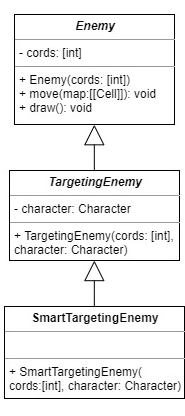
**Hierarchy Description**

We have decided to design *Enemy* as an abstract superclass because all of the types of enemies will have common attributes, like coordinates on the map, for example. Additionally, all enemies will be able to move and be drawn, hence why the ‘move’ and ‘draw’ methods are designed to be empty; Subclasses of *Enemy* will have to fulfil the method move otherwise the instances will not work. Empty methods make sure that certain aspects of superclasses are followed through and that is exactly what we wanted with our *Enemy* hierarchy. *Enemy* has three subclasses, all modelling different type of possible enemy: WallFollowingEnemy, StraightLineEnemy and *TargetingEnemy*.

*TargetingEnemy* inherits everything from its superclass, *Enemy*, yet is also abstract and keeps the move method empty for its own subclasses to fulfil. This class is a generalization of any type of enemy that will have to target the Character instance’s coordinates. This class will have two separate subclasses which will implement their own way of moving inside of the move method. *TargetingEnemy* has its own reference to a Character instance because all its subclasses will need to use the Character instance’s current coordinates in relation to their own position and the move method’s map parameter. There are two *TargetingEnemies*, DumbTargetingEnemy and SmartTargetingEnemy.

SmartTargetingEnemy is a root class that can produce actual instances. Any single instance of SmartTargetingEnemy should have, because of the inheritance from its hierarchy, a list of 2 integers referring to its coordinates, a reference to the Character object and implementation of move to fulfil its own way of finding which direction to move and change its coordinates. Using its own available information and a list of Cell objects making up a map passed to the method as a parameter, the SmartTargetingEnemy object will have to use a short path algorithm to find which direction would lead it to the Character object’s coordinates the fastest.

By placing these classes into an inheritance hierarchy, we get to generalise the aspects of all the possible enemies in the game and leave room for additions to the roster. Furthermore, efficiency of our code is increased as we will have less of it to maintain as well as bugfix in the future due to the reduction in duplicated sections. Moreover, it is essential for the abstract *Enemy* class to exist so that the runTurn method in the Level object has the ability to run through each instance of Enemy in its list and call each one’s move method without having to worry which type of enemy it is. This should simplify the process greatly.