Design Rationale for Going to Town and New Weapons: Shotgun and Sniper

There are a few classes which are related in implementing the new functions (going to town and new weapons) in this application, namely Vehicle, Direction, DrivingBehaviour, DrivingAction, Gun, Shotgun, Sniper, Ammunition, ShogunAmmunition and SniperAmmunition, ShootingAction class.

Each class has their respective responsibility. From the new functions and the classes added into the application, we can separate them into two parts. Vehicle, Direction, DrivingBehaviour and DrivingAction classes are created for Going to Town function, while the others are for New Weapon function.

Encapsulation is applied where inheritance is applied, such as Vehicle class inheriting Item class, DrivingAction class inheriting Action class, both ShotgunAmmunition and SniperAmmunition extending Ammunition class which extends PortableItem class, Shotgun and Sniper extending Gun class which extends WeaponItem class. All of the children class behave like their parents except they consist of a few methods which overrides the parent class' methods as well as they have additional functionality.

Abstraction is applied to all classes, where private and protected attributes are declared (information hiding), and getters and setters are declared when necessary.

Other than that, I prevented privacy leak by returning copy of object. One of the example is in ShootingAction class, where there are a few places that I return a shallow copy of String output, namely in checkSurvivors(), showShotgunSubMenu() and execute().

Moreover, I refactored the code to apply the DRY principle in ShootingAction class, where I separated the task of checking the condition of the attacked target actor from the execute() method to the checkSurvivor() method.