Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa E06

Esercizi su
Strutture collegate

A. Miola

Marzo 2008

Contenuti

- La keyword *static* in Java
  - Metodi e variabili static
- Esercizio: metodi per creare una struttura collegata che rappresenta una data sequenza di stringhe
- Esercizio: estendere l'esercizio precedente per gestire la struttura collegata che rappresenta sequenze di stringhe

La keyword static in Java

- Quando si definisce una classe Java non si alloca spazio per le variabili, e i metodi non possono essere richiamati direttamente.
- Solo dopo aver creato una istanza della classe con la keyword new il Java Run-Time alloca spazio per le variabili e i metodi diventano disponibili al programma

```
Class MyClass {
 int x;
 void setX(int x) { this.x = x; }
 public static void main(String[] args){
 MyClass myClass = new MyClass();
 myClass.setX(7);
} }
```

La keyword static in Java

- Ci sono due situazioni particolari:
  - Condividere una singola istanza di una variabile per tutti gli oggetti di una certa classe (anche nel caso in cui nessun oggetto della classe sia stato creato)
  - Richiamare un metodo A.f() anche nel caso in cui nessun oggetto della classe di appartenenza A sia stato creato
- Una variabile o un metodo static indicano che la variabile o il metodo non sono legati ad una particolare istanza della classe

Variabili statiche

- Nel seguente esempio st1 e st2 condividono la variabile i, perciò st1.i e st2.i hanno lo stesso valore pari a 22
- Si può accedere ad una variabile static anche con il nome della classe di appartenenza `StaticTest.i`

```
class StaticTest {
 // variabile condivisa a tutte le istanze di StaticTest
 static int i = 22;
 public static void main(String[] args){
 StaticTest st1 = new StaticTest();
 StaticTest st2 = new StaticTest();

 // stampa tre volte 22
 System.out.println(st1.i);
 System.out.println(st2.i);
 System.out.println(StaticTest.i);
} }
```

Metodi statici

- Possiamo definire un metodo static e richiamarlo anche senza istanziare la classe di appartenenza:

```
class StaticTest {
 static int i = 22;
 static void incr() { i = i + 1; }

 public static void main(String[] args){
  StaticTest.incr();
  // stampa 23
  System.out.println(StaticTest.i);
 }
}
```

Metodi statici e variabili non-statiche...

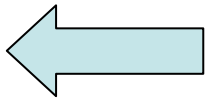
- Poiché un metodo definito static non è vincolato ad una particolare istanza, esso non può accedere direttamente agli elementi non-static della classe

```
Class MyOnlyPrinter {
 PrintStream printer = new PrintStream();
 MyOnlyPrinter() {}

 void static print(int x) {
  printer.print(x); // ERRORE
 }
}
```

... Metodi statici e variabili non-statiche

- Esempio: nella classe Persona non possiamo accedere a nome da un metodo static

```
class Persona {
  private String nome;

  public Persona(String nome) {
    this.nome = nome;
  }
  public static String getNome() {
    return nome;
  }


  public String toString() {
    return "Mi chiamo " + getNome() + ".";
  }
}
```



Errore di compilazione Java: "non-static variable nome *cannot be referenced from a static context* return nome*"*

Usi frequenti di static . . .

- Si usa spesso static per definire una costante che si può riutilizzare in diversi contesti

```
Class MyIOClass {
 public static final int OK = 0;
 public static final int ERROR = 1;
 . . .
 public int writeChar(char ch) {
    int ret;
    try {
       . . .
       ret = MyIOClass.OK;
    } catch(IOException) {
       ret = MyIOClass.ERROR;
    }
    return ret;
}}
```

Usi frequenti di static . . .

- Si usa static anche per segnalare che un metodo non accede alle variabili di una classe

```
Class MyClass {
   int tot, x, y;

   public static double mean(int[] p) {
     int sum = 0;
     for (int i=0; i<p.length; i++)
       sum += p[i];
     return ((double)sum) / p.length;
} }
```

Esercizi

- Scrivere i metodi necessari per costruire una struttura collegata per rappresentare una data sequenza di stringhe
  - disponibile come un array di stringhe
  - disponibile come un file di input
  - mantenendo l'ordine degli elementi dal primo all'ultimo
  - invertendo l'ordine degli elementi dall'ultimo al primo

Esercizio: sequenza di stringhe . . .

