Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa E09

Altri Esercizi

C. Limongelli

Marzo 2008

Contenuti

- Definizione di classi con ereditarietà
- Uso di tipi di dati astratti
- Ricorsione
- Complessità

Definizione di classi con ereditarietà…

- Data la classe Poligono con
  - gli attributi:
    - numeroLati (di tipo int)
    - colore (di tipo String)
  - e con i relativi metodi getNumeroLati, getColore e toString e il metodo (astratto) area
- Definire tre sottoclassi:
  - Rettangolo, che ha come nuovi attributi base e altezza
  - TriangoloEquilatero ha un nuovo attributo lato
  - Quadrato ha un nuovo attributo lato
- Per tutte e tre le classi si deve definire area e ridefinire toString

… Definizione di classi con ereditarietà

- Dati tre array di triangoli, rettangoli e quadrati, rispettivamente, si vuole costruire un nuovo array di elementi Poligono che sia ordinato per aree crescenti e si vuole stampare l'elenco ordinato

La classe Poligono…

```
abstract class Poligono{

  protected int numeroLati;
  protected String colore;

  public Poligono(int nl, String c) {
     this.numeroLati = nl;
     this.colore = c;}

  public int getNumeroLati(){
     return numeroLati;}
```

...La classe Poligono

```
  // non e' definita per i poligoni
public abstract double area();

public String getColore() {
  return colore;}

public String toString(){
  return "POLIGONO: \n" +
        "Numero di Lati: " + numeroLati + "\n"+
        "Colore:          " + colore + "\n";}
}
```

La classe TrinagoloEquilatero

```
class TriangoloEquilatero extends Poligono {

  /** Il lato del triangolo. */
 private double lato;


  /** Crea una nuovo Triangolo equilatero. */
 public TriangoloEquilatero(double lato,
                 String colore, int numeroLati) {
      super(numeroLati,colore);
      this.lato = lato; }


  /** Restituisce il lato del quadrato. */
 public double lato() {
      return lato; }
```

La classe TrinagoloEquilatero

```
/** Restituisce l'area del triangolo equilatero. */
    // implementa il metodo astratto area()
  public double area() {
            return (lato*lato*Math.sqrt(3.))/4; }

    /** Overriding del metodo toString dei Poligoni */
  public String toString(){
      return "TRIANGOLO EQUILATERO: \n" +
             "Numero di Lati: " + numeroLati + "\n"+
             "Colore:         " + colore + "\n" +
             "Lunghezza lato: " + lato + "\n" +
             "Area:           " + area() + "\n";
   }
}
```

La classe Quadrato…

```
/** La forma geometrica quadrato. */
class Quadrato extends Poligono {

    /** Il lato del quadrato. */
    private double lato;


    /** Crea una nuovo Quadrato. */
    public Quadrato(double lato, String colore,
                                        int numeroLati) {
        super(numeroLati,colore);
        this.lato = lato; }


    /** Restituisce il lato del quadrato. */
    public double lato() {
        return lato; }
```

La classe Quadrato

```
/** Restituisce l'area del quadrato. */
    // implementa il metodo astratto area()
    public double area() { return lato*lato; }

    /** Overriding del metodo toString dei Poligoni */
    public String toString(){
      return "QUADRATO: \n" +
             "Numero di Lati: " + numeroLati + "\n"+
             "Colore:         " + colore + "\n" +
             "Lunghezza lato: " + lato + "\n" +
             "Area:           " + area() + "\n";
  }
}
```

La classe Rettangolo…

```
/** La forma geometrica rettangolo. */
class Rettang extends Poligono {

    /** I lati del rettangolo. */
      private double base;
      private double altezza;

    /** Crea una nuovo rettangolo. */
    public Rettang(double base, double altezza,
                                String colore, int numeroLati) {
        super(numeroLati,colore);
        this.base = base;
        this.altezza = altezza;}

    /** Restituisce la base del rettangolo. */
    public double base() {
        return base; }
```

