Corso di Programmazione ad Oggetti - a.a. 2013/2014
Corso di laurea in Ingegneria e Scienze Informatiche - Università di Bologna

Relazione relativa all'ingegnerizzazione di menùBuilder: applicazione per la creazione di menù stampabili

Componenti del gruppo: Giacomo Rossi , Raffaele Gori.

1 Analisi del problema

Ai programmatori è stata commissionata la realizzazione di un applicativo che consenta ai gestori di qualunque tipo di esercizio ristorativo di creare e modificare in maniera semplice e intuitiva i propri menù.
L'obiettivo è quello di permettere ad un qualsiasi utilizzatore dell'applicazione di generare, e in seguito di poter stampare, un menù tramite l'inserimento delle informazioni necessarie.
Nello specifico un generico utente deve avere la possibilità di:

- Scegliere la lingua preferita tra Italiano e Inglese.
- Impostare la cartella in cui salvare il menù.
- Inserire le proprie informazioni e quelle relative all'attività.
- Gestire gli elenchi di vivande proposte ai clienti.
- Scegliere la formattazione grafica del menù.

Una analisi del problema mostra che una applicazione di tipo GUI-based è la soluzione più semplice ed efficace per questo tipo di problema, poiché rende molto più semplice e intuitiva l'interazione uomo-macchina, e, con i suoi campi a completamento o a scelta guidata, fornisce una struttura solida per un immagazzinamento consistente dei dati.

2 Progettazione architetturale

L'intera fase di progettazione è basata sull'utilizzo del pattern Model-View-Controller (MVC), che viene applicato nell'immagazzinamento dei dati relativi all'esercizio e alla lista dei cibi, e nella raccolta delle informazioni per il settaggio della veste grafica del menù in formato PDF.
L'idea di fondo è quella di disaccoppiare:

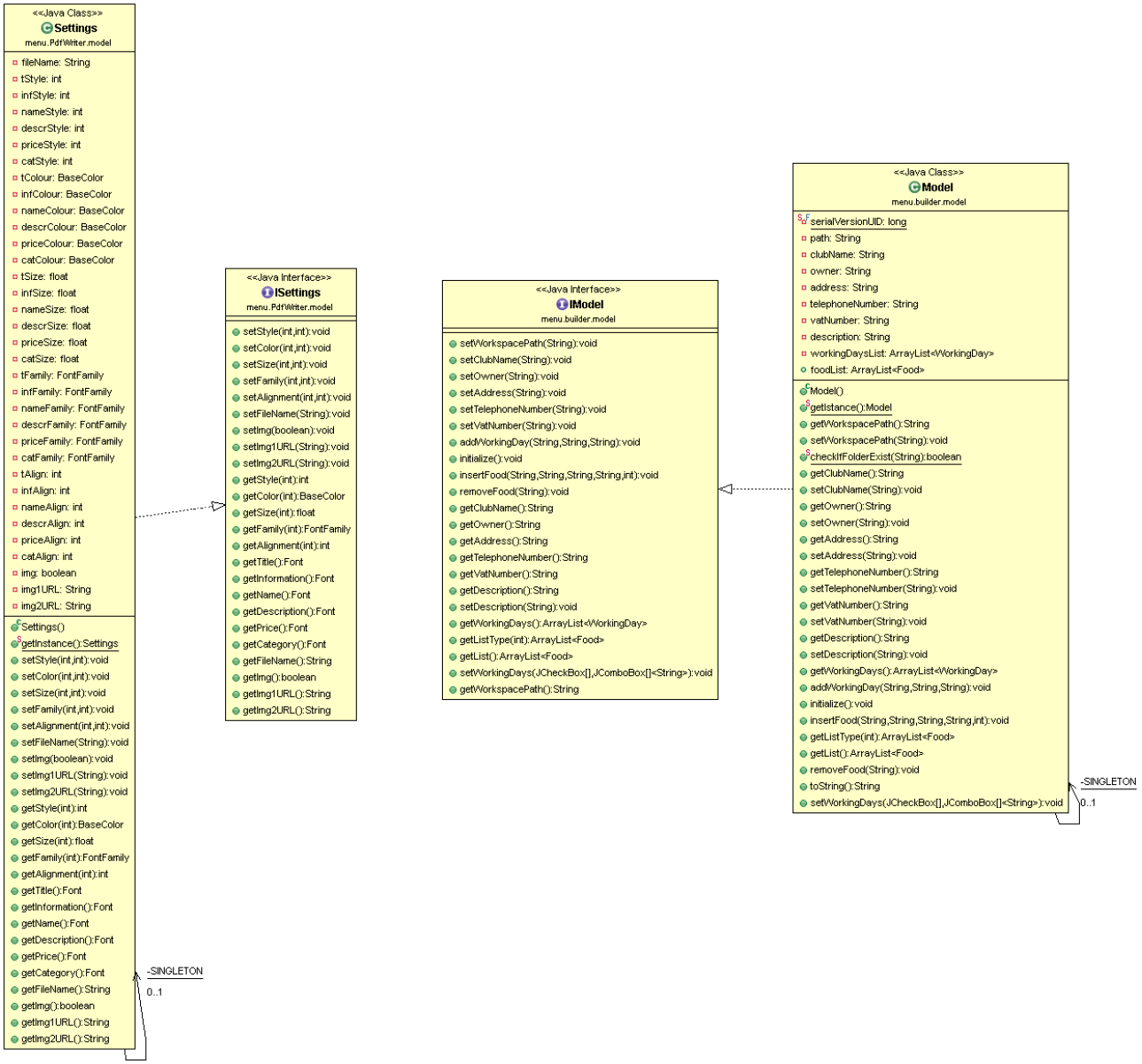
- rappresentazione del modello di dominio (**model**)
- interfaccia utente (**view**)
- controllo dell'interazione uomo-macchina (**controller**).

Il modello deve risultare, quindi, indipendente dall'ambiente grafico, che non deve poterlo modificare direttamente. Le viste, formate da componenti grafici riutilizzabili (bottoni, scrollbar, liste, ecc.), forniscono l'interfaccia del programma con l'utente umano, ed i controller devono mediare i rapporti tra questo e la concreta modificazione del modello, incorporando come campi sia view che model, e fornendo i metodi adeguati per collegarli.
Il disaccoppiamento derivante da tale separazione consente il riuso del modello, delle view o del controller nel contesto di scenari e applicazioni differenti.
L'obiettivo è raccogliere attraverso l'MVC tutte le informazioni necessarie alla scrittura del menù per poi andarle a scrivere sul pdf, che costituirà il menù da stampare.

Per la scrittura del PDF si fa riferimento alla libreria iText (di proprietà di iText®), scaricabile gratuitamente dal sito *sourceforge.net*, che rende possibile creare e modificare file in formato PDF, e quindi permetterà di generare il menù finale.
Nello sviluppo dell'applicazione la sezione che gestisce l'immissione e l'organizzazione in una apposita struttura dati delle informazioni riguardanti le vivande e i dati del locale (sezione della quale si è occupato Giacomo Rossi), è stata tenuta separata da quella relativa alla immissione delle informazioni necessarie per la personalizzazione dell'aspetto grafico del testo in formato PDF, anch'esse memorizzate in una struttura dati dedicata (sezione di cui si è occupato Raffaele Gori).
Il risultato è una organizzazione in due package principali: **builder** e **PdfWriter**, entrambi internamente organizzati secondo una struttura MVC.

Segue la presentazione di alcuni degli aspetti più rilevanti relativi all'architettura MVC dell'applicativo.

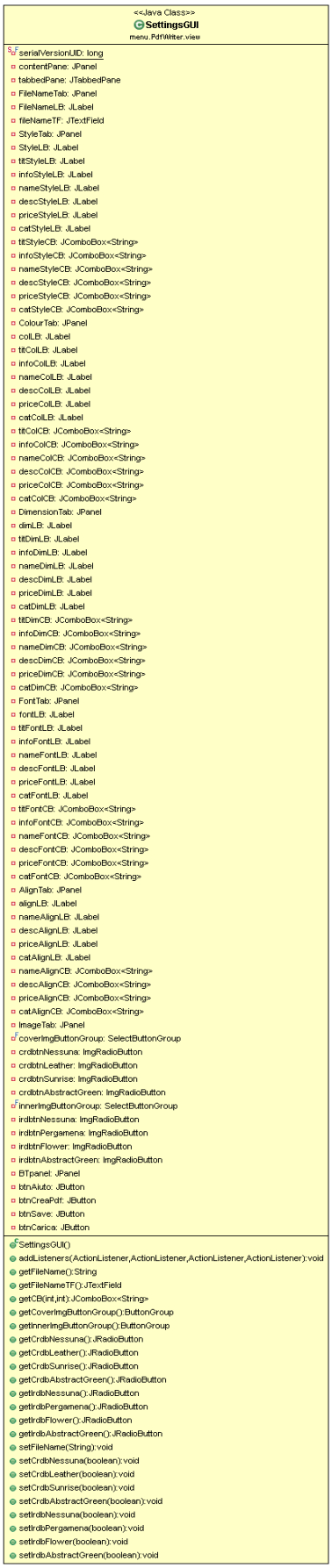
Diagramma UML delle classi relative alla parte del modello dell'applicazione.

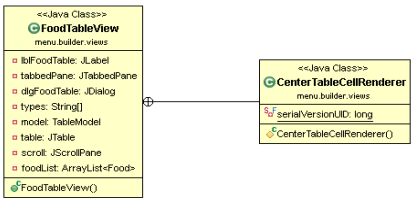


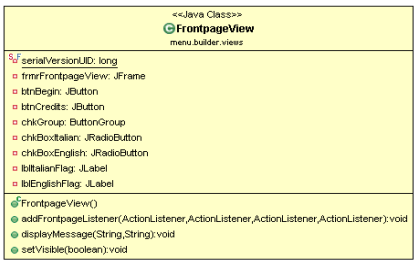
Descrizione degli aspetti principali:

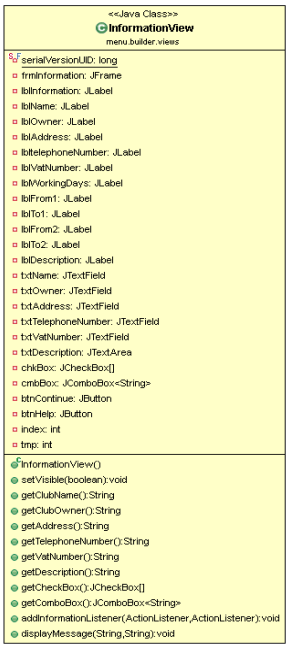
- **IModel** e **Model**: interfaccia e implementazione concreta dell'insieme dei dati (model) usati dal sistema per costruire il testo del menù.
  Per garantire la persistenza dei dati la classe implementa l'interfaccia **Serializable**.
- **ISettings** e **Settings:** Interfaccia e implementazione concreta dell'insieme dei dati (model) usati dalla applicazione per la formattazione del testo e la gestione di tutto quanto riguarda la veste grafica del PDF finale.

