# Coding conventions

*“Master programmers think of systems as stories to be told rather than programs to be written”*

## Class

- All Groovy class names should be nouns, starting with capital letters.
  GOOD -: Person, Country, FileName, Activity
  BAD -: state, Run, Calculatesalary

- Name should not be plural.
  BAD -: Persons, Users

- Name should be self explanatory (longer names are better than crisp, but non understandable names). e.g; Employee of a company class shown as
  GOOD -: Employee
  BAD -: Person

- Class having abbreviations in or as names should be sensible like HTML URL etc.
  GOOD -: AdMaterial
  BAD -: OpAccount ( for OperationalAccount)

- Structure of a Class

```
class MyClass{

        /*Fields */

        /* Getter Method*/

        /*Methods */

        /*Static Methods */
}
```

## Package

- Name should be in small case
  GOOD -: com.intelligrape.hityashit.vo
  BAD -: vo.com.AdLempa.ig

- Should have structure as ‘**com.companyname.projectname.packagename**’
  GOOD -: com.lempa7.adlempa.util
  BAD -: com.util.groovy.enums

- vo, co, enums etc., should be in src/groovy with their respective package name as com.companyname.projectname.vo/co/enums etc.

## Variable

- Name should be self explainatry
  GOOD -:
  int userCount
  List<User> activeUsers = User.findAllByActive(true)

  Bad -:
  int count
  int activeUsers = User.countByActive(true)

- Should be camel case
  GOOD -: adMaterials, advertiserNames
  BAD -: admaterials, advertisernames

- Variable holding a collection should be plural.
  GOOD -:
  List<State> states = State.list()
  List<State> stateList = State.list()

  BAD-:
  def states = State.list()
  List state = State.list()
  static hasMany = [state: State]

- Avoid def
  GOOD -: int x = 1
  BAD -: def x = 1

- Add data type of variable holded by collection variable
  GOOD - : List<User> users = User.list()

AVOID -: List users = User.list()

- If a variable is expected to have a default value, then assign it at the time of declaration
  GOOD -: int x=0
  BAD -:
  int x
  x=0

- Declare each variable in a different line rather than declaring them in the same line e.g.,
  GOOD -:
  int x
  int y

  AVOID -:
  int x,y

- Constant variables should be in capital letters and separated by an underscore
  GOOD -: public static final int MAX_HEIGHT = 100
  BAD -: public static final int maxHeight = 100

- Use pronounceable names
  GOOD -: List<String> vowels = [‘a’,’e’,’i’,’o’,’u’]
  BAD -: List<String> aeiou = [‘a’,’e’,’i’,’o’,’u’]

## Method

- Name should be able to explain the intent of the method (longer names are better than crisp, but non understandable names)
  GOOD -:
  createAccountForUser(User user)
  activateAccountAndSendMail(Account account)
  updateUser(User user)

  BAD -:
  convert()
- Method should do just what its name suggests
  GOOD -:
  getActiveUsers() // It should only read the active users not more than that
  BAD -:
  activateAccount() //sends the mail to the user as well

- Should be camel case
  GOOD -: calculateSalary(User user)
  BAD -: calculatesalary(User user)

- First word should be a verb
  GOOD -:
  openAccount()
  save()
  delete()

  BAD -:
  userAccountOpen()

- Give the return type rather than using def
  GOOD -:
  List<User> getAllActiveUsersForProject(Project project)

  BAD -:
  def getAllActiveUsersForProject(Project project)

- Declare the local variable just before its use
- Follow The Thirty-Second Rule ( Your method should be readable and its intend should be understandable wtithin 30 seconds )
- Smaller is better
- Step Down Rule – Code read from top to bottom, means all the called methods should be written after the calling method
- DRY (DO NOT REPEAT YOURSELF)
- Long descriptive name is better than long comment
- AVOID more than 20 lines
- Divide one big method into small methods
- Well defined parameters (with descriptive names) are better than a map as parameter
- Each line should contain at most one statement.
  GOOD -:
  argv++
  argc++

  BAD -:
  argv++; argc--;

- More than 3 args. can be wrapped in a class. E,g.,
  GOOD -:
  ```
  class PageProperty {
          int max
          int offset
          String sort
          String order
  }
  ```

```
PageProperty pageProperty = new PrageProperty(max:10,offset:0,sort:'age',order:'asc')
getFilteredUsers(userName, age, pageProperty)
```

BAD -:

```
getFilteredUsers(userName, age, max,offset,sort,order)
```

## HTML/CSS/JS

- All the css statements should be clubbed into one block similarly all js statement
  GOOD -:

```
        <link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css',
file: 'all.css')}"/>
        <link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css',
file: 'tip.css')}"/>
        <script type="text/javascript" src="${resource(dir: 'js/lempa', file: 'all.js')}"></script>
        <script type="text/javascript" src="${resource(dir: 'js/jquery', file: 'tip-1.3.1.js')}"></script>
```

