Sistemi per l'Elaborazione dell'Informazione 2

Intervalli

Lo scopo di questa esercitazione è la creazione di tre implementazioni di una classe che rappresenta intervalli di interi. Ogni intervallo definito da due estremi interi, quello sinistro e quello destro, che sono inclusi nell'intervallo. In tutte le implementazioni le istanze sono immutabili.

Prima implementazione (elementare)

La prima implementazione, che chiameremo `BasicInterval`, ha attributi `left` e `right` pubblici e finali, e due costruttori, uno che chiede entrambi gli intervalli mentre l'altro chiede solo un punto. Il metodo `length()` deve fornire il numero di interi contenuto nell'intervallo. Il metodo `contains(int)` deve restituire `true` se l'argomento cade nell'intervallo, `false` altrimenti. Ricordatevi di implementare ragionevolmente `equals()` e `hashCode()` (potete ad esempio fare riferimento al terzo capitolo di J. Bloch, *Effective Java*, disponibile on-line all'indirizzo `http://java.sun.com/developer/Books/effectivejava/Chapter3.pdf`).

È consigliabile creare un'opportuna interfaccia `Interval` che verrà implementata da tutte le classi rappresentanti intervalli che andremo a realizzare durante questa esercitazione (vi semplificherà molto la risoluzione dell'ultimo esercizio); è anche consigliabile dividere implementazioni differenti in package differenti.

Seconda implementazione (con metodi di fabbricazione e pesi piuma per gli intervalli di lunghezza uno)

Nella seconda implementazione, che chiameremo `EnhancedInterval`, i costruttori sono privati e sostituiti da due metodi di fabbricazione. La classe costruisce all'inizializzazione tutti gli intervalli singoletto nonnegativi fino a una costante `MAX_FLYWEIGHT`, e li memorizza in un vettore. I metodi di fabbricazione devono restituire le copie memorizzate ogni volta che questo è possibile.

Terza implementazione (con metodi di fabbricazione che restituiscono sottoclassi specifiche, pesi piuma per gli intervalli di lunghezza uno e un singoletto per l'intervallo vuoto)

La terza implementazione spinge al massimo l'ottimizzazione dello spazio. Per ottenere questo scopo, aumentiamo l'astrazione: gli estremi vengono ottenuti tramite i metodi `left()` e `right()`. La classe principale, che chiameremo `AbstractInterval`, diventa ora astratta.

Vogliamo inoltre avere la possibilità di rappresentare l'intervallo vuoto. L'intervallo vuoto ha lunghezza zero, non contiene nessun intero e lancia un'eccezione `UnsupportedOperationException` se si cercano di sapere i suoi estremi.

La classe `AbstractInterval` implementa solo alcuni dei metodi (`length()` e `contains(int)` rappresentano ovvi esempi, ma potete anche ragionare su `equals()`), lasciando i rimanenti astratti. A questo punto, ci sono tre sottoclassi concrete di `AbstractInterval`:

1. `EmptyInterval`, privo di attributi, che rappresenta l'intervallo vuoto. La classe va implementata come un singoletto (non ha senso avere più intervalli vuoti).

2. `PointInterval`, con un solo attributo `point`, che rappresenta gli intervalli di lunghezza uno. Questa classe deve occuparsi anche della gestione dei pesi piuma.

3. `FullInterval`, con attributi `left` e `right`, che rappresenta gli intervalli rimanenti.

Tutte le classi devono ovviamente implementare `left()` e `right()`. Inoltre, devono fornire dei metodi di fabbricazione con visibilità limitata al pacchetto (opzionalmente, `EmptyInterval` può rendere accessibile la sua sola istanza in un campo statico finale con visibilità di pacchetto inizializzato direttamente). Notate che ci vuole un po' di cura nell'implementazione di `equals()` e `hashCode()`.

Infine, `Interval` conterrà i metodi (statici) di fabbricazione generici per gli intervalli. Potete scegliere i metodi in vario modo, ad esempio chiamandoli `getInstance(int,int)`, `getInstance(int)` e `getInstance()` (per l'intervallo vuoto), oppure in modo più creativo, come `getInterval(int,int)`, `getPointInterval(int)` e `getEmptyInterval()`. I metodi di fabbricazione restituiscono sempre un `Interval`.

Quarta implementazione (tramite fabbrica astratta che contiene metodi di fabbricazione, e può venire istanziata in almeno due forme)

La quarta implementazione cerca di essere, per quanto possibile, flessibile ed efficiente, lasciando all'utente la scelta dell'implementazione. Dovete dichiarare una fabbrica astratta di intervalli (cioè un'interfaccia) che contiene i metodi di fabbricazione. Dovete quindi implementare almeno due diverse fabbriche concrete (per esempio, con e senza pesi piuma, ma se avete seguito i suggerimenti dati alla fine del primo esercizio dovreste riuscire a riutilizzare le tre implementazioni fatte finora). Le fabbriche concrete devono essere restituite da un metodo `getFactory(String)`, da inserire in una utility (cioè una classe che contiene esclusivamente metodi e attributi statici, come per esempio accade con `Math`), a cui viene fornito un nome simbolico (che dovreste mettere in una stringa pubblica, statica e finale). Verificate anche se abbia senso oppure no implementare le fabbriche concrete come singoletti.

Per verificare che tutto avviene come si deve, scrivete un semplice `main()` che chiede all'utente il nome di una fabbrica, la istanza, crea i tre tipi fondamentali di intervalli e ne stampa la classe implementante (usate `getClass()` sugli oggetti restituiti dalla fabbrica), in modo da controllare che la creazione degli oggetti sia andata come vi aspettavate.

Parte facoltativa

Utilizzando UML scrivete i diagrammi delle classi e i diagrammi di sequenza per tutti gli esercizi risolti. Scrivete la documentazione JavaDoc per tutte le classi e i metodi coinvolti, evidenziando gli invarianti. Infine, scrivete dei test di funzionamento per tutte le classi coinvolte.