29-01-2014

Fondamenti di Informatica 2 – Prova Scritta

Nome: …..... Cognome: …..... Matricola: ..... Postazione N. : .....

TEMA ALTERNATIVO DI VERIFICA DELL'APPRENDIMENTO

Sviluppare una semplice Interfaccia Grafica che serva ad eseguire dei test sui metodi di java.awt.List e sulle sue due implementazioni ArrayList e LinkedList. L'Utente deve scegliere (1) l'Algoritmo da testare, e (2) la lista di test su cui effettuare i calcoli, e sull'interfaccia compaiono i tempi di calcolo necessari all'esecuzione del metodo con ArrayList e LinkedList
Algoritmi da Testare:

- **test di add; prende i primi 10 elementi della lista (con get) e li riaggiunge in fondo alla lista.**
- **test di remove; il test rimuove i primi 10 elementi della lista.**
- **test di indexof; il test usa il metodo 'get' per recuperare il decimo elemento (indice 9). Poi effettua in indexof dello stesso, ottenendo come risultato 9 (il decimo indice)**

Liste su cui effettuare i test:

- **Una lista di 20 Parole Italiane, dai 4 ai 6 caratteri di lunghezza.**
- **Una lista di 100 interi ordinati da 0 a 99.**
- **Una lista di Double contenente le radici quadrate dei primi 100 numeri da 1 a 100.**

I Test dovrebbero avvenire utilizzando System.nanoTime() per ottenere 2 volte il tempo di sistema, prima e dopo l'esecuzione degli algoritmi da testare, in modo da ottenere il tempo di calcolo dalla differenza dei tempi.

1. Creare un Workspace **Eclipse**. Creare un Progetto **esame**. Dopo aver studiato il problema, implementare in **Java** una possibile soluzione modulare e ad oggetti.
2. Su foglio protocollo, **a titolo di documentazione e ai fini della valutazione**, si realizzi uno schema UML sintetico che metta in luce le relazioni che intercorrono tra i moduli implementati. E' possibile utilizzare ObjectAID UML, ma in quel caso è obbligatorio esportare gli schemi UML in formato immagine png. **Si invita inoltre ad utilizzare la documetnazione Javadoc nel codice dove lo si ritenga opportuno**.
3. Lo studente può accedere al percorso **/home/etc/FDI2** per recuperare la documentazione **Javadoc**, i cosiddetti **esempi forniti** e altro materiale utile. E' inoltre possibile consultare qualsiasi testo scritto.
4. Alla fine dell'esame, esportare un file zip attraverso la funzionalità **Export...** di eclipse (vedi le **istruzioni di salvataggio dati**) e salvarlo come **/home/esm/esame_N/esame_N.zip** (ad esempio **/home/esm/esame_20/esame_20.zip**)

Istruzioni

1) I Temi d'Esame visti a lezione più vicini a questo problema sono Risolutore di Equazioni e Cruciverba.
2) Il codice dovrebbe seguire l'Architettura Model-View-Control. In particolare, dovrebbe esserci una classe contenente il Modello del Problema priva di dipendenze funzionali con i moduli dell'interfaccia grafica.
3) I Due Metodi dovrebbero essere descritti da una interfaccia contenente questo metodo

```
/**Calcola il tempo di esecuzione in nanosecondi di un
metodo della classe List*/
public long testMetodo(List<Object> daTestare);
```

Usiamo 'Object', in modo che le liste possano contenere un qualsiasi tipo di oggetto. Per lavorare con gli Interi bisognerà usare la classe Wrapper java.lang.Integer.

4) Non c'è invece motivo di astrarre sulle liste di Test. Il Modello del sistema conterrà le due istanze delle liste da confrontare, e l'istanza di un oggetto che implenta l'interfaccia di cui al punto 3.
5) E' opportuno usare il Pattern Proxy, che in questo caso serve all'implementazione dei moduli dell'interfaccia che servono all'utente per selezionare il metodo da testare.

Istruzioni

4) I Temi d'Esame visti a lezione più vicini a questo problema sono Risolutore di Equazioni e Cruciverba.
5) Il codice dovrebbe seguire l'Architettura Model-View-Control. In particolare, dovrebbe esserci una classe contenente il Modello del Problema priva di dipendenze funzionali con i moduli dell'interfaccia grafica.
6) I Due Metodi dovrebbero essere descritti da una interfaccia contenente questo metodo

```
/**Calcola il tempo di esecuzione in nanosecondi di un
metodo della classe List*/
public long testMetodo(List<Object> daTestare);
```

Usiamo 'Object', in modo che le liste possano contenere un qualsiasi tipo di oggetto. Per lavorare con gli Interi bisognerà usare la classe Wrapper java.lang.Integer.

4) Non c'è invece motivo di astrarre sulle liste di Test. Il Modello del sistema conterrà le due istanze delle liste da confrontare, e l'istanza di un oggetto che implenta l'interfaccia di cui al punto 3.
5) E' opportuno usare il Pattern Proxy, che in questo caso serve all'implementazione dei moduli dell'interfaccia che servono all'utente per selezionare il metodo da testare.