8 - Stringhe
e altre classi dalla Libreria Standard (Java API)

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://pages.di.unipi.it/milazzo
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2018/2019

Introduzione

Abbiamo visto come creare classi che includono metodi diversi

- In particolare, metodi ausiliari che risolvono sottoproblemi

Vedremo più avanti (nella seconda parte del corso) come realizzare programmi che consistono di più classi che si dividono il lavoro

- I metodi di una classe potranno invocare i metodi di un'altra

Per il momento vediamo come utilizare classi già fatte

- La Libreria Standard di Java
  - Java API (Application Programming Interface)
- Una enorme collezione di classi già implementate!

La Libreria Standard di Java (1)

Le classi della Libreria Standard sono organizzate in pacchetti tematici. Ad esempio:

- `java.lang` contiene le classi più comunemente utilizzate (considerate "di base" per il linguaggio)
- `java.util` contiene classi di utilità (strutture dati, data/ora, ...)
- `java.io` contiene classi per l'input/output (tramite console, files, ...)
- `java.awt` e `java.swing` contengono classi per costruire interfacce grafiche
- `java.security` contiene classi per crittografia e altri aspetti di sicurezza
- .....

La Libreria Standard di Java (2)

Le classi del pacchetto `java.lang` sono immediatamente disponibili al programmatore

Le classi degli altri pacchetti devono essere importate all'inzio del programma:

```
import java.util.Scanner; //Importa la classe Scanner
import java.util.*; //Importa tutte le classi del pacchetto java.util
```

All'interno del proprio codice, un metodo di un'altra classe si invoca così:

`<nomeclasse>.<nomemetodo>(<parametri>)`

La Libreria Standard di Java (3)

La documentazione ufficiale della Libreria Standard di Java è disponibile qui:

`http://docs.oracle.com/javase/8/docs/api/`

Trovate un link nella pagina web del corso

USATE QUESTA DOCUMENTAZIONE!!!

- Contiene TUTTE le informazioni necessarie di TUTTE le classi
- E' navigabile per singolo pacchetto, per singola classe e singolo metodo...

ATTENZIONE: dalla versione 9 di Java i pacchetti di sono ulteriormente raggruppati in moduli. Se consultate la documentazione della versione 9 (o supertiori) le classi a cui faremo riferimento le trovate nel modulo java.base.

La classe Math

Esempio: la classe `Math`

- contiene numerosi metodi che eseguono calcoli matematici

Alcuni tra i metodi più comuni della classe `Math`

- `Math.abs(a)` restituisce il valore assoluto del numero `a`
- `Math.pow(a,b)` restituisce l'elevamento a potenza a^b
- `Math.sqrt(a)` restituisce la radice quadrata di `a`
- `Math.round(a)` restituisce l'arrotondamento di `a` all'intero più vicino
- `Math.floor(a)` restituisce l'arrotondamento di `a` per difetto
- `Math.ceil(a)` restituisce l'arrotondamento di `a` per eccesso
- `Math.sin(a)` restituisce il seno di `a`
- `Math.random()` restituisce un numero frazionario casuale compreso tra 0.0 (compreso) e 1.0 (escluso). Esiste anche la classe `Random` che fa cose più sofisticate...)
- .....

Esempio di uso

```
// genera un numero casuale tra 0.0 e 1.0
double numeroCasuale = Math.random();

// lo trasforma in un numero tra 0.0 e 10.0
numeroCasuale *= 10;

// lo trasforma in un numero tra 1.0 e 10.0 senza decimali
double interoCasuale = Math.ceil(numeroCasuale);

// ne calcola il quadrato
int quadrato = (int) Math.pow(interoCasuale,2);
```

Oggetti

Moltissime delle classi della Libreria Standard si usano per creare oggetti

- Questa è la caratteristica principale di Java (e di tutti gli altri linguaggi di programmazione orientati agli oggetti)

Un oggetto è una istanza di una classe:

- un oggetto può possedere un proprio stato interno
- la classe specifica i metodi che possono essere invocati sull'oggetto (metodi d'istanza, specificano il comportamento dell'oggetto)

La relazione tra classi e oggetti è la seguente:

Una classe definisce un tipo (non primitivo) i cui valori sono oggetti

La classe String (1)

Esempio: la classe String

- contiene metodi per l'elaborazione di stringhe

Creare oggetti di tipo String

```
String nome = new String("Mario");
String cognome = new String("Rossi");
```

Le variabili nome e cognome

- sono due oggetti di tipo String
- sono due istanze della classe String
- hanno stati interni diversi ("Mario" e "Rossi");

Per creare un oggetto si usa la parola chiave new

- new è seguito dalla chiamata a un costruttore della classe: uno speciale metodo che ha lo stesso nome della classe e inizializza il nuovo oggetto

La classe String (2)

La classe String in realtà è speciale. I suoi oggetti possono essere creati anche così:

```
String nome = "Mario"; // equivalente a new String("Mario");
String cognome = "Rossi"; // equivalente a new String("Rossi");
```

Le variabili di tipo String possono essere usate come tutte le altre variabili. Ad esempio:

```
String primonome = nome; // in assegamenti
System.out.println(cognome); // stampate
metodoAusiliario(nome,cognome); // nelle chiamate a metodi ausiliari
```

La classe `String` (3)

La classe `String` mette a disposizione numerosi metodi per elaborare i propri oggetti

La chiamata di un metodo su un oggetto ha la seguente sintassi:

```
<nomeoggetto>.<nomemetodo>(<parametri>)
```

La classe String (4)

Supponendo che `str` sia un oggetto di tipo `String`, ecco alcuni tra i metodi più comuni che si possono chiamare su esso:

- `str.length()` restituisce la lunghezza della stringa (numero di caratteri)
- `str.isEmpty()` restituisce `true` se la stringa è vuota (0 caratteri) o `false` altrimenti
- `str.substring(i,j)` restituisce la porzione di `str` che va dal carattere in posizione `i` al carattere in posizione `j` (escluso).
- `str.substring(i)` restituisce la porzione di `str` che va dal carattere in posizione `i` alla fine della stringa.

ATTENZIONE A `substring`

- si inizia a contare da 0, quindi:

```
String saluto = "Hello, World!";
System.out.println(saluto.substring(7,12)); // stampa World
```

- `i` e `j` devono essere minori di `String.length()`

La classe String (5)

Un esempio: stampa un parola tre caratteri per volta

```
public class ParolaSpezzettata {

  public static void main(String[] args) {

    // parola da spezzettare
    String parola = "Supercalifragilistichespiralidoso!!";

    // scandisce la stringa 3 caratteri per volta
    for (int i=0; i<parola.length(); i+=3) {

      /* assicura che la porzione considerata (da i a i+3)
         non superi la fine della stringa */
      if (i+3 < parola.length())
        System.out.println(parola.substring(i,i+3));
      else
        System.out.println(parola.substring(i,parola.length()));

    }
  }
}
```

Il tipo primitivo `char`

Quando si elaborano stringhe spesso si deve lavorare con i singoli caratteri.

Tra i tipi primitivi di Java, il tipo `char` descrive valori che sono singoli caratteri

- Codifica Unicode: rivedere la lezione sui tipi primitivi...

I letterali di tipo carattere si descrivono tra singoli apici

- `'a'`, `'b'`, `'c'`, `'1'`, `'2'`, `'['`, `']'`, `'_'`, `' '`, ...
- caratteri speciali: `'\n'` (a capo) e altri poco usati...

Un esempio di assegnamento:

```
char carattere = 'a';
```

I valori di tipo `char` sono in realtà codificati come interi (Unicode), quindi si possono applicare operatori aritmetici:

```
carattere++; // "incrementa" di 1 il carattere
System.out.println(carattere); // stampa b
```

Switch su char

Il tipo char può essere usato anche nell'ambito di uno switch

```
char scelta = 'd';
switch (scelta) {
  case 'a': System.out.println("Hai scelto a"); break;
  case 'b': System.out.println("Hai scelto b"); break;
  case 'c': System.out.println("Hai scelto c"); break;
  case 'd': System.out.println("Hai scelto d"); break;
  case 'e': System.out.println("Hai scelto e"); break;
  default: System.out.println("Lettera sbagliata");
}
```

