Programmazione di base in Java

Paolo Bison

Technical Report 12/00, LADSEB-CNR
Padova, Italy, Novembre 2000.

LADSEB-CNR
Corso Stati Uniti 4
I-35127 Padova, Italy
e-mail:paolo@ladseb.pd.cnr.it
tel: 049 8295765
fax: 049 8295763

Printed on October 12, 2001

SOMMARIO

Questo rapporto inizia con una breve introduzione alla programmazione ad oggetti ed una sintetica descrizione del linguaggio di programmazione Java. Poi prosegue con lo sviluppo di programmi in Java che realizzano algoritmi base dell'informatica, spaziando da semplici e banali programmi procedurali fino alla realizzazione di programmi basati sulla programmazione ad oggetti. Vengono inoltre illustrati programmi ricorsivi, di ricerca ed ordinamento, di gestione di strutture dinamiche e di ingresso/uscita. Infine vengono presentati i risultati in termini di tempi di calcolo per alcuni sempli benchmarks.
Tutti i programmi sono stati sviluppati ed eseguiti su una workstation Sun utilizzando l'ambiente di sviluppo JDK (Java Development Kit).

ABSTRACT

In the first part of this report a short introduction to Object Oriented programming and a brief description of Java programming language are given. Then the use of this language is shown through programs, which realize some basic algorithms, starting from simple programs to complex full OO programs. Moreover, there are programs which deals with the following points: recursive programming, search and sort algorithms, dynamic structures and input/output programming. At the end, the perfomance of the Java run-time system are shown through simple benchmarks.
All programs has been developed and tested on a Sun workstation using the Java Development Kit (JDK).

INDICE

Capitolo 1

Programmazione ad oggetti

In questo capitolo verranno introdotti brevemente i concetti generali della programmazione ad oggetti che si ritrovano in tutti quei linguaggi che adottano tale metodologia.

1.1 Oggetto

Nell'ambito della programmazione con il termine oggetto (object) si individua un agente computazionale, ovvero un agente che può eseguire delle elaborazioni, caratterizzato da due elementi:

- uno stato interno
  che rappresenta l'individualità dell'oggetto stesso e che viene descritto da delle variabili
- un insieme di metodi
  che definiscono le capacità elaborative dell'oggetto, o in altre parole, le computazioni che l'oggetto può operare. Ogni metodo rappresenta una sequenza di istruzioni che l'oggetto può eseguire.

Eseguendo le istruzioni contenute nei suoi metodi un oggetto può essenzialmente:

- modificare il proprio stato interno cambiando i valori assegnati alle variabili che lo descrivono
- creare nuovi oggetti
- interagire con altri oggetti, compreso se stesso, attraverso l'invio di messaggi

L'unica modalità di interazione che esiste tra oggetti è quella elencata nel terzo punto, scambio di messaggi, per cui un oggetto non può modificare i valori delle variabili che descrivono lo stato di un altro oggetto.

Gli unici elementi visibili di un oggetto sono i suoi metodi che definiscono la sua interfaccia o protocollo. Questo

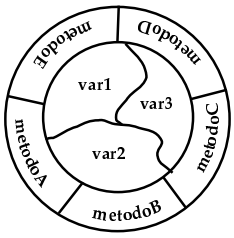

Figura 1.1: Rappresentazione di un oggetto

modello inglobando in una unica entità dati (variabili di stato) e programmi che li manipolano (metodi) permette di modularizzare e di nascondere i dettagli non significativi (information hiding).

1.2 Messaggio

Un messaggio è l'unità di informazione che viene scambiata tra due oggetti e rappresenta la richiesta di esecuzione di un particolare metodo. È composto dai seguenti elementi:

- oggetto destinatario
  è l'oggetto che deve eseguire il metodo a cui si riferisce il messaggio
- nome del metodo
  è utilizzato per identificare il metodo che deve essere eseguito
- eventuali parametri
  sono riferimenti ad altri oggetti che forniscono ulteriore informazione al metodo

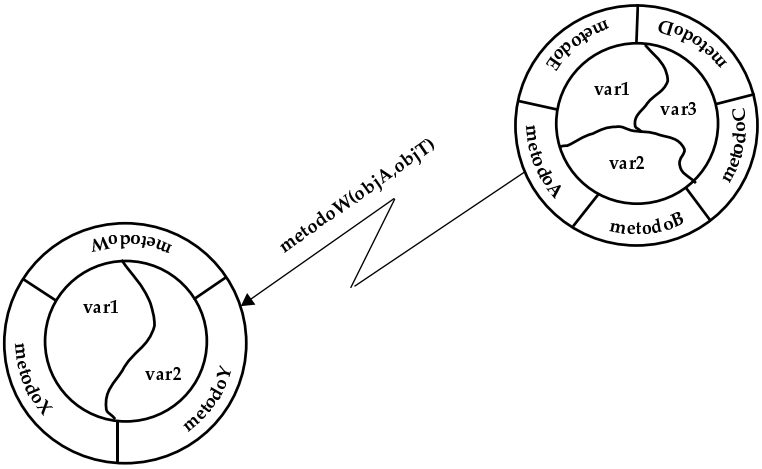

Figura 1.2: Scambio di un messaggio

1.3 Classe

Col termine classe si individua un elemento che permette di descrivere le componenti di un particolare oggetto. Una classe definisce le variabili ed i metodi di un insieme di oggetti. Tali oggetti vengono creati dinamicamente e vengono detti istanze di tale classe. In un certo senso la classe rappresenta un oggetto astratto. È possibile definire sia variabili che metodi di classe. Nel primo caso tali variabili sono condivise tra tutti gli oggetti che appartengono alla classe, per cui modifiche effettuate da un oggeto sulle variabili di classe sono viste da tutti gli altri oggetti della stessa classe.

La classe può essere vista come un contenitore per la definizione delle variabili e per i metodi associati con gli oggetti. Concettualmente quando un oggetto viene creato si cerca nella classe la definizione delle variabili per poter generare lo stato associato all'oggetto e quando egli riceve un messaggio, si cerca nella classe di cui è istanza il metodo associato al nome che ha ricevuto con il messaggio.

1.4 Ereditarietà

Definizione di una classe (subclass) in termini di altre classi (superclass). Questo processo di ereditarietà è iterativo per cui una subclass può diventare a sua volta una superclass. La classe eredita tutte le componenti (variabili e metodi) definite nelle sue superclassi, ma inoltre può:

- definire nuove variabili e/o metodi
- ridefinire variabili e/o metodi presenti nelle sue superclassi

In ogni sistema OO esiste una classe che é la madre di tutte le classi, ovvero il punto d'origine per l'ereditarietà. Tale classe non ha avviamente nessuna superclasse. In Java è la classe Object.

L'ereditarietà definisce una gerarchia di classi, che può essere strutturata sia ad albero (ereditarietà singola) sia a grafo (ereditarietà multipla). Dal punto di vista della programazione, questo concetto permette di strutturare il codice in maniera tale da facilitare il suo riuso. Infatti il codice definito ad alti livelli gerarchici viene condiviso da tutte le classi che stanno al di sotto e che lo ereditano. Le classi al livello superiore definiscono codice di uso generale mentre le classi che vengono via via definite ai livelli sottostanti lo specializzano in funzione degli oggetti che rappresentano.

Inoltre la struttura ereditaria stabilisce gli ambiti di definizione per i vari elementi del linguaggio (ad esempio variabili e metodi) e le modalità per le loro associazioni. Ad esempio, se un oggetto riceve un messaggio il cui codice non é presente nella classe dell'oggetto si ricerca nelle superclassi di tale classe, cosi via in maniera ricorsiva finché

non si trova tale codice o viene generato un errore. L'associazione tra un messaggio ed un metodo che lo implementa può essere fatta al tempo della compilazione, oppure all'atto dell'esecuzione. Nel primo caso si ottiene un programma piú efficiente, ma con la necessità di aver definito prima tutte le classi e i loro rapporti ereditari. Nel secondo, si ha la possibilità di una programmazione incrementale per cui non è necessario definire i vari elementi se non vengono utilizzati all'esecuzione; d'altra parte si ha un sistema meno effciente perché il codice da eseguire deve essere cercato durante l'esecuzione.

1.4.1 Ereditarietà singola

Quando una classe può avere solamente una superclasse e quindi ereditare attraverso una sola classe, si ha il caso di ereditarietà singola.

La relazione di ereditarietà viene rappresentata da un albero come in figura 1.3 che visualizza le relazioni per tre classi che rappresentano possibili elementi di una interfaccia utenet grafica: una finestra generica, una finestra di dialogo ed una finestra con titolo. La classe genitrice, che in questo caso corrisponde alla radice dell'albero, è la classe FinestraSemplice i cui elementi (le variabili x,y,alt,larg ed i metodi disegna, trasla) sono ereditati sia dalla classe Dialogo sia dalla classe FinestraTitolo. Nella classe Dialogo vengono definete due nuove variabili (mes, bottone) ed un nuovo metodo (premi) mentre si ridefinisce il metodo disegna per tener conto della diversa immagine grafica. IN maniera simile per la classe FinestraTitolo che ridefinisce sempre il metodo Disegna, ma che aggiunge la variabile titolo.

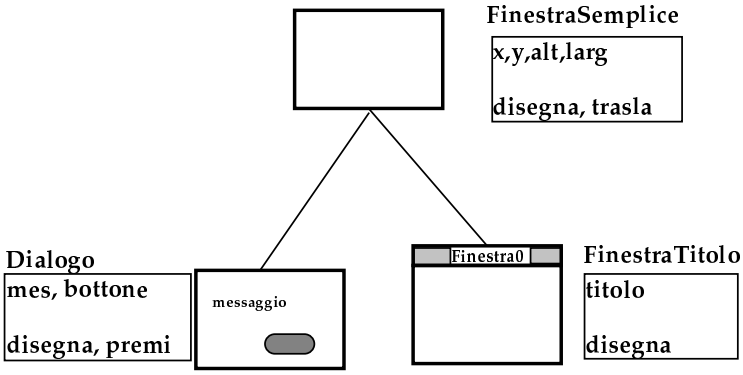

Figura 1.3: Albero di ereditarietà

1.4.2 Ereditarietà multipla

Se una classe può ereditare da piú classi si ha l'ereditarietà multipla ed in questo la relazione di discendenza tra classi è rappresentata da un grafo.

Un esempio di questo tipo è mostrato in figura 1.4. Si immagini che la classe BeatCount descriva degli oggetti che consentono di misurare il tempo e si voglia costruire degli orologi che visualizzino l'ora. Si può ottenere questo definindo delle sottoclassi che aggiungono alla classe BeatCount la rappresentazione grafica. In questo caso vi sono due classi: la classe AnaClock per gli orologi con qudrante analogico e la classe DigiClock 09:05 per gli orologi con visualizzazione digitale. Per rappresentare degli orologi che abbiano sia il quadrante analogico che quello digitale, si definisce la classe DiAnaClock che raggruppa le definizioni delle due classi di visualizzazione in un unico elemento attraverso il meccanismo dell'ereditarietà multipla. A questo punto si hanno a disposizione degli oggeti con entrambe le funzionalità.

Benché questo metodo sia concettualmente semplice, la sua realizzazione pratica presenta delle difficoltà del tipo:

- ambiguità sui nomi
  quando le superclassi hanno metodi e/o variabili con lo stesso nome, l'uso di tale nome nella sottoclasse è ambiguo perché non si sa a quale elemento associarlo

- esecuzione dei metodi
  il come ed in quale sequenza vengono eseguiti i metodi delle superclassi influenza il comportamento complessivo del programma, ad esempio, sotto certe condizioni è possibile che alcuni metodi possano essere eseguiti due volte.

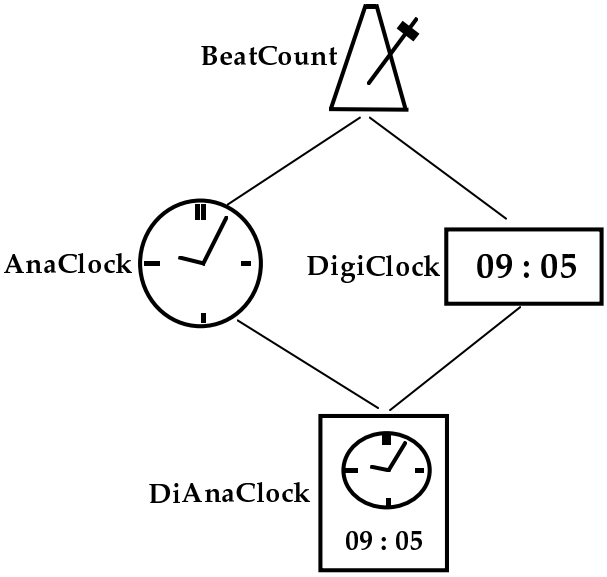

Figura 1.4: Grafo di ereditarietà

Capitolo 2

Il linguaggio Java

In questo capitolo vengono richiamati brevemente i concetti del linguaggio Java.

Il linguaggio Java non è un linguaggio object-oriented puro, ma il suo nucleo base, costituito dai tipi base e dalle istruzioni, rispecchia il modello computazionale dei linguaggi procedurali. È a livello delle classi che si utilizza pienamente la programmazione orientata agli oggetti. Questa dicotomia tra i due livelli che comporta l'uso di due paradigmi differenti complica la programmazione, come si può vedere dal fatto che vengono definite delle classi utilizzate per rappresentare i tipi base a livello di classe. Dato che si ha sia il tipo base `int` che la classe `Int` per rappresentare gli interi, quale si usa?

2.1 Commento

In java è possibile scrivere i commenti al codice in tre forme:

// commento	da // fino a fine linea
/* commento */	tutti i caratteri tra /* */ inclusi
/** commento */	commento di documentazione prima dichiarazione di classi od elementi della classe. È utilizzato da tool di generazione automatica della documentazione (javadoc)

2.2 Dichiarazione di variabile

In Java le variabili vengono dichiarate con la seguente notazione:

```
<tipo> <listadiidentificatori>;
```

Il tipo può essere seguito da una o più coppie di parentesi quadre aperte e chiuse `[]` per indicare che la variabile è un array. Il numero delle coppie di parentesi definisce la dimensione dell'array. Per ogni dimensione l'indice che ne individua le componenti va da 0 a N-1 dove N è il numero di tali componenti.

Il tipo è un tipo base oppure una classe. Il valore che può assumere una variabile dipende dal tipo. Nel caso dei tipi base è un valore definito per il tipo, mentre nel caso di una classe è un riferimento ad un oggetto appartenente a tale classe o sottoclasse, oppure il valore **null** per indicare che la variabile non si riferisce a nessun oggetto.

Per accedere al valore memorizzato in una variabile si usa il nome della variabile, oppure, nel caso di un array, il nome può essere seguito da elementi denotati con `[i]` dove `i` è una espressione di valore intero. In tal caso ogni elemento individua la componente i-esima della dimensione k-esima, con k il numero d'ordine dell'elemento `[i]`.

La dichiarazione di variabile può essere preceduta da parole chiavi che definiscono alcune caratteristiche quali lo scope. Una variabile può essere inizializzata facendo seguire il nome da un = ed una espressione la cui valutazione da il valore di inizializzazione.Nel caso di un array i valori dei singoli elementi sono separati da virgola e racchiusi tra parentesi grafe

2.3 Tipi

Un tipo è caratterizzato da tre elementi:

- insieme dei valori rappresentati dal tipo
- operazioni eseguibili su tali valori

- costanti rappresentate in qualche formalismo sintattico

In Java i tipi possono essere divisi in due categorie:

tipi base sono i tipi definiti dal linguaggio stesso di cui fanno parte integrante. L'insieme dei valori, le operazioni e le costanti sono prestabiliti e non sono modificabili.

classi l'insieme dei valori e le operazioni permesse sono definite attraverso opportune notazioni sintattiche. Inoltre viene esteso secondo i meccanismi dell'eriditarietà, nel senso che il tipo associato ad una classe comprende anche le superclassi e le interfacce che la classe implementa con le loro superinterfacce. Non esiste la possibilità di definire costanti se non in qualche caso isolato (ad esempio la classe **String**)

2.4 Tipi base

byte interi 8-bit complemento a due

short interi 16-bit complemento a due

int interi 32 bit complemento a due

long interi 64 bit complemento a due

float reali 32 bit IEEE 754-1985

double reali 64 bit IEEE 754-1985

char caratteri 16 bit in codice Unicode

boolean costanti **true** o **false**

2.5 Costanti

In java si possono esprimere i seguenti valori costanti:

valori boolean
le costanti **true** e **false**

valori interi
stringhe di cifre ottali, decimali o esadecimali. I simboli iniziali dichiarano la base del numero: se inizia per 0 la base è ottale, se inizia per 0x o 0X la base è esadecimale, in ogni altro caso la base è decimale. Se la costante è seguita dalla lettera l o L è di tipo long altrimenti di tipo int.

valori reali
le costanti reali sono espresse come stringhe di cifre decimali con un punto decimale seguite da un esponente opzionale preceduto dalla lettera e o E. /possono essere seguite dalla lettera f o F per indicare il tipo float, oppure dalla lettera d o D per il tipo double. Se non vi è l'indicazione esplicita del tipo viene assunto il tipo double.

caratteri
caratteri costanti sono espressi con un simbolo tra apici come 'A'. alcuni caratteri possono essere rappresentati mediante una sequenza (valori tra parentesi corrspondono alla codifica Unicode):

`\n`	newline (`\u000A`)
`\t`	tab (`\u0009`)
`\b`	backspace (`\u0008`)
`\r`	return (`\u000D`)
`\f`	form feed (`\u000C`)
`\\`	backslash (`\u005C`)
`\'`	apice (`\u0027`)
`\"`	doppio apice (`\u0022`)
`\`*ddd*	rappresentazione ottale del carattere

riferimento ad oggetti
la costante **null** rappresenta un riferimento all'oggetto che non esiste, ovvero non è un riferimento valido.

stringhe
i caratteri racchiusi tra doppi apici (") sono le costanti del tipo **String**, che è una classe.

