Capitolo 4

Tipi di dati fondamentali

Cay S. Horstmann
Concetti di informatica e fondamenti di Java
quarta edizione

Obiettivi del capitolo

- Apprendere l'utilizzo di numeri interi e di numeri in virgola mobile
- Identificare i limiti dei tipi numerici
- Acquisire consapevolezza degli errori di trabocco e di arrotondamento
- Apprendere il corretto utilizzo delle costanti
- Scrivere espressioni aritmetiche in Java
- Usare il tipo `String` per definire e manipolare sequenze di caratteri
- Imparare ad acquisire dati in ingresso e a presentare dati impaginati in uscita

Tipi di numeri

- `int`: numeri interi, no parti frazionarie

```
1, -4, 0
```

- `double`: numeri in virgola mobile

```
0.5, -3.11111, 4.3E24, 1E-14
```

Tipi numerici

- Un calcolo numerico trabocca se il risultato esce dall'intervallo del tipo numerico.

```
int n = 1000000;
System.out.println(n * n); // visualizza -727379968
```

- Java ha otto tipi primitivi, fra i quali quattro tipi interi e due tipi in virgola mobile.

Tipi primitivi

Tipo	Descrizione	Dimensione
`int`	Tipo intero con intervallo –2147483648...2147483647 (circa 2 miliardi)	4 byte
`byte`	Tipo che descrive un singolo byte, con intervallo –128...127	1 byte
`short`	Tipo intero "corto", con intervallo –32768...32767	2 byte
`long`	Tipo intero "lungo", con intervallo –9223372036854775808... 9223372036854775807	8 byte
`double`	Tipo in virgola mobile a doppia precisione, con intervallo circa $\pm 10^{308}$ e circa 15 cifre decimali significative	8 byte
`float`	Tipo in virgola mobile a singola precisione, con intervallo circa $\pm 10^{38}$ e circa 7 cifre decimali significative	4 byte
`char`	Tipo che rappresenta caratteri codificati secondo lo schema Unicode (Argomenti avanzati 4.5)	2 byte
`boolean`	Tipo per i due valori logici `true` e `false` (Capitolo 6)	1 bit

Tipi numerici: virgola mobile

- Avvengono errori di arrotondamento quando non si può fare una conversione numerica esatta.

```
double f = 4.35;
System.out.println(100 * f); // visualizza 434.99999999999994
```

- In Java non si può assegnare un valore in virgola mobile a una variabile intera:

```
double balance = 13.75;
int dollars = balance; // Errore
```

Continua

Tipi numerici: virgola mobile

- Cast (forzatura): converte in numero intero il valore in virgola mobile

```
int dollars = (int) balance;
```

Il cast (`int`) ignora la parte frazionaria.

- Il metodo `Math.round` arrotonda un numero in virgola mobile all'intero più vicino

```
long rounded = Math.round(balance);
```

Se `balance` vale 13.75, allora a `rounded` viene assegnato il valore 14.

Sintassi 4.1: Cast

```
(nomeTipo) espressione
```

Esempio:

```
(int) (balance * 100)
```

Obiettivo:

Convertire un'espressione in un tipo di dati diverso.

Numeri grandi e precisione

- classi BigInteger (numeri interi grandi) e BigDecimal (numeri frazionari grandi) in java.math
  - Pro: Altissima “capienza” e precisione
  - Contro: Lentezza. Bisogna usare metodi per eseguire operazioni (add, subtract e multiply)

```
BigInteger n = new BigInteger(“1000000”);
BigInteger r = n.multiply(n);
System.out.println(r); // Visualizza 1000000000000
```

```
BigDecimal d = new BigDecimal(“4.35”);
BigDecimal e = new BigDecimal(“100”);
BigDecimal f = d.multiply(e);
System.out.println(f); // Visualizza 435.00
```

Costanti

- Una variabile `final` è una costante
  - dopo che le è stato assegnato un valore, non può più essere modificato.
- Dare un nome alle costanti rende i programmi più facili da leggere e da modificare.
  - Non usare numeri “magici”! Introdurre una costante
- Convenzione: per le costanti utilizzare nomi con tutte le lettere maiuscole.

```
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;
payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
   + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
```

Costanti

- Spesso le costanti vengono usate in più metodi, per cui è necessario dichiararle insieme ai campi di esemplare della classe, identificandole come `static`, oltre che `final`.
- Metodi di altre classi possono accedere a una costante pubblica indicando il nome della classe in cui essa è definita, seguito da un punto e dal nome della costante stessa. Esempio `Math.PI`.

La classe Math della libreria standard definisce un paio di utili costanti:

```
public class Math
{
   . . .
   public static final double E = 2.7182818284590452354;
   public static final double PI = 3.14159265358979323846;
}
```

Sintassi 4.2: Definizione di costante

Come variabile locale di un metodo o come attributo di un oggetto:

```
final nomeTipo nomeVariabile = espressione;
```

In una classe:

```
specificatoreDiAccesso static final nomeTipo nomeVariabile
                                = espressione;
```

Esempio:

```
final double NICKEL_VALUE = 0.05;
public static final double LITERS_PER_GALLON = 3.785;
```

Obiettivo:

Definire una costante come variabile locale di un metodo, o come attributo di un oggetto, o come variabile di classe.

