

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Ivan Rizzo  Domenico Liguori  Marco Feoli |  |  | Francesco Randillo  Fabrizio Pecoraro  Antonio Memoli | |  |  |  |

ODD Object Design Document

Dashboard Dipartimento Informatica

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Cambiamenti | Autori |
| 10/12/2015 | 0.1 | Aggiunta: Object Design; Descrizione librerie; Convenzioni. | Tutti |
| 11/12/2015 | 0.2 | Aggiunta: Design Patterns; Packages. | Marco Feoli, Domenico Liguori, Ivan Rizzo. |
| 11/12/2015 | 1.0 | Revisione | Domenico Liguori |
| 14/01/2016 | 2.0 | Revisione | Tutti |

Sommario

1. [1.Introduzione 3](#_Toc437636265)

[1.1. Trade-off](#_Toc437636266)

[1.2. Componenti off-the-shelf](#_Toc437636267)

[1.3. Linee guida per la documentazione dell’interfaccia](#_Toc437636268)

[1.4. Design pattern](#_Toc437636269)

[1.5. Definizioni, acronimi e abbreviazioni](#_Toc437636270)

[1.6. Riferimenti](#_Toc437636271)

1. [2.Packages 9](#_Toc437636272)
2. [3.Interfacce delle classi 9](#_Toc437636273)

# Introduzione

## Trade-off

### Comprensibilità vs costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

### Interfaccia vs Easy-use

La Dashboard è molto semplice e di facile utilizzo poiché ha un interfaccia chiara e intuitiva.

### Memoria vs efficienza

Si è preferita la memoria anziché le prestazioni perché salvare risultati parziali o complessivi avrebbe causato il mantenimento di informazioni incompleti, difficili da integrare e interpretare. La principale difficoltà che ci ha portato a questa scelta è relativa alle date. Infatti, se avessimo salvato i dati in modo tale che tutte le richieste dell’utente possano essere soddisfatte avremmo dovuto memorizzare tutte le possibili combinazioni dei dati di cui disponiamo. Rischiamo dunque di effettuare operazioni onerose e non si ha mai la certezza che il risultato sia esatto.

Si preferisce quindi mantenere in memoria solo i dati "grezzi", cioè dati non processati, e in funzione di essi calcolare i risultati per restituire all’utente grafici e misure nei periodi da lui richiesti.

## Componenti off-the-shelf

Per il progetto software che si vuole realizzare facciamo uso di componenti***off-the-shelf****,* che sono componenti software disponibili sul mercato per facilitare la creazione del progetto.

Per il sistema che si vuole realizzare ci interessa un framework per applicazioni web e librerie per la gestione di grafici.

Il framework che andremo ad utilizzare è il Bootstrap, che è un framework open source che contiene una raccolta di strumenti [liberi](https://it.wikipedia.org/wiki/Software_libero) per la creazione di [siti](https://it.wikipedia.org/wiki/Sito_web) e [applicazioni](https://it.wikipedia.org/wiki/Applicazione_web) per il [Web](https://it.wikipedia.org/wiki/Web). Essa contiene [modelli](https://it.wikipedia.org/wiki/Template) di progettazione basati su [HTML](https://it.wikipedia.org/wiki/HTML) e [CSS](https://it.wikipedia.org/wiki/CSS), sia per la [tipografia](https://it.wikipedia.org/wiki/Tipografia), che per le varie componenti dell'interfaccia, come [moduli](https://it.wikipedia.org/wiki/Form), bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di [JavaScript](https://it.wikipedia.org/wiki/JavaScript).

Per la creazione e la manipolazione dei grafici usiamo invece le librerie Highchart e una sua variante Highstock.

