Azzolini Riccardo 2018-11-23

Implementazione delle classi

1 Componenti e progettazione di una classe

Una classe è una *ricetta* che descrive come sono fatti i suoi oggetti. In particolare, definisce:

campi: i dati che costituiscono lo stato di ogni oggetto

costruttori: porzioni di codice che creano nuovi oggetti, inizializzandone lo stato

metodi: i comportamenti di ogni oggetto

Le classi devono essere progettate, per quanto possibile, in modo indipendente dalle applicazioni che la utilizzeranno.

2 Campi della classe `Frazione`

Una frazione è definita da due numeri interi, il numeratore e il denominatore:

```
private int num;
private int den;
```

La rappresentazione degli oggetti riguarda solo chi implementa la classe, non chi la usa, quindi i campi vengono nascosti all'esterno della classe dichiarandoli `private`.

3 Costruzione di un oggetto

1. Viene riservato lo spazio in memoria per
   - le informazioni di controllo (il nome della classe)
   - lo stato dell'oggetto (i campi)
2. Viene eseguito il codice del costruttore invocato (che spesso assegna ai campi valori ricevuti come argomenti)

4 Costruttori di `Frazione`

```
public Frazione(int x, int y) {
    num = x;
    den = y;
}

public Frazione(int x) {
    num = x;
    den = 1;
}
```

All'interno di un costruttore, è possibile richiamarne un altro scrivendo, come *prima istruzione* del corpo, la parola chiave `this` seguita dalla lista degli argomenti. Il secondo costruttore può quindi essere riscritto sfruttando il primo, fornendo 1 come valore da assegnare al denominatore:

```
public Frazione(int x) {
    this(x, 1);
}
```

5 Accesso ai campi

```
nome_riferimento.nome_campo
```

I campi `private` di un oggetto sono accessibili solo all'interno della classe che definisce tale oggetto.

All'interno di costruttori e metodi è disponibile il riferimento `this`:

- in un costruttore, si riferisce all'oggetto che si sta costruendo
- in un metodo, si riferisce all'oggetto che lo esegue

L'uso di `this` non è in genere obbligatorio: l'accesso ai campi della classe stessa si può effettuare scrivendo solo il nome del campo. Esso è comunque utile per scrivere codice più leggibile e per risolvere le ambiguità tra un campo e una variabile (o un parametro) con lo stesso nome.

6 Variabili locali

Sono variabili definite nel corpo di un metodo/costruttore, usate per la memorizzazione temporanea di valori durante l'esecuzione del metodo/costruttore.

- Non vengono inizializzate automaticamente.
- Vengono distrutte quando termina l'esecuzione del metodo/costruttore.

7 Parametri formali e parametri attuali

I **parametri formali** di un metodo/costruttore sono variabili che contengono i valori degli argomenti.

I **parametri attuali** (o **argomenti**) sono invece le espressioni specificate all'atto dell'invocazione del metodo/costruttore, che vengono valutate e usate per inizializzare i parametri formali corrispondenti. Devono quindi essere di tipi compatibili con quelli dei rispettivi parametri formali.

8 Metodi aritmetici di `Frazione`

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bc}$$

```
public Frazione per(Frazione f) {
    int n = this.num * f.num;
    int d = this.den * f.den;
    return new Frazione(n, d);
}
```

$$\frac{a}{b} \pm \frac{c}{d} = \frac{ad \pm bc}{bd}$$

```
public Frazione piu(Frazione f) {
    int n = this.num * f.den + this.den * f.num;
    int d = this.den * f.den;
    return new Frazione(n, d);
}

public Frazione meno(Frazione f) {
    int n = this.num * f.den - this.den * f.num;
    int d = this.den * f.den;
```

```
    return new Frazione(n, d);
}
```

$$\frac{a}{b} : \frac{c}{d} = \frac{a}{b} \cdot \frac{d}{c} = \frac{ad}{bc}$$

```
public Frazione diviso(Frazione f) {
    int n = this.num * f.den;
    int d = this.den * f.num;
    return new Frazione(n, d);
}
```

9 Metodi di confronto tra frazioni

Poiché esistono più frazioni equivalenti, ma con numeratori e denominatori diversi (ad esempio $\frac{1}{2} = \frac{2}{4} = \frac{-2}{-4}$), non è sufficiente confrontare direttamente i valori dei campi. Una possibile soluzione è calcolare la differenza tra le due frazioni da confrontare:

- se è zero, le due frazioni sono uguali
- se è negativa, la prima frazione è minore della seconda
- se è positiva, la prima frazione è maggiore della seconda

```
public boolean equals(Frazione f) {
    Frazione g = this.meno(f);
    return g.num == 0;
}

public boolean isMinore(Frazione f) {
    Frazione g = this.meno(f);
    return (g.num < 0 && g.den > 0) || (g.num > 0 && g.den < 0);
}

public boolean isMaggiore(Frazione f) {
    Frazione g = this.meno(f);
    return (g.num < 0 && g.den < 0) || (g.num > 0 && g.den > 0);
}
```

10 Metodo `toString` di `Frazione`

```
public String toString() {
    if (den == 1)
        return String.valueOf(num);
    else
        return num + "/" + den;
}
```

Siccome il tipo restituito da questo metodo non è `void`, `return` deve essere presente in ogni possibile percorso di esecuzione.

