B1 – Gestione liste	**MODULO JAVA7**

(A) CONOSCENZA TERMINOLOGICA

Dare una breve descrizione dei termini introdotti:

- Sequenzialità fisica
- Sequenzialità logica
- Lista concatenata
- Classe **LinkedList**

(B) CONOSCENZA E COMPETENZA

Rispondere alle seguenti domande producendo anche qualche esempio

B1) Conoscenza

1. Quali sono i criteri di scelta tra *strutture statiche* e *dinamiche*?
2. Quali sono le differenze tra *array dinamico* e *lista concatenata*?
3. Come scegliere un dato *tipo di lista*?

B2)Competenza

1. Quali operazioni occorre implementare se si vuole realizzare un'applicazione che faccia uso di una lista di oggetti?
2. Come testare le singole classi?
3. A quale tipo di lista corrisponde una **LinkedList**?

(C) ESERCIZI DI COMPRENSIONE

1. In generale, quando non si conosce a priori il numero di oggetti che il programma dovrà gestire, possiamo utilizzare la tecnica detta …… …… che utilizza il segmento di memoria ……..
2. L'array dinamico è una struttura dati ibrida, da un lato simile all'array, in quanto le componenti presentano sequenzialità …….. e ……, dall'altro simile alle liste intese come strutture dinamiche, grazie al fatto che in esse si varia la ……. Durante l'esecuzione duna lista ……, e preveel programma.
3. In Java le liste sono già implementate nella classe ……., i cui oggetti si comportano come una lista ……., potendosi effettuare la navigazione in ……. i versi.
4. Indicare, per ciascuna delle strutture dati (S.D) riportate in tabella, se si tratti di una struttura dati statica o dinamica

Struttura dati	statica	dinamica	Struttura dati	statica	dinamica
lista			pila		
array			coda		
array dinamico			albero		
matrice					

5. Disegnare la UML di una classe **Nodo**, formata da un campo intero e da un campo riferimento (puntatore), con i metodi ritenuti necessari.

6. Disegnare la UML di una classe **Lista**, formata da celle di classe *Nodo*, definite nell'esercizio precedente.

7. Per ciascuno dei seguenti metodi della classe **Nodo** (avente attributi *info* di tipo stringa e *next* di classe **Nodo**) completare il codice e verificarne la correttezza:
   a. **public** node()
      { info=""; ….. = **null**; }
   b. **public** node (**String** str)
      { info = ……; next = ……..; }
   c. **public String** getInfo ()
      { **return** ……..; }
   d. **public void** setInfo (**String** str)
      { …… = str; }
   e. **public** node getNext ()
      { **return** ……; }
   f. **public void** setNext (node next)
      { **this**…… = ……..; }
8. Per ciascuno dei seguenti metodi della classe **Lista** (avente attributo *head* che indica il riferimento di accesso) e formata da celle di classe **Nodo**, completare il codice e verificarne la correttezza:
   a. **public** List()
      { head = …….; }
   b. **public void** insTesta(**String** str)
      { ..... c = **new** (str);
        c.setNext(……);
        head = ...;
      }
   c. **public void** printList ()
      { node c = ……;
        while (……)
        { **System.out.println** (……getInfo());
          c = ……;
        }
      }
   d. **public void** insCoda (**String** str)
      { node c = ……;
          if (……..)
            insTesta(……);
          else
            { **while**(……)
              c=c.getNext();
              node n =……;
              c.setNext(n);
            }
      }
   e. **public boolean** contained(**String** str)
      { node c=……..;
        while ((…….) && (……))
          c=c.getNext();
        return (c);
      }
9. Completare il seguente codice, che realizza la stampa sequenziale di una lista (*p* si assuma di classe **Lista** e rappresenta il puntatore di accesso; il metodo StampaLista si assume appartenente alla classe **Lista**):

```
public void StampaLista ()
{   Lista t=p;
    while (.....)
{
            System.out.println(……..);
            t=…….;
}
```