Highcharts è una libreria commerciale (non è freeware per scopi professionali) ma è scaricabile gratuitamente dal sito ufficiale ([www.highcharts.com](http://www.highcharts.com)) che consente di creare grafici di vario genere comprensivi delle opzioni di navigazione più sofisticate, come intervalli di date predefinite, selettore data, e altro, in più ci mette a disposizione un gran numero di tipi di grafici : Istogrammi, grafici azionari, a barre, linee, torta, dispersione, area, bolle …

Si basa esclusivamente su tecnologie native del browser e non richiede plugin lato client come Flash o Java. Inoltre non c’è bisogno di installare nulla su server. Per funzionare ha bisogno solo di due file JS da eseguire: il nucleo highstock.js e un framework a scelta tra jQuery, MooTools o Prototype.

Per convertire il foglio Excel in cui sono contenuti i dati immessi dall’utente abbiamo bisogno di un parser che converte il foglio Excel in JSON, in questo modo possiamo gestire i dati più facilmente con Javascript ed è possibile inserirli in un database.

Il parser che andremo ad utilizzare è JS-XLXS ed è scaricabile da: https://github.com/SheetJS/js-xlsx.

## Linee guida per la documentazione dell’interfaccia

* Prima dell’implementazione della logica di un qualsiasi indicatore è opportuno sottomettere alle Project Manager, l’algoritmo che si intende seguire per l’implementazione di questi, in modo che eventuali correzioni della logica da seguire possano essere apportate prima di imbattersi nella sintassi degli strumenti scelti.
* Ogni metodo e ogni file devono essere preceduti da un commento, o più precisamente da una documentazione che riporti l’obiettivo che si vuole e deve raggiungere con il nome/i dell’autore/i. Inoltre bisogna commentare, giustificare delle decisioni particolari o dei calcoli.
* La convenzione che deve essere adottata da tutti i team member per quanto riguarda i nomi delle variabili, è la nota Camel Notation.

**Organizzazione dei file**

* + Ogni file deve essere:
* Sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un’unica funzionalità che persegue. Ogni pagina della Dashboard (login, inserimento file, visualizzazione indicatori, etc.) deve essere implementata in file separati;
* Diviso in più file, se raggiunge una lunghezza tale da divenire difficile da leggere e comprendere;
* Se sono presenti diversi script per il calcolo degli indicatori è opportuno organizzarli in una apposita cartella.
* La convenzione per quanto riguarda i nomi dei file, delle operazioni e delle variabili è quella di avere nomi evocativi, ma soprattutto in lingua italiana. Le uniche eccezioni sono: DASHBOARD, GET e SET. I termini GET e SET sono stati inseriti nella nostra terminologia data la familiarità con il linguaggio di programmazione JAVA e soprattutto perché anche la migliore traduzione in italiano non regge il confronto.
* Organizzare in una cartella i file delle libreria usate e le altre risorse scaricate necessarie per lo sviluppo del progetto.

**Spostamento di linee**

* Quando un’espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:
  + - Interrompere la linea dopo una virgola;
    - Interrompere la linea prima di un operatore;
    - Preferire interruzioni di alto livello rispetto ad interruzioni di basso livello (interrompere laddove non si interrompe un discorso logico, discorso valido soprattutto per le formule es. (3+4) \* 2 interrompere prima della moltiplicazione senza spezzare gli operandi in parentesi);
    - Allineare la nuova linea con l’inizio dell’espressione nella linea precedente;
    - Se le regole precedenti rendono il codice più confuso o il codice è troppo spostato verso il margine destro, utilizzare solo otto spazi di indentazione.

**Indentazione**

* L’indentazione deve essere effettuata con un TAB e qualunque sia il linguaggio usato per la produzione di codice, ogni istruzione deve essere opportunamente indentata.

*Es.*

<html>

<head>

</head>

<body>

</body>

</html>

Deve essere sostituita da:

<html>

<head>

</head>

<body>

</body>

</html>

Questa pratica deve essere usata soprattutto per le istruzioni FOR, IF.

È buona pratica scendere di livello.

**Inizializzazione**

Inizializzare le variabili locali nel punto in cui sono state dichiarate a meno che il suo valore iniziale non dipenda da un calcolo che occorre eseguire prima.

