I’ve seen Grails code you wouldn’t believe...

Iván López   @ilopmar
I’ve seen Grails code you wouldn’t believe...

...Or how to properly write maintainable Grails applications

Iván López   @ilopmar
Hello!
I am Iván López (@ilopmar)
THIS IS A TRUE STORY.
The events depicted in this talk took place in “some projects” in 2017.
At the request of the survivors, the names have been changed.
Out of respect for the dead, the rest has been told exactly as it occurred.
Great examples

(Please don’t do this)
The larger a class, the better
The larger a class, the better

Controller

15 beans injected

3868 lines!
The larger a method, **the better**
The larger a method, the better

def index(Integer max)

Some stats...
- 41 if / 16 else
- Up to 5 nested ifs
- 19 service calls
- 12 dynamic finders
- 87 access to “params”
- 0 tests
The larger a *setup* method in a test, the better
The larger a *setup* method in a test, the better

Test for controller

```java
@TestFor(
  "com.example.Controller"
)
@Mock(["com.example.Service"])
@Build(["com.example.Configuration", "com.example.Middleware", "com.example.ExceptionHandler", "com.example.Strategy"])
class TestAppController extends Specification {
  SpringSecurityService springSecurityService
  SERVICE
}
```
The larger a *setup* method in a test, the better

Creating shared data

Mocking dependencies the same way for all the tests!
The larger a *setup* method in a test, the better

All same data and mocks shared among 129 tests
How to **detect** it?
- You can’t test it (or too complicated)
- IDE is slow
- Big setup methods because of lots of dependencies

How do we **fix** it?
- Small classes (it’s free write more classes)
- Small methods
- Small (or non-existent) shared test code

Why does it **matter**?
- To write clean and maintainable code
- Be able to test the code
Because **copy-paste** is always the fastest way...
Because **copy-paste** is always the fastest way...

```java
static constraints = {
    code unique: true, nullable: false
    name nullable: false, validator: { val -> if (val?.length() > 255) "default.invalid.max.size.message", 255 }
    email email: false, nullable: true, size: 1..255
    address nullable: true, validator: { val -> if (val?.length() > 500) "default.invalid.max.size.message", 500 }
    city nullable: true, validator: { val -> if (val?.length() > 255) "default.invalid.max.size.message", 255 }
    state nullable: true, validator: { val -> if (val?.length() > 255) "default.invalid.max.size.message", 255 }
    zipCode nullable: true, validator: { val -> if (val?.length() > 255) "default.invalid.max.size.message", 255 }
    phone nullable: true, validator: { val -> if (val?.length() > 1000) "default.invalid.max.size.message", 1000 }
    vStyle nullable: true, maxSize: 128
    origin nullable: true, maxSize: 128
    defaultgl nullable: true
    currencyCode nullable: true
}
```
Because **copy-paste** is always the fastest way...

```java
static constraints = {
    code unique: true, nullable: false
name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
email email: false, nullable: true, size: 1..255
address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }
city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
zipCode nullable: true
phone nullable: true
bU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }
vStyle nullable: true, maxSize: 128
origin nullable: true, maxSize: 128
defaultgl nullable: true
currencyCode nullable: true
}```
Because **copy-paste** is always the fastest way...

```kotlin
static constraints = {
    code unique: true, nullable: false,
    name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    email nullable: true, size: 1..255
    address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }
    city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    zipCode nullable: true
    phone nullable: true
    BU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }
    vStyle nullable: true, maxSize: 128
    origin nullable: true, maxSize: 128
    defaultgl nullable: true
    currencyCode nullable: true
}
```
Because **copy-paste** is always the fastest way...

```plaintext
static constraints = {
    code unique: true, nullable: false
    name nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    email nullable: false, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    address nullable: true, validator: { val -> if (val?.length() > 500) ["default.invalid.max.size.message", 500] }
    city nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    state nullable: true, validator: { val -> if (val?.length() > 255) ["default.invalid.max.size.message", 255] }
    zipCode nullable: true
    phone nullable: true
    BU nullable: true, validator: { val -> if (val?.length() > 1000) ["default.invalid.max.size.message", 1000] }
    vStyle nullable: true, maxSize: 128
    origin nullable: true, maxSize: 128
    defaultgl nullable: true
    currencyCode nullable: true
}
```
Because **copy-paste** is always the fastest way...