…La classe Rettangolo

```
    /** Restituisce la'altezza del rettangolo. */
    public double altezza() {
        return altezza   ; }

    /** Restituisce l'area del rettangolo. */
    // implementa il metodo astratto area()
    public double area() { return base*altezza; }

    /** Overriding del metodo toString dei Poligoni */
    public String toString(){
      return "RETTANGOLO: \n" +
             "Numero di Lati: " + numeroLati + "\n"+
             "Colore:         " + colore + "\n" +
             "Lunghezza base: " + base + "\n" +
             "Lunghezza altezza: " + altezza + "\n" +
             "Area:           " + area() + "\n";
  }
}
```

La classe UsoPoligoni

- Crea tre array di rettangoli, triangoli e quadrati

```
class UsoPoligoni{

   public static void main(String[] args){
        //lunghezze degli array di poligoni
        final int T = 3;
        final int R = 1;
        final int Q = 2;

        Poligono[] poligoni;
        int i,p;

        poligoni = creaPoligoni(T,R,Q);
        ordinaPoligoni(poligoni);

        p = poligoni.length;
        for (i=0; i<p; i++)
               System.out.println(poligoni[i].toString());
   }
```

Creazione degli array di poligoni…

```
public static Poligono[] creaPoligoni(int t, int r, int
  q){
  TriangoloEquilatero[] triangoli;
  Rettang[] rettangoli;
  Quadrato[] quadrati;
  Poligono[] poligoni;

  // creazione degli array
  triangoli = new TriangoloEquilatero[t];
  triangoli[0] = new
                  TriangoloEquilatero(5.0,"verde",3);
  triangoli[1] = new
                  TriangoloEquilatero(2.0,"rosso",3);
  triangoli[2] = new TriangoloEquilatero(6.0,"blu",3);

  rettangoli = new Rettang[r];
  rettangoli[0] = new Rettang(5.0,3.0,"rosa",4);

  quadrati = new Quadrato[q];
  quadrati[0] = new Quadrato(4.0,"rosso",4);
  quadrati[1] = new Quadrato(3.0,"rosa",4);
```

…Creazione degli array di poligoni

```
// creazione dell'array di poligoni
   poligoni = new Poligono[t+r+q];

   poligoni[0] = triangoli[1];
   poligoni[1] = rettangoli[0];
   poligoni[2] = triangoli[0];
   poligoni[3] = quadrati[0];
   poligoni[4] = quadrati[1];
   poligoni[5] = triangoli[2];

   return poligoni;
}
```

Ordinamento di un array di poligoni mediante il bubbleSort…

```
public static void
        ordinaPoligoni(Poligono[] pol){
  int i,j,p;
  Poligono app;
  boolean finito,fattoscambio;
  finito = false;
  i = 0;
  p = pol.length;
```

…Ordinamento di un array di poligoni mediante il bubbleSort

```
 while(!finito){
     i = i+1;
     fattoscambio = false;
     for (j=0; j<p-1; j++)
          if(pol[j].area() > pol[j+1].area()){
               app = pol[j];
               pol[j] = pol[j+1];
               pol[j+1] = app;
               fattoscambio = true;
          }
     if ((!fattoscambio)||(i==p-1))
          finito = true;
  }
}
```

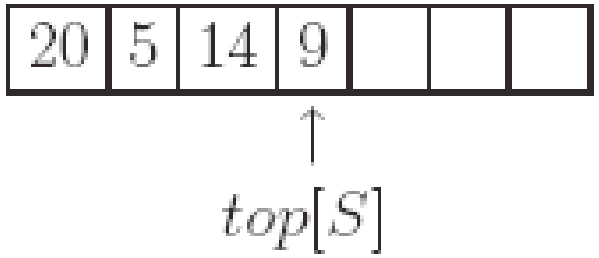
Esempio d'esecuzione

```
TRIANGOLO EQUILATERO:
Numero di Lati: 3
Colore:         rosso
Lunghezza lato: 2.0
Area:           1.732050807568772

QUADRATO:
Numero di Lati: 4
Colore:         rosa
Lunghezza lato: 3.0
Area:           9.0

TRIANGOLO EQUILATERO:
Numero di Lati: 3
Colore:         verde
Lunghezza lato: 5.0
Area:           10.825317547305483

RETTANGOLO:
Numero di Lati: 4
Colore:         rosa
Lunghezza base: 5.0
Lunghezza altezza: 3.0
Area:           15.0

TRIANGOLO EQUILATERO:
Numero di Lati: 3
Colore:         blu
Lunghezza lato: 6.0
Area:           15.588457268119894

QUADRATO:
Numero di Lati: 4
Colore:         rosso
Lunghezza lato: 4.0
Area:           16.0

Press any key to continue . . .
```