File CashRegister.java

```
01: /**
02:    Un registratore di cassa somma gli articoli venduti
    e calcola il resto dovuto al cliente.
03: */
04: public class CashRegister
05: {
06:    /**
07:       Costruisce un registratore di cassa senza soldi
       nel cassetto.
08:    */
09:    public CashRegister()
10:    {
11:       purchase = 0;
12:       payment = 0;
13:    }
14:
```

Continua

File CashRegister.java

```
15:    /**
16:       Registra la vendita di un articolo.
17:       @param amount il prezzo dell'articolo
18:    */
19:    public void recordPurchase(double amount)
20:    {
21:       purchase = purchase + amount;
22:    }
23:
24:    /**
25:       Registra la quantità di denaro ricevuta per il pagamento.
26:       @param dollars il numero di dollari pagati
27:       @param quarters il numero di quarti di dollaro pagati
28:       @param dimes il numero di decimi di dollaro pagati
29:       @param nickels il numero di nickel di dollaro pagati
30:       @param pennies il numero di penny di dollaro pagati
31:    */
```


Continua

File CashRegister.java

```
32:    public void enterPayment(int dollars, int quarters,
33:          int dimes, int nickels, int pennies)
34:    {
35:       payment = dollars + quarters * QUARTER_VALUE
                + dimes * DIME_VALUE
36:             + nickels * NICKEL_VALUE + pennies
                * PENNY_VALUE;
37:    }
38:
39:    /**
40:       Calcola il resto dovuto al cliente e azzera
          la macchina per la vendita successiva.
41:       @return il resto dovuto al cliente
42:    */
```

Continua

File CashRegister.java

```
43:     public double giveChange()
44:     {
45:        double change = payment - purchase;
46:        purchase = 0;
47:        payment = 0;
48:        return change;
49:     }
50:
51:     public static final double QUARTER_VALUE = 0.25;
52:     public static final double DIME_VALUE = 0.1;
53:     public static final double NICKEL_VALUE = 0.05;
54:     public static final double PENNY_VALUE = 0.01;
55:     // campi di esemplare
56:     private double purchase;
57:     private double payment;
58: }
```

File CashRegisterTester.java

```
01: /**
02:    Classe che collauda la classe CashRegister.
03: */
04: public class CashRegisterTester
05: {
06:    public static void main(String[] args)
07:    {
08:       CashRegister register = new CashRegister();
09:
10:       register.recordPurchase(0.75);
11:       register.recordPurchase(1.50);
12:       register.enterPayment(2, 0, 5, 0, 0);
13:       System.out.print("Change=");
14:       System.out.println(register.giveChange());
15:
```

Continua

File CashRegisterTester.java

```
16:         register.recordPurchase(2.25);
17:         register.recordPurchase(19.25);
18:         register.enterPayment(23, 2, 0, 0, 0);
19:         System.out.print("Change=");
20:         System.out.println(register.giveChange());
21:     }
22: }
```

Visualizza

```
Change=0.25
Change=2.0
```

Assegnazione, incremento e decremento

- L'operatore = è detto operatore di *assegnazione*
  - alla sua sinistra ci deve essere il nome di una variabile, mentre alla destra ci può essere un singolo valore o un'espressione
  - imposta la variabile al valore dato
- L'assegnazione di un valore a una variabile non è un'uguaglianza matematica.
- Gli operatori ++ e – – incrementano e decrementano una variabile di un'unità.
  - `items++` ha lo stesso effetto di `items = items + 1`
  - `items--` sottrae 1 a `items`