```java
class CustomValidations {

    protected static final String INVALID_MAX_SIZE_MESSAGE = "default.invalid.max.size.message"

    static def validateLength(Integer maxLength) {
        return { val, obj ->
            if (val?.length() > maxLength) {
                return [INVALID_MAX_SIZE_MESSAGE, maxLength]
                // else {  
                //     true
            } else {
                true
            }
        }
    }

    static constraints = {
        code unique: true, nullable: false
        name nullable: false, validator: CustomValidations.validateLength(255)
        email email: false, nullable: true, size: 1..255
        address nullable: true, validator: CustomValidations.validateLength(500)
        city nullable: true, validator: CustomValidations.validateLength(255)
        state nullable: true, validator: CustomValidations.validateLength(255)
        zipCode nullable: true
        phone nullable: true
        BU nullable: true, validator: CustomValidations.validateLength(1000)
        vStyle nullable: true, maxSize: 128
        defaultgl nullable: true
        currencyCode nullable: true
    }
```
Because copy-paste is always the fastest way...

class CustomValidations {

    protected static final String INVALID_MAX_SIZE_MESSAGE = "default.invalid.max.size.message"

    static def validateLength(Integer maxLength) {
        return { val, obj ->
            if (val?.length() > maxLength) {
                return [INVALID_MAX_SIZE_MESSAGE, maxLength]
            } else {
                true
            }
        }
    }

    static constraints = {
        code unique: true, nullable: false
        name nullable: false, validator: CustomValidations.validateLength(255)
        email email: false, nullable: true, size: 1..255
        address nullable: true, validator: CustomValidations.validateLength(500)
        city nullable: true, validator: CustomValidations.validateLength(255)
        state nullable: true, validator: CustomValidations.validateLength(255)
        zipCode nullable: true
        phone nullable: true
        bu nullable: true, validator: CustomValidations.validateLength(1000)
        vStyle nullable: true, maxSize: 128
        origin nullable: true, maxSize: 128
        defaultgl nullable: true
        currencyCode nullable: true
    }
}
Because *copy-paste* is always the fastest way...

class CustomValidations {
    protected static final String INVALID_MAX_SIZE_MESSAGE = "default.invalid.max.size.message"

    static def validateLength(Integer maxLength) {
        return { val, obj ->
            if (val?.length() > maxLength) {
                return [INVALID_MAX_SIZE_MESSAGE, maxLength]
            } else {
                true
            }
        }
    }

    static constraints = {
        code unique: true, nullable: false
        name nullable: false, validator: CustomValidations.validateLength(255)
        email email: false, nullable: true, size: 1..255
        address nullable: true, validator: CustomValidations.validateLength(500)
        city nullable: true, validator: CustomValidations.validateLength(255)
        state nullable: true, validator: CustomValidations.validateLength(255)
        zipCode nullable: true
        phone nullable: true
        bU nullable: true, validator: CustomValidations.validateLength(1000)
        vStyle nullable: true, maxSize: 128
        defaultgl nullable: true
        currencyCode nullable: true
    }
}
How to **detect** it?
- Lot of duplicated code

**How do we fix it?**
- Don’t repeat yourself (DRY)
- If you copy-paste +2 times → extract, refactor & reuse

**Why does it matter?**
- Only need to change code in one place
- Reuse the code
def createUserWithActionPermissions(final itemInstance, final currentParams, final currentUser) {
    def actionList = null
    def actionPermissionError = null
    def actionPermissionList = null
    def userInstance = null
    def errorInstance = null
    def exceptionInstance = null
    def statusList = null
    final customParams = this.buildUserParams((BookType) itemInstance, currentParams, currentUser)
    ...
    ...
}
Use `def` everywhere because `#yolo`

