Soldier/Zombie Simulation Design Document

I drawed a class diagram with UML to show my design more properly. Added it under doc folder as classdiagram.jpeg.

In this simulation our base class is SİmulationObject class which all objects in the simulation will derive from this abstract base class. The subclasses of this class is Soldier, Zombie and Bullet. Soldier and Zombie are abstract classes. Types of soldiers (Commando, RegularSoldier, Sniper) are another classes which extends Soldier class, Also, types of zombies (SlowZombie, RegularZombie, FastZombie) are all another classes which extends Zombie class.

Soldier class is the base class for soldiers. All soldier types extends this base class, so that, if another type of soldier wants to added, it needs only extend this base class. That is, soldier type is extendable. There are three state of soldiers and all have different step function derives also according to type. So, I write methods for these states in Soldier class, and subclasses implements or overrides these methods according to their states. I direct implemented some state methods in Soldier class to decrease code repetition. If subclass needs to step differently, it is overriding. In soldier class, step function, checks the state of the soldier, and runs the behaviour method according to it. Since soldier state is not extendable, there is no harm check their states with if/else structure.

Zombie class is the base class for zombies. All zombie types extends this base class, so that, if another type of zombie wants to added, it needs only extend this base class. That is, zombie type is extendable. There are three state of zombies and all have different step function derives also according to type. I defined these methods as abstract in Zombie class and implement them in subclasses. In zombie class, step function, checks the state of the zombie, and runs the behaviour method according to it. Since zombie state is not extendable, there is no harm check their states with if/else structure.

I implemented two class which are ZombieSpecificValues and SoliderSpecificValues. These are classes that contain constant values. ZombieSpecificValues contains constants for zombies only and SoldierSpecificValues contains constants for soldiers only. With these classes, we can change their constant values easily, and it also increases code readibility.

In SimulationController class, I have 5 ArrayList. 3 of them are for storing SimulationObjects seperatly, one of the is objectToAdd, other is objectsToRemove.

For storing SimulationObject in different class, before adding I need to find the type of the SimulationObject (Soldier, Zombie or Bullet) and add it correct array. I could be use instanceof but this is not good solution for OOP. So, I implemented visitor design pattern. I implemented one interface which contains three add functions, on efor each type. And there are two other classes which implements this interface. These two class for adding and removing from the arrayList in the simulationController. By implementing this classes, I do not need to check SimulationObject type, visitor does it form me.

If we evaluate my project according to OOP design principles, I declared all my variables private and implemented getter/setter methods for these variables. Also, declared my

method protected instead of public. These all minimizes the accessibility and limits the ripple effect. Also, I tried to avoid code repeating, writed method instead to. I tried to stick to open/closed principle. My abstract classes is open for extension, yet closed for modificition as I already explained above. I have also tried to stick to Liskov substitution principle. In my project, any subclass can be used wherever its base class is used. It also supports single responsibility property by implementing the property: a class should only have one responsibility. Furthermore, it should only have one reason to change. Also, I used the dependency injection pattern while writing my constructors.