Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa 12

ADT: Lista, Pila, Coda

A. Miola

Marzo 2008

Contenuti

- Tipo astratto Lista
- Realizzazione del Tipo astratto Lista
- Esempi d'uso della classe Lista
- Tipo astratto Pila
- Realizzazione del Tipo astratto Pila
- Esempi d'uso della classe Pila
- Tipo astratto Coda
- Realizzazione del Tipo astratto Coda
- Esempi d'uso della classe Coda

Prerequisiti

- Questo capitolo presuppone la conoscenza di tutti gli argomenti trattati in precedenza nel corso e in particolare le Dispense n. 07 e n. 11

Il Tipo astratto Lista

- Il tipo astratto Lista è una tripla

  < S , F , C >

  dove

  - S = { Lista , V , Boolean }
    - Lista è il dominio di interesse
    - V è il dominio di sostegno, dominio degli elementi che formano le liste
  - F = { listaVuota, vuota, cons, car, cdr }
  - C = { Lista_Vuota }

Funzioni

- **listaVuota :** $() \longrightarrow$ **Lista**
  - Costruisce la lista vuota
- **vuota :** **Lista** $\longrightarrow$ **Boolean**
  - Verifica se una lista data è vuota
- **cons :** **V x Lista** $\longrightarrow$ **Lista**
  - Costruisce una lista inserendo un elemento di V come primo elemento di una lista data
- **car :** **Lista** $\longrightarrow$ **V**
  - Calcola e restituisce il primo elemento di una lista data
- **cdr :** **Lista** $\longrightarrow$ **Lista**
  - Calcola e restituisce la lista ottenuta eliminando il primo elemento di una lista data

Esempi

- Supponendo di operare con liste di caratteri, che vengono rappresentate in forma parentetica, si ha
- vuota((A B C)) → false
- vuota(()) → true
- car ((A B C)) → A
- car (()) → *indefinito*
- cdr ((A B C)) → (B C)
- cdr (()) → *indefinito*
- cons (A, (B C D)) → (A B C D)

Realizzazione del tipo astratto Lista . . .

- La realizzazione del tipo astratto Lista come classe Java richiede di fare delle scelte non immediate
- Rappresentazione dei valori:
  - un modo naturale di rappresentare i valori di tipo Lista in Java è attraverso una struttura collegata
  - in particolare utilizziamo un campo dati nodoinit di tipo Nodo che denota l'inizio della struttura collegata
- Schema realizzativo:
  - possiamo scegliere sia uno schema realizzativo funzionale che uno schema realizzativo con side-effect
  - nella realizzazione di seguito riportata scegliamo lo schema realizzativo funzionale

. . . Realizzazione del tipo astratto Lista . . .

❑ Interfaccia di classe:

- definiamo un metodo per ciascuna delle funzioni del tipo astratto
- realizziamo la funzione listaVuota attraverso un costruttore senza parametri
- ridefiniamo in modo opportuno i metodi equals e toString ereditati da Object in modo da verificare l'uguaglianza profonda degli oggetti Lista
  - due liste sono uguali se rappresentano sequenze degli stessi elementi

❑ Realizzazione dei metodi:

- notiamo solo che nella realizzazione dei metodi facciamo uso di RuntimeException quando sono violate le precondizioni di applicabilità delle funzioni del tipo astratto (faremo questo anche negli esempi successivi)

. . . Realizzazione del tipo astratto Lista . . .

- Il codice della classe Java **Lista** è il seguente, dove si è assunto che gli elementi della lista siano di tipo **Object**

```
class Nodo {
  public Object info;
  public Nodo next; }

public class Lista {
  // rappresentazione degli oggetti
  private Nodo nodoinit;
  . . .
```

. . . Realizzazione del tipo astratto Lista . . .

```
   . . .
// realizzazione dei costruttori del tipo
public Lista() {
   //realizza listaVuota
   nodoinit = null;
}

private Lista(Nodo n) {
   // costruttore privato
   nodoinit = n;
}

   . . .
```

. . . Realizzazione del tipo astratto Lista . . .

```
  . . .
// realizzazione delle funzioni del tipo
public boolean vuota() {
  return nodoinit == null; }

public Lista cons(Object o) {
  Nodo aux = new Nodo();
  aux.info = o;
  aux.next = nodoinit;
  return new Lista(aux); }
  . . .
```