```python
def createUserWithActionPermissions(final itemInstance, final currentParams, final currentUser) {
    def actionList = null
    def actionPermissionError = null
    def actionPermissionList = null
    def userInstance = null
    def errorInstance = null
    def exceptionInstance = null
    def statusList = null
    final customParams = this.buildUserParams((BookType) itemInstance, currentParams, currentUser)
    ...
    ...
}
```

What about the returned type?

Everything final but without type

A bunch of variables without type
How to detect it?
- Code-reviews

How do we fix it?
- Always write your methods signatures as in Java
- If the type is not obvious, write it

Why does it matter?
- Be able to read the code
- Help IDE to help us: auto-completion, refactor, searches,…
- Help future you (and your team) to read the code
Use **String** everywhere
Use **String** everywhere

**Targets**
- Occurrences of "Identifier" in Project

**Found Occurrences** 65 occurrences
- Usage in string constants 65 occurrences

**Targets**
- Occurrences of "Identifier" in Project

**Found Occurrences** 55 occurrences
- Usage in string constants 55 occurrences
How to detect it?
- Code-reviews
- Codenarc

How do we fix it?
- Use Enum/Constants
- Hide behavior inside enum (static methods)

Why does it matter?
- Find usages of the constants
- Refactor the code
Use **GORM** everywhere...
Use **GORM** everywhere...

- **Targets**
  - Occurrences of `find*By` in Project with mask `**Service.groovy`
  - **Found Occurrences**: 382 occurrences
    - Unclassified occurrence: 382 occurrences

- **Targets**
  - Occurrences of `find*By` in Project with mask `*.gsp`
  - **Found Occurrences**: 59 occurrences
    - Unclassified occurrence: 59 occurrences

- **Targets**
  - Occurrences of `find*By` in Project with mask `*Controller.groovy`
  - **Found Occurrences**: 277 occurrences
    - Unclassified occurrence: 277 occurrences
Use **GORM** everywhere...

- **Targets**
  - Occurrences of `find*By` in Project with mask `**Service.groovy`

- **Found Occurrences** 382 occurrences
  - Unclassified occurrence 382 occurrences

- **Targets**
  - Occurrences of `find*By` in Project with mask `**Controller.groovy`

- **Found Occurrences** 277 occurrences
  - Unclassified occurrence 277 occurrences
Use **GORM** everywhere...

**Targets**
- Occurrences of 'find*By' in Project with mask '*Service.groovy'

**Found Occurrences** 382 occurrences
- Unclassified occurrence 382 occurrences

**Targets**
- Occurrences of 'find*By' in Project with mask '*.gsp'

**Found Occurrences** 59 occurrences
- Unclassified occurrence 59 occurrences

**Targets**
- Occurrences of 'find*By' in Project with mask '*Controller.groovy'

**Found Occurrences** 277 occurrences
- Unclassified occurrence 277 occurrences
How to detect it?
- Code-reviews

How do we fix it?
- Create new layer RepositoryService, GormService, ...
- Only access db from this layer

Why does it matter?
- Decouple from the persistence
- Integration tests vs Unit tests
Life is too short for NullPointerException...
Life is too short for `NullPointerException`...
How to **detect** it?
- Code-reviews
- Codenarc (UnnecessarySafeNavigationOperator Rule)

How do we **fix** it?
- Only use `?' when values **really** can be null
- Think twice before using it

Why does it **matter**?
- Easy to read/maintain code
Pass **params** to Services, mutate and return
Pass *params* to Services, mutate and return
How to detect it?
- Code-reviews

How do we fix it?
- Just don’t do it
- Use types, static inner classes, @Builder

Why does it matter?
- Params belongs to Controllers layer
- Decouple Controllers from Services
Who needs Command Objects? **params FTW!**
Who needs Command Objects? params FTW!
Who needs Command Objects? **params FTW!**

def updateVehicle() {
    
    // Code implementation...

    def vehicleService = new VehicleService()
    vehicleService.updateVehicle(vehicleInstance, params, vehicleCommand, vehicleDetailCommand)
}
Who needs Command Objects? **params FTW!**

```java
def updateVehicle() {
    params.dateFormat = currentUser.dateFormat.format
    params.buildStatus = BuildStatus.findById(uiStatus).workflowStatus

    VehicleCommand vehicleCommand = new VehicleCommand()
    bindData(vehicleCommand, params)
    vehicleCommand.description = params.description
    vehicleCommand.category = params.category
    vehicleCommand.messageSource = messageSource