2.6 Classi

Una classe è composta da un insieme di variabili e da un insieme di metodi che rappresentano il codice per elaborare l'informazione codificata nelle variabili.
Una classe può estendere un'altra classe, la sua superclasse, e/o implementare delle interfacce.
La sintassi:

```
[public | abstract | final] class <id> [extends <classid>]
   [implements <interfaceid list>]
     {

  <dichiarazione di variabili>
  <dichiarazione di metodi>
}
```

dove:

public classe accessibile dall'esterno

abstract la classe non può essere istanziata, può essere solo utilizzata come superclasse. Definisce degli elementi (variabili o metodi) abstract che devono essere realizzati dalle sottoclassi.Definizione parziale dell'implementazione.

final la classe non può essere utilizzata come supeclasse, ovvero non è possibile classi derivate da una classe definita final.

extends clausola opzionale che definisce la superclasse da cui la classe eredita. Se non presente, per default si eredita dalla classe Object.

implements clausola che definisce le interfacce che la classe implementa.

2.6.1 Opzioni per variabili e metodi

Nelle dichiarazioni di variabili e metodi possono essere presenti delle opzioni espresse attraverso la presenza di parole chiave che modificano le caratteristiche degli elementi.
Controllo dello scope ed ereditarietà.
Le keyword sono in alternativa.

public gli elementi sono accessibili ovunque la classe è accessibile e possono essere ereditati dalle sottoclassi

private gli elementi sono accessibili solo all'interno della classe

protected gli elementi sono accessibili solamente nella classe e sue sottoclassi

final gli elementi non possono più essere ridefiniti.

nel caso non ci sia nessuna di queste parole chiave gli elementi sono accessibili da sottoclassi nello stesso package

Elementi di classe.

static elementi dichiarati con questa opzione appartengono alla classe. In altri termini nel caso di variabili sono variabili di classe ovvero è unica indipendemente da qunati sono gli oggetti istanziati. Nel caso di metodi sono metodi di classe e possono essere attivati utilizzando come riferimento l'identificatore di classe. Inoltre non è possibile usare le variabili `this` e `super` nel loro corpo.

Programmazione concorrente

synchronized definisce metodi di una classe che non possono essere eseguiti in parallelo. Le varie attivazioni vengono serializzate.

volatile definisce una variabile che può essere modificata in maniera asincrona rispetto all'esecuzione di un thread

Metodi nativi

native definisce un metodo scritto in un altro linguaggio (ad esempio C).

2.6.2 Variabili

Si può accedere alle variabili dichiarate public con la notazione

```
oggetto.variabile
```

mentre se sono static

```
classe.variabile
```

Si possono esprimere valori costanti definendo variabili public static final con valori di inizializzazione.

2.6.3 Metodi

Sintassi:

```
<tipo> <metodoId> ( < lista parametri> )
           throw <exceptionList>
     { <body> }
```

dove

`<tipo>` è il tipo del valore di ritorno del metodo. Può essere un tipo base(ritorna un valore corrispondente), una classe (ritorna un riferimento ad un oggetto appartenente a tale classe o sottoclasse) oppure **void** (non ritorna nessun valore)

`<metodoId>` identificatore del metodo

`<lista parametri>` lista di parametri formali separati da virgola. Ogni parametro è formato da una coppia tipo variabile. I parametri sono passati per valore.

`<exceptionList>` lista di eccezioni, separate da virgola, che possono essere generate durante l'esecuzione del metodo

`<body>` corpo del metodo formato da dichiarazione di variabili locali e istruzioni

Ogni metodo è univocamente individuato dal suo nome e dal numero e tipo dei parametri. Quindi si possono avere due metodi con lo stesso nome (overloading).

Attivazione di un metodo:

```
<riferimento>.<metodoId>(<lista parametri attuali>)
```

dove

`<riferimento>` definisce l'oggetto a cui viene applicato il metodo. Può essere una variabile, un identificatore di classe, o un'espressione che ritorna un riferimento ad un oggetto.

`<metodoId>` è l'identificatore del metodo

`<lista parametri attuali>` lista di espressioni, separate da virgola, i cui valori sono assegnati ai corrispondenti parametri formali

All'interno del corpo di un metodo è possibile utilizzare due variabili predefinite:

this contiene il riferimento all'oggetto su cui il metodo è stato invocato

super contiene il riferimento all'oggetto su cui il metodo è stato invocato come istanza della sua suprclasse. In altri termini attraverso la variabile super è possibile accedere ai metodi della superclasse.

2.6.4 Creazione di oggetti

Gli oggetti vengono creati con la seguente notazione:

```
new <tipo>
```

dove

`<tipo>` è un tipo base od una classe. Il tipo può essere seguito da un valore numerico racchiuso tra parentesi quadre che può a sua volta essere seguito dalla medesima struttura. Nel primo caso viene creato un array con un numero di elementi del dato tipo pari al valore numerico, mentre nel secondo un array of arrays.

All'interno di una classe possono essere definiti dei metodi (constructor) che vengono attivati in maniera implicita alla creazione di un oggetto. Il loro identificatore è uguale a quello della classe e possono avere dei parametri. Sono utilizzati per funzioni di inizializzazione. L'uso della variabile `this` al loro interno nel formato `this(<eventuali parametri>)` attiva il corrispondente costruttore della medesima classe, mentre l'uso della variabile `super` nella forma `super(<eventuali parametri>)` attiva il corrispondente costruttore della superclasse. Nel caso in cui non vi sia il costruttore i valori delle variabili sono inizializzati ai valori definiti nelle dichiarazioni o a valori di default.

2.6.5 Interfacce

Dato che il linguaggio java ha solamente ereditarietà singola l'uso delle interfacce permette di ovviare a tale problema introducendo una metodologia che presenta caratteristiche simili all'ereditarietà multipla, ma che comunque non la equivale.

La sintassi per la definizione di una interfaccia è simile quella usata per definire una classe:

```
interface <id> [extends <interfaceid list>]
     {

  <definizione di variabili>
  <definizione di metodi>
}
```

ma in questo caso non non viene definito il corpo dei metodi, dei quali si dichiara solamente l'intestazione. Inoltre le variabili devono avere valori di inizializzazione e sono considerate delle costanti il cui valore non può essere modificato. La dichiarazione completa del metodo con il relativo corpo viene demandata alla classe che implementa una interfaccia. Un esempio dell'uso delle interfacce è proposto nell'esempio Ereditarietà multipla.

2.6.6 Classi interne (inner classes)

È possibile definire delle classi in ogni scope, ovvero come membri di altre classi, localmente all'interno di un blocco di istruzioni o in maniera anonima come elemento di una espressione. Il loro uso è utile in contesti tipo l'Abstract Window Kit (AWK) per associare elementi dell'interfaccia grafica con azioni da eseguire alla loro attivazione.

2.6.7 Packages

Le classi in java possono essere raggruppate in packages che permettono di delimitare l'ambito di validità degli identificatori utilizzati in tali classi.
Si usa il costrutto

```
package <nome\_qualificato> ;
```

posto all'inizio del file sorgente per assegnare le classi ivi definite a tale package. Il `<nome_qualificato>` del package è formato da uno o piú identificatori separati da . (punto) come descritto da:

```
<nome\_qualificato> ::= <identificatore> [.<identificatore>]
```

e definisce il percorso della directory in cui risiede il codice compilato di tale package. In altre parole sostituendo al punto il simbolo di separazione per le sottodirectory (ad esempio il carattere
per Unix) si ottiene il nome della directory, riferita alla directory base utilizzata dalla macchina virtuale, dove deve risiedere il codice compilato.

Per poter utilizzare gli elementi di un package si deve utilizzare il costrutto

```
import <nome\_package>.* ;
```

se esistessero conflitti di nomi con altri packages si può utilizzare il nome del package come qualificatore dei nomi con il formato

`<nome\_package>.<nome\_elemento>`

2.7 Conversioni di tipo

Java è un linguaggio fortemente tipizzato.Questo significa che il compilatore verifica sempre la compatibilità dei tipi. Vi possono essere conversioni implicite oppure esplicite.

2.7.1 Implicite

Java esegue in maniera implicita le seguenti conversioni per i tipi base:

1. char → int
2. tipi interi → float
3. float → double

Nel caso di riferimenti ad oggetti, si può usare il riferimento ad un oggetto di una certa classe ovunque sia richiesto un riferimento ad una sua superclasse.
Infine, vi è un conversione implicita ad oggetti di tipo String. I tipi base vengono convertiti a stringhe che rappresentano il loro valore, mentre per gli oggetti si usa il metodo toString.

2.7.2 Esplicita

Attraverso l'espressione

`(<tipo>)<espressione>`

è possibile convertire in maniera esplicita un valore di un certo tipo in un altro tipo.
Nel caso dei tipi base è possibile convertire tipi con maggior precisione in tipi con minor precisione, quindi ,ad esempio, long in int, double in float, tipi reali in tipi interi. Non è possibile convertire alcun tipo nel tipo boolean.
Nel caso di oggetti, è possibile convertire un riferimento da una classe ad un altra che sia sottoclasse della prima solo se l'oggetto individuato dal riferimento è un istanza della seconda classe. Se questo non fosse vero verrebbe generata una eccezione `ClassCastException`. È possibile verificare l'appartenenza di un oggetto ad una certa classe con l'operatore `instanceof`:

`<oggetto> instanceof <tipo>`

che ritorna `true` se l'oggetto appartiene al tipo, `false` altrimenti.

Listato 2.1 - TypeCast.java: conversione di tipi esplicita.

```
 class Data{
}

class DataA extends Data {
}

class DataB extends Data {
}

public class TypeCast{

public static void main(){
char c='A';
byte b=13;
short s=1024;
int i=0x0fffffff;
long l=0x0fffffffffL;
```

```
float f=1.5f;
double d=3.5567890754334D;

// conversioni implicite da char
i=c; l=c; f=c; d=c;

// conversioni implicite da byte

s=b; i=b; l=b; f=b; d=b;

// conversioni implicite da int
l=i; f=i; d=i;

// conversioni implicite da long
f=l; d=l;

// conversioni implicite da float
d=f;

// non esiste nessuna conversione implicita da double ad altri tipi

// tutte le rimanenti conversioni devono essere fatte in maniera esplicita

// da char
b=(byte)c; s=(short)c;

// da byte
c=(char)b;

// da int
c=(char)i; b=(byte)i;

// da long
b=(byte)l; c=(char)l; i=(int)l;

// da float
c=(char)f; b=(byte)f; s=(short)f; i=(int)f; l=(long)f;

// da double
c=(char)d; b=(byte)d; s=(short)d; i=(int)d; l=(long)d;

Data data;
DataA dataA = new DataA();
DataB dataB = new DataB();

data = dataA; // conversione implicita da sottoclasse
dataA = (DataA)data; // conversione esplicita da superclasse
dataB = (DataB)data; // conversione esplicita da superclasse. durante l'esecuzione
                     // viene segnalata l'eccezione ClassCastException

/* le seguenti conversioni non sono permesse, generano errore in compilazione

dataA = (DataA)dataB;
dataB = (DataB)dataA;
*/

}
}
```

2.8 Espressioni ed operatori

2.8.1 Precedenza degli operatori

postfissi	`[] . (`*parametri*`)` *espr*`++` *espr*`--`
unari	`++`*espr* `--`*espr* `+`*espr* `-`*espr* `~ !`
creazione/conv. tipo	`new (`*tipo*`)`*espr*
moltiplicativi	`* / %`
additivi	`+ -`
shift	`<< >> >>>`
relazionali	`< > <= >= instanceof`
uguaglianza	`== !=`
AND bit	`&`
XOR bit	`^`
OR bit	`|`
AND logico	`&&`
OR logico	`||`
condizionale	`?:`
assegnazione	`= += -= *= /= %= >>= <<= >>>= & = ^= |=`

2.8.2 Associatività

Tutti gli operatori binari tranne che quelli di assegnazione sono associativi a sinistra (left-associative). Quelli di assegnazione son associativi a destra (right-associative).

2.8.3 Ordine di valutazione

Il linguaggio Java garantisce che gli operandi sono valutati da sinistra a destra.

2.8.4 Operatori aritmetici

`+`	addizione(binario), segno positivo(unario)
`-`	sottrazione(binario), negazione(unario)
`*`	moltiplicazione
`/`	divisione
`%`	resto
`++`	autoincremento prefisso o postfisso
`--`	autodecremento prefisso o postfisso

2.8.5 Concatenazione di stringhe

`+`	concatenazione di due stringhe

2.8.6 Operatori di confronto e logici

`>`	maggiore
`<`	minore
`>=`	maggiore uguale
`<=`	minore uguale
`==`	uguale
`!=`	diverso
`&&`	and logico
`||`	or logico
`!`	not logico

Nell'and e or logico la valutazione del secondo operando è condizionale al valore nel primo, ovvero viene valutato solamente se necessario.

2.8.7 Operatori sui bit

`&`	and bit a bit
`|`	or bit a bit
`~`	complemento a uno
`<<`	shift a sinistra introducendo zeri
`>>`	shift a destra aggiungendo bit di segno
`>>>`	shift a destra introducendo zeri

Gli operatori and, or e complemento poossono essere applicati solamente a tipi interi o booleani, quelli di shift solo a tipi interi.

2.8.8 Operatori di assegnazione

In java l'assegnazione è una espressione che ritorna come valore il valore assegnato.
Dato un operatore *op*=, si ha che l'espressione

var *op*= expr

equivale a

var = var *op* (expr)

eccetto che var è valutata una sola volta.

2.9 Eccezioni

Una eccezione è generata quando avviene un errore durante l'esecuzione. Le eccezioni sono oggetti definiti da classi che estendono la classe Throwable.

Eccezioni sono di solito "checked exceptions", che significa che il compilatore verifica che i metodi generino solamente le eccezioni che dichiarano. Errori standard di esecuzione estendono le classi RuntimeException e Error, che sono "unchecked exceptions". Per convezione, nuove eccezioni devono essere definite come estensione della classe Exception, rendendole di tipo "checked".

Esistono istruzioni per generare eccezioni(`throw`) ed istruzioni per gestirle (`try-catch-finally`).

2.10 Istruzioni

espression statements

ogni espressione che sia di uno dei tipi seguenti:

- espressione di assegnazione
- espressioni con gli operatori `++` e `--`
- attivazione di metodi
- creazioni di oggetti (operatore new)

può diventare un istruzione se seguita da punto e virgola (`;`)

blocco

zero o piú istruzioni racchiuse da parentesi grafe (`{ }`) compongono un blocco

if (*boolean expr***)** *istr1* **else** *istr2*

se il valore dell'espressione booleana è true viene eseguita l'istruzione istr1 altrimenti istr2. La clausola else è opzionale.

switch

valuta un espressione intera il cui valore è utilizzato per scegliere una appropriata clausola `case`.

```
switch(int-expr){
        case costante-int : istr-list
        case costante-int : istr-list
        .......
        default:   istr-list
        }
```

Se esiste un `case` il cui valore costante corrisponde alla valutazione dell'espressione, si continua l'esecuzione con la prima istruzione che segue il `case`. Se non vi è, si salta all'istruzione contrassegnata da `default` se presente, altrimenti si salta l'intera istruzione `switch`.

while (*boolean-expr***)** *istr*

Finché la'espressione booleana è vera si esegue l'istruzione istr. L'istruzione non viene eseguita se l'espressione booleana è inizialmente falsa.

do *istr* **while (***bool-expr***);**

Si continua ad eseguire l'istruzione *istr* fino a quando l'espressione booleana diventa falsa. L'istruzione *istr* viene eseguita almeno una volta.

for (*init-expr-list***;***boolean-expr***;***inc-expr-list***)** *istr*

equivalente al seguente segmento di programma:

init-expr-list;
while (*boolean-expr*){
    istr;
    inc-expr-list;
    }

dove *inc-expr-list* è una lista di espressioni, separate da virgola, che vengono valutate sequenzialmente.

break

può essere usato per uscire da un blocco qualunque. Nella maggior parte dei casi è utilizzato per terminare un ciclo. Può essere seguito da una label, ovvero da un identificatore, che permette di individuare una particolare istruzione con la notazione *label*:*istruzione*. Nel caso senza label il break termina il blocco più interno, mentre nel formato con la label termina il blocco contrassegnato da tale label.

continue

salta alla fine del corpo di un ciclo facendo partire una successiva iterazione con la valutazione dell'espressione booleana che controlla il ciclo. Nel caso non sia seguito da una label, si applica al ciclo più interno, altrimenti a quello contrassegnato dalla label.

return *expr*

termina l'esecuzione di un metodo assegnando come valore di ritorno il valore dell'espressione *expr*. Se il metodo non ritorna nessun valore (tipo di ritorno `void`) non ci deve essere alcuna espressione.

throw *exception*

genera una eccezione rappresentata dall'oggetto *exception*.

try-catch-finally

utilizzato per gestire le eccezioni.

try
    blocco
catch (*tipo-eccezione identificatore*)
    blocco
catch (*tipo-eccezione identificatore*)
    blocco
    ....
finally
    blocco

Il blocco del `try` viene eseguito finché o avviene una eccezione oppure il blocco termina con successo. Nel caso in cui sia generata una eccezione, viene bloccata l'esecuzione del blocco del `try` e si cerca una clausola `catch` che abbia come *tipo-eccezione* la classe o una superclasse dell'eccezione generata. Se esiste tale clausola, si associa al corrispondente *identificatore* l'eccezione e si esegue il blocco associato. Se non viene trovato una clausola `catch` appropriata, l'eccezione viene passata aa un eventuale istruzione `try` che includa l'istruzione `try` corrente. Se non esiste alcuna clausola `catch` che possa gestire l'eccezione, viene passata al codice che aveva invocato il metodo. Il codice della clausola `finally` viene sempre eseguito al termine dell'esecuzione

del `try` sia nel caso di successo sia nel caso di eccezione. Quest'ultimo codice può modificare il valore di ritorno dell'istruzione `try` con l'utilizzo di una istruzione `return` o generando una eccezione che sostituisce l'eventuale eccezione generata durante l'esecuzione del blocco del `try`.