11 Metodi `get` di `Frazione`

Solitamente i campi di una classe costituiscono un dettaglio dell'implementazione, e non devono quindi essere accessibili dall'esterno (`private`). In certi casi, tra cui la classe `Frazione`, è invece utile poter ottenere i valori di alcuni campi. Per questo, si aggiungono dei metodi che restituiscono semplicemente tali valori:

```
public int getNumeratore() {
    return num;
}

public int getDenominatore() {
    return den;
}
```

12 Limiti di quest'implementazione di `Frazione`

- È possibile rappresentare frazioni con denominatore 0, che matematicamente non hanno senso. Di conseguenza, alcune operazioni possono dare risultati altrettanto insensati. Ad esempio:
  - la divisione per zero ($\frac{a}{b} : \frac{0}{d}$) dà una frazione con denominatore zero
  - la divisione per una frazione con denominatore zero ($\frac{a}{b} : \frac{c}{0}$) dà come risultato zero
- Le frazioni sono rappresentate, e quindi anche visualizzate (`toString`) senza semplificazioni. Questo problema ha due possibili soluzioni:
  - semplificare le frazioni subito prima di costruire le stringhe

  - semplificare le frazioni al momento della costruzione, cioè rappresentarle sempre in modo semplificato

- Oltre alla mancanza di semplificazioni, lo stesso valore può avere più rappresentazioni anche perché il segno è determinato sia dal numeratore che dal denominatore. Sarebbe meglio usare uno solo dei due campi per il segno e rendere l'altro sempre positivo.

Rappresentare le frazioni in modo uniforme relativamente a semplificazioni e segno consentirebbe inoltre di scrivere in modo più semplice i metodi di confronto:

- `equals` potrebbe confrontare direttamente i valori dei numeratori e dei denominatori
- `isMinore` e `isMaggiore` dovrebbero esaminare solo il numeratore della differenza per determinarne il segno

13 Costruttore migliorato di `Frazione`

Per risolvere i problemi della prima implementazione di `Frazione`, si effettuano le seguenti modifiche al costruttore:

- se il denominatore è 0, viene sollevata un'eccezione
- la frazione viene semplificata in fase di costruzione
- il segno viene memorizzato al numeratore

```
public Funzione(int x, int y) {
    if (y == 0)
        throw new ArithmeticException("Frazione non valida: "
                                      + "denominatore 0");

    boolean negativo = (x < 0 && y > 0) || (x > 0 && y < 0);

    // Trasforma x e y nei loro valori assoluti
    // per il calcolo dell'MCD
    if (x < 0)
        x = -x;
    if (y < 0)
        y = -y;

    int m = mcd(x, y);
    if (negativo)
        num = -x / m; // Memorizza il segno al numeratore
    else
```

```
        num = x / m;
    den = y / m;
}
```

Il metodo `mcd` calcola il massimo comune divisore mediante l'algoritmo di Euclide:

```
private static int mcd(int a, int b) {
    int resto;
    do {
        resto = a % b;
        a = b;
        b = resto;
    } while (resto != 0);
    return a;
}
```

`mcd` è dichiarato come metodo:

- statico perché esegue un calcolo utile alla classe per la costruzione di oggetti, quindi non è legato a un singolo oggetto
- privato perché è un metodo utile interno, che non deve essere reso disponibile agli utenti della classe

13.1 Tipo di eccezione sollevata

Questo nuovo costruttore solleva un'eccezione della classe `ArithmeticException`, che è non controllata.

Supponendo di sollevare invece un'eccezione controllata (ad esempio `Exception`), sono necessarie varie modifiche al codice.

- Il costruttore `Frazione(int x, int y)` e il metodo `diviso`, cioè le porzioni di codice in cui si possono concretamente verificare anomalie, devono delegare esplicitamente l'eccezione:

  ```
  public Frazione(int x, int y) throws Exception { ... }
  public Frazione diviso(Frazione f) throws Exception { ... }
  ```

- Nel costruttore `Frazione(int x)` è garantito che il denominatore sia diverso da 0, ma se lo si implementa mediante `this(x, 1)` il compilatore richiede di gestire comunque l'eccezione potenzialmente sollevata dal costruttore `Frazione(int x, int y)`. La soluzione più semplice è scrivere questo costruttore senza riutilizzare l'altro:

```java
public Frazione(int x) {
    num = x;
    den = 1;
}
```

- Gli altri metodi `piu`, `meno` e `per` non possono creare frazioni con denominatore zero, quindi idealmente non dovrebbero sollevare eccezioni, ma siccome utilizzano al loro interno il costruttore `Frazione(int x, int y)` devono gestirle comunque. È quindi necessario scrivere del codice "inutile" che intercetti l'eccezione "impossibile". Ad esempio, il metodo `per` diventa:

  ```java
  public Frazione per(Frazione f) {
      int n = this.num * f.num;
      int d = this.den * f.den;
      try {
          return new Frazione(n, d);
      } catch (Exception e) {
          return null; // Impossibile
      }
  }
  ```

  L'istruzione `return null` nel blocco `catch` è necessaria perché ogni possibile percorso di esecuzione del un metodo deve restituire un valore.

14 Implementazione di `Comparable<T>`

Per implementare l'interfaccia generica `Comparable<T>`, la classe `Frazione` deve:

1. Dichiarare che implementa `Comparable<T>`, istanziando `T` con `Frazione`:

   ```java
   public class Frazione implements Comparable<Frazione>
   ```

2. Implementare il metodo `public int compareTo(T o)` specificato dall'interfaccia:

   ```java
   public int compareTo(Frazione altra) {
       if (this.equals(altra))
           return 0;
       else if (this.isMinore(altra))
           return -1;
       else
           return 1;
   }
   // oppure
   public int compareTo(Frazione altra) {
   ```

```
    return this.meno(altra).num;
}
```