Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa E05

Esercizi su ADT e rappresentazioni

A. Miola

Marzo 2007

Contenuti

- I tipi astratti Coppia, Razionale, Complesso
- Rappresentazione dei tipi Coppia, Razionale, Complesso
- Rappresentazione compatta di matrici sparse

Un altro esempio: i numeri razionali

- Consideriamo il tipo astratto *Q* dei numeri razionali, ricordando quanto già visto nel corso precedente circa la classe `Razionale`
  - Gli elementi a/b del dominio di interesse possono essere denotati da una coppia di numeri interi (a, b) : a denota il numeratore della frazione, mentre b denota il denominatore
  - Le operazioni del tipo *Q* sono le usuali operazioni : Addizione, Moltiplicazione, ecc.
- In effetti anche Coppia è un tipo astratto di dato che può essere opportunamente rappresentato da una classe `Coppia`

Il tipo astratto Coppia . . .

- Il tipo astratto Coppia rappresenta il prodotto cartesiano di due domini
- I valori di tale tipo sono coppie di valori presi da due domini specificati
- Le operazioni associate sono semplicemente
  - la costruzione
  - il calcolo del valore della prima componente
  - il calcolo del valore della seconda componente

. . . Il tipo astratto Coppia

La specifica del tipo Coppia, con i suoi domini (tra cui quello di interesse) e le sue funzioni, è quindi la seguente

Domini

- **Coppia : dominio di interesse del tipo**
- **T1 : dominio dei valori che possono comparire come prima componente delle coppie**
- **T2 : dominio dei valori che possono comparire come seconda componente delle coppie**

. . . Il tipo astratto Coppia

Funzioni

- formaCoppia(T1 a, T2 b) ⟶ Coppia
  - pre: nessuna
  - post: RESULT è il valore corrispondente alla coppia la cui prima componente è *a* e la seconda è *b*
- primaComponente(Coppia c) ⟶ T1
  - pre: nessuna
  - post: RESULT è la prima componente della coppia *c*
- secondaComponente(Coppia c) ⟶ T2
  - pre: nessuna
  - post: RESULT è la seconda componente della coppia *c*

Rappresentazione del tipo Coppia . . .

- La realizzazione del tipo astratto Coppia come una classe Java `Coppia` è immediata
  - Rappresentiamo i valori del tipo astratto utilizzando due campi dati (variabili di istanza) di tipo opportuno per memorizzare le due componenti della coppia
  - Schema realizzativo: utilizziamo lo schema realizzativo funzionale come illustrato precedentemente
  - Interfaccia di classe: come visto definiamo un metodo per ciascuna delle funzioni del tipo astratto
  - Realizziamo la funzione formaCoppia attraverso un costruttore

