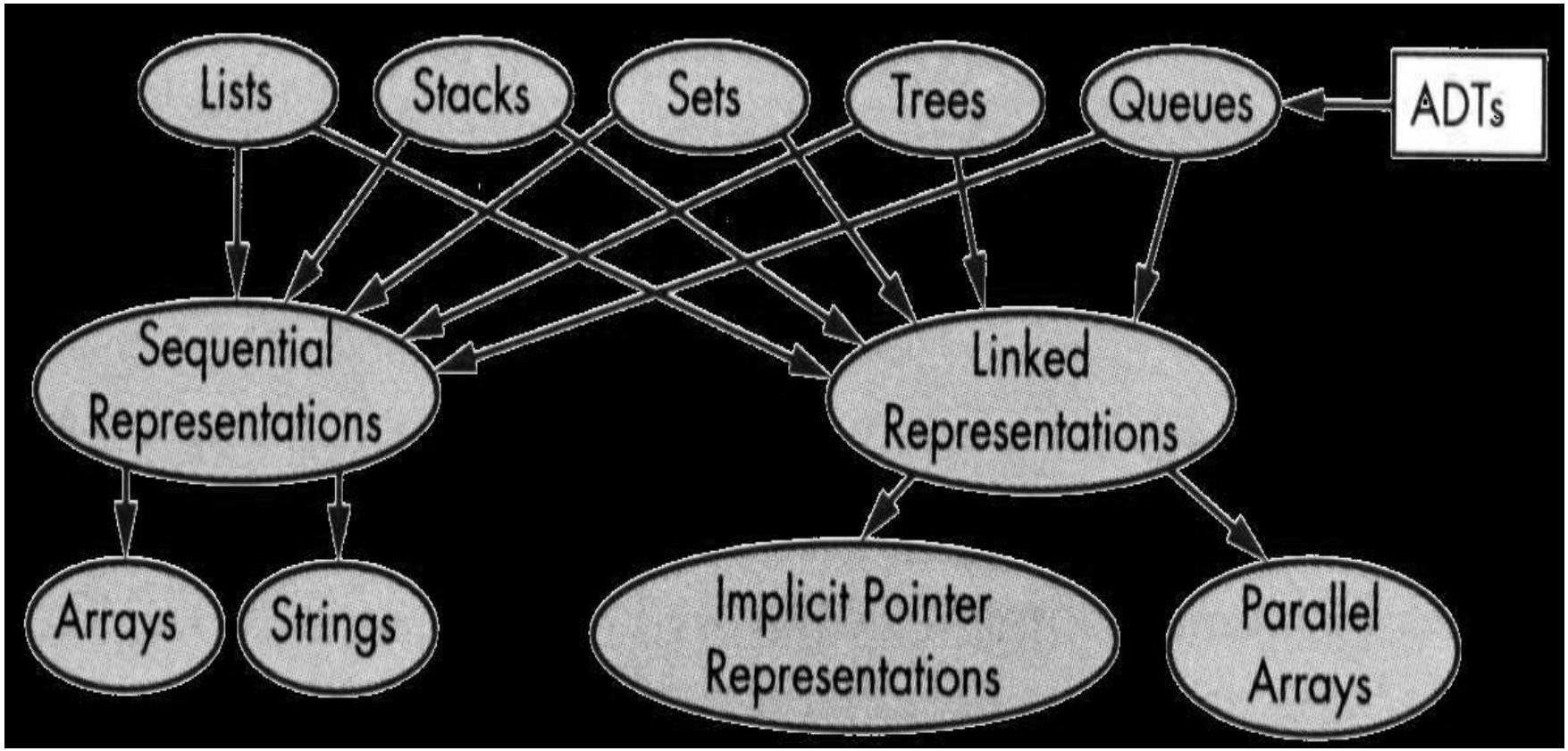
Esercitazione 1

- Implementazione ADT Lista, Coda e Pila



E1: lista (list): definizione

- Una lista L è una sequenza finita di elementi di un insieme fissato U
  - $L = <a_0, a_1, \ldots, a_n>$
- Operazioni fondamentali sono:
  - eVuota(L): vero se L=<>, falso altrimenti
  - elemento(L,k): a_k se 0<=k<=n, errore altrimenti
  - cancella(L,k): $< a_0,\ldots,a_{k-1},a_{k+1},\ldots,a_n>$ se 0<=k<=n, errore altrimenti