(D) ESERCIZI DI APPLICAZIONE

1. Scrivere la versione ricorsiva del metodo **Lunghezza**() che calcola la lunghezza di una lista.
2. Scrivere la versione ricorsiva del metodo ***int*** *contains (**String** str)* che stabilisce la presenza della stringa *str* nella lista.
3. Aggiungere alla classe **Lista** i seguenti metodi:
   a. **void** insertAt (**String** str, **int** n) /* inserisce str in posizione n */
   b. **void** deleteAt (**int** n) /* elimina in posizione n */
   c. **int** contains (**String** str); /* dà la posizione di str, se esiste */
   d. **node** contains (**String** str); /* dà il riferimento alla cella trovata*/

e. **int** count(); /* dà la lunghezza corrente */

4. Creare una nuova versione della classe **Lista**, implementandone i metodi in modo ricorsivo.
5. Per ciascuno studente si hanno a disposizione il cognome, il nome ed il voto. Si vogliono memorizzare i dati di un numero di studenti non noto a priori e si vuole stampare il voto massimo, quello minimo e il voto medio, con associato il cognome dello studente. Progettare l'applicazione relativa *voti.java* completa di analisi del problema, UML e codifica.
6. Aggiungere alla classe **Lista** il metodo **Insord** (**String** str) che inserisce gli elementi nella lista in modo ordinato, secondo una tecnica:
   a. iterativa
   b. ricorsiva.
7. Una matrice di interi si dice **sparsa** se si gli elementi diversi da 0 sono in numero molto ridotto. Individuare un modo per rappresentare efficacemente una matrice sparsa mediante lista e scrivere le seguenti procedure di gestione:
   a. caricamento dati
   b. ricerca di un elemento.
8. Scrivere un'applicazione che date due liste l1 ed l2 costruisca una terza lista l3, concatenando l2 in coda ad l1.

LinkedList

9. Ripetere l'esercizio precedente, usando però la classe **LinkedList**.
10. Tramite la classe **LinkedList**, realizzare un esempio di vocabolario bilingue.

(E) ESERCITAZIONI PRATICHE
Esercitazione n. 1

Obiettivi: utilizzo allocazione dinamica, attributi e metodi della classe **Lista** e della classe **Nodo**.
Problema: scrivere un'applicazione per la gestione di una lista di stringhe.

L'applicazione prevede la realizzazione di due classi: **Nodo** e **Lista**.

1) Creare la classe **Nodo** con il seguente schema UML:
2) Scrivere l'applicazione *testNode* per testare il funzionamento della classe e inserire le seguenti istanze di prova:
   a. Creare tre oggetti n1, n2, n3;
   b. Registrare una stringa in ciascuna cella;
   c. Stampare il contenuto *info* di ciascuna cella;
   d. Collegare *n1* con *n2*, *n2* con *n3* e impostare **null** nel campo *next* di *n3*;
   e. Stampare la lista ottenuto, navigando mediante un puntatore d'appoggio *t*;
3) Creare la classe *list* secondo la UML seguente:

Nodo
- **Stringa** info;
- node next;
+ node();
+ node(**Stringa** str);
+ **String** getInfo();
+ **void** setInfo(**String** x)**;**
+ node getNext();
+ **void** setNext(node next);

List	
- node head	Attributo
+ list()	Costruttore
+ **void** printList()	Stampa della lista
+ **void** insTesta (**Stringa** str)	Inserisce *str* in testa alla lista
+ **void** insCoda (**Stringa** str)	Inserisce *str* in coda alla lista
+ **void** insInterno (**String** str, **Intero** pos)	Inserisce *str* in posizione *pos*
+ **void** eliTesta ()	Elimina dalla testa
+ **void** eliCoda ()	Elimina dalla coda
+ **void** eliInterno (**Intero** pos)	Elimina posizioni interne
+ node contains (**String** str)	Dà il riferimento al nodo avente *str*
+ **boolean** contained (**Stringa** str)	**true** se la lista contiene la stringa *str*
+ **void** menu ()	Visualizza menu delle operazioni
+ **void** opera (**Intero** s)	Selezione ed esegue l'operazione *s*

4) Scrivere il *main* nella classe *list* e provare le varie operazioni di manipolazione della lista.