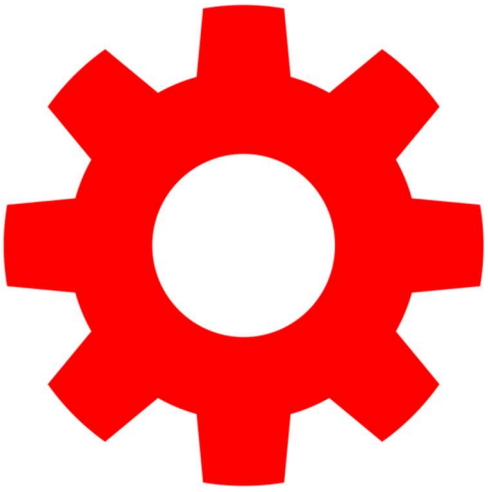
Università degli Studi di Salerno
Corso di Ingegneria del Software

ePADs
Object Design Document - ODD
Versione 2.0



INDICE

1. Introduzione

1.1 Object design trade-off

1.1.1 Portabilità contro efficienza

La portabilità del sistema epads è garantita dalla scelta del linguaggio di programmazione Java. Lo svantaggio dato da questa scelta è nella perdita di efficienza introdotta dal meccanismo della macchina virtuale Java. Tale compromesso è accettabile per i numerosi supporti forniti dal linguaggio Java.

1.2 Interface Documentation Guidelines

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice.

1.2.1 File Java

Ogni file sorgente deve contenere una sola classe o interfaccia pubblica. Ogni file deve contenere nel seguente ordine:

- Commenti per una migliore comprensione
- Dichiarazione del package
- Sezione import
- Dichiarazione di interfaccia o classe:
  - Attributi pubblici
  - Attributi privati
  - Attributi protetti
  - Costruttori
  - Altri metodi
- Classi interne

È previsto l'utilizzo di commenti JavaDoc.

1.2.2 Naming

L'utilizzo di convenzioni sui nomi rendono il programma più leggibile e comprensibile da tutti i membri del team. In particolare secondo il modello del codice programmato, è auspicabile che tutti siano in grado di intervenire su una qualsiasi linea di codice.

Classi e interfacce

I nomi delle classi sono nomi (composti anche da più parole) la cui iniziale è in maiuscolo. Ogni parola che compone un nome ha l'iniziale in maiuscolo.

I nomi delle classe devono essere semplici e descrittivi. Evitare l'uso di acronimi e abbreviazioni per i nomi delle classi.

Nel caso una o più classi incarnino design patterns noti è consigliato l'utilizzo di suffissi (inglesi) che richiamano lo specifico componente del design pattern (esempio: DatabaseAdapter, GiocatoreFactory, …).

E' consigliato l'uso della lingua italiana per i nomi, fatta eccezione per nomi inglesi di uso comune (esempio: TestingClass, …).

Metodi

I metodi devono essere verbi (composti anche da più parole) con iniziale minuscola.

Costanti

In accordo con le convenzioni suggerite dalla Sun, i nomi di constanti vengono indicati da nomi con tutte le parole in maiuscolo. Le parole vengono separate da underscore "_".

Ad esempio:

```
staticfinalint MAX_LENGHT = 24;
```

1.2.3 Uso dei commenti

E' permesso l'utilizzo di due tipi di commenti:

Commenti Javadoc (aree di testo compresa tra il simbolo /** e */)

Commenti in stile C (righe delimitate da //)

L'utilizzo dei commenti Javadoc è suggerito prima della dichiarazione di:

classi e interfacce

costruttori

metodi di almeno 3 righe di codice

variabili di classe

Ogni commento, compreso tra il simbolo /** e */, deve specificare le funzionalità e le specifiche del codice, senza esplicitare dettagli legati all'implementazione, in maniera tale da rendere leggibile tale documentazione anche a sviluppatori che non posseggono l'implementazione.

I commenti di Javadoc consentono la generazione automatica della documentazione del codice, attraverso l'utilizzo di appositi tools.