    VehicleDetailCommand vehicleDetailCommand = new VehicleDetailCommand()
    bindData(vehicleDetailCommand, params)
    vehicleDetailCommand.messageSource = messageSource

    vehicleService.updateVehicle(vehicleInstance, params, vehicleCommand, vehicleDetailCommand)
}
```

- **Modifying params**
- **Use COs the wrong way**
- **Pass params and COs to service**
How to **detect** it?
- Code-reviews

How do we **fix** it?
- Always use Command Objects
- Never use/modify “params"

Why does it **matter**?
- Know exactly the expected params
- Test data binding and params transformations
Who needs Services if we have `@Transactional` Controllers?
Who needs Services if we have `@Transactional` Controllers?

208 actions with business logic that can't be reused.
How to **detect** it?

- Code-reviews
- Codenarc

**How do we fix it?**

- Put business logic on services
- If you need to add `@Transactional` in controller → Move to service

**Why does it matter?**

- Reuse business logic if it’s in Services
- HTTP Layer ≠ Business Logic
Tests are for people who write bugs...

/* This is not used yet, should be fine but no tests */

def doStuff(final BookType bookTypeInstance, final Map incomingParams) {
    def results = [:]
Tests are for people who write bugs...

```java
/* This is not used yet, should be fine but no tests */
def doStuff(final BookType bookTypeInstance, final Map incomingParams) {
    def results = [:]
}
How to **detect** it?
- Code-reviews
- Coverage reports

How do we **fix** it?
- TDD
- Just don't push code without tests

Why does it **matter**?
- Modifications, refactors, new features
- Sleep well at night
Should I write tests?

- Yes

  What if I just need to finish something fast?

    - Yes

      What if I just...

        - OMG YES!!!
Default **UrlMappings** because I’m too lazy...

```java
class UrlMappings {

    static mappings = {
        "/$controller/$action?/$id?($.format)?$"{
            constraints {
                // apply constraints here
            }
        },

        "/"(view:'/index')
        "500"(view:'/error')
        "404"(view:'/notFound')
    }
}
```
Default UrlMappings because I'm too lazy...

class UrlMappings {
    static mappings = {
        "/$controller/$action?/$id?($format)?{"
            constraints {
                // apply constraints here
            }
        }

        "/"(view:'/index')
        "500"(view:'/error')
        "404"(view:'/notFound')
    }
}
How to detect it?
- Code-reviews

How do we fix it?
- Delete the default mappings
- Create manually a mapping per controller action

Why does it matter?
- Have in one place all urls in application
The master piece

def getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    Sql sql = Sql.newInstance(dataSource) as Sql
    def insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);
    String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("[",""").replace("",""").replace(" ",""").trim()
    String companies=companyList?.id?.toString()?.replace("[",""").replace("",""").replace(" ",""").trim()
    String departments=departmentList?.id?.toString()?.replace("[",""").replace("",""").replace(" ",""").trim()
    String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("[",""").replace("",""").replace(" ",""").trim()
    String regions=regionList?.id?.toString()?.replace("[",""").replace("",""").replace(" ",""").trim()
    String divisions=divisionList?.id?.toString()?.replace("[",""").replace("",""").replace(" ",""").trim()
    String stmt="select count(*) as counter from insurance_policy_condition where"
    if(insurancePolicyList?.size()>0){
        stmt=stmt+" and insurance_policy_id in (${insurancePolicies})"
    }
    if(companyList?.size()>0){
        stmt=stmt+" and company_id in (${companies})"
    }
    if(departmentList?.size()>0){
        stmt=stmt+" and department_id in (${departments})"
    }
    if(subdivisionTypesList?.size()>0){
        stmt=stmt+" and subdivision_type_id in (${subdivisionTypes})"
    }
    if(regionList?.size()>0){
        stmt=stmt+" and region_id in (${regions})"
    }
    if(divisionList?.size()>0){
        stmt=stmt+" and division_id in (${divisions})"
    }
    stmt=stmt.replaceFirst("and","")
    def row;
    if(stmt!=""){
        row = sql.firstRow(stmt)
        return row.counter
    }else{
        return 0
    }
}
The master piece

```java
def getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
  Sql sql = Sql.newInstance(dataSource) as Sql
  def insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);
  String insurancePolicies=insurancePolicyList?.id?.toString()?.replace("["","")?.replace("","]")?.replace(" "," ")?.trim()
  String companies=companyList?.id?.toString()?.replace("["","")?.replace("","]")?.replace(" "," ")?.trim()
  String departments=departmentList?.id?.toString()?.replace("["","")?.replace("","]")?.replace(" "," ")?.trim()
  String subdivisionTypes=subdivisionTypesList?.id?.toString()?.replace("["","")?.replace("","]")?.replace(" "," ")?.trim()
  String regions=regionList?.id?.toString()?.replace("["","")?.replace("","]")?.replace(" "," ")?.trim()
  String divisions=divisionList?.id?.toString()?.replace("["","")?.replace("","]")?.replace(" "," ")?.trim()

  def stmt="""select count(*) as counter from insurance_policy_condition where"""
  if(insurancePolicyList?.size()>0){
    stmt=stmt+""" and insurance_policy_id in (${insurancePolicies})"""
  }
  if(companyList?.size()>0){
    stmt=stmt+""" and company_id in (${companies})"""
  }
  if(departmentList?.size()>0){
    stmt=stmt+""" and department_id in (${departments})"""
  }
  if(subdivisionTypesList?.size()>0){
    stmt=stmt+""" and subdivision_type_id in (${subdivisionTypes})"""
  }
  if(regionList?.size()>0){
    stmt=stmt+""" and region_id in (${regions})"""
  }
  if(divisionList?.size()>0){
    stmt=stmt+""" and division_id in (${divisions})"""
  }

  stmt=stmt.replaceFirst("and","")
  def row
  if(stmt!="""
    row = sql.firstRow(stmt)
    return row.counter
  }else{
    return 0
  }
}
```
Let's pretend we didn't see this signature...
The master piece

