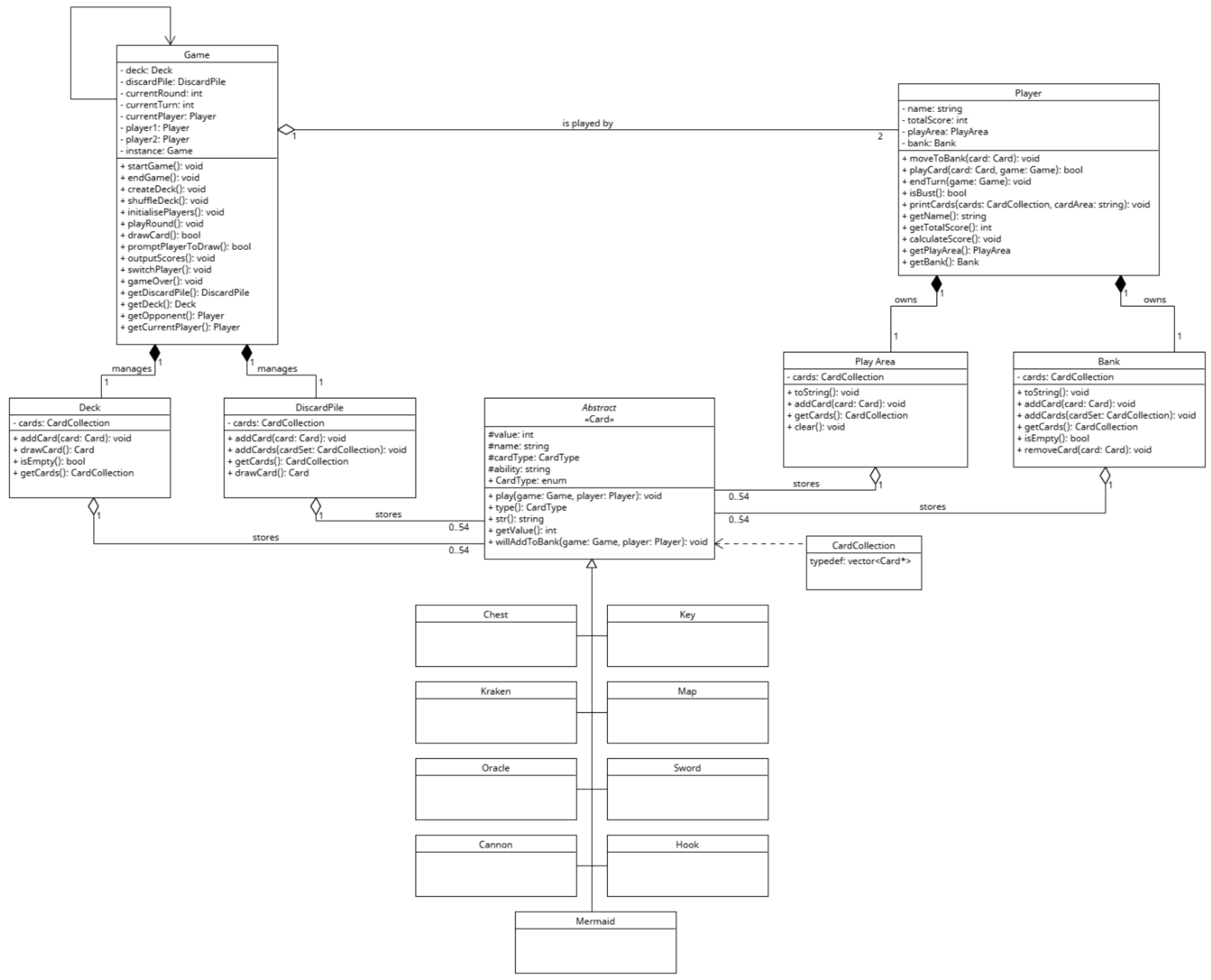
Design Document – Design Patterns with C++



<u>Please note – GIT repository is in the folder level above the code implementation.</u>

Using Singleton pattern for Game class. This way only one instance exists of the program and needs to exist while everything runs. Hence **instance: Game** and using **static Game* instance** in the Game class. getInstance has been implemented to adhere to Singleton pattern, where it either returns the existing instance, or makes a new one if none exist.

Each card type has its own specific class implementation. At this point it's a given, because they have their own crucial abilities, but in initial planning I was unsure if they needed their own classes or if an enumerator was enough. Because of the complexity of abilities and how they interact with the game, there is a base card class with 9 different suits as child classes.

I have made a small box to represent the Card Collection, because it is quite important to the function of each of the different types of vectors.

Using a deck (and play area, bank, discard pile etc) class – deck has a card collection that represents all of the cards. Initially I was torn between making classes for these or just using built-in vector methods. This would probably have worked, but the readability and responsibilities of each class and method would have been more complicated. Creating individual classes allowed for more flexibility and abstraction in implementation. One thing I might implement in future is using an abstract/base class for card collections like deck and discard pile, because I found they all utilised very similar methods and achieved the same functionality but were just used for different reasons.