HAPPY CODE INC

BR-jsys *business rules* per sistemi gestionali in architettura J2EE

Norme Di Progetto

Versione 2.0 - 16 febbraio 2008

Capitolato: "BR-jsys"

Data creazione:	12/11/07
Versione:	2.0
Stato del documento:	Formale ad uso Interno
Revisione RR	
Redazione:	Filippo Carraro
Revisione:	Marco Tessarotto
Approvazione:	Elena Trivellato
Revisione RPD	
Redazione:	Elena Trivellato
Revisione:	Filippo Carraro
Approvazione:	Mattia Meroi

Lista di distribuzione

Tutta la HappyCode inc	Gruppo di lavoro

Diario delle modifiche

Versione	Data rilascio	Descrizione
2.0	15/02/2008	Correzione del documento.
1.9	05/02/2008	Aggiunta del nome del file nel modello di documento.
1.8	31/01/2008	Riorganizzazione dei capitoli. Aggiunta delle norme per la codifica degli statements. Modifica alle Norme di comportamento e alle Norme di documento.
1.7	30/01/2008	Modifica al layout degli esempi.
1.6	30/01/2008	Aggiunta del capitolo 'Linee e spazi bianchi'
1.5	28/01/2008	Aggiunta del capitolo 'Dichiarazioni'
1.4	25/01/2008	Aggiunta dei capitoli 'Indentazione' e 'Commenti'
1.3	24/01/2008	Aggiunta del capitolo 'File Java'
1.2	23/01/2008	Aggiunta del capitolo 'Condivisione, archiviazione e versionamento dei file'
1.1	22/01/2008	Modifica al layout dei documenti.
1.0	21/12/2007	Documento sottoposto a revisionamento automatico.
0.3	03/12/2007	Revisione del documento. Modifiche al Comportamento generale e all'uso dello spazio web in Google Groups.
0.2	26/11/2007	Revisione del documento. Correzione errori ortografici.
0.1	12/11/2007	Stesura preliminare delle norme per i documenti.

Indice

Capitolo 1

Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di fornire una lista di norme generali sia per il comportamento dei vari membri del gruppo, sia per la stesura di tutta la documentazione interna ed esterna. L'obiettivo è quello di fissare delle norme di codifica ben precise, in modo da poter migliorare la leggibilità del codice e facilitarne la comprensione. Tutti i componenti del gruppo dovranno attenersi a tali regole.

1.2 Riferimenti

- "Ingegneria del software" 8a edizione - Ian Sommerville
- Molte delle norme di codifica sono state scritte riferendosi alle convenzioni sul codice java della Sun Microsystems.

1.3 Glossario

Il glossario viene fornito come file esterno chiamato Glossario.1.8.pdf .

Capitolo 2

Norme di Comportamento

2.1 Uso della Mail del Gruppo

La mail verrà utilizzata principalmente come mezzo di comunicazione con cui rivolgersi a committente e proponente. Ogni membro del gruppo ha accesso, previa autenticazione, alla mail del gruppo.

2.2 Uso dello spazio web in Google Groups

Lo spazio web in Google Groups raggiungibile tramite il seguente URL: $http://groups.google.com/group/happycodeinc$ è riservato ai soli membri del gruppo. Dovrà essere utilizzato per:

- Gestire discussioni su argomenti non chiari;
- Condividere pagine statiche contenenti risorse pertinenti i linguaggi e gli strumenti utilizzati.
- Condividere file utili allo sviluppo del progetto, come manuali e tutorial.

2.3 Comportamento Generale

Ad ogni componente del gruppo verrà richiesto di partecipare alla stesura dei documenti e del codice. Il documento verrà creato inserendo l'elenco dei capitoli, delle sezioni e delle sottosezioni all'interno di un modello in latex. Una volta creato il documento dovrà poi essere caricato nel repository SVN, nel solo formato `.tex` .

2.4 Condivisione, archiviazione e versionamento dei file

Per garantire la correttezza di ogni documento e file condiviso, a fronte di cambiamenti (anche simultanei) da parte di tutto il gruppo, verrà utilizzato un server SVN con opportuni client grafici (RapidSVN, KdeSvn). Verrà garantita in questo modo l'atomicità di ogni aggiornamento inviato al repository.
Il server Subversion utilizzato dal gruppo (versione 1.4.0) è messo a disposizione da Google all'indirizzo *http://code.google.com/p/br-jsys/*.