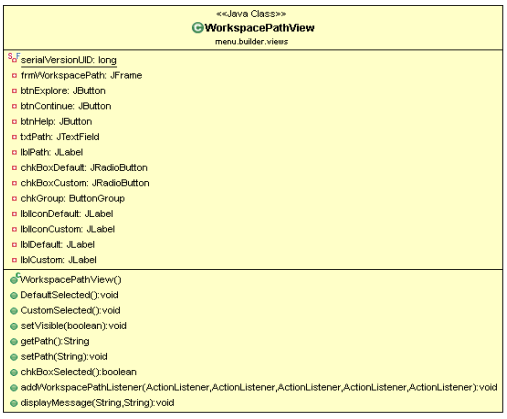
Diagramma UML delle classi relative alla parte della vista dell'applicazione.

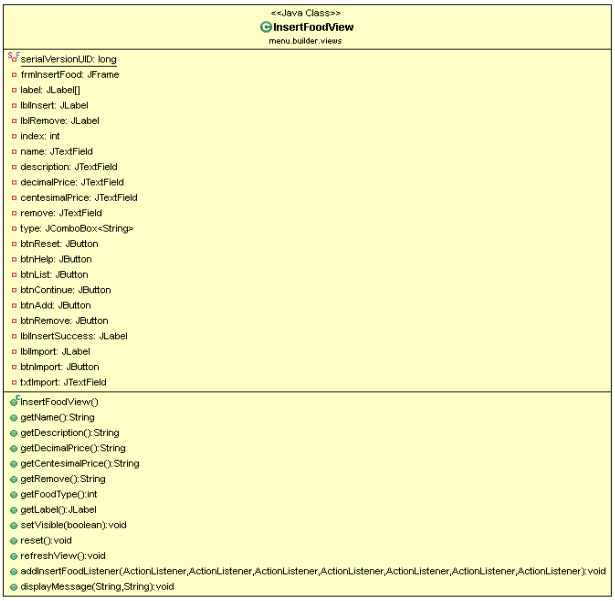










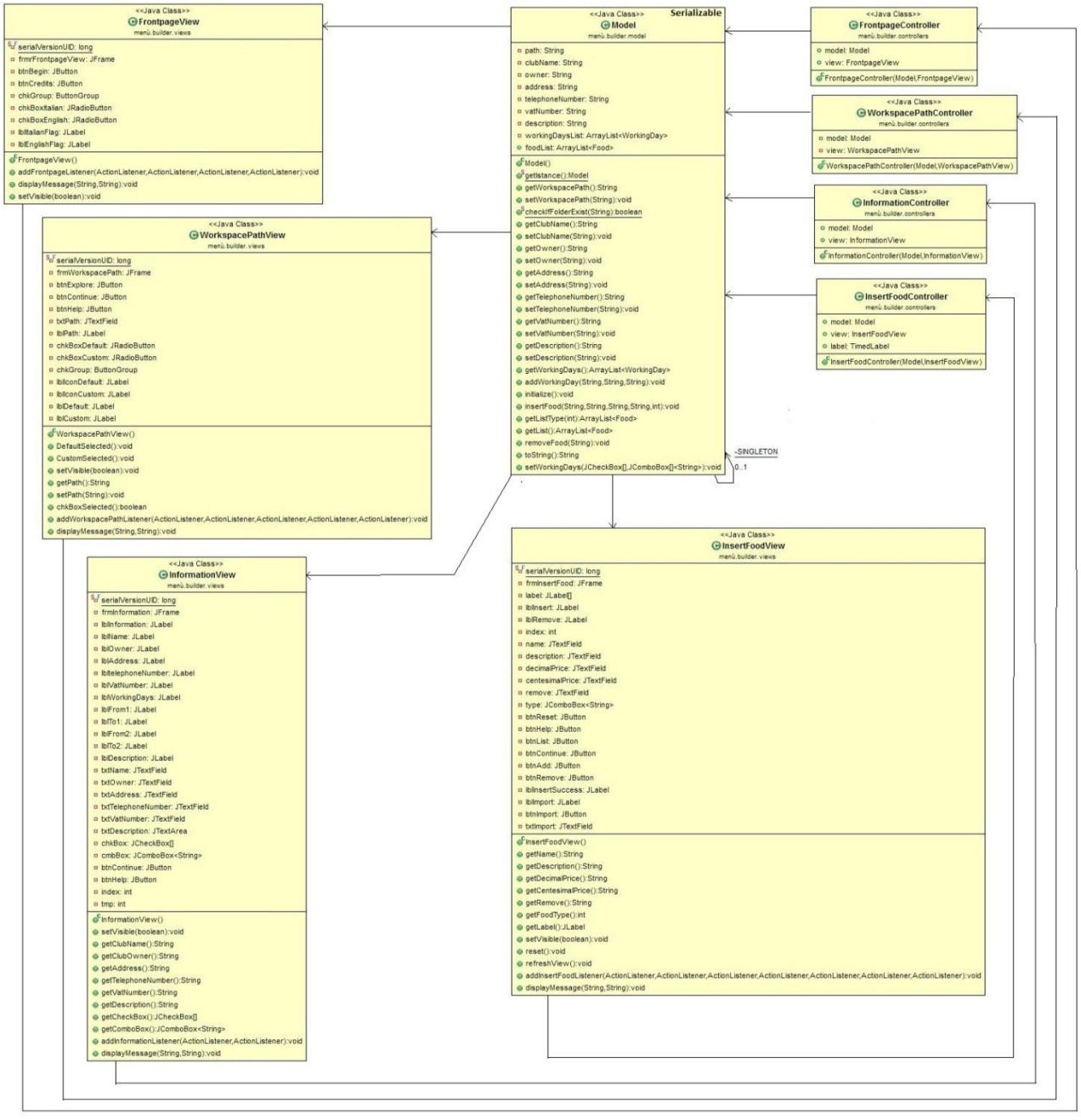


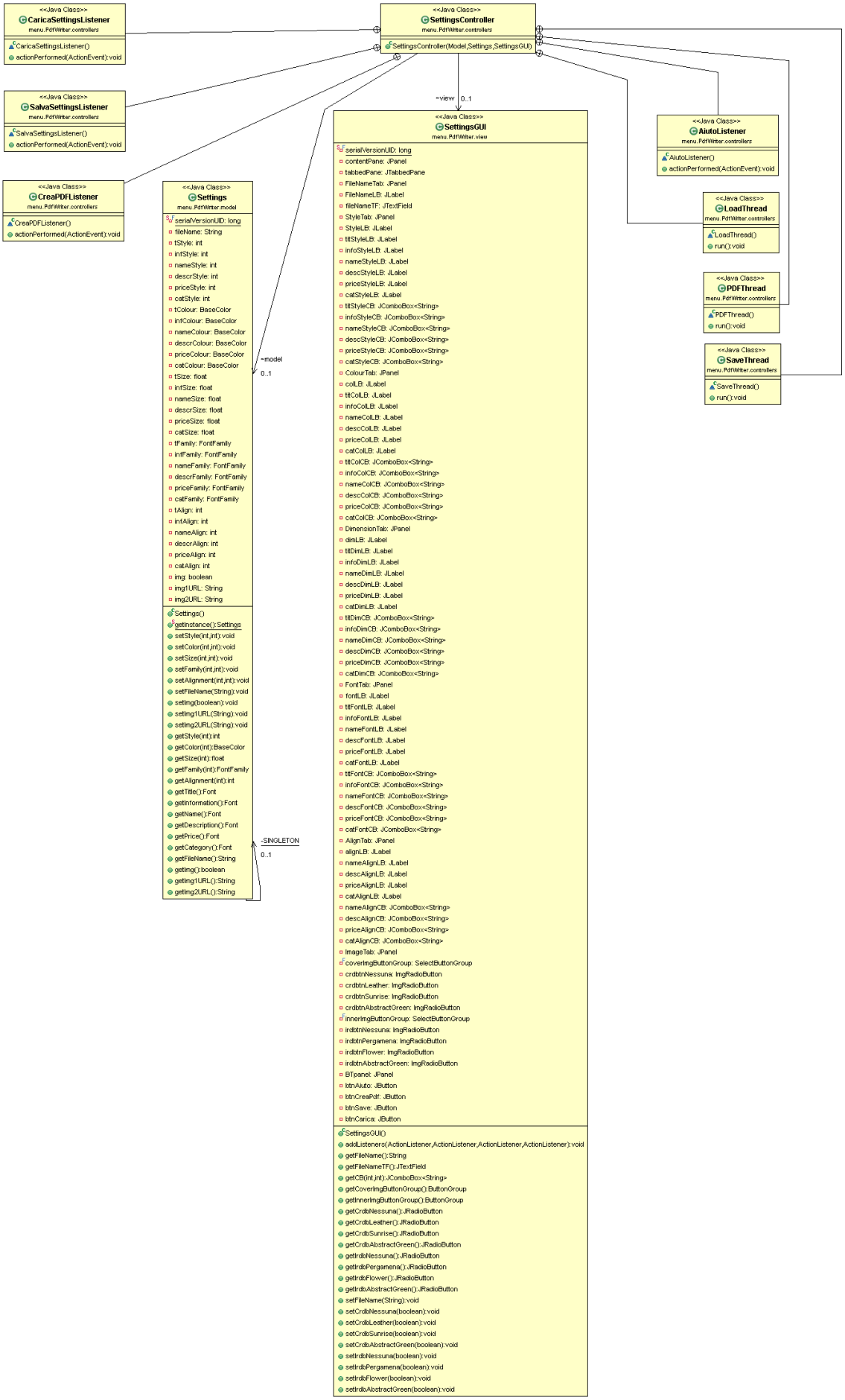
Per seguire un processo ordinato nella raccolta di dati si comincia dalle informazioni relative alla esecuzione dell'applicativo, quali la scelta della lingua e della cartella di salvataggio del progetto, per continuare con quelle relative all'esercizio e al suo proprietario, quelle relative alle liste dei piatti offerti, e, infine, quelle deputate al settaggio della veste grafica che dovrà avere il menù in formato PDF.

La presentazione delle viste segue questa logica, e si organizza come segue:

1. **FrontpageView:** Interfaccia grafica che presenta l'applicazione e permette di impostare la lingua prescelta per le successive finestre e per la stampa del menù.
2. **WorkspacePathView:** Interfaccia grafica che permette la scelta della destinazione in cui il menù verrà salvato all'ultimo passaggio.
3. **InformationView:** Interfaccia grafica per la raccolta delle informazioni relative all'esercizio e al gestore/proprietario.
4. **InsertFoodView:** Finestra per l'inserimento delle informazioni relative alle vivande nella struttura dati deputata.
5. **FoodTableView:** Finestra che compare a seguito della pressione del tasto LISTA e permette di vedere lo stato attuale delle liste di cibi contenute nella struttura dati.
6. **SettingGUI:** Interfaccia grafica preposta alla raccolta delle informazioni per il settaggio della veste grafica del PDF finale e al comando di scrittura di quest'ultimo.