E1: lista

- inserisci(L,k,u):
  - <*u*> se L=<> e k=0,
  - < a_0,…,a_{k-1}, *u*, a_k,…,a_n> se 0<=k<=n+1,
  - errore altrimenti
- lunghezza(L): *0* se L=<>, *n+1* altrimenti

- Altre operazioni
  - cambia(L,k,u)=elimina(inserisci(L, k, u), k+1)
  - appartiene(L,u)= k se a_k=u, -1 altrimenti

E1: lista - Implementazione

- Studio di due implementazioni:
  - tipo sequenziale mediante array
  - tipo dinamica mediante puntatori (oggetti Java)

E1: lista - Interfaccia Java

```
public interface Lista {
    /**
     * eVuota() restituisce vero se la lista è vuota, falso altrimenti
     */
    boolean eVuota();

    /**
     * lunghezza() restituisce il numero degli oggetti presenti nella lista
     */
    int lunghezza();

    /**
     elemento(int k) restituisce l'oggetto nella posizione k-ma se 0<=k<=n,
     * altrimenti lancia un'eccezione IllegalArgumentException.
     */
    Object elemento(int k) throws IllegalArgumentException;
```

E1: lista - Interfaccia Java

```
    /**
     * cancella(int) cancella l'oggetto alla posizione k-ma se
     * 0<=k<=n,
     * altrimenti lancia un'eccezione IllegalArgumentException.
     */
     void cancella(int k) throws IllegalArgumentException;

    /**
     * inserisci(k,u) inserisce nella posizione k-ma l'oggetto u se
     * 0<=k<=n+1,
     * altrimenti lancia un'eccezione IllegalArgumentException.
    */
     void inserisci(int k, Object u) throws IllegalArgumentException;

}
```

E1: lista - implementazione con array

- Struttura dati di base: array di Object

```
...
private static int dimensioneIniziale = 20; //dimensione iniziale della lista

private Object[] lista;
private int nOggetti;  //variabile di supporto
...
public ListaArray() {
    lista = new Object[dimensioneIniziale];
    nOggetti = 0;
}
```

E1: lista - implementazione con array

- Metodi più significativi: inserisci(int, Object)

```
public void inserisci(int k, Object u) throws IllegalArgumentException {
    if (k<0 || k>nOggetti)
        throw new IllegalArgumentException();
    else {
        if (nOggetti < lista.length) {       //c'è ancora spazio
            for (int i=nOggetti ; i > k; i--) //sposta di 1 pos. elementi a dx di k
                lista[i]=lista[i-1];
            lista[k]=u;
            nOggetti++;
        }
        else {                  //è necessario aumentare la dimensione dell'array
            Object[] listaNuova = new Object[lista.length + dimensioneIniziale];
```

E1: lista - implementazione con array

```
        for(int i=0; i < k; i++)       //copia tutti gli elementi prima di k
            listaNuova[i]=lista[i];

        listaNuova[k]=u;   //inserisco il nuovo elemento in posizione k

        for(int i=k; i < nOggetti; i++)//copia tutti gli altri elementi dopo k
            listaNuova[i+1]=lista[i];

        nOggetti++;
        lista=listaNuova;
    } //else controllo spazio
  } // else parametri corretti
} // fine metodo
```

E1: lista - implementazione con array

- Metodi più significativi: cancella(int, Object)

```
public void cancella(int k) throws IllegalArgumentException {
    if (k<0 || k>=nOggetti)
      throw new IllegalArgumentException();
    else {
      nOggetti--;//se sono all'ultimo elemento, ho già fatto
      if (k != nOggetti) {//non sono nell'ultimo elemento
        for ( ; k < nOggetti; k++)
          lista[k]=lista[k+1];
      } // if
    } // else
} //fine metodo
```

E1: lista - implem. con puntatori impliciti

- Struttura dati di base: classe con autoriferimento

```
private class Node {  /*nodo della lista: contiene l'elemento e l'indirizzo
                              del successivo */
   Object key;
   Node next;

   Node(Object u) {
     key = u;
     next = null;
   }
 }
```

