[Development Project-specific Anti-Patterns](https://wiki.eisgroup.com/display/CRC/EIS+Development+Project-specific+Anti-Patterns)

The purpose of the document is to enlight the anti-patterns found in some EIS projects during code reviews. Developers should avoid such anti-patterns and adopt correct solution.

## **1 - General anti-patterns**

1. Creating services without interfaces.
   1. public class ProjectInstalmentServiceImpl {
   2. Use IoC and reduce coupling.
2. Injecting implementations instead of interfaces
   1. @Autowired private ProjectInstalmentServiceImpl  ProjectInstalmentServiceImpl;
3. Usage of "magic numbers"
   1. Days.daysBetween(pendingTransactionDueDate, nextInstallmentDueDate).getDays() <= 3
   2. Create configurable Spring property instead. It could be changed using PropertyConfigurerEntity or even via Admin application

## **2 - Storing mutable data in object cache**

**Problem statement**: Mutable objects are stored in object cache

**Consequences:** Mutable objects can be changed by different threads. For example, there is DAO:

|  |
| --- |
| **public** **class** ABCBillingPaymentPlanJpaDao **extends** BillingPaymentPlanJpaDao  {      @Transactional( readOnly = **true** )      @Cacheable(modelId = BaseCache.LOOKUP\_CACHE\_MODULE\_ID)      @Override  **public** List<BillingPaymentPlan> findBy( Map<String, Object> parameters ) { ...  }}    Thread 1:  **final** List<BillingPaymentPlan> plan = dao.findBy(…);plan.add(**new** BillingPaymentPlan());    Thread 2:  **final** List<BillingPaymentPlan> plan = dao.findBy(…); |

As a result Thread 2 gets incorrect data. The scenario is:

1. Thread 1 calls dao.findBy
2. Spring cache is caching the result (the reference to list of BillingPaymentPlan)
3. Thread 1 updates list of BillingPaymentPlan
4. Spring cache holds reference to the same object, thus object in cache is also updated (the list is not immutable)
5. Thread 2 calls dao.findBy

Spring cache returns updated list of BillingPaymentPlan to Thread 2, while Thread 2 expected only list of BillingPaymentPlan returned from DB query.

**The solution**

**Use immutable objects for caching. For example above we need to define immutable DTO for storing BillingPaymentPlan data and Collections.unmodifiableList(…) for wrapping the result list to prevent adding/removing elements:**

|  |
| --- |
| **public** **class** ABCBillingPaymentPlanJpaDao **extends** BillingPaymentPlanJpaDao  {      @Transactional( readOnly = **true** )      @Cacheable(modelId = BaseCache.LOOKUP\_CACHE\_MODULE\_ID)      @Override  **public** List<BillingPaymentPlanDTO> findBy( Map<String, Object> parameters ) {          ...  **return** Collections.unmodifiableList(result);      }  } |

## **3 - “if-else” in implementation of product/state specific behavior**

**Problem statement**: Implicit (by using product/state code) or explicit (by checking some coverage that exists only in one specific state) checking is used for implementation product/state specific behavior. For example:

|  |
| --- |
| PolicySummary policy = getPolicy();  ComponentInfo rootObject = (ComponentInfo) quoteLifeCycleService.rate(getPolicy());  **if** ("ABC\_CSA".equals(policy.getProductCd())) {      discountRatingService.ratePolicy( policy, Collections.<String>emptyList() );  } |

**Consequences:**Using if-else instead of polymorphism is common anti-pattern in OOP that leads to the code which is hard to maintain. Polymorphism gives you many advantages. The biggest gain occurs when the same set of conditions appears in many places in the program. If you want to add a new type, you must find and update all the conditionals. But with subclasses you just create a new subclass and provide the appropriate methods. Clients of the class don't need to know about the subclasses, which reduces the dependencies in your system and makes it easier to update.

**The  solution**: For product specific logic, a Product Aware Bean facility should be utilized: [Spring bean override during runtime (execution context aware proxies)](https://wiki.eisgroup.com/pages/viewpage.action?pageId=99029251).

## **4 - Business logic in View Model**

**Problem statement**:View Model contains business logic (does operations with entities and/or DAO).

**Consequences:**View Model is part of Presentation Layer, so business logic in it breaks separation responsibilities between layers and decreases maintainability of the project.

