Wyett Considine
Tinie Doan

Project 2: Pet Store - Part One

1. There are three things that make Java interfaces unique from a traditional OO interface design:
   I. Abstraction: allows coder to hide unnecessary information

```
1   abstract class Vehicle {
2       String color;
3                   Vehicle.Vehicle(String color)
4       public Vehicle(String color){
5           this.color = color;
6       }
7       public String getColor() {
8           return this.color;
9       }
10
11      // this method is abstract, hasn't yet be defined
12      public abstract drive();
13  }
14
15  class Motorcycle extends Vehicle {
16      // this subclass could override drive() method
17  }
```

   II. Polymorphism: allows coder to allow multiple implementations of a method

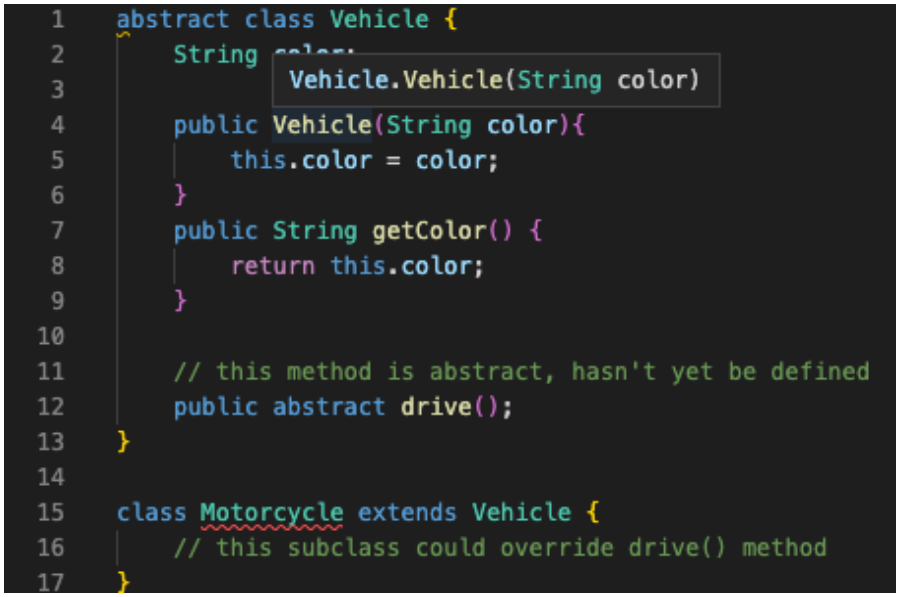
```
abstract class Vehicle {
    String color;

    public Vehicle(String color){
        this.color = color;
    }
    public String getColor() {
        return this.color;
    }

    public abstract drive();
}

class Motorcycle extends Vehicle {
    @Override
    public String drive() {
        return "On two wheels";
    }
}

class SemiTruck extends Vehicle {
    @Override
    public String drive() {
        return "On 10 wheels";
    }
}
```

   III. Multiple Inheritance: allows coder to create a multi-level hierarchy
      A. A good example of this is the Item class, Pet class, and Dog class from this project. The Dog class inherits all traits and methods from the upper two hierarchies.

2. Because abstraction and encapsulation are both focused on hiding code from unwanted accessors, they are often easily confused. The difference between the two is that abstraction focuses on convenience while encapsulation focuses on security. Basically, abstraction only provides that user with information and data that is relevant to know. For example, a person using a lamp doesn't need to know exactly what is going on inside the lamp, just simply that they pull the string to turn it off and on. On the other hand, encapsulation is protecting certain data from the user, and can only be accessed by specific methods that can then access that protected data.

As stated by Javin Paul, abstraction is defined at the design level and encapsulation is defined during implementation (Paul, BlosSpot).

An example of Abstraction is:



An example of Encapsulation is:

```java
class Person {
    //these are private data that are kept hidden and inaccessible ot the user
    private String name;
    private String birthday;
    private int password;

    //public methods can be used to access this private info

    // getters
    public String getName() {
        return name;
    }
    public String getBirth() {
        return birthday;
    }

    private String getPass() { // private because users should'nt be able to access the password
        return password
    }


    //setters
    public void setName(String name){
        this.name = name;
    }
    public void setBirth(String birth){
        this.birthday = birth;
    }
    public void setPass(String pass){
        this.password = pass;
    }
}
/* This class can be used by the user update certain information */
```

3. Draw a UML class diagram for the FNPS simulation described in part 2. The class diagram should contain any classes, abstract classes, or interfaces you plan to implement to make the system work. Classes should include any key methods or attributes (not including constructors). Delegation or inheritance links should be clear. Multiplicity and accessibility tags are optional. Design of the UML diagram should be a team activity if possible.