**Posizione**

Mettere le dichiarazioni all’inizio dei blocchi. Non aspettare di dichiarare le variabile al loro primo uso: può confondere il programmatore inesperto e impedire la portabilità del codice dentro lo scope. L’unica eccezione a questa regola sono gli indici dei cicli for che in Java possono essere dichiarati nell’ istruzione stessa. Evitare dichiarazioni locali che nascondono dichiarazioni a più alto livello. Ad esempio, non dichiarare una variabile con lo stesso nome in un blocco interno.

**Parentesi**

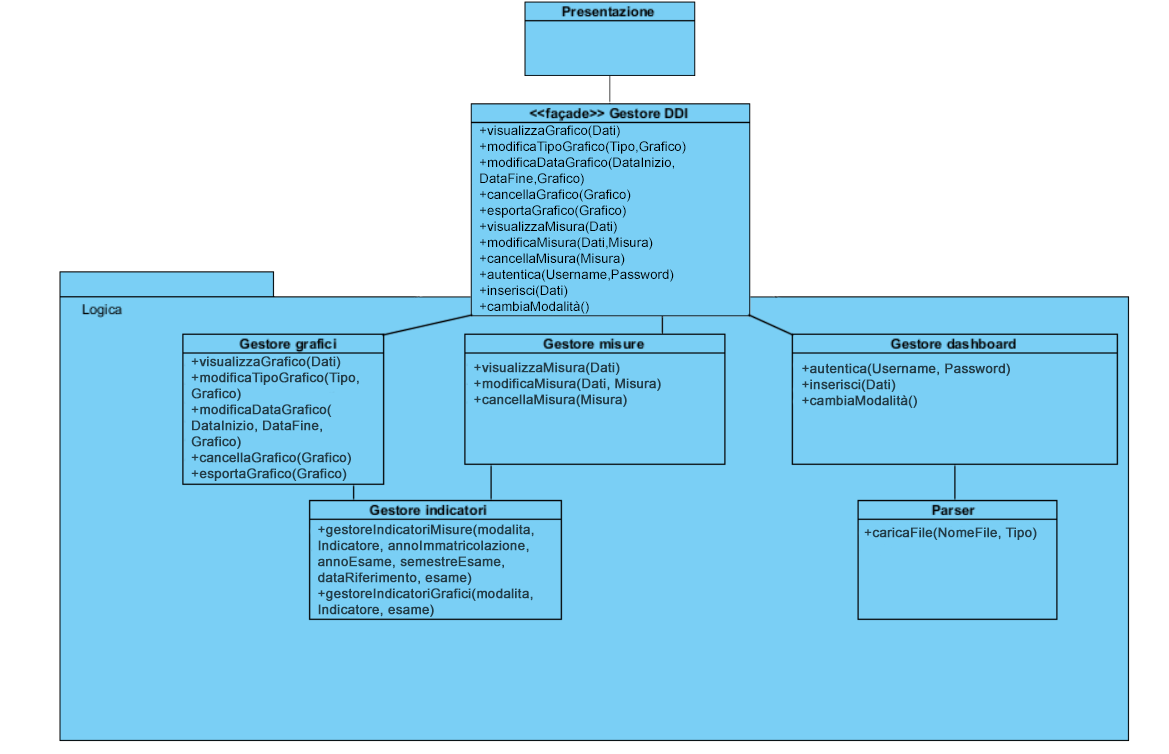
A prescindere dalle istruzioni che seguono un IF, è necessario, laddove ci fosse anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe.

Ogni tag di apertura deve essere necessariamente seguito dall’apposito tag di chiusura (eccetto i tag self-closing).

Una convenzione importante, per quanto riguarda l’inserimento di numeri o di valori costanti, è quella di non usare una codifica fissa (hard coding) che è fortemente sconsigliata ma di associare sempre il valore ad una variabile o semplicemente definire una macro che può essere richiamata da eventi ed essere parametrizzata. In questo modo si facilita la modifica, sostituendo solo il valore della variabile o macro, in un unico posto.