Diagrammi UML delle interazioni tra modelli, viste e controllori della applicazione.
In particolare per il settaggio della classe Model attraverso le prime quattro finestre, e della classe Settings attraverso l'ultima finestra.




«Java Class»
CaricaSettingsListener
menu.PdfWriter.controllers
CaricaSettingsListener()
actionPerformed(ActionEvent):void
«Java Class»
SettingsController
menu.PdfWriter.controllers
SettingsController(Model,Settings,SettingsGUI)
«Java Class»
SalvaSettingsListener
menu.PdfWriter.controllers
SalvaSettingsListener()
actionPerformed(ActionEvent):void
~view
0..1
«Java Class»
SettingsGUI
menu.PdfWriter.view
«Java Class»
AiutoListener
menu.PdfWriter.controllers
AiutoListener()
actionPerformed(ActionEvent):void
«Java Class»
LoadThread
menu.PdfWriter.controllers
LoadThread()
run():void
«Java Class»
CreaPDFListener
menu.PdfWriter.controllers
CreaPDFListener()
actionPerformed(ActionEvent):void
«Java Class»
Settings
menu.PdfWriter.model
~model
0..1
«Java Class»
PDFThread
menu.PdfWriter.controllers
PDFThread()
run():void
«Java Class»
SaveThread
menu.PdfWriter.controllers
SaveThread()
run():void
-SINGLETON
0..1

Descrizione degli aspetti principali:

Come si può vedere dai diagrammi, i controller sono gli unici ad avere la possibilità di modificare i model, che incorporano come campi, e mediano le interazioni tra essi e le view, le quali attingono, però, dai model per rappresentare le informazioni quando necessario.
La presenza di due diagrammi rispecchia la separazione logica a cui si è accennato in precedenza tra la parte di esecuzione deputata alla raccolta delle informazioni che andranno a costituire il contenuto del documento, e quella per il settaggio della sua impaginazione.
Trait d'union tra le due sezioni è il fatto che alla pressione del tasto "Crea PDF" in SettingsGUI l'actionListener deputato accede all'unica istanza di Model, necessaria insieme a Settings, per scrivere il PDF.
Per garantire la consistenza delle informazioni in questa fase, Model segue il pattern Singleton, in modo che l'istanza passata allo scrittore in fase di costruzione non possa essere diversa da quella settata nella fase precedente.

3 Organizzazione in Package

Al livello di astrazione maggiore la applicazione è organizzata in un package **menu**, che contiene tutto il codice operativo, al quale si affiancano due cartelle sorgenti (**res** e **language**) contenenti dati cui si farà riferimento nel programma, quali i bundle per il linguaggio (l'applicazione offre la possibilità di multilanguage), le immagini che compariranno nel background di alcune finestre, o quelle che possono essere impostate come sfondi di copertina e pagine interne per il PDF.

All'interno di **menu** i due package principali sono, come detto in precedenza, **builder** e **PdfWriter**.
Il primo contiene tutte le classi connesse alla organizzazione e popolazione di una struttura dati dedicata alla memorizzazione di ciò che verrà scritto nel menù, mentre nel secondo sono organizzate tutte le classi connesse alla sua scrittura materiale e alla personalizzazione della sua veste grafica.

Viene ora effettuata una analisi dell'organizzazione in package dell'applicazione.

- **menu.builder.constants:**
  Package contenente le costanti utilizzate in questa parte dell'applicazione.
- **menu.builder.controllers:**
  Package contenente tutti i controller che interagiranno con le varie view e il model dell'applicazione nella prima fase di esecuzione.
  - *FrontpageController*: controller relativo alla interazione tra Model e la prima finestra che viene mostrata all'apertura, che ha la funzione di presentare l'applicazione e consentire la scelta della lingua.
  - *WorkspacePathController*: controller relativo alla interazione tra Model e la finestra di impostazione del cammino per il salvataggio del menù, che viene mostrata per seconda.
  - *InformationController*: controller relativo alla interazione tra Model e la finestra di raccolta delle informazioni di proprietario ed esercizio, che viene mostrata per terza.
  - *InsertFoodController*: controller relativo alla interazione tra Model e la finestra di gestione delle vivande dell'applicazione, che viene mostrata per quarta ed è l'ultima della sezione curata da Giacomo Rossi.

- **menu.builder.exceptions:**
  Package contenente le eccezioni che possono essere lanciate dall'applicazione.
  - *AnyWorkingDaySelected*: eccezione lanciata quando non è stato selezionato alcun giorno lavorativo nella schermata di raccolta delle informazioni del locale.
  - *FoodAlreadyPresent*: eccezione lanciata quando la vivanda che si cerca di inserire è già presente nella struttura dati. In questo modo risulterà impossibile inserire più volte una pietanza nello stesso menù.
  - *FoodNotFound*: eccezione lanciata quando si cerca di rimuovere una vivanda che non è contenuta nella struttura dati.
  - *InvalidDirectory*: eccezione lanciata quando viene inserito un percorso invalido per il salvataggio del menù scegliendo l'opzione di cammino personalizzato nella seconda schermata presentata dal programma.
  - *InvalidPrice*: eccezione lanciata quando il prezzo inserito della vivanda non corrisponde al formato richiesto.
  - *MissingField*: eccezione lanciata quando si cerca di proseguire con la schermata successiva, ma un campo richiesto dall'applicazione non è ancora stato compilato.
- **menu.builder.language:**
  Package contenente la classe Text avente il compito di gestire la lingua dell'applicazione.
- **menu.builder.main:**
  Package contenente il main dell'applicazione.
- **menu.builder.model:**
  Package contenente il model dell'applicazione e le sue classi di supporto.
  - *Food*: classe che modella una ipotetica vivanda, di cui vengono memorizzati nome, prezzo e descrizione.
  - *FoodTableModel*: classe che permette la costruzione della tabella contenente tutte le vivande inserite.
  - *Model/IModel*: classe che costituisce propriamente il modello per questa sezione di applicativo e relativa interfaccia. Questa classe contiene tutte le informazioni inerenti il contenuto del menù.
  - *WorkingDay*: classe che modella un ipotetico giorno lavorativo, memorizzandone nome, orario di apertura e orario di chiusura.
- **menu.builder.utilities:**
  Package contenente una classe utilizzata durante lo svolgimento dell'applicazione che, attraverso la Reflection, mostra a video tutte le informazioni immesse per verificarne il corretto inserimento.
- **menu.builder.view:**
  Package contenente le view che interagiranno con i relativi controller e con la classe Model, che è il model di questa sezione, e qualche classe di supporto.
  - *FoodTableView*: view che mostra una tabella contenente le informazioni di tutte le vivande inserite, organizzate per categoria.
  - *FrontpageView*: view preposta alla presentazione dell'applicazione e alla scelta della lingua.
  - *WorkspacePathView*: view relativa alla scelta del percorso di salvataggio del menù.
  - *InformationView*: view relativa alla raccolta delle informazioni di esercizio e proprietario.
  - *InsertFoodView*: view relativa alla gestione della struttura dati contenente la lista delle vivande.
  - *JtextFieldLimit*: campo di testo utilizzato in alcune view con una limitazione per il numero di caratteri che si possono immettere.
  - *TimedLabel*: label temporizzato utilizzato per notificare all'utente l'avvenuto inserimento con successo di una vivanda nella struttura dati.

- **menu.PdfWriter.constants**
  Package che si compone di tre classi di sorgente contenenti valori costanti che costituiscono i domini dei valori assumibili dai campi delle varie classi durante l'esecuzione.
    - *SettingsConstants*: contiene i valori che potranno essere assunti dai campi della classe *Settings*.
    - *ExceptionsConstants*: contiene i messaggi che saranno stampati a video nella console a fronte di determinate eccezioni durante l'esecuzione.
    - *ImagesPaths*: contiene i percorsi relativi per l'accesso alle immagini di sfondo per i PDF, contenute nella cartella sorgente **res.**
- **menu.PdfWriter.controllers**
  Contiene la classe dell'unico controller previsto per questa sezione di programma, che media l'interazione tra la vista SettingsGUI e il modello Settings, e comanda la creazione del PDF a fronte della pressione dell'apposito tasto nella interfaccia grafica.
- **menu.PdfWriter.exceptions**
  Contiene le classi di eccezioni che si è ritenuto di dover aggiungere a quelle già presenti nel JDK.
    - *CategoryException*: eccezione lanciata dal metodo writeFoodPage() di PDF_MWriter qualora l'indice passato per la categoria non risulti ammissibile.
    - *InvalidComboboxSelection*: eccezione lanciata dai setters di Settings in caso gli indici passati loro nel parametro *index** non risultino ammissibili per il particolare tipo di settaggio.
    - *InvalidParameterToSet*: eccezione lanciata dai setters di Settings in caso gli indici passati loro nel parametro *par** non risultino ammissibili per il particolare tipo di settaggio.
    - *InvalidParameterToGet*: eccezione lanciata dai setters e getters di Settings e SettingsGUI in caso gli indici passati loro nei parametri *par**, *par1**, *par2** non risultino ammissibili.

  **Il parametro* par *costituisce un indice utilizzato per selezionare quale campo settare in particolare, mentre* index *è l'indice per selezionare quale valore il campo indicato da par debba assumere.*
  I parametri par1 *e* par2 *servono, accoppiati, per selezionare di quale JComboBox della GUI si desidera leggere il valore.*
  Per maggiori informazioni su par*,* par1*,* par2 *e* index *si faccia riferimento alla Javadoc dei metodi interessati.*
- **menu.PdfWriter.model**
  Contiene la classe di modello per quel che riguarda la gestione dei settaggi della veste grafica del PDF.
    - *Settings*: Questa classe implementa il pattern Singleton, e si compone di una serie di campi di vario tipo necessari alla formattazione del testo del PDF finale o alla selezione dell'immagine di background per copertina e pagine interne.
      I campi hanno i rispettivi domini nella classe SettingsConstants, e sono settati automaticamente a valori di default all'atto della creazione dell'unico oggetto istanziabile, così da ridurre il rischio di eccezioni dovute a sviste nei settaggi tramite GUI.
- **menu.PdfWriter.utilities:** Contiene le classi *PDF_MWriter* e *ProjectSets*, che non appartengono logicamente a nessuno degli altri package, poiché non sono parte né delle strutture dati del modello, né della vista, e nemmeno del controllore, se non per il fatto che loro istanze vengono create ed utilizzate da alcuni dei listeners dei bottoni per generare il PDF finale e per salvare/caricare vecchie impostazioni dei campi della GUI.