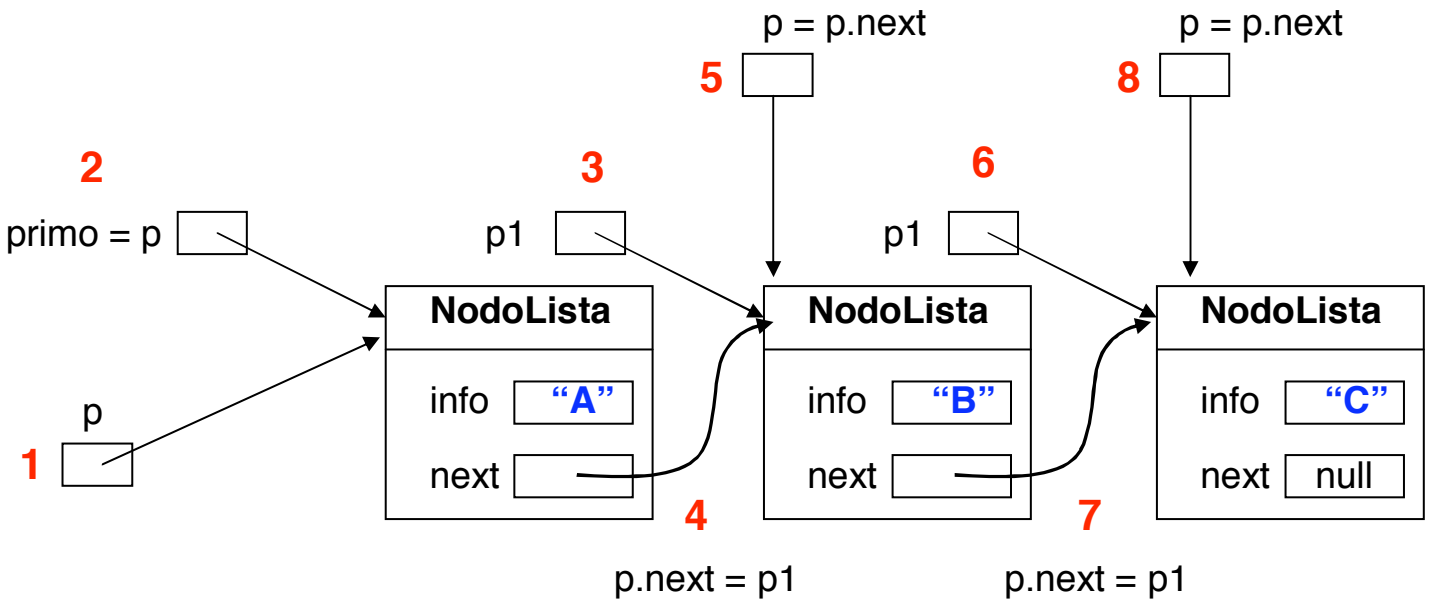
```
class NodoLista {

 public String info;
 public NodoLista next;

 public NodoLista() {
   info = new String();
   next = null;
 }
 // costruttore alternativo
 public NodoLista(String s, NodoLista n) {
   info = s;
   next = n;
 }
}
```

. . .Esercizio: sequenza di stringhe . . .

```
public class SequenzaStringhe {
 // crea una lista da un array di stringhe
 public static NodoLista crea(String[] stringhe) {
   NodoLista primo = null;
   NodoLista p = new NodoLista(stringhe[0], null);
   // supponiamo almeno un elemento nell'array
   primo = p;
   for (int i = 1; i < stringhe.length; i++) {
     NodoLista p1 = new NodoLista(stringhe[i], null);
     p.next = p1;
     p = p.next;
   }
   return primo;
 }
```

. . .Esercizio: sequenza di stringhe . . .

```
crea(String[] stringhe)
con stringhe[] = { “A”, “B”, “C” }
```



. . .Esercizio: sequenza di stringhe . . .

```
/* crea una lista da uno stream di input */
public static NodoLista crea(Lettore in) {
  NodoLista primo = null; /* radice della lista */
  String line = in.leggiLinea();
  if (line != null) {
    NodoLista p = new NodoLista(line, null);
    primo = p;
    while (!in.eof()) {  /* finche' ho stringhe */
      line = in.leggiLinea();
      NodoLista p1 = new NodoLista(line, null);
      p.next = p1;
      p = p.next;
    }
  }
  return primo;
}
```

. . . Esercizio: sequenza di stringhe . . .