Ogni membro del gruppo è tenuto a utilizzare il server SVN , eseguendo un *update* prima di ogni modifica ed eseguendo un *commit* al termine della stessa.

Capitolo 3

Norme di Documento

3.1 Nomenclatura

In questo capitolo verranno illustrate le norme per la nomenclatura dei file sorgente. I nomi dei file dovranno essere significativi rispetto al loro contenuto. Inoltre:

- Potranno essere usati solo caratteri dell'alfabeto inglese dalla "a" alla "z" , minuscole e maiuscole;
- Non dovranno essere utilizzati catteri speciali;
- Non dovranno essere utilizzati catteri accentati;
- I nomi utilizzati non dovranno contenere spazi;
- Se il nome da utilizzare è composto da piu' parole, queste dovranno essere disposte come in questo esempio:
  "EsempioDiNomeFile"
- Alla fine del nome del file dovrà esserci il codice identificativo dell'ultima versione del documento. Esempio di un nome corretto:
  "NormeDiProgetto.0.1"

3.2 Impaginazione documenti

Questa seziona descriverà l'impaginazione dei documenti ad uso interno ed esterno. Di seguito verrà fornita la lista delle regole da seguire per la corretta stesura di un documento.

3.2.1 Struttura documento

Ogni documento sarà composto di queste parti:

1. Una prima pagina con il logo del gruppo, titolo del documento, versione e data ultima modifica;
2. Una seconda parte composta da tre tabelle contenti:
   - Data creazione (anno,mese,giorno),versione, stato del documento, redazione,revisione,approvazione.
   - Lista di Distribuzione
   - Diario delle modifiche (dalla versione più recente fino alla prima stesura del documento)
3. L'indice dei contenuti;
4. I contenuti del documento;

È fornito un modello di documento generale ModelloUnicoDiLayout.tex, nel quale sono state applicate le regole date precedentemente. Il modello va utilizzato per la prima stesura di ogni nuovo documento.

3.2.2 Intestazione/Piè di pagina

L'intestazione e il piè di pagina verranno generati automaticamente corretti dal file .tex con i relativi loghi, il numero della pagina e i titoli dei capitoli.

3.2.3 Carattere Testo e Titoli

Il formato (dimensione e font utilizzato) sono quelli standard per un documento latek e sono parte del Modello.

3.2.4 Immagini

Le immagini inserite nel corpo del documento dovranno essere necessariamente salvate nella cartella pics. Per garantire la massima qualità dell'immagine ogni immagine dovrà essere nel formato vettoriale *.eps* (Encapsulate PostScript).

Capitolo 4

Norme di codifica

4.1 File sorgente

Ogni file sorgente dovrà contenere una sola classe pubblica o interfaccia. I file sorgente dovranno avere il seguente ordine:

- Commenti iniziali;
- Statements di Import e dichiarazioni di Package;
- Dichiarazioni di classi e interfacce.

4.1.1 Commenti iniziali

Tutti i file sorgente dovranno inziare con un commento che elenca il nome della classe, informazioni sulla versione e la data dell'ultima modifica come nell'esempio:

```
/*
* Nome classe
*
* Informazioni Versione
*
* Data Ultima Modifica
*/
```

4.1.2 Statements di Package e di Import

La prima linea non di commento di un file dovrà essere uno statement di Package. Seguiranno poi gli statement di import come nell'esempio:

```
package java.awt;
import org.w3c.dom;
```
Tutti i package e import dovranno essere scritti in minuscolo.

4.1.3 Dichiarazioni di Classi e di Interfacce

Prima di tutto verranno dichiarate le varibili statiche nel seguente ordine: pubbliche, protette, quelle a livello package e poi le private. Successivamente le variabili di istanza sempre nello stesso ordine. Solo infine i costruttori seguiti dai metodi. I metodi dovranno essere raggrupati secondo la funzionalità e non secondo l'accessibilità. I nomi delle classi dovranno essere al singolare e iniziare con la lettera maiuscola.

4.1.4 Variabili

Dovranno essere scritte al singolare e in minuscolo. Nel caso di un nome di variabile composto da più parole, la prima lettera delle parole verrà scritta in maiuscolo, ad eccezione della prima parola che sarà sempre in minuscolo (compresa la prima lettera).
Esempio: dataNascita
Le variabili non dovranno avere ambiguità nei nomi.