- *PDF_MWriter*: Questa classe prende in ingresso le due istanze dei modelli (**Model** e **Settings**) ed il nome che si è scelto per il progetto (che nel file finale sarà preceduto dalla stringa "Menù_") e permette di creare la versione stampabile del menù.
  I metodi pubblici sono createMenùProject() e writeMenùPDF():
  il primo scrive tutto quello che sarà contenuto nel menù seguendo già la formattazione richiesta dalle specifiche impostate dall'utente, ma su un file temporaneo in formato .dat e senza l'immagine di sfondo, mentre il secondo legge il contenuto del file temporaneo di cui sopra e vi inserisce, se richiesto, una immagine di sfondo.
- *ProjectSets*: Questa classe costituisce una piccola struttura dati deputata alla memorizzazione di tutti i valori assunti dai vari componenti della vista nel momento del salvataggio.
  In particolare memorizzerà il nome scelto per il menù (scritto nel JTextField del primo pannello di SttingsGUI), il nome che si è impostato per il file in cui verranno salvati i dati, una serie di interi rappresentanti gli indici di selezione delle JComboBox contenute nei vari pannelli dell'interfaccia grafica, e i valori booleani rappresentanti lo stato dei JRadioButtons connessi alla scelta delle immagini di sfondo per copertina e pagine interne.
  Gli unici metodi forniti dalla classe sono i getter per i suoi campi, che serviranno, in sede di caricamento di una vecchia configurazione, ad ottenere i valori per settare opportunamente i campi della vista e riprodurre lo stato precedente.

- **menu.PdfWriter.view:**
  Contiene la classe della view (*SettingsGUI*) e due classi di supporto.
  Di queste due classi solo una è stata scritta appositamente per questa GUI (*ImgRadioButton*), mentre l'altra (*SelectButtonGroup*) è stata trovata sulla rete e porta la firma di tale Darryl Burke, che ne ha reso libero il download a questo indirizzo:
  http://tips4java.wordpress.com/2008/11/09/select-button-group/ .
  - *SettingsGUI*: Questa classe estende JFrame e si compone di due pannelli principali: nella parte SOUTH presenta un pannello fisso per accogliere i JButtons per aiuto, caricamento e salvataggio delle impostazioni, e creazione del PDF, mentre nella sezione centrale ospita un JTabbedPane organizzato in sette pannelli, ciascuno dei quali deputato ad un particolare tipo di settaggio.
    Fatta eccezione per il metodo che collega ogni JButton al rispettivo ActionListener, la classe fornisce solo setter e getter.
  - *ImgRadioButton*: Questa classe estende JRadioButton, dalla quale si differenza esclusivamente per la presenza di un campo di tipo String deputato ad accogliere il path per una immagine, che fornirà al thread che gestisce il settaggio del modello quando gli sarà richiesto.
  - *SelectButtonGroup*: Dell'esistenza di questa classe, che estende ButtonGorup, si è venuti a sapere tramite un forum, che rimandava alla pagina di "Java Tips Weblog" il cui indirizzo è stato fornito sopra.
    Benché fornisca diversi metodi non disponibili nella classe padre, si è scelto di utilizzarla per la presenza del metodo getSelectedButton(), che non fornisce in output un oggetto di tipo ButtonModel, ma un AbstractButton, cosa che facilita la conversione al giusto tipo di bottone e la richiesta dello stato.

4 Suddivisione del lavoro

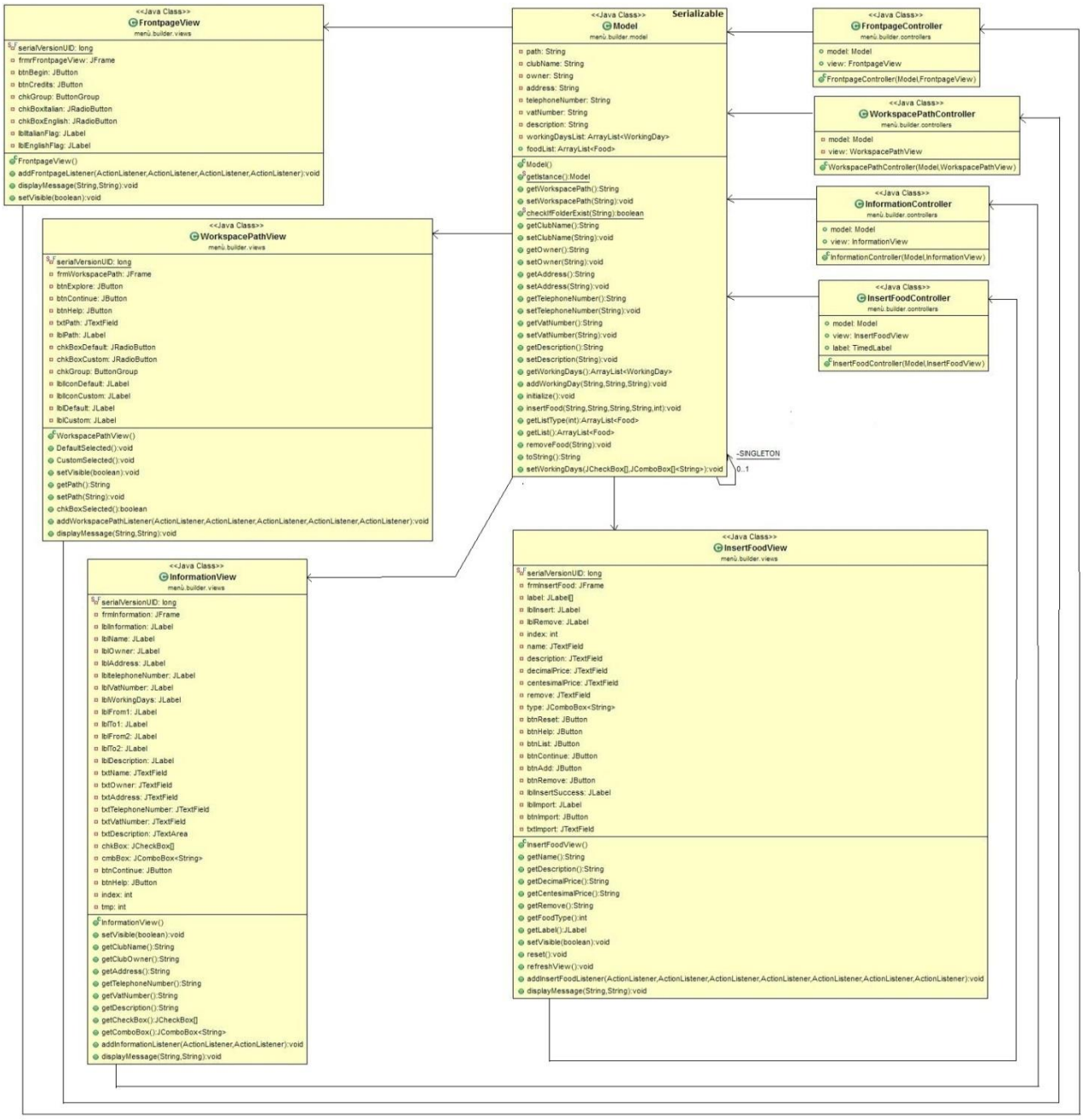
Nel contesto della realizzazione del progetto il lavoro tra i vari componenti del gruppo è stato suddiviso come segue:

- Giacomo Rossi si è occupato della raccolta di tutti i dati e le informazioni necessarie alla creazione del contenuto del menù, e in particolare:
  - Directory in cui salvare il progetto.
  - Informazioni riguardanti il titolare/l'attività in questione.
  - Gestione delle liste di vivande.
  - Rappresentazione dei cibi.
  - Supporto multilingua (Italiano/Inglese).

- Raffaele Gori si è occupato della scrittura del menù in formato PDF facendo uso della libreria iText e della organizzazione di tutti gli aspetti relativi alla formattazione della sua veste grafica.
  In particolare:
  - Settaggi dei font (colore, dimensione, stile, famiglia del font).
  - Scelta dell'immagine di sfondo per la copertina e per le pagine interne.
  - Allineamento dei paragrafi personalizzabile.
  - Scrittura del menù in formato PDF.
  - Struttura/lettura dei dati serializzati per il salvataggio/caricamento dei settaggi grafici prescelti.

5.1 Progettazione di dettaglio: Parte di Giacomo Rossi

MVC:



Il diagramma mostra come ogni volta che una view viene presentata all'utente il relativo controller aggiorna il model dell'applicazione.
L'applicazione viene gestita in maniera sequenziale: una volta completata una parte si procede al relativo salvataggio e viene presentata la seguente. In questo caso:

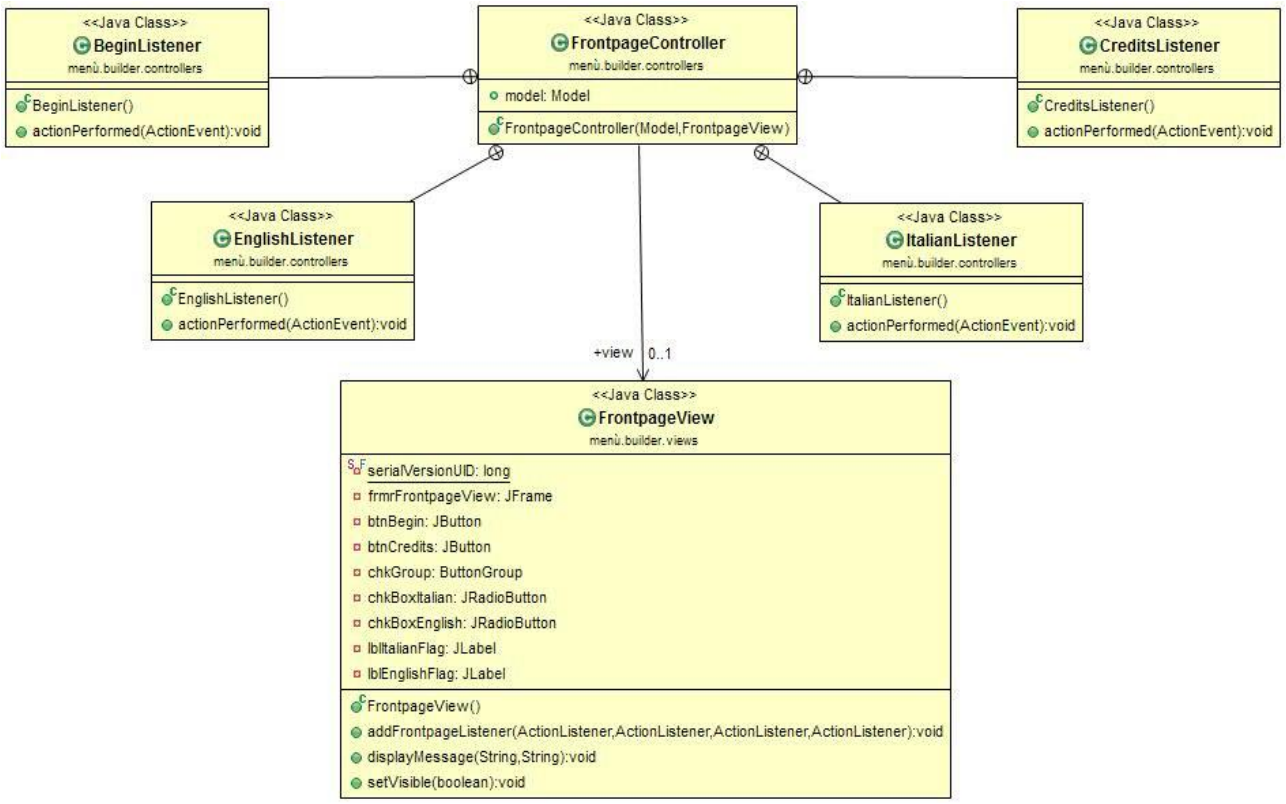
- Viene presentata la FrontpageView, avente il compito di introdurre l'applicazione all'utente e di dargli la possibilità di scegliere la lingua preferita tramite opportuno selettore. Per chi fosse interessato tramite l'ooportuno pulsante sarà possibile conoscere alcune piccole informazioni riguardanti il progetto.
- Segue la WorkspacePathView, avente il compito di far scegliere la directory ove salvare il menù.
  Anche qui, tramite selettore, sarà possibile scegliere tra due differenti possibilità, una veloce (salvataggio su desktop) ed una più personalizzata (salvataggio in una cartella specifica).

