Q4: Code review challenge

For the purposes of best practice as a software engineer, one ought to abide by the SOLID principles when writing code. This acronym stands for:

S – segregation principle

O – open/ closed principle

L – Linkov substitution

I – interface segregation

D – dependency inversion

I will use this to analyse the given piece of code. I will write my recommendations as a set of bullet points.

1. The \_\_init\_\_ method has too many attributes. This method is for the necessary requirements of a class. I would remove the is\_active attribute because is is not necessary to know if your employee is active or not for a class called Employee (not EmployeeActiveness) – so we could have a child class with this attribute instead. I initially thought that the db\_engine in there was unnecessary but it seems that this has already been discussed as a disclaimer.
2. Overall, the class is huge! It should not be this big and complex. Too many methods. This goes against the segregation principle because each class should have a single role. This class has more than one role.
3. In order to abide by interface segregation (which tells us that many interfaces are better than one general interface) we need to split this general class into classes and subclasses.
   1. We can do this by setting up update\_departement, update\_status, save\_employee, remove\_employee and print\_employee as subclasses
4. Since they have now been set as subclasses this means that they can inherit from the parent class called Employee (as they all seem to require the attributes from there)
5. Dependency inversion askes that we rely on abstraction as opposed to subclasses. And abstraction is this idea of only ‘showing’ what is necessary. We can adhere to this principle by having the subclasses inherit from the parent class.
6. The open/ closed principle tells us that code should be open for extension but closed for modification. I noticed that in update\_status there is the attribute new\_is\_active to signify that an employee is active (?). But what if an inactive employee joins? Then you would have to modify this piece of code to include an option called new\_is\_inactive, for example. But modification is not allowed by the open/ closed principle. Thus, what can be done is to create a new class called ‘activeness’ then have two child classes called ‘active’ and ‘inactive’.
7. I think that the same argument can be used with the print\_employee\_report. Suppose in the future if I not only want to *open* files but I want to *read* or *write* them too. But this method only gives the open to open. We need to adhere to the open/ closed principles and perhaps set up a class called ‘access\_employee\_info’ with subclasses called ‘print’, ‘read’, and ‘write’ for example.
8. Linkov substitution states that we should be able to ‘swap’ a parent and child class without any major effects to our code. But this principle has not been applied in this code. Firstly, because there are no child classes. If this was to be done again one might consider having a subclass ‘def Department(Employee)’ with attributes: self, department. This would contain the attributes from employee as it is a child class. And it would thus satisfy the Linkov substitution.