Capitolo 5

Uso degli oggetti

Andrea Gini

Il paradigma della programmazione a oggetti si è affermato gradualmente a partire dalla metà degli anni Ottanta per le sue caratteristiche di eleganza e naturalezza. La metafora degli oggetti, grazie alla sua analogia con il mondo reale, offre una modalità di rappresentazione dei problemi intuitiva, naturale e a volte persino divertente. Nei prossimi capitoli verranno introdotti in modo graduale i principi base della programmazione in Java, dall'utilizzo di oggetti di libreria alla progettazione di classi fino agli aspetti più sottili, e spesso trascurati o fraintesi, della filosofia degli oggetti.

La metafora degli oggetti

Durante la progettazione di un sistema informatico, viene spontaneo modellare la struttura e il comportamento dei programmi secondo una metafora che rispecchi l'attività che si intende automatizzare. Le attività del mondo reale sono classificabili come trasformazioni tra oggetti: quando si cucina, tanto per fare un esempio, si trasforma un insieme di ingredienti in una pietanza. Per portare a termine tali compiti, è normale fare ricorso a un certo numero di strumenti: nell'esempio della cucina si può pensare a pentole, posate e fornelli. Nel gergo della programmazione a oggetti non si distingue tra gli oggetti che provocano trasformazioni e quelli che le subiscono (strumenti e ingredienti): gli uni e gli altri vengono descritti col termine generico di *oggetti*. Si esamini ora una procedura che descrive le operazioni necessarie a lessare una patata:

1. Sciacqua la patata sotto al rubinetto fino a quando è pulita.

2. Sbuccia la patata con un coltello.

3. Riempi una pentola con l'acqua del rubinetto.
4. Accendi un fornello nella cucina.
5. Metti la patata nella pentola.
6. Fai bollire la patata fino a che non è cotta.
7. Spegni il fornello.
8. Togli la patata dalla pentola con una forchetta.

Per portare a termine il compito "cottura di una patata" vengono utilizzati 6 oggetti di uso comune: una patata, un coltello, un rubinetto, una pentola, un fornello e una forchetta. Nel corso dell'operazione, vengono svolte alcune interazioni tra questi oggetti: per sbucciare la patata si utilizza un coltello; per scaldare la pentola si usa un fornello e per cuocere la patata si utilizza la pentola. Le interazioni sono rese possibili dalla natura dell'oggetto stesso: la lama del coltello serve a tagliare, il fuoco a cuocere e così via. Alcuni degli oggetti coinvolti subiscono delle trasformazioni o dei cambiamenti di stato: la patata, inizialmente cruda, è stata dapprima lavata (prima trasformazione), poi sbucciata (seconda trasformazione) e infine cucinata; il fornello, inizialmente spento (primo stato), è stato dapprima acceso (secondo stato) e poi spento di nuovo (terzo stato). D'altra parte, nel descrivere la procedura è stata utilizzato un linguaggio di tipo algoritmico, ricorrendo ai familiari costrutti di sequenza, iterazione e selezione.

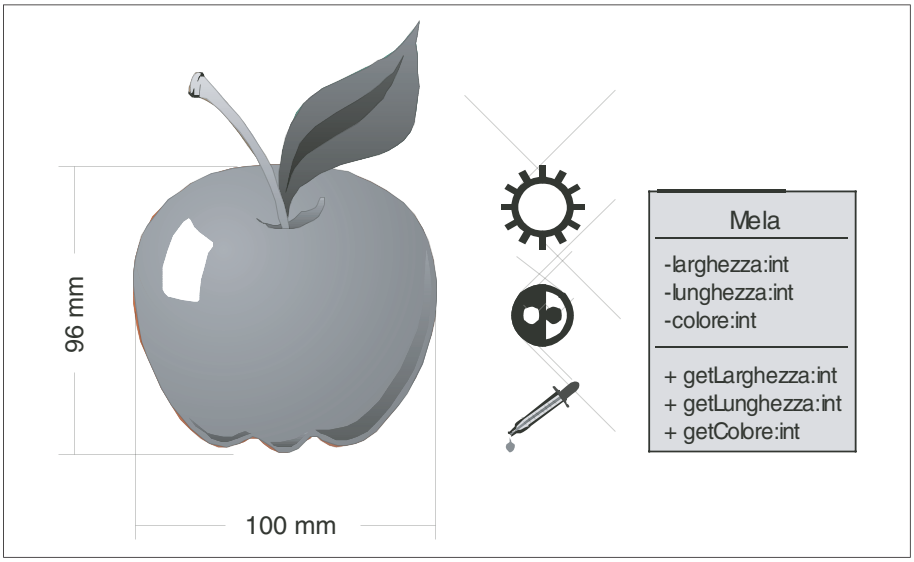
Prima di introdurre la metafora degli oggetti nel mondo della programmazione è bene chiarire i concetti di stato e comportamento negli oggetti del mondo reale.