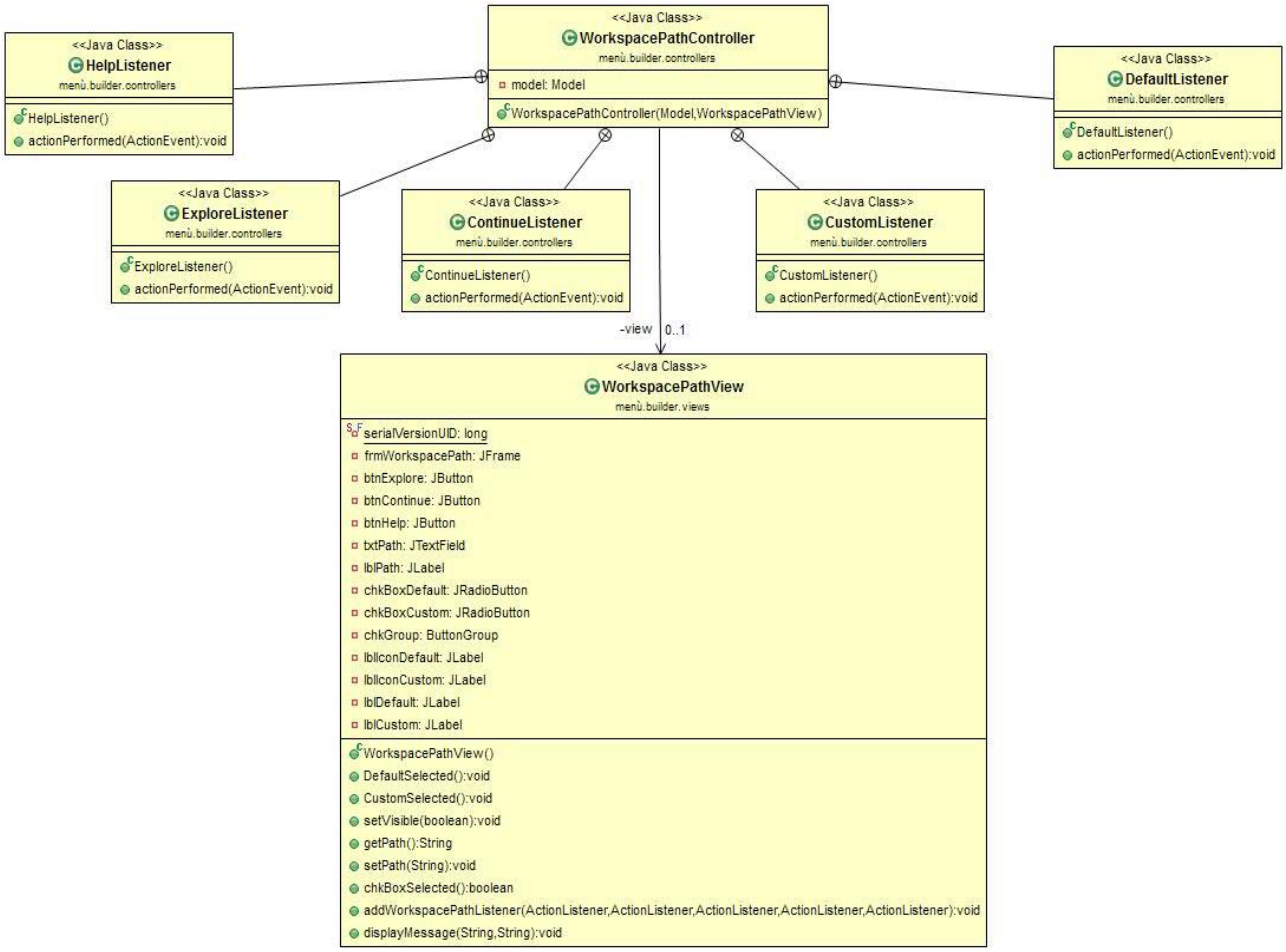
- Viene poi mostrata l'InformationView, avente il compito di raccogliere le informazioni su titolare e attività. Sono presenti diversi campi da compilare (informazioni generali) e, per quanto riguarda i giorni lavorativi, caselle da spuntare per selezionare il giorno ed in seguito completare l'orario con i relativi menù a tendina.
- Per concludere InsertFoodView, avente il compito di consentire all'utente l'inserimento e la gestione delle vivande. L'inserimento avviene mediante compilazione di determinati campi (nome, prezzo) e del menù a tendina (tipo) mentre la rimozione riguarda solo la compilazione di un solo campo (nome).

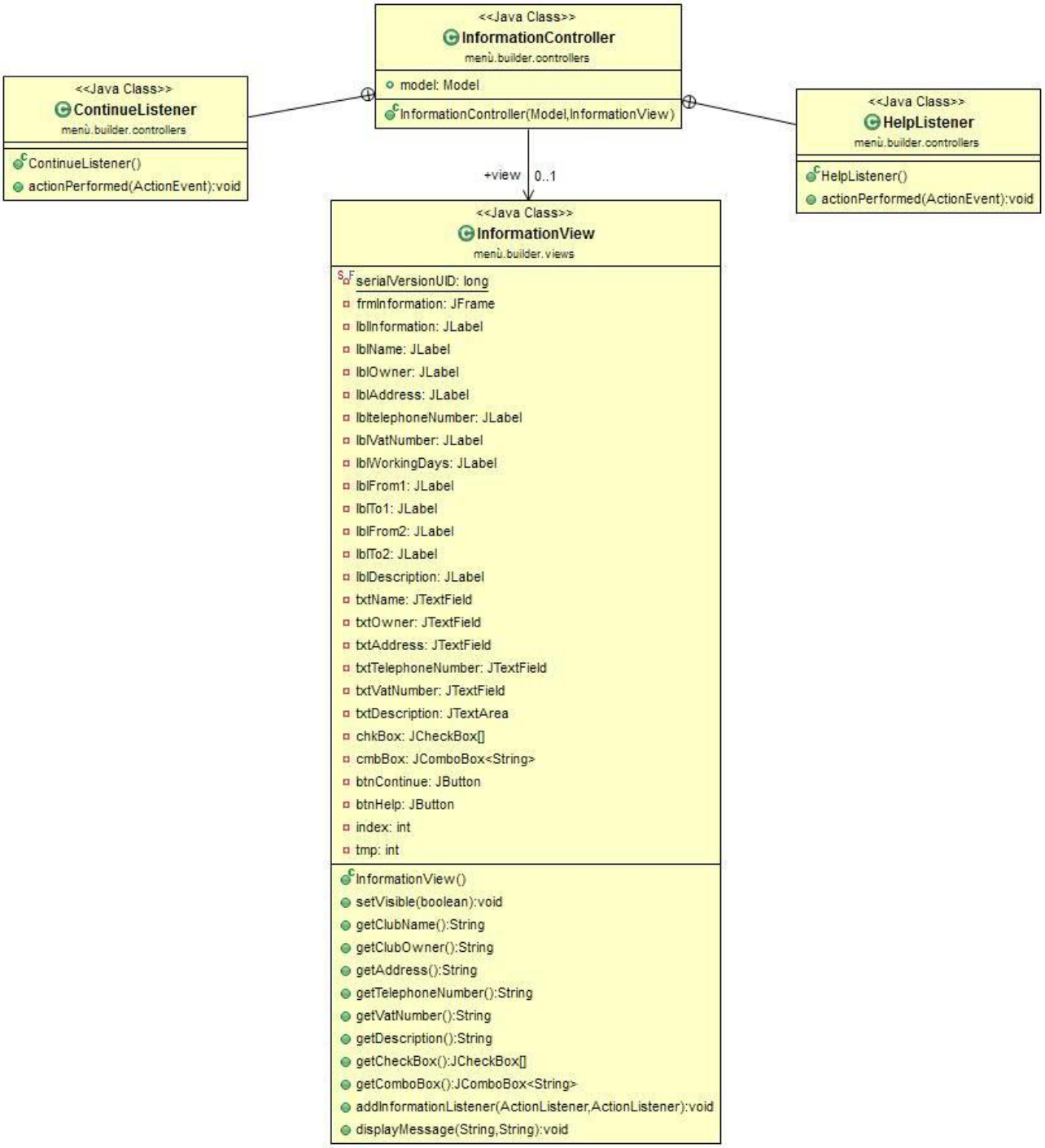
Ogni view interagisce con il proprio controller che, una volta completata una parte, avrà il compito di aggiornare correttamente il modello dell'applicazione.
Si può notare l'uso del pattern MVC e del pattern SINGLETON per il modello. Si è preferito l'uso di questo pattern in quanto l'applicazione utilizza necessariamente un unico modello contenente tutti i dati necessari all'esecuzione.