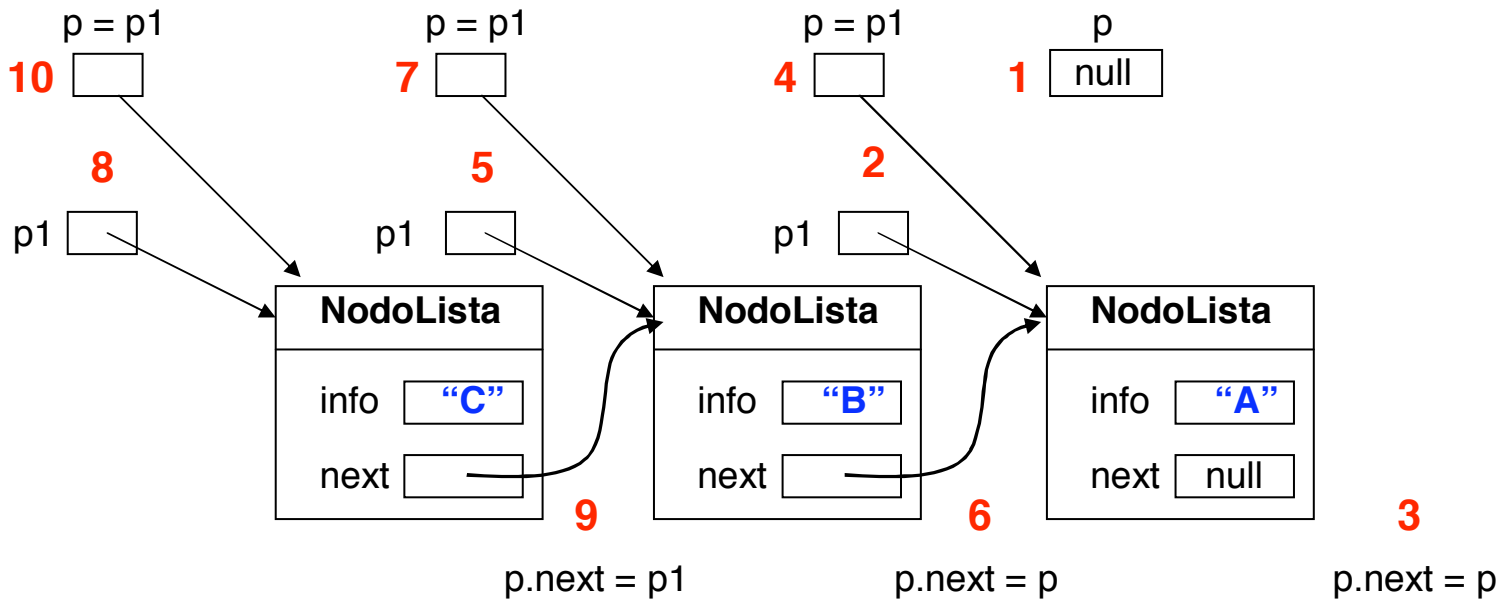
- Il metodo crea() prende in input un Lettore che per mezzo del metodo leggiLinea() permette di leggere una linea alla volta da un file
- Supponiamo perciò che il file sia già stato aperto

. . . Esercizio: sequenza di stringhe . . .

```
// crea una lista da un array di stringhe
// in ordine inverso
public static NodoLista creaInv(String[]
  stringhe){
  NodoLista p = null;
  for (int i = 0; i < stringhe.length; i++) {
    NodoLista p1 = new NodoLista(stringhe[i],p);
    p = p1;
  }
  return p;
}
```

. . .Esercizio: sequenza di stringhe . . .

```
creaInv(String[] stringhe)
con stringhe[] = { “A”, “B”, “C” }
```



. . . Esercizio: sequenza di stringhe . . .

```
// crea una lista da uno stream di input
// in ordine inverso
public static NodoLista creaInv(Lettore in) {
  NodoLista p = null;
  String line;
  while (!in.eof()) {
    line = in.leggiLinea();
    NodoLista p1 = new NodoLista(line, p);
    p = p1;
  }
  return p;
}
```

. . . Esercizio: sequenza di stringhe . . .

```
// converte la lista in una stringa;
// utile per stamparne il contenuto
public static String toString(NodoLista p) {
  StringBuffer sb = new StringBuffer();
  while(p != null) {
    sb.append(p.info+"\n");
    p = p.next;
  }
  return sb.toString();
}
```