Il commenti stile C, ovvero le linee di codice precedute da //, sono utilizzati all'interno dei metodi, al fine di descrivere in maniera concisa e sintetica branch, cicli, condizioni o altri passi del codice.

1.2.4 Altre regole di stile

E' importante che vengano seguite anche ulteriori "regole di stile", al fine di produrre codice chiaro, leggibile e privo di errori.

Tra queste "regole di stile" elenchiamo le seguenti:

- I nomi di package, classi e metodi devono essere nomi descrittivi, facilmente pronunciabili e di uso comune
- Evitare l'utilizzo di abbreviazioni di parole
- Utilizzare, dove possibile, nomi largamente in uso nella comunità informatica (ie: i nomi dei design patterns)
- Preferire nomi con senso positivo a quelli con senso negativo
- Omogeneità dei nomi all'interno dell'applicazione
- Ottimizzazioni del codice non devono comunque inficiare la leggibilità dello stesso. Se si è costretti a sviluppare codice poco leggibile, perché le estreme prestazioni sono indispensabili è necessario documentarlo adeguatamente.
- Evitare la scrittura di righe di codice più lunghe di 80 caratteri e di file con più di 2000 righe
- È consigliato, per l'indentazione, l'utilizzo di spazi al posto dei "tab". Questo rende il codice ugualmente leggibile su tutti gli editor (alcuni editor convertono in automatico le tabulazioni in 4/6 spazi)
- È consigliato l'utilizzo di nomi in italiano. Tuttavia è consigliato l'utilizzo di termini inglesi laddove si tratta di uso comune o nel caso, molto comune, di termini comunemente usati nella loro versione inglese. E' di fondamentale importanza l'utilizzo di un dizionario dei nomi unico per tutto il progetto, che tutti i programmatori saranno tenuti a seguire.
- È consigliato l'utilizzo di nomi inglesi anche nel caso si adoperino termini della libreria standard di Java (ie: OptimizedList anziché ListaOttimizzata)
- Si consiglia l'utilizzo di parti standard dei nomi in casi come:
- Classi astratte, suffisso Abstract- (ie: AbstractProdotto)
- Design patterns (ie: se si usa l'MVC utilizzare ListModel)
- Accezioni terminanti per Exception (ie: UtenteNonTrovatoException)
- Altre situazioni analoghe
- I nomi delle interfacce segue le regole standard dei nomi. E' sconsigliato usare il prefisso o suffisso "Interface"
- È consigliato l'utilizzo di suffissi "standard" come "get", "set", "is" o "has" in inglese

- È possibile scrivere dichiarazioni di metodi e classi in due righe, se eccessivamente lunghi
- Evitare la notazione ungherese. La notazione ungherese, che prevede l'utilizzo di prefissi per descrivere il tipo di dato, non dovrebbe essere utilizzata. La motivazione è semplice: la notazione ungherese va bene per linguaggi che hanno tipi semplici, e dove è possibile creare un vocabolario di prefissi limitato. In linguaggi OOP i tipi primitivi hanno un uso più limitato, mentre sono gli oggetti a farla da padrone.
- Dichiarare le variabili ad inizio blocco, sia questo un metodo o una classe, in modo da raccogliere in un unico punto tutte le dichiarazioni.
- Utilizzare la dichiarazione per definire una sola variabile – evitando più dichiarazioni sulla stessa riga
- L'inizializzazione delle variabili deve essere eseguita in fase di dichiarazione, impostando un valore di default o il risultato di un metodo. Se proprio ciò non è possibile, in quanto il valore da impostare è il risultato di una elaborazione compiuta nel metodo stesso, inizializzare la variabile appena prima del suo utilizzo
- Allineare la dichiarazione delle variabili per renderle più leggibili, strutturandole in blocchi omogenei per contesto (e non per tipo di dato)
- Nel caso di algoritmi troppo complessi, eseguire un refactoring per separarlo in diversi sotto-metodi più semplici.
- I cicli devono seguire le seguenti regole:
- Per le variabili, utilizzare l'area di visibilità più stretta possibile, dichiarando le variabili appena prima del loro utilizzo.
- Per le chiamate a metodo non utilizzare spazi dopo il nome del metodo.