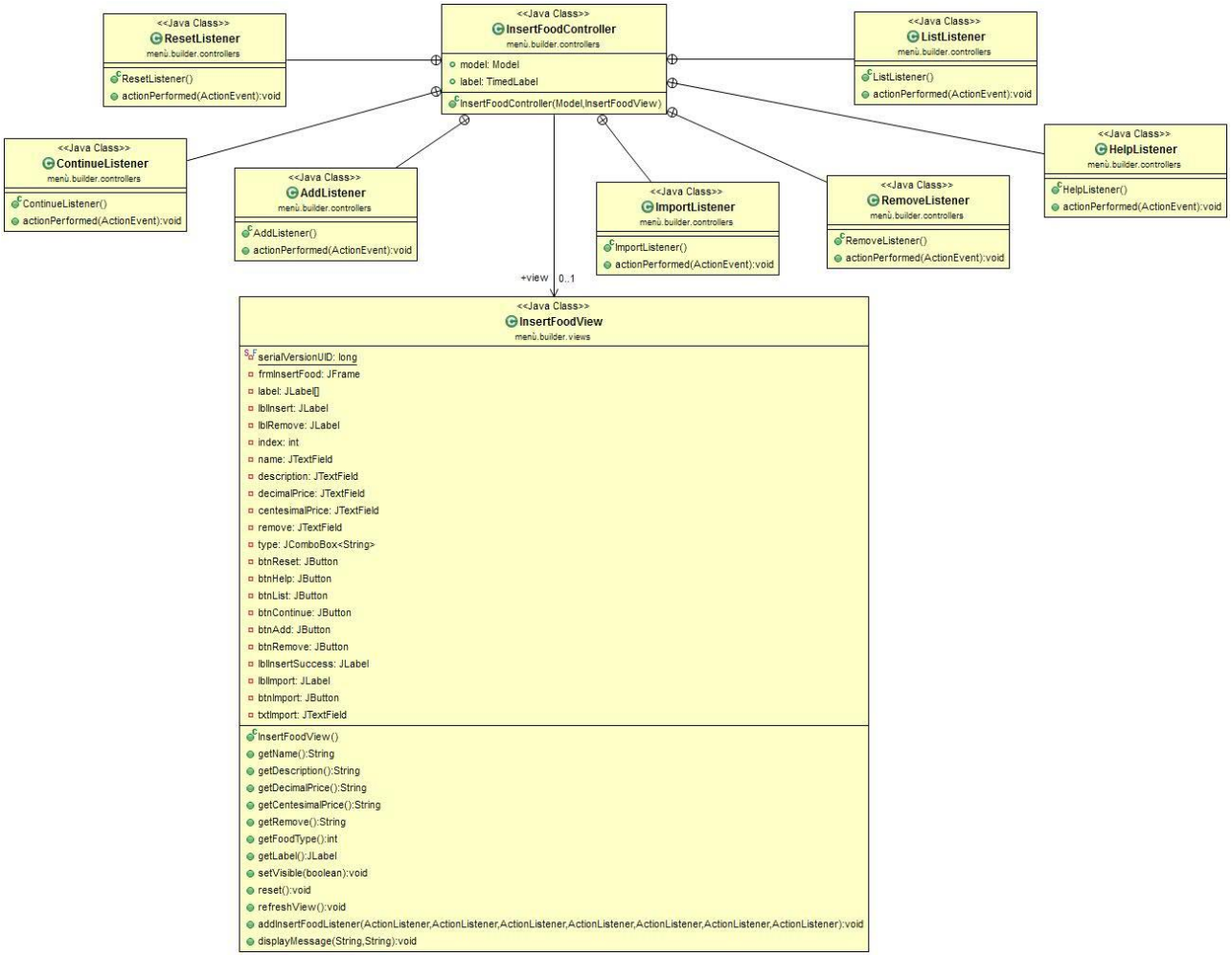
Vengono mostrate nel dettaglio tutte le coppie controller/view relative all'applicazione.

Controller / View:



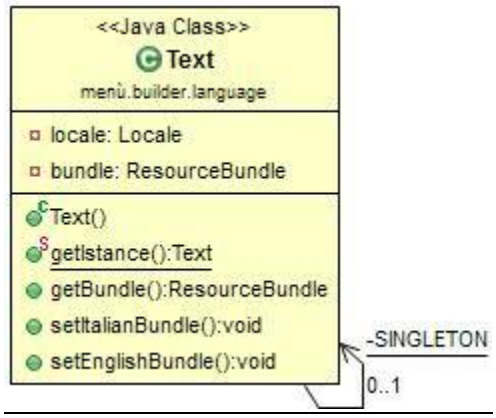

<<Java Class>>
HelpListener
menù.builder.controllers
HelpListener()
actionPerformed(ActionEvent):void
<<Java Class>>
WorkspacePathController
menù.builder.controllers
model: Model
WorkspacePathController(Model,WorkspacePathView)
<<Java Class>>
DefaultListener
menù.builder.controllers
DefaultListener()
actionPerformed(ActionEvent):void
<<Java Class>>
ExploreListener
menù.builder.controllers
ExploreListener()
actionPerformed(ActionEvent):void
<<Java Class>>
ContinueListener
menù.builder.controllers
ContinueListener()
actionPerformed(ActionEvent):void
<<Java Class>>
CustomListener
menù.builder.controllers
CustomListener()
actionPerformed(ActionEvent):void
-view
0..1
<<Java Class>>
WorkspacePathView
menù.builder.views
serialVersionUID: long
frmWorkspacePath: JFrame
btnExplore: JButton
btnContinue: JButton
btnHelp: JButton
txtPath: JTextField
lblPath: JLabel
chkBoxDefault: JRadioButton
chkBoxCustom: JRadioButton
chkGroup: ButtonGroup
lblIconDefault: JLabel
lblIconCustom: JLabel
lblDefault: JLabel
lblCustom: JLabel
WorkspacePathView()
DefaultSelected():void
CustomSelected():void
setVisible(boolean):void
getPath():String
setPath(String):void
chkBoxSelected():boolean
addWorkspacePathListener(ActionListener,ActionListener,ActionListener,ActionListener,ActionListener):void
displayMessage(String,String):void


<<Java Class>>
InformationController
menù.builder.controllers
model: Model
InformationController(Model,InformationView)
<<Java Class>>
ContinueListener
menù.builder.controllers
ContinueListener()
actionPerformed(ActionEvent):void
<<Java Class>>
HelpListener
menù.builder.controllers
HelpListener()
actionPerformed(ActionEvent):void
+view
0..1
<<Java Class>>
InformationView
menù.builder.views
serialVersionUID: long
frmInformation: JFrame
lblInformation: JLabel
lblName: JLabel
lblOwner: JLabel
lblAddress: JLabel
lbltelephoneNumber: JLabel
lblVatNumber: JLabel
lblWorkingDays: JLabel
lblFrom1: JLabel
lblTo1: JLabel
lblFrom2: JLabel
lblTo2: JLabel
lblDescription: JLabel
txtName: JTextField
txtOwner: JTextField
txtAddress: JTextField
txtTelephoneNumber: JTextField
txtVatNumber: JTextField
txtDescription: JTextArea
chkBox: JCheckBox[]
cmbBox: JComboBox<String>
btnContinue: JButton
btnHelp: JButton
index: int
tmp: int
InformationView()
setVisible(boolean):void
getClubName():String
getClubOwner():String
getAddress():String
getTelephoneNumber():String
getVatNumber():String
getDescription():String
getCheckBox():JCheckBox[]
getComboBox():JComboBox<String>
addInformationListener(ActionListener,ActionListener):void
displayMessage(String,String):void

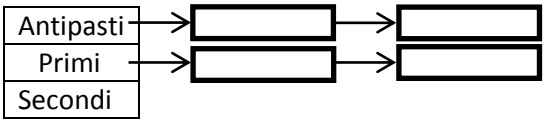


Text:

Classe dedicata al supporto multilingua. Settata di default su normal. Nel momento in cui l'utente andrà a selezionare la lingua inglese nella classe verrà modificato opportunamente il resourceBundle consentendo l'utilizzo della nuova lingua. Anche qui viene utilizzato il pattern SINGLETON in quanto è necessario compaia una e una sola classe avente il compito di gestire la lingua dell'applicazione.

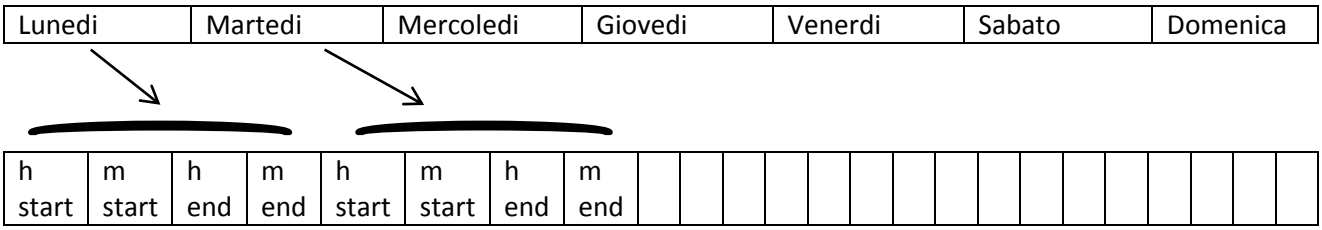


La struttura dati utilizzata per la memorizzazione delle vivande è un array di arrayList.



Altro

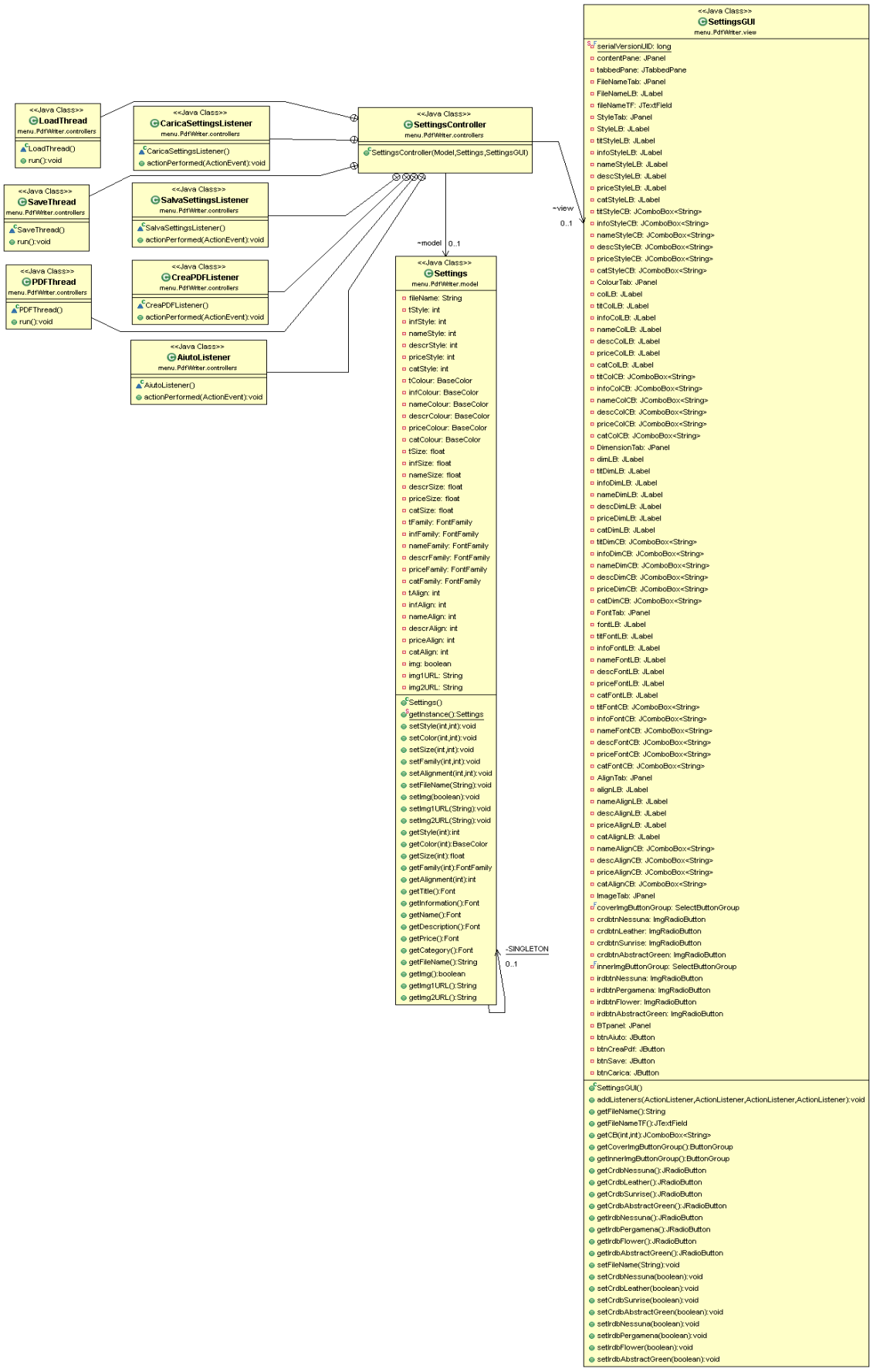
L'array ha come dimensione il numero delle tipologie delle vivande. Ogni posizione dell'array contiene una arrayList, ovvero una lista contenente tutte le singole vivande che mano a mano vengono aggiunte.
In questo modo all'aggiunta di una vivanda, una volta ricevuta la tipologia, sarà possibile inserirla nell'arrayList appropriata.
Si è deciso di tenere le vivande ordinate alfabeticamente.
Prima di procedere con l'effettivo inserimento si controlla che non sia già presente una vivanda con lo stesso NOME, in quanto non esistono vivande con stesso nome ma aventi tipologie diverse.
Questo implica una scansione di ogni arrayList, ma considerando che nel caso peggiore ogni tipologia di vivanda ne contenga una ventina, si tratta di scandire 160 elementi (caso peggiore).
La stessa cosa avviene per la rimozione: si tratta di una scansione sequenziale di ogni arrayList (nel caso peggiore la vivanda non esiste e si ha l'intera scansione della struttura dati).
Data l'unicità delle vivande si sarebbe anche potuto utilizzare un array di set.
Anche per la memorizzazione dei giorni lavorativi è stato usato un'arrayList: l'effettiva memorizzazione avviene tramite un array di checkBox (giorni lavorativi) e un array di comboBox (ognuno contiene una parte di un orario).