. . . Realizzazione del tipo astratto Lista . . .

```
  . . .
public Object car() {
  if (vuota()) throw new RuntimeException
  ("Lista: car applicato a una lista vuota");
  else
     return nodoinit.info;}

public Lista cdr() {
  if (vuota()) throw new RuntimeException
  ("Lista: cdr applicato a una lista vuota");
  else
     return new Lista(nodoinit.next); }
  . . .
```

. . . Realizzazione del tipo astratto Lista . . .

```
  . . .
// uguaglianza
public boolean equals(Object o) {
  boolean uguale;
  if (o == null ||
     !getClass().equals(o.getClass()))
          uguale = false;
  Lista l = (Lista)o;
  Nodo n1 = nodoinit;
  Nodo n2 = l.nodoinit;
  . . .
```

. . . Realizzazione del tipo astratto Lista . . .

```
   . . .
  while (n1 != null && n2 != null) {
     if (!n1.info.equals(n2.info))
                              uguale = false;
     n1 = n1.next;
     n2 = n2.next;
  }
     if (n1 != null || n2 != null)
                              uguale = false;;
     else uguale = true;
  return uguale;
}
   . . .
```

. . . Realizzazione del tipo astratto Lista . . .

```
  . . .
// toString
public String toString() {
  String s;
  if (vuota()) s = "";
  else
     s = car()+""+cdr().toString();
  return s;
}
} // end class Lista
```

Esempi di uso della classe Lista . . .

- Vediamo ora alcune operazioni non primitive che possono essere definite sulle liste realizzate come visto in precedenza
- Calcolo della lunghezza della lista
  - dato un oggetto Lista calcola la lunghezza della lista che esso rappresenta
- Aggiunta di un elemento in coda alla lista
  - dato un oggetto Lista ed un elemento da aggiungere restituisce un nuovo oggetto Lista ottenuto dal primo aggiungendo il nuovo elemento come ultimo elemento della lista

. . . Esempi di uso della classe Lista . . .

- **Concatenazione (append) di due liste**
  - dati due oggetti Lista restituisce un nuovo oggetto Lista ottenuto concatenando le due liste
  - cioè restituendo una lista composta dagli elementi della prima lista e seguiti dagli elementi della seconda lista
- **Calcolo dell'elemento i-esimo della lista**
  - preso un oggetto Lista e un intero *i* (dove *i* è minore della lunghezza della lista) restituisce l'elemento in posizione *i*
- **Inserimento di un nuovo elemento nella posizione i-esima**
  - preso un oggetto Lista, un intero *i* (dove *i* è minore della lunghezza della lista) e un elemento da inserire, restituisce un nuovo oggetto Lista ottenuto dal primo aggiungendo in posizione *i*-esima il nuovo elemento

. . . Esempi di uso della classe Lista

- Per realizzare queste operazioni definiamo la classe ListaUso e facciamo uso della ricorsione, sfruttando il fatto che le liste sono definite induttivamente
- Infatti
  - la lista vuota è una lista
  - aggiungendo un elemento in testa ad una lista (mediante l'operazione *cons*) otteniamo una lista

Calcolo della lunghezza

```
public class ListaUso {
  public static int
                lunghezzaLista(Lista l) {
     int lun;
     if (l.vuota()) lun = 0;
     else lun =
          1 + lunghezzaLista(l.cdr());
     return lun;
  }
  . . .
```

Aggiunta in coda

```
  . . .
public static Lista
      aggiungiUltimo(Object o, Lista l) {
   Lista a;
   if (l.vuota()) a = l.cons(o);
   else
     a = aggiungiUltimo(o,l.cdr()).cons(l.car());
   return a;
}
  . . .
```

Append di due liste

```
  . . .
public static Lista
            append(Lista l1, Lista l2) {
  Lista a;
  if (l1.vuota()) a = l2;
  else
     a = append(l1.cdr(),l2).
                          cons(l1.car());
   return a;
}
  . . .
```

Elemento in una posizione data

```
  . . .
public static Object
     elementoInPosizione(Lista l, int i) {
  Object o;
  if (l.vuota() || i<0) throw new
     RuntimeException("ListaUso:
                 indice fuori dai limiti");
  else
     if (i==0) o = l.car();
  else o =
           elementoInPosizione(l.cdr(), i-1);
  return o;
}
  . . .
```

