ITIS Belluzzi a.s. 2013/14	**Verifica di Informatica** **17/12/13**	Classe: 4Ai	Nome e cognome:

Testo A – Obiettivi minimi

1) – *(3 punti)* Si consideri seguente **interfaccia**:

```
/**
 * ADT stack, descrive le proprietà base della catasta (LIFO)
 * Le eventuali condizioni di errore vanno gestite producendo eccezioni uncatched.
 * @author giovanni.ragno
 */
public interface Stack<T> {
  /**
   * Inserisce un dato in cima allo stack
   * @param dato da inserire
   */
  public void push(T dato);
  /**
   * Restituisce ed elimina il dato in cima allo stack
   * @return il dato
   */
  public T pop();
  /**
   * Test se lo stack è vuoto
   * @return true se vuoto, false in caso contrario
  */
  public boolean isEmpty();
}
```

Si noti che l'interfaccia è leggermente diversa rispetto a quella vista a lezione.
Si chiede lo sviluppo di una classe che implementa esattamente questa interfaccia basandosi su di un array.

2) - *(2 punti)* **Inversione file**
Dato il file “elenco.txt” si vuole ottenere il file “inverso.txt” che contiene le stesse linee del primo file, ma nell'ordine inverso, cioè la prima linea di “elenco” sarà l'ultima di “inverso”, la seconda diventa penultima e così via fino all'ultima linea di “elenco” che sarà la prima di “inverso”.
Si chiede lo sviluppo di una applicazione TUI che risolve il problema utilizzando la classe del punto 1).

3) – *(2 punti)* **Riconoscimento palindrome**
Si richiede una applicazione TUI che data una stringa determina se è palindroma utilizzando la classe di cui al punto 1).

Testo B – Obiettivi avanzati

Si consideri seguente **interfaccia**:

```
/**
 * ADT insieme, presenta le proprietà tipiche dell'insieme: aggrega elementi
 * tutti dello stesso tipo, non esiste un legame posizionale fra gli elementi,
 * non è possibile avere elementi duplicati nell'insieme.
 * Sono definite alcune operazioni fra insiemi e fra elemento e insieme.
 * Eventuali eccezioni saranno di tipo uncatched
 * Nota Bene: <T extends Comparable<T>> implica che T dispone del metodo compareTo()
 * @author giovanni.ragno
 */
public interface Insieme<T extends Comparable<T>> {
  /**
   * Operazione per includere nell'insieme l'elemento passato come parametro.
   * Non sono ammessi duplicati nell'insieme, in caso esista già elem non si fa
   * alcuna azione, senza nemmeno produrre eccezioni.
   * @param elem il dato da inserire nell'insieme
   */
  public void includi(T elem);
  /**
   * Operazione per rimuovere dall'insieme l'elemento passato come parametro.
   * La richiesta di escludere un elemento non presente nell'insieme non produce
   * errore, semplicemente non comporta azioni.
   * @param elem il dato da rimuovedere dall'insieme
   */
```

```
  public void escludi(T elem);
  /**
   * Operazione che restituisce un insieme contenente gli elementi presenti sia
   * nell'insieme (this) sia in quello passato come parametro.
   * L'insieme (this) nè quello passato non vengono modificati.
   * @param ins il secondo insieme su cui operare
   * @return il nuovo insieme ottenuto dall'intersezione.
   */
  public Insieme<T> intersezione(Insieme<T> ins);
  /**
   * Operazione che restiuisce un insieme contenente gli elementi presenti
   * indifferentemente nell'insieme (this) oppure in quello passato come parametro.
   * L'insieme (this) nè quello passato non vengono modificati.
   * @param ins il secondo insieme su cui operare.
   * @return il nuovo insieme ottenuto dall'unione.
   */
  public Insieme<T> unione(Insieme<T> ins);
  /**
   * Operazione che restituisce un insieme contenente gli elementi che sono
   * presenti nell'insieme (this) e non sono presenti nell'insieme passato come
   * parametro. Dagli elementi presenti nell'insieme (this) cioè sono esclusi
   * gli elementi presenti anche nel secondo insieme.
   * L'insieme (this) nè quello passato non vengono modificati.
   * @param ins il secondo insieme su cui operare
   * @return il nuovo insieme ottenuto dalla differenza
   */
  public Insieme<T> differenza(Insieme<T> ins);
  /**
   * Operazione per determinare se l'elemento passato come parametro appartiene
   * all'insieme.
   * @param elem l'elemento da testare nell'insieme.
   * @return true se elem è presente nell'insieme.
   */
  public boolean appartiene(T elem);
  /**
   * Operazione per testare se l'insieme è vuoto
   * @return true se l'insieme non ha elementi.
   */
  public boolean isEmpty();
  /**
   * Restituisce un nuovo insieme che ha gli stessi elementi dell'insieme di partenza
   * @return il nuovo insieme
   */
  public Insieme<T> clone();
}
```

Si consideri poi il seguente **problema**.
Una associazione organizza una serie di cinque eventi. Il file di testo “soci.txt” contiene in ogni linea il cognome e nome dei soci. Non esistono omonimi tra i soci e in generale fra i partecipanti agli eventi.
I cinque file “presenti1.txt”,...,”presenti5.txt” contengono (per ciascuno dei cinque eventi) in ogni linea il cognome nome dei presenti.
Si desidera sapere:

a) per ogni evento il cognome e nome dei presenti che non siano soci;
b) il cognome e nome dei soci che non hanno partecipato a nemmeno un evento.

Quesiti:

1) ***(5 punti)*** Si richiede lo sviluppo di una classe che implementi esattamente questa interfaccia. L'implementazione può essere realizzata a partire da classi/tipi/strutture di base o basata su ADT già realizzato/i a lezione in questo ultimo periodo. In questo ultimo caso è necessario esplicitare la relativa dichiarazione degli elementi pubblici, mentre può essere omessa la sua implementazione.
2) ***(3 punti)*** Si richiede lo sviluppo di una applicazione TUI in grado soddisfare le richieste a) e b) del problema presentato. Lo sviluppo deve utilizzare la classe di cui al punto 1). Eventuali classi già sviluppate ed utilizzate a lezione nell'ultimo periodo potranno essere utilizzate senza il relativo sviluppo completo ma presentando comunque ogni relativo elemento pubblico.
3) ***(2 punti)*** Si analizzi quali altri metodi potrebbero essere aggiunti nella classe di cui al punto 1) per rendere più agevole lo sviluppo al punto 2). Si indichino le eventuali modifiche.