Lo stato di un oggetto e i suoi attributi

Ogni oggetto del mondo reale può essere descritto mediante un certo numero di attributi, come forma, dimensioni e colore. Qualsiasi qualità misurabile o enumerabile può costituire un attributo. Alcuni attributi, come le dimensioni, sono presenti in tutti gli oggetti; altri invece sono caratteristici di una particolare *classe* di oggetti e non compaiono in altri: "crudo" e "cotto" sono attributi che hanno senso per una patata, ma di certo non per un coltello.

Lo *stato* di un oggetto è l'insieme dei valori dei suoi attributi. Nel corso del suo ciclo di vita, un oggetto può cambiare stato per diverse ragioni: alcuni attributi possono essere modificati dall'utilizzatore tramite un'interazione diretta (il fornello può essere acceso e spento direttamente, utilizzando l'apposita manopola), mentre altri cambiano a seguito di interazioni con altri oggetti (la patata può essere sbucciata solo ricorrendo a un coltello) e altri, infine, cambiano spontaneamente per ragioni interne all'oggetto stesso (una patata matura spontaneamente nel corso del tempo). D'altra parte, ci sono anche attributi immutabili: il peso del coltello, per esempio, non può cambiare, a meno che il coltello non venga distrutto.

Figura 5.1 – *Ogni oggetto può essere descritto mediante il valore dei suoi attributi.*



Le modalità di utilizzo di un oggetto

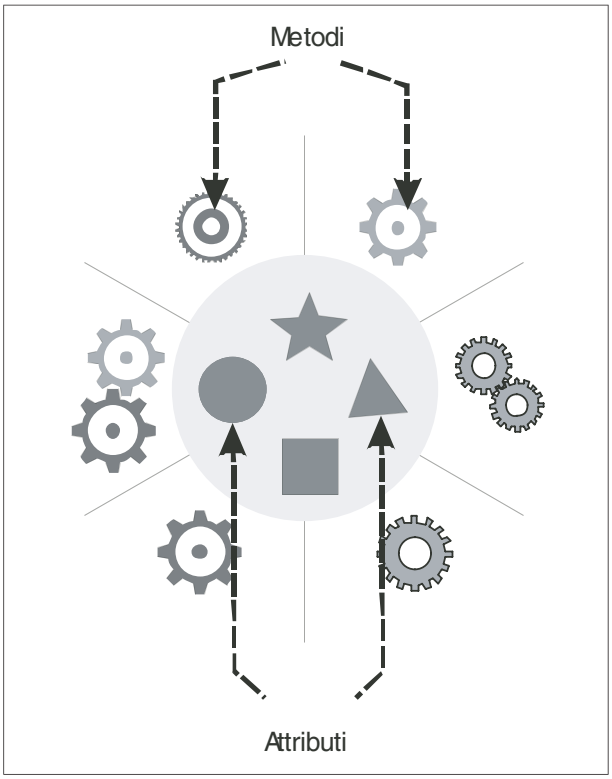
Ogni oggetto ha specifiche modalità di utilizzo: il coltello può tagliare, il fornello può scaldare, la forchetta può infilzare. Alcuni oggetti hanno una sola modalità di utilizzo, mentre altri ne hanno più di una. Un normale coltello da cucina, tanto per fare un esempio, ha almeno tre modalità di utilizzo: taglia, spalma e infilza.

Le modalità di utilizzo di un oggetto caratterizzano il suo comportamento, dal momento che esse costituiscono l'unica via per interagire con l'oggetto stesso. Per portare a termine compiti complessi, normalmente si utilizzano più oggetti, facendoli interagire tra loro. Le modalità di utilizzo prevedono di solito limiti espliciti alla possibilità di interazione con altri oggetti: un coltello da cucina permette di sbucciare una mela, ma non di abbattere un albero, cosa che invece è possibile fare con la lama di un machete. Il coltello da cucina e il machete sono oggetti simili tra loro, dal momento che sono entrambi attrezzi da taglio, ma è evidente a livello intuitivo che le rispettive modalità di utilizzo sono diverse.

La metafora degli oggetti nella programmazione

Un oggetto è un'entità software dotata di uno stato (i suoi attributi) e di un insieme di metodi che permettono all'utente di interagire con esso. Lo stato di un oggetto è accessibile solo tramite i suoi metodi: per questo, spesso gli oggetti vengono rappresentati come "contenitori" di attributi, che mostrano al proprio esterno soltanto i metodi.

Figura 5.2 – *Rappresentazione grafica di un oggetto software.*



Gli oggetti software, al pari degli oggetti reali, possiedono una loro coerenza, che ne favorisce il corretto utilizzo. Verrà ora mostrato come si creano e si utilizzano gli oggetti in un qualsiasi programma Java: sarà interessante notare la somiglianza tra un algoritmo che fa uso di oggetti e la descrizione dell'operazione di cottura di una patata vista nei paragrafi precedenti.