. . . Rappresentazione del tipo Coppia

```
public class Coppia {
// rappresentazione dei valori
   protected Object c1;
   protected Object c2;

// realizzazione delle funzioni del tipo astratto

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

. . . Rappresentazione del tipo Coppia

- Sebbene alcune signature non sembrano strettamente funzionali o coincidenti con le funzioni del tipo di dato astratto, come ad esempio:

  Java: `public Object primaComponente();`

  ADT: primaComponente(Coppia c) $\longrightarrow$ T1

  lo schema realizzativo del tipo di dato astratto è funzionale poiché i metodi della classe operano sui valori del tipo astratto e non eseguono operazioni di modifica di oggetti Java (oggetti di invocazione e parametri) che rappresentano tali valori

Il tipo astratto Razionale . . .

- Un numero razionale è un numero reale ottenibile come rapporto tra due numeri interi, il secondo dei quali diverso da 0. Ogni numero razionale quindi può essere espresso mediante una frazione a/b.
- A questo punto possiamo definire il tipo di dato astratto Razionale riprendendo il tipo Coppia, notando che:
  - I due domini T1 e T2 possono essere i due domini di valori che possono assumere numeratore e denominatore del numero razionale
  - Le funzioni del tipo Razionale sono quelle classiche, ad esempio addizione e moltiplicazione.

. . . Il tipo astratto Razionale . . .

Domini

- **Razionale : dominio di interesse del tipo**
- **T1 : dominio dei valori che può assumere il numeratore**
- **T2 : dominio dei valori che può assumere il denominatore**

. . . Il tipo astratto Razionale . . .

Funzioni (1 di 2)

- formaRazionale(T1 a, T2 b) ⟶ Razionale
  - pre: b non rappresenti 0
  - post: RESULT è il valore corrispondente al numero razionale *a/b*
- numeratore(Razionale q) ⟶ T1
  - pre: nessuna
  - post: RESULT è il numeratore del razionale *q*
- denominatore(Razionale q) ⟶ T2
  - pre: nessuna
  - post: RESULT è il denominatore del razionale *q*
- . . .

. . . Il tipo astratto Razionale . . .

Funzioni (2 di 2)

- add(Razionale q1, Razionale q2) ⟶ Razionale
  - pre: nessuna
  - post: RESULT è il numero razionale corrispondente alla somma dei due razionali rappresentati da *q1* e *q2*
- molt(Razionale q1, Razionale q2) ⟶ Razionale
  - pre: nessuna
  - post: RESULT è il numero razionale corrispondente alla moltiplicazione dei due razionali rappresentati da *q1* e *q2*

Rappresentazione del tipo Razionale . . .

- **L'implementazione della classe Razionale può essere basata su Coppia estendendo opportunamente le funzionalità di quest'ultima classe**
  - **Rappresentiamo i valori del tipo astratto con oggetti Coppia**
  - **Schema realizzativo funzionale**
  - **Interfaccia di classe: definiamo un metodo per ciascuna delle funzioni del tipo astratto**
  - **Realizziamo la funzione formaRazionale attraverso un costruttore**

. . . Rappresentazione del tipo Razionale

```
class Razionale extends Coppia {
  public Razionale (Long c1, Long c2){
    super (c1,c2);       // richiama il costruttore di Coppia
  }

  public void simpl() { // semplifica il razionale
    long app, num, den;
    num = ((Long)(this.c1)).longValue();
    den = ((Long)(this.c2)).longValue();
    app = mcd(num,den);
    c1 = new Long(num / app);
    c2 = new Long(den / app);
  }

  public String toString(){
    simpl();
    return  primaComponente()+ "/" +secondaComponente();
  } . . .
```

. . . Rappresentazione del tipo Razionale

```
public Razionale add(Razionale q) {
   long a = ((Long)c1).longValue();
   long b = ((Long)c2).longValue();
   long c = ((Long)q.c1).longValue();
   long d = ((Long)q.c2).longValue();
   return new Razionale(new Long(a*d+b*c),
                        new Long(b*d));
}

public Razionale mult(Razionale q) {
   long a = ((Long)c1).longValue();
   long b = ((Long)c2).longValue();
   long c = ((Long)q.c1).longValue();
   long d = ((Long)q.c2).longValue();
   return new Razionale(new Long(a*c),new Long(b*d));
} . . .
```

. . . Rappresentazione del tipo Razionale

```
  private static long mcd(long x, long y){
    long t,r;
    if ( x < y ) {/* se x < y li scambio */
      t = x;
      x = y;
      y = t;
    }
    while ( y > 0 ) {
      r = x % y;
      x = y;
      y = r;
    }
    return x;
  }
}
```

. . . Rappresentazione del tipo Razionale

- Le due funzioni **add** e **molt** sono state implementate con uno schema funzionale, ovvero esse non modificano gli oggetti su cui esse vengono invocate.