Assegnazione, incremento e decremento

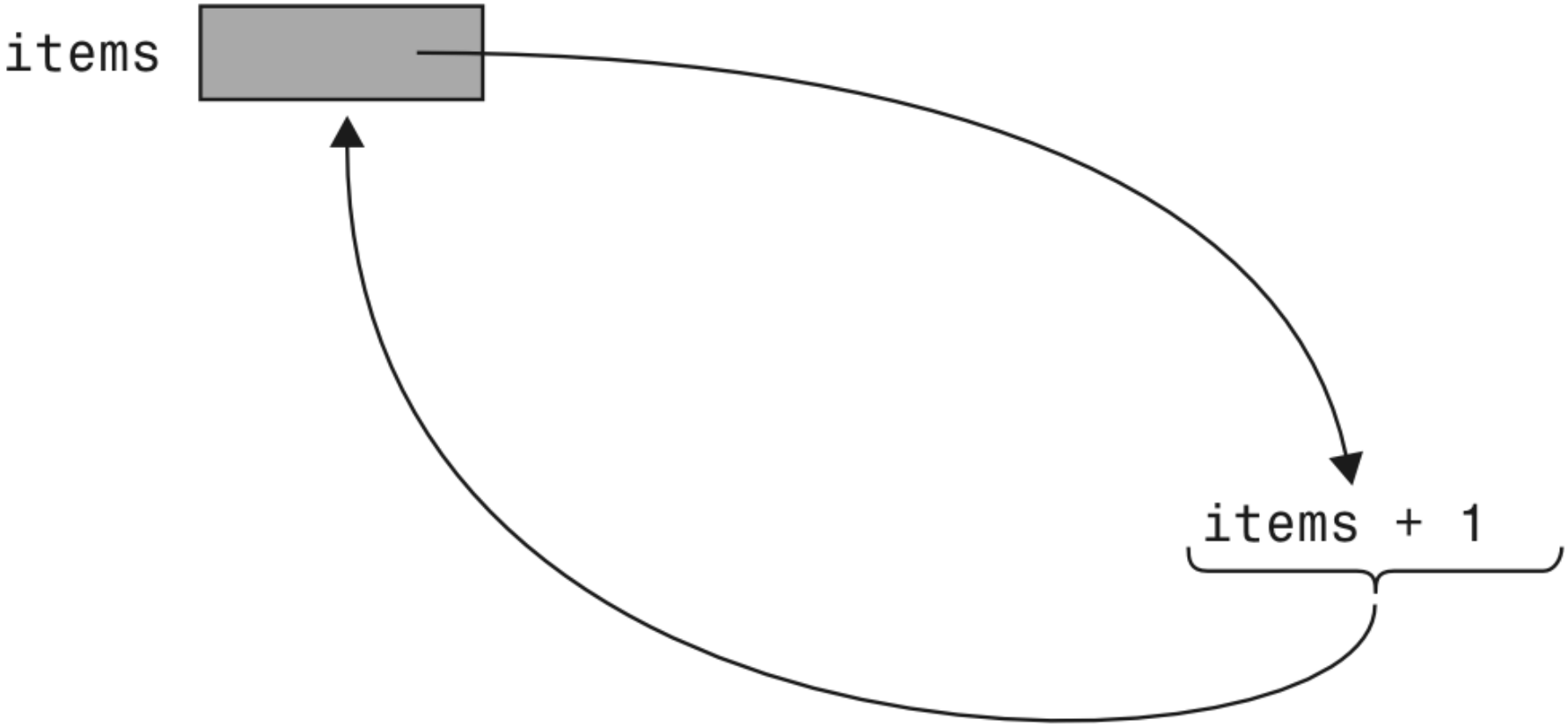


Figura 1 Incrementare una variabile

Operazioni aritmetiche

- `/` è l'operatore per la divisione
- Se entrambi gli argomenti dell'operatore `/` sono di tipo intero, il risultato è un numero intero e il resto viene ignorato.
  - `7.0 / 4` restituisce `1.75`
    `7 / 4` restituisce `1`
- L'operatore `%` calcola il resto di una divisione.
  - `7 % 4` è `3`

Divisione tra interi

- Errore comune

```
int s1 = 5; // punteggio della prova 1
int s2 = 6; // punteggio della prova 2
int s3 = 3; // punteggio della prova 3

double averageErr = (s1 + s2 + s3) / 3; // Errore

double averageOk1 = (s1 + s2 + s3) / 3.0; // OK

double sum = s1 + s2 + s3;
double averageOk2 = sum / 3; // OK
```

Operazioni aritmetiche

Un tipico utilizzo della divisione intera / e di operazioni con il resto %

```
final int PENNIES_PER_NICKEL = 5;
final int PENNIES_PER_DIME = 10;
final int PENNIES_PER_QUARTER = 25;
final int PENNIES_PER_DOLLAR = 100;

// Calcola il valore totale in centesimi
int total = dollars * PENNIES_PER_DOLLAR +
            quarters * PENNIES_PER_QUARTER +
            nickels * PENNIES_PER_NICKEL +
            dimes * PENNIES_PER_DIME + pennies;


// Usa la divisione intera per convertire in dollari e centesimi
int dollars = total / PENNIES_PER_DOLLAR;
int cents = total % PENNIES_PER_DOLLAR;
```

La classe `Math`

- La classe `Math` contiene i metodi `sqrt` e `pow` per calcolare radici quadrate e potenze.
- Per calcolare x^n, si usa il metodo `Math.pow(x, n)`
  - tuttavia, per calcolare x^2 è molto più efficiente scrivere semplicemente `x * x`
- Per estrarre la radice quadrata di un numero, si usa il metodo `Math.sqrt` scrivendo `Math.sqrt(x)`
- In Java, $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$ diventa