Quando il checkBox del giorno lavorativo è selezionato si va a leggere il valore delle rispettive combobox per comporre gli orari del giorno. Si legge quindi sequenzialmente l'array di checkBox e, se il checkBox è selezionato, tramite un indice inizializzato a zero si ricavano i valori delle comboBox, altrimenti si passa al checkBox successivo e si incrementa l'indice di 4 (per allinearlo con il nuovo giorno che si andrà ad analizzare).

Non sono state utilizzate librerie esterne per lo sviluppo di questa parte di progetto.

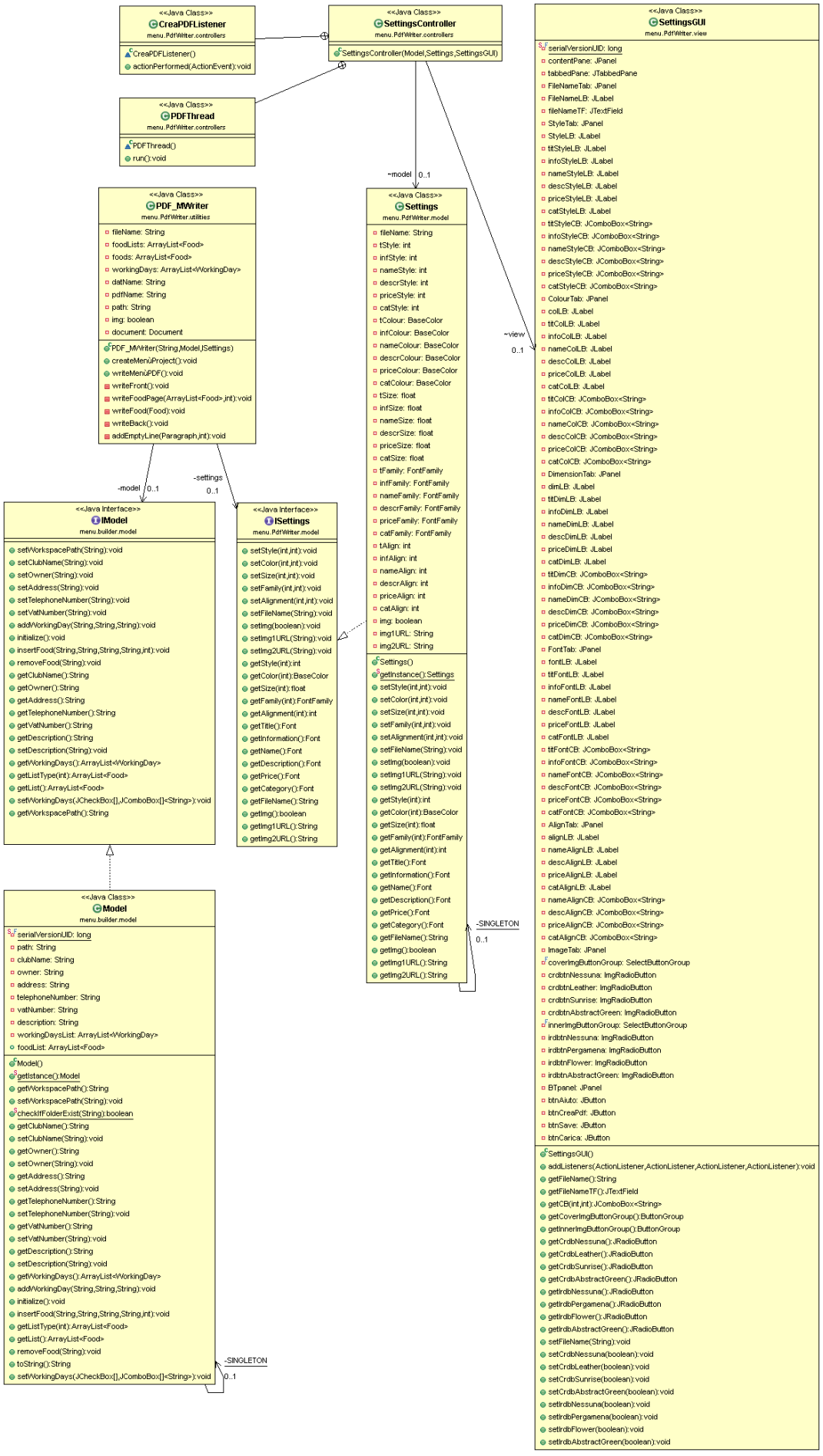
5.2 Progettazione di dettaglio: Parte di Raffaele Gori



La struttura MVC di questa sezione è piuttosto semplice:
Si dispone di un'unica vista, un unico modello e un unico controllore.
Quando la GUI è stata settata e si verifica un certo evento, il controller, che ha quella GUI come campo, setta opportunamente il model (in questo caso la classe ***Settings***), che ha come altro campo, mediando le interazioni tra i due, come richiesto dalle specifiche del pattern.

In particolare, quando viene premuto il bottone "Crea PDF", l'ActionListener corrispondente crea e lancia un thread di tipo PDFThread, il quale setta il campo model del controller in maniera coerente con i valori selezionati nel suo campo view (di tipo SetingsGUI), dopo di che istanzia un oggetto PDF_MWriter, cui passa nel costruttore il campo model del controllore (avrebbe potuto equivalentemente passargli l'unica istanza della classe Settings tramite il metodo getInstance(), visto che Settings implementa il pattern Singleton) e l'unica istanza della classe Model.
A questo punto chiama i metodi createMenùProject() e writeMenùPDF() dell'oggetto appena creato per ottenere il menù finale in formato PDF.

Quanto appena descritto è espresso dal diagramma seguente.