```
public Razionale add(Razionale q) {
    . . .
    return new Razionale(. . .);
}
```

Il tipo astratto Complesso . . .

- I numeri complessi sono formati da due parti, una parte reale ed una parte immaginaria:

$$a + ib$$

- Ancora una volta possiamo definire il tipo di dato astratto riprendendo il tipo Coppia, notando che:
  - I due domini T1 e T2 possono essere i due domini di valori che possono assumere parte reale e parte immaginaria del numero complesso
  - Le funzioni del tipo Complesso sono quelle classiche, ad esempio addizione, moltiplicazione, etc.

Rappresentazione del tipo Complesso. . .

- Come per il Razionale, una classe che rappresenta il tipo astratto Complesso può essere basata sulle seguenti scelte:
  - Rappresentiamo il dominio dei valori di interesse del tipo astratto con oggetti Coppia
  - Schema realizzativo funzionale
  - Interfaccia di classe: definiamo un metodo per ciascuna delle funzioni del tipo astratto
  - Realizziamo la funzione formaComplesso attraverso un costruttore

. . . Rappresentazione del tipo Complesso

La specifica del tipo Complesso, con i suoi domini (tra cui quello di interesse) e le sue funzioni, è quindi la seguente

Domini

- **Complesso : dominio di interesse del tipo**
- **T1 : dominio dei valori che può assumere la parte reale di un numero complesso**
- **T2 : dominio dei valori che può assumere la parte immaginaria di un numero complesso**
- **T3 : dominio dei valori che può assumere il modulo di un numero complesso**
- **T4 : dominio dei valori che può assumere la fase di un numero complesso**

. . . Rappresentazione del tipo Complesso

Funzioni (1 di 2)

- formaComplesso (T1 a, T2 b) → Complesso
  - pre: nessuna
  - post: RESULT è il valore corrispondente al numero complesso *a+ib*
- reale(Complesso c) → T1
  - pre: nessuna
  - post: RESULT è il valore corrispondente alla prima componente del numero complesso
- immaginaria(Complesso c) → T2
  - pre: nessuna
  - post: RESULT è il valore corrispondente alla seconda componente del numero complesso

. . . Rappresentazione del tipo Complesso

❑ Funzioni (2 di 2)

- modulo(Complesso c) → T3
  - pre: nessuna
  - post: RESULT è il valore corrispondente al modulo del numero complesso
- fase(Complesso c) → T4
  - pre: nessuna
  - post: RESULT è il valore corrispondente alla fase del numero complesso
- add(Complesso c1, Complesso c2) → Complesso
  - pre: nessuna
  - post: RESULT è il valore che rappresenta il numero complesso somma dei numeri complessi c1 e c2
- molt(Complesso c1, Complesso c2) → Complesso
  - pre: nessuna
  - post: RESULT è il valore che rappresenta il numero complesso prodotto dei numeri complessi c1 e c2

. . . Rappresentazione del tipo Complesso

```
public class Complesso extends Coppia {

  // realizzazione delle funzioni del tipo
  astratto
  public Complesso(Double r, Double i) {
    super(r, i);
  }

  public Double reale() {
    return ((Double) c1).doubleValue();
  }

  public Double immaginaria() {
    return ((Double) c2).doubleValue();
  } . . .
```

. . . Rappresentazione del tipo Complesso

```
public Double modulo() {
  double re = ((Double) c1).doubleValue();
  double im = ((Double) c2).doubleValue();
  return new Double(Math.sqrt(re * re + im * im));
}

public Double fase() {
  double re = ((Double) c1).doubleValue();
  double im = ((Double) c2).doubleValue();
  double ret;
  if (im >= 0)
    ret = Math.acos(re / modulo());
  else
    ret = -Math.acos(re / modulo());
  return new Double(ret);
} . . .
```

. . . Rappresentazione del tipo Complesso

```
  public Complesso add(Complesso cc) {
    double a = ((Double) c1).doubleValue();
    double b = ((Double) c2).doubleValue();
    double c = ((Double) cc.c1).doubleValue();
    double d = ((Double) cc.c2).doubleValue();
    return new Complesso(new Double(a+c),
                                    new Double(b+d));
  }