def getInsurancePolicyConditions(bookType, companyList, departmentList, regionList, subdivisionTypesList, divisionList):
    sql = Sql.newInstance(dataSource) as Sql
    insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);
    insurancePolicies = insurancePolicyList?.id?.toString()?.replace("[","").replace("]","").replace(" ","").trim()
    companies = companyList?.id?.toString()?.replace("[","").replace("]","").replace(" ","").trim()
    departments = departmentList?.id?.toString()?.replace("[","").replace("]","").replace(" ","").trim()
    subdivisionTypes = subdivisionTypesList?.id?.toString()?.replace("[","").replace("]","").replace(" ","").trim()
    regions = regionList?.id?.toString()?.replace("[","").replace("]","").replace(" ","").trim()
    divisions = divisionList?.id?.toString()?.replace("[","").replace("]","").replace(" ","").trim()
    stmt = "select count(*) as counter from insurance_policy_condition where"
    if(insurancePolicyList?.size()>0):
        stmt=stmt+" and insurance_policy_id in ("+insurancePolicies+")"
    if(companyList?.size()>0):
        stmt=stmt+" and company_id in ("+companies+")"
    if(departmentList?.size()>0):
        stmt=stmt+" and department_id in ("+departments+")"
    if(subdivisionTypesList?.size()>0):
        stmt=stmt+" and subdivision_type_id in ("+subdivisionTypes+")"
    if(regionList?.size()>0):
        stmt=stmt+" and region_id in ("+regions+")"
    if(divisionList?.size()>0):
        stmt=stmt+" and division_id in ("+divisions+")"
    stmt=stmt.replaceFirst("and",""")
def row:
    if(stmt!="”):
        row = sql.firstRow(stmt)
        return row.counter
    else:
        return 0
}
The master piece

```
String companies =
    companyList?.
    id?.
    toString()?.
    replace("[","")?.
    replace("]","")?.
    replace(" ","")?.
    trim()
```

The result will be a String, at least is not def
String companies =
  companyList?.
  id?.
  toString()?.
  replace("[",""])?.
  replace(""]",""])?.
  replace(" ",""])?.
  trim()

List<Company> companyList
according method signature
String companies = companyList?.id?.toString()?.replace("[","")?.replace("]","")?.replace(" ","")?.trim()

Collect and extract the id of the domain objects [2, 5, 7]
String companies =
  companyList?.
  id?.
  toString()?.
  replace("[","")?
  replace("]","")?
  replace(" ","")?
  trim()

String representation of the list  "[2, 5, 7]"
String companies =
    companyList?.
    id?.
    toString()?
    replace("[","")?.
    replace("]","")?.
    replace(" ","")?.
    trim()