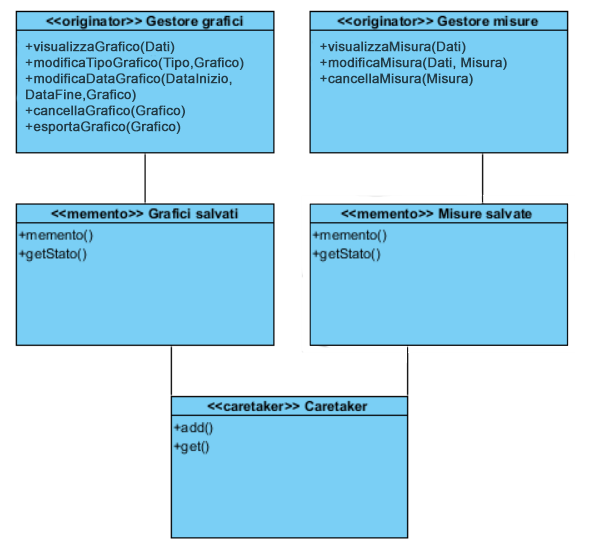
## Design pattern

**Façade Pattern**



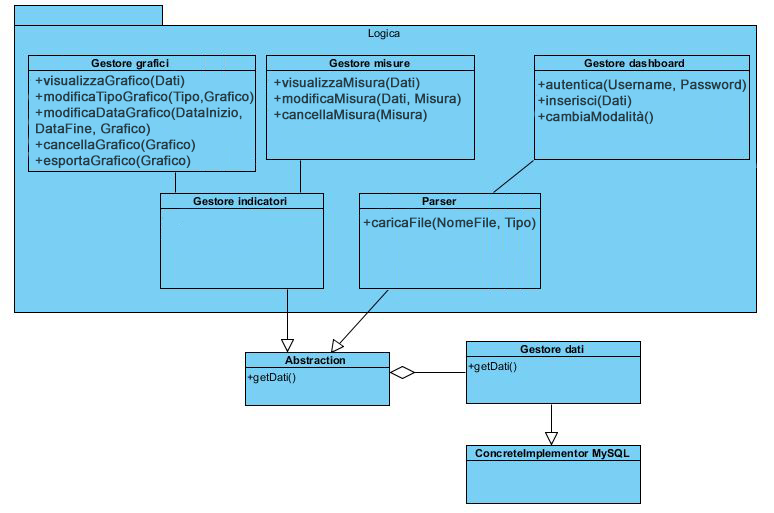
DDI fa uso del Façade Pattern per definire un’unica interfaccia a livello di logica che permette all’utente di interagire, attraverso l’interfaccia, con le funzionalità del sistema vedendole come un unico sistema.

**Memento Pattern**



DDI fa suo del Memento Pattern perché abbiamo bisogno di mantenere storia di stati precedenti del sistema: in particolare per il cambio di modalità tra triennale e magistrale.

**Bridge pattern**



DDI fa uso del Bridge Design Pattern perché abbiamo bisogno di utilizzare un’unica interfaccia per diversi accessi allo storage: offrendo un’unica interfaccia si garantisce che l’eventuale cambio di implementazione del database usato comporta la modifica solo a una componente e non a svariate componenti del sistema.