  public Complesso mult(Complesso cc) {
    double a = ((Double) c1).doubleValue();
    double b = ((Double) c2).doubleValue();
    double c = ((Double) cc.c1).doubleValue();
    double d = ((Double) cc.c2).doubleValue();
    return new Complesso(new Double(a*c - b*d),
                                    new Double(b*c + a*d));
  }
}
```

Esercizio

- Implementare il tipo di dato astratto Complesso basandosi sui valori di modulo e fase (insiemi T3 e T4 del ADT) piuttosto che parte reale e parte immaginaria (insiemi T1 e T2 del ADT)

Matrici sparse . . .

- Ci sono dei casi in cui le caratteristiche dei valori di una matrice, che si utilizzano in un programma, consentono una memorizzazione più efficiente rispetto a quella tradizionale con array
  - Ad esempio, per una matrice di interi in cui sia noto che la grande maggioranza degli elementi è uguale a zero, si potrebbero memorizzare i soli elementi diversi da zero
- Si usa il termine matrice sparsa . . .

. . . Matrici sparse

- Si usa il termine matrice sparsa per indicare una matrice in cui la gran parte degli elementi ha un valore prefissato (detto valore dominante)
- Per tali matrici si possono utilizzare rappresentazioni ad hoc, dette rappresentazioni compatte
- L'idea fondamentale di tali rappresentazioni è quella di memorizzare solo gli elementi con valore diverso dal valore dominante

Esempio di matrice sparsa

	0	1	2	3	4	5
0	8	0	0	11	0	0
1	0	0	21	0	0	0
2	0	0	0	15	0	0
3	0	0	3	16	0	0

- Elemento dominante = 0
- Ogni elemento con valore diverso dal dominante viene rappresentato da una terna
  - Indice di riga
  - Indice di colonna
  - Valore dell'elemento

La rappresentazione compatta di una matrice sparsa...

- Queste terne sono a loro volta rappresentate mediante un array C di oggetti Terna con tre attributi: riga, colonna e valore
- Un indice lastValue rappresenta l'indice dell'ultima terna significativa di C
- C viene detta rappresentazione compatta di A

. . . La rappresentazione compatta di una matrice sparsa...

- Gli oggetti Terna di C vengono memorizzati in modo ordinato, ad esempio ordinando per riga e, nell'ambito della stessa riga, per colonna
  - La prima terna di C è utilizzata per memorizzare le informazioni relative al numero di righe N ed il numero di colonne M di A, e nel campo relativo al valore, il valore dell' elemento dominante
  - Se max è il numero di componenti di C, e m (con m < max) è il numero di elementi di A memorizzati in C, le informazioni ad essi relative sono rappresentate negli elementi di indice da 1 a m di C, e nelle eventuali componenti successive di C (cioè quelle di indice maggiore di m) viene memorizzata terna di valori che non corrisponde ad alcuna componente della matrice A

...La rappresentazione compatta di una matrice sparsa

- Esempio: la rappresentazione compatta della seguente matrice

	0	1	2	3	4	5
0	8	0	0	11	0	0
1	0	0	21	0	0	0
2	0	0	0	15	0	0
3	0	0	3	16	0	0

è

	rig	col	val
0	4	6	0
1	0	0	8
2	0	3	11
3	1	2	21
4	2	3	15
5	3	2	3
6	3	3	16
7	9	9	0

lastValue = 6

Quando conviene usare la rappresentazione compatta

- Quando la matrice è sufficientemente sparsa
- Sia A una matrice di m righe e n colonne
- Se maxA è il numero di elementi che si devono memorizzare esplicitamente,
  - l'occupazione di memoria richiesta per rappresentare la matrice A in forma compatta è 3*(maxA+1) + 1
  - l'occupazione di memoria richiesta per rappresentare la matrice A in forma usuale è m*n
- La rappresentazione compatta risparmia memoria quando

$$3*(maxA+1) + 1 < m*n \ldots maxA < (m*n - 4)/3$$

La segnatura per il tipo matrice

```
interface MatriceDiInteri {
  boolean isEmpty();
  //post: restituisce true sse la matrice e'  vuota
  //r=c=0

  int numRighe();
  //post: numero di righe della matrice

  int numColonne();
  // post: numero di colonne della matrice

  int accedi(int r, int c);
  //pre: r,c>=0
  //post: restituisce l'lemento in pos izione r,c

  void memorizza(int r, int c, int n);
  //pre: r,c>=0
  //post: memorizza l'elemento n in posizione r,c
}
```

Due diverse implementazioni

- Matrice con array: rappresentazione di Java
- Matrice Compatta: rappresentazione efficiente di una matrice sparsa

...Matrice con Array...

- **Attributi**