«Java Class»
CreaPDFListener
menu.PdfWriter.controllers
CreaPDFListener()
actionPerformed(ActionEvent):void
«Java Class»
SettingsController
menu.PdfWriter.controllers
SettingsController(Model,Settings,SettingsGUI)
«Java Class»
SettingsGUI
menu.PdfWriter.view
serialVersionUID: long
contentPane: JPanel
tabbedPane: JTabbedPane
FileNameTab: JPanel
FileNameLB: JLabel
fileNameTF: JTextField
StyleTab: JPanel
StyleLB: JLabel
titStyleLB: JLabel
infoStyleLB: JLabel
nameStyleLB: JLabel
descStyleLB: JLabel
priceStyleLB: JLabel
catStyleLB: JLabel
titStyleCB: JComboBox<String>
infoStyleCB: JComboBox<String>
nameStyleCB: JComboBox<String>
descStyleCB: JComboBox<String>
priceStyleCB: JComboBox<String>
catStyleCB: JComboBox<String>
ColourTab: JPanel
colLB: JLabel
titColLB: JLabel
infoColLB: JLabel
nameColLB: JLabel
descColLB: JLabel
priceColLB: JLabel
catColLB: JLabel
titColCB: JComboBox<String>
infoColCB: JComboBox<String>
nameColCB: JComboBox<String>
descColCB: JComboBox<String>
priceColCB: JComboBox<String>
catColCB: JComboBox<String>
DimensionTab: JPanel
dimLB: JLabel
titDimLB: JLabel
infoDimLB: JLabel
nameDimLB: JLabel
descDimLB: JLabel
priceDimLB: JLabel
catDimLB: JLabel
titDimCB: JComboBox<String>
infoDimCB: JComboBox<String>
nameDimCB: JComboBox<String>
descDimCB: JComboBox<String>
priceDimCB: JComboBox<String>
catDimCB: JComboBox<String>
FontTab: JPanel
fontLB: JLabel
titFontLB: JLabel
infoFontLB: JLabel
nameFontLB: JLabel
descFontLB: JLabel
priceFontLB: JLabel
catFontLB: JLabel
titFontCB: JComboBox<String>
infoFontCB: JComboBox<String>
nameFontCB: JComboBox<String>
descFontCB: JComboBox<String>
priceFontCB: JComboBox<String>
catFontCB: JComboBox<String>
AlignTab: JPanel
alignLB: JLabel
nameAlignLB: JLabel
descAlignLB: JLabel
priceAlignLB: JLabel
catAlignLB: JLabel
nameAlignCB: JComboBox<String>
descAlignCB: JComboBox<String>
priceAlignCB: JComboBox<String>
catAlignCB: JComboBox<String>
ImageTab: JPanel
coverImgButtonGroup: SelectButtonGroup
crdbtnNessuna: ImgRadioButton
crdbtnLeather: ImgRadioButton
crdbtnSunrise: ImgRadioButton
crdbtnAbstractGreen: ImgRadioButton
innerImgButtonGroup: SelectButtonGroup
irdbtnNessuna: ImgRadioButton
irdbtnPergamena: ImgRadioButton
irdbtnFlower: ImgRadioButton
irdbtnAbstractGreen: ImgRadioButton
BTpanel: JPanel
btnAiuto: JButton
btnCreaPdf: JButton
btnSave: JButton
btnCarica: JButton
SettingsGUI()
addListeners(ActionListener,ActionListener,ActionListener,ActionListener):void
getFileName():String
getFileNameTF():JTextField
getCB(int,int):JComboBox<String>
getCoverImgButtonGroup():ButtonGroup
getInnerImgButtonGroup():ButtonGroup
getCrdbNessuna():JRadioButton
getCrdbLeather():JRadioButton
getCrdbSunrise():JRadioButton
getCrdbAbstractGreen():JRadioButton
getIrdbNessuna():JRadioButton
getIrdbPergamena():JRadioButton
getIrdbFlower():JRadioButton
getIrdbAbstractGreen():JRadioButton
setFileName(String):void
setCrdbNessuna(boolean):void
setCrdbLeather(boolean):void
setCrdbSunrise(boolean):void
setCrdbAbstractGreen(boolean):void
setIrdbNessuna(boolean):void
setIrdbPergamena(boolean):void
setIrdbFlower(boolean):void
setIrdbAbstractGreen(boolean):void
«Java Class»
PDFThread
menu.PdfWriter.controllers
PDFThread()
run():void
~model
0..1
«Java Class»
PDF_MWriter
menu.PdfWriter.utilities
fileName: String
foodLists: ArrayList<Food>
foods: ArrayList<Food>
workingDays: ArrayList<WorkingDay>
datName: String
pdfName: String
path: String
img: boolean
document: Document
PDF_MWriter(String,Model,ISettings)
createMenùProject():void
writeMenùPDF():void
writeFront():void
writeFoodPage(ArrayList<Food>,int):void
writeFood(Food):void
writeBack():void
addEmptyLine(Paragraph,int):void
«Java Class»
Settings
menu.PdfWriter.model
fileName: String
tStyle: int
infStyle: int
nameStyle: int
descrStyle: int
priceStyle: int
catStyle: int
tColour: BaseColor
infColour: BaseColor
nameColour: BaseColor
descrColour: BaseColor
priceColour: BaseColor
catColour: BaseColor
tSize: float
infSize: float
nameSize: float
descrSize: float
priceSize: float
catSize: float
tFamily: FontFamily
infFamily: FontFamily
nameFamily: FontFamily
descrFamily: FontFamily
priceFamily: FontFamily
catFamily: FontFamily
tAlign: int
infAlign: int
nameAlign: int
descrAlign: int
priceAlign: int
catAlign: int
img: boolean
img1URL: String
img2URL: String
Settings()
getInstance():Settings
setStyle(int,int):void
setColor(int,int):void
setSize(int,int):void
setFamily(int,int):void
setAlignment(int,int):void
setFileName(String):void
setImg(boolean):void
setImg1URL(String):void
setImg2URL(String):void
getStyle(int):int
getColor(int):BaseColor
getSize(int):float
getFamily(int):FontFamily
getAlignment(int):int
getTitle():Font
getInformation():Font
getName():Font
getDescription():Font
getPrice():Font
getCategory():Font
getFileName():String
getImg():boolean
getImg1URL():String
getImg2URL():String
~view
0..1
-model
0..1
-settings
0..1
«Java Interface»
IModel
menu.builder.model
setWorkspacePath(String):void
setClubName(String):void
setOwner(String):void
setAddress(String):void
setTelephoneNumber(String):void
setVatNumber(String):void
addWorkingDay(String,String,String):void
initialize():void
insertFood(String,String,String,String,int):void
removeFood(String):void
getClubName():String
getOwner():String
getAddress():String
getTelephoneNumber():String
getVatNumber():String
getDescription():String
setDescription(String):void
getWorkingDays():ArrayList<WorkingDay>
getListType(int):ArrayList<Food>
getList():ArrayList<Food>
setWorkingDays(JCheckBox[],JComboBox[]<String>):void
getWorkspacePath():String
«Java Interface»
ISettings
menu.PdfWriter.model
setStyle(int,int):void
setColor(int,int):void
setSize(int,int):void
setFamily(int,int):void
setAlignment(int,int):void
setFileName(String):void
setImg(boolean):void
setImg1URL(String):void
setImg2URL(String):void
getStyle(int):int
getColor(int):BaseColor
getSize(int):float
getFamily(int):FontFamily
getAlignment(int):int
getTitle():Font
getInformation():Font
getName():Font
getDescription():Font
getPrice():Font
getCategory():Font
getFileName():String
getImg():boolean
getImg1URL():String
getImg2URL():String
«Java Class»
Model
menu.builder.model
serialVersionUID: long
path: String
clubName: String
owner: String
address: String
telephoneNumber: String
vatNumber: String
description: String
workingDaysList: ArrayList<WorkingDay>
foodList: ArrayList<Food>
Model()
getistance():Model
getWorkspacePath():String
setWorkspacePath(String):void
checkIfFolderExist(String):boolean
getClubName():String
setClubName(String):void
getOwner():String
setOwner(String):void
getAddress():String
setAddress(String):void
getTelephoneNumber():String
setTelephoneNumber(String):void
getVatNumber():String
setVatNumber(String):void
getDescription():String
setDescription(String):void
getWorkingDays():ArrayList<WorkingDay>
addWorkingDay(String,String,String):void
initialize():void
insertFood(String,String,String,String,int):void
getListType(int):ArrayList<Food>
getList():ArrayList<Food>
removeFood(String):void
toString():String
setWorkingDays(JCheckBox[],JComboBox[]<String>):void
-SINGLETON
0..1
-SINGLETON
0..1

A fronte di un evento sui JButton di caricamento e salvataggio, inoltre, gli ActionListener preposti si comportano come segue:

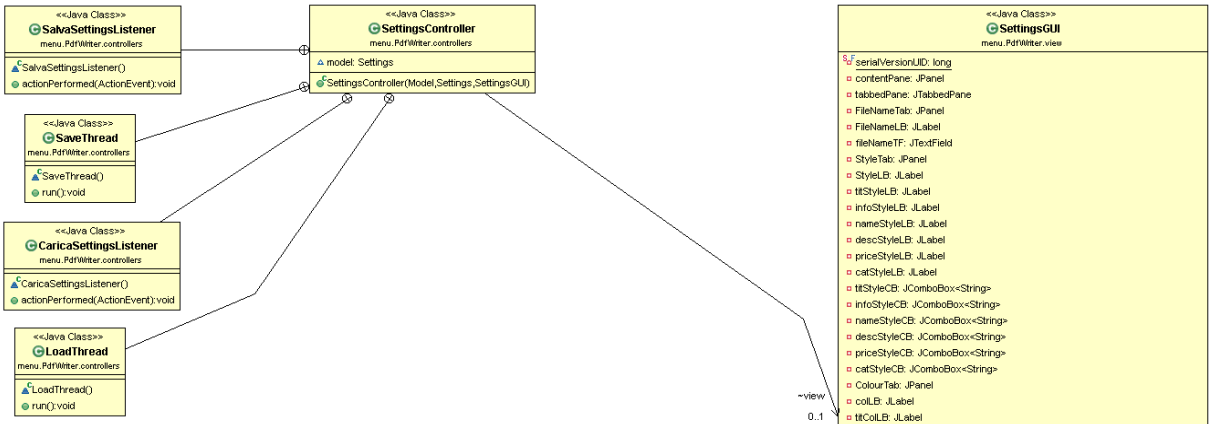
A fronte della pressione del bottone di salvataggio, l'ActionListener SalvaSettingsListener crea un oggetto SaveThread e lo lancia.
Il thread mostra una finestra JFileChooser, attraverso cui l'utente può selezionare il percorso al quale salvare la struttura dati e settare il nome del file che la ospiterà.
A questo punto viene istanziato un oggetto della classe SettingsProject, al cui costruttore vengono forniti in input la stringa contenuta nel JTexField del primo pannello di SettingsGUI (dedicato al nome che si desidera per il file PDF), i valori degli indici di selezione di tutte JComboBox della GUI e dei JRadioButtons relativi alla scelta delle immagini di sfondo per le pagine del PDF, ed il nome inserito nel JFileChooser per il file in cui questi dati verranno memorizzati.
Questa struttura dati, che implementa l'interfaccia *Serializable*, verrà salvata in un file in formato .dat, consentendo all'utente di riportare in ogni momento le selezioni effettuate sull'interfaccia grafica ai valori settati al momento del salvataggio.
Si è deciso di salvare i valori degli indici delle selezioni, i valori booleani dei JRadioButtons e la stringa relativa al nome del file inserito nel JTexfield invece di serializzare l'istanza attuale di Settings per il semplice fatto che è il thread preposto alla scrittura del PDF a chiamare tutti i setter per il modello subito prima di creare l'oggetto scrittore, e quindi caricare una diversa istanza di *Settings* non avrebbe dato risultati diversi da un errore per la mancata compilazione del campo del nome, o una sovrascrittura degli eventuali campi inseriti manualmente a quelli deserializzati.

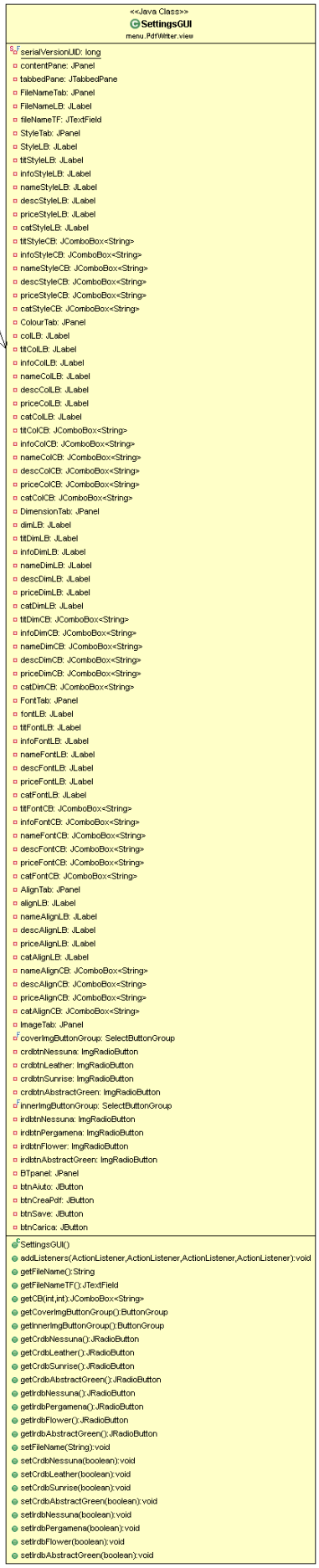
Questa scelta ha portato due principali effetti collaterali:

