**UML Assignment #2**

**1. How does your design implement the four pillars of OOP (abstraction, encapsulation, inheritance and composition, and polymorphism)?**

**Encapsulation** is implemented through the use of private and public attributes and methods. If we set instance variables and methods to public, we are essentially allowing other objects to interact with our given object *through* these attributes and methods (i.e. adding them to the public interface). As you can see in our design, we have strategically set all of our instance variables to private and all of our class methods to public.

With **Abstraction**, we believe that in order to achieve the desired task of adding a soldier from a particular division (JTF2 or CSOR) to the inventory where an operation can access it to create an operation team requires that all methods be public.

A more generalized form of composition – **Aggregation** – exists in our design. *Operation Room* “has-a” *SpecialForcesSoldier*; an Operation Room can have many Special Forces Soldiers but a Special Forces Soldier can only be in one Operation Room.

In terms of **Inheritance**, our child entities *CSOR* and *JTF2* inherit attributes and methods from the parent, abstract entity *SpecialForcesSoldier*; Special Forces Soldier “is-a” CSOR soldier or “is-a” JTF2 soldier.

And lastly, regarding **Polymorphism**, in the abstract entity *SpecialForcesSoldier*, we have abstract methods: *expire\_training* and *train* which raise not-implemented errors if called with the parent entity rather than the child entities. They’re not fully defined in the parent class; instead, these methods are overwritten in *CSOR* and *JTF2*.

**2. Why are your classes good abstractions (i.e., models) of the real-world entities they represent?**

Well, our classes include the *key* attributes and methods required in a class to accurately represent our objects. Take a look at our parent entity: S*pecialForcesSoldier*. We’ve given it the necessary attributes: a service number (SIN), a rank, the soldiers’ first and last names, what division (JTF2 or CSOR) they’re in, etc. Now take a look at the child entities (*JTF2* and *CSOR*), here we’ve added even more necessary attributes (distinguished by class): CSOR soldiers have a kill count and a call sign, JTF2 soldiers have a role and a mission number. All of this information is then used in methods to identify a specific soldier, add them to the soldier inventory where an operation can be made, and operation statistics can be set and retrieved in the *Operation* class. Again, these methods represent how our objects behave in the real-world (i.e. special force soldier gets deployed when assigned to an operation). Essentially, all of this detail is what makes our classes good abstractions of the real-world entities they represent.