Creazione di un oggetto

Prima di usare un oggetto all'interno di un programma è necessario crearlo. La creazione di un oggetto richiede tre fasi: dichiarazione, creazione e assegnamento, in modo simile a quello che avviene per gli array. Per esempio, ecco le fasi di creazione di un ipotetico oggetto "Patata", a partire dalla dichiarazione:

```
Patata p;
```

La dichiarazione costruisce una variabile dello stesso tipo dell'oggetto che si desidera creare.

Tale variabile non è l'oggetto vero e proprio bensì un reference, ossia una specie di "centrale di controllo" che permette di comunicare con l'oggetto vero e proprio. Senza una centrale di controllo, l'oggetto risulta irraggiungibile dal programma, e gli oggetti irraggiungibili vengono classificati come rifiuti (garbage) ed eliminati. Come si è già visto per i vettori, è possibile avere più di un reference allo stesso oggetto. La creazione e l'assegnamento richiedono l'uso della parola riservata new:

```
p = new Patata();
```

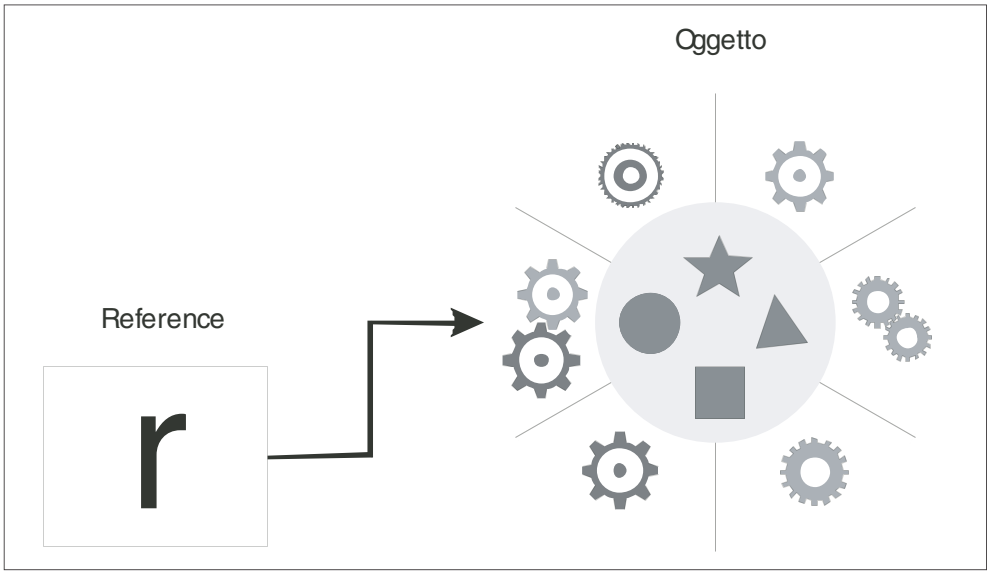
Ovviamente è possibile dichiarare, creare e assegnare un oggetto in un'unica riga:

```
Patata p = new Patata();
```

Alcuni oggetti richiedono uno o più parametri in fase di creazione: tali parametri sono necessari per impostare lo stato iniziale dell'oggetto. Si immagini un oggetto "Cucina" che permetta di stabilire, in fase di creazione, il numero di fornelli a gas e di quelli elettrici: per creare una cucina con quattro fornelli a gas e uno elettrico sarebbe necessario utilizzare un'istruzione del tipo:

```
Cucina c = new Cucina(4,1);  // quattro fornelli a gas e uno elettrico
```

Figura 5.3 – *Un reference è una centrale di controllo che permette di inviare le direttive all'oggetto vero e proprio.*



Il comportamento di un oggetto e i suoi metodi

Per eseguire un'azione su un oggetto è necessario effettuare una chiamata a metodo. Tale operazione richiede tre informazioni: l'oggetto su cui invocare il metodo, il nome del metodo e i parametri richiesti dal metodo stesso.

La sintassi della chiamata a metodo in Java ha la forma:

```
oggetto.metodo(p1,p2,…,pn);
```

I metodi possono anche restituire un valore; il valore di ritorno e i parametri possono essere sia valori primitivi (int, boolean, float ecc.) sia oggetti.
Nell'esempio seguente:

```
Soldi s = banca.riscuoti(assegno);
```

la chiamata al metodo riscuoti() dell'oggetto banca applicata a un oggetto di tipo assegno restituisce un oggetto di tipo Soldi.

Gli attributi di un oggetto e il suo stato

Gli attributi sono valori che descrivono lo stato di un oggetto. In un oggetto software, gli attributi sono accessibili solo tramite i metodi; questa proprietà permette a chi progetta l'oggetto di proteggere l'integrità dell'oggetto stesso in almeno due maniere:

- Impedendo la modifica di un attributo al fine di renderlo accessibile solo in lettura.
- Imponendo precise restrizioni ai valori che un determinato attributo può assumere, in modo da vietare l'uso improprio dell'oggetto.