Capitolo 3

Programmi procedurali.

Nei programmi presentati in questo capitolo si è utilizzato uno stile di programmazione procedurale. Questo significa che non sono state utilizzate le caratteristiche object oriented presenti in Java. Ovviamente si ritroverà la definizione di qualche classe, ma tali classi vengono utilizzate solamente come contenitori per i metodi, per cui ad esempio non vengono mai instanziate. Tale stile è simile quello utilizzato per programmare in Pascal o C. Rispetto a questi linguaggi il vantaggio di usare Java sta nella possibilità di sfruttare il polimorfismo per applicare il medesimo nome di metodo a dati di tipo differente, come verrà descritto nel programma Fattoriale polimorfo ”overloaded” a pagina 26.

3.1 Programmi banali

In questa sezione vengono mostrati due programmi alquanto banali. Il primo non fa assolutamente niente, mentre il secondo stampa un messaggio a terminale. Anche se sembrano assolutamente inutili, vengono spesso usati come primi programmi nei libri di programmazione. In specie la seconda versione è presente in tutti i libri in varianti praticamente uguali, cambia solamente il messaggio, ed ovviamante questo libercolo non fa eccezione. Un altro loro possibile uso è come programmi di prova per un ambiente di programmazione, essenzialmente per vedere se si è capaci di compilare ed eseguire un semplice programma.

Listato 3.1 - Niente.java - programma minimo in Java.

```
/**
*
*  programma minimo in Java
*
*  @author P.Bison Copyright 1997
*
*/
public class Niente {

/**
 * programma principale. Programma che non fa niente.
 */
  public static void main(String[] args) {
    }
  }
```

Il minimo programma che si possa scrivere in Java è mostrato nel listato 3.1. Da questo semplice programma si possono vedere i due elementi minimi che devono esserci in una applicazione Java: la definizione di una classe e la dichiarazione del metodo `main` che è il punto di attivazione dell’intero programma. Il sistema a cui sarà dato il compito di attivare questa applicazione cederà il controllo al metodo `main` che inizierà l’esecuzione. È obbligatorio che la definizione del metodo `main` segua la struttura mostrata nel listato, altrimenti viene segnalato un errore quando si inizia l’esecuzione. Il parametro args è un array di stringhe che contiene gli argomenti del comando che è stato usato per attivare l’applicazione.

L’esecuzione di questo programma ovviamente non darà alcun segno visibile della propria esecuzione. Se si vuole avere qualche indicazione di una avvenuta esecuzione si può modificare il programma aggiungendo una istruzione che stampa un messaggio sul terminale, come mostrato nel listato 3.2 .

Listato 3.2 - SciavoVostro.java - semplice stampa.

```
/**
*
*  programma minimo in Java che stampa <b> Sciavo vostro! </b>
*
*  @author P.Bison Copyright 1987
*
*/
public class SciavoVostro {

/**
 * programma principale
 */
  public static void main(String[] args) {
    System.out.println("Sciavo vostro!"); //stampa il messaggio.
    }
  }
```

In questo caso l'istruzione `System.out.println("Sciavo vostro!");` stampa il valore del parametro, in questo caso la stringa `"Sciavo vostro!"`, seguito da un a capo sul terminale. Si interpreta tale istruzione come l'attivazione del metodo println dell'oggetto out della classe System con il parametro `"Sciavo vostro!"`.

Per inciso, la frase *Sciavo vostro* era un saluto utilizzato nella Serenissima Repubblica di Venezia, da cui per elisione è derivato l'attuale saluto informale *Ciao*.

3.2 Stampa degli argomenti.

Il programma mostrato nel listato 3.3 stampa a terminale gli eventuali argomenti passati all'applicazione dal comando di esecuzione.

Listato 3.3 - Eco.java - stampa degli argomenti.

```
/**
 *
 *  stampa degli argomenti a terminale. Stampa degli argomenti passati al programma
 * dal commando di esecuzione, uno per linea.
 *
 *  @author P.Bison Copyright 1987
 *
 */
public class Eco {


/**
 * programma principale
 */
  public static void main(String[] args) {

   for(int i=0; i < args.length; i++) // itera sul numero di argomenti
    System.out.println(args[i]); //stampa l'i-esimo argomento.
    }
  }
```

Un ciclo `for` è utilizzato per iterare sul numero degli argomenti passati al programma. La variabile `i` è l'indice utilizzato per scorrere i singoli argomenti e varia da 0 a n-1, dove n è il numero di argomenti passati al programma nell'esecuzione corrente. Tale numero viene calcolato attraverso la notazione `args.length` che ritorna il numero di elementi presenti nell'array `args`. Tale notazione si può applicare a qualunque array per valutare il numero dei suoi componenti. All'usuale istruzione di stampa viene passato come parametro l'i-esimo argomento che viene quindi stampato a terminale seguito da un a capo. Si noti che gli argomenti sono stringhe per cui se si volesse passare dei valori numerici si devono usare appositi metodi, tipo parseInt della classe Integer, per convertire l'argomento nel corrispondente valore numerico.

3.3 Programmi sulle stringhe

Nel primo esempio, mostrato nel listato 3.4, si verifica in maniera iterativa se una stringa è palindroma, ovvero se risulta uguale sia leggendola da sinistra che da destra. Utilizzando un ciclo `for` si scorre la stringa sia dall'inizio che dalla fine utilizzando i due indici `i` e `j`. Il valore ottenuto confrontando i due caratteri indicati da tali indici viene memorizzato nel flag `isPalinFlag` che alla fine indica se la stringa è o non è palindroma .

Listato 3.4 - Palin.java - verifica se una stringa è palindroma.

```
/**
 *
 *  verifica se una stringa e' palindroma
 *
 *  @author P.Bison Copyright 1987
 *
 */
public class Palin {

public static boolean isPalin(String st) {
 boolean isPalinFlag;
 int i,j;

 i=0; j=st.length()-1;
 isPalinFlag = true;
 while((i<=j) && isPalinFlag)
   isPalinFlag = st.charAt(i++)==st.charAt(j--);
 return isPalinFlag;
}

/**
 * programma principale
 */
  public static void main(String[] args) {

   for(int i=0; i < args.length; i++){ // itera sul numero di argomenti
    System.out.print(args[i]);
    if (isPalin(args[i])) System.out.println(" e'palindroma");
    else System.out.println(" non e'palindroma");
    }
  }
}
```

Nell'esempio riportato nel listato 3.5 si calcola, sempre in maniera iterativa, il rovescio di una stringa, ovvero la stringa ottenuta leggendo la stringa di partenza a rovescio. Utilizzando i metodi `concat` e `substring` definiti per la classe `string`, attraverso un ciclo `for` che scorre tutta la stringa partendo dall'ultimo carattere, si concatenano i singoli caratteri dall'ultimo al primo.

Listato 3.5 - Reverse.java - capovolge una stringa.

```
/**
 *
 *  esegue il reverse di una stringa
 *
 *  @author P.Bison Copyright 1987
 *
 */
public class Reverse {

public static String doReverse(String st) {
 int i,j;
 String t = new String();

 for(i=st.length()-1; i>=0;i--)
  t=t.concat(st.substring(i,i+1));

 return t;
}
```

```
/**
 * programma principale
 */
  public static void main(String[] args) {

   for(int i=0; i < args.length; i++){ // itera sul numero di argomenti
    System.out.print(args[i] + " ");
    System.out.println(doReverse(args[i]));
    }
  }
}
```

3.4 Calcolo del fattoriale

Il calcolo del fattoriale è un esempio molto comune nel campo della programmazione. In questo caso nel listato 3.6 viene mostrata la sua realizzazione nel linguaggio java utilizzando un algoritmo iterativo basato sul fatto che il fattoriale di n è uguale al prodotto di tutti i numeri da 1 a n:

$$n! = n(n-1)(n-2)\cdots 1$$

Quindi con un semplice ciclo `while` si puó iterare su tutti in numeri tra n e 1 memorizzando ad ogni passo il prodotto parziale in una variabile che alla fine conterrá il valore cercato. Si noti inoltre i limiti di questo programma riportati nel listato stesso.

Listato 3.6 - Factorial.java - fattoriale iterativo.

```
/**
 *  calcolo del fattoriale con algoritmo iterativo.
 *
 * @author P.Bison Copyright 1997
 */
public class Factorial{

/**
 * fattoriale iterativo con risultato e parametro di tipo int.
 * calcola il fattoriale del numero passato come parametro.
 * utilizza il tipo int, percui si ha un errato valore di ritorno
 * quando si supera il valore massimo rappresentabile
 * ad esempio il fact(15) risulta 2004310016 mentre il risultato
 * corretto e' 1307674368000
 * risultato e' corretto fino a fact(12)
 * @param n valore di cui si deve calcolare il fattoriale
 * @return il fattoriale di n
 */
public static int fact(int n) {
  int ris;

  ris=1;
  while(n!=0)
    ris = ris * n--;
  return ris;
  }


public static void main(String[] args) {
  int n=5;

  if (args.length == 1)
     n= Integer.parseInt(args[0]);


  System.out.println("fact("+n+") = "+fact(n));
  }

} // end of class Factorial
```

3.5 Massimo Comun Divisore di Euclide

Euclide, un matematico greco vissuto nel IV secolo A.C., nella sua opera *Gli Elementi* descrive un algoritmo per il calcolo del massimo comun divisore (MCD) di due numeri. Dati due numeri - m e n - il massimo comun divisore di m e n, denotato con $gcd(m, n)$, è il piú grande numero che divide sia m che n senza lasciare alcun resto, ad esempio $gcd(12, 40) = 4$. L'algoritmo di Euclide, come è ora conosciuto, consiste nei seguenti passi:

1. se m è minore di n allora scambia m con n
2. poni m uguale al resto ottenuto da m diviso n
3. se m non è uguale a 0, ricomincia dal punto 1, con i nuovi valori di m e n
4. il valore di n è il MCD

Listato 3.7 - MCD.java - Massimo Comun Divisore.

```
/**
 *
 * Algoritmo iterativo di Euclide per il calcolo del Massimo Comun Divisore.
 *
 *  @author P.Bison Copyright 1997
 *
 */
public class  MCD{

/**
 * calcolo del massimo comun divisore con il metodo di Euclide
 */
public static int calcolaMCD(int n, int m)
{
  int t;

  while(m!=0) {
    if (m<n) {t=m; m=n; n=t;} // scambia m con n
    m = m % n;
    }
  return n;
}



/**
 * programma principale
 */
  public static void main(String[] args) {
   int m=24,n=12,mcd;

   if (args.length == 2) {
     m= Integer.parseInt(args[0]);
     n= Integer.parseInt(args[1]);
     }

   mcd=calcolaMCD(m,n);

   System.out.println(mcd);
  }
}
```

3.6 Crivello di Erastotene.

Erastotene (276-196 A.C.) inventò un metodo efficiente per costruire tabelle di numeri primi. Tale metodo è chiamato ”Crivello di Erastotene“ e può essere descritto nella maniera seguente.

Si scriva una lista di interi che cominci per 2 e che termini con un qualche numero, per esempio 23. Il metodo permette di trovare tutti i numeri primi compresi tra questi due valori.

```
 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23
```

Partendo dal primo numero, in questo caso il valore 2, si marchi tutti i multipli di tale numero:

```
 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23
       x     x     x     x     x     x     x     x     x     x
```

Si consideri il successivo numero non marcato, in questo caso 3, e si marci tutti i suoi multipli:

```
 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23
       x     x     x  x  x     x     x  x  x     x     x  x  x
```

Si continui i questa maniera, marcando tutti i multipli dei successivi numeri non marcati finché non rimane alcun nuovo numero non marcato. I numeri che non risultano marcati sono numeri primi. Nell’esempio la tabella finale corrisponde all’ultima tabella data per cui i numeri primi sono:

```
 2  3  5  7  11  13  17  19  23
```

Il programma java che realizza questo algoritmo è mostrato nel listato 3.6.8.

Listato 3.8 - Sieve.java - il crivello di Erastotene.

```
/**
 *
 * Crivello di Erastotene. Calcola i numeri primi con il metodo
 * del crivello di Erastotene. L’argomento, se presente, da
 * il valore massimo entro cui calcolare i numeri primi. Se non
 * e’ presente si usa il valore 23
 *
 *  @author P.Bison Copyright 1997
 *
 */
public class  Sieve{

/**
 * calcolo dei numeri primi con il crivello di Erastotene
 */
public static void sieveMethod(boolean[] flags)
{
   int i,multiplo;

      for(i=2; i<flags.length; i++) // inizializza l’array
         flags[i]=true;
      for(i=2; i<flags.length; i++) { // itera su tutti i numeri
         if (flags[i]) // se non e’ stato marcato
               // marca tutti i suoi multipli
               for(multiplo=i+i; multiplo<flags.length; multiplo+=i)
                 flags[multiplo]=false;
         }

}



/**
 * programma principale
 */
  public static void main(String[] args) {
```

```
   int size=23;
   boolean[] flags;

   if (args.length > 0) size= Integer.parseInt(args[0]);
   flags = new boolean[size+1];  // array indicizzato da 0 a size

   sieveMethod(flags);

   for(int i=2; i <= size; i++)  // itera sul numero di argomenti
    if (flags[i]) {
      System.out.print(i); //stampa numero primo.
      System.out.print(" ");
    }
   System.out.println();
  }
}
```

Oltre al programma principale `main`, nella classe `Sieve` è dichiarato il metodo `SieveMethod` che implementa l'algoritmo di Erastotene.

Il programma principale dichiara due variabili: `size` che contiene il valore massimo che si deve considerare per la ricerca dei numeri primi e `flags`, un array di boolean, che associa ad ogni numero, rappresentato dal corrispondente indice, il fatto di essere primo o no: un valore `true` nella i-esima posizione indica che il numero i è un numero primo.

All'inizio se vi sono argomenti passati all'applicazione si valuta il primo argomento come intero e lo si assegna alla variabile `size`. Se non ci fosse alcun argomento si usa il valore 23, che è il valore definito nella dichiarazione della variabile stessa. Poi si alloca un array di dimensione pari a `size+1`. Questo perché la prima posizione di un array è indicizzata dal valore 0, quindi se si vuole che l'indice corrisponda al numero e che il massimo numero sia il valore di `size` si deve allocare un numero di elementi pari al numero massino da considerare piú uno. Si noti inoltre che vengono effettivamente utilizzate le posizioni dall'indice 2 al valore definito da `size`.

Dopo aver attivato il metodo `sieveMethod` passandogli come parametro l'array `flags`, si stampano quei numeri indicati come primi dal fatto che la loro corrispondente posizione nell'array `flags` contiene il valore `true`.

Il metodo `sieveMethod` utilizza due variabili locali: `i` con funzione di indice che scorre sui singoli numeri da valutare e `multiplo` che dato un numero itera su tutti i suoi multipli inferiori al valore massimo. Inoltre al metodo viene passato come parametro l'array che rappresenta la proprietà di essere un numero primo.

Dopo aver inizializzato tale array a valori tutti `true`, iterando su tutti i numeri da considerare attraverso un ciclo `for`, si valuta se un numero non sia stato marcato, ovvero la corrispondente posizione nell'array è `false`. Se ció è vero si marcando tutti i multipli di tale numero con un ulteriore ciclo `for`.

3.7 Divisione esatta

L'algoritmo realizzato nel programma presentato in questa sezione risolve il problema di calcolare in maniera esatta l'inverso per ciascuno dei primi 50 numeri primi. Ovviamente non potendo usare le primitive definite nel linguaggio(l'operatore /) dal momento che non si otterrebbe un valore esatto, si utilizza il classico metodo che si applica alla divisione con carta e matita.

Utilizzando degli array per memorizzare le cifre del quoziente via via calcolate e i resti ottenuti ad ogni passo, si itera con operazioni di divisione parziale finché si ottiene un resto il cui valore o è nullo oppure è uguale ad un resto già apparso nei passi precedenti. Vengono inoltre utilizzate due variabili intere come indici di inizio e fine dell'eventuale periodo presente nel quoziente. Infine il programma stampa la sequenza di cifre che rappresentano il risultato sottolineando quelle appartenenti al periodo.