Altri metodi di `String` (1)

Altri metodi utili per gli oggetti di tipo `String` (in cui `c` è, ad esempio, una variabile di tipo `char`)

- `str.charAt(i)` restituisce il carattere della stringa in posizione `i`. Analogo a `str.substring(i,i+1)`, ma il risultato è di tipo `char` anziché `String`
- `str.indexOf(c)` restituisce la posizione della prima occorrenza del carattere `c` nella stringa `str` oppure -1 se non presente
- `str.indexOf(c,i)` restituisce la posizione della prima occorrenza successiva a `i` del carattere `c` nella stringa `str` oppure -1 se non presente

Altri metodi di String (2)

Esempio: stampa un carattere ogni tre

```
public class UnoOgniTre {

  public static void main(String[] args) {

    // parola da stampare
    String parola = "Supercalifragilistichespiralidoso!!";

    // scandisce la stringa 3 caratteri per volta
    for (int i=0; i<parola.length(); i+=3) {

        System.out.println(parola.charAt(i));

    }
  }
}
```

Esempio: in che posizione è la h?

```
    String parola = "Supercalifragilistichespiralidoso!!";
    System.out.println(parola.indexOf('h')); // stampa 20
```

Esempio d'uso

Conta i caratteri di spazio in un testo

```
import java.util.Scanner;

public class ContaSpazi {
  public static void main(String[] args) {

    Scanner input = new Scanner(System.in);

    System.out.println("Inserisci una riga di testo");
    String s = input.nextLine();

    int cont=0;    // contatore degli spazi
    int i=s.indexOf(' '); // indice usato per scandire il testo
    if (i!=-1) cont++;

    while (i!=-1) {
      // cerca il prossimo spazio a partire dalla posizione i+1
      i = s.indexOf(' ',i+1);
      if (i!=-1) cont++;
    }

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

Concatenazione di stringhe (1)

La concatenazione di due stringhe `"Mario"` e `"Rossi"` è semplicemente la stringa `"MarioRossi"`

L'operatore di concatenazione in Java si scrive semplicemente $+$

```
String nome = "Paolo";
String cognome = "Milazzo";
System.out.println(nome+cognome); // stampa PaoloMilazzo
System.out.println(nome+" "+cognome); // stampa Paolo Milazzo
```

L'operatore $+$ in Java è sovraccaricato di significati (overloaded): può essere applicato sia a numeri che a stringhe

Che cosa succede se mischiamo numeri e stringhe in una espressione?

```
System.out.println("Rai" + 3); // stampa Rai3
System.out.println(3 + 4 + 5);          //stampa 12
System.out.println("" + 3 + 4 + 5);     //stampa 345
System.out.println(4 + 5 + "pippo");    //stampa 9pippo
```

Concatenazione di stringhe (2)

La concatenazione è molto utile per ridurre il numero degli enunciati `System.out.print(...)`

Per esempio possiamo combinare :

```
//stampa Il risultato e':
System.out.print("Il risultato e': ");
//stampa il valore di ris sulla stessa riga e va a capo
System.out.println(ris);
```

nella singola invocazione:

```
System.out.println("Il risultato e': " + ris);
```

Convertire numeri in stringhe e viceversa

Per convertire un numero intero `i` (o double `d`) in una stringa:

- `Integer.toString(i)` (oppure `Double.toString(d)`)
- `i+""` (oppure `d+""`) (concatenazione con stringa vuota)

Per convertire una stringa `s` che rappresenta un numero in un `int` (o `double`)

- `Integer.parseInt(s)` (oppure `Double.parseDouble(s)`)

Esempi:

```
int anno = 2013;
String annoTesto = anno;          // ERRORE: tipi incompatibili
String annoTesto = "" + anno;     // OK

double pi = "3.1415926";         // ERRORE: tipi incompatibili
double pi = Double.parseDouble("3.1415926"); // OK
int miliardo = Integer.parseInt("1000000000"); // OK
```

Nota: anche `Integer` e `Double` sono classi della Java API

Input di stringhe (1)

Per chiedere all'utente di inserire una stringa si possono usare due metodi di `Scanner` (oggetto `input`)

- `input.next()` legge una singola parola
- `input.nextLine()` legge un'intera riga (tutti i caratteri fino al ritorno a capo)