Aggiunta di un elemento in una posizione data

```
  . . .
public static Lista
     inserisciElementoInPosizione
          (Lista l, int i, Object o) {
  Lista aux;
  if (i<0) throw new
     RuntimeException("ListaUso:
               indice fuori dai limiti");
  else if (i==0) aux = l.cons(o);
     else aux = inserisciElementoInPosizione
          (l.cdr(),i-1,o).cons(l.car());
  return aux;
}
```

Commento

- Si noti che tutte le operazioni viste sono definite in modo indipendente dalla rappresentazione scelta per le liste, usando le operazioni pubbliche fornite dalla classe
- Se cambia la rappresentazione il codice delle operazioni qui definite rimane invariato

Il Tipo astratto Pila

- Il tipo astratto Pila (Stack), che rappresenta sequenze di elementi di un tipo prefissato che vengono gestite con la politica LIFO (Last In First Out), ossia l'ultimo elemento entrato è il primo ad uscire, è una tripla

  < S , F , C >

  dove

  - S = { Pila , V , Boolean }
    - Pila è il dominio di interesse
    - V è il dominio di sostegno, dominio degli elementi che formano la pila
  - F = { pilaVuota, vuota, push, top, pop }
  - C = { Pila_Vuota }

Funzioni

- **pilaVuota :** $() \longrightarrow$ **Pila**
  - Costruisce la pila vuota
- **vuota :** **Pila** $\longrightarrow$ **Boolean**
  - Verifica se una pila data è vuota
- **push :** **V x Pila** $\longrightarrow$ **Pila**
  - Costruisce una pila inserendo un elemento di V come primo elemento di una pila data
- **top :** **Pila** $\longrightarrow$ **V**
  - Calcola e restituisce il primo elemento di una pila data
- **pop :** **Pila** $\longrightarrow$ **Pila**
  - Calcola e restituisce la pila ottenuta eliminando il primo elemento di una pila data

Realizzazione del tipo astratto Pila . . .

- La realizzazione del tipo astratto Pila come classe Java richiede di fare delle scelte non immediate
- Rappresentazione dei valori:
  - un modo naturale di rappresentare i valori di tipo Pila in Java è attraverso una struttura collegata
  - in particolare utilizziamo un campo dati nodoinit di tipo Nodo che denota l'inizio della struttura collegata
- Schema realizzativo:
  - possiamo scegliere sia uno schema realizzativo funzionale che uno schema realizzativo con side-effect
  - nella realizzazione di seguito riportata scegliamo lo schema realizzativo con side-effect

. . . Realizzazione del tipo astratto Pila

❑ Interfaccia di classe:

- definiamo un metodo per ciascuna delle funzioni del tipo astratto
- realizziamo la funzione pilaVuota attraverso un costruttore senza parametri
- ridefiniamo in modo opportuno i metodi equals e toString ereditati da Object in modo da verificare l'uguaglianza profonda degli oggetti Pila
  - due pile sono uguali se rappresentano la stessa sequenza di elementi

❑ Realizzazione dei metodi:

- notiamo solo che nella realizzazione dei metodi facciamo uso di RuntimeException quando sono violate le precondizioni di applicabilità delle funzioni del tipo astratto (faremo questo anche negli esempi successivi)

Realizzazione del tipo astratto Pila con side-effect . . .

- Il codice della classe Java Pila è il seguente, dove si è assunto che gli elementi della pila siano di tipo Object

```
class Nodo {
  public Object info;
  public Nodo next; }

public class Pila {
  // rappresentazione degli oggetti
  private Nodo nodoinit;
  . . .
```

. . . Realizzazione del tipo astratto Pila con side-effect . . .

```
  . . .
// realizzazione delle funzioni del tipo
  public Pila () {
     // realizza pilaVuota
     nodoinit = null;
  }
  public boolean vuota() {
     return nodoinit == null;
  }
  . . .
```

. . . Realizzazione del tipo astratto Pila con side-effect . . .

```
. . .
public void push (Object o) {
   Nodo aux = new Nodo();
   aux.info = o;
   aux.next = nodoinit;
   nodoinit = aux;
}
. . .
```