Listato 3.9 - Div50.java - calcolo della divisione esatta.

```
/**
 * calcolo della divisione esatta di 1 / N con n = 2..50
 *
 * @author P.Bison Copyright 1997
 *
 */
public class Div50 {

/**
 * metodo principale
```

```
 */
public static void main(String[] args){
  DivisioneEsatta num = new DivisioneEsatta();
  int i;

  for(i=2;i<=50;i++) num.dividi(i).stampa();
}

} // end of class Div50


/**
 *oggetto per rappresentare una divisione esatta
 *
 * @author P.Bison Copyright 1997
 *
 */
 class DivisioneEsatta {
  int n;
  int[] cifre;
  int[] resti;
  int indice; // indica quale cifra del quoziente si sta calcolando (da 0 a 49)
  int inizio,fine; // inizio e fine dell'eventuale parte periodica del numero

// construttore
/**
 *
 */
 DivisioneEsatta() {
  cifre = new int[50];
  resti = new int[50];
  indice = -1;
}


/**
 * controlla se e' terminata la divisione, ovvero se l'ultimo resto e'
 * zero oppure e' un resto che e' gia apparso precedentemente
 * inoltre calcola l'eventuale periodo
 */
boolean finito(){
  boolean f;
  int i;

  f = resti[indice] == 0;
  if (!f) {
    for (i = 0; i <= indice-1; ++i)
      if (resti[i] == resti[indice]) {
        f = true;
        inizio = i + 1;
        fine = indice;
      }
  }
  return f;
}


/**
 * calcola la divisione esatta 1/n
 * @param n denominatore della frazione
 * @return l'oggetto stesso (this)
 */
DivisioneEsatta dividi(int n) {
  int i;

  this.n = n;
  indice = 0;    // la prima cifra
  cifre[0] = 0;  // e' zero
  resti[0] = 10; // con resto 10
  inizio = fine = -1; // senza periodo

  while (!finito()) {
    indice++ ;
    cifre[indice] = resti[indice-1] / n;
```

```
    resti[indice] = (resti[indice-1] % n) * 10;
  }
  return this;
}

/**
 * stampa il risultato della divisione
 */
void stampa() {
  int i;

 if (n<10) System.out.print("1/ ");
 else System.out.print("1/");
 System.out.print(n+" = " + cifre[0] +".");
  for (i = 1; i <= indice; ++i)
    System.out.print(cifre[i]);
  System.out.println();
  // visualizza eventuale periodo
  if (inizio !=-1) {
    for (i = 1; i <= inizio + 8; ++i)  System.out.print(" ");
    for (i = 1; i <= fine - inizio + 1; ++i) System.out.print("-");
    System.out.println();
        }
}

} // end of class DivisioneEsatta
```

Eseguendo il programma si ottiene l'uscita seguente:

```
1/ 2 = 0.5
1/ 3 = 0.3
         -
1/ 4 = 0.25
1/ 5 = 0.2
1/ 6 = 0.16
          -
1/ 7 = 0.142857
         ------
1/ 8 = 0.125
1/ 9 = 0.1
         -
1/10 = 0.1
1/11 = 0.09
         --
1/12 = 0.083
           -
1/13 = 0.076923
         ------
1/14 = 0.0714285
           ------
1/15 = 0.06
          -
1/16 = 0.0625
1/17 = 0.0588235294117647
         ----------------
1/18 = 0.05
          -
1/19 = 0.052631578947368421
         ------------------
1/20 = 0.05
1/21 = 0.047619
         ------
1/22 = 0.045
          --
1/23 = 0.0434782608695652173913
         ----------------------
1/24 = 0.0416
            -
1/25 = 0.04
1/26 = 0.0384615
           ------
```

```
1/27 = 0.037
         ---
1/28 = 0.03571428
           ------
1/29 = 0.034482758620689655172413793 1
         ---------------------------
1/30 = 0.03
          -
1/31 = 0.032258064516129
         ---------------
1/32 = 0.03125
1/33 = 0.03
         --
1/34 = 0.02941176470588235
          ----------------
1/35 = 0.0285714
          ------
1/36 = 0.027
           -
1/37 = 0.027
         ---
1/38 = 0.0263157894736842105
          ------------------
1/39 = 0.025641
         ------
1/40 = 0.025
1/41 = 0.02439
         -----
1/42 = 0.0238095
          ------
1/43 = 0.023255813953488372093
         ---------------------
1/44 = 0.0227
           --
1/45 = 0.02
          -
1/46 = 0.02173913043478260869565
          ----------------------
1/47 = 0.021276595744680851063829787234042553191489361 7
         -----------------------------------------------
1/48 = 0.02083
             -
1/49 = 0.020408163265306122448979591836734693877551
         ----------------------------------------
1/50 = 0.02
```

3.8 Fattoriale “overloaded”.

Esempio di “overloading” in cui ci sono più metodi di una classe che hanno lo stesso identificatore. Vengono distinti dal tipo dei parametri formali. In questo caso si definiscono varie implementazioni del fattoriale a seconda dei tipi a cui viene applicato.

Listato 3.10 - Factorial.java - classe Factorial con overloading dei metodi.

```
// java1.1

import  java.math.*;

/**
 *  esempi di calcolo del fattoriale con algoritmo iterativo.
 *
 * @author P.Bison Copyright 1997
 */
public class Factorial{

// costanti di tipo BigInteger
  static final BigInteger bigzero = new BigInteger("0");
  static final BigInteger biguno = new BigInteger("1");

// private constructor
// la classe non puo’ essere instanziata
```

```
private Factorial() {}


/**
 * fattoriale iterativo con risultato e parametro di tipo int.
 * calcola il fattoriale del numero passato come parametro. utilizza il tipo int
 * percui si ha un errato valore di ritorno quando si supera il valore massimo rappresentabile
 * ad esempio il fact(15) risulta 2004310016 mentre il risultato corretto e' 1307674368000
 * risultato e' corretto fino a fact(12)
 * @param n valore di cui si deve calcolare il fattoriale
 * @return il fattoriale di n
 */
public static int fact(int n) {
  int ris;
  ris=1;
  while(n!=0)
    ris = ris * n--;
  return ris;
  }

/**
 * fattoriale iterativo con risultato e parametro di tipo long.
 * @param n valore di cui si deve calcolare il fattoriale
 * @return il fattoriale di n
 */
public static long fact(long n) {
  long ris;
  ris=1;
  while(n!=0)
    ris = ris * n--;
  return ris;
  }

/**
 * fattoriale iterativo con risultato e parametro di tipo BigInteger.
 * calcola il fattoriale del numero passato come parametro. utilizza il tipo BigInteger
 * percui non si ha alcun limite nella rappresentazione del numero ad esempio si puo'
 * calcolare il valore fact(5000)
 * @param n valore di cui si deve calcolare il fattoriale
 * @return il fattoriale di n
 */
public static BigInteger fact(BigInteger n) {
 BigInteger ris;

 ris = biguno;
 while(!n.equals(bigzero)){
   ris = ris.multiply(n);
   n = n.subtract(biguno);
   }
 return ris;
}

public static void main(String[] args) {
  int nint=5;
  int nlong=5;
  BigInteger nbig = new BigInteger("5");

  System.out.println("fact("+nint+") = " + fact(nint));
  System.out.println("fact("+nlong+") = " + fact(nlong));

  System.out.print("fact("+nbig.toString() +") =");
  BigInteger bignum;
  bignum = fact(nbig);
  System.out.println(bignum.toString());
  }

} // end of class Factorial
```

Capitolo 4

Programmi orientati agli oggetti.

In questo capitolo vengono illustrati alcuni programmi che utilizzano le funzionalità orientate agli oggetti presenti in Java.

4.1 Numeri complessi

La classe `Complex` implementa il concetto di numero reale mediante l'uso di due variabili di tipo double che rappresentano la parte reale e quella immaginaria di un numero complesso. Inoltre nella classe stessa sono definiti alcuni metodi per creare, manipolare e visualizzare numeri complessi. Vi sono due costruttori per l'istanziazione di elementi di tipo Complex: uno crea il numero complesso nullo mentre l'altro lo crea con valori passati come parametri. Altri metodi permettono di ottenere la parte reale o quella immaginaria (metodi parteReale e parteImmaginaria), oppure di convertire il numero complesso in una rappresentazione testuale in formato stringa (metodo toString). Infine vi sono metodi che realizzano le operazioni tra numeri complessi (metodi multiply e add) o con valori scalari (metodo scalar che è overloaded in tre varianti).

Listato 4.1 - Complex.java - rappresentazione dei numeri complessi

```
/**
 * numeri complessi
 *
 * @author P.Bison
 *
 */

public class Complex
{

  double parteReale;
  double parteImmaginaria;

/* constructors */
/**
 * crea un numero complesso con valori nulli
 * @param n la dimensione del quadrato
 */
  public Complex ()
  {

    parteReale = parteImmaginaria = 0.0;


  }

/**
 * crea un numero complesso
 * @param x parte reale
 * @param y parte immaginaria
```

```
 */
  public Complex (double x, double y)
  {

    parteReale = x;
    parteImmaginaria = y;


  }


/*
   metodi pubblici
 */


/**
 * ritorna parte reale
 */
  public double parteReale()
  {
    return parteReale;
  }

/**
 * ritorna parte immaginaria
 */
  public double parteImmaginaria()
  {
    return parteImmaginaria;
  }

/**
 * somma due numeri complessi
 * @param x il secondo operando della somma
 * @return un numero complesso il cui valore e' this+x
 */
  public Complex add (Complex x)
  {
    return new Complex (parteReale + x.parteReale, parteImmaginaria + x.parteImmaginaria);
  }

/**
 * moltiplica due numeri complessi
 * @param x il secondo operando della somma
 * @return un numero complesso il cui valore e' this*x
 */
  public Complex multiply (Complex x)
  {
    return new Complex (parteReale * x.parteReale - parteImmaginaria * x.parteImmaginaria,
         parteReale * x.parteImmaginaria + parteImmaginaria * x.parteReale);
  }

/**
 * moltiplica un numero complesso per uno scalare di tipo double
 * @param x il valore scalare
 * @return un numero complesso il cui valore e' x*this
 */
  public Complex scalar (double x)
  {
    return new Complex (parteReale * x, parteImmaginaria * x);
  }

/**
```

```
 * moltiplica un numero complesso per uno scalare di tipo float
 * @param x il valore scalare
 * @return un numero complesso il cui valore e' x*this
 */
  public Complex scalar (float x)
  {
    return new Complex (parteReale * x, parteImmaginaria * x);
  }

/**
 * moltiplica un numero complesso per uno scalare di tipo int
 * @param x il valore scalare
 * @return un numero complesso il cui valore e' x*this
 */
  public Complex scalar (int x)
  {
    return new Complex (parteReale * x, parteImmaginaria * x);
  }

/**
 * calcola valore assoluto di un numero complesso
 * @return il valore assoluto
 */
  public double abs ()
  {
    return Math.sqrt (parteReale * parteReale + parteImmaginaria * parteImmaginaria);
  }



/**
 * converte un numero complesso in una stringa
 * @return una stringa che rappresenta il numero complesso
 */
  public String toString ()
  {
    String stReale;
    String stImmaginaria;

      stReale = String.valueOf (parteReale);
      stImmaginaria = String.valueOf (parteImmaginaria);
      return new String (stReale + "+i" + stImmaginaria);
  }


/**
 * metodo principale
 */
  public static void main (String[]args)
  {
    Complex num1 = new Complex ();
    Complex num2 = new Complex (-5.3, 17.56);
    Complex num3;


      System.out.println (num1);
      System.out.println (num2);
      num3 = num2.add (new Complex (-1.2, 0.8));
      System.out.println (num3);
      num3 = num2.scalar (1.5);
      System.out.println (num3);
      System.out.println (num3.abs ());
  }
```

```
} // end of class Complex
```

4.2 Calcolo matriciale

In maniera simile al caso precednte si definisce la classe SimpleMatrixche rappresenta le matrici NxM con elementi di tipo reale. Si utilizzano due variabili intere per memorizzare il numero di righe e di colonne per una particolare istanza ed un array di array per memorizzare gli elementi della matrice. Al solito si definiscono costruttori per creare elementi di tipo SimpleMatrix, metodi per le operazioni matriciali o con scalari e metodi per la converisone a stringa.

Listato 4.2 - SimpleMatrix.java - rappresentazione di matrici NxM

```
/**
 * numeri complessi
 *
 * @author P.Bison
 *
 */

public class SimpleMatrix
{

  int righe;                    /* numero di righe */
  int colonne;                  /* numero di colonne */
  double[][] data;

/* constructors */
/**
 * crea una matrice 4x4
 */
  public SimpleMatrix ()
  {
    righe = colonne = 4;
    data = new double[4][4];
  }

/**
 * crea una matrice di zeri
 * @paramn n numero di righe
 * @param m numero di colonne
 */
  public SimpleMatrix (int n, int m)
  {
    righe = n;
    colonne = m;
    data = new double[n][m];
  }


/**
 * crea una matrice di zeri
 * @paramn n numero di righe
 * @param m numero di colonne
 */
  public SimpleMatrix (double[][]initdata)
  {
    int i, j;

      righe = initdata.length;
      colonne = initdata[0].length;
      data = new double[righe][colonne];
```

```
    for (i = 0; i < righe; i++)
      for (j = 0; j < colonne; j++)
          data[i][j] = initdata[i][j];
  }


/*
   metodi pubblici
 */

/**
 * somma due matrici
 * @param x il secondo operando della somma
 * @return una matrice il cui valore e' this+x
 */
  public SimpleMatrix add (SimpleMatrix x)
  {
    SimpleMatrix ris;
    int i,j;

    if ((righe != x.righe) || (colonne != x.colonne))
      throw new ArithmeticException ("Incompatible matrix");
    ris = new SimpleMatrix (righe, colonne);
    for(i=0;i<righe;i++)
      for(j=0;j<colonne;j++)
        ris.data[i][j] = data[i][j]+x.data[i][j];
    return ris;
  }

/**
 * moltiplica due matrici
 * @param x il secondo operando della moltiplicazione
 * @return una matrice il cui valore e' this*x
 */
  public SimpleMatrix multiply (SimpleMatrix x)
  {
    SimpleMatrix ris;
    double temp;
    int i,j,k;

    if ((righe != x.colonne) || (colonne != x.righe))
      throw new ArithmeticException ("Incompatible matrix");
    ris = new SimpleMatrix (righe, colonne);
    for(i=0;i<righe;i++)
      for(j=0;j<colonne;j++) {
        temp=0.0;
        for(k=0;k<righe;k++) temp = temp + data[i][k]*x.data[k][j];
        ris.data[i][j] = temp;
       }
    return ris;
  }

/**
 * moltiplica una matrice per uno scalare di tipo double
 * @param x il valore scalare
 * @return una matrice il cui valore e' x*this
 */
  public SimpleMatrix scalar (double x)
  {
    SimpleMatrix ris;
    int i, j;

      ris = new SimpleMatrix (righe, colonne);
    for (i = 0; i < righe; i++)
```

```
      for (j = 0; j < colonne; j++)
          data[i][j] *= x;
      return ris;
  }



/**
 * converte un numero complesso in una stringa
 * @return una stringa che rappresenta il numero complesso
 */
  public String toString ()
  {
    String matrep = "[";
    int i, j;

    for (i = 0; i < righe; i++)
      {
        matrep += "[";
        for (j = 0; j < colonne; j++)
          {
            matrep = matrep + String.valueOf (data[i][j]);
            if (j != colonne - 1)
              matrep += ",";
          }
        matrep += "]";
      }
    matrep += "]";
    return matrep;
  }


/**
 * metodo principale
 */
  public static void main (String[]args)
  {
    SimpleMatrix mat1, mat2, mat3;
    double[][] a ={{1.2, -0.75, 1.8},
                   {-0.99, 3.45, -4.89},
                   {-9.8, 0.08, 1.98}};
 double[][] a1 ={{1.0, 0.0, 0.0},
                   {0.0, 1.0, 0.0},
                   {0.0, 0.0, 1.0}};

    mat1 = new SimpleMatrix (a1);
    System.out.println (mat1.toString ());
    mat2 = new SimpleMatrix (a);
    System.out.println (mat2.toString ());
    mat3=mat2.add (mat1);
    System.out.println (mat3.toString ());
    mat3=mat2.multiply (mat1);
    System.out.println (mat3.toString ());
  }

}  /* end of class SimpleMatrix */
```

4.3 Quadrato magico

Un **quadrato magico perfetto**, normale, di ordine n è una scacchiera $n \times n$ che contiene nelle sue n^2 caselle una ed una sola volta ciascuno dei numeri interi da 1 ad n^2, in modo che la somma degli n valori su ogni riga, su ogni colonna

è su ciascuna delle due diagonali principali sia la stessa. Questa somma viene chiamata **costante magica**.
Ad esempio il seguente quadrato magico di ordine 5 ha per costante magica il valore 65.

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

Per costruire un quadrato magico di ordine n, con n dispari, esiste un algoritmo di costruzione relativamente semplice dovuto a De la Loubère, fondato sui seguenti quattro principi:

- si inserisce il primo valore 1 nella casella situata immediatamente al di sotto della casella centrale;
- se si è inserito l'intero $x-1$ nella casella (i,j) (con i indice di riga e j indice di colonna), il valore intero successivo x deve essere inserito nella casella $(i+1, j+1)$;
- se nel calcolo di $i+1$ oppure di $j+1$ si ottiene un valore che supera n, tale indice viene posto uguale ad 1
- se si deve inserire un valore di una casella già occupata (sia ad esempio occupata la casella (k,m)), si inserisce il valore nella casella $(k+1, m-1)$. Se $m-1$ vale 0, si prende come indice il valore n.

