**Fake immutability**

Fake immutability for a class can be achieved in three ways:

1. Having non-final parameters with a package default visibility. Other classes in the package can change the state of the class by accessing its non-final parameters.
2. The class implements a read-only interface. Any reference to this class is done through that interface. Other classes having a reference to the class can change its state but those having a reference to its read-only interface cannot.
3. Providing a one-time setter. Once the getter is called a flag is set that would avoid setting the property again.

**Partial immutability**

A class that has only one field than can be changed whenever the side-effects are predictable and kept under control.

**Why ClassName.Params?**

If a class uses a public final Params field, the class is basically ready for Rest API calls. The constructor of the class must be private and accept a Params object. The builder should also accept a Params object.

**VoltageDrop class**.

This has been the pilot class so far~~. It should provide static methods to conduct all the calculations in only one class, with only static methods~~. That same class could have a Params and Results static inner classes. ~~Another different class should be~~

**Class Conductor.**

**Problem:**

* A conductor needs to know if it belongs to a conduit or to a bundle. So, if the conductor is immutable, it means it needs to know in advance which conduit or bundle contains it. So, the conduit must be created first to be passed as a parameter to the conductor constructor. But the conduit needs also to know which conductor to add. Who is created first? The conductor must be created first, and later assigned the conduit. The conductor will be partially immutable. The methods leaveConduit and leaveBundle must be used.
* We need to be able add a conductor to a conduit (or bundle) once the conduit and the conductor are created. Changing the container of a conductor (conduit or bundle) affects the results of what the conductor calculates. There might be a couple or results fields that do not change but must of them change.
* Since the class conductor contains a reference to the conduit or bundle, that reference must be immutable. So, it should be impossible to change the state of the bundle or the conduit from inside the conductor class. The bundle class is already immutable, so this requirement is met.
* Since the class bundle and conduit have a reference to conductor objects, they can change the state of the conductor objects. In fact, when a conductor in added by the bundle object, somehow the bundle must tell the conductor: “hey, you are part of me now”, and so the conductor can compute and update the results fields that are mutable (to be determined). The class conductor and the class cable must have a read-only interface that is shared among all other classes except bundle and conduit. For these two, the reference must be the mutable conductor. But keep in mind that only the conduit and bundle field can be altered from inside the bundle or conduit class.

**Solution**:

* The bundle and conduit class should not use the conductor passed as parameter. They should make a copy of it, and while doing so, set the conductor's bundle or conduit fields. This way, the conductor class can perfectly be immutable.
* Bundle and conduit classes should provide a method to return ~~the first conductor, the last conductor, and~~ a list of conductors by its tag (or groupId), which will turn out to be the circuit number later.
* They should also provide a method to construct a bundle or conduit with N conductors at a time. For example, “add(conductorHot, 5)” to add 5 copies of the conductorHot. ~~Also, instead of passing a conductor to the conduit, consider passing a Conductor.Params object, which is immutable but must have their fields publicly accessible~~. We are somehow asking the bundle to build the conductors for us. ~~I should implement all these ways of building a conduit or a bundle. The Conduit params~~
* The conductor class should use an integer tag to identify the conductor. No verification of said tag is required. If the user does not provide a tag when creating a conductor, it will default to zero. Several conductors can have the same tag. This means a tag can represents a group. So, why not just name it group id instead of tag? I like it!
* With this approach the bundle, conduit and conductor classes can be truly immutable.

**Construction**:

* This class must check its creational parameters to avoid undesired state. Since this is an immutable class, all parameters are set at construction time only. That is the moment where all parameters must be verified for validity. ~~We do not want a conductor with null values for its size, metal, copper coating or any other class instance field~~. So, a ResultMessages object is needed as part of the conductor fields, to report any invalid parameter. The client app can know if a parameter was invalid by checking if ResultMessages has errors or warnings. This is especially useful for Rest API. But for standalone applications throwing exceptions is the preferred approach.
* Using any invalid parameter during construction should add an error to the ResultMessages object ~~and a default value should be used~~. The caller must check if there are errors or warning messages.
* Since this software is intended to be used as a Rest API, we are using the ResultMessages approach which includes providing default parameter and error or warning messages. This way, our Conductor object is always created in a consistent state even though that state might not be the one the client wanted to create. Having a consistent conductor object makes it unnecessary to make a lot of verifications for nullity or for wrong parameters, since the conductor object will always be in a consistent state.
* Fgf

GENERAL DESIGN RULES

* Why do I need Params? To quickly receive a JSON string to be used to create the object.
* Why do I need Results? To be converted to a JSON string and send it back to the client.
* Params and results do not need getters. They are final and public. No need for them.
* Params contains the parameters that where necessary to create the object.
* Resulting objects contained should be placed inside Results:
  + For example, VoltageDrop.Results contains resultMessages (which should be able to be converted to JSON).
  + A bundle of conductors is a container that is also part of the Results object. It’s not a param (by convention).
* Params, and its fields must be public final. Its constructor must be private. Must implement toString using JSON.
* Results , and its fields must be public final. Its constructor must be private. Must implement toString using JSON.
* Both Params and Results must have a private constructor with gazillions of parameters to construct them.