-test     run-war
create-plugin               schema-export
create-script               set-proxy
create-service              set-version
create-tag-lib              shell
create-unit-test            stats
doc                         test-app
generate-all                uninstall-plugin
generate-controller         upgrade
generate-views              war
help
```

Controller

actionName	

afterInterceptor	`def afterInterceptor = { model ->`
	`  println model.someVar`
	`}`
	
	`def afterInterceptor = { model,modelAndView ->`
	`  println modelAndView.viewName`
	`}`

allowedMethods	`static allowedMethods = [`
	`  delete:'POST',`
	`  search: ['GET', 'POST']`
	`]`

beforeInterceptor	`def beforeInterceptor = {`
	`  println 'before action'`
	`}`
	`def beforeInterceptor = [this.&auth,`
	`  exception: 'login']`
	`def auth = {`
	`  if (!session.user) {`
	`  redirect(action: 'login')`
	`  return false`
	`  }`
	`}`

bindData	`bindData(domain, params)`
chain	`chain([controller,] action, [id,]` `  model[, params])`
controllerName	
defaultAction	`static defaultAction = 'list'`
flash	
forward	
grailsApplication	`grailsApplication.config.`*`some.category.value`* `grailsApplication.metadata['app.version']`
params	

redirect	`redirect(controller: 'book', action: 'show',`
	`  id: 1, fragment: 'editions')`
	`// /book/show/1#editions`

render	`render 'some text'`
	`render text:'<xml>data</xml>',`
	`  contentType:'text/xml',`
	`  encoding:'UTF-8'`
	`render template:'book', model:[book:aBook]`
	`render template:'book', collection:[b1,b2,b3]`
	`render template:'book', bean:aBook`
	`render view:'viewName', model:[book:aBook]`
	`render view:'viewName'`
	`render {`
	`  div(id:'myDiv', 'some text')`
	`}`
	`render(contentType:'text/xml') {`
	`  books {`
	`  for (b in bookList) {`
	`  book(title:b.title, author:b.author)`
	`  }`
	`  }`
	`}`
	`render(contentType:'text/json') {`
	`  book(title:b.title, author:b.author)`
	`}`
	
	`import grails.converters.*`
	`...`
	`render Book.list(params) as JSON`
	`render Book.get(params.id) as XML`

request	`if (request.method == 'GET') {`
	`  ...`
	`}`

response	`response.outputStream << bytes`
servletContext	
session	

withForm	`withForm {`
	`  // good request`
	`}.invalidToken {`
	`  // bad request`
	`}`
	
	`// <g:form useToken='true' ...>`

withFormat	`withFormat {`
	`  html bookList:books`
	`  js { render "alert('hello')" }`
	`  xml { render books as XML }`
	`}`

Codecs

```
class HTMLCodec {
    static encode = { theTarget ->
        HtmlUtils.htmlEscape(theTarget.toString())
    }
    static decode = { theTarget ->
        HtmlUtils.htmlUnescape(theTarget.toString())
    }
}

// ${author.name.encodeAsHTML()}
```

TagLib

```
class MyTagLib {
  def tag = { attrs, body ->
    def name = attrs.remove('name')
    out << "${name}: "
    body()
    out.println("")
  }
}