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

Metodi matematici in Java

`Math.sqrt(x)`	radice quadrata di x (≥ 0)
`Math.pow(x, y)`	x^y ($x > 0$, o $x = 0$ e $y > 0$, o $x < 0$ e y è un intero)
`Math.sin(x)`	seno di x (x in radianti)
`Math.cos(x)`	coseno di x
`Math.tan(x)`	tangente di x
`Math.asin(x)`	arcoseno ($\sin^{-1} x \in [-\pi/2, \pi/2]$, $x \in [-1,1]$)
`Math.acos(x)`	arcocoseno ($\cos^{-1} x \in [0, \pi]$, $x \in [-1,1]$)
`Math.atan(x)`	arcotangente ($\tan^{-1} x \in (-\pi/2, \pi/2)$)
`Math.atan2(x, y)`	arcotangente ($\tan^{-1} (y/x) \in [-\pi, \pi]$, x può essere 0)
`Math.toRadians(x)`	converte x da gradi a radianti (cioè restituisce $x \cdot \pi/180$)
`Math.toDegrees(x)`	converte x da radianti a gradi (cioè restituisce $x \cdot 180/\pi$)
`Math.exp(x)`	e^x
`Math.log(x)`	logaritmo naturale ($\log(x)$, $x > 0$)
`Math.round(x)`	il numero intero (di tipo `long`) più prossimo a x
`Math.ceil(x)`	il più piccolo numero intero (di tipo `double`) che sia $\geq x$
`Math.floor(x)`	il più grande numero intero (di tipo `double`) che sia $\leq x$
`Math.abs(x)`	valore assoluto, $\|x\|$
`Math.max(x, y)`	il numero maggiore tra x e y
`Math.min(x, y)`	il numero minore tra x e y