. . . Esercizio: sequenza di stringhe . . .

```
public static void main(String[] args){
 try {
  String[] stringhe = new
  String[]{"stringa1","stringa2","stringa3"};
  FileReader fr = new FileReader("stringhe.txt");
  Lettore in = new Lettore(fr);
  /* crea da stream */
  NodoLista p = ListaLettore.crea(in);
  System.out.println(SequenzaStringhe.toString(p));
  /* crea da un array di stringhe */
  p = SequenzaStringhe.crea(stringhe);
  System.out.println(SequenzaStringhe.toString(p));
  ...
}
```

. . . Esercizio: sequenza di stringhe . . .

```
                                . . .
  // apro di nuovo il file e mi riposiziono
  // all'inizio con lo stream
  fr = new FileReader("listaStringhe.txt");
  in = new Lettore(fr);
  p = SequenzaStringhe.creaInv(in);
  System.out.println(SequenzaStringhe.toString(p));

  p = SequenzaStringhe.creaInv(stringhe);
  System.out.println(SequenzaStringhe.toString(p));
 } catch(Exception ex) { . . . }
}
```

. . . Esercizio: sequenza di stringhe

- **Output del programma:**

```
prima-stringa
seconda-stringa
terza-stringa
quarta-stringa

stringa1
stringa2
stringa3

quarta-stringa
terza-stringa
seconda-stringa
prima-stringa

stringa3
stringa2
stringa1
```

Esercizio: sequenza di stringhe v2. . .

- Vogliamo estendere la classe SequenzaStringhe con metodi vari per la gestione della lista:
  - Aggiungere una nuova stringa alla lista
  - Rimuovere una stringa esistente
  - Recuperare una stringa in base alla posizione
  - Trovare la posizione di un certa stringa
  - Restituire la dimensione della lista
  - Creare una sotto-lista
  - Convertire la lista in un array
  - . . .

. . . sequenza di stringhe v2. . .

```
public class SequenzaStringhe {

private NodoLista nodoinit;

// costruttore che crea una lista vuota
public SequenzaStringhe() { ... }

// aggiunge la stringa x in posizione k
public void add (int k, String x) { ... }

// aggiunge la stringa x in coda alla lista
public void add (String x) { ... }

// concatena alla lista this una lista passata come argomento
public void append (SequenzaStringhe l) { ... }

// elimina l'elemento in posizione k
public void remove (int k) { ... }

// elimina la prima occorrenza della stringa x
public void remove (String x) { ... }
```

. . . sequenza di stringhe v2. . .

```
// elimina tutti gli elementi della lista
public void clear () { ... }

// modifica il valore del k-esimo elemento con la stringa x
public void set (int k, String x) { ... }

// restituisce la stringa contenuta nel k-esimo elemento
public String get (int k) { ... }

// verifica se una stringa x è presente nella lista
public boolean contains (String x) { ... }

// restituisce l'indice della prima occorrenza di x nella lista
// o -1 se questa nonè presente
public int indexOf (String x) { ... }

// restituisce l'indice dell'ultima occorrenza di x nella lista
// o -1 se questa non è presente
public int lastIndexOf (String x) { ... }
```

. . . sequenza di stringhe v2. . .

```
// verifica se la lista è vuota
public boolean isEmpty () { ... }

// restituisce la lunghezza della lista
public int size () { ... }

// copia il contenuto della lista e restituisce una nuova lista
public SequenzaStringhe copy () { ... }

// restituisce la lista composta dagli elementi j,j+1,...,k-1
public SequenzaStringhe subList (int j, int k) { ... }

// restituisce una stringa con il contenuto della lista
public String toString () { ... }

// restituisce un array di stringhe contenente tutti
// gli elementi della lista
public String[] toArray () { ... }
}
```

Condizioni di errore . . .

- Alcuni metodi prendono in input un indice relativo alla posizione (set, add, remove, …)
- Puo' accadere che condizioni errate portino a chiamate con indici fuori da limiti

  `remove(10)` ma `size() < 10`
- Occorre che la classe che gestisce la lista segnali l'errore in qualche modo al metodo chiamante.

