

Assignment 2

SE 450: OO Software Development

Design Principles and Design Patterns

Instructor: Vahid Alizadeh

Email: v.alizadeh@depaul.edu

Quarter: Winter 2021

Manoj Kumar Vemuri

Student ID:2022213

Exercise 1:

- Sol:

Here the Tiger extends Feline, then we can presume that it must be the case that all Tiger's are Feline's. However, here the run() method in the Tiger is private, i.e., inaccessible to the outside, whereas the run() method in Feline is public. As the Tiger's do not have a public method run(), like the Felines, they are not felines.

The given code violates the Liskov substitution principle which states that " *Objects in a program should be replaceable with instances of their subtypes without altering the correctness of the program*".

Exercise 2:


- Sol:

In the given code, we have a *worker* interface with the methods *work* and *eat* which are both implemented by the *Worker* class and *SuperWorker* class. Both the workers and Super workers work and eat. The work that they do and when they eat is managed by the Manager class. It is also stated that the company brings some robots which work but don't eat.

Considering the given code and the information, we can see that the interface is 'fat', i.e., the interface forces options that are not used. Robots work but do not need to eat. Thus there is an unnecessary dependence in the Robot class. This violates the Interface Segregation Principle.

Using the principle, we can correct the violation by using two interfaces, to make the settings optional.

An abstraction of the optimized code can be as shown here,



```
interface WorkerInterface extends EatsInterface, WorksInterface {}

interface WorksInterface {
    public void work();
}
interface EatsInterface {
    public void eat();
}
class Worker implements WorksInterface, EatsInterface {
    public void work() {}//Work
    public void eat() {}//Eat
}

class SuperWorker implements WorksInterface, EatsInterface {
    public void work() {}//More Work
    public void eat() {}//Eat
}

class Robot implements WorksInterface {
    public void work() {}//Work
}

class Manager {
    WorkerInterface worker;

    public void setWorker(WorkerInterface w) {
        worker=w;
    }
    public void manage() {
        worker.work();
    }
}
```

Exercise 3:

- Sol:

The Code for Exercise 3 is placed in the folder Q3 under the files SingletonAssignment2.java, Assignment2Driver.java and the Q3UML.png .

Exercise 4:

- Sol

The solution to exercise 4 is under the folder 4 which consists of the files Engine.java, Frame.java, Kitchen.java, motorhomedriver.java , motorhomeabstractfactory.java, Style.java, Type.java, typeaengine.java, typeafactory.java, typeaframe.java, typeakitchen.java, typeastyle.java, typebengine.java, typebfactory.java, typebframe.java, typebkitchen.java, typebstyle.java, typecengine.java, typecfactory.java, typecframe.java, typeckitchen.java, typecstyle.java