Di norma, gli attributi sono accessibili mediante opportuni metodi, detti getter e setter", così definiti a causa del prefisso "get" o "set".. I metodi getter permettono di leggere il valore di un attributo, e i setter di modificarlo:

```
int saldo = contoCorrente.getSaldo();          // interroga il conto corrente per conoscere il saldo
automobile.setRapportoDiCambio(4);             // imposta la quarta marcia sull'oggetto automobile
```

In alcuni casi, gli attributi non possono essere modificati direttamente tramite un metodo setter: essi cambiano in seguito alla chiamata di altri metodi che producono una transizione di stato complessa. Nell'esempio della cucina, è abbastanza intuitivo pensare che una pentola disponga di un metodo getTemperatura() ma non del corrispondente metodo setTemperatura(), dato che l'unico modo per alzare la temperatura di una pentola è quella di scaldarla sul fuoco:

```
fornello.accendi();                          // accende il fornello
while(pentola.getTemperatura() < 100)        // scalda la pentola fino a che non raggiunge i 100°C
  fornello.scalda(pentola);
fornello.spegni();                           // spegne il fornello
```

Interazione complessa tra oggetti

Per mostrare un esempio concreto di interazione tra oggetti, verrà ora mostrato come la procedura di cottura di una patata, descritta in modo informale nei paragrafi precedenti, verrebbe descritta in Java:

```
Fornello f = new Fornello();                 // Crea il Fornello
Coltello c = new Coltello();                 // Crea il Coltello
Patata p = new Patata();                     // Crea la Patata
Rubinetto r = new Rubinetto();               // Crea il Rubinetto
Pentola pent = new Pentola();                // Crea la Pentola
Forchetta fork = new Forchetta();            // Crea la Forchetta

While(p.èSporca())                           // Fino a che la patata è sporca
  r.risciaqua(p);                            // Risciacqua la patata con il rubinetto
c.sbuccia(p);                                // Sbuccia la patata con il coltello
pent.riempi(r);                              // Riempi la pentola con il rubinetto
f.accendi()                                  // Accendi il fornello
pent.inserisci(p);                           // Metti la patata nella pentola
while(p.èCruda())                            // Finché la patata è cruda
f.scalda(pent);                              // scalda la pentola sul primo fornello

f.spegni();                                  // Spegni il primo fornello
fork.inforca(p)                              // Raccogli la patata con la forchetta
```

Si può già notare come gli oggetti aggiungano espressività al codice, e come tendano a fornire una dimensione materiale alle entità evanescenti che compongono un programma. Come diverrà più chiaro in seguito, il ricorso agli oggetti fornisce un approccio concettuale di altissimo livello a qualsiasi contesto applicativo, favorendo la modularità e la riutilizzabilità dei diversi componenti di un programma. Grazie agli oggetti, il lavoro di programmazione diventa qualcosa di simile a un gioco di costruzioni a incastro, come il Lego.

Oggetti di sistema

Dopo aver introdotto in modo informale l'uso degli oggetti nella programmazione, è ora di studiare alcuni oggetti di sistema che vengono usati nella grande maggioranza dei programmi realizzati con Java. Il primo di questi è la stringa, che permette di manipolare sequenze di caratteri e comporre parole e frasi. A differenza che in altri linguaggi, come C e Pascal, in Java le

stringhe vengono implementate come oggetti, con il vantaggio di permetterne un uso ad alto livello. Dopo le stringhe verranno illustrati i vettori dinamici e le mappe hash, due importanti strutture di dati. Infine, verranno trattate le wrapper class, che permettono di trattare come oggetti anche i valori appartenenti ai tipi primitivi.

Stringhe

L'oggetto String contiene una sequenza di caratteri di lunghezza arbitraria, che può essere utilizzata per memorizzare parole, frasi o testi di qualsiasi dimensione. Il contenuto di un oggetto String viene deciso al momento della sua creazione, e non può essere modificato in seguito:

```
String s = new String("Questo è un esempio di stringa di testo");
```

I metodi di String permettono di ispezionare il contenuto della stringa, di estrarne dei frammenti (sottostringhe), di verificare l'uguaglianza con un'altra stringa e di effettuare altre interessanti operazioni. Nei prossimi paragrafi verranno analizzate a fondo le caratteristiche di questo oggetto fondamentale.

Le stringhe e Java: creazione, concatenazione e uguaglianza

Le stringhe sono uno strumento così importante nella programmazione che i progettisti di Java hanno deciso di introdurre dei costrutti nel linguaggio per semplificarne l'uso.

Creazione

In fase di creazione, invece di ricorrere all'operatore new, è possibile utilizzare direttamente un letterale stringa, ossia una qualsiasi sequenza di caratteri racchiusa tra doppi apici come nel seguente esempio:

```
String s = "Ciao";                    // è eqivalente a String s = new String("Ciao");
```

Ogni letterale stringa viene considerato automaticamente come un oggetto String, pertanto è possibile applicare la chiamata a metodo direttamente su di esso. La seguente riga di codice:

```
System.out.println("Ciao".length());
```

stampa a schermo la *lunghezza* della stringa, ossia il risultato della chiamata del metodo length() sulla stringa "Ciao", e non il suo contenuto.

