Brian Salinas
SE 350 ASYNC
Prof. V. Alizadeh
Assignment 3

Exercise 1.

The classes do not follow the SOLID design principle. First by the Single Responsibility Principle the class Feline and Tiger have the methods run and eat. The class should only have one of these methods maybe by creating a separate Diet class for eat. The other issue is the doubling of methods between Feline and Tiger where both have methods called “run”. If run is a method that is a universal characteristic of Feline then it should stay as a part of Feline and not reimplemented as part of Tiger. This would allow the method of camouflage being unique to the class Tiger to be appropriate for SOLID design principle.

Exercise 2.

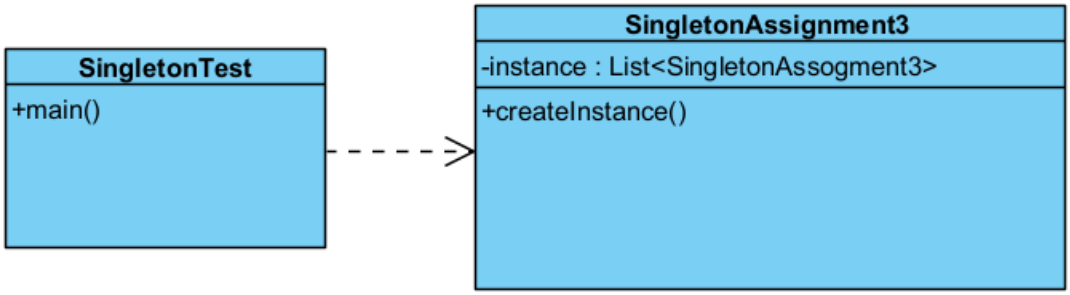
This code violates the Interface Segregation Principle. If we are to implement robots that work but do not eat then we should create a separate human interface that adds eat(). This gives us flexibility if we get different types of robots, since none of them would need to eat, much like we have multiple types of workers. One way to do this is have a worker interface that only has work();

```
interface WorkerInterface {
        public void work();
}
interface HumanWorkerInterface extends WorkerInterface {
        public void eat();
}
class Worker implements HumanWorkerInterface{
        public void work() { }
```

```
        public void eat() { ]
}
class SuperWorker implements HumanWorkerInterface{
        public void work() { }
        public void eat() { }
}
Class RobotWorker implements WorkerInterface{
        public void work() { }
}
Class Manager {
HumanWorkerInterface worker;
Public void setWorker(HumanWorkerInterface w) {
worker = w;
}
public void manage() {
worker.work();
}
}
```

Something extra we could do is make a separate robot interface so that we can implement specific methods that only robot workers would use and need.

Exercise 3.



Code found in folder Q3

Exercise 4.

Code found in folder Q4