. . . Condizioni di errore

- **I modi utilizzati piu' frequentemente sono:**
  - Ritorno di un valore fuori da normale codominio del metodo:

    `subList(0, 10)` → `null`
  - Lanciare (throws) delle eccezioni che vengono catturate (catch) da qualche parte nel programma

```
throws new ArrayIndexOutOfBoundsException()
```

- **Per semplicita' nel codice che proponiamo le condizioni di errori sono trattate con return senza valore**

. . . sequenza di stringhe v2. . .

- Prima vediamo l'implementazione dei metodi che modificano la struttura della lista, ovvero inserimenti e cancellazioni:

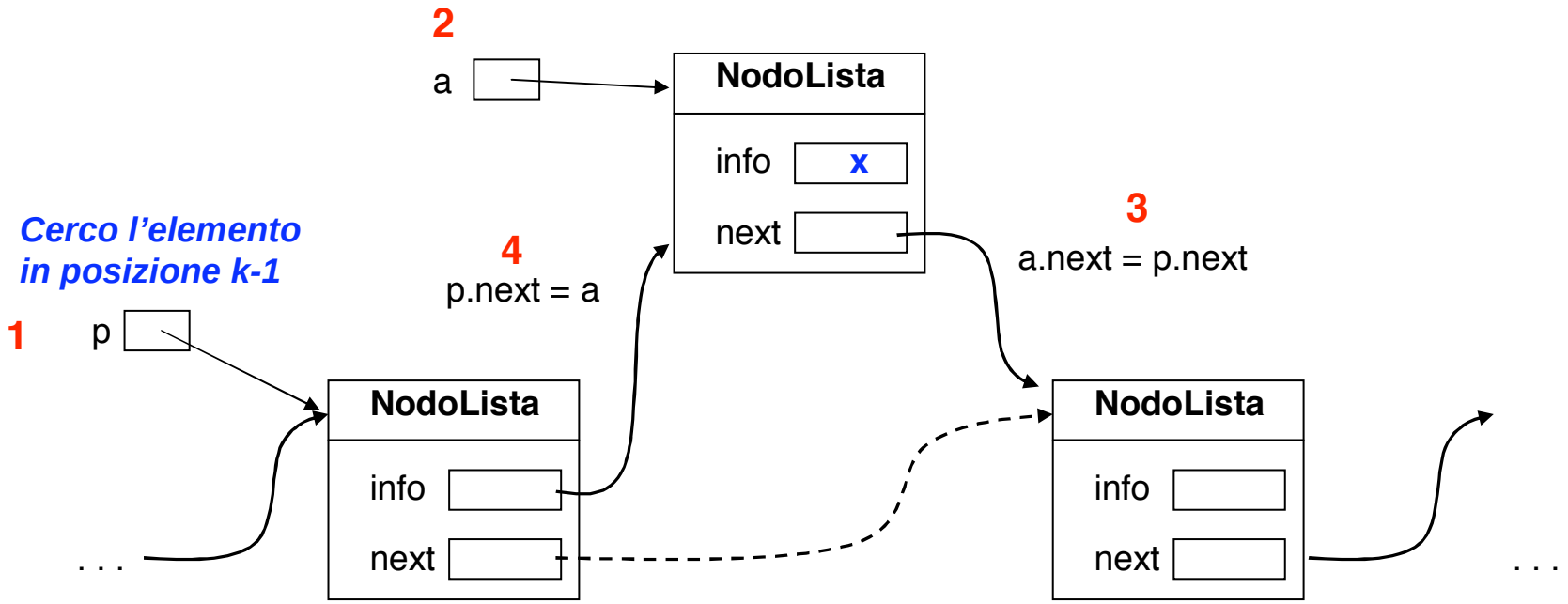
```
// costruttore che crea una lista vuota
public SequenzaStringhe() {
  nodoinit = null;
}

// aggiunge la stringa x in posizione k
public void add(int k, String x) {
  if (k < 0)
    return; /* Indice fuori dei limiti */
  else if (k == 0) {
    // Inserimento in testa alla lista
    NodoLista a = new NodoLista(x, nodoinit);
    nodoinit = a;
  } else {
    // Inserimento in posizione qualsiasi
    . . .
```

. . . sequenza di stringhe v2. . .

```
    NodoLista p = nodoinit;
    for (int i=0; i < k-1 && p != null; i++)
      p = p.next;
    if (p == null)
       return; /* Indice fuori dei limiti */
    else {
      // p è il riferimento al nodo in posizione k-1
      NodoLista a = new NodoLista(x, p.next);
      p.next = a;
    }
  } // fine else inserimento posizione qualsiasi
}

// aggiunge la stringa x in coda alla lista
public void add(String x){
  this.add(this.size(), x);
}
```

. . . sequenza di stringhe v2. . .

