Università degli Studi di Camerino
Scuola di Scienze e Tecnologie
Corso di Laurea in Informatica
e in
Informatica per la Comunicazione Digitale
Corso di Algoritmi e Strutture Dati 2022/2023
Docenti: Emanuela Merelli e Luca Tesei

Istruzioni per la realizzazione del Miniprogetto 1

ASDL2223MP1

Descrizione

Il miniprogetto ASDL2223MP1 consiste nei seguenti task:

1. Implementare la classe `CrivelloDiEratostene` i cui oggetti costruiscono un crivello per un numero dato `c` e sono in grado di restituire tutti i numeri primi minori o uguali di `c`.
2. Implementare la classe `Factor` i cui oggetti rappresentano fattori primi di numeri naturali con una certa molteplicità.
3. Implementare la classe `Factoriser`, una classe singoletto che realizza un oggetto attore che fa la fattorizzazione in numeri primi di un dato numero naturale.
4. Implementare la classe `Shelf` i cui oggetti rappresentano mensole su cui diversi oggetti possono essere poggiati. Le mensole hanno una superficie massima di appoggio e un massimo di carico di peso. La classe deve controllare che questi limiti vengano rispettati.
5. Implementare le classi `Book` e `RoundLamp` che rappresentano possibili oggetti che possono essere appoggiati su una mensola.

Crivello di Eratostene

Il crivello deve essere creato con una capacità iniziale immutabile c, che deve essere almeno 2. Esso può essere semplicemente rappresentato da un array `crivello` di `boolean` di lunghezza `c+1` in cui le posizioni 0 e 1 non sono usate. Dopo aver inizializzato il crivello, deve valere che per ogni `i = 2, 3, ..., c crivello[i] == true` se e solo se il numero `i` è un numero primo. L'inizializzazione del crivello è visualizzata con una animazione su https://it.wikipedia.org/wiki/Crivello_di_Eratostene .

Una volta inizializzato il crivello, per sapere se un numero `n` minore o uguale a `c` è primo basta consultare `crivello[n]`. Se il valore è true allora `n` è primo, altrimenti non è primo.

La classe fornisce anche un modo per elencare, anche più volte, tutti i numeri primi da 2 al numero primo più grande che sia minore o uguale di `c`. In ogni momento è attivo un *elenco corrente*, che parte dall'inizio (cioè dal numero 2) alla creazione dell'oggetto. Per realizzare questa funzionalità devono essere forniti 3 metodi:

- `boolean hasNextPrime()` - Indica se l'elenco corrente dei numeri primi di questo crivello ha ancora un numero disponibile da elencare o se l'elenco è giunto al termine perché sono già stati elencati tutti i numeri primi minori uguali alla capacità.
  Se il metodo restituisce true, può essere fatta una ulteriore chiamata al metodo `nextPrime()` per ottenere il numero successivo nell'elenco. Se il metodo restituisce false non si potrà più chiamare il metodo `nextPrime()` fino a quando l'elenco non viene fatto ripartire tramite il metodo `restartPrimeIteration()`.
- `int nextPrime()` - Restituisce il prossimo numero primo in questo crivello nell'elenco corrente. L'elenco parte sempre dal numero 2 e si interrompe non appena il metodo `hasNextPrime()` diventa false. Il metodo lancia l'eccezione `IllegalStateException` se si prova a chiedere il prossimo numero primo quando l'elenco corrente è terminato. L'elenco può essere fatto ripartire in qualsiasi momento chiamando il metodo `restartPrimeIteration()`.
- `void restartPrimeIteration()` - Fa ripartire da 2 l'elenco corrente dei numeri primi fino alla capacità di questo crivello. Questo metodo può essere chiamato in qualsiasi momento, anche se l'elenco corrente non è ancora terminato. L'effetto è comunque di ricominciare da 2.

Factor

Un oggetto `Factor` è essenzialmente una coppia di valori `int`, il primo rappresenta un numero primo e il secondo la sua molteplicità. La molteplicità deve essere almeno 1. I `Factor` sono ordinati in base al numero primo e, in caso di numero primo uguale, in base alla molteplicità. Due `Factor` sono uguali se e solo se sono la stessa coppia di numero primo e molteplicità.

Factoriser

Un oggetto `Factoriser` fornisce un metodo `Factor[] getFactors(int n)` che restituisce la sequenza ordinata di tutti i `Factor` primi del numero `n`. Il metodo deve utilizzare un crivello per il suo lavoro e restituire il risultato come array di fattori primi (classe `Factor`). Si veda https://it.wikipedia.org/wiki/Fattorizzazione per alcuni suggerimenti sull'algoritmo di fattorizzazione. Esistono algoritmi complicati che sono più efficienti di quello "di base", ma non è necessario ottimizzare in questo progetto l'efficienza di questo metodo. In effetti al momento non si conosce un metodo veramente efficiente per fare la fattorizzazione.

Shelf

Un oggetto Shelf rappresenta una mensola su cui possono essere appoggiati degli oggetti. Tali oggetti possono essere di diverso tipo, ma tutti implementano l'`interface ShelfItem` fornita nel template del codice. Un oggetto non può essere appoggiato sulla

mensola se ha lunghezza o larghezza che eccedono quelle della mensola stessa. La mensola può contenere un numero non precisato di oggetti, ma ad un certo punto non si possono appoggiare oggetti la cui superficie occupata o il cui peso fanno eccedere la massima superficie occupabile o il massimo peso sostenibile definiti nel costruttore della mensola.