## Definizioni, acronimi e abbreviazioni

**DBMS:** Data Base Management System.

**Dashboard:** Piattaforma informatica interattiva, che consente l’uso di strumenti facilmente accessibili.

**Off-The-Shelf:** Servizi esterni di cui viene fatto utilizzo da terzi.

**Framework:** Software di supporto allo sviluppo web.

**Highcharts:** Insieme di librerie per la gestione di grafici.

**Plugin:** Un programma non autonomo che interagisce con un altro programma per ampliarne le funzioni.

**HTML:** Linguaggio di mark-up per pagine web.

**CSS:** Linguaggio usato per definire la formattazione di pagine web.

**JavaScript:**  [Linguaggio di scripting](https://it.wikipedia.org/wiki/Linguaggio_di_scripting) [orientato agli oggetti](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti) e agli [eventi](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_eventi), comunemente utilizzato nella [programmazione Web](https://it.wikipedia.org/wiki/Programmazione_Web) [lato client](https://it.wikipedia.org/wiki/Lato_client) per la creazione, in [siti web](https://it.wikipedia.org/wiki/Sito_web) e [applicazioni web](https://it.wikipedia.org/wiki/Applicazione_web), di effetti dinamici [interattivi](https://it.wikipedia.org/wiki/Interattivit%C3%A0)tramite [funzioni](https://it.wikipedia.org/wiki/Funzione_(informatica)) di [script](https://it.wikipedia.org/wiki/Script) invocate da *eventi* innescati a loro volta in vari modi dall'utente sulla [pagina web](https://it.wikipedia.org/wiki/Pagina_web) in uso

**Bootstrap:** Framework che contiene librerie utili per lo sviluppo responsive di pagine web.

**Excel:** Formato di documento utilizzato per tener traccia di informazioni.

**Parser:** Processo che analizza un flusso di dati e ne determina la struttura.

**JSON:** E’ un formato adatto per lo scambio dei dati in applicazioni client-server.

**JS-XLXS:** Nome del parser che utilizzeremo per analizzare e convertire i documenti excel.

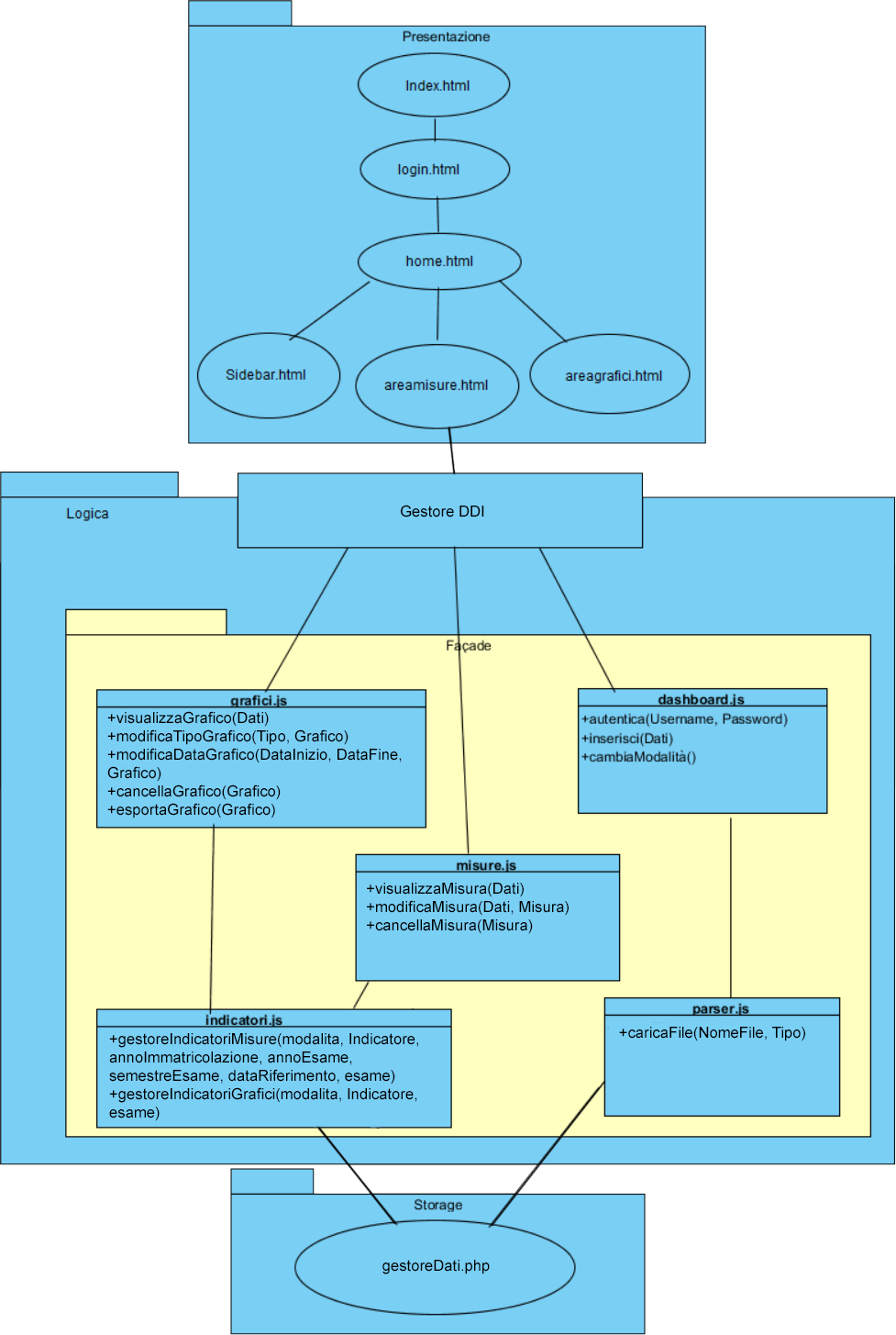
**Camel Notation:** Consiste nello scrivere più parole insieme delimitando la fine e l’inizio di una nuova parola con una lettera maiuscola.