`add(int k, String x) con k > 0`



- La sequenza di operazioni è adatta anche nel caso in cui k-1 sia l'ultimo elemento della lista, cioè p.next = null

. . . sequenza di stringhe v2. . .

```
// concatena alla lista this la lista l
public void append(SequenzaStringhe l) {
  // 1.crea un nodo ausiliario per trattare
  uniformemente
  // il caso in cui l'oggetto di invocazione sia
  la
  // lista vuota
  NodoLista a = new NodoLista(null, nodoinit);
  // 2.vai fino all'ultimo elemento della lista
  NodoLista p = a;
  while (p.next != null)
    p = p.next;
  . . .
```

. . . sequenza di stringhe v2. . .

```
  . . .
  // 3.copia gli elementi di l in coda a p
  NodoLista q = l.nodoinit;
  while (q != null) {
    p.next = new NodoLista(q.info, null);
    p = p.next;
    q = q.next;
  }
  // 4.elimina il nodo ausiliario
  nodoinit = a.next;
}
```

. . . sequenza di stringhe v2. . .

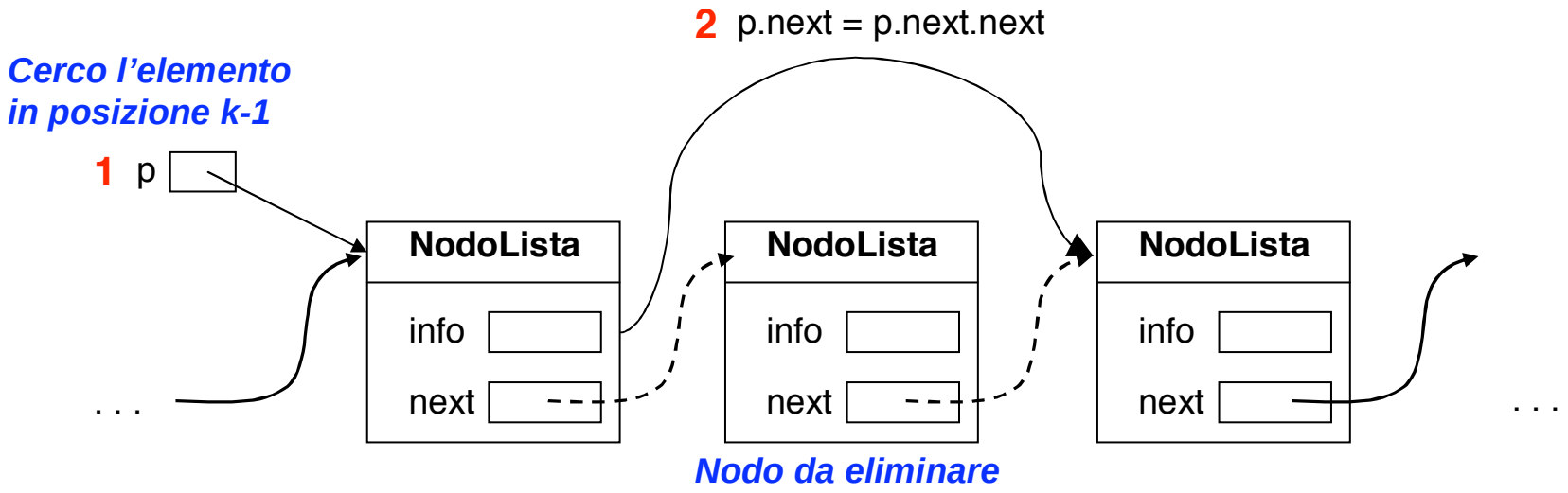
```
// elimina l'elemento in posizione k
public void remove(int k){
  if (k < 0)
     return; /* Indice fuori dei limiti */
  else if (nodoinit == null)
     return; /* Lista vuota */
  else if (k == 0)
    // Eliminazione in testa
    nodoinit = nodoinit.next;
  . . .
```

. . . sequenza di stringhe v2. . .

```
  . . .
  else {
    // Eliminazione in posizione qualsiasi
    NodoLista p = nodoinit;
    for (int i=0; i < k-1 && p != null; i++)
      p = p.next;
    if ((p == null) || (p.next == null))
       return; /* Indice fuori dei limiti */
    else
      // p e' il riferimento al nodo in k-1
      p.next = p.next.next;
  }
}
```

. . . sequenza di stringhe v2. . .