1.3Definizioni, acronimi e abbreviazioni

ACID	Atomicità,Consistenza,Isolamento e Durabilità
RAD	Requirements Analysis Document
RAD	Documento di Analisi dei requisiti
SDD	System Design Document

1.4Riferimenti

- RAD ePADs documento analisi dei requisiti

- SDD ePADs documento di system design

- dispense dei corsi seguiti in precedenza

1.5Overview

Nelle sezioni successive sarà descritta l'architettura del sistema e le sue componenti principali. Saranno esposte le tipologie di utenza ed i comportamenti del sistema previsti per ogni tipologia, nonché le funzionalità delle componenti invocate.

Saranno inoltre descritti i requisiti minimi per la macchina che ospiterà il sistema e le politiche di sicurezza adottate dal sistema.

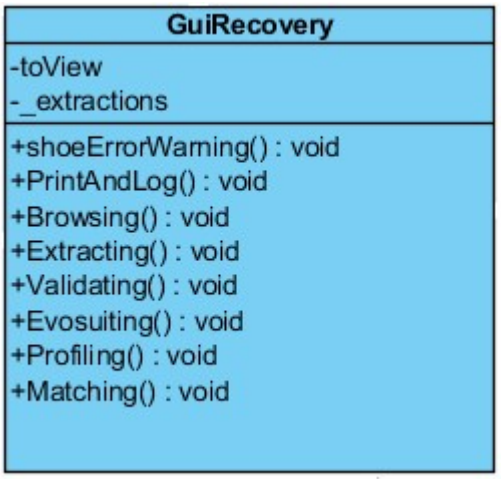
2.Interfaccedelle classi

Si procede all'analisi dettagliata delle piccole classi implementate nel sistema.

L'analisi serve ad evidenziare le interfacce di interazione utilizzate nella progettazione del software.

2.1Descrizione delle classi

2.1.1 **GuiRecovery**



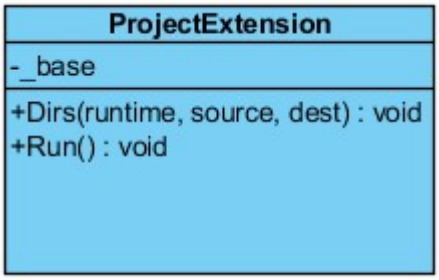
La classe recupera le checkoptions inserite dal developer e svolge le fasi di estrazione dei design pattern.

- Private toView as Map<String, Boolean>
  Rappresenta le checkOptions inserite dal developer
- Private _extractions as Extractions
  La variabile che si carica le definizioni dei pattern
- Public PrintAndLog() as void
  Stampa a video e fa il log
- Public Browsing() as void

  vengono recuperate informazioni quali nomi delle classi, tipo delle classi, nomi dei metodi e delle variabili, tipo dei metodi e delle variabili, parametri dei metodi, associazione tra le classi.
- Public Extracting() as void
  estrae le informazioni strutturali dal codice OO in input per il recovery delle istanze di Design Pattern, come le relazioni tra le classi, le dichiarazioni dei metodi e le invocazioni dei metodi
- Public Validating() as void
  vengono creati due nuovi file di testo, uno per le istanze di design pattern strutturali e uno per le istanze di design pattern comportamentali.
- Public Evosuiting() as void
  si generano automaticamente test suite
- Public Profiling() as void
  si generano dei file probe in grado di iniettare codice Java all'ingresso e all'uscita dei metodi
- Public Matching() as void
  la sequenza di metodi viene confrontata con il comportamento dei Design Pattern specificato nel sequence diagram presente nella Design Pattern Library.

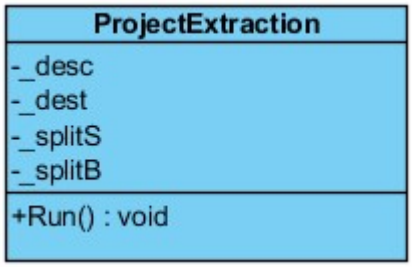
2.1.2 **ProjectExtension**

Questa è classe che viene istanziata per effettuare la fase di browsing.



