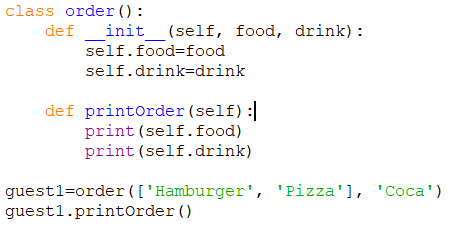
SOLID – Homework

1. Single responsibility principle

Suppose we need to develop and application to print out the customer’s order in a restaurant.

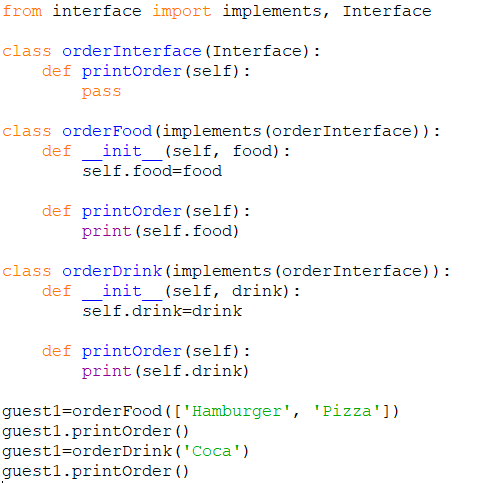
The violated example will look like this:



The problem is: What if the customer only order food or drink and not both?

To fix this problem, I think we should create two separate classes: orderFood and orderDrink. These classes will implement an interface to make sure that they will print out the order.

The fixed example should look like this:

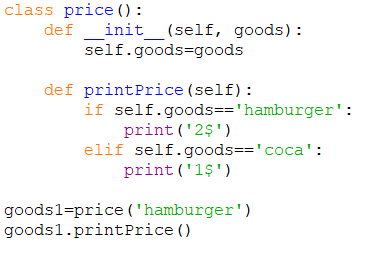


By doing this, the orderFood class will focus on printing out the food order and the orderDrink class will focus on printing out the drink order. Moreover, the customers are not forced to order both food and drink, they can order food or drink or both as their wishes.

1. Open-closed principle

Suppose we need to develop and application to print out the goods’ price in a restaurant.

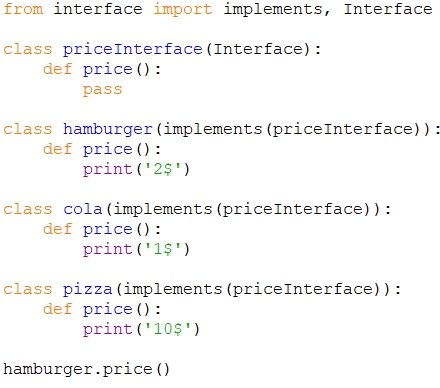
The violated example will look like this:



The problem is: What if the restaurant have a new goods called pizza? Do we have to modify the price class?

That fails the Open-closed principle. To fix this, I think we should apply class for each kind of goods, these classes will implement a priceInterface.

The fixed example should look like this:

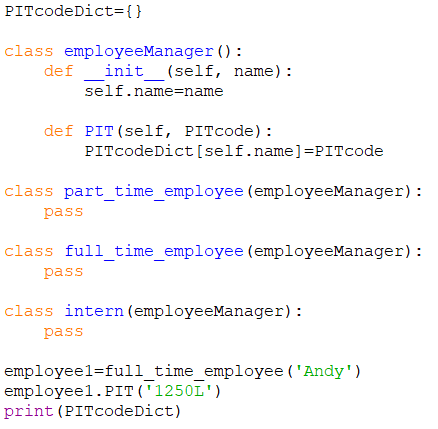


By doing this, when the restaurant have new goods to offer, we can add a new class to print out the goods’ price, so we don’t have to modify any of the existing class.

1. Liskov substitution principle

Suppose we need to develop an application to gather Personal Income Tax code (PIT code) from the employees.

The violated example will look like this:



The problem is: Interns don’t have PIT code! So the PIT function in this case does nothing for intern class.

To fix this, I think we should create a PITInterface and do not implement it in the intern class.

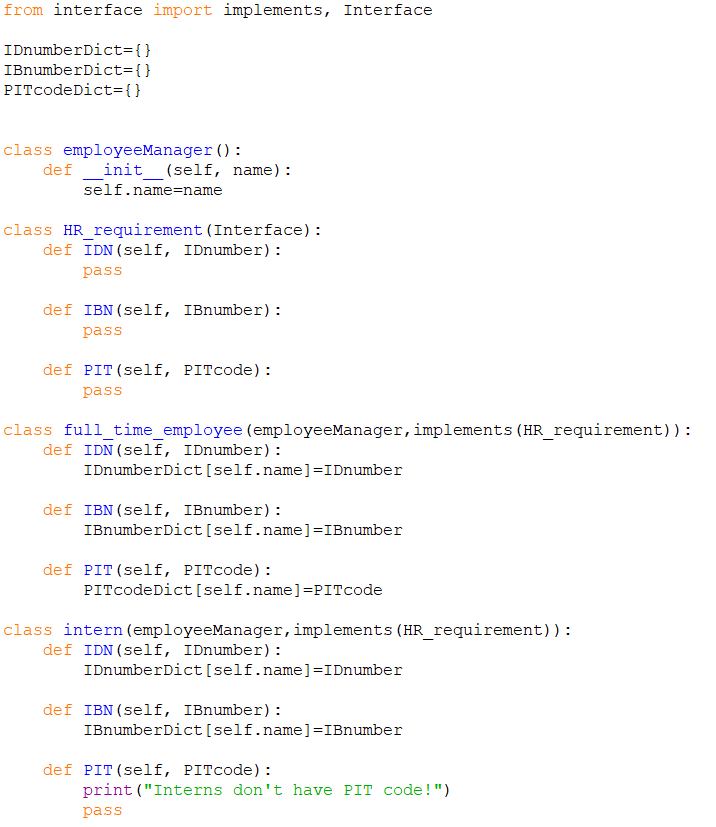
The fixed example should look like this:



1. Interface segregation principle

Suppose we need to develop an application to gather Identity Card number (IDN), Insurance Book number (IBN) and Personal Income Tax code (PIT code) from the employees.

