Antonio Scalfaro

03/14/2024

CMSC 335

Homework 1

1. This snippet of code is incorrect in a few manners, the first being that it calls a no-argument constructor, and one does not exist and therefore one must be added. Second, if the default value for int v is equal to 12 then it should not be instantiated as such on the field level, rather should be instantiated in the no-argument constructor. These changes should produce the outcome the designer is attempting to achieve.
2. This snippet is similar to the first, the field int v should not be instantiated to 12, rather there should be a no-argument constructor that sets that value as the default. Though the code should still work as the designer is expecting it to because a value is passed during the instantiation of the MyClassB instance and should thus have the correct field value for v.
3. This snippet also should change how the field value is instantiated along with adding a no-argument constructor, but the real issues lie in the signature declaration of the MyClassC, it is missing the key word public (or other equivalent keyword) and is therefore not reachable by the MyClassD main method. Adding public would fix this issue, adding private would correct the signature but still keep the class unreachable from MyClassD. This all said with the caveat that the public keyword can be omitted if the classes reside in the same package. Finaly, in the constructor, it is written incorrectly by stating int v = pV, which will cause a second instantiation of the variable v which would not be allowed. Int should be removed to correct this.
4. The same can be said for the next snippet as the previous in terms of the signature. This one adds the additional mistake of the constructor being private and thus even adding the public keyword to the MyClassF signature would still make the constructor unreachable. The signature of the constructor would need to be changed to public to be reached.
5. With this snippet the largest issue that I can see is that the Boolean being passed into the constructor is never used, since there is no field for it. That information will be lost after instantiation, the fix would be to add a Boolean field and set the field in the constructor, either the private or public one.
6. The provided hierarchy does not make sense because in terms of class hierarchies, the more detailed class should be lower on the hierarchy. Since a general would have all the same aspects of a private and then some, it would make more sense to place the general lower than the private.
7. Potential fields for the given hierarchy could look as follows:
   * String engineType
     1. Int numWheels, int kbbValue (Kelly Blue Book)
     2. Int tailNum
        1. Int maxPassengers
        2. Int numOfUniqueWeaponSystems
        3. Int maxPayload
     3. Int numStages
8. Potential methods for the given hierarchy could look as follows:
   * Boolean engineInWorkngCondition()
     1. Int totalTripMiles()
     2. Int maxFlightTime()
        1. Void loadPassengers()
        2. void engageAimLock()
        3. double maxRadiusOfArmamentBlast()
     3. double minThrustToReachEscapeVelocity()
9. An example of encapsulation and inheritance relationship could look like the following:
   * Class Gods
     1. Private String name;
     2. Public Getters/Setters
   * Class Greek extends Gods
     1. Public String domain
     2. Public Constructor(name, domain)
        1. Super(name), this.domain = domain

This is an example of both encapsulation and inheritance because the parent class Gods provides a private name field that is accessible only through getters and setters. The Greek class inherits the Gods class so it must use the name field and adds a new field, domain.

1. Class Tree
   * Private int age
   * Public getters/setters/constructor

Class OrangeTree extends Tree

* Public String fruitType
* Public getter/setters/constructor
  + Invokes super(age) in constructor

This is an acceptable parent child relationship between classes because Tree as the parent has a field that could be used to describe every type of Tree and the OrangeTree class that extends Tree provides more detail to what kind of Tree. OrangeTree still needs an age input but also adds fruitType as a field. Though it is a Tree, it is uniquely an OrangeTree by adding the fruitType field (which could also help to discern between OrangeTrees, i.e blood oranges).