Concatenazione

Java prevede l'uso di + come operatore di concatenazione in alternativa al metodo concat(String s). Grazie all'operatore +, un'istruzione come:

```
String s = "Ecco ".concat("una ").concat("stringa ").concat("concatenata");
```

può essere sostituita dalla più sintetica e intuitiva:

```
String s = "Ecco " + "una " + "stringa " + " concatenata";
```

L'operatore di concatenazione è particolarmente utile quando si devono creare stringhe così lunghe da occupare più di una riga. Dal momento che i letterali stringa non possono essere interrotti con un ritorno a capo, è possibile concatenare più letterali stringa da una riga ciascuno ricorrendo all'operatore +:

```
String s = "La vispa Teresa, " +
           "avea tra l'erbetta " +
           "a volo sorpresa " +
           "gentil farfalletta";
```

L'operatore + effettua in modo automatico e trasparente la conversione dei tipi primitivi in stringa:

```
int risultato = 12 * a;
System.out.println("Il contenuto della variabile risultato è: " + risultato);
```

Operatore di uguaglianza e metodo equals

Quando si lavora su oggetti o vettori, bisogna fare molta attenzione all'uso dell'operatore di uguaglianza == che, a differenza di quanto ci si potrebbe aspettare, non serve a testare l'uguaglianza di contenuto, ma solo l'*uguaglianza di riferimento*. In figura 5.4 si vedono due variabili s1 e s2 che puntano a due distinti oggetti stringa. Nonostante entrambe le stringhe contengano la parola "Ciao!", il test con l'operatore == restituirà false, dal momento che gli oggetti puntati dalle due variabili sono due oggetti differenti, localizzati in due zone di memoria distinte. Per verificare se due stringhe hanno lo stesso contenuto, come in questo caso, è necessario ricorrere al metodo equals(String s).

In figura 5.5, invece, le variabili s1 e s2 puntano allo stesso oggetto stringa: in questo caso, il test con l'operatore == restituirà true. Scenari di questo tipo si presentano ogni volta che si assegna lo stesso oggetto a più di una variabile:

```
s1 = "Ciao! ";
s2 = s1;                                    // s1 e s2 puntano allo stesso oggetto stringa
```

Figura 5.4 – *Le variabili s1 e s2 puntano a due oggetti stringa con lo stesso contenuto.*

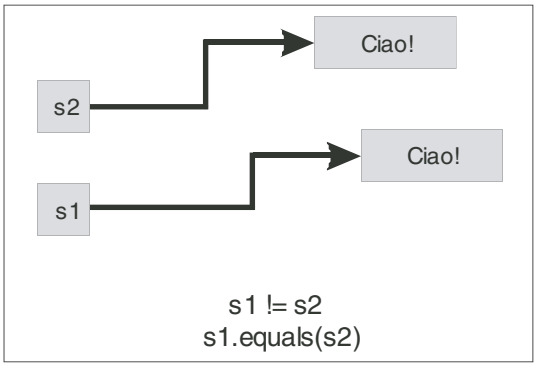
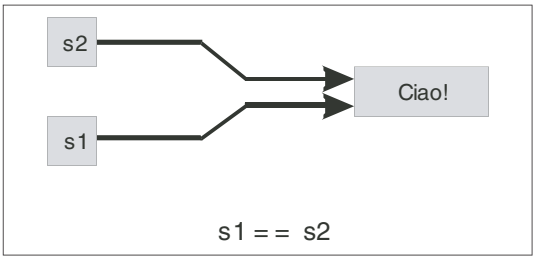


Figura 5.5 – *Le variabili s1 e s2 puntano allo stesso oggetto stringa.*



Operazioni fondamentali

L'oggetto String comprende più di 50 metodi; quelli fondamentali, comunque, sono appena 6:

- String substring(int beginIndex , int endIndex): restituisce una sottostringa che contiene il testo incluso tra beginIndex e endIndex-1.
- char charAt(int i): restituisce l'i-esimo carattere della stringa.
- int length(): restituisce la lunghezza della stringa.
- int compareTo(String anotherString): effettua una comparazione con un'altra stringa, e restituisce: 0 se l'argomento è una stringa uguale; un valore inferiore a 0 se l'argomento è una stringa più avanti rispetto all'ordine alfabetico; un valore superiore a 0 in caso contrario. Se si desidera che nella comparazione non venga considerata la differenza tra maiuscole e minuscole, si può utilizzare il metodo compareToIgnoreCase(String s).

