Spiegazione Design Pattern utilizzati

Per lo sviluppo degli adapter MapAdapter, SetAdapter e ListAdapter (Il collectionAdapter non l'ho sviluppato in quanto ListAdapter implementa java.util.List che a sua volta estende java.util.Collection e quindi un oggetto della classe List si comporta anche come un oggetto di tipo Collection e quindi non occorre sviluppare una classe specifica) ho usato due design pattern della GoF che sono il pattern strutturale Adapter e l'iterator pattern.
L'adapter pattern permette di convertire un'interfaccia in un'altra che i clienti si aspettano, consente alle classi di lavorare insieme anche se hanno interfacce incompatibili.
Ho scelto di usare un object adapter perché il client usa l'Adapter come se fosse l'oggetto di libreria e l'adapter possiede il riferimento all'oggetto di libreria (detto Adaptee) e sa come invocarlo.
Quindi l'object Adapter(contiene l'adaptee) utilizza la composizione e può avvolgere classi o interfacce. Può farlo poiché contiene, come membro privato l'istanza dell'oggetto classe o dell'interfaccia che avvolge mentre il class Adapter(estende l'adaptee e implementa target) utilizza l'ereditarietà e può solo avvolgere una classe. Non può avvolgere un'interfaccia poiché per definizione deve derivare da una classe di base.
Ho usato l'object Adapter in ListAdapter su un oggetto di tipo Vector, in MapAdapter e SetAdapter su un oggetto di tipo HashTable.
L'iterator pattern l'ho utilizzato per quando il codice richiedeva l'uso dell'iteratore.
Per lo sviluppo del software ho usato la versione 2020.1.1 di IntelliJ mentre per i test la versione 4.13 di Junit.