`remove(int k) con k > 0`



- La sequenza di operazioni è adatta anche nel caso in cui k sia l'ultimo elemento della lista, cioè p.next.next = null
- Il garbage collector si occupa di cancellare l'oggetto dalla memoria

. . . sequenza di stringhe v2. . .

```
// elimina la prima occorrenza della stringa x
public void remove(String x) {
  boolean b = false;
  if (nodoinit.info.equals(x)) {
    // eliminazione del primo nodo della lista
    nodoinit = nodoinit.next;
  } else {
    NodoLista p = nodoinit;
    while (p.next!=null && !p.next.info.equals(x))
      p = p.next;
    if (p.next != null)
      // p e' il riferimento al nodo che precede
      // il nodo contenente x
      p.next = p.next.next;
  }
}
```

. . . sequenza di stringhe v2. . .

```
// elimina tutti gli elementi della lista
public void clear () {
  nodoinit = null;
}
```

. . . sequenza di stringhe v2. . .

- Il metodo equals(Object obj) di String ritorna true se l'argomento obj non è nullo e se (String)obj rappresenta la stessa sequenza di caratteri dell'oggetto String su cui viene richiamato equals().
- Il metodo clear() sfrutta il garbage collector per eliminare tutti gli elementi della lista

. . . sequenza di stringhe v2. . .

- Il metodo set modifica il contenuto della lista, ma non la sua struttura

```
// modifica il valore del k-esimo elemento con x
public void set(int k, String x) {
  if (k < 0)
     return; /* Indice fuori dei limiti */
  else {
    NodoLista p = nodoinit;
    for (int i=0; i < k && p != null; i++)
      p = p.next;
    if (p == null)
       return; /* Indice fuori dei limiti */
    else
      // p e' il riferimento al nodo in posizione k
      p.info = x;
  }
}
```

. . . sequenza di stringhe v2. . .

- Il metodo set è implementato effettuando un ciclo per accedere al nodo in posizione k e successivamente si modifica il campo info di tale nodo
- A differenza dei metodi add e remove, l'operazione di accesso al nodo k non deve terminare nel nodo precedente, in quanto in questo caso non è necessario modificare la struttura collegata aggiungendo o rimuovendo nodi

. . . sequenza di stringhe v2. . .

- In questa sezione descriviamo i metodi che non modifcano la lista su cui sono invocati

```
// restituisce la stringa contenuta nel k-esimo elemento
public String get(int k) {
  if (k < 0)
     return; /* Indice fuori dei limiti */
  NodoLista p = nodoinit;
  for (int i = 0; i < k && p != null; i++)
    p = p.next;
  if (p == null)
     return; /* Indice fuori dei limiti */
  else
    // p e' il riferimento al nodo in posizione k
    return p.info;
 }

// verifica se una stringa x è presente nella lista
public boolean contains(String x) {
  NodoLista p = nodoinit;
  while (p != null && !p.info.equals(x))
    p = p.next;
  return p != null;
}
```

. . . sequenza di stringhe v2. . .

```
// restituisce l'indice della prima occorrenza di x nella lista
// o -1 se questa non è presente
public int indexOf(String x) {
  int r = -1, c = 0;
  NodoLista p = nodoinit;
  while (p != null && r < 0) {
    if (p.info.equals(x))
    r = c;
    c++; p = p.next;
  }
  return r;
}

// restituisce l'indice dell'ultima occorrenza di x nella lista
// o -1 se questa non è presente
public int lastIndexOf(String x) {
  int r = -1, c = 0;
  NodoLista p = nodoinit;
  . . .
```

. . . sequenza di stringhe v2. . .

```
  while (p != null) {
    if (p.info.equals(x))
    r = c;
    c++; p = p.next;
  }
  return r;
}

// verifica se la lista e' vuota
public boolean isEmpty() {
  return nodoinit == null;
}

// restituisce la lunghezza della lista
public int size() {
  int c = 0;
  NodoLista p = nodoinit;
  while (p != null) {
    c++;
    p = p.next;
  }
  return c;
}
```