Analisi di un'espressione

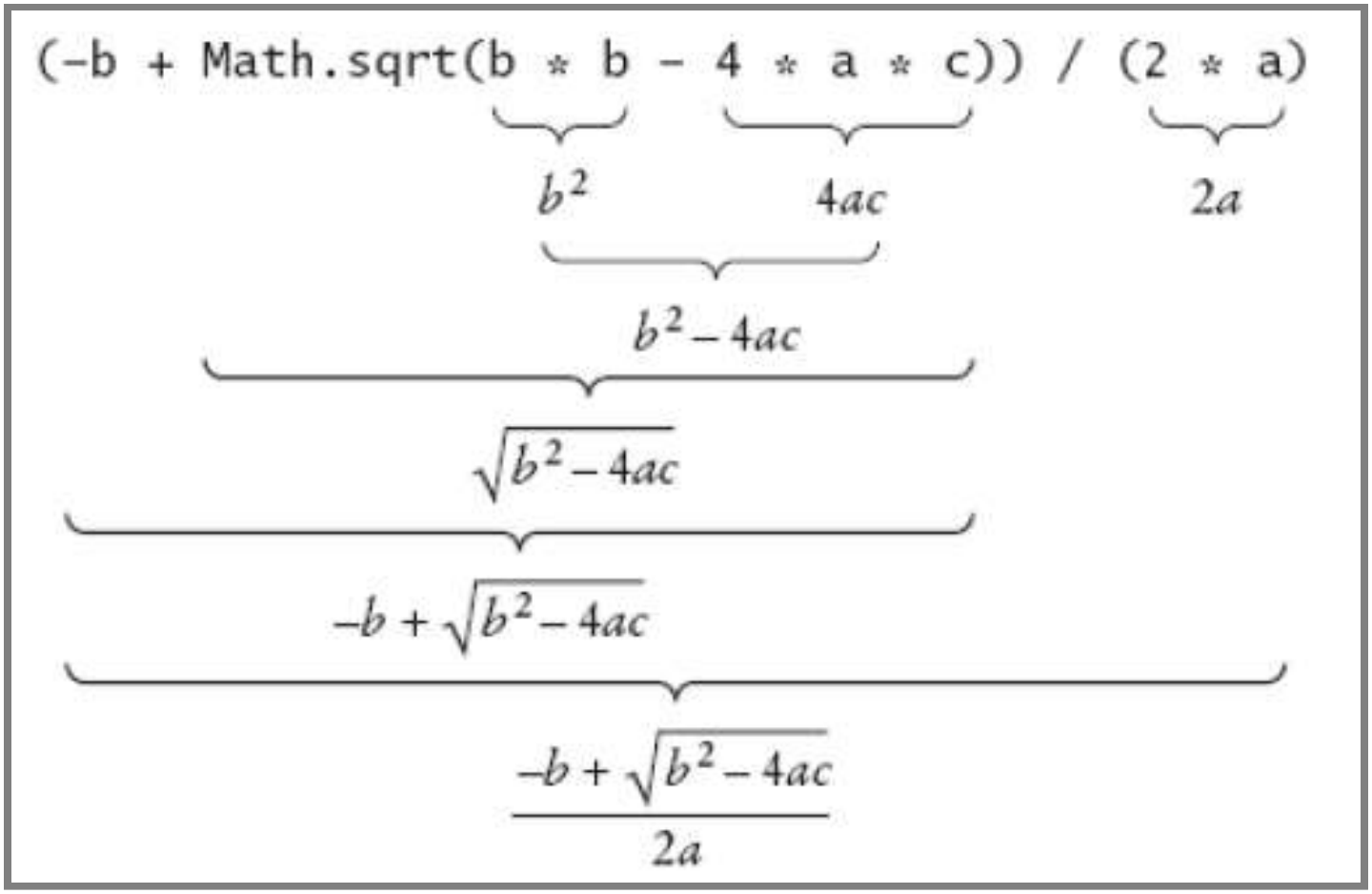


Figura 3 Analisi di un'espressione

Invocare metodi statici

- Un metodo `static` non agisce su un oggetto.
  NON si può scrivere

```
double x = 4;
double root = x.sqrt(); // Errore
4.sqrt(); // Errore
```

- I metodi statici sono sempre definiti all'interno di classi
- Per convenzione: i nomi delle classi iniziano con una lettera maiuscola, quelli di oggetti e metodi con una lettera minuscola (le invocazioni di metodi sono seguite da parentesi)

```
Math
System.out
```

Sintassi 4.3: Invocazione di un metodo statico

```
NomeClasse.nomeMetodo(parametri)
```

Esempio:

```
Math.sqrt(4)
```

Obiettivo:

Invocare un metodo statico (un metodo che non agisce su un oggetto) e fornire i suoi parametri.

Stringhe

- Una stringa è una sequenza di caratteri

```
"Hello, World!"
```

- Le stringhe sono esemplari della classe `String`
- Per calcolare la lunghezza di una stringa:

```
int n = message.length();
```

- Stringa vuota:

```
""
```

Concatenazione

- Usando l'operatore + si possono concatenare più stringhe:

```
String name = "Dave";
String message = "Hello, " + name;
   // message è "Hello, Dave"
```

- Ogni volta che uno degli argomenti dell'operatore + è una stringa, l'altro argomento viene convertito in una stringa.

```
String a = "Agent";
int n = 7;
String bond = a + n; // bond è "Agent7"
```

Concatenazione negli enunciati print

- La concatenazione è molto utile per ridurre il numero degli enunciati `System.out.print`.
- Per esempio,

```
System.out.print("The total is ");
System.out.println(total);
```

può essere combinato nella singola invocazione:

```
System.out.println("The total is " + total);
```

Conversione tra stringhe e numeri

- Se una stringa contiene le cifre di un numero, viene usato il metodo `Integer.parseInt` o `Double.parseDouble` per ottenere il valore numerico.
- Cosa succede se la stringa non contiene un valore corrispondete al tipo usato?

```
int parsedInt1 = Integer.parseInt("45.2");
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string:
"45.2"
at java.lang.NumberFormatException.
        forInputString(NumberFormatException.java:48)
at java.lang.Integer.parseInt(Integer.java:458)
at java.lang.Integer.parseInt(Integer.java:499)
at capitolo4.NumericParsingTester.main(NumericParsingTester.java:14)
```

Eccezioni e studenti

- Reazione tipica: Non funziona! Il programma è morto! Non so che fare!
- Il messaggio d'errore contiene informazioni importanti.
- L'eccezione
  - java.lang.NumberFormatException: For input string: "45.2"
- File e riga dove viene lanciata l'eccezione
  - at capitolo4.NumericParsingTester.
    main(NumericParsingTester.java:14)
- Cercare la prima linea del **vostro codice** che appare nella segnalazione di eccezione.

substring

- Per estrarre una parte di una stringa, viene usato il metodo `substring`.

```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5); // sub è "Hello"
```

- La prima posizione della stringa è contrassegnata dall'indice 0, la seconda da 1 e così via.
- Le posizioni dei caratteri in una stringa si contano a partire da zero.

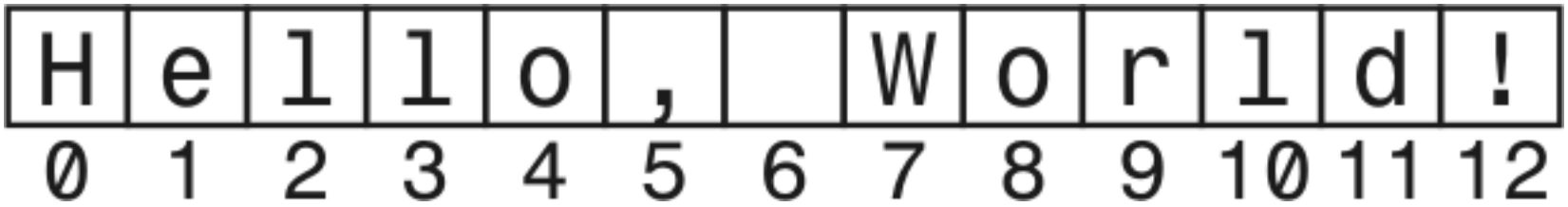


Figura 3 Posizioni di una stringa

Continua

substring

- È possibile calcolare la lunghezza di una sottostringa mediante l'operazione `pastEnd - start`
  - per esempio, la stringa “World” ha lunghezza 12 – 7 = 5.

