1. Si consideri la gerarchia di classi delle esercitazioni precedenti che rappresenta **Shape**, **Circle**, **Rectangle**, **Square** e aggiungete l'implementazione, in ciascuna classe, dell'interfaccia **Comparable**. Il confronto tra forme sia fatto attraverso il valore dell'area. Verificate il corretto funzionamento con un programma che crea alcuni oggetti e li confronta.

2. Definite una classe **ListaCircolare<T>** per la gestione di una lista circolare di oggetti T. La classe dovrà prevedere metodi per inserire un nuovo elemento alla posizione corrente, per cancellare l'elemento corrente, per leggere l'elemento corrente e per spostarsi avanti e indietro.

3. Definite una classe **Vector<T>** per la gestione di un array dinamico di oggetti di classi che estendono **Number**. La dimensione del **Vector<T>** viene definita come parametro del costruttore (di default sarà 10) e il **Vector<T>** viene inizializzato con il valore 0. L'accesso ai membri deve avvenire mediante metodi **set** e **get** che verificano le dimensioni del **Vector<T>** ed eventualmente lanciano un'eccezione. Prevedete metodi per la somma, la differenza che restituiscono un nuovo **Vector<T>** contenente il risultato e che verificano che i **Vector<T>** su cui fare l'operazione siano della stessa dimensione (eventualmente lanciate un'eccezione). Prevedete anche un metodo che restituisce un **Vector<T>** ottenuto moltiplicandolo per un **Byte** e un metodo che restituisce il modulo (double). Utilizzate un'ArrayList per memorizzare internamente i dati e implementate l'interfaccia **Comparable<Vector<T>>** col metodo **compareTo** che restituisce -1, 0 oppure 1 in base al confronto dei moduli dei **Vector<T>**. Scrivete infine un programma per testare la classe.

4. Si consideri un **Mazzo** di carte siciliane. Ogni **Carta** è caratterizzata da un `seme` (`ORO, COPPE, BASTONI, SPADE`) ed un `valoreCarta` (`ASSO, DUE, TRE, QUATTRO, CINQUE, SEI, SETTE, DONNA, CAVALLO, RE`). Un **Mazzo** è composto da 40 carte. Utilizzare delle costanti statiche per rappresentare semi e valori di ogni carta. All'interno della classe **Carta**, prevedere un metodo `puntiBriscola()` che restituisce i punti di ogni carta nel gioco della briscola (`ASSO`=11 punti, `TRE`=10 punti, `DONNA`=2 punti, `CAVALLO`=3 punti e `RE`=4 punti, tutte le altre valgono 0 punti).

5. Nella classe **Mazzo** dell'esercizio precedente, prevedere un metodo per simulare il mescolamento delle carte. Tale metodo sceglie casualmente un punto del mazzo (un numero inferiore a 40) in cui spaccare il mazzo in due sottomazzi. Quindi le carte vengono riordinate in modo che le posizioni dispari siano occupate dalle carte del secondo sottomazzo e le posizioni pari da quelle del primo mazzo fino ad esaurimento del sottomazzo più piccolo. Poi si appendono le carte rimanenti nel sottomazzo più grande. Per es., se il primo sottomazzo contiene le carte da 1 a 10 e il secondo sottomazzo da 11 a 40 il mescolamento darà luogo a: 11, 1, 12, 2,..., 20, 10, 21, 22, 23....40. Tale procedura va ripetuta per un numero casuale di volte maggiore di 5 e minore di 10. Si visualizzi il risultato del mescolamento del mazzo.

6. In occasione di manifestazioni sportive, il proprietario di una casa noleggia posti auto nel suo vialetto di casa, che può essere rappresentato da una pila, con il consueto comportamento "last-in, first-out". Quando il proprietario di un'automobile se ne va e la sua automobile non è l'ultima, tutte quelle che la bloccano devono essere spostate temporaneamente sulla strada, per poi rientrare nel vialetto. Scrivete un programma che simuli questo comportamento, usando una pila per il vialetto e una per la strada, con numeri interi a rappresentare le targhe delle automobili. Un numero positivo inserisce un'automobile nel vialetto, un numero negativo la fa uscire definitivamente e il numero 0 termina la simulazione. Visualizzate il contenuto del vialetto al termine di ciascuna operazione.

NOTE PER COMPILAZIONE E TEST A RIGA DI COMANDO IN AMBIENTE LINUX:

`javac -d ../classes -cp ../classes nomeClasse.java` *compila e genera il bytecode*
`java -cp ../classes nomePackage.nomeClasse` *esegue il bytecode sulla JVM*