4.1.5 Metodi

Seguono lo stesso standard adottato per le variabili.

4.2 Indentazione

Come unità di indentazione dovranno essere usati 4 spazi.

4.2.1 Lunghezza della linea

Una linea di codice non dovrà superare 80 caratteri.

4.2.2 Interruzione delle linee

Quando un'espressione non sta su una singola linea, la si dovrà interrompere seguendo le seguenti regole:

- Interruzione dopo una virgola;
- Interruzione prima di un operatore;
- La nuova linea dovrà essere allineata con l'inizio dell'espressione allo stesso livello nella linea precedente;

- Se le regole suddette portano a codice confuso oppure a codice che troppo a ridosso del margine destro si dovrà invece usare una indentazione con 8 spazi.

4.3 Commenti

Ci potranno essere due tipi di commenti: di implementazione e di documentazione. Quest'ultimi possono essere a loro volta suddivisi in: commenti a blocco, a linea singola e commenti alla fine della linea.

- I commenti di implementazione, delimitati da `/*...*/` e `//`, sono commenti riguardanti il codice nell'ottica della sua particolare implementazione.
- I commenti di documentazione, delimitati da `/**...*/`, descrivono la specifica del codice astraendo dall'implementazione.

4.3.1 Commenti a blocco

I commenti a blocco dovranno essere usati per descrivere files, metodi, strutture dati e algoritmi. Dovranno essere usati all'inizio di ogni file e prima di ogni metodo; potranno inoltre essere usati anche in altre parti del file (ad esempio all'interno di un metodo). All'interno di una funzione o di un metodo dovranno essere indentati allo stesso livello del codice che descrivono. Questo commento dovrà inoltre essere preceduto da una linea bianca per poterlo separare dal resto del codice.

Esempio:
```
/*
* Questo  un commento a blocco
*/
```

4.3.2 Commenti a linea singola

Questi commenti dovranno apparire in una linea singola ed essere usati per brevi commenti, indentandoli al livello del codice che segue. Se un commento non può essere scritto su una linea singola allora si dovrà usare un commento a blocco. Dovrà essere preceduto da una linea bianca.

Esempio:
```
if (condition) {
    /* Handle the condition.  */
    ......
```

```
}
```

4.3.3 Commenti alla fine della linea

Il delimitatore di commento "`//`" potrà commentare una linea completa oppure solo una parte di essa. Non dovrà essere utilizzato su linee consecutive per commenti di testo. Potrà però essere utilizzato su linee consecutive per eliminare (commentando) sezioni di codice.

4.3.4 Commenti di documentazione

I commenti di documentazione dovranno descrivere classi, interfacce, costruttori, metodi e campi dato. Ogni commento di documentazione dovrà essere situato all'interno dei delimitatori `/**...*/`. Ci dovrà essere un solo commento per ogni classe, interfaccia o membro. Non dovranno inoltre essere posizionati all'interno del blocco di definizione di un metodo o di un costruttore. Questo tipo di commento dovrà apparire subito prima della dichiarazione:

Esempio:

```
/**
* Commento di documentazione
*/
public class Esempio {
    .......
}
```

4.4 Dichiarazioni

Per facilitare i commenti ci dovrà essere una sola dichiarazione per linea. Non si dovranno mettere tipi differenti sulla stessa linea. Le variabili locali dovranno essere inizializzate dove sono dichiarate se ciò è possibile. Le dichiarazioni dovranno essere posizionate solamente all'inizio dei blocchi di codice. L'unica eccezione che potrà essere fatta per gli indici dei cicli *for*; in questo caso la dichiarazione potrà essere fatta nello statement del ciclo *for*. Non ci dovranno inoltre essere dichiarazioni che rendono invisibili altre dichiarazioni a livello più alto; non si dovrà in pratica usare un nome di variabile già usato in un blocco soprastante.

4.4.1 Dichiarazioni di Classi o di Interfacce

Nella codifica di classi ed interfacce dovranno essere seguite le seguenti regole:

- Non ci dovranno essere spazi tra il nome di un metodo e le parentesi che apre la sua lista dei parametri;
- La parentesi graffa che apre un blocco di codice dovrà apparire alla fine della stessa linea dello statement di dichiarazione;
- La parentesi graffa che chiude un blocco di codice dovrà apparire su una linea singola a sè stante, indentata in modo da potersi allineare alla corrispondente parentesi di apertura del blocco, ad eccezione del caso in cui lo statement sia vuoto;
- La parentesi di chiusura apparirà subito dopo quella di apertura;
- I metodi dovranno essere separati da una linea bianca.