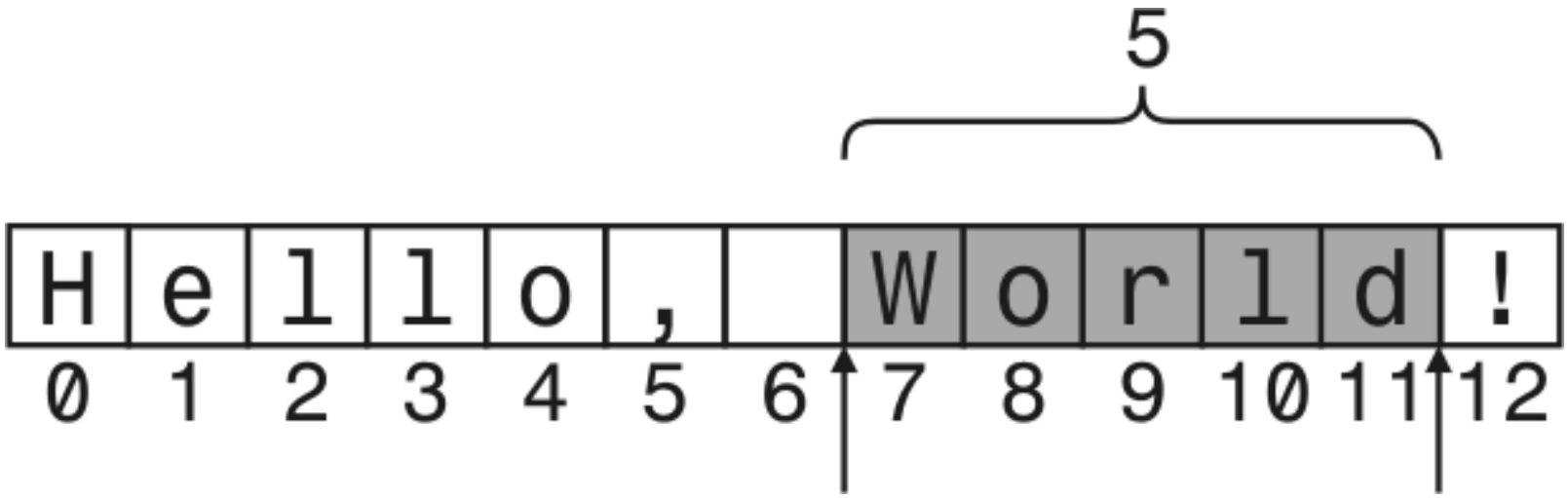


Figura 4 Estrazione di una sottostringa

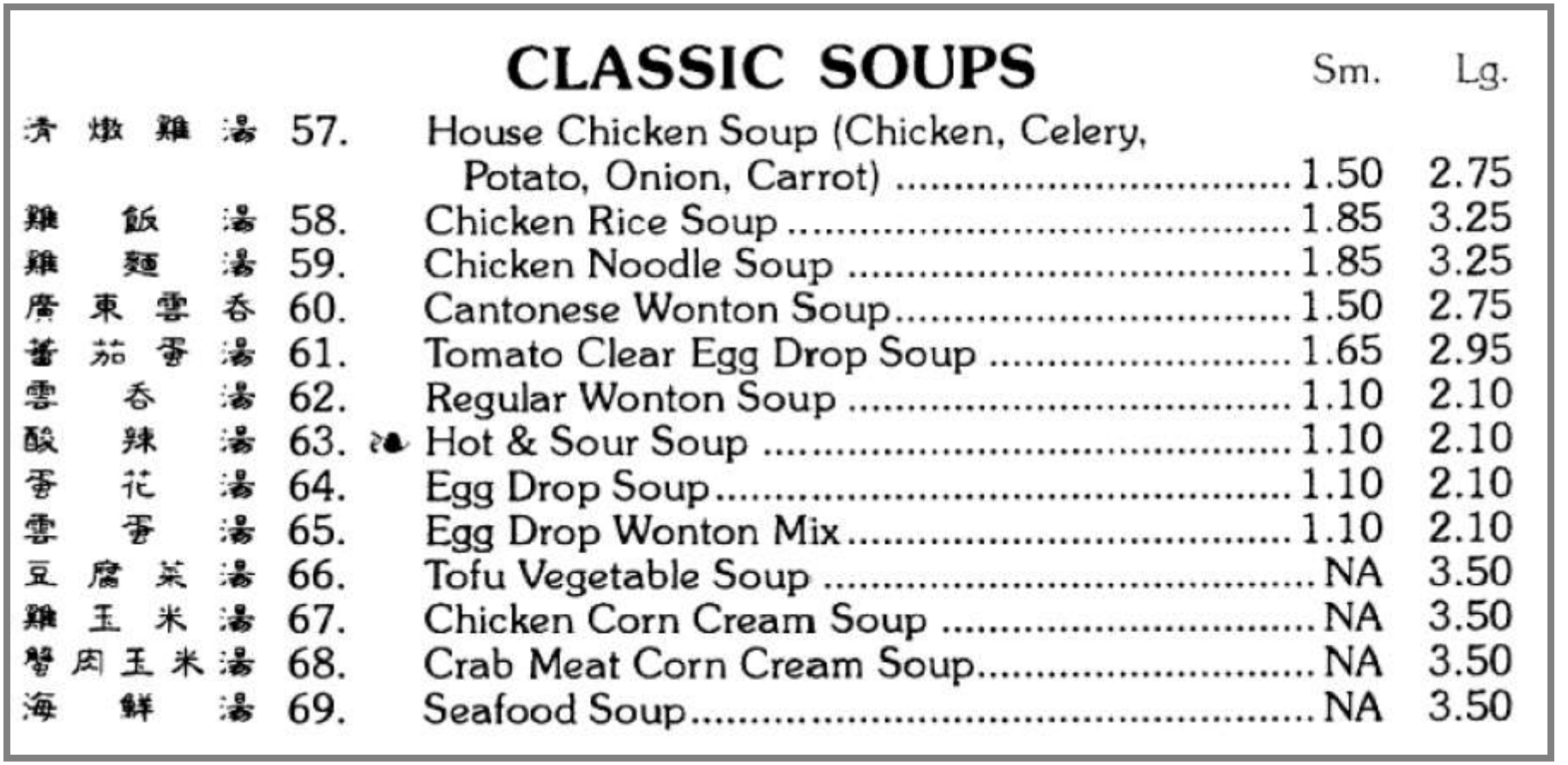
Alfabeti

Figura 5 Una tastiera tedesca

Alfabeti

Figura 6 L'alfabeto thai

Alfabeti



CLASSIC SOUPS

			Sm.	Lg.
清燉雞湯	57.	House Chicken Soup (Chicken, Celery, Potato, Onion, Carrot)	1.50	2.75
雞飯湯	58.	Chicken Rice Soup	1.85	3.25
雞麵湯	59.	Chicken Noodle Soup	1.85	3.25
廣東雲吞	60.	Cantonese Wonton Soup	1.50	2.75
蕃茄雪湯	61.	Tomato Clear Egg Drop Soup	1.65	2.95
雲吞湯	62.	Regular Wonton Soup	1.10	2.10
酸辣湯	63.	Hot & Sour Soup	1.10	2.10
雪花湯	64.	Egg Drop Soup	1.10	2.10
雲雪湯	65.	Egg Drop Wonton Mix	1.10	2.10
豆腐菜湯	66.	Tofu Vegetable Soup	NA	3.50
雞玉米湯	67.	Chicken Corn Cream Soup	NA	3.50
蟹肉玉米湯	68.	Crab Meat Corn Cream Soup	NA	3.50
海鮮湯	69.	Seafood Soup	NA	3.50

Figura 7 Menu con caratteri cinesi

Leggere dati in ingresso

- All'oggetto `System.in` è associato un insieme di funzionalità minimali: può leggere solo un byte alla volta
- In Java 5.0, è stata aggiunta una classe `Scanner` che consente la lettura semplice e comoda dei dati inseriti in ingresso da tastiera.

```
Scanner in = new Scanner(System.in);
System.out.print("Enter quantity: ");
int quantity = in.nextInt();
```

- Quando viene invocato il metodo `nextInt` o `nextDouble`, il programma si arresta in attesa che l'utente digiti un numero e prema il tasto "Enter" ("Invio"):
- Il metodo `nextLine` restituisce la successiva riga di testo fornita in ingresso (fino alla pressione del tasto "Enter"), sotto forma di oggetto di tipo `String`.
- Il metodo `next` restituisce, invece, la parola successiva, una sequenza di caratteri terminata da un carattere di spaziatura ("white space")

Leggere dati in ingresso

- Gli oggetti di tipo Scanner hanno metodi per verificare il tipo del dato che sta per essere letto
  - public boolean hasNextInt()
  - public boolean hasNextLong()
  - public boolean hasNextLine()
  - ....