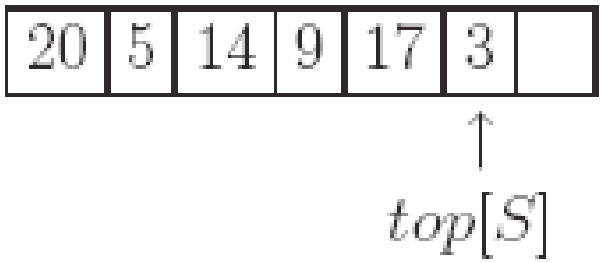
Rappresentazione delle pile mediante array…

- Una pila di al più *n* elementi è rappresentata da un array *S*[0..*n*]
- La pila ha un attributo *top*[*S*]: l'indice dell'elemento inserito in cima
- Gli elementi della pila sono contenuti in *S*[0..*top*[*S*]]
- *S*[*top*[*S*]]: elemento in cima alla pila
- *S*[0]: elemento in fondo alla pila
- Pila vuota: *top*[*S*] = -1

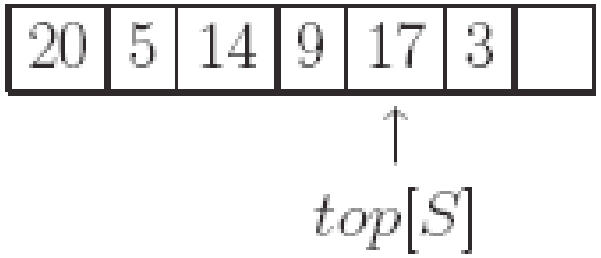
…Rappresentazione delle pile mediante array



Dopo l'esecuzione di `PUSH(S,17)` e `PUSH(S,3)`:



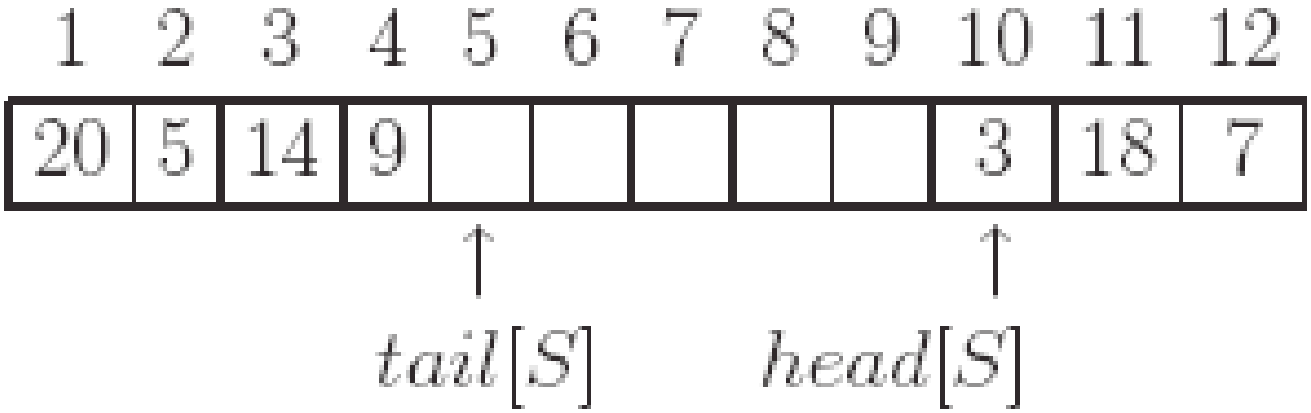
Dopo l'esecuzione di `POP(S)`, che riporta 3:



Rappresentazione delle code mediante array . . .