```
String s1, s2, s3;
System.out.println("Inserisci due righe di testo");
s1 = input.nextLine();
s2 = input.next();
s3 = input.nextLine();
System.out.println("s1 --" + s1);
System.out.println("s2 --" + s2);
System.out.println("s3 --" + s3);
```

Esecuzione:

```
Inserisci due righe di testo
ecco la prima riga
ecco la seconda riga
s1 -- ecco la prima riga
s2 -- ecco
s3 -- la seconda riga
```

Input di stringhe (2)

Il metodo `next()` viene solitamente usato anche per leggere singoli caratteri, come segue:

```
System.out.println("Vuoi continuare? [S/N]");

//legge il primo carattere della prossima parola
char scelta = input.next().charAt(0);

switch (scelta) {
  case 'S': System.out.println("continua"); break;
  case 'N': System.out.println("basta"); break;
  default: System.out.println("ERRORE");
}
```

Esempio di esecuzione

```
Vuoi continuare? [S/N]
N
basta
```

ma in realtà succede anche quanto segue

```
Vuoi continuare? [S/N]
NOOOO!!!!
basta
```

Input di stringhe (3)

Il metodo `nextLine()` ha un problema noto

Consideriamo la seguente porzione di programma:

```
int n = input.nextInt();
String s1 = input.nextLine();
String s2 = input.nextLine();
```

CASO 1. Supponiamo che l'utente voglia inserire il seguente input:

```
44 gatti in
fila per 3
```

Abbiamo

- `n` diventa `44`
- `s1` diventa `" gatti in"`
- `s2` diventa `"fila per 3"`

Input di stringhe (4)

Consideriamo sempre la stessa porzione di programma:

```
int n = input.nextInt();
String s1 = input.nextLine();
String s2 = input.nextLine();
```

CASO 2. Supponiamo che l'utente voglia inserire il seguente input:

```
44
gatti in
fila per 3
```

Abbiamo

- `n` diventa 44
- `s1` diventa `""`
- `s2` diventa `"gatti in"`

Il metodo `nextLine()` ha letto la stringa vuota
tra 44 e il ritorno a capo della prima riga

Non c'e' una soluzione generale a questo problema... bisogna tenerne conto quando si usano questi metodi...

Confronto tra stringhe (1)

Per dire se due stringhe sono uguali non si può usare ==

- questo vale in generale per tutti gli oggetti
- == confronterebbe l'indirizzo di memoria dei due oggetti.....

La classe `String` fornisce metodi per confrontare stringhe

- `s1.equals(s2)` restituisce `true` se `s1` e `s2` sono uguali (stessa sequenza di caratteri), e `false` altrimenti
- `s1.compareTo(s2)` restituisce
  - un valore minore di 0 se `s1` precede `s2` lessicograficamente
  - un valore maggiore di 0 se `s2` precede `s1` lessicograficamente
  - il valore 0 se `s1` e `s2` sono uguali

L'ordine lessicografico è simile all'ordine alfabetico, con la differenza che nessun simbolo è ignorato (spazi, accenti, apostrofi, ecc...)

Confronto tra stringhe (2)

Esempio (NOTA: Questo è un ottimo esempio su come si fanno i cicli!!!):

```
import java.util.Scanner;
public class IndovinaParola {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    String parolaMisteriosa = "ciao";

    int tentativi=0; //contatore dei tentativi
    boolean indovinato=false; // "flag" che dice quando terminare
    do {
      String s = input.next();
      if (s.equals(parolaMisteriosa)) {
        System.out.println("INDOVINATO!!!");
        indovinato=true;  // interrompe il ciclo
      } else {
        System.out.println("Sbagliato...");
        tentativi++;
        if (parolaMisteriosa.compareTo(s)<0)
          System.out.println("La parola misteriosa precede " + s);
        else
          System.out.println("La parola misteriosa segue " + s);
      }
    } while (!indovinato && tentativi<10); // due condizioni di uscita

    // verifica il motivo dell'uscita dal ciclo
    if (!indovinato) System.out.println("Tentativi esauriti");
  }
}
```