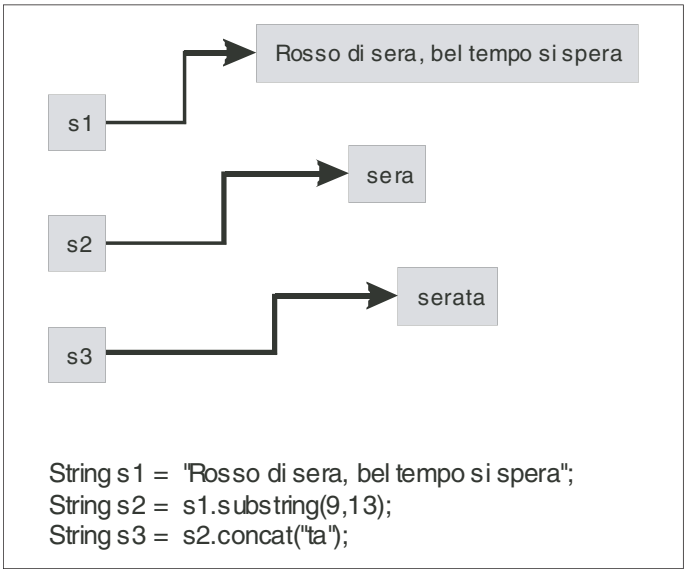
- String concat(String str): effettua la concatenazione con la stringa str. Come si è già visto, è equivalente all'operatore +.

- boolean equals(Object o): restituisce true se la stringa risulta uguale all'oggetto passato come parametro. Se si desidera che venga ignorato l'ordine tra maiuscole e minuscole, si può utilizzare il metodo equalsIgnoreCase(String s).

Si noti che i metodi concat() e substring() non modificano l'oggetto stringa su cui vengono invocati, ma creano un nuovo oggetto che ha per valore il risultato dell'operazione. In figura 5.8 si può osservare una rappresentazione della memoria dopo l'esecuzione delle seguenti istruzioni:

```
String s1 = "Rosso di sera, bel tempo si spera";
String s2 = s1.substring(9,13);
String s3 = s2.concat("ta");
```

Figura 5.6 – *Rappresentazione della memoria del computer dopo l'esecuzione di una serie di operazioni su stringhe.*



Altri metodi utili

L'oggetto String prevede diversi altri metodi utili. Ecco un gruppo di metodi che effettuano interrogazioni:

- boolean startsWith(String prefix): restituisce true se la stringa inizia con il prefisso specificato dal parametro.

- boolean endsWith(String suffix): restituisce true se la stringa finisce con il suffisso specificato dal parametro.

- int indexOf(String str): scandisce la stringa dall'inizio alla fine, e verifica se contiene o meno la stringa specificata come parametro. Se la trova, restituisce l'indice del primo carattere della sotto stringa cercata, altrimenti restituisce –1.

Un esempio di uso di questi metodi è il controllo di validità dell'estensione di un file:

```
import javax.swing.*;

public class ProvaEndsWith {

  public static void main(String argv[]) {
    String file = JOptionPane.showInputDialog(null, "Inserisci il nome di un file java valido");
    if( file.endsWith(".java") )
      JOptionPane.showMessageDialog(null,"Il nome del file è valido");
    else
      JOptionPane.showMessageDialog(null,"Il nome del file non termina con .java");
  }
}
```

Un altro gruppo di metodi fornisce alcune funzionalità utili in fase di manipolazione:

- String replace(char oldChar , char newChar): restituisce una stringa in cui il carattere specificato dal primo parametro è stato sostituito da quello indicato dal secondo.

- String toLowerCase(): restitituisce una stringa in cui tutti i caratteri sono minuscoli.

- String toUpperCase(): restitituisce una stringa in cui tutti i caratteri sono maiuscoli.

- String trim(): restituisce una stringa dalla quale sono stati rimossi gli eventuali spazi all'inizio e alla fine.

Come esempio riepilogativo, ecco un programma che inverte le frasi fornite in input:

```
import javax.swing.*;

public class Invertitore {

  public static void main(String argv[]) {
```

```
    String s = JOptionPane.showInputDialog(null, "Inserisci una frase");
    s = s.trim();
    String inversa = "";

    for( int i = s.length() -1 ; i >= 0  ; i-- )
      inversa = inversa + s.charAt(i);

    JOptionPane.showMessageDialog(null,inversa);
  }
}
```

Il programma chiede all'utente di inserire una frase, quindi la scandisce lettera per lettera dall'ultima alla prima e le aggiunge alla stringa "inversa", creando in tal modo una frase speculare rispetto a quella inserita dall'operatore.

in questi e nei prossimi esempi si farà uso di piccole finestre di dialogo per l'input e l'output dei dati sullo schermo. La forma base dell'istruzione di input è:

```
String s = JOptionPane.showInputDialog(null, messaggio)
```

Questa istruzione mostra all'utente una finestra di dialogo con il messaggio specificato dal parametro, e restituisce la stringa inserita dall'operatore. Per creare invece una finestra di output che visualizzi a schermo un messaggio si ricorre alla seguente istruzione:

```
JOptionPane.showMessageDialog(null,messaggio);
```

Se si desidera utilizzare queste istruzioni nei propri programmi, è necessario inserire la direttiva:

```
import javax.swing.*;
```

all'inizio del programma. Una descrizione completa dei controlli grafici e del loro utilizzo è rimandata ai capitoli 12, 13, 14 e 15.