```
class MatriceConArray implements MatriceDiInteri {
private int[][] mat;
private int nrighe;
private int ncolonne;
```

- **Il costruttore inizializza a 0 tutte le componenti**

```
public MatriceConArray (int r, int c) {//costruttore
  int i,j;
  this.mat = new int[r][c];
  //inizializza a zero
  for (i=0; i<r; i++)
    for(j=0; j<c; j++)
          this.mat[i][j] = 0;
  this.nrighe = r;
  this.ncolonne = c;
}
```

...Matrice con Array...

Metodi d'istanza

```
public int numRighe(){
// post: numero di righe della matrice
  return this.nrighe;
}

public int numColonne(){
// post: numero di colonne della matrice
  return this.ncolonne;
}

public boolean isEmpty() {
// post:  restituisce true sse la matrice
//        ha dimensioni nulle
  return (this.nrighe==0 && this.ncolonne==0);
} . . .
```

...Matrice con Array

```
public int accedi(int r, int c){
//pre: r,c>=0
//post:restituisce l'elemento in posizione r,c
  return this.mat[r][c];
}

public void memorizza(int r, int c, int n){
//pre: r,c>=0
//post:memorizza l'elemento n in posizione r,c
  this.mat[r][c] = n;
} . . .
```

...Matrice con Array

```
  public String toString(){
     int i,j;
     String res = "";
     for (i=0; i<nrighe; i++){
       for(j=0; j<ncolonne; j++)
         res = res + mat[i][j] +" ";
       res = res +"\n";
     }
     return res;
   }
}
```

Matrice Compatta: la classe terna ...

❑ Attributi

```
class Terna {
   int riga;
   int colonna;
   int valore;
```

❑ Costruttore

```
   public Terna(int r, int c, int v){
     riga = r;
     colonna = c;
     valore = v;
   }
```

❑ Metodi d'istanza

```
   public int getRiga(){
     return riga;  }
```

...Matrice Compatta: la classe terna

```
  public int getColonna(){
    return colonna;
  }
  public int getValore(){
    return valore;
  }
  public String toString(){
    return (riga +" " + colonna +" " + valore);
  }
}//end class Terna
```

Matrice Compatta...

```
class MatriceCompatta implements MatriceDiInteri {
  private Terna[] info;
  private int dim;              // dimensione dell'array
  private int lastValue;        // dimensione effettiva
                                // dell'array
```

- **Costruttore: inizializza tutte le terne a 0**

```
  public MatriceCompatta (int dim, int rig,
                                   int col, int dom) {
    int i;
    this.info = new Terna[dim];
    this.info[0] = new Terna(rig,col,dom);
    for(i = 1; i < dim; i++)
      this.info[i] = new Terna(0,0,0);
    this.dim = dim;
    this.lastValue = 0;
  }
```

...Matrice Compatta

```
public int numRighe() {
// post: numero di righe della matrice
  return this.info[0].riga; }

public int numColonne() {
// post: numero di colonne della matrice
  return this.info[0].colonna; }

public int getLastValue() {
// post: dimensioni effettive della matrice
  return this.lastValue; }

public boolean isEmpty() {
//post: restituisce true sse la matrice
//       ha dimensioni nulle
  return (this.info[0].riga==0 &&
                          this.info[0].colonna==0);
}
```

Matrice Compatta: il metodo accedi

```
public int accedi(int r, int c) {
//pre: r,c>=0
//post: restituisce l'elemento in posizione r,c
  int i = 1, res;
  while (info[i].riga < r && i < this.lastValue)
    i++; //esco sulla riga r se esiste
  while (info[i].riga == r && info[i].colonna < c
                         && i < this.lastValue)
    i++; //esco sulla colonna r se esiste
  if (info[i].riga == r && info[i].colonna == c)
    res = info[i].valore;
  else
    res = info[0].valore;
  return res;
}
```

Matrice Compatta: il metodo memorizza...