// <g:tag name="abc">How are you?</g:tag>
// -> abc: How are you?
```

Tags

actionSubmit	`<g:actionSubmit value="Save" action="save" />`

applyLayout	`<g:applyLayout name="myLayout template="display"`
	`  collection="${books}" />`
	`<g:applyLayout name="myLayout" url="..." />`
	`<g:applyLayout>...</g:applyLayout>`

checkBox	`<g:checkBox name="myCheckbox" value="${true}" />`

collect	`<g:collect in="${books}" expr="it.title">`
	`  <p>Title: ${it}</p>`
	`</g:collect>`

cookie	`Hello, <g:cookie name="myCookie" />.`
country	`<g:country code="gbr" />`

countrySelect	<g:countrySelect name="user.country" from="['gbr', 'usa', 'deu'] valueMessagePrefix="countryname" /> countryname.gbr=United Kingdom countryname.usa=USA countryname.deu=Germany
createLink	<g:createLink action="show" id="1" /> <g:createLink action="show" params="[foo: 'bar', boo: 'far']"/> <g:createLink url="[action:'list',controller:'book']" />
currencySelect	<g:currencySelect name="myCurrency" value="${currency}" />
datePicker	<g:datePicker name="myDate" value="${aDate}" precision="day" years="${1930..1970}"/>
each	<g:each status="i" in="${items}" var="item"> ... </g:each>
eachError	<g:eachError bean="${book}"> <g:message error="${it}"/> </g:eachError>
else	<g:if test=".."> ... </g:if> <g:else> ... </g:else>
elseif	<g:if test=".."> ... </g:if> <g:elif> ... </g:elif>
fieldValue	<g:fieldValue bean="${book}" field="title" />
findAll	<g:findAll in="${books}" expr="it.author == 'Stephen King'"> Title: ${it.title} </g:findAll>
form	<g:form name="myForm" action="update" id="1"> ... </g:form> <g:form name="myForm" url="[controller:'book', action: 'list']"> ... </g:form>
formRemote	<g:formRemote controler="..." action="..." update="elementId" onEvent="alert('!')"> ... </g:form> Events: onSuccess, onFailure, on404, onUninitialized, onLoading, onLoaded, onComplete
formatBoolean	<g:formatBoolean boolean="${myBoolean}" />
formatDate	<g:formatDate date="${date}" format="dd/MM" />
formatNumber	<g:formatNumber number="${n}" format="0.00" />
grep	<g:grep in="${books}" filter="NonFictionBooks.class"> Title: ${it.title} </g:grep>
hasErrors	<g:hasErrors bean="${book}"> <g:eachError> <p>${it.defaultMessage} </g:eachError> </g:hasErrors>
header	<g:header name="Content-Type" />
hiddenField	<g:hiddenField name="myField" value="myValue" />
if	<g:if test="...">...</g:if> <g:if env="development">...</g:if>
include	<g:include controller="book" action="list" />
javascript	<g:javascript src="script.js" /> <g:javascript library="jquery" /> <g:javascript>...</g:javascript>
layoutBody	
layoutHead	
layoutTitle	
link	<g:link controller="controller" action="show" id="1"> ...</g:link> <g:link action="list" params="[sort:'name']"> ...</g:link>
localeSelect	<g:localeSelect name="myLocale" value="${locale}" />
message	<g:eachError bean="${book}"> <g:message error="${it}"/> </g:eachError>
meta	Version: <g:meta name="app.version"/>
pageProperty	<body onload="${pageProperty(name:'body.onload')}"> <g:layoutBody /> </body>
paginate	<g:paginate next="Forward" prev="Back" controller="book" action="list" total="${Book.count()}" />
passwordField	<g:passwordField name="myPwd" value="${myPwd}" />
radio	<g:radio name="myGroup" value="2" checked="true"/>
radioGroup	<g:radioGroup name="lovesGrails" labels="['Yes!','Of course!','Always!]" values="[1,2,3]"> ${it.label}" ${it.radio} </g:radioGroup>
remoteField	<g:remoteField action="changeTitle" update="titleDiv" name="title" value="${book?.title}"/> *See g:formRemote for events*
remoteFunction	$('mydiv').onclick = <g:remoteFunction action="show" id="1" update="myDiv"/>
remoteLink	<g:remoteLink action="show" id="1" update="success" onLoading="showSpinner();" onLoaded="hideSpinner();"> Test 3</g:remoteLink>
render	<g:render template="displaybook" collection="${books}" var="myBook"/>
renderErrors	<g:renderErrors bean="${book}" as="list" />
resource	<g:resource dir="css" file="main.css" />
select	<g:select id="type" name="type.id" value="${person?.type?.id}" noSelection="${['null':'Select One...']}" from="${PersonType.list()}" optionKey="id" optionValue="name" />
set	<g:set var="tomorrow" value="${new Date() + 1}" /> <g:set var="name">Hello</g:set>
sortableColumn	<g:sortableColumn property="releaseDate" defaultOrder="desc" title="Release Date" titleKey="book.releaseDate" />
submitButton	<g:submitButton name="update" value="Update" />
submitToRemote	<g:submitToRemote update="myDiv" /> *See g:formRemote for events*
textArea	<g:textArea name="myField" value="myValue" rows="5" cols="40"/>
textField	<g:textField name="field" value="${value}" />
timeZoneSelect	<g:timeZoneSelect name="myZone" value="${tz}" />
uploadForm	<g:uploadForm name="myUpload"> <input type="file" name="myFile" /> </g:uploadForm>
while	<g:while test="${i < 5}"> <%i++%> Current i = ${i} </g:while>

Reference

This cheatsheet is based on:

The Grails Framework Reference Documentation - http://grails.org/doc/1.1/

Isak Rickyanto – www.grailsdeveloper.com

Chanwit Kaewkasi - http://docs.codehaus.org/download/attachments/40788/Grails_DC_Cheat_Sheet_1.09.pdf