Vettori dinamici

Nel capitolo 2 sono stati introdotti gli array, che permettono di lavorare su insiemi di variabili di lunghezza prefissata. Il principale difetto degli array è la loro natura statica: la dimensione viene stabilita al momento della creazione, e non può cambiare in alcun modo. Questa limitazione viene superata dai vettori dinamici, entità software simili agli array ma senza il limite della dimensione prefissata. Le librerie Java offrono un buon numero di vettori dinamici, simili tra

loro nell'interfaccia di programmazione ma diversi nell'implementazione interna: il più usato di questi è senza dubbio Vector.

Uso di Vector

Vector è un oggetto concettualmente simile a un array, ma a differenza di quest'ultimo prevede un utilizzo esclusivamente orientato agli oggetti. Esso va in primo luogo creato allo stesso modo di qualsiasi altro oggetto Java:

```
Vector v = new Vector();
```

Successivamente, esso può essere riempito mediante il metodo add(Object o), che aggiunge in coda alla lista l'oggetto passato come parametro:

```
String nome = "Un nome";
v.add(nome);
```

A differenza dell'array, che viene creato in modo da contenere valori di un certo tipo, il Vector memorizza oggetti di tipo generico: per questa ragione, al momento del prelievo, è necessario ricorrere all'operatore di casting per riconvertire l'elemento al suo tipo di origine:

```
String n = (String)v.get(1);
```

L'approccio completamente orientato agli oggetti ha permesso di incorporare un certo numero di funzionalità all'interno di Vector, e di renderle disponibili sotto forma di metodi. Per esempio, se si desidera conoscere la posizione di un elemento nella lista, è sufficiente chiamare l'apposito metodo di ricerca:

```
int position = v.indexOf("Un nome");
```

Se invece si vuole creare una copia del vettore, si può utilizzare il metodo clone():

```
Vector nuovoVector = (Vector)v.clone();
```

> Nota: per usare il Vector all'interno di un programma è necessario aggiungere la direttiva
>
> ```
> import java.util.*;
> ```
>
> in testa al programma. Il significato e l'uso della direttiva import verranno chiariti nel capitolo 11.

Metodi fondamentali di Vector

Ecco un elenco e una descrizione dei 9 metodi principali di Vector:

- boolean add(Object o): aggiunge l'elemento in coda alla lista.
- void add(int index , Object element): aggiunge un elemento nella posizione specificata; gli elementi successivi vengono spostati in avanti di uno.
- void clear(): svuota completamente la lista.
- boolean isEmpty(): restituisce true se la lista è vuota.
- Object get(int i): restituisce l'i-esimo elemento della lista.
- int indexOf(Object o): restituisce l'indice dell'elemento passato come parametro, o –1 se l'elemento non è presente nella lista.
- Object remove(index i): rimuove l'i-esimo elemento della lista, e sposta l'indice di tutti gli elementi successivi in avanti di un valore. Il metodo restituisce l'elemento appena rimosso.
- Object set(int i , Object element): mette l'elemento specificato in i-esima posizione, in sostituzione dell'elemento preesistente. Il metodo restituisce l'elemento appena rimosso.
- int size(): restituisce la dimensione della lista.

Iterator

Per effettuare una determinata operazione su tutti gli elementi di un vettore dinamico, è possibile ricorrere a un ciclo for simile a quelli che si usano con gli array:

```
for(int i=0;i<v.size();i++)
  System.out.println((String)v.get(i));
```

Tuttavia, per semplificare questo tipo di operazione e per aggiungere un tocco di eleganza, è possibile ricorrere a una soluzione orientata agli oggetti che prevede il ricorso a un apposito oggetto, Iterator, che può essere prelevato dal Vector mediante un apposito metodo:

```
Iterator i = nomi.iterator();
```

Iterator è un oggetto molto semplice e intuitivo, che permette di scandire la lista dal primo all'ultimo elemento per effettuare una determinata operazione su ciascuno dei suoi elementi. L'utilizzo canonico di un Iterator ha la forma:

```
while(i.hasNext()) {                    // verifica se c'è un ulteriore elemento
  String s = (String)i.next();          // preleva l'elemento
  System.out.println(s);                // lo utilizza
}
```

Conversione in array

Il Vector dispone anche di un metodo che permette di trasferire il suo contenuto in un array:

```
Object[] toArray(Object[] a)
```

L'array fornito come parametro viene ridimensionato, riempito con gli elementi del Vector e restituito all'utente. Dal momento che il Vector memorizza gli elementi con il tipo generico Object, è necessario passare come parametro un array del tipo giusto, e utilizzare l'operatore di casting sul valore di ritorno:

```
String[] lista = (String[])v.toArray(new String[0]);
```