Remove [ "2, 5, 7]"
The master piece

String companies = companyList?.
id?.
toString()?.
replace("[","").
replace("]","").
replace(" ","").
trim()  Remove ]

"2, 5, 7"
The master piece

String companies =
    companyList?.
    id?.
    toString()?.
    replace("[",""])?.
    replace("]",""])?.
    replace(" ",""])?.
    trim()

Remove "space"  "2,5,7"
String companies = companyList?.id?.toString()?
.replace("[","")?
.replace("]","")?
.replace(" ","")?
.trim()

String insurancePolicies=insurancePolicyList?.id?.toString()?
.replace("[","")?
.replace("]","")?
.replace(" ","")?
.trim()

String regions=regionList?.id?.toString()?
.replace("[","")?
.replace("]","")?
.replace(" ","")?
.trim()
String stmt=""
  select count(*) as counter from insurance_policy_condition where"
"
if(insurancePolicyList?.size()>0){
  stmt=stmt+"""" and insurance_policy_id in (${insurancePolicies})"""
}
if(companyList?.size()>0){
  stmt=stmt+"""" and company_id in (${companies})"""
}
if(departmentList?.size()>0){
  stmt=stmt+"""" and department_id in (${departments})"""
}
if(subdivisionTypesList?.size()>0){
  stmt=stmt+"""" and subdivision_type_id in (${subdivisionTypes})"""
}
if(regionList?.size()>0){
  stmt=stmt+"""" and region_id in (${regions})"""
}
if(divisionList?.size()>0){
  stmt=stmt+"""" and division_id in (${divisions})"""
}
""""
String stmt="""select count(*) as counter from insurance_policy_condition where"""
   if(insurancePolicyList?.size()>0){
      stmt=stmt+"""" and insurance_policy_id in (${insurancePolicies})"""
   }
   if(companyList?.size()>0){
      stmt=stmt+"""" and company_id in (${companies})"""
   }
   if(departmentList?.size()>0){
      stmt=stmt+"""" and department_id in (${departments})"""
   }
   if(subdivisionTypesList?.size()>0){
      stmt=stmt+"""" and subdivision_type_id in (${subdivisionTypes})"""
   }
   if(regionList?.size()>0){
      stmt=stmt+"""" and region_id in (${regions})"""
   }
   if(divisionList?.size()>0){
      stmt=stmt+"""" and division_id in (${divisions})"""
The master piece

```java
String stmt="""select count(*) as counter from insurance_policy_condition where""
if(insurancePolicyList?.size()>0){
    stmt=stmt+""" and insurance_policy_id in ("${insurancePolicies}\")"
}
if(companyList?.size()>0){
    stmt=stmt+""" and company_id in ("${companies}\")"
}
if(departmentList?.size()>0){
    stmt=stmt+""" and department_id in ("${departments}\")"
}
if(subdivisionTypesList?.size()>0){
    stmt=stmt+""" and subdivision_type_id in ("${subdivisionTypes}\")"
}
if(regionList?.size()>0){
    stmt=stmt+""" and region_id in ("${regions}\")"
}
if(divisionList?.size()>0){
    stmt=stmt+""" and division_id in ("${divisions}\")"
}
The string with the ids: "2,5,7"
Build the sql query concatenating strings
```

Variable on method signature
String stmt=""""select count(*) as counter from insurance_policy_condition where"""" if(insurancePolicyList?.size()>0){
    stmt=stmt+"""" and insurance_policy_id in (${insurancePolicies})"""
}
if(companyList?.size()>0){
    stmt=stmt+"""" and company_id in (${companies})"""
}
if(departmentList?.size()>0){
    stmt=stmt+"""" and department_id in (${departments})"""
}
if(subdivisionTypesList?.size()>0){
    stmt=stmt+"""" and subdivision_type_id in (${subdivisionTypes})"""
}
if(regionList?.size()>0){
    stmt=stmt+"""" and region_id in (${regions})"""
}
if(divisionList?.size()>0){
    stmt=stmt+"""" and division_id in (${divisions})"""
}
stmt=stmt.replaceFirst(""""and"""","")
The master piece