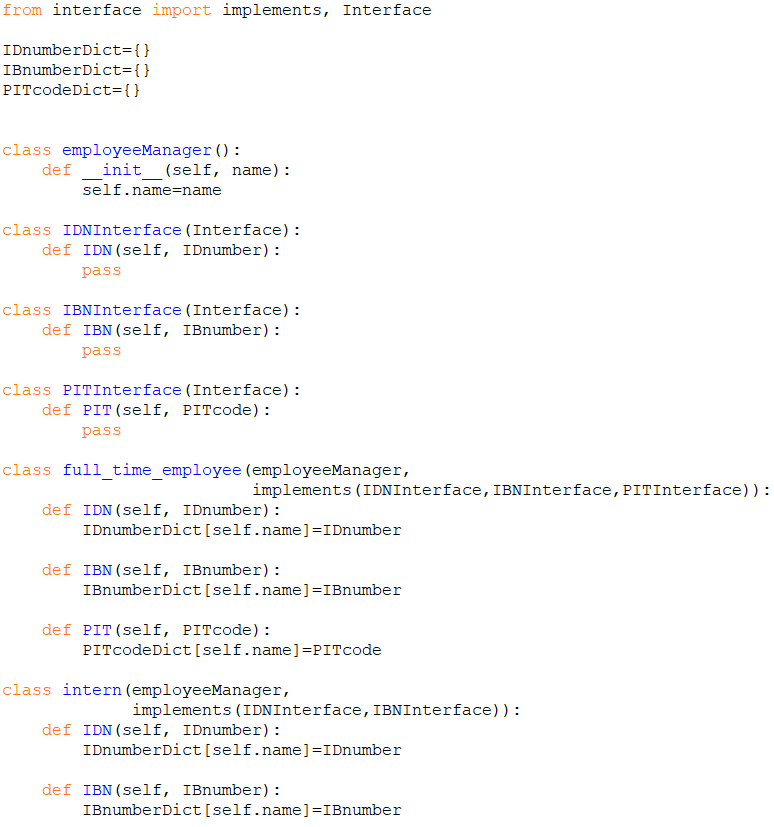
The violated example will look like this:



The problem is: Again, interns don’t have PIT code! We are forcing the intern class to define PIT function.

To fix this, I think we should segregate the HR\_requirement interface into three small interfaces and implement them in suitable classes.

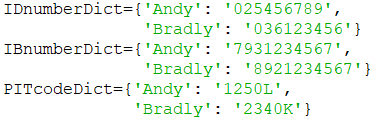
The fixed example should look like this:



By doing this, the intern class don’t have to define unwanted function.

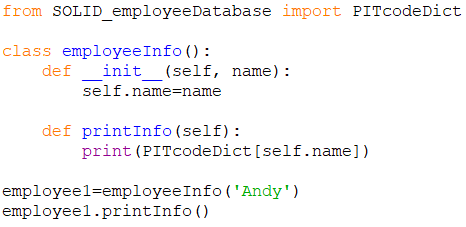
1. Dependency inversion principle

Suppose we have a file named SOLID\_employeeDatabase like this:



Suppose we need to develop an application to print out the Personal Income Tax code (PIT code) of some employees.

The violated example will look like this:

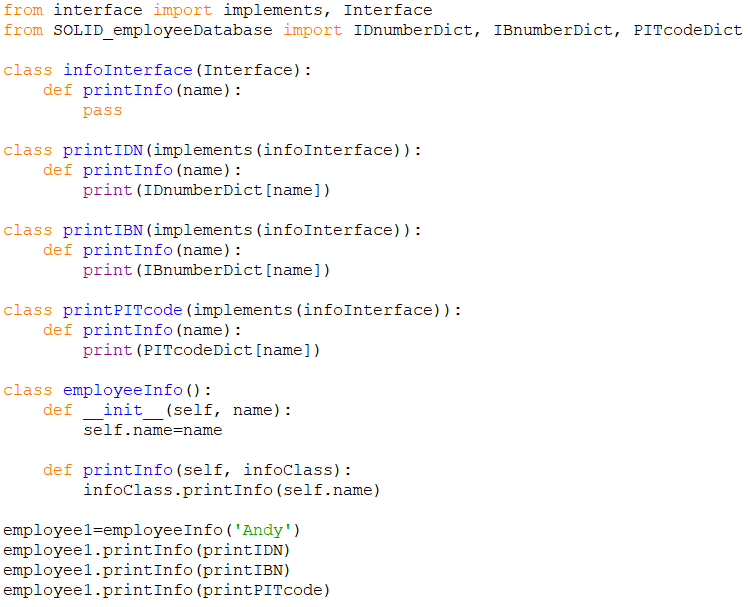


The problem is: What if someday we want to print out Identity Card number (IDN) instead of PIT code? Do we have to modify the employeeInfo class?

To fix this, I think we should creat a separate class for each kind of number (IDN, IBN and PIT code), each class will print out the corresponding number, these classes will implement the infoInterface to make sure they have the print out function.

The printInfo function of employeeInfo class will determine which number do we need to know. We can do this by inserting the name of the class corresponding to the number we need.

The fixed example should look like this:



By doing this, we can get the information we need (IDN, IBN or PIT code) but don’t have to modify the classes.