**The solution:**If you need invoke some business logic from component view, move the business logic into corresponding Service and invoke it from [View Model](https://wiki.eisgroup.com/display/CRC/%5BProductFactory%5D%5BComponent%5D+Product+Runtime+Invoking+Services#[ProductFactory][Component]ProductRuntimeInvokingServices-Serviceinvokingfromcomponentview) or [External Bean](https://wiki.eisgroup.com/display/CRC/%5BProductFactory%5D%5BComponent%5D+Product+Runtime+Invoking+Services#ProductRuntimeInvokingServices-Externalbeaninvoking).

## **5 - Business/application logic in DTO**

**Problem statement**: DTO contains any kind of logic (validation, transformation, etc).

**Consequences:**From DTO pattern description: “The difference between data transfer objects and business objects or data access objects is that a DTO does not have any behavior except for storage and retrieval of its own data (accessors and mutators)”. So include of any kind of logic in DTO breaks the DTO pattern and also it breaks Single Responsibility Principle.

**The solution:**Create DTO only with getters and setters.

## **6 - Constants duplication**

**Problem statement**: The same constant has multiple definitions.

**Consequences:**See the [well-known problems of the duplicated code](http://en.wikipedia.org/wiki/Duplicate_code#Problems_associated_with_duplicate_code)

**The solution:**

1. Module-specific constants should be collected in class like com.exigen.some-module.Constants.
2. Project-specific constants should be collected in classes of abc-utils module.

## **7 - Synchronization in business code**

**Problem statement**: Business code (a service, a view model, a component, a DAO, etc.) that contains any synchronization (synchronization primitives, classes from java.util.concurrent.\*, ThreadLocal variables, Thread creation and management, etc.).

**Consequences:**This is forbidden because of the following:

* Such code can lead to critical performance degradation, because it affects threads of application server.
* Synchronization doesn’t work between servers in a cluster. So it is quite often case when synchronized code works perfectly on single server cluster, but it creates a total mess or even corrupts business consistency of a DB when it is executed on multi-server cluster.

So, synchronization in business code leads to bugs that appear irregularly and that is difficult to reproduce.

**The solution:**

1. If instance of your class can be invoked by different threads, the class should be either stateless or immutable. If it cannot – don’t share the instance between threads, create a new one for each thread and pass it as parameter to its consumers.
2. If you can't implement some feature without using java synchronization, please ask a responsible architect to help you resolve this issue. All multi-threading must be moved into frameworks and should be in the responsibility of architect team.

## **8 - Checking if a changeset was executed  within it**

**Problem statement**: Checking if liquibase changeset was executed inside it itself:

|  |
| --- |
| <changeSet id="some\_id" author="some\_author">  <preConditions onFail="MARK\_RAN">  <not>  <changeSetExecuted id="some\_id" author="some\_author" changeLogFile="the same file name"/>  </not> |

**Consequences:**The check won't have any effect.

## **9 - Extending base entity with adding only one transient field**

**Problem statement**: Extending base entity by adding only one transient field, which themselves seems strange.

|  |
| --- |
| @Entity **public** **class** IaahEmailInfo **extends** EmailInfo {@Transientprivate String emailStatus;// !!! with filling it in run-time from other entity:ContactsAndExpressConsentEntity consent = IaahCustomerUtils.extension(customer).getContactsAndExpressConsent();  **if** (consent != **null**)  {      info.setEmailStatus(consent.getActivationEmailStatusCd());  } |

with only one purpose to show this transient field on ui:<e:entityWidget id="emailInfo#{postfixId}\_#{emailCounter.index}"

|  |
| --- |
| value="#{email}" template="multiEdit.xhtml" disabled="#{inquiry}" bundle="#{email\_messages}" seqInd="#{emailCounter.index}"> <e:showFields>type emailId emailStatus -- **this** is only one added **transient** field </e:showFields> |

**The solution**: Instead of filling this field one could show value directly from entity were it is stored.

|  |
| --- |
| <e:entityWidget id="emailInfo#{postfixId}\_#{emailCounter.index}"value="#{email}" template="multiEdit.xhtml" disabled="#{inquiry}" bundle="#{email\_messages}" seqInd="#{emailCounter.index}">      <e:showFields>type emailId </e:showFields>      .. show fields from EmailInfo      </e:entityWidget>  <!--Here we are showing on UI field were is stored email status directly from related entity-->  <e:entityWidget id="emailInfoStatus#{postfixId}\_#{emailCounter.index}"value="#{selectedCustomer.contactsAndExpressConsent}"template="multiEdit.xhtml" disabled="#{inquiry}" bundle="#{email\_messages}" seqInd="#{emailCounter.index}">  <e:showFields>      activationEmailStatusCd  </e:showFields> |