**Hard Coding:** Codifica fissa.

## Riferimenti

* Bernd Bruegge & Allen H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns and Java*, (2nd edition),Prentice-Hall, 2003
* Ian Sommerville, Software Engineering, Addison Wesely
* SDD\_DDI\_1.0
* <http://getbootstrap.com/>
* <http://www.highcharts.com/>
* <https://github.com/SheetJS/js-xlsx>

# Packages



# Interfacce delle classi

|  |  |
| --- | --- |
| Nome classe | GestoreDashboard |
| Descrizione | Rappresenta il gestore delle funzionalità relate alla Dashboard quale login, modifica modalità e caricamento dati. |
| Pre-condizione | **context** GestoreDashboard:: autentica(Username, Password); **pre:** username!=null && password!=null  **context** GestoreDashboard:: inserisci(Dati); **pre:** Dati!=null  **context** GestoreDashboard:: cambiaModalità(); |
| Post-condizione |  |
| Invarianti |  |

Per Dati intendiamo il nome del file in cui i dati sono localizzati.

|  |  |
| --- | --- |
| Nome classe | GestoreMisure |
| Descrizione | Questa classe gestisce la logica applicativa attinente la gestione delle misure all’interno della Dashboard permettendo la loro visualizzazione, modifica e cancellazione. |
| Pre-condizioni | **context** GestoreMisure:: Misura visualizzaMisura(Dati); **pre** Dati!=null Dati.idIndicatore>=0 && Dati.idIndicatore<numeroIndicatori  **context** GestoreMisure:: Misura modificaMisura(Dati, Misura);  **pre**: Dati!=null && Misura!=null  **context** GestoreMisure:: cancellaMisura(Misura) |
| Post-condizioni |  |
| Invarianti |  |

Per numeroIndicatori intendiamo il massimo numero di indicatori di cui il sistema dispone.