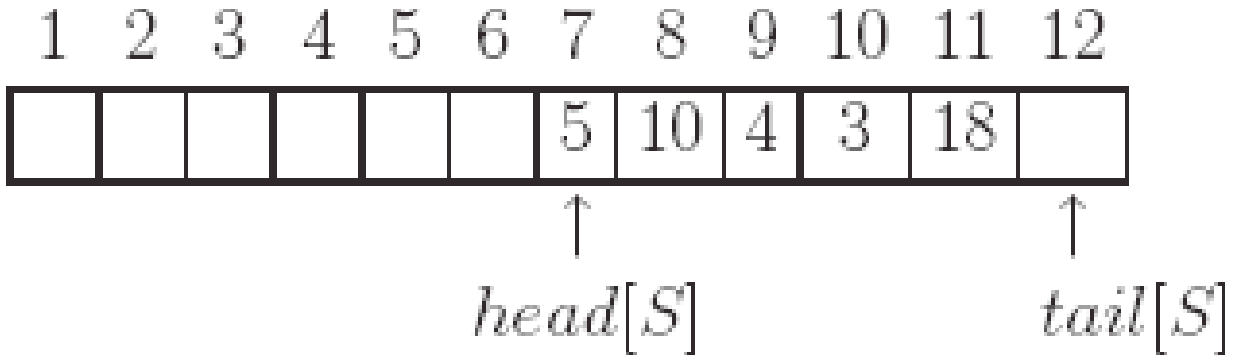
- Una coda di al più *n* elementi è rappresentata da un array *S*[0..*n*].
- Attrubuti della coda:
- *head*[*S*]: indice della "testa" della coda: *S*[*head*[*S*]] è l'elemento in testa
- *tail*[*S*]: indice della locazione in cui si inserirà il prossimo elemento
- Gli elementi della coda sono nelle posizioni con indici:
  - *head*[*S*], *head*[*S*] + 1, ..., *tail*[*S*] – 1
  - ma l'array è considerato circolare: dopo la locazione *length*[*S*] viene la locazione 0

. . . Rappresentazione delle code mediante array . . .

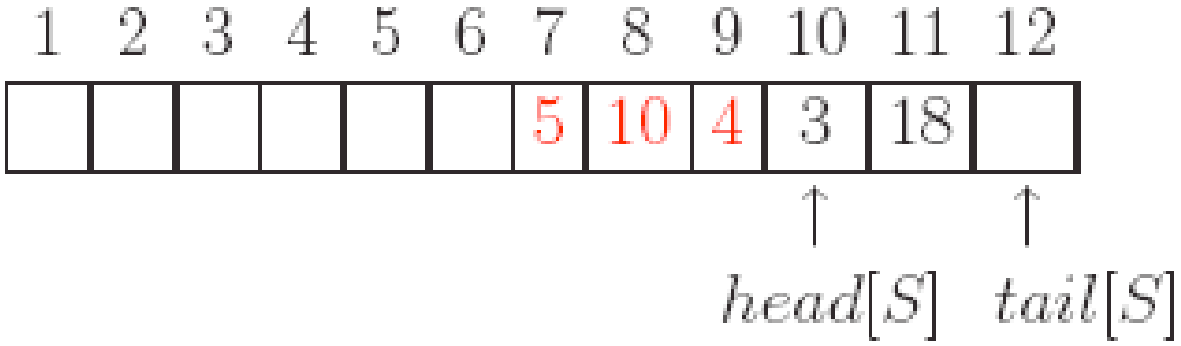


- La coda è vuota quando *head*[*S*] = *tail*[*S*]
- inizialmente *head*[*S*] = *tail*[*S*]
- La coda è piena quando *head*[*S*] = *tail*[*S*] + 1 oppure *tail*[*S*] = *length*[*S*] e *head*[*S*] = 1 (una posizione resta sempre vuota)

. . . Rappresentazione delle code mediante array . . .

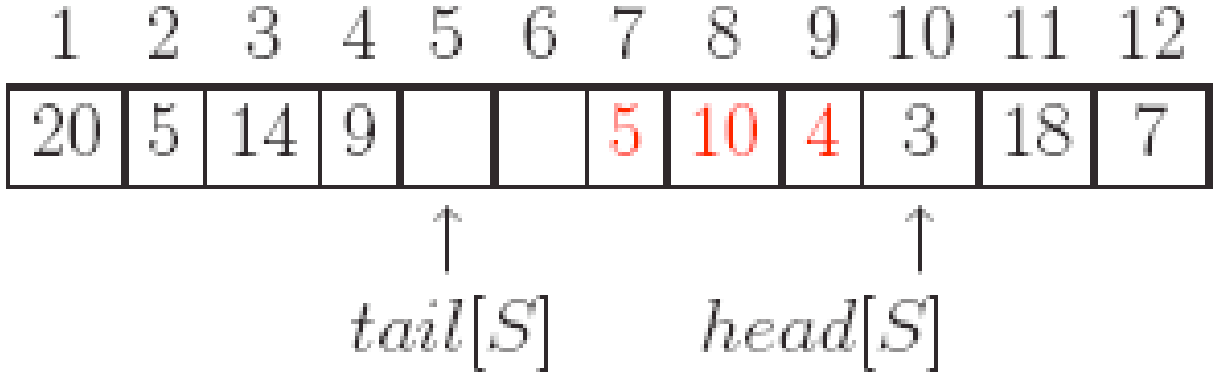


Dopo l'esecuzione di tre operazioni DEQUEUE[S], che riportano, rispettivamente, 5, 10 e 4:



. . . Rappresentazione delle code mediante array

- Dopo l'esecuzione delle operazioni ENQUEUE(S,7), ENQUEUE(S,20), ENQUEUE(S,5), ENQUEUE(S,14) e ENQUEUE(S,9):



N.B.: gli elementi in rosso non appartengono alla coda e possono essere sovrascritti

Uso di tipi di dato astratti

- Data la segnatura per il tipo di dato astratto lista, definire un metodo che, data una lista di oggetti che rappresentano coppie di valori interi, crea una nuova lista che contiene tutti gli elementi della prima che soddisfano una certa proprietà
  - Esempio: la somma degli elementi della coppia deve essere = 10
  - Oppure: gli elementi della coppia devono essere relativamente primi tra loro
- Consideriamo la lista di coppie di interi

Segnatura per il tipo di dato astratto lista

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

La classe Coppia

```
// NON E' ASTRATTA PERCHE' TUTTI
// I METODI  SONO ISTANZIATI

public class Coppia {
protected Object c1;
   protected Object c2;

      public Coppia (Object c1, Object c2) {
      // realizza formaCoppia
            this.c1 = c1;
            this.c2 = c2; }

  public Object primaComponente() { return c1; }
  public Object secondaComponente() { return c2; }

  public String toString(){
      return "(" + c1 + "," + c2 + ")";
  }
}
```

La lista di Coppie di interi

- La coppia di interi è una particolare istanza della coppia di Object

```
public class Lista_Coppie{
public static void main(String[] args){
   Coppia c1, c2;
   Lista l1,l2;

   l1 = creaLista();
   l2 = selezionaProp(l1);

   // la nuova lista l2 contiene solo
   // le coppie la cui somma e' 10
   System.out.println("nuova lista ");
   System.out.println(l2.toString());

}
```

Selezione di una sottolista con una data proprietà

```
public static Lista selezionaProp(Lista l){
  Lista app;

  if (l.vuota())
     app = new Lista();
  else
     if (verificaProp( (Coppia)(l.car()) ) )
          app = selezionaProp((l.cdr())).
                cons(l.car());
     else app = selezionaProp(l.cdr());
  return app;
}
```

Proprietà: la somma degli elementi della coppia deve essere 10

```
public static boolean verificaProp(Coppia c){
  return (
  ((Integer)
   (c.primaComponente())).intValue() +
  ((Integer)
   (c.secondaComponente())).intValue()
     == 10);
}

}
```

Creazione di una lista di coppie

```
public static Lista creaLista(){
   Lista l;
   Coppia c1,c2;

   //creo la lista vuota
   l = new Lista();

   //diverse modalita' per la creazione
   // - assegno dei valori ad una coppia
   // oppure creo la coppia direttamente all'interno della cons
   c1 = new Coppia(Integer.valueOf("3"),Integer.valueOf("4") );
   c2 = new Coppia(Integer.valueOf("6"),Integer.valueOf("4") );

   l = l.cons(c1).cons(c2)
        .cons(new Coppia(Integer.valueOf("10"),Integer.valueOf("0") ))
        .cons(new Coppia(Integer.valueOf("30"),Integer.valueOf("40") ))
        .cons(new Coppia(Integer.valueOf("5"),Integer.valueOf("5") ));

   return l;
}
```