La collezione degli oggetti appoggiati in un certo momento viene mantenuta in un array di `ShelfItem`. Tale array ha una capacità iniziale predefinita da una costante della classe. Qualora arrivasse una richiesta di inserimento di un nuovo oggetto che facesse superare la capacità dell'array la classe deve provvedere a raddoppiare l'array per fare posto al nuovo oggetto.

Book e RoundLamp

Due classi che rappresentano libri o lampade dalla base circolare. Esse implementano l'interface `ShelfItem` e quindi devono contenere i campi e i metodi necessari.

Traccia e Implementazione

La traccia del codice è fornita come .zip contenente i template delle seguenti classi/interfacce:

1. `it.unicam.cs.asdl2223.mp1.CrivelloDiEratostene`
2. `it.unicam.cs.asdl2223.mp1.CrivelloDiEratosteneTest`
3. `it.unicam.cs.asdl2223.mp1.Factor`
4. `it.unicam.cs.asdl2223.mp1.FactorTest`
5. `it.unicam.cs.asdl2223.mp1.Factoriser`
6. `it.unicam.cs.asdl2223.mp1.FactoriserTest`
7. `it.unicam.cs.asdl2223.mp1.ShelfItem`
8. `it.unicam.cs.asdl2223.mp1.Shelf`
9. `it.unicam.cs.asdl2223.mp1.ShelfTest`
10. `it.unicam.cs.asdl2223.mp1.Book`
11. `it.unicam.cs.asdl2223.mp1.RoundLamp`

che si possono importare nel proprio IDE preferito. La versione del compilatore Java da definire nel progetto è la 1.8 (Java 8) e devono essere incluse nel BuildPath le librerie JUnit 5.

Nell'implementazione:

- **non si possono usare versioni del compilatore superiori a 1.8**;
- **è vietato l'uso di lambda-espressioni** e di funzionalità avanzate di Java 8 o superiore (es. stream, ecc…);
- **è obbligatorio usare il set di caratteri utf8** per la codifica dei file del codice sorgente.

Sono fornite le classi di test JUnit che saranno utilizzate per testare il codice consegnato. Quindi è **estremamente importante** leggere bene questo documento e le API

scritte nel formato javadoc nei template delle interfacce/classi date. In caso di dubbi si utilizzi il documento google condiviso associato a questo compito denominato "`ASDL2223MP1 Q&A Traccia`" per controllare le risposte a domande già fatte e/o per fare una nuova domanda.

Modalità di Sviluppo e Consegna

Vanno implementati tutti i metodi richiesti (segnalati con commenti della forma `// TODO implementare` nel template delle classi). Nelle altre posizioni segnate con `// TODO` si debbono/possono inserire le parti richieste.
Non è consentito:

- aggiungere classi pubbliche;
- modificare la firma (signature) dei metodi già specificati nella traccia;
- modificare le variabili istanza già specificate nella traccia.

E' consentito:

- aggiungere classi interne per fini di implementazione;
- aggiungere metodi privati per fini di implementazione;
- aggiungere variabili istanza private per fini di implementazione.

Nel file sorgente di ogni classe implementata, nel commento javadoc della classe, modificare il campo @author come indicato:

```
@author Luca Tesei (template) // TODO INSERIRE NOME, COGNOME ED EMAIL
         xxxx@studenti.unicam.it DELLO STUDENTE (implementazione)
```

Creare una cartella con il seguente nome con le **lettere maiuscole e i trattini** (non underscore!) **esattamente come indicato senza spazi aggiuntivi o altri caratteri**:

Valutazione

ATTENZIONE: Il codice consegnato verrà sottoposto a un **software antiplagio** che lo confronterà con tutti gli altri codici consegnati dagli altri studenti. Il software segnala una percentuale di somiglianza e dei gruppi di studenti con codice probabilmente plagiato.

La valutazione si baserà sui seguenti criteri, in ordine decrescente di importanza:

1. **Codice scritto individualmente. Nel caso di conclamato "plagio" il voto del miniprogetto 1 di tutti i "plagi" verrà decurtato di 10 punti.**
2. **Compilabilità**. Il codice consegnato deve essere in formato **utf8** e deve essere compilabile con il compilatore Java standard **versione 1.8**. In caso contrario ci sarà una penalizzazione in punti del voto finale.
3. **Correttezza**. Il codice consegnato verrà sottoposto ai test JUnit dati. Il mancato superamento di un test o degli errori nella codifica comporteranno il decurtamento di un certo numero di punti (a seconda del test e degli errori, una griglia di valutazione verrà fornita all'atto della riconsegna con il voto). Se una classe o una parte di essa

non vengono implementate del tutto allora il punteggio assegnato per quella parte sarà zero e il punteggio del numero di test falliti non sarà conteggiato.

4. **Prova Parziale 1 - scritto**: la non risposta alla domanda relativa al progetto della prova scritta comporta una penalizzazione in punti del voto finale del miniprogetto
5. **Prova Parziale 1 - scritto**: se si risponde alla domanda, ma la risposta non è congruente con il codice consegnato, ci sarà una **penalizzazione di 10 punti** del voto finale del miniprogetto
6. **Codice chiaro, leggibile e ben commentato**.
7. Scelta di strutture dati e implementazione **efficienti** sia dal punto di vista del tempo di esecuzione che dello spazio richiesto.