Listato 4.3 - MagicSquare.java - rappresentazione del quadrato magico

```
/**
 * implementazione del quadrato magico
 *
 * @author P.Bison
 *
 */

public class MagicSquare {

 int dim; // dimensione del quadrato
 int[] celle; // vettore che memorizza i valori delle celle.
              // e' composto da dim x dim elementi e la cella in posizione i,j
              // e' indicizzata da dim x i + j

// constructor
/**
 * crea un quadrato magico con dimensione definita dal parametro e valori tutti nulli
 * @param n la dimensione del quadrato
 */
public MagicSquare(int n){
  int i,j; // indici di cella

  dim = n;
  celle = new int[n * n];

// inizializzazione a zero
  for(i=0;i<n;i++)
    for(j=0;j<n;j++) celle[n * i +j] = 0;

}

// metodi privati

// calcola il successore per un indice l
private int succIndex(int l) {
  if (l==dim-1) return 0;
  else return l+1;
}

// calcola il predecessore per un indice l
```

```
private int predIndex(int l) {
  if (l==0) return dim-1;
  else return l-1;
}


// metodi pubblici

/**
 * genera i valori magici
 * @return l'oggetto stesso (this)
 */
public MagicSquare makeMagic(){
  int i,j; // indici di cella
  int k; // indice per il vettore celle (va da 1 a dim x dim)


  i = dim / 2 + 1; j = dim / 2;
  for(k=1; k<= dim*dim;k++) {
    celle[dim*i + j] = k;

    // ricerca successiva cella libera se non e' l'ultima

    if ( k!= dim*dim) {
      i = succIndex(i); j= succIndex(j);
      while(celle[dim*i+j] != 0) {
        i = succIndex(i); j = predIndex(j);
        }
      }
   }
  return this;
}

/**
 * calcola la costante magica
 * @return il valore della costante
 */
public int magicConst()
{
  int i,j;
  int magicnum;
  int temp;
  boolean notmagic;

  magicnum = 0;

  for (i=0;i<dim;i++) magicnum += celle[i]; // somma prima riga

  if (magicnum <= 1) return -1; // non puo' essere magico

  // controlla se il valore di magicnum e' uguale dappertutto
  notmagic = false;
  for(i=1;i<dim;i++) {  // controlla righe
    for(temp=0,j=0;j<dim;j++) temp += celle[dim*i+j];
        if (notmagic=notmagic || (magicnum!=temp)) break;
        }

  for(j=0;j<dim;j++) {   // controlla colonne
    for(temp=0,i=0;i<dim;i++) temp += celle[dim*i+j];
        if (notmagic=notmagic || (magicnum!=temp)) break;
        }

  for(temp=0,i=0,j=0;i<dim;i++,j++)  // controlla diagonale
    temp += celle[dim*i+j];
```

```
  notmagic=notmagic || (magicnum!=temp);

  for(temp=0,i=0,j=dim-1;i<dim;i++,j--)  // controlla altra diagonale
    temp += celle[dim*i+j];
 notmagic=notmagic || (magicnum!=temp);

  if (notmagic) return -1;
  else return magicnum;
}

/**
 * stampa la dimensione, i valori e la costante magica del quadrato magico
 * @return l'oggetto stesso (this)
 */
public MagicSquare display() {
  int i,j,k; // contatori
  String value;

  System.out.println("Dimensione: "+dim);
  for(i=0;i<dim;i++) {
    for(j=0;j<dim;j++) {
      value = String.valueOf(celle[dim * i +j]);
      for(k=0;k<3-value.length();k++) System.out.print(" ");
      System.out.print(value);
      }
    System.out.println();
  }
System.out.println("Costante magica: " + magicConst());
System.out.println();
return this;
}

/**
 * metodo principale
 */
public static void main(String[] args){
  MagicSquare square1 = new MagicSquare(5);
  MagicSquare square2 = new MagicSquare(9);

  square1.display();
  square1.makeMagic();
  square1.display();

  square2.makeMagic().display();
}

} // end of class MagicSquare
```

Eseguendo il programma si ottiene l'uscita seguente:

```
Dimensione: 5
  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
Costante magica: -1

Dimensione: 5
 11 24  7 20  3
  4 12 25  8 16
 17  5 13 21  9
```

```
 10 18  1 14 22
 23  6 19  2 15
Costante magica: 65

Dimensione: 9
 37 78 29 70 21 62 13 54  5
  6 38 79 30 71 22 63 14 46
 47  7 39 80 31 72 23 55 15
 16 48  8 40 81 32 64 24 56
 57 17 49  9 41 73 33 65 25
 26 58 18 50  1 42 74 34 66
 67 27 59 10 51  2 43 75 35
 36 68 19 60 11 52  3 44 76
 77 28 69 20 61 12 53  4 45
Costante magica: 369
```

4.4 Figure geometriche

In questa sezione si illustra l'utilizzo del concetto di classe astratta per definire degli elementi comuni ad un insieme di oggetti che dovranno essere realizzati nelle sottoclassi definenti tali oggetti. Inoltre vi è l'uso della funzionalitá package.

In questo esempio si considera il caso delle figure geometriche caratterizzate dal loro perimetro ed area e dai metodi per calcolare tali valori. Mentre la rappresentazione del perimetro ed area sono comuni a tutte le figure geometriche, le procedure per calcolare tali elementi sono proprie di ciascun tipo di figura geometrica e non è possibile avere dei metodi generali. Quindi si utilizza una classe astratta, che non puó essere istanziata, per definire le variabili che rappresentano l'area ed il perimetro e per indicare i metodi che tutte le sottoclassi devvono realizzare.

Listato 4.4 - FiguraGeometrica.java - una classe astratta

```
package FigureGeometriche;
public abstract class FiguraGeometrica{

  float area=0;
  float perimetro=0;

   abstract void calcolaArea();
   abstract void calcolaPerimetro();

  public float getArea()
  {
    if (area == 0) calcolaArea();
    return area;
  }
  public float getPerimetro()
  {
    if (perimetro == 0) calcolaPerimetro();
    return perimetro;
  }

}
```

Data questa classe astratta, si possono definire delle sottoclassi che rappresentano particolati figure geometriche. Queste sottoclassi oltre a realizzare i metodi imposti dalla classe astratta, possono ovviamente introdurre nuove variabili e/o metodi.

La classe `Triangolo` definisce le proprietá di un generico triangolo introducendo tre variabili per rappresentare i tre lati di un triangolo e realizzando il metodo per calcolo del perimetro come somma dei tre lati e il calcolo dell'area attraverso la formula di Erone.

Listato 4.5 - Triangolo.java - definizione di triangolo

```
package FigureGeometriche;
public class Triangolo extends FiguraGeometrica {
  float a,b,c; // i tre lati di un triangolo

public Triangolo(float l1, float l2, float l3)
  {
    a=l1;
    b=l2;
    c=l3;
  }

   void calcolaArea()
  {
    float p;

    p = (a + b + c)/2.0f;
    area = (float) Math.sqrt(p*(p-a)*(p-b)*(p-c)); // formula di Erone
  }

  void calcolaPerimetro()
  {
    perimetro = a + b + c;
  }

  public String toString()
  {
    return "Triangolo ";
  }
}
```

A sua volta questa classe puó essere specializzata per tipi particolari di triangoli dove viene ridefinito il metodo per il calcolo dell'area. Quindi si ha la classe per i triangoli rettangoli,

Listato 4.6 - TriangoloRett.java - definizione di triangolo rettangolo

```
package FigureGeometriche;
public class TriangoloRett extends Triangolo {
  // a e b cateti, c ipotenusa

public TriangoloRett(float c1, float c2)
  {
   super(c1,c2,(float)Math.sqrt(c1*c1+c2*c2)); // costruttore della superclasse
  }

   void calcolaArea()
  {
    area = (a*b)/2.0f;
   }

  public String toString()
  {
    return "TriangoloRett ";
  }
}
```

quella per i triangoli isosceli

Listato 4.7 - TrinagoloIsoscele.java - definizione di triangolo isoscele

```
package FigureGeometriche;
public class TriangoloIsoscele extends Triangolo {
  // a base, b e c lati uguali

public TriangoloIsoscele(float base, float lato)
  {
   super(base,lato,lato); // costruttore della superclasse
  }

   void calcolaArea()
  {
    area = a/2f*(float)Math.sqrt(b*b-(a*a)/4f);
   }

  public String toString()
  {
    return "TriangoloIsoscele ";
  }
}
```

e quella per i triangoli equilateri

Listato 4.8 - TriangoloEquilatero.java - definizione di triangolo equilatero

```
package FigureGeometriche;
public class TriangoloEquilatero extends Triangolo {
  // a base, b e c lati uguali

public TriangoloEquilatero( float lato)
  {
   super(lato,lato,lato); // costruttore della superclasse
  }

   void calcolaArea()
  {
    area = a*a*0.433f; //sqrt(3)/4
   }

  public String toString()
  {
    return "TriangoloEquilatero ";
  }
}
```

Altro esempio è la classe `PoligonoRegolare` che rappresenta i poligoni formati da `n` lati, ciascun lato di lunghezza `lato`.

Listato 4.9 - PoligonoRegolare.java - definizione del poligono regolare

```
package FigureGeometriche;
public class PoligonoRegolare extends FiguraGeometrica {
   int nlati; // numero di lati
   float lato; // lungheza del lato

public PoligonoRegolare(int initnlati, float initlato)
  {
```

```
    nlati = initnlati;
    lato = initlato;
  }

   void calcolaArea()
  {

    area =  lato*lato*nlati/4f/(float)Math.tan(Math.PI/nlati); // formula di Erone
  }

  void calcolaPerimetro()
  {
    perimetro =nlati * lato;
  }

  public String toString()
  {
    return "PoligonoRegolare ";
  }
}
```

Infine la classe `main` mostra un semplice uso delle classi descritte precedentemente.

Listato 4.10 - Main.java - programma di esempio

```
import FigureGeometriche.*;
public class Main{
  public static void main(String[] args)
  {
    FiguraGeometrica fig;

   fig = new Triangolo(2f,6f,5f);
   System.out.println(fig+ " " +fig.getArea()+", "+fig.getPerimetro());
   fig = new TriangoloRett(2f,6f);
   System.out.println(fig+ " " +fig.getArea()+", "+fig.getPerimetro());
  fig = new TriangoloIsoscele(2f,6f);
   System.out.println(fig+ " " +fig.getArea()+", "+fig.getPerimetro());
  fig = new TriangoloEquilatero(6f);
   System.out.println(fig+ " " +fig.getArea()+", "+fig.getPerimetro());
  fig = new PoligonoRegolare(5,4f);
   System.out.println(fig+ " " +fig.getArea()+", "+fig.getPerimetro());
  }
}
```

Eseguendo tale programma si ottiene:

```
Triangolo  4.6837487, 13.0
TriangoloRett  6.0, 14.324555
TriangoloIsoscele  5.91608, 14.0
TriangoloEquilatero  15.588, 18.0
PoligonoRegolare  27.527637, 20.0
```

4.5 Ereditarietá multipla

Il programma presentato in questa sezione mostra come è possibile avere in Java qualcosa simile all'ereditarietá multipla utilizzando le interfacce. Nell'esempio si realizza un orologio composto da un orologio digitale e da uno analogico.L'ereditarietá multipla è solo mimata nel senso che vengono forniti solamente i metodi propri di ciascun

componente (digitale o analogico) per la visualizzazione dell´ora dell´orologio, mentre i dati (variabili che memorizzano l'ora) sono unici. Diverso approccio si dovrebbe adottare nel caso in cui si voglia ereditare da diversi oggetti sia variabili che metodi.

Listato 4.11 - DiAnaClock.java - ereditarietá multipla per mezzo dell'interfacce.

```
/**
* esempio di ereditarieta' multipla
* mediante uso di interfacce
*
*/

import java.util.*;

/**
* classe Clock
* definisce i dati relativi ad un orologio:
* ore e minuti
*/
class Clock{
int hours;
int minutes;

/**
* costruttore
* assegna ai due campi l'ora del calcolatore
*/
Clock(){
hours=new Date().getHours();
minutes=new Date().getMinutes();
}
} // endClass Clock

/**
* interfaccia per un orologio digitale
*
*/
interface DigitalClock {
void showTime();
}

/**
* interfaccia per un orologio analogico
*
*/
interface AnalogClock {
void showTime();
}

/**
le classi ADigitalClock e AAnalogClock realizzano le due interfacce
Definiscono un metodo statico di classe per implementare le differenti
visualizzazioni dell'ora. questo metodo viene chiamato dal metodo che realizza
il corrispondente metodo dell'interfaccia associata.
Inoltre il metodo statico puo' essere attivato da altre classi che vogliano realizzare
la stessa interfaccia. In questo modo vi e' una sola implementazione dell'interfaccia
*/

class ADigitalClock extends Clock
      implements DigitalClock{

/**
 metodo statico che implementa la visualizzazione per un orologio
 di tipo digitale. stampa le ore ed i minuti separati da :
```

```
@param cl un oggetto di tipo Clock i cui dati verrano visualizzati
*/
public static void classShowTime(Clock cl){
  System.out.println(cl.hours+":"+cl.minutes);
  }

/**
realizzazione del metodo definito nell'interfaccia
chiama il metodo statico passando come parametro l'oggetto su cui
e' stato attivato
*/
public void showTime(){
  classShowTime(this);
  }
} // endClass ADigitalClock

class AAnalogClock extends Clock
   implements AnalogClock{

/**
 metodo statico che implementa la visualizzazione per un orologio
 di tipo digitale. stampa le ore ed i minuti come linee di *
@param cl un oggetto di tipo Clock i cui dati verrano visualizzati
*/
public static void classShowTime(Clock cl){
  int i;
  System.out.print("h:");
  for(i=0;i<cl.hours;i++)
    System.out.print("*");
  System.out.println();
  System.out.print("m:");
  for(i=0;i<cl.minutes;i++)
    System.out.print("*");
  System.out.println();
  }

/**
realizzazione del metodo definito nell'interfaccia
chiama il metodo statico passando come parametro l'oggetto su cui
e' stato attivato
*/
public void showTime(){
  classShowTime(this);
  }
}

/**
realizzazione di un orologio digitale-analogico
*/
class DiAnaClock extends Clock
  implements DigitalClock,AnalogClock{

/**
relizzazione del metodo definito per entrambe le interfacce
chiama le singole realizzazioni, rappresentate dai metodi statici delle
classi che implementano le interfacce, passando come parametro l'oggetto
su cui e' stato attivato
*/
public void showTime(){
ADigitalClock.classShowTime(this);
AAnalogClock.classShowTime(this);
}

public static void main(){
```

```
  ADigitalClock orologiodigitale = new ADigitalClock();
  AAnalogClock orologioanalogico = new AAnalogClock();
  DiAnaClock orologio = new DiAnaClock();

  System.out.println("Orologio digitale");
  orologiodigitale.showTime();
  System.out.println("Orologio analogico");
  orologioanalogico.showTime();
  System.out.println("Orologio digitale-analogico");
  orologio.showTime();
  }
} // endClass DiAnaClock
```

Capitolo 5

Programmi ricorsivi.

In questo capitolo si mostra come è possibile realizzare programmi ricorsivi nel linguaggio java.

5.1 Fattoriale

La funzione fattoriale, già vista nel capitolo precedente, può essere espressa in termini ricorsivi secondo la seguente definizione per ricorrenza:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n(n-1)! & \text{per } n > 0 \end{cases}$$

Il listato 5.1 mostra la definizione di una classe `Ricfactorial` contenente due metodi, entrambi con nome `ricFact`, che implementano in maniera ricorsiva il calcolo del fattoriale, uno per dati di tipo `int` l'altro per dati `BigInteger`.

Il primo metodo è molto semplice. Se il valore del parametro `n` è 0 si ritorna il valore 1, altrimenti si ritorna il prodotto tra `n` ed il fattoriale di `n - 1`.

Il secondo metodo è simile solo che sono stati utilizzati i metodi definiti nella classe `BigInteger` per eseguire le operazioni. L'espressione `n.multiply(ricFact(n.subtract(biguno)))` ritorna un dato (oggetto) di tipo BigInteger ottenuto dalla moltiplicazione di n con il numero ritornato dall'applicazione del fattoriale stesso ad un numero che è ottenuto dalla sottrazione di uno a `n`.

Listato 5.1 - RicFactorial.java - fattoriale ricorsivo.

```
// java1.1

import  java.math.*;

/**
 *  esempi di calcolo del fattoriale con algoritmo ricorsivo.
 *
 * @author P.Bison Copyright 1997
 */
public final class RicFactorial{

// costanti di tipo BigInteger
static final BigInteger bigzero = new BigInteger("0");
static final BigInteger biguno = new BigInteger("1");

// private constructor
// la classe non puo' essere instanziata
private RicFactorial() {}

/**
 * fattoriale ricorsivo con risultato e parametro di tipo int.
 * calcola il fattoriale del numero passato come parametro. utilizza il tipo int
 * percui si ha un errato valore di ritorno quando si supera il valore massimo rappresentabile
 * ad esempio il ricFact(15) risulta 2004310016 mentre il risultato corretto e' 1307674368000
 * risultato e' corretto fino a Ricfact(12)
 * @param n valore di cui si deve calcolare il fattoriale
```

```
 * @return il fattoriale di n
 */
public static int ricFact(int n) {

  if (n==0) return 1;
  else return n * ricFact(n-1);
  }

/**
 * fattoriale ricorsivo con risultato e parametro di tipo BigInteger.
 * calcola il fattoriale del numero passato come parametro. utilizza il tipo BigInteger
 * percui non si ha alcun limite nella rappresentazione del numero ad esempio si puo’
 * calcolare il valore BigRicfact(5000)
 * @param n valore di cui si deve calcolare il fattoriale
 * @return il fattoriale di n
 */
public static BigInteger ricFact(BigInteger n) {

 if (n.equals(bigzero)) return biguno;
  else return n.multiply(ricFact(n.subtract(biguno)));
}

public static void main(String[] args) {
  int nint=5;
  BigInteger nbig = new BigInteger("500");

  System.out.println("ricFact("+nint+") = " + ricFact(nint));

  System.out.print("ricFact("+nbig.toString() +") =");
  BigInteger bignum;
  bignum = ricFact(nbig);
  System.out.println(bignum.toString());
  }
} // end of class RicFactorial
```

Il programma stampa la seguente uscita, da cui si vede che utilizzando il tipo BigInteger si puó calcolare il fattoriale di 500:

```
ricFact(5) = 120
ricFact(500) = 12201368259911100687012387854230469262535743428031928
42192413588385845373153881997605496447502203281863013616477148203584
16337872207817720048078520515932928547790757193933060377296085908627
04291745478824249127263443056701732707694610628023104526442188787894
65754777149863494367781037644274033827365397471386477878495438489595
53753799042324106127132698432774571554630997720278101456108118837370
95310163563244329870295638966289116589747695720879269288712817800702
65174507768410719624390394322536422605234945850129918571501248706961
56814162535905669342381300885624924689156412677565448188650659384795
17753608940057452389403357984763639449053130623237490664450488246650
75946735862074637925184200459369692981022263971952597190945217823331
75693458150855233282076282002340262690789834245171200620771464097945
61161276291459512372299133401695523638509428855920187274337951730145
86357570828355780158735432768888680120399882384702151467605445407663
53598417443048012893831389688163948746965881750450692636533817505547
8128640000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000
```

5.2 Programmi sulle stringhe

I due programmi visti nel capitolo 1 che operavano su dati di tipo String vengono qui proposti nella loro versione ricorsiva.

Nel caso della verifica se una stringa è palindroma, si è definito il metodo `isPalin` con il parametro `st` che è la stringa da verificare. Se la lunghezza della stringa è uguale a 1 (stringa formata da un solo carattere) si ritorna il valore `true` poiché tale stringa è palindroma. Altrimenti si confrontano il primo ed ultimo carattere e si vede se la stringa ottenuta togliendo il primo e l'ultimo carattere è palindroma.

Listato 5.2 - RicPalin.java; verifica se una stringa é palindroma.

```
/**
 *
 *  verifica se una stringa e' palindroma in maniera ricorsiva
 *
 *  @author P.Bison Copyright 1997
 *
 */
public class RicPalin {

public static boolean isPalin(String st) {

if(st.length()==1) return true;
else return  (st.charAt(0)==st.charAt(st.length()-1)) &&
             isPalin(st.substring(1,st.length()-1));
}

/**
 * programma principale
 */
  public static void main(String[] args) {

   for(int i=0; i < args.length; i++){ // itera sul numero di argomenti
    System.out.print(args[i]);
    if (isPalin(args[i])) System.out.println(" e'palindroma");
    else System.out.println(" non e'palindroma");
    }
  }
}
```

Nel caso del Reverse se la lunghezza della stringa vale 1 si ritorna la stringa stessa, altrimenti si concatena alla stringa ottenuta prendendo l'ultimo carattere (`st.substring(st.length()-1)` il reverse della stringa ottenuta da quella di partenza togliendo l'ultimo carattere (`st.substring(0,st.length()-1)`).

Listato 5.3 - RicReverse.java - capovolge una stringa in maniera ricorsiva.

```
/**
 *
 *  verifica se una stringa e' palindroma in maniera ricorsiva
 *
 *  @author P.Bison Copyright 1997
 *
 */
public class RicReverse {

public static String doReverse(String st) {

if(st.length()==1) return st;
else return  st.substring(st.length()-1).concat(doReverse(st.substring(0,st.length()-1)));
}

/**
 * programma principale
 */
  public static void main(String[] args) {
```

```
   for(int i=0; i < args.length; i++){ // itera sul numero di argomenti
    System.out.print(args[i]+" ");
    System.out.println(doReverse(args[i]));
    }
  }
}
```

5.3 Anagrammi di una lista

L'anagramma di una lista è una combinazione qualunque degli elementi componenti la stringa. Se la stringa è composto da n elementi, si hanno $n!$ possibili combinazioni. In questa sezione sono riportati due programmi che permettono di calcolare gli anagrammi di una lista. Il primo (listato 5.4) genera e stampa tutti gli anagrammi con una sola attivazione del metodo, mentre nel secondo programma (listato 5.5) il metodo genera un anagramma diverso ad ogni attivazione, finchè non ha generato tutti i possibili anagrammi.

Listato 5.4 - SimpleAnag.java - stampa degli anagrammi di una stringa.

```
/**
 * stampa degli anagrammi di una stringa
 *
 * @author P.Bison Copyright 1997
 *
 */
public class SimpleAnag
{
  static int livelloricorsione = -1;    /* indica il livello di ricorsione */
/**
 * costruttore di classe
 * la classe non puo' essere instanziata
 */
  private SimpleAnag ()
  {
  }

  public static void calcolaAnag (StringBuffer temp, StringBuffer anag)
  {
    int i, j;
    StringBuffer subtemp; /* stringa vuota */

      livelloricorsione++;

    if (temp.length () == 1)
      {
        anag.setCharAt (livelloricorsione, temp.charAt (0));
        System.out.println (anag);
      }
    else
      for (i = 0; i < temp.length (); i++)
        {
          anag.setCharAt (livelloricorsione, temp.charAt (i));
          subtemp = new StringBuffer ();

          /* crea una nuova stringa di n-1 caratteri senza l'i-esimo carattere */
          for (j = 0; j < temp.length (); j++)
            if (j != i)
              subtemp.insert (subtemp.length (), temp.charAt (j));
          calcolaAnag (subtemp, anag);
        }
```

```
    livelloricorsione--;
  }


/**
 * metodo principale
 */
  public static void main (String[]args)
  {
    String unaStringa;
    String temp;

    if (args.length > 0)
        unaStringa = args[0];
    else
        unaStringa = "AbC";

      calcolaAnag (new StringBuffer (unaStringa), new StringBuffer (unaStringa));

  }
}               // end of class SimpleAnag
```

Listato 5.5 - Anagrammi.java - generazione di anagrammi.

```
/**
 * calcolo degli anagrammi di una stringa
 *
 * @author P.Bison Copyright 1997
 *
 */
public class Anagrammi
{
  int livelloricorsione;        /* indica il livello di ricorsione */
  int numGenerati;      /* numero di anagrammi generati */
  int numGenerandi;         /* numero di anagrammi che si sta generando */
  String st;                    /* stringa di cui si calcola gli anagrammi */
  StringBuffer anag;            /* stringa di lavoro in cui viene generato l'anagramma */

/**
 * costruttore di classe
 */
  public Anagrammi (String t)
  {
    int i;

      livelloricorsione = -1;
      numGenerati = 0;          /* nessun anagramma generato */
      st = t;
      anag = new StringBuffer (st);
  }
/*

   metodi privati

 */
  boolean calcolaAnag (StringBuffer temp)
  {
    int i, j;
    StringBuffer subtemp;
```

```
    boolean generato = false;

      livelloricorsione++;
    if (temp.length () == 1)
      {
        anag.setCharAt (livelloricorsione, temp.charAt (0));
        numGenerandi++;
        if (numGenerandi > numGenerati)
          {
            numGenerati = numGenerandi;
            generato = true;
          }
        else
            generato = false;
      }
    else
      for (i = 0; i < temp.length (); i++)
        {
          anag.setCharAt (livelloricorsione, temp.charAt (i));
          subtemp = new StringBuffer ();

          /* crea una nuova stringa di n-1 caratteri senza l'i-esimo carattere */
          for (j = 0; j < temp.length (); j++)
            if (j != i)
              subtemp.insert (subtemp.length (), temp.charAt (j));
          generato = calcolaAnag (subtemp);
          if (generato)
            break;
        }
    livelloricorsione--;
    return generato;
  }


/*

   metodi pubblici

 */

  public String getAnagramma ()
  {
    boolean generato;
      numGenerandi = 0;
      generato = calcolaAnag (new StringBuffer (st));
    if (generato)
        return anag.toString ();
    else
      {
        numGenerati = 0;
        return "";
      }
  }

/**
 * metodo principale
 */
  public static void main (String[]args)
  {
    Anagrammi unaStringa;
    String temp;

    if (args.length > 0)
        unaStringa = new Anagrammi (args[0]);
```

```
    else
        unaStringa = new Anagrammi ("TaFo");

      temp = unaStringa.getAnagramma ();
    while (temp.length () != 0)
      {
        System.out.println (temp);
        temp = unaStringa.getAnagramma ();
      }

  }

}               // end of class Anagrammi
```

5.4 Torre di Hanoi

Nel gioco della torre di Hanoi si hanno a disposizione una tavola con tre pioli ed un certo numero di dischi forati di diametro differente. La configurazione iniziale è formata da tutti i dischi con diametro decrescente infilati in un piolo. Obiettivo del gioco è muovere tale torre su un altro piolo con le condizioni che si può muovere un solo disco alla volta e che nessun disco più largo può essere su uno più piccolo.

Un semplice algoritmo ricorsivo può essere sviluppato considerando che per muovere una torre di n dischi da un piolo ad un altro si può, prima muovere la torre formata dagli n-1 dischi superiori sul piolo non di arrivo, poi muovere l'unico disco rimasto sul piolo d'arrivo ed infine muovere la torre di n-1 dischi sul piolo finale. Ora per muovere la torre di n-1 dischi si possono ripetere i medesimi passi considerando una torre formata da n-2 dischi, poi una da n-3 dischi, e così via finché si arriva ad una torre formata da un solo disco che si può muovere liberamente.

Un quesito che ci si chiede è in quante mosse si muove una torre di n dischi da un piolo ad un altro. Dalla strategia di gioco visto prima si deduce che il numero di mosse è pari a due volte il numero di mosse necessarie per spostare la torre di n-1 dischi più uno, che corrsponde alla mossa di muovere l'ultimo disco. Dato che si conosce il numero di mosse per muovere una torre formata da un solo disco (1 mossa), si può definire la funzione $M(n)$, che ritorna il numero di mosse in funzione di n, con la seguente definizione per ricorrenza:

$$M(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2M(n-1)+1 & \text{per } n > 1 \end{cases}$$

da cui è possibile dimostrare che $M(n) = 2^n - 1$.

Listato 5.6 - HanoiTower.java - la torre di Hanoi.

```
// java 1.1

import java.math.*;

/**
 * rappresentazione della torre di hanoi
 *
 * @author P.Bison Copyright 1997
 *
 */
public class HanoiTower {
  static final int maxdischi = 64;

  int ndischi; // numero totale dei dischi
  int[] numDischiPiolo; // dischi per piolo. pioli 0,1 e 2
  int[][] dischiPiolo; // lista dei dischi, rappresentati dal loro diametro,
                       // presenti in ciascun piolo.
/**
 * creazione di una torre contente n dischi sul piolo 0
 * @param n numero di dischi
```

```
 */
public HanoiTower(int n) {

  ndischi = n;

  numDischiPiolo = new int[3];
  numDischiPiolo[0]=n;
  numDischiPiolo[1]=numDischiPiolo[2]=0;

  dischiPiolo = new int[3][n];

  for(int i =0;i<n;i++) dischiPiolo[0][i]=n-i; // inserimento dischi nel piolo 0
  for(int i =0;i<n;i++) dischiPiolo[1][i]=dischiPiolo[2][i]=0;
}

// metodi

/**
 * muove tutti i dischi dal piolo 0 al piolo 2
 */
public void esegui() {
  muoviTorre(ndischi,0,2,1);
}

/**
 * muove ndischi da un piolo ad un altro usando il terzo
 * come piolo d'appoggio
 * @param ndischi il numero di dischi da muovere
 * @param da piolo in cui vi sono i dischi
 * @param a piolo dove portare i dischi
 * @param usando piolo d'appoggio
 */
public void muoviTorre(int ndischi, int da, int a, int usando) {

  if(ndischi > 0) {
    muoviTorre(ndischi-1,da,usando,a);
    muoviDisco(da,a);
    muoviTorre(ndischi-1,usando,a,da);
  }
}

/**
 * muove un disco da un piolo ad un altro
 * @param da piolo da cui prendere il disco
 * @param a piolo dove portare il disco
 */
public void muoviDisco(int da, int a) {

  numDischiPiolo[da] -= 1;
  dischiPiolo[a][numDischiPiolo[a]]=dischiPiolo[da][numDischiPiolo[da]];
  dischiPiolo[da][numDischiPiolo[da]] = 0;
  numDischiPiolo[a] += 1;

  System.out.println(da + " -> " + a);
  stampaConfig();
}


/**
 * visualizza la configurazione della torre
 */
public void stampaConfig() {
  int riga,nspazi,nstar,piolo;
  int i,j;
```

```
// stampa indici di piolo
  for(j=0;j<3;j++) {
    for(i=0;i<ndischi;i++) System.out.print(" ");
    System.out.print(j);
    for(i=0;i<ndischi;i++) System.out.print(" ");
    }
  System.out.println();

// stampa i dischi
  for(riga=1;riga<=ndischi;riga++) {
    for(piolo=0;piolo<3;piolo++) {
      nstar = dischiPiolo[piolo][ndischi-riga];
      nspazi = ndischi - nstar;
      for(i=0;i<nspazi;i++) System.out.print(" ");
      for(i=0;i<nstar;i++) System.out.print("*");
      System.out.print("|");
      for(i=0;i<nstar;i++) System.out.print("*");
      for(i=0;i<nspazi;i++) System.out.print(" ");
      }
    System.out.println();
    }
}

/**
 * calcola il numero di mosse necessario a muovere una
 * torre complete
 */
public BigInteger numeromosse(){

// chiama calcolamosse convertendo il il valore del campo ndischi
// a valore di tipo BigInteger
 return calcolamosse(new BigInteger(String.valueOf(ndischi)));
}

static final BigInteger biguno = new BigInteger("1");
static final BigInteger bigdue = new BigInteger("2");
BigInteger calcolamosse(BigInteger bign){
  if (bign.equals(biguno)) return biguno;
  else return bigdue.multiply(calcolamosse(bign.subtract(biguno))).add(biguno);
}

/**
 * metodo principale
 */
public static void main(String[] args){
  HanoiTower torre;
  int ndisk;

  if (args.length > 0)
    ndisk = Integer.valueOf(args[0],10).intValue();
  else
    ndisk=3;

  torre = new HanoiTower(ndisk);
  torre.esegui();
  System.out.print("Numero di mosse: ");
  System.out.println(torre.numeromosse());

 // calcolo numero mosse per una torre di 64 elementi
  System.out.print("Numero di mosse per 64 elementi: ");
  System.out.println((new HanoiTower(64)).numeromosse());
}
```

```
} // end of class HanoiTower
```

Attivando il programma senza alcun argomento, si ottiene la seguente uscita:

```
0 -> 2
   0      1      2
   |      |      |
 **|**    |      |
***|***   |     *|*
0 -> 1
   0      1      2
   |      |      |
   |      |      |
***|*** **|**   *|*
2 -> 1
   0      1      2
   |      |      |
   |     *|*     |
***|*** **|**    |
0 -> 2
   0      1      2
   |      |      |
   |     *|*     |
   |    **|** ***|***
1 -> 0
   0      1      2
   |      |      |
   |      |      |
  *|*   **|** ***|***
1 -> 2
   0      1      2
   |      |      |
   |      |    **|**
  *|*     |   ***|***
0 -> 2
   0      1      2
   |      |     *|*
   |      |    **|**
   |      |   ***|***
Numero di mosse: 7
Numero di mosse per 64 elementi: 18446744073709551615
```

5.5 Fai da te

1. Nel caso del calcolo degli anagrammi definito nella classe Anagrammi per calcolare un nuovo anagramma si devono trovare tutti quelli precedenti. Si modifichi il programma in maniera tale che non si abbia questa inefficienza, ovvero che non si debba ogni volta calcolare tutti gli anagrammi precedenti.

2. Si sviluppi un programma che risolva il problema delle otto regine. Su una scacchiera 8x8 si deve poszionare otto regine in maniera tale che, secondo le regole del gioco degli scacchi, nessuna regina possa mangiare un'altra. Si estenda anche al caso di N regine in una scacchiera NxN.