Posizione dispari

- Scrivere un metodo che, data una lista l di stringhe, restituisce la lista dei nodi che si trovano in posizione dispari (primo nodo terzo nodo...)
- Usare eventualmente i metodi di Lista_Uso

- Algoritmo
- Se l e' vuota viene restituita la lista vuota;
- Se l e' composta da un solo elemento viene restituita l stessa
- Altrimenti (almeno due elementi)
  - Il primo elemento di l viene concatenato con il resto del resto di l a cui viene applicato il metodo ricorsivamente

Codifica ricorsiva

```
public static Lista dispariRIC(Lista l){
  Lista app;
  app = new Lista();

  if (!l.vuota())
     if ((l.cdr()).vuota()) app = l;
     else app = dispariRIC(l.cdr().cdr()).
                                  cons(l.car());
  return app;
}
```

Codifica iterativa

- Bisogna usare l'append per costruire la lista nell'ordine corretto

```
public static Lista dispari(Lista l){
   Lista app,nulla;
   int i,lung;
   nulla = new Lista();
   lung = Lista_Uso.lunghezzaLista(l);
   app = new Lista();
   i = 0;
   while (i<lung){
        if (i%2 == 0)
               app = Lista_Uso.append(app,
               nulla.cons(Lista_Uso.elementoInPosizione(l,i)));
        i++;
   }
   return app;
}
```

Nodi dispari con strutture collegate...

- Generazione dei nodi dispari a partire da una struttura collegata

```
class NodoLista {
      public String info;
      public NodoLista next;
      public NodoLista(String s, NodoLista n)
            {info = s; next = n;}
}
```

... Nodi dispari con strutture collegate

```
public static NodoLista nodiDispari(NodoLista l){
  if (l==null) return null;
  else
     if (l.next == null)
          return new NodoLista(l.info,null);
     else
          return
          new NodoLista(
                l.info,
                nodiDispari(l.next.next)
              );
}
```

Cosa fa?

- Dato il metodo ricorsivo:

```
public static Lista xxx(Lista l){
  if (l.vuota()) return l;
  else
     return append(
                 xxx(l.cdr()),
                (new Lista()).cons(l.car())
              );
 }
```

- Dire cosa calcola

Esercizio complessità …

- Dato il seguente metodo:

```
public static boolean nodiComuni
                              (NodoLista l1, NodoLista l2){
    boolean comune;
    comune = false;
    while (l1 != null && !comune){
        if (isMember(l1.info,l2))
               comune = true;
        l1 = l1.next;
    }
    return comune;
}
```

- Assumendo che il metodo isMember ha complessità asintotica lineare rispetto alla lunghezza di l2, dire qual è l'istruzione dominante e qual è la complessità del metodo nodiComuni

…Esercizio complessità …

- L'istruzione dominante è l'istruzione

```
if (isMember(l1.info,l2))
    comune = true;
```

- Il cui costo è dato dal costo della condizione + il costo del massimo tra la parte if e la parte else (che non è presente)

- Essendo isMember lineare in l2, se la lunghezza di l2 è n, il costo dell'istruzione dominante è

$$O(n) + \max(O(1), \_\_) = O(n)$$

- Se anche la lunghezza di l1 è n, poiché nel caso peggiore l'istruzione while viene ripetuta n volte, il costo del metodo nodiComuni è

$$n*O(n) = O(n^2)$$

… Esercizio complessità

- Confrontarlo con la complessità asintotica del relativo metodo ricorsivo:

```
public static boolean nodiComuniRIC
                    (NodoLista l1, NodoLista l2){
  boolean comune;

  if (l1 == null) comune = false;
  else
     comune = (isMember(l1.info,l2)) ||
               nodiComuniRIC(l1.next,l2);
  return comune;
}
```