. . . Realizzazione del tipo astratto Pila con side-effect . . .

```
  . . .
public Object top() {
  if (vuota()) throw new
     RuntimeException("Pila:
           top applicato ad una pila vuota");
  else return nodoinit.info; }

public void pop() {
  if (vuota()) throw new
     RuntimeException("Pila:
           pop applicato ad una pila vuota");
  else nodoinit = nodoinit.next; }
  . . .
```

. . . Realizzazione del tipo astratto Pila con side-effect . . .

```
  . . .
//uguaglianza
public boolean equals(Object o) {
  boolean uguale;
  if (o == null ||
     !getClass().equals(o.getClass()))
          uguale = false;
  Pila p = (Pila)o;
  Nodo n1 = nodoinit;
  Nodo n2 = p.nodoinit;
  . . .
```

. . . Realizzazione del tipo astratto Pila con side-effect

```
  . . .
 while (n1 != null && n2 != null) {
    if (!n1.info.equals(n2.info))
                          uguale = false;
    n1 = n1.next;
    n2 = n2.next;
 }
    if (n1 != null || n2 != null)
                          uguale = false;;
    else uguale = true;
 return uguale;
}
} // end class Pila
```

Realizzazione funzionale del tipo astratto Pila . . .

- Se invece scegliamo uno schema realizzativo funzionale dobbiamo modificare la realizzazione della classe Java definendo la classe PilaF come segue

```
class Nodo {
  public Object info;
  public Nodo next; }

public class PilaF {
  // rappresentazione degli oggetti
  private Nodo nodoinit;
  . . .
```

. . . Realizzazione funzionale del tipo astratto Pila . . .

```
   . . .
// realizzazione dei costruttori del tipo
public PilaF() {
   //realizza pilaVuota
   nodoinit = null;
}

private PilaF(Nodo n) {
   // costruttore privato
   nodoinit = n;
}
   . . .
```

. . . Realizzazione funzionale del tipo astratto Pila . . .

```
   . . .
// realizzazione delle funzioni del tipo
public boolean vuota() {
   return nodoinit == null; }

public PilaF push(Object o) {
   Nodo aux = new Nodo();
   aux.info = o;
   aux.next = nodoinit;
   return new PilaF(aux);
      // uso costruttore privato
}
   . . .
```

. . . Realizzazione funzionale del tipo astratto Pila . . .

```
  . . .
public Object top() {
  if (vuota()) throw new RuntimeException
  (“PilaF: top applicato a una pila vuota");
  else
     return nodoinit.info;}

public PilaF pop() {
  if (vuota()) throw new RuntimeException
  ("PilaF: pop applicato a una pila vuota");
  else
     return new PilaF(nodoinit.next);
          // uso costruttore privato
}
  . . .
```

. . . Realizzazione funzionale del tipo astratto Pila . . .

```
  . . .
// uguaglianza
public boolean equals(Object o) {
  boolean uguale;
  if (o == null ||
     !getClass().equals(o.getClass()))
          uguale = false;
  PilaF l = (PilaF)o;
  Nodo n1 = nodoinit;
  Nodo n2 = l.nodoinit;
  . . .
```

. . . Realizzazione funzionale del tipo astratto Pila

```
  . . .
  while (n1 != null && n2 != null) {
     if (!n1.info.equals(n2.info))
                              uguale = false;
     n1 = n1.next;
     n2 = n2.next;
  }
  if (n1 != null || n2 != null)
                          uguale = false;
  else uguale = true;
  return uguale;
}
} // end class PilaF
```

Il Tipo astratto Coda . . .

- Il tipo astratto Coda (Queue) rappresenta sequenze di elementi di un tipo prefissato che vengono gestite con la politica FIFO (First In First Out)
  - l'elemento inserito per primo è il primo ad essere eliminato
- Coerentemente con questa politica, i valori del tipo Coda servono a rappresentare situazioni in cui, ad esempio, si vuole simulare l'andamento ad uno sportello di servizio, in cui i clienti sono serviti secondo l'ordine di arrivo

. . . Il Tipo astratto Coda

- **Il tipo astratto Coda è una tripla**

  < S , F , C >

  dove

  - **S = { Coda , V , Boolean }**
    - **Coda è il dominio di interesse**
    - **V è il dominio di sostegno, dominio degli elementi che formano le liste**
  - **F = { codaVuota, vuota, inCoda, outCoda, primo }**
  - **C = { Coda_Vuota }**