4.5 Statements

4.5.1 Statements semplici

Ogni linea dovrà contenere un solo statement, per facilitare la lettura e la comprensione del codice, nonchè la sua commentazione.

4.5.2 Statements Composti

Uno statement composto conterrà una lista di statements racchiusi tra parentesi graffe. Per gli statements composti valgono le seguenti regole:

- Il livello di indentazione di ogni statement racchiuso tra parentesi grafe deve essere aumentato di uno rispetto al livello dello statement che lo contiene.
- La parentesi graffa di apertura deve essere alla fine della linea che inizia lo statement composto; la parentesi graffa di chiusura deve essere l'unico carattere di una linea indentata corrispondentemente all'inizio dello statement composto.

Ad esempio:

```
statementComposto {
    statement;
    ...
```

```
    ....
}
```

Le parentesi dovranno essere usate per racchiudere l'elenco degli statement, anche se ve ne fosse uno solo.

4.5.3 Statements di Return

Uno statement di return non dovrà usare parentesi a meno che ciò non renda il valore di ritorno in qualche maniera più ovvio.

4.5.4 Statements if, if-else

Gli statement del tipo if-else dovranno avere la seguente forma:

```
if (condizioni) {
    statement;
    ...
}
```

```
if (condizioni) {
    statement;
    ...
} else {
    statement;
    ...
}
```

Da notare che gli statement if devono sempre usare le parentesi graffe.

4.5.5 Statements for

Uno statement *for* dovrà avere la seguente forma:

```
for (inizializzazioni; condizioni; incrementi){
    statement;
    ...
}
```

4.5.6 Statements while e do-while

Uno statement *while* dovrà avere la seguente forma:

```
while (condizioni){
```

```
    statement;
    ...
}
```

Uno statement *do-while* deve avere la seguente forma:

```
do {
    statement;
    ...
} while (condizioni);
```

4.5.7 Statements switch case

Uno statement *switch* dovrà avere la seguente forma:

```
switch (condizioni) {
case A:
    statement;
    ...
    break;
case B:
    statement;
    ...
    break;
default:
    statement;
    ...
    break;
}
```

Uno statement *switch* deve includere un caso di *default*.

4.5.8 Statements try-catch

Uno statement *try-catch* dovrà avere il seguente formato:

```
try {
    statement;
    ...
} catch (ExceptionClass e) {
    statement;
    ...
} finally {
    statement;
```

```
    ...
}
```

4.6 Linee e spazi bianchi

Le linee bianche dovranno essere utilizzate per migliorare la leggibilità del codice. Nei seguenti casi dovranno sempre essere usate due linee bianche:

- Tra sezioni di un file sorgente;
- Tra definizioni di classi ed interfacce.

Nei seguenti casi dovrà essere invece usata una una sola linea bianca:

- Tra metodi;
- Tra le variabili locali in un metodo e il suo primo statement;
- Prima di un commento a blocco o un commento a linea singola;
- Tra sezioni logiche all'interno di un metodo in modo da migliorare la leggibilità.

Nelle seguenti situazioni dovranno essere usati spazi bianchi:

- Una parola chiave seguita da una parentesi dovrà essere separata da uno spazio bianco. Da notare che uno spazio bianco non deve essere usato tra il nome di un metodo e la sua parentesi di apertura;
- Uno spazio bianco dovrà apparire dopo le virgole nelle liste degli argomenti;
- Le espressioni in uno statement *for* dovranno essere separate da spazi bianchi.

4.7 Standard di denominazione di entità e relazioni

4.7.1 Entità

Le entità componenti il "BR-jsys" verranno identificate usando una notazione al singolare, in carattere minuscolo e con l'iniziale di ogni parola in maiuscolo. Il nome dei campi dato e dei metodi verranno descritti con la stessa notazione, ad eccezione del primo carattere della prima parola che compare in carattere minuscolo.

4.7.2 Relazioni

Le relazioni verranno descritte in minuscolo, utilizzando il nome della tabella da cui se ne prende l'ID preceduto da "fk_".