. . . sequenza di stringhe v2. . .

```
// copia il contenuto della lista e restituisce
// una nuova lista
public SequenzaStringhe copy() {
  // Crea il primo nodo della lista risultato
  // (nodo generatore)
  NodoLista a = new NodoLista(); a.next = null;
  NodoLista p = nodoinit;
  NodoLista q = a;
  while (p != null) {
    q.next = new NodoLista();
    q = q.next;
    q.info = p.info;
    p = p.next;
  }
  . . .
```

. . . sequenza di stringhe v2. . .

```
    . . .
    // bisogna terminare la lista con il riferimento
    // null
    q.next = null;
    SequenzaStringhe r = new SequenzaStringhe();
    r.nodoinit = a.next;
    return r;
}
```

. . . sequenza di stringhe v2. . .

```
// restituisce la lista composta dagli elementi
// j,j+1,...,k-1
public SequenzaStringhe subList (int j, int k) {
  // 1. crea il primo nodo della lista risultato
  // (nodo generatore)
  NodoLista a = new NodoLista();
  a.next = null;
  NodoLista q = a;
  // 2. vai alla posizione j
  NodoLista p = nodoinit; int i=0;
  while (i < j && p != null)
    p = p.next; i++;
  . . .
```

. . . sequenza di stringhe v2. . .

```
  . . .
  // 3. copia gli elementi fino a k
  while (i < k && p != null) {
    q.next = new NodoLista(); q = q.next;
    q.info = p.info; q.next = null;
    p = p.next; i++;
  }
  // 4. restituisci la lista calcolata
  SequenzaStringhe r = new SequenzaStringhe();
  r.nodoinit = a.next;
  return r;
}
```

. . . sequenza di stringhe v2. . .

```
// restituisce una stringa con il contenuto della lista
public String toString() {
  NodoLista p = nodoinit;
  String ris = "( ";
  while (p != null) {
    ris += p.info + " ";
    p = p.next;
  }
  ris += ")";
  return ris;
}

// restituisce un array di stringhe contenente tutti
// gli elementi della lista
public String[] toArray() {
  // 1. Calcola la dimensione della struttura
  int n = this.size();
  . . .
```

. . . sequenza di stringhe v2

```
  . . .
  // 2. Crea l'array
  String[] a = new String[n];
  // 3. Copia le informazioni nell'array
  NodoLista p = nodoinit;
  int i = 0;
  while (p != null) {
    a[i] = p.info;
    i++; p = p.next;
  }
  return a;
}
```

Sequenza di stringhe v2: Esempio . . .

- Vogliamo sfruttare la classe SequenzaStringhe e implementare due metodi statici
  - accoda(SequenzaStringhe *l*, String[ ] *v*) che accoda alla lista *l* gli elementi dell'array di stringhe *v*
  - rimuovitutti(SequenzaStringhe *l*, String *x*) che rimuove dalla lista *l* tutte le occorrenze della stringa *x*
- Nel metodo *main* richiameremo questi metodi per creare e operare su una lista

. . .Sequenza di stringhe v2: Esempio . . .

```
public class ProvaSequenzaStringhe {
public static void accoda(SequenzaStringhe l,
  String[] v) {
  int k = l.size();
  for (int i=0; i < v.length; i++, k++)
    l.add(k, v[i]);
}

public static void rimuovitutti
                (SequenzaStringhe l, String x) {
  while (l.contains(x))
    l.remove(x);
}

   . . .
```

. . .Sequenza di stringhe v2: Esempio . . .

```
public static void main (String[] args) {
  SequenzaStringhe l = new SequenzaStringhe();
  l.add(0,"B");
  l.add(1,"C");
  l.add("D");
  l.add(0,"A");
  System.out.println(l);
  String[] v = {"E","F","G"};
  accoda(l,v);
  System.out.println(l);
  l.add(0,"X");
  l.add(4,"X");
  l.add(l.size(),"X");
  l.add("X");
  System.out.println(l);
  rimuovitutti(l,"X");
  System.out.println(l);
  System.out.println("Lunghezza="+l.size());
  System.out.println("Lunghezza="+
                          l.toArray().length);
} }
```

. . . Esempio: sequenza di stringhe v2

- Output del programma:

```
( A B C D )
( A B C D E F G )
( X A B C X D E F G X X )
( A B C D E F G )
Lunghezza = 7
Lunghezza = 7
```