Funzioni

- **codaVuota :** () → **Coda**
  - Costruisce la coda vuota
- **vuota :** **Coda** → **Boolean**
  - Verifica se una coda data è vuota
- **inCoda :** **V x Coda** → **Coda**
  - Costruisce una coda inserendo un elemento di V come ultimo elemento di una coda data
- **outCoda :** **Coda** → **Coda**
  - Calcola e restituisce la coda ottenuta eliminando l'elemento di testa di una coda data (il primo elemento inserito)
- **primo :** **Coda** → **V**
  - Calcola e restituisce l'elemento di testa di una coda data (il primo elemento inserito)

Realizzazione del tipo astratto Coda . . .

- La realizzazione del tipo astratto Coda come classe Java richiede di fare delle scelte non immediate
- Rappresentazione dei valori:
  - un modo naturale di rappresentare i valori di tipo Coda in Java è attraverso una struttura collegata
  - per rendere semplice l'inserimento in coda alla struttura stessa dobbiamo fare uso di un riferimento addizionale all'ultimo elemento della struttura collegata
  - utilizziamo due campi dati nodotesta e nodocoda di tipo Nodo che denotano rispettivamente il primo e l'ultimo elemento della struttura collegata

. . . Realizzazione del tipo astratto Coda . . .

- **Schema realizzativo:**
  - possiamo scegliere sia uno schema realizzativo funzionale che uno schema realizzativo con side-effect
  - scegliamo lo schema realizzativo con side-effect
- **Interfaccia di classe:**
  - definiamo un metodo per ciascuna delle funzioni del tipo astratto
  - realizziamo la funzione codaVuota attraverso un costruttore senza parametri
  - ridefiniamo in modo opportuno il metodo equals ereditato da Object in modo da verificare l'uguaglianza profonda degli oggetti Coda
    - due code sono uguali se rappresentano la stessa sequenza di elementi

. . . Realizzazione del tipo astratto Coda . . .

```
class Nodo {
  public Object info;
  public Nodo next;
}

public class Coda {
  //rappresentazione degli oggetti
  private Nodo nodotesta;
  private Nodo nodocoda;
  . . .
```

. . . Realizzazione del tipo astratto Coda . . .

```
 . . .
//realizzazione delle funzioni del tipo
public Coda() {
  // realizza codaVuota
  nodotesta = null;
  nodocoda = null;
}
public boolean vuota() {
  return nodotesta == null;
}
 . . .
```

. . . Realizzazione del tipo astratto Coda . . .

```
  . . .
public void inCoda(Object o) {
   Nodo aux = new Nodo();
   aux.info = o;
   aux.next = null;
   if (nodotesta == null) {
      nodotesta = aux;
      nodocoda = aux; }
   else {
      nodocoda.next = aux;
      nodocoda = aux; }
}
  . . .
```

. . . Realizzazione del tipo astratto Coda . . .

```
  . . .
public void outCoda() {
   if (vuota()) throw new
      RuntimeException("Coda: outCoda applicato
                        ad una coda vuota");
   else {
      nodotesta = nodotesta.next;
      if (nodotesta == null)
           nodocoda = null;
   }
}
  . . .
```

. . . Realizzazione del tipo astratto Coda . . .

```
  . . .
public Object primo() {
  if (vuota()) throw new
     RuntimeException("Coda: primo
          applicato ad una coda vuota");
  else return nodotesta.info;
}
  . . .
```

. . . Realizzazione del tipo astratto Coda . . .

```
 . . .
//uguaglianza
public boolean equals(Object o) {
  boolean uguali;
  if (o == null ||
     !getClass().equals(o.getClass()))
           uguali = false;
  Coda c = (Coda)o;
  Nodo n1 = nodotesta;
  Nodo n2 = c.nodotesta;
   . . .
```

. . . Realizzazione del tipo astratto Coda . . .

```
. . .
 while (n1 != null && n2 != null) {
    if (!n1.info.equals(n2.info))
                          uguali = false;
    n1 = n1.next;
    n2 = n2.next;
 }
 if (n1 != null || n2 != null)
                          uguali = false;
 else uguali = true;
 return uguali;
}
} // end class Coda
```