String stmt="""select count(*) as counter from insurance_policy_condition where"""

def row

if(stmt!=''){
    row = sql.firstRow(stmt)
    return row.counter
}else{
    return 0
}
The master piece

String stmt="""select count(*) as counter from insurance_policy_condition where"""
...
def row
if(stmt!=''){
    row = sql.firstRow(stmt)
    return row.counter
}else{
    return 0
}
The master piece: The solution

```java
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
    return InsurancePolicyCondition.createCriteria().count {
        if (insurancePolicyList) {
            'in' 'insurancePolicy', insurancePolicyList
        }
        if (companyList) {
            'in' 'company', companyList
        }
        if (departmentList) {
            'in' 'department', departmentList
        }
        if (subdivisionTypesList) {
            'in' 'subdivision', subdivisionTypesList
        }
        if (regionList) {
            'in' 'region', regionList
        }
        if (divisionList) {
            'in' 'division', divisionList
        }
    }
}
```
The master piece: The solution

```java
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList,
List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)
    return InsurancePolicyCondition.createCriteria().count {
        if (insurancePolicyList) {
            'in' 'insurancePolicy', insurancePolicyList
        }
        if (companyList) {
            'in' 'company', companyList
        }
        if (departmentList) {
            'in' 'department', departmentList
        }
        if (subdivisionTypesList) {
            'in' 'subdivision', subdivisionTypesList
        }
        if (regionList) {
            'in' 'region', regionList
        }
        if (divisionList) {
            'in' 'division', divisionList
        }
    }
}
```
The \textbf{master} piece: The solution

```java
Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);

    return InsurancePolicyCondition.createCriteria().count {
        if (insurancePolicyList) {
            'in' 'insurancePolicy', insurancePolicyList
        }
        if (companyList) {
            'in' 'company', companyList
        }
        if (departmentList) {
            'in' 'department', departmentList
        }
        if (subdivisionTypesList) {
            'in' 'subdivision', subdivisionTypesList
        }
        if (regionList) {
            'in' 'region', regionList
        }
        if (divisionList) {
            'in' 'division', divisionList
        }
    }
}
```
The **master piece**: The solution

Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList, List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {
    List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType);
    return InsurancePolicyCondition.createCriteria().count {
        if (insurancePolicyList) {
            'in' 'insurancePolicy', insurancePolicyList
        }
        if (companyList) {
            'in' 'company', companyList
        }
        if (departmentList) {
            'in' 'deparment', departmentList
        }
        if (subdivisionTypesList) {
            'in' 'subdivision', subdivisionTypesList
        }
        if (regionList) {
            'in' 'region', regionList
        }
        if (divisionList) {
            'in' 'division', divisionList
        }
    }
}
The master piece: The solution

Integer getInsurancePolicyConditions(String bookType, List<Company> companyList, List<Department> departmentList,
List<SubdivisionType> subdivisionTypesList, List<Region> regionList, List<Division> divisionList) {

List<InsurancePolicy> insurancePolicyList = InsurancePolicy.findAllByBookType(bookType)

    return InsurancePolicyCondition.createCriteria().count {
        if (insurancePolicyList) {
            'in' 'insurancePolicy', insurancePolicyList
        }
        if (companyList) {
            'in' 'company', companyList
        }
        if (departmentList) {
            'in' 'department', departmentList
        }
        if (subdivisionTypesList) {
            'in' 'subdivision', subdivisionTypesList
        }
        if (regionList) {
            'in' 'region', regionList
        }
        if (divisionList) {
            'in' 'division', divisionList
        }
    }
}
How to **detect** it?
- Code-reviews

**How do we fix it?**
- Teach developers their tools, languages,…
- Mentoring
- Buy books, send them to conferences,…

**Why does it matter?**
- Write clean and maintainable code
- Help your team grow
With great power comes great responsibility
Do you want to learn more?

- http://guides.grails.org/
- http://grails.org/community.html
- https://grails.org/documentation.html
UPCOMING EVENTS AND TRAINING

Events:

- objectcomputing.com/events

Training:

- objectcomputing.com/training
- grailstraining.com
- micronauttraining.com

Or email info@ocitraining.com to schedule a custom training program for your team online, on site, or in our state-of-the-art, Midwest training lab

© 2018, Object Computing, Inc. (OCI). All rights reserved.