Capitolo 6

Ricerca ed ordinamento

6.1 Ricerca su array

Listato 6.1 - SearchArray.java - ricerca su array di interi.

```
public class SearchArray {
/*
 * private constructor. non si puo' istanziare\
 */
private SearchArray(){}

/**
 * ricerca lineare. dati un array di interi ed un valore
 * ritorna l'indice della locazione in cui si trova
 * tale valore. Ritorna -1 se non esiste.
 */

  public static int linearSearch(int[] a,int value)
  {
    int i;

    for(i=0;i<a.length;i++)
      if(value==a[i]) break; // esci dal ciclo se trovato

    if (i==a.length) return -1;
    else return i;
  }

/**
 * ricerca binaria. dati un array ordinato decrescente di interi ed un valore
 * ritorna l'indice della locazione in cui si trova
 * tale valore. Ritorna -1 se non esiste.
 */

  public static int binarySearch(int[] a,int value)
  {
    int i,j,k;

    i=-1; j=0; k=a.length-1;
    while (j<=k)
    {
     i=(j+k)/2;
     if(value==a[i]) break; // esci dal ciclo se trovato
     else
       if (value<a[i]) k=i-1;
       else j=i+1;
    }
    if (j>k) return -1;
```

```
    else return i;
  }

  public static void main (String[]args)
  {
    int[] a ={ 2, -8, 23, 90, 33, 25, 37, 89, -45, 27};
    int[] b ={ -45, -8, 2, 23, 25, 27, 33, 37, 89, 90};

    System.out.println(a);
    System.out.println(linearSearch(a,33));
    System.out.println(linearSearch(a,100));
    System.out.println(binarySearch(b,33));
    System.out.println(binarySearch(b,100));
  }
}
```

6.2 Ordinamento di array

Listato 6.2 - SortArray.java - ordinamento di un array di interi.

```
public class SortArray {
/*
 * private constructor. non si puo' istanziare\
 */
private SortArray(){}

/*
 ordinamento lineare/selezione

 questo metodo e' basato sul seguente principio :

 1. seleziona l'elemento con la chiave piu' piccola
 2. scambialo con il primo elemento a1

 quindi si ripete queste operazioni con i rimanenti n-1 elementi,
 poi con n-2, finche' rimane solamente un elemento, il piu' grande.

*/
public static void linearSort(int[] dati)
  {
    int i,j,k;
    int min;
    int x;

    for (i=0;i<dati.length-1;i++) {

     // ricerca minimo
     k=i;
     min=dati[i];
     for (j=i+1;j<dati.length;j++)
       if (dati[j] < min) {
         k=j;
         min=dati[j];
         }

     // esegui lo scambio
     x = dati[k];
     dati[k] = dati[i];
     dati[i] = x;
     }
```

```
}

/*
  ordinamento per inserimento/tabellare

 gli elementi da ordinare sono concettualmente divisi in una sequenza
 di destinazione Ai... Ai-1 e una sequenza d'ingresso A1 ... An

 Ad ogni passo, partendo da i=2 ed incrementando i di uno, il i-esimo elemento
 della sequenza d'ingresso viene messo nella sequenza di destinazione
 inserendolo nel posto appropriato.

 Per trovare il posto d'inserimento, si alterna tra test e spostamenti, cioe'
 si compare x con Aj e o si inserisce x oppure si muove Aj a destra e si
 procede verso sinistra. Vi sono due distinte condizioni che causano la fine
 di questo ciclo :
 1. si e' trovato un elemento Aj con una chiave minore della chiave di X
 2. si e' raggiunto l'elemento piu' a sinistra nella sequenza di destinazione

*/
  public static void insertSort(int[] dati)
  {
    int i,j;
    int x;

    for (i=1;i<dati.length;i++) {
      x = dati[i];
      j = i-1;
      while (j>=0 && x < dati[j]) {
        dati[j+1] = dati[j];
        j = j-1;
        }
      dati[j+1] = x;
    }
  }

/*
 ordinamento per scambio diretto(bubble sort)

 compara ed eventualmente scambia coppie adiacenti
 finche' tutti gli elementi sono ordinati
*/
public static void bubbleSort(int dati[])
  {
    int i,j;
    int x;

    for(i=1;i<dati.length-1;i++)
      for(j=dati.length-1;j>=i;j--)
        if (dati[j-1] > dati[j]){
          x = dati[j-1]; dati[j-1]=dati[j]; dati[j] = x;
          }
}

/*
 ordinamento per scambio (shake sort)

simile al bubblesort
si alternano passate dal basso verso l'alto con passate dall'alto verso il basso

*/
public static void shakeSort(int[] dati)
{
    int j,k,sup,inf;
```

```
    int x;

    sup=1; inf=k=dati.length-1;

    do {
      // passata dal basso verso l'alto
      for(j=inf;j>=sup;j--)
      if (dati[j-1] > dati[j]){
        x = dati[j-1]; dati[j-1]=dati[j]; dati[j] = x;
        k=j;
        }
      sup = k+1;
      // passata dall'alto verso il basso
      for(j=sup;j<=inf;j++)
      if (dati[j-1] > dati[j]){
        x = dati[j-1]; dati[j-1]=dati[j]; dati[j] = x;
        k=j;
        }
      inf = k-1;
      }       while(sup<=inf);

}

/*
 ordinamento per partizione (quick sort)


*/
public static void quickSort(int[] dati)
{
  partitionSort(dati,0,dati.length-1);
}

static void partitionSort(int[] dati, int l,int r)
{
    int i,j;
    int med; // valore mediano
    int temp;

    i=l; j=r;
    med = dati[(l+r) / 2];
    do {
      while (dati[i] < med) i = i+1;
      while (dati[j] > med) j = j-1;
      if (i<=j) {
        temp = dati[i]; dati[i]=dati[j]; dati[j]=temp;
        i = i+1; j = j-1;
        }
      }       while(i<=j);
    if (l<j) partitionSort(dati,l,j);
    if (i<r) partitionSort(dati,i,r);
}


public static void stampa(int[] a)
{
  for(int i=0;i<a.length;i++)
    System.out.print(a[i]+", ");
  System.out.println();
}

public static void copia(int[] a,int[] b)
{
  for(int i=0;i<a.length;i++)
```

```
    a[i]=b[i];
}

  public static void main (String[]args)
  {
    int[] a ={ 5, -8, 23, 90, 33, 25, 37, 89, -45, 27};
    int[] b = new int[a.length];

    stampa(a);
    copia(b,a);linearSort(b); stampa(b);
    copia(b,a); insertSort(b);  stampa(b);
    copia(b,a); bubbleSort(b); stampa(b);
    copia(b,a); shakeSort(b); stampa(b);
    copia(b,a); quickSort(b); stampa(b);
  }
}
```

6.3 Ordinamento di array generico

Listato 6.3 - SortAbstract.java - ordinamento di array generico.

```
abstract class Data{
abstract boolean gt(Data dato);
abstract void print();
}

class DataInt extends Data {
int key;
int info;

 DataInt(int k,int i)
{
 key=k;
info=i;
}

boolean gt(Data dato){
DataInt altrodato = (DataInt) dato;
return (key>altrodato.key);
}

void print(){
System.out.println(key+" "+info);
}

}

class DataString extends Data {
String key;
String info;

 DataString(String k,String i)
{
 key=k;
info=i;
}

boolean gt(Data dato){
DataString altrodato = (DataString) dato;
return (key.compareTo(altrodato.key)>0);
```

```
}

void print(){
System.out.println(key+" "+info);
}

}

public class SortAbstract{

int numDati=0;
Data[] dati;

public SortAbstract(int n){
dati = new Data[n];
}



public void insert(Data dato){
dati[numDati]=dato;
numDati=numDati+1;
}

public  void linearSort()
  {
    int i,j,k;
    Data min,temp;

    for (i=0;i<numDati;i++) {

     // ricerca minimo
     k=i;
     min=dati[i];
     for (j=i+1;j<numDati;j++)
       if (min.gt(dati[j])) {
         k=j;
         min=dati[j];
         }

     // esegui lo scambio
     temp = dati[k]; dati[k] = dati[i]; dati[i] = temp;
     }
}

public void print(){

for(int i=0;i<numDati;i++) dati[i].print();
System.out.println();
}

public static void main(String[] argv){
int i,j;
SortAbstract daint = new SortAbstract(10);
SortAbstract dastr = new SortAbstract(10);

// inserisce i numeri da 100 a 1
for(i=0,j=100;i<10;i++,j--) daint.insert(new DataInt(j,3*j));
daint.print();
daint.linearSort();
daint.print();

for(i=0,j=99;i<10;i++,j--) dastr.insert(new DataString("ABC"+j,"Info"+3*j));
dastr.print();
```

```
dastr.linearSort();
dastr.print();
}
}
```

Capitolo 7

Strutture dinamiche

7.1 Liste su interi

7.1.1 Nodo della lista

Listato 7.1 - Nodo.java - definizione di elemento della lista

```
public class Nodo {
  Nodo next;
  int valore;

  Nodo(int val)
  {
    valore=val; next=null;
  }

  public int getValore()
  {
    return valore;
  }

  public Nodo getNext()
  {
    return next;
  }

  public void setNext(Nodo nuovo)
  {
    next = nuovo;
  }
}
```

7.1.2 Lista FirstInFirstOut

Listato 7.2 - FIFOint.java - lista First In First Out

```
public class FIFOint {
  Nodo primo;

  public FIFOint()
  {
    primo=null;
  }
```

```
  public void inserisci(int valore)
  {
    Nodo nuovo,ultimo;

    nuovo = new Nodo(valore);
    if (primo == null) primo = nuovo;
    else { // ricerca fine della lista
      ultimo=primo;
      while (ultimo.getNext() != null) ultimo =ultimo.getNext();
      ultimo.setNext(nuovo);
      }
  }

  public int preleva()
  {
     int val=0;

     if (primo==null) throw new IllegalAccessError("coda vuota");
     val = primo.getValore();
     primo = primo.getNext();
     return val;
  }

  public static void main(String[] args)
  {
     FIFOint lista=new FIFOint();

     lista.inserisci(10);
     lista.inserisci(20);
     lista.inserisci(30);
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
  }
}
```

7.1.3 Lista FirstInFirstOut ottimizzata

Listato 7.3 - FIFOintOpt.java - lista First In First Out ottimizzata

```
public class FIFOintOpt extends FIFOint {
  Nodo ultimo;

  public FIFOintOpt()
  {
    super();
    ultimo=null;
  }

  public void inserisci(int valore)
  {
    Nodo nuovo;

    nuovo = new Nodo(valore);
    if (primo == null) primo = nuovo;
    else ultimo.setNext(nuovo);
    ultimo = nuovo;
  }

  public int preleva()
```

```
  {
     int val;


     val = super.preleva();
     if (primo == null) ultimo = null;
     return val;
  }

  public static void main(String[] args)
  {
     FIFOintOpt lista=new FIFOintOpt();

     lista.inserisci(10);
     lista.inserisci(20);
     lista.inserisci(30);
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
  }
}
```

7.1.4 Lista LastInFirstOut

Listato 7.4 - LIFOint.java - lista Last In First Out

```
public class LIFOint extends FIFOint{

  public void inserisci(int valore)
  {
    Nodo nuovo;

    nuovo = new Nodo(valore);
    nuovo.setNext(primo);
    primo = nuovo;
  }

  public static void main(String[] args)
  {
     LIFOint lista=new LIFOint();

     lista.inserisci(10);
     lista.inserisci(20);
     lista.inserisci(30);
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
     System.out.println(lista.preleva());
  }
}
```

Capitolo 8

Files e Input/Output.

8.1 Gestione dei file.

Listato 8.1 - FileTest.java: gestione del file system esterno.

```
/**
esempio di utilizzo della classe File per la gestione dei file esterni

il programma verifica la presenza di una directory. Se e' presente la cancella
con i file in essa contenuti, altrimenti crea tale directory con dieci file e
rinomina uno dei file o con un metodo diretto o con una copia e cancellazione
All'inizio stampa i valori di alcune proprieta' del sistema.

*/
import java.io.*;

class FileTest {

/**
 genera la copia di un file
*/
static void copyFile(File from, File to){
int ch;

try {
FileInputStream ffrom = new FileInputStream(from);
FileOutputStream fto = new FileOutputStream(to);

while ((ch = ffrom.read())!=-1) fto.write(ch);
ffrom.close();
fto.close();
} catch (IOException e){
 System.out.println("error in copy"+e);
 }
}

public static void main(){
File fdir;
File fdat;
FileOutputStream fout;
String[] filelist;

// stampa alcune proprieta' del sistema
System.out.println("file.separator: "+System.getProperty("file.separator"));
System.out.println("java.version: "+System.getProperty("java.version"));
System.out.println("path.separator: "+System.getProperty("path.separator"));
System.out.println("user.name: "+System.getProperty("user.name"));
System.out.println("user.home: "+System.getProperty("user.home"));
```

```
System.out.println("user.dir: "+System.getProperty("user.dir"));

// file dir1 nella directory corrente
fdir= new File(System.getProperty("user.dir"),"dir1");

// vede se la dir1 esiste
if(fdir.exists()){
  // cancellazione della directory
  // non viene cancellata se contiene dei file
     if(!fdir.delete()){
       // lista i file contenuti nella directory
       filelist = fdir.list();

       // ciclo di cancellazione dei file
       for(int i=0;i<filelist.length;i++) {
         System.out.println("deleting "+filelist[i]);
        // creazione di un oggetto File associato all'i-esimo nome
         fdat = new File(fdir,filelist[i]);
         fdat.delete(); // cancellazione del file
         }
      fdir.delete(); // cancellazione della directory
      }
   }

else {
// crea directory
System.out.println("created dir: "+fdir.mkdir());

  // creazione di dieci file nella directory
  for(int i=0;i<10;i++) {
    // creazione di un oggetto File per indicare il nome e path
    fdat = new File(fdir,"data"+i);
    try{
      fout = new FileOutputStream(fdat);
      fout.close();
      } catch (IOException e) {
        System.out.println(fdat.getName()+" IOException" + e);
     }
  // rename del file data0 in data99
  if(i==0) {
        File newfile = new File(fdir,"data99");
        fdat = new File(fdir,"data0");
        if(fdat.renameTo(newfile))
           System.out.println("file renamed");
        else {
           FileTest.copyFile(fdat,newfile);
          fdat.delete();
          System.out.println("file copied");
         }
    }
   }
}
}
}
```

8.2 Lettura file di tipo ASCII.

Listato 8.2 - AsciiReader.java: lettura file ASCII.

```
/**
classe che implementa
```

```
*/
import java.io.*;

class AsciiReader extends FilterInputStream{
int ch=’ ’;
char nlchar=’\n’;

 AsciiReader(InputStream in){
 super(in);
 nlchar = System.getProperty("line.separator").charAt(0);
}

public String readString()
 throws IOException{
 StringBuffer temp= new StringBuffer();


while (ch!=-1 && (ch == ’ ’ || ch == nlchar)) ch=read() ; // salta spazi

if (ch==-1) return(null);
temp.append((char)ch);
while ((ch=read())!=-1 && ch != ’ ’ && ch!=nlchar) temp.append((char)ch); // salta spazi
return temp.toString();
}

public int readInt() throws IOException {
//String st =this.readString();
//System.out.println("’"+st+"’");
//return 0;
return Integer.parseInt(this.readString());
}

public static void main(){
String st;
try {
AsciiReader ar = new AsciiReader(new FileInputStream("test"));
while((st=ar.readString())!=null) {
 System.out.print(st);
 System.out.print(ar.readInt());
 System.out.println();
}
} catch (IOException e) {}
}
}
```

8.3 Lettura mediante StreamTokenizer.

Listato 8.3 - TokenTest.java: uso del StreamTokenizer.

```
/**
Esempio d’uso della classe StreamTokenizer
*/
import java.io. *;

class TokenTest
{
  public static final char strDelimiter = ’"’;

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

```
    try
    {
      Reader r = new BufferedReader (new FileReader ("test"));
      StreamTokenizer st = new StreamTokenizer (r);

        st.quoteChar (strDelimiter);    // definisce carattere delimitatore delle stringhe

        st.commentChar ('#');     // definsce carattere per commentare singole linee

      while (st.nextToken () != StreamTokenizer.TT_EOF)
        {
          switch (st.ttype)
            {
            case StreamTokenizer.TT_WORD:
              System.out.println ("Word: " + st.sval);
              break;
              case StreamTokenizer.TT_NUMBER:
              System.out.println ("Number: " + st.nval);
              break;
              case strDelimiter:
              System.out.println ("String: " + st.sval);
              break;
              default:
              System.out.println ("Special char: " + (char) st.ttype);
            }

        }
    }
    catch (IOException e)
    {
    }
  }
}
```

8.4 Archiviazione in formato ZIP.

Listato 8.4 - Zip.java: generazione di un archivio.

```
/**

Zip

memorizza tutto il contenuto della directory di lavoro, comprese le sottodirectory.

java Zip [nome_archivio]

se non e' presente il nome_archivio salva in archive.zip

*/
import java.io. *;
import java.util. *;
import java.util.zip. *;


class Zip
{

  static byte[] buffer = new byte[1024];
  static String sep;
```

```
static byte lsep;

static void storeEntry (ZipOutputStream zipf, InputStream entryfile)
throws Exception
{
  int len;

  while ((len = entryfile.read (buffer)) != -1)
      zipf.write (buffer, 0, len);
    entryfile.close ();
}

static void storeDir (ZipOutputStream zipf, String[]filelist, String path)
throws Exception
{
  int i;
  File fin;
    String[] subfilelist;
  ZipEntry entry;
  String fname;

  for (i = 0; i < filelist.length; i++)
    {
      if (filelist[i].equalsIgnoreCase ("ZipStore"))
        continue;

      fname = path + filelist[i];
      System.out.println (fname);
      fin = new File (fname);
      if (fin.isDirectory ())
        {
          entry = new ZipEntry (fname + sep);
          zipf.putNextEntry (entry);
          zipf.closeEntry ();
          subfilelist = fin.list ();
          storeDir (zipf, subfilelist, path + filelist[i] + sep);
        }
      else
        {
          entry = new ZipEntry (fname);
          zipf.putNextEntry (entry);
          storeEntry (zipf, new FileInputStream (fin));
          zipf.closeEntry ();
        }
    }

}

public static void main (String argv[])
{
  File fdir;
  File fdat;
    String[] filelist;
  ZipOutputStream zipf;
  String actualdir;
  String zipname = "archive.zip";
  int i;

  for (i = 0; i < argv.length; i++)
    {
        zipname = argv[i];
    }

  fdat = new File (zipname);
```

```
    if (fdat.exists ())
      fdat.delete ();


    actualdir = System.getProperty ("user.dir");
    System.out.println ("actual directory: " + actualdir);
    sep = System.getProperty ("file.separator");
    System.out.println ("file.separator: " + sep);
    lsep = (byte) System.getProperty ("line.separator").charAt (0);
    System.out.println ("line.separator: " + lsep);

    try
    {

      fdir = new File (System.getProperty ("user.dir"));
      filelist = fdir.list ();
      zipf = new ZipOutputStream (new FileOutputStream (zipname));
      storeDir (zipf, filelist, "");
      zipf.close ();
    }
    catch (Exception e)
    {
      System.out.println ("Entry Exception " + e);
    }
    System.out.println("DONE");
  }
}
```

Listato 8.5 - UnZip.java: estrazione di un archivio.

