Metodologie di Programmazione - Canale P-Z
A.A. 2008/09 - Prova simulata del 08/06/2009

Avvertenze Le soluzioni devono essere scritte in modo chiaro e ordinato. Il codice deve essere indentato per facilitarne la lettura. Si possono consultare libri e appunti ma non è permesso usare computer.

Ogni esercizio o parte di esercizio ha un punteggio massimo. Il voto (in trentesimi) sarà determinato dalla somma dei punteggi ottenuti nei vari esercizi. (Notare che la somma dei punti degli esercizi proposti supera 30. Quindi, per ottenere il massimo non è necessario svolgere e ottenere il massimo dei punti in tutti gli esercizi.)

Esercizio 1 [16 punti]

Si vuole progettare e sviluppare un'applicazione per la riproduzione di file musicali. In particolare, si consideri il seguente frammento di specifica dei requisiti (relativo solo ad una parte del sistema complessivo):

L'applicazione deve poter rappresentare oggetti che denotano *brani musicali* presenti nel file system della macchina dell'utente. In particolare, di ogni brano il sistema deve poter rappresentare le seguenti informazioni: directory e nome del file nel quale è memorizzato, titolo, autore, genere (sotto forma di stringhe), e durata in secondi (un intero). Si assuma che tali informazioni siano tutte disponibili all'atto della creazione di questi oggetti. **[2 punti]**

L'utente dell'applicazione deve poter anche definire delle *playlist*, che sono delle sequenze *ordinate* di brani da riprodurre. Di ogni playlist il sistema deve rappresentare il titolo (una stringa), l'elenco dei brani da riprodurre (con il relativo ordinamento), e la durata complessiva in secondi. La durata di una playlist è pari alla somma della durata dei relativi brani. **[6 punti]**

La creazione di una playlist deve avvenire fornendone esclusivamente il titolo (creando quindi una playlist "vuota"). Successivamente l'utente deve poter inserire e rimuovere singoli brani. L'inserimento dei brani deve poter essere effettuato in due modi: aggiunta di un brano in ultima posizione e aggiunta di un brano in una posizione specifica. Una playlist può contenere lo stesso brano più volte.

Infine, su una playlist p esistente devono potersi effettuare le seguenti operazioni:

1. Dato un genere g, restituire una nuova playlist q che contiene tutti e soli i brani di p di genere g. L'ordinamento dei brani nella nuova playlist deve essere coerente con quello della playlist p: il brano b_1 verrà prima di b_2 in q se e solo se b_1 viene prima di b_2 in p. **[2 punti]**
2. Rimescolamento casuale: i brani nella playlist p vengono ri-ordinati in modo casuale, facendo attenzione che due brani successivi siano di *artisti diversi*. Se ciò non è possibile, l'operazione deve restituire una opportuna eccezione. **[6 punti]**

Si richiede di progettare e realizzare un insieme di classi Java che permettano di rappresentare brani musicali e playlist, con tutti i campi dati e i metodi necessari a fornire i servizi descritti. Si ignori invece la parte di sistema che visita il file system dell'utente alla ricerca di file musicali.

Esercizio 2 [14 punti]

Si vuole realizzare un framework che permetta l'implementazione di ordini parziali, invece che di ordini totali come quelli definibili con le classi generiche `Comparable` e `Comparator`. A tale scopo si vogliono utilizzare due interfacce generiche che rispettano le seguenti specifiche:

1. un'interfaccia `ParzOrdinabile` (simile alla `Comparable`) che contiene un metodo `confrontabileCon` e un metodo `confrontaCon`. Il metodo `confrontabileCon` restituisce un `boolean` che indica se l'oggetto della classe che implementa l'interfaccia è confrontabile con quello di tipo generico `T` che il metodo riceve come argomento. Il metodo `confrontaCon` restituisce un intero negativo, `0` o un intero positivo in base alle stesse regole della `compareTo` della interfaccia `Comparable` quando gli oggetti sono confrontabili o solleva un'eccezione controllata `OggettiInconfrontabiliException` se gli oggetti non sono confrontabili; **[2 punti]**

2. un'interfaccia `ParzOrdinatore` (simile alla `Comparator`) che contiene un metodo `confrontabili` e un metodo `confronta`. Il metodo `confrontabili` riceve due oggetti del tipo generico `T` e restituisce un `boolean` che indica se tali oggetti sono confrontabili. Il metodo `confronta` restituisce un intero negativo, `0` o un intero positivo in base alle stesse regole della `compare` della interfaccia `Comparator` quando gli oggetti sono confrontabili o solleva un'eccezione controllata `OggettiInconfrontabiliException` se gli oggetti non sono confrontabili. **[2 punti]**

Si definisca la classe dell'eccezione controllata `OggettiInconfrontabiliException` (si ricorda che tutte le sottoclassi di `Exception` che non sono sottoclassi di `RuntimeException` sono controllate). Al momento della creazione ogni istanza di questa classe riceve il messaggio associato all'eccezione. **[2 punti]**

Si prenda ora in esame l'ordine parziale tra stringhe definito dalla relazione di prefisso. Più precisamente, rispetto all'ordine parziale dei prefissi una stringa s_1 precede una stringa s_2 se e solo se s_1 è un prefisso di s_2 (quindi due stringhe sono confrontabili se e solo se una delle due è un prefisso dell'altra).

1. Si definisca una classe `PrefixParzOrdString` estensione della classe `String` che implementa l'interfaccia `ParzOrdinabile` rispetto all'ordine parziale dei prefissi. **[4 punti]**

2. Si definisca una classe `PrefixStringOrdinatore` che implementa l'interfaccia `ParzOrdinatore` rispetto all'ordine parziale dei prefissi. **[4 punti]**

Esercizio 3 [10 punti]

Si realizzi una classe generica `Coppia` che permette di memorizzare coppie di oggetti di tipo generico `T` assumendo che, al momento della creazione della coppia, gli oggetti da memorizzare vengano clonati (ovvero, la classe non deve contenere riferimenti ad oggetti non creati al suo interno) e che gli elementi della coppia siano immutabili. **[3 punti]**

Si vuole inoltre che due istanze di `Coppia` siano considerate uguali se contengono gli stessi elementi, indipendentemente dall'ordine in cui appaiono nella coppia. Si sovrascriva il metodo `equals` della classe in modo che rispetti questa equivalenza. **[3 punti]**

Avendo riscritto `equals` è necessario riscrivere di conseguenza anche `hashcode`. Per quale motivo? Va bene utilizzare come hashcode della coppia la somma degli hascode degli elementi? (Motivare la risposta). **[4 punti]**