File CashRegisterSimulator.java

```
01: import java.util.Scanner;
02:
03: /**
04:    Programma che simula una transazione in cui un utente paga
05: per un articolo e riceve il resto
06: */
07: public class CashRegisterSimulator
08: {
09:    public static void main(String[] args)
10:    {
11:       Scanner in = new Scanner(System.in);
12:
13:       CashRegister register = new CashRegister();
14:
15:       System.out.print("Enter price: ");
16:       double price = in.nextDouble();
17:       register.recordPurchase(price);
18:
19:       System.out.print("Enter dollars: ");
20:       int dollars = in.nextInt();
```

Continua

File CashRegisterSimulator.java

```
21:          int dollars = in.nextInt();
22:          System.out.print("Enter quarters: ");
23:          int quarters = in.nextInt();
24:          System.out.print("Enter dimes: ");
25:          int dimes = in.nextInt();
26:          System.out.print("Enter nickels: ");
27:          int nickels = in.nextInt();
28:          System.out.print("Enter pennies: ");
29:          int pennies = in.nextInt();
30:          register.enterPayment(dollars, quarters, dimes, nickels,
                pennies);
31:
32:          System.out.print("Your change: ");
33:          System.out.println(register.giveChange());
34:       }
35:    }
```

Continua

File CashRegisterSimulator.java

Visualizza

```
Enter price: 7.55
Enter dollars: 10
Enter quarters: 2
Enter dimes: 1
Enter nickels: 0
Enter pennies: 0
Your change is 3.05
```

Lettura di dati da una finestra di dialogo

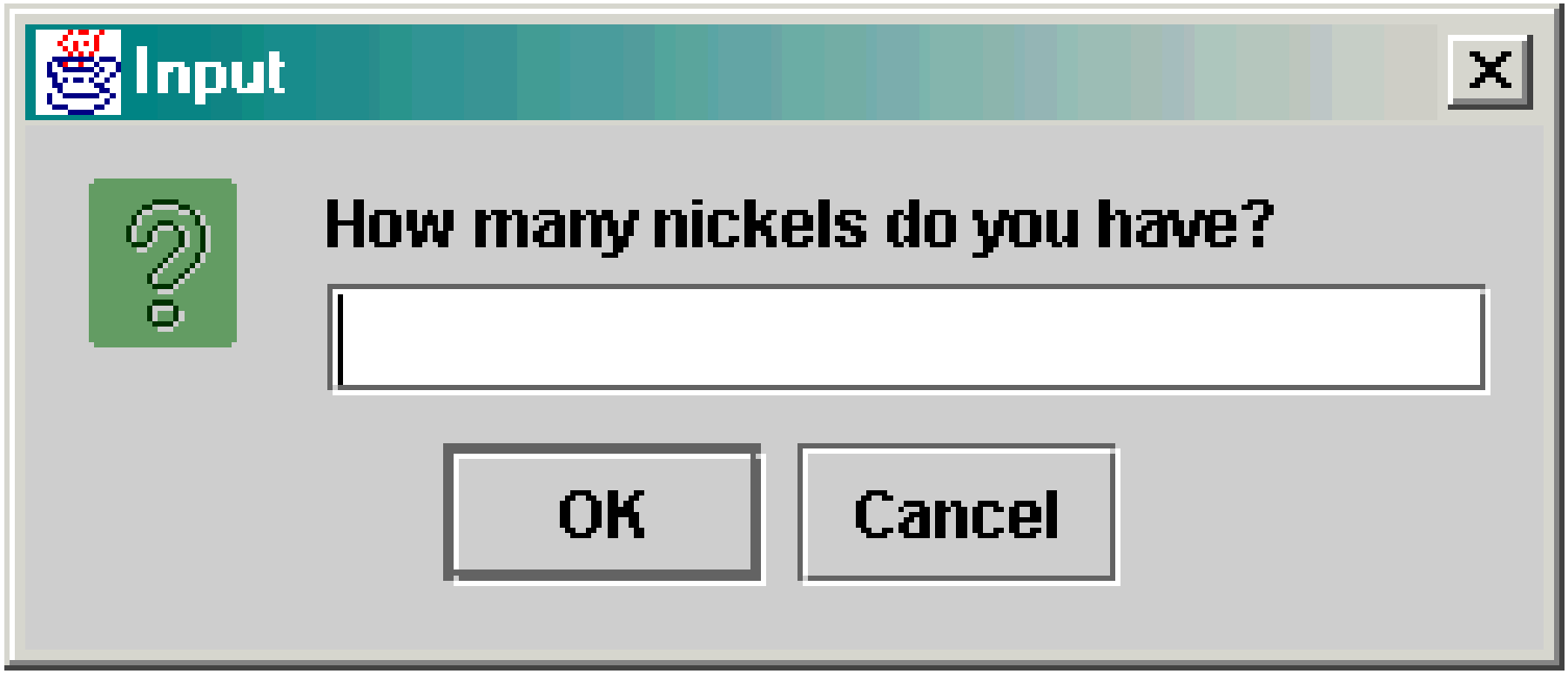


Figura 8 Una finestra di dialogo

Lettura di dati da una finestra di dialogo

- ```
  String input = JOptionPane.showInputDialog
  ("How many nickels do you have?")
  ```

- Convertire le stringhe in numeri se necessario:

```
int nickels = Integer.parseInt(input);
```

- Va aggiunto `System.exit(0)` alla fine del metodo `main` ogniqualvolta venga invocato `showInputDialog` o `showMessageDialog`
```