```
/**

UnZip

programma per unzippare un archivio

unzip [-x] [nome_archivio]

se presente il flag -x estrae i file dall'archivio,
altrimenti visulaizza solamente la lista di tali file

se e' presente nome_archivio opera su tale archivio,
segnalando errore nel caso non esistesse,
altrimenti opera su tutti gli eventuali file con estensione .zip
che si trovano nella directory di lavoro

converte fine linea dei file di testo individuati analiizando
il primo buffer letto

converte inoltre il separtore usato nei pathname

*/
import java.io. *;
import java.util. *;
import java.util.zip. *;


class UnZip
{
```

```
  static byte[] buffer = new byte[1024];
  static String fsep;
  static byte lsep;

/**
analizza il buffer per vedere se di tipo text
ritorna falso se vi sono caratteri <32 e non sono
\n, \t \r \f
*/

  static boolean isText (int len)
  {
    if (len<=0) return false;
    for (int i=0;i<len;i++)
     if ((buffer[i]<32)&&!((buffer[i]==(byte)'\n')||(buffer[i]==(byte)'\t')||
(buffer[i]==(byte)'\r')||(buffer[i]==(byte)'\f'))) return false;
    return true;
  }


  static void saveEntry (InputStream entryfile, FileOutputStream fout)
  throws Exception
  {
    int len;
boolean textfile;

len = entryfile.read (buffer);
if (textfile = isText(len))
 System.out.print(" (text)");

    while (len != -1)
      {
        if (textfile)
          for (int i = 0; i < len; i++)
            if ((buffer[i] == 13) || (buffer[i] == 10))
              buffer[i] = lsep;
        fout.write (buffer, 0, len);
len = entryfile.read (buffer);
      }
    fout.close ();
      entryfile.close ();
  }



  public static void main (String argv[])
  {
    File fdir;
    File fdat;
      String[] filelist;
      String[] zipfiles;
    ZipFile zipf = null;
    String zipname = null;
    String topdir;
    Enumeration listentry;
    ZipEntry entry;
    String entryname;
    int i;
    int zipidx;
    boolean extract = false;


   // analizza argomenti
    for (i = 0; i < argv.length; i++)
```

```
  {
    if (argv[i].compareTo ("-x") == 0)
      extract = true;
    else
      zipname = argv[i];
  }

// estrae proprieta del sistema
fsep = System.getProperty ("file.separator");
System.out.println ("file.separator: " + fsep);
lsep = (byte) System.getProperty ("line.separator").charAt (0);
System.out.println ("line.separator: " + lsep);
topdir = System.getProperty ("user.dir");
System.out.println ("actual directory: " + topdir);

if (zipname != null)
  {
    zipfiles = new String[1];
    zipfiles[0] = zipname;
  }
else
  {
    fdir = new File (topdir);
    zipfiles = fdir.list ();
  }

for (zipidx = 0; zipidx < zipfiles.length; zipidx++)
  if (zipfiles[zipidx].endsWith (".zip"))
    {
      System.out.println ();
      System.out.println ("Zip file: " + zipfiles[zipidx]);
      try
      {
        zipf = new ZipFile (zipfiles[zipidx]);
      }
      catch (Exception e)
      {
        System.out.println ("Exception " + e);
        System.exit (1);
      }

      listentry = zipf.entries ();

      while (listentry.hasMoreElements ())
        {
          entry = (ZipEntry) listentry.nextElement ();
          entryname = entry.getName ();
          entryname = entryname.replace ('\\', '/');
          entryname = entryname.replace ('\\', fsep.charAt (0));

          System.out.print ("Entry: " + entryname);
          if (extract)
            try
            {
              fdat = new File (entryname);


              if (entry.isDirectory ())
                fdir = fdat;
              else if (fdat.getParent () != null)
                fdir = new File (fdat.getParent ());
              else
                fdir = null;
              if ((fdir != null) && (!fdir.exists ()))
```

```
                  fdir.mkdirs ();
                if (!entry.isDirectory ())
                  {
                    saveEntry (zipf.getInputStream (entry), new FileOutputStream (fdat));
                  }
              }
            catch (Exception e)
            {
              System.out.println ("Entry Exception " + e);
              continue;
            }
System.out.println();
          }
        }
    System.out.println ("DONE");
  }
}
```

Appendice A

Prestazioni

Usualmente il codice generato da un compilatore Java viene eseguito da un interprete software che realizza la macchina virtuale per il linguaggio Java. Per aumentare l'efficienza dell'esecuzione si possono seguire due strade:

- compilatori JIT
  i compilatori Just-In-Time vengono utilizzati dalla macchina virtuale per compilare al volo durante il caricamento il codice java nel linguaggio macchina del calcolatore su cui funziona la macchina virtuale
- chips Java
  questi chips sono delle cpu che esegueno il codice java

Un semplice programma che permette di analizzare le prestazioni ottenibili utilizzando il linguaggio Java, è mostrato nel seguente listato:

Listato A.1 - Benchmark.java - programma di test in Java

```
/**
 * implementa la classe Benchmark che definisce un insieme di
 * algoritmi di benchmark
 *
 * @author P.Bison Copyright 1997
 */

public class Benchmark
{
// intTest constants
  public static final int INT_LOOP = 2000000;
// floattest constants
  public static final float CONST1 = 3.141597E0f;
  public static final float CONST2 = 1.7839032e4f;
  public static final int FLOAT_LOOP = 4000000;
// fiboTest constants
  public static final int FIBO_LOOP = 100;
  public static final int FIBO_NUM = 24;
// sieveTest constants
  public static final int SIEVE_SIZE = 8190;
  public static final int SIEVE_LOOP = 1000;
// savageTest constants
  public static final int SAVAGE_LOOP = 200000;


/**
 *  test delle quattro operazioni sugli interi
 */
  public static long intTest ()
  {
    long i=10000 , iter;
```

```
    for (iter = 1; iter <= INT_LOOP; iter++)
      {
        i = -3 + (5 * i) / 5 + 3;

      }
    return i;
  }


/**
 * test sulle quattro operazioni sui reali (float)
 */
  public static float floatTest ()
  {
    float a=0f, b=0f, c=0f, d=0f;
    int i;

    for (i = 1; i <= FLOAT_LOOP; ++i)
      {
        a = CONST1 * CONST2 * (float)i;
        b = CONST1 / CONST2 / (float)i;
        c = CONST1 + CONST2 + (float)i;
        d = CONST1 - CONST2 - (float)i;
      }
  return a+b+c+d;
  }

/**
 * calcolo di numeri primi con il crivello di Erastotene
 */
  public static boolean sieveTest ()
  {
    int i, prime, k, count, iter;
      boolean[] flags = new boolean[SIEVE_SIZE + 1];


    for (iter = 1; iter <= SIEVE_LOOP; iter++)
      {
        count = 0;
        for (i = 0; i <= SIEVE_SIZE; i++)
          flags[i] = true;
        for (i = 0; i <= SIEVE_SIZE; i++)
          {
            if (flags[i])
              {
                prime = i + i + 3;
                for (k = i + prime; k <= SIEVE_SIZE; k += prime)
                  flags[k] = false;
                count++;
              }
          }
      }
   return flags[SIEVE_SIZE];

  }


/**
 * calcolo dei numeri di Fibonacci in maniera ricorsiva
 */

  public static long fib (long n)
  {
    if (n > 2)
```

```
      return (fib (n - 1) + fib (n - 2));
    else
      return (1);
  }

  public static long fiboTest (long n)
  {
    long ris = 0;
    int i;
    for (i = 1; i <= FIBO_LOOP; i++)
      {
        ris = fib (n);
      }
    return ris;
  }


/*
 * test funzioni trigonometriche
 */
  public static double savageTest ()
  {
    int i;
    double a;

      a = 1.0;
    for (i = 1; i <= SAVAGE_LOOP; i++)
        a = Math.tan (Math.atan (Math.exp (Math.log (Math.sqrt (a * a)))));
      return a;

  }

  public static void main (String[]args)
  {
    long t1, t0, maint0, maint1;
    long a;
    float b;
    long c;
    boolean d;
    double e;

      maint0 = System.currentTimeMillis ();
      System.out.println ("Java Benchmarks");
      System.out.println ();

      System.out.println ("Integer");
      t0 = System.currentTimeMillis ();
      a=intTest ();
      t1 = System.currentTimeMillis ();
      System.out.println (a);
      System.out.println ("Time: " + (t1 - t0) + " millisec.");
      System.out.println ();

      System.out.println ("Float");
      t0 = System.currentTimeMillis ();
      b=floatTest ();
      t1 = System.currentTimeMillis ();
      System.out.println (b);
      System.out.println ("Time: " + (t1 - t0) + " millisec.");
      System.out.println ();

      System.out.println ("Fibonacci");
      t0 = System.currentTimeMillis ();
      c = fiboTest (FIBO_NUM);
```

```
      t1 = System.currentTimeMillis ();
      System.out.println (c);
      System.out.println ("Time: " + (t1 - t0) + " millisec.");
      System.out.println ();

      System.out.println ("Sieve");
      t0 = System.currentTimeMillis ();
      d=sieveTest ();
      t1 = System.currentTimeMillis ();
      System.out.println (d);
      System.out.println ("Time: " + (t1 - t0) + " millisec.");
      System.out.println ();

      System.out.println ("Savage");
      t0 = System.currentTimeMillis ();
      e = savageTest ();
      t1 = System.currentTimeMillis ();
      System.out.println (e);
      System.out.println ("Time: " + (t1 - t0) + " millisec.");
      System.out.println ();
      maint1 = System.currentTimeMillis ();
      System.out.println ("Total time: " + (maint1 - maint0) + " millisec.");
  }

}// end of class Benchmark
```

Il programma implementa cinque tests:

- calcolo su valori interi
- calcolo su valori reali
- attivazione di metodo/procedura attraverso l'esecuzione della funzione per il calcolo dei numeri di Fibonacci
- il crivello di Erastotene
- calcolo di funzioni trascendenti

e calcola i tempi di esecuzione dei singoli test e del programma complessivo.

Il corrispondente programma in C è:

Listato A.2 - Benchmark.c - programma di test in C

```
/*
   *
 */
#include <stdio.h>
#include <time.h>

#define TRUE 1
#define FALSE 0

#define INT_LOOP 2000000

#define CONST1 3.141597E0
#define CONST2 1.7839032e4
#define FLOAT_LOOP 4000000

#define FIBO_LOOP 100
#define FIBO_NUM 24

#define SIEVE_SIZE 8190
```

```
#define SIEVE_LOOP 1000

#define SAVAGE_LOOP 200000

long
intTest ()
{
  long i = 10000, iter;


  for (iter = 1; iter <= INT_LOOP; iter++)
    {
      i = -3 + (5 * i) / 5 + 3;

    }
  return i;
}

/**
 * test sulle quattro operazioni sui reali (float)
 */
float
floatTest ()
{
  float a=0.0, b=0.0, c=0.0, d=0.0;
  int i;

    for (i = 1; i <= FLOAT_LOOP; ++i)
      {
        a = CONST1 * CONST2 * (float)i;
        b = CONST1 / CONST2 / (float)i;
        c = CONST1 + CONST2 + (float)i;
        d = CONST1 - CONST2 - (float)i;
      }
  return a+b+c+d;
  }


long
fib (long x)
{
  if (x > 2)
    return (fib (x - 1) + fib (x - 2));
  else
    return (1);
}

long
fiboTest (long n)
{
  long ris;
  int i;
  for (i = 1; i <= FIBO_LOOP; i++)
    {
      ris = fib (n);
    }
  return ris;
}


char
sieveTest ()
{
  int i, prime, k, count, iter;
```

```
  char flags[SIEVE_SIZE + 1] =
  {0};

  for (iter = 1; iter <= SIEVE_LOOP; iter++)
    {
      count = 0;
      for (i = 0; i <= SIEVE_SIZE; i++)
        flags[i] = TRUE;
      for (i = 0; i <= SIEVE_SIZE; i++)
        {
          if (flags[i])
            {
              prime = i + i + 3;
              for (k = i + prime; k <= SIEVE_SIZE; k += prime)
                flags[k] = FALSE;
              count++;
            }
        }
    }
 return flags[SIEVE_SIZE];
}


extern double tan (), atan (), exp (), log (), sqrt ();

double
savageTest ()
{
  int i;
  double a;


  a = 1.0;
  for (i = 1; i <= SAVAGE_LOOP; i++)
    a = tan (atan (exp (log (sqrt (a * a)))));

  return (a);
}

int
main ()
{

  clock_t start, stop, mainstart, mainstop;
  long a;
  float b;
  long c;
  char d;
  double e;

  mainstart = clock ();
  printf ("C benchmark\n\n");
  printf ("Integer\n");
  start = clock ();
  a=intTest ();
  stop = clock ();
  printf ("%d\nTime: %d millisec\n\n",a, (int) (((float) (stop - start)) / CLOCKS_PER_SEC * 1000.0));

  printf ("Float\n");
  start = clock ();
  b=floatTest ();
  stop = clock ();
  printf ("%e\nTime: %d millisec\n\n", b,(int) (((float) (stop - start)) / CLOCKS_PER_SEC * 1000.0));
```

```
  printf ("Fibonacci\n");
  start = clock ();
  c = fiboTest (FIBO_NUM);
  stop = clock ();
  printf ("%d\nTime: %d millisec\n\n", c, (int) (((float) (stop - start)) / CLOCKS_PER_SEC * 1000.0))

  printf ("Sieve\n");
  start = clock ();
  d=sieveTest ();
  stop = clock ();
  printf ("%d\nTime: %d millisec\n\n",d, (int) (((float) (stop - start)) / CLOCKS_PER_SEC * 1000.0));

  printf ("Savage\n");
  start = clock ();
  e = savageTest ();
  stop = clock ();
  printf ("%f\nTime: %d millisec\n\n", e, (int) (((float) (stop - start)) / CLOCKS_PER_SEC * 1000.0))
  mainstop = clock ();
  printf ("Total time: %d millisec\n", (int) (((float) (mainstop - mainstart)) / CLOCKS_PER_SEC * 100

  return (0);

}
```

I tempi di esecuzione, espressi in millisecondi, riportati in tabella sono relativi ad una workstation Sun ULTRA1 creator. Per la compilazione ed esecuzione del programma Java si è utilizzato il package Solari SPARC Edition JDK/JIT v1.1.3 con i comandi

```
javac Benchmark.java
java Benchmark
```

mentre per il programma C il compilatore GNU gcc versione 2.7.1 con i comandi:

```
gcc -o Benchmark Benchmark.c
Benchmark
```

Nel caso del programma Java vengono riportati i tempi sia con il compilatore Just-In-Time che senza.

test	Java		C
	w JIT	w/o JIT	
Integer	1551	4689	170
Float	1887	8438	2119
Fibonacci	1836	11840	1669
Sieve	2326	29897	4219
Savage	1144	1652	1029
Totale	8782	56540	9210

Come si puó notare dalla tabella, utilizzando un compilatore Just-in-Time si ha un notevole aumento delle prestazioni. Nel particolare caso di questo programma si è ottenuta una diminuzione del tempo di esecuzione per ogni singolo test, che nel caso del crivello di Erastotene (Sieve) è pari al 92%.

Confrontando con il programma C si nota che, nel caso JTC, tre test (Float, Fibonacci, Savage) vengono eseguiti in tempi paragonabili, mentre gli altri due sono contradditori. Nel caso Integer il programma in C surclassa l'equivalente Java essendo quest'ultimo 9.12 volte piú lento. D'altro canto il programma Java ha prestazioni migliori nel caso Sieve in cui è 1.81 volte piú veloce.

Ovviamente questo esempio da solo una stima parziale delle effettive prestazioni. Innumerevoli fattori possono concorrere alla determinazione dei tempi di esecuzione (efficienza del compilatore, carico della macchina) per cui si devono utilizzare test piú complessi se si vuole avere un quadro piú completo.

Appendice B

Risorse al Ladseb

In questa appendice vengono elencati le risorse disponibili localmente per la programmazione in Java

B.1 Macintosh

Per il Macintosh sono disponibili:

MRJ 2.x sistema runtime per Java è disponibile nell'ultima release del sistema operativo (8.1)

MRJ SDK 2.x sistema di sviluppo per Java comprendente il compilatore

tutorial1.1 un tutorial on line in formato HTML

Queste risorse sono disponibili la prima nel CDrom del sistema operativo, le altre come file .sea nel folder Java in Public.

B.2 Sun450

Il software per lo sviluppo e' installato in `/opt/java/jdkx.x` dove `x.x` è la versione. Per utilizzarlo si deve metter nella propria path la directory `/opt/java/jdkx.x/bin`.
È possibile leggere la documentazione aprendo con netscape (la versione 3 si trova in `/usr/local/netscape`) il file `/opt/java/jdk1.1.6/docs/index.html`.
È presente anche il tutorial in `/opt/java/tutorial`.

Bibliografia

[1] K.Arnold, J.Gosling. *The Java™ Programming Language*. Addison-Wesley, Reading, Massachuttes, 1996.

[2] J.Gosling, B.Joy, G.Steele. *The Java™ Language Specification*. Addison-Wesley, Reading, Massachuttes, 1996.

[3] T.Lindholm, F.Yellin. *The Java™ Virtual Machine Specification*. Addison-Wesley, Reading, Massachuttes, 1996.

[4] J.Gosling, F.Yellin, The Java team. *The Java™ Application Programming Interface, Volume 1*. Addison-Wesley, Reading, Massachuttes, 1996.

[5] J.Gosling, F.Yellin, The Java team. *The Java™ Application Programming Interface, Volume 2*. Addison-Wesley, Reading, Massachuttes, 1996.

[6] M.Campione, K.Walrath . *The Java™ Tutorial*. Addison-Wesley, Reading, Massachuttes, 1996. (`http://java.sun.com/tutorial/`)

Indice delle figure

Indice dei listati