```
public void memorizza(int r, int c, int n) {
//pre: r,c>=0
//post: memorizza l'elemento n in posizione r,c
// se c'e' spazio

/* se si sostituisce un elemento diverso dal
dominante con uno dominante, si devono shiftare tutti
gli elementi in alto */
  int i = 1, j;
  /* vedo se la componente da memorizzare esiste */
  while (info[i].riga < r && i <= this.lastValue)
    i++; //esco sulla riga r se esiste
  while (info[i].riga == r && info[i].colonna < c
                        && i <= this.lastValue)
    i++;
```

...Matrice Compatta: il metodo memorizza...

```
    /* se il valore da memorizzare e' il dominante e
       se esiste gia' un valore nella matrice, allora
       devo eliminare la componente */
  if (n == info[0].valore) {
    if (info[i].riga == r && info[i].colonna == c) {
      //elimino l'el. in posizione i
      for (j = i+1; j < this.dim; j++)
        info[j-1] = info[j];
      this.lastValue = this.lastValue - 1;
    }
    /* se il valore da memorizzare e' dominante
       e se non esiste gia' un valore nella
       matrice, allora non devo fare niente */
  }
```

...Matrice Compatta: il metodo memorizza...

```
    else { // n != info[0].valore
    /*il val. da memorizzare non e' dominante:
      se gia' esiste sovrascrivo altrimenti aggiungo */
      if (info[i].riga == r && info[i].colonna == c)
        info[i].valore = n; // sovrascrivo
      else //aggiungo se c'e' posto
        if (this.lastValue == this.dim-1)
          System.out.println
                 ("** Memorizzazione impossibile **");
        else {
          for(j = this.lastValue; j >= i; j--)
            this.info[j+1] = this.info[j];
          this.lastValue = this.lastValue + 1;
          info[i] = new Terna(r,c,n);
        }
     }
}
```

...Matrice Compatta: il metodo toString

```
public String toString() {
   int i;
   String res;
   res = "";
   for (i=0; i<=lastValue; i++)
     res = res +(info[i].toString()) + "\n";
   return res;
 }
```

Matrice Compatta: esempi d'uso...

- La classe UsaMatrici usa una MatriceConArray

```
class UsaMatrici{

  public static void main(String[] args){
  MatriceConArray emme;

  emme = new MatriceConArray(4,6);
  System.out.println(emme.toString());
  emme.memorizza(1,2,9);
  emme.memorizza(3,4,7);
  emme.memorizza(2,2,8);
  System.out.println(emme.toString());
  }
}
```

Matrice Compatta: esempi d'uso...

- Esempio di esecuzione con MatriceConArray

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

0 0 0 0 0 0
0 0 9 0 0 0
0 0 8 0 0 0
0 0 0 0 7 0

Press any key to continue . . .
```

...Matrice Compatta: esempi d'uso...

- La classe UsaMatrici usa una MatriceCompatta

```
class UsaMatrici{
  public static void main(String[] args){
  MatriceCompatta emme;
  emme = new MatriceCompatta(5,4,6,0);
  emme.memorizza(1,2,9);
  emme.memorizza(3,4,7);
  emme.memorizza(2,2,8);
  System.out.println(emme.toString());
}}
```

Matrice Compatta: esempi d'uso...

- Esempio di esecuzione con MatriceCompatta

```
4 6 0

inserisco in riga 1, colonna 2 il valore 9
4 6 0
1 2 9

inserisco in riga 3, colonna 4 il valore 7
4 6 0
1 2 9
3 4 7

inserisco in riga 2, colonna 2 il valore 8
4 6 0
1 2 9
2 2 8
3 4 7
```

Esercizio

- Definire, secondo la segnatura `MatriceDiInteri` una classe concreta che rappresenta una matrice compatta usando una sequenza di terne (struttura collegata) invece di un array di terne