E1: lista - implem. con puntatori impliciti

- Dichiarazione di lista

```
private Node head;          //inizio della lista
private int nOggetti;       //numero elementi (supporto)
...
public ListaLink() {
    head = null;
    nOggetti = 0;
}
```

E1: lista - implem. con puntatori impliciti

- Metodi più significativi: inserisci(int, Object)

```
public void inserisci(int k, Object u)
                                    throws IllegalArgumentException {
    if (k<0 || k>nOggetti)
      throw new IllegalArgumentException();
    else {
     Node nuovo = new Node(u);   //crea nuovo nodo da inserire
      if (nOggetti == 0) {    //è il primo elemento da inserire
              head = nuovo;
      } else {
```

E1: lista - implem. con puntatori impliciti

```
    if (k==0) { //è da inserire in testa
       nuovo.next = head;
       head = nuovo;
    } else {// posizionati nell'elemento prec. a quello k-mo
       Node indice = head;
       for (int i=0 ; i < k-1; i++)
          indice = indice.next;
       nuovo.next = indice.next;          //a 'nuovo' attacco il
                                          //resto della lista
       indice.next = nuovo; //al precedente attacco il nuovo
    } }
  nOggetti++;
} } }// fine metodo
```

E1: lista - implem. con puntatori impliciti

- Metodi più significativi: cancella(int, Object)

```
public void cancella(int k) throws IllegalArgumentException {
    if (k < 0 || k >= nOggetti)
      throw new IllegalArgumentException();
    else {
      if (nOggetti == 1) {    //è l'unico elemento
        head = null;
      } else {
        if (k == 0) {   //è il primo elemento
          head = head.next;
        } else {          //è un elemento generico
```

E1: lista - implem. con puntatori impliciti

```
        //posizionati nell'elemento precedente a quello da cancel.
          Node indice = head;
          for (int i=0 ; i < k-1; i++)
            indice = indice.next;

          indice.next= indice.next.next;   //cancello
        }
      }
      nOggetti--;
    }
  }
```

E1: lista - Utilizzo implementazioni

```
public static void main(String[] args) {
    Lista listaA = new ListaArray();
    Lista listaB = new ListaLink();
    long inizio, fine;

    //Calcolo il tempo per inserire 100 elementi
    inizio = System.currentTimeMillis();
    for (int i=0; i < 100; i++) {
      listaA.inserisci(i, new Integer(i));
    }
    fine = System.currentTimeMillis();
    System.out.println("Tempo richiesto (ms) per la lista array: "+
                                (fine-inizio));
```

E1: lista - Confronto

- Costo in tempo caso peggiore

Operazione	Impl. con array	Impl. con puntatori
Lunghezza()	O(1)	O(1)
Inserisci(k,u)	O(n)	O(n)
Cancella	O(n)	O(n)
Elemento(k)	O(1)	O(n)

E1: lista - Confronto

- Costo in spazio
  - sia *c* costo in spazio per un riferimento ad un *Object*
  - sia *l* la capacità dell'array in ListaArray
  - sia *n* il numero di elementi da memorizzare
  - Costo di ListaArray: n*c + (l - n)*c = l*c
  - Costo di ListaLink: 2*n*c
  - n = l/2 $\Rightarrow$ la rappr. con array è più vantaggiosa quando la lista è più lunga della metà della capacità

E1: lista

- Stato dell'arte
  - esistono già le classi pronte che implementano l'adt *lista*
    - java.util.List è l'interfaccia (per documentazione **`file://jdk1.3/docs/api/java/util/List.html)`**
    - java.util.ArrayList è l'implementazione con array, **`file://jdk1.3/docs/api/java/util/ArrayList.html`**
    - java.util.Vector è un'altra implementazione (migliore), **`file://jdk1.3/docs/api/java/util/Vector.html`**

E1: lista

- Stato dell'arte (continua)
  - java.util.LinkedList è un implementazione ottimizzata per inserimento e cancellazione in testa o in coda,
    `file://jdk1.3/docs/api/java/util/LinkedList.html`
  - java.util.AbstractList è un'implementazione astratta: perché serve?
    `file://jdk1.3/docs/api/java/util/AbstractList.html`

E1: lista

- Compito dell'esercitazione
  - completare le due classi con i metodi mancanti
  - proporre un'interfaccia per la struttura dati pila e una per la struttura dati coda con un esempio di implementazione per ciascuna