Azzolini Riccardo 2018-10-05

Uso di classi e metodi

1 La classe `Frazione`

- Contratto: le sue istanze modellano frazioni ($\frac{2}{3}$, $\frac{1}{2}$, ecc.).
- Mette a disposizione metodi per eseguire operazioni aritmetiche e di confronto tra frazioni.

2 Espressione

Un'**espressione** è una sequenza di **operatori** e **operandi** costruita secondo le regole sintattiche del linguaggio.

Ogni espressione:

- ha un **tipo complessivo** determinato *in fase di compilazione* a seconda degli operatori e dei tipi degli operandi
- *in fase di esecuzione* produce un valore di tale tipo

3 Costruttore ed espressione di creazione

Per utilizzare un oggetto (istanza di una classe) è prima necessario costruirlo. Ciò si effettua tramite il **costruttore**, un servizio fornito dalla classe stessa.

Il costruttore viene invocato tramite un'**espressione di creazione**:

```
new nome_costruttore(lista_argomenti)
```

`new`: operatore unario prefisso

`nome_costruttore`: sempre uguale al nome della classe

`lista_argomenti`: dipende dai costruttori disponibili

Il tipo di tale espressione è la classe dell'oggetto costruito, e il valore prodotto è un **riferimento** all'oggetto. Per memorizzare tale riferimento è necessario:

1. dichiarare una variabile di tipo opportuno (la classe dell'oggetto)

2. assegnare a essa il valore restituito dall'espressione di creazione

In seguito a questo procedimento, la variabile conterrà, appunto, solo un riferimento all'oggetto, mentre l'oggetto conterrà le informazioni relative al suo stato e quelle necessarie alla JVM per poterlo gestire.

3.1 Costruttori multipli

Una classe può mettere a disposizione più costruttori, ciascuno con argomenti diversi. Ad esempio, `Frazione` ha due costruttori:

- `new Frazione(2, 3);` crea un oggetto corrispondente alla frazione $\frac{2}{3}$
- `new Frazione(2);` crea un oggetto corrispondente a $\frac{2}{1}$: è equivalente a `new Frazione(2, 1);`

3.2 Esempi

```
Frazione f;
f = new Frazione(2, 3);
```

oppure

```
Frazione f = new Frazione(2, 3);
```

4 Invocazione di un metodo

Corrisponde all'invio di un *messaggio* tra oggetti.

```
riferimento_a_oggetto.nome_metodo(lista_argomenti)
```

4.1 Esempio

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);

f.piu(g); // Somma f e g, producento un riferimento a un nuovo oggetto
          // Frazione (7/6).

Frazione h = f.piu(g); // Salva il riferimento a una Frazione 7/6 nella
                       // nuova variabile h.

f = f.piu(g); // Sovrascrive il riferimento a 2/3 contenuto in f con uno
```

```
                // a una nuova Frazione 7/6.
// La frazione 2/3 non ha più riferimenti ed è quindi inaccessibile.

f = g; // Sovrascrive nuovamente il riferimento (a 7/6) contenuto in f
       // con un riferimento a 1/2.
// Ci sono due variabili, f e g, che fanno riferimento allo stesso
// oggetto Frazione 1/2.
```

5 Direttiva di importazione

Indica al compilatore dove trovare il bytecode di una particolare classe.

```
import percorso.della.Classe;
```

In questa direttiva il `.` è un separatore di percorso (come `/` o \), e non l'operatode di invocazione di metodo.

Le classi sono raggruppate in librerie dette **package**. In particolare, il package `java.lang` contiene le classi fondamentali di Java e viene sempre importato automaticamente.

5.1 Esempio

```
import prog.utili.Frazione;
```

6 La classe `ConsoleOutputManager`

- Contratto: le sue istanze realizzano canali di comunicazione con il dispositivo di output standard (il monitor).
- Ha un costruttore privo di argomenti.
- Fornisce metodi per visualizzare a schermo vari tipi di dati.

6.1 Esempio di programma

```
import prog.io.ConsoleOutputManager;

class PrimoProgramma {
    public static void main(String[] args) {
        ConsoleOutputManager video = new ConsoleOutputManager();
        video.println("Ecco il mio primo programma!");
```

```
    }
}
```

Per eseguire questo programma, è necessario:

1. salvare il codice in un file chiamato `PrimoProgramma.java`
2. compilarlo con il comando `javac PrimoProgramma.java`, che produce il file `PrimoProgramma.class`
3. eseguirlo con il comando `java PrimoProgramma`

7 La classe `ConsoleInputManager`

- Contratto: le sue istanze realizzano canali di comunicazione con il dispositivo di input standard (la tastiera).
- Ha un costruttore senza argomenti.
- Fornisce metodi per la lettura di vari tipi di dati:
  - `readLine`: legge una riga di testo
  - `readInt`: legge un numero intero
  - altri

7.1 Esempio

```
import prog.io.ConsoleInputManager;
import prog.io.ConsoleOutputManager;

class Pappagallo {
    public static void main(String[] args) {
        ConsoleInputManager tastiera = new ConsoleInputManager();
        ConsoleOutputManager video = new ConsoleOutputManager();

        String messaggio = tastiera.readLine();
        video.println(messaggio);
    }
}
```

8 Segnatura e prototipo

La **segnatura** di un metodo è costituita da

- il nome del metodo
- i tipi dei suoi argomenti

Esempi:

- `println(String s)`
- `readLine()`

Il **prototipo** di un metodo è costituito da

- la sua segnatura (nome e tipi degli argomenti)
- il tipo del valore restituito (`void` se non restituisce un valore)

Esempi:

- `void println(String s)`
- `String readLine()`

9 Overloading

L'**overloading** ("sovraccarico") si ha quando una classe ha più metodi con lo *stesso nome* ma con *segnature diverse*.

Il compilatore capisce quale invocare in base alla lista di argomenti forniti.

10 La classe `String`

- Contratto: le sue istanze modellano stringhe, cioè sequenze arbitrarie di caratteri.
- Tra i costruttori, uno riceve come argomento una sequenza di caratteri racchiusa tra doppi apici. Ad esempio:

  ```
  String s = new String("Java");
  ```

- Mette a disposizione numerosi metodi per la manipolazione di stringhe.
- È immutabile, cioè le sue istanze non possono essere modificate. Di conseguenza, ogni operazione di "scrittura" sulle stringhe crea invece una nuova istanza.

10.1 Letterali di tipo `String`

I letterali di tipo `String` sono sequenze di caratteri racchiuse tra doppi apici. Essi fungono da sintassi abbreviata per la creazione di stringhe:

```
String s = "Java";
// equivale a
String s = new String("Java");
```

All'interno di questi letterali, è possibile rappresentare caratteri speciali (doppi apici, caratteri di controllo come "a capo", ecc.) tramite apposite *sequenze di escape* (es. `\"`, `\n`, `\\`).

10.2 Esempi di metodi

`String toUpperCase()`

Restituisce un riferimento a una nuova stringa ottenuta sostituendo, nella stringa che esegue il metodo, tutte le lettere minuscole con le maiuscole corrispondenti.

Esempi:

```
String s1 = "Ciao";
String s2 = s1.toUpperCase(); // "CIAO"
// oppure
String s2 = "Ciao".toUpperCase();
```

`String toLowerCase()`

È l'opposto di `toUpperCase`: restituisce un riferimento a una nuova stringa nella quale le maiuscole sono state trasformate in minuscole.

`int length()`

Restituisce un numero intero uguale alla lunghezza della stringa rappresentata dall'oggetto che esegue questo metodo.

Esempi:

```
"Ciao".length() // 4
"".length() // 0
```

`String concat(String str)`

Restituisce un riferimento a una stringa costruita dalla concatenazione di altre due stringhe: quella che esegue il metodo e quella specificata come argomento.

Siccome quest'operazione è molto comune, Java consente di esprimerla mediante l'operatore `+`.

```
String nome = "Mario";

String saluto = "Buongiorno ".concat(nome).concat("!");
// oppure
String saluto = "Buongiorno ".concat(nome.concat("!"));
// oppure
String saluto = "Buongiorno " + nome + "!";
```

```
String substring(int beginIndex, int endIndex)
```

Restituisce un riferimento a una *sottostringa* della stringa che esegue il metodo, costruita estraendo i caratteri situati nelle posizioni da `beginIndex` a `endIndex - 1` (cioè i caratteri le cui posizioni rientrano nell'intervallo [`beginIndex`, `endIndex`). Le posizioni dei caratteri in una stringa sono numerate a partire da 0.

Esempio:

```
"distruggere".substring(2, 9) // Estrae i caratteri da 2 (il terzo)
                              // a 8 (il nono): "strugge"
```

```
String substring(int beginIndex)
```

Restituisce un riferimento a una sottostringa costruita estraendo i caratteri dalla posizione `beginIndex` fino alla fine della stringa.

Esempio:

```
"distruggere".substring(2) // "struggere"
```