- private _base as ArrayList<BaseExtension>
  variabile che contiene le informazioni sulle cartelle che il tool userà
- Public Dirs(Runtime,source,dest) as void
  Setta le varie destinazioni dove il tool salverà i file
- Public Run() as void
  Effettua la fase di browsing.

2.1.3 **ProjectExtraction**

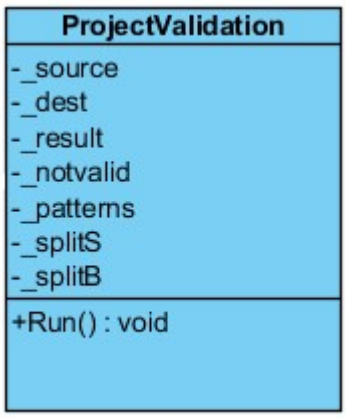


Questa è la classe che viene istanziata dove vengono cercati tutti i design pattern.

- Private _desc as String
  Nome del file che contiene i primi design pattern estratti
- Private _dest as String
  Nome del file che contiene i design pattern validati
- Private _splitS as String
  Nome del file che contiene i design pattern strutturali estratti,splittati
- Private _splitB as string
  Nome del file che contiene i design pattern comportamentali estratti,splittati
- Public Run() as void
  Estrae tutte le informazioni dei design pattern istanziati nella fase precedente

2.1.4 **ProjectValidation**

Questa è la classe che viene istanziata dove vengono splittati i design patten sia comportamentali che strutturali



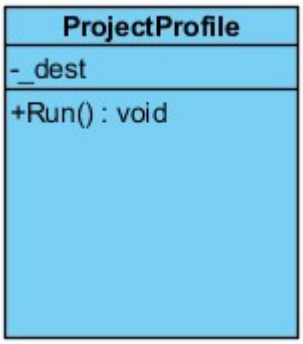
- Private _source as String
  Nome del file che contiene i primi design pattern estratti
- Private _dest as String
  Nome del file che contiene i design pattern validati
- Private _splitS as String
  Nome del file che contiene i design pattern strutturali estratti,splittati
- Private _splitB as string
  Nome del file che contiene i design pattern comportamentali estratti,splittati
- Private _result as String
  Nome del file HTML che contiene i risulati finali
- Private _notvalid as String

Nome del file dei design pattern non validi

- Private _patterns as String
  Nome del file che contiene tutti i pattern
- Public Run() as void
  Esegue lo split dei design pattern nei rispettivi file di testo

2.1.5 **ProjectProfile**

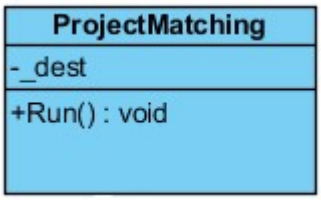
Questa è la classe che viene istanziata per eseguire la fase di profiling



- Private _dest as String
  Nome del file che contiene i pattern comportamentali
- Public Run() as void
  Vengono creati dei file probe dove è possibile iniettare codice all'interno all'ingresso e all'uscita dei metodi

2.1.6 **ProjectMatching**

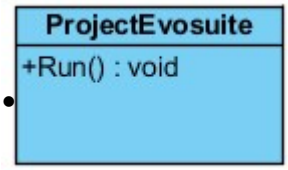
Questa è la classe che viene istanziata per eseguire la fase di matching



- Public Run() as void
  Per ogni canditato probe si effettua il matching in fase di esecuzione
- Private _dest as String
  Nome del file che contiene i design pattern che hanno passato il matching

2.1.7 **ProjectEvosuiting**

Questa è la classe che viene istanziata per eseguire l'evosuite test



- Public Run() as void

si generano automaticamente test suite

3. GLOSSARIO

Termini	Descrizione
ODD	Object Design Document
SDD	System Design Document