Un esempio riepilogativo

Un piccolo esempio permetterà di illustrare un uso tipico di Vector. All'utente viene richiesto di inserire un elenco di nomi mediante una serie di finestre di dialogo; non appena l'utente avrà segnalato, premendo il pulsante Annulla, la conclusione della fase di inserimento, il vettore verrà scandito con un Iterator, e i suoi valori verranno stampati a schermo:

```
import java.util.*;
import javax.swing.*;

public class ListaNomi {

  public static void main(String argv[]) {
    Vector v = new Vector();

    while(true) {
      String nome = JOptionPane.showInputDialog(null, "Inserisci un nome");
      if(nome == null || nome.equals(""))
        break;
      else
        v.add(nome);
    }

    Iterator i = v.iterator();

    System.out.println("I nomi inseriti sono:");
    while(i.hasNext()) {
      System.out.println((String)i.next());
    }
  }
}
```

Mappe hash

La mappa hash è un contenitore di oggetti simile al vettore, che memorizza una collezione di oggetti associandoli a stringhe di testo invece che alla loro posizione. La mappa hash più comunemente utilizzata è Hashtable:

```
Hashtable h = new Hashtable();
```

Per depositare elementi nella mappa, bisogna utilizzare il metodo put(Object chiave,Object valore), che richiede come chiave un valore univoco (tipicamente una stringa di testo, ma può andare bene qualsiasi tipo di oggetto) e come valore l'oggetto da memorizzare:

```
h.put("Nome","Mario");
h.put("Cognome","Rossi");
h.put("Età","25 anni");
h.put("Lavoro","Insegnante");
```

Tabella 5.1 – *Una tabella hash è una collezione di oggetti, in cui gli elementi sono indicizzati tramite stringhe chiave.*

Chiave	Valore
Nome	Mario
Cognome	Rossi
Età	25 anni
Lavoro	Insegnante

La mappa può ovviamente contenere elementi duplicati; non può invece associare più di un elemento alla stessa chiave. Per recuperare l'elemento, sarà sufficiente fornire la chiave attraverso il metodo get:

```
String s = (String)h.get("nome");
```

Anche in questo caso, in fase di recupero è necessario ricorrere all'operatore di casting.

Metodi principali

I metodi principali delle mappe hash sono:

- Object put(Object key , Object value): inserisce un nuovo oggetto nella mappa, associandolo alla chiave specificata.

- Object get(Object key): preleva dalla mappa l'oggetto associato con la chiave specificata.
- Object remove(Object key): rimuove dalla mappa l'oggetto associato alla chiave specificata.
- void clear(): svuota la mappa.
- int size(): restituisce il numero di coppie chiave-valore contenute nella mappa.
- boolean isEmpty(): restituisce true se la mappa è vuota.
- boolean containsKey(Object key): verifica se la mappa contiene la chiave specificata.
- boolean containsValue(Object value): verifica se la mappa contiene il valore specificato.

Estrazione dell'insieme di chiavi o valori

Per ottenere l'insieme delle chiavi o dei valori, esiste un'apposita coppia di metodi:

- Set keySet(): restituisce l'insieme delle chiavi.
- Collection values(): restituisce la lista dei valori.

Questi metodi restituiscono oggetti di tipo Set e Collection. Entrambi questi oggetti hanno metodi comuni a Vector, e in particolare il metodo iterator() che rende possibile la scansione degli elementi:

```
Iterator keyIterator = h.keySet().iterator();
System.out.println("L'insieme delle chiavi è:");
while(keyIterator.hasNext())
  System.out.println((String)keyIterator.next());

Iterator elementIterator = h.values().iterator();
System.out.println("L'insieme degli elementi è:");
while(elementIterator.hasNext())
   System.out.println((String)elementIterator.next());
```

Wrapper class

Vector e Hashtable possono memorizzare solamente oggetti: se si desidera memorizzare al loro interno valori primitivi è necessario racchiuderli in apposite wrapper class (classi involucro). Il linguaggio Java prevede una wrapper class per ogni tipo primitivo: Boolean, Byte, Short, Integer, Long, Float e Double. Il loro uso è abbastanza semplice: in fase di creazione è sufficiente specificare nel costruttore il valore da inglobare:

```
Integer i = new Integer(15);
```

È disponibile anche un costruttore che accetta valori sotto forma di String, una funzionalità estremamente utile in fase di inserimento dati:

```
String numeroStringa = JOptionPane.showInputDialog(null, "Inserisci un numero intero");
Integer numeroInteger = new Integer(numeroStringa);
int numero = numeroInteger.intValue();
```

Per recuperare il valore nel suo formato naturale, ogni wrapper class dispone di un apposito metodo:

boolean BooleanValue()	su oggetti di tipo Boolean.
byte byteValue()	su oggetti di tipo Byte.
short shortValue()	su oggetti di tipo Short.
int intValue()	su oggetti di tipo Integre.
long longValue()	su oggetti di tipo Long.
float floatValue()	su oggetti di tipo Float.
double doubleValue()	su oggetti di tipo Double.