  BAD -:

```
        <link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css',
file: 'all.css')}"/>
        <script type="text/javascript" src="${resource(dir: 'js/lempa', file: 'all.js')}"></script>
        <link media="all" rel="stylesheet" type="text/css" href="${resource(dir: 'css',
file: 'tip.css')}"/>
        <script type="text/javascript" src="${resource(dir: 'js/jquery', file: 'tip-1.3.1.js')}"></script>
```

## Indentation

- Follow the indentation of 4,4,8.
- Keep formatting your code. ( DO NOT commit without formatting).
- Do not write more than 120 character in one line ( Line should be visible at one glance). If the line is more than 120 characters then  break it to next line.
- Line break should be logical like comma, brackets etc. e.g.,
  GOOD-:

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                        + 4 * longname6; // PREFER
```

  BAD-:

```
longName1 = longName2 * (longName3 + longName4
                        - longName5) + 4 * longname6; // AVOID
```

- Blank lines improve readability by setting off sections of code that are logically related.
- One blank line should always be used in the following circumstances:
  - Between methods
  - Between the local variables in a method and its first statement
  - Between logical sections inside a method to improve readability

## Comments

- Avoid them as much as possible
- Should be used if your method do multiple logical things at a time, e.g,.

```
        /*Part 1 of DR generation
*Find CalendarItems whose start date is less than or equal to run date
* Check whether it has any parent calendar items
* If it has parent calendar items than those calendar items need to be completed first
* if the parent calendar item also has the daysgap than calendarItem need to start after
  that daysgap
* if it dont have any paren than just add the calendaritems for DR generation* */

//SOME CODE

/*Part 2 of DR generation
*Find inspection tasks from yesterday daily report which are not completed
*add all the incompleted tasks to the todays daily report * */

//SOME CODE
```

- A block comment should be preceded by a blank line to set it apart from the rest of the code. e.g,.

```
/*
* Here is a block comment.
*/
```

- Short comments can appear on a single line indented to the level of the code that follows..A single-line comment should be preceded by a blank line. e.g.,

```
if (condition) {

/* Handle the condition. */
…
}
```

- Very short comments can appear on the same line but all these comments should be indented to the same tab setting. e.g.,

```
if (a == 2) {
        return TRUE;                    /* special case */
} else {
        return isPrime(a);              /* works only for odd a */
}
```

## General

- The application level constants should be in a separate file named as ProjectNameConstants.groovy
- Remove Idea generated template
- DO NOT write same query again and again
- Use Ternary/Elvis Operators instead of simple if-else.
  GOOD -:
  ```
  int x = (a>2)?b:c
  int x = a?:b
  ```
  BAD -:
  ```
  int x
  if (a>2) {
          x=b
  }else{
          x=c
  }

  int x
  if (a) {
          x=a
  }else{
          x=c
  }
  ```
- Avoid statements like return (this.active? true : false) just write return this.active
  GOOD -:
  ```
  hasBalance(User user){
          return (user.balance > 0)
  }
  ```
  BAD -:
  ```
  hasBalance(User user){
          return ((user.balance > 0)?true:false)
  }
  ```
- Avoid multiple return statements.
  GOOD -:
  ```
  String type
   if (age <13) {
                  type = ‘Child’
   }elseif ( age < 20 && age >= 13){
                          type = ‘Teenager’
   }elseif ( age < 41 && age >= 20 ) {
                  type = ‘Young’’
  ```

```
}else {
        type = ‘Old’
}
return type


 BAD-:
 if (age <13) {
                return ‘Child’
 }elseif ( age < 20 && age >= 13){
                        return ‘Teenager’
 }elseif ( age < 41 && age >= 20 ) {
                return ‘Young’’
}else {
        return ‘Old’
}
```

- Remove all unused variables, methods, imports etc. Don’t have anything in the code which is not used. (In Idea you can easily identify them)
- Avoid multiple if, else, else if. It shows the program is not object oriented.
- Do not writing printlns
- Listen to your IDE. There are cases where your IDE suggest you the better way, unused things, unused assignments.
- Have consistency in writing the return statement (as groovy allows last statement to be returned) . Either write return everywhere or nowhere.
- Do not leave commented code. You can always get the code from history.
- Put things on right place. Service for Security should have methods only related to security.
- Never ever do any operation in a println

**Bad**

```
println “Number of updated employees are : “ + service.updateEmployees(.. parameters..)
```

**Good**

```
int numberOfUpdatedEmployees = service.updateEmployees(.. parameters ..)
println “Number of updated employees are ${numberOfUpdatedEmployees}”
```

- DO NOT use magic numbers or words. Put such things in Constants which make it more readable

  ```
  GOOD -:
  public static final MAX_HEIGHT = 600
  if ( height > MAX_HEIGHT ){
          //SOME CODE
  ```

```
}
BAD -:
if(height > 600 ){
        //SOME CODE
}
```

*“Leave the campground cleaner than you found it”*