|  |  |
| --- | --- |
| Nome Classe | GestoreGrafici |
| Descrizione | Questa classe gestisce la logica applicativa inerente ai grafici, permette la modifica la cancellazione, e la visualizzazione. |
| Pre-condizioni | **context** GestoreGrafici:: Grafico visualizzaGrafico(Dati); **pre**: Dati!=null && Dati.IdIndicatore >=1 && Dati.IdIndicatore<=numeroIndicatori  **context** GestoreGrafici:: Grafico modificaTipoGrafico(Tipo,Grafico);  **pre:** Tipo!=null && Grafico!=null  **context** GestoreGrafici:: Grafico modificaDataGrafico(DataInizio, DataFine, Grafico);  **pre**: DataInizio!=null && DataFine!=null && DataInizio<=DataFine && Grafico!=null  **context** GestoreGrafici:: cancellaGrafico(Grafico) **pre:** Grafico!=null  **context** GestoreGrafici:: esportaGrafico(Grafico) **pre:** Grafico!=null |
| Post-condizione |  |
| Invarianti |  |

Dati è una struttura dati che contiene tutte le informazioni necessarie per creare opportunamente un grafico.

Dati.idIndicatore è una forma contratta che usiamo per far riferimento all’identificativo dell’indicatore memorizzato nella struttura dati Dati.

Per numeroIndicatori intendiamo il massimo numero di indicatori di cui il sistema dispone.

|  |  |
| --- | --- |
| Nome Classe | GestoreIndicatori |
| Descrizione | Questa classe rappresenta la sezione degli indicatori, si occupa di fornire dati relativi ad essi. |
| Pre-condizione | **context** GestoreIndicatori:: Valore gestoreIndicatoriMisure(modalita, Indicatore, annoImmatricolazione, annoEsame, semestreEsame, dataRiferimento, esame);  **pre**: modalita!=null && Indicatore!=null && Indicatore>=0 && Indicatore<numeroIndicatori && (modalita==’triennale’ || modalita==’magistrale’)  **context** GestoreIndicatori:: Valore gestoreIndicatoriGrafici(modalita, Indicatore, esame);  **pre**: modalita!=null && Indicatore!=null && Indicatore>=0 && Indicatore<numeroIndicatori && (modalita==’triennale’ || modalita==’magistrale’) |
| Post-condizione |  |
| Invariante |  |

|  |  |
| --- | --- |
| Nome classe | Parser |
| Descrizione | Questa classe rappresenta la gestione del parsing del file Excel. |
| Pre-condizioni | **context** Parser:: caricaFile(NomeFile, Tipo);  **pre:** nomeFile!=null && Tipo!=null |
| Post-condizioni |  |
| Invarianti |  |

Nel caricamento file il tipo da specificare è relativo alla tipologia di dati che si sta inserendo, cioè se sono dati relativi agli esami superati degli studenti, se sono dati relativi alla coda degli esami…

|  |  |
| --- | --- |
| Nome classe | GestoreDati |
| Descrizione | Questa classe gestisce l’interfaccia tra database e lo strato di logica; essa si occupa in particolare di trasformare delle richieste dell’utente in query da inviare al DBMS.  L’oggetto che funge da intermediario per lo scambio di dati di cui necessita lo strato di business per calcolare i valori richiesti dall’utente è chiamato Dati. |
| Pre-condizioni | **context** GestoreDati:: Dati getDatiMisure(indicatore, modalita, annoImmatricolazione, dataRiferimento, esami);  **pre** indicatore!=null && indicatore>=1 && indicatore<= numeroIndicatori && modalita!=null && (modalita==’triennale’ || modalita==’magistrale’)  **context** GestoreDati:: Dati getDatiGrafici(indicatore, modalita, esami); **pre** indicatore!=null && indicatore>=1 && indicatore<= numeroIndicatori && modalita!=null && (modalita==’triennale’ || modalita==’magistrale’) |
| Post-condizioni | **context** GestoreDati:: Dati getDatiMisure(indicatore, modalita, annoImmatricolazione, dataRiferimento, esami);  **post** Dati!=null || Dati==null  **context** GestoreDati:: Dati getDatiGrafici(indicatore, modalita, esami);  **post** Dati!=null || Dati==null |
| Invarianti |  |

Per numeroIndicatori intendiamo il massimo numero di indicatori di cui il sistema dispone.