Azzolini Riccardo 2018-10-26

Caratteri, classi involucro, tipi enumerativi e `switch`

1 Tipo `char`

Rappresenta caratteri Unicode.

1.1 Letterali

- Sono racchiusi tra apici singoli.
- Contengono un singolo carattere (es. `'A'`) o una *sequenza di escape*: `'\b' '\t' '\n' '\f' '\r' '\"' '\'' '\\' '\uXXXX'`

1.2 Confronti e operazioni aritmetiche

`char` è un tipo intero senza segno su 16 bit (viene considerato come un sottoinsieme di `int`), quindi si possono usare gli stessi operatori relazionali e aritmetici definiti sui numeri interi.

Esiste una conversione implicita da `char` a `int` (ma non ai tipi numerici più piccoli, o in direzione opposta).

Il confronto avviene in base all'ordine stabilito dalla tabella dei caratteri Unicode. In particolare:

- i caratteri alfabetici sono in ordine alfabetico
- le maiuscole precedono le minuscole

Sfruttando gli operatori aritmetici e relazionali, è possibile ad esempio scrivere cicli `for` con variabile di controllo di tipo `char`:

```
for (char c = 'a', c <= 'z', c++) {
    out.println(c);
}
```

2 Classi involucro

I tipi primitivi sono stati introdotti principalmente per motivi di efficienza. Può comunque essere utile rappresentare dati di tipi primitivi sotto forma di oggetti.

A tale scopo, per ogni tipi primitivo esiste una **classe involucro** (o *wrapper*), definita in `java.lang`:

Tipo primitivo	Classe involucro
`boolean`	`Boolean`
`char`	`Character`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`

Ogni classe involucro è anche dotata di metodi (statici e non) e campi utili per trattare sia istanze della classe che valori del tipo primitivo corrispondente.

2.1 Conversioni implicite

Esistono conversioni implicite tra tipi primitivi e classi involucro:

- **autoboxing**: da un valore di tipo primitivo a un'istanza della classe involucro corrispondente
- **unboxing**: da un'istanza della classe involucro a un valore del tipo primitivo

Esempio:

```
Integer i = 123; // autoboxing, equivalente a: new Integer(123)
int j = i; // unboxing, equivalente a: i.intValue()
```

3 Tipi enumerativi

Un **tipo enumerativo** è una particolare classe, caratterizzata da:

- un elenco di valori consentiti (*costanti*), fissati all'atto della definizione del tipo
- alcuni metodi comuni a tutti i tipi enumerativi, più quelli eventualmente definiti dal programmatore

Le variabili di tipo enumerativo sono variabili riferimento, quindi:

- vengono dichiarate con la stessa sintassi usata per gli altri tipi riferimento

  `tipo_enumerativo nome_variabile;`
- i valori di tipo enumerativo sono a tutti gli effetti dei riferimenti a degli oggetti
- è possibile assegnare a esse il valore `null`

Non è però possibile creare istanze di un tipo enumerativo: si possono solo utilizzare quelle disponibili tramite le costanti, alle quali si accede tramite la sintassi

```
tipo_enumerativo.valore
```

3.1 Metodi comuni

```
public String name()
```

Restituisce il nome della costante, così come è scritto nella definizione del tipo (per convenzione, i nomi delle costanti si scrivono solitamente con tutte le lettere maiuscole).

```
public int ordinal()
```

Restituisce la posizione della costante nella sequenza ordinata dei valori possibili. Le posizioni sono numerate a partire da 0.

```
public String toString()
```

Restituisce una rappresentazione testuale della costante. Questo metodo può essere ridefinito dal programmatore che crea il tipo enumerativo.

3.2 Esempio

```
MeseDellAnno mese = MeseDellAnno.FEBBRAIO;

mese.name();      // "FEBBRAIO"
mese.ordinal();   // 1
mese.toString(); // "Febbraio"

mese.precedente(); // MeseDellAnno.GENNAIO
mese.successivo(); // MeseDellAnno.MARZO

mese.numeroGiorni();     // 28
mese.numeroGiorni(2016); // 29
mese.numeroGiorni(true); // 29
```

4 Istruzione `switch`

```
switch (espressione) {
case val1:
    ist1;
case val2:
    ist2;
// ...
case valN:
    istN;
default:
    ist;
}
```

- `espressione` (**selettore**) è un'espressione di tipo
  - `char`, `byte`, `short`, `int` o classi involucro corrispondenti
  - `String`
  - un tipo enumerativo
- `case val1:`, `case val2:`, …, `case valN:` e `default:` sono chiamate **etichette**
- `val1`, `val2`, …, `valN` sono *espressioni costanti* assegnabili al tipo del selettore (se il tipo del selettore è una classe involucro, devono essere espressioni del tipo primitivo corrispondente)
- `ist1`, `ist2`, …, `istN` e `ist` sono istruzioni singole o sequenze di istruzioni (oppure possono anche essere omesse)
- l'etichetta `default` è opzionale e si può trovare in qualunque posizione (ma per convenzione si mette solitamente alla fine)
- non possono esserci più etichette con lo stesso valore, o più etichette `default`

Semantica:

1. Viene valutato il selettore (`espressione`)
   - se è di tipo riferimento e il suo valore è `null` si verifica un errore in fase di esecuzione
   - se il suo valore corrisponde a una delle etichette (`valK`), si inizia a eseguire le istruzioni a partire da quella successiva all'etichetta (`istK`)
   - se non c'è un'etichetta uguale al valore del selettore, ma c'è il `default`, vengono eseguite le istruzioni a partire dalla prima successiva al `default`

- se non c'è né un'etichetta corrispondente al valore del selettore, né il `default`, l'esecuzione prosegue dalla prima istruzione dopo il blocco `switch`

2. L'esecuzione continua finché non si raggiunge la fine del blocco switch o viene incontrata un'istruzione `break`, quindi passa al codice successivo al blocco `switch`.

Se le istruzioni successive a un'etichetta con contengono un `break`, verranno eseguite anche quelle dopo l'etichetta successiva, e così via. Questo fenomeno, che prende il nome di **fallthrough**, può essere causa di errori, ma può anche essere sfruttato, ad esempio per eseguire le stesse istruzioni in corrispondenza di valori diversi del selettore, scrivendo due `case` di fila.

4.1 Esempio

```
String s = in.readLine();
int na = 0, ne = 0, ni = 0, no = 0, nu = 0;

for (int i = 0; i < s.length(); i++) {
    switch (s.charAt(i)) {
    case 'a':
    case 'A':
        na++;
        break;
    case 'e':
    case 'E':
        ne++;
        break;
    case 'i':
    case 'I':
        ni++;
        break;
    case 'o':
    case 'O':
        no++;
        break;
    case 'u':
    case 'U':
        nu++;
        break;
    }
}

out.println("Numero di vocali: " + na + " a, " + ne + " e, "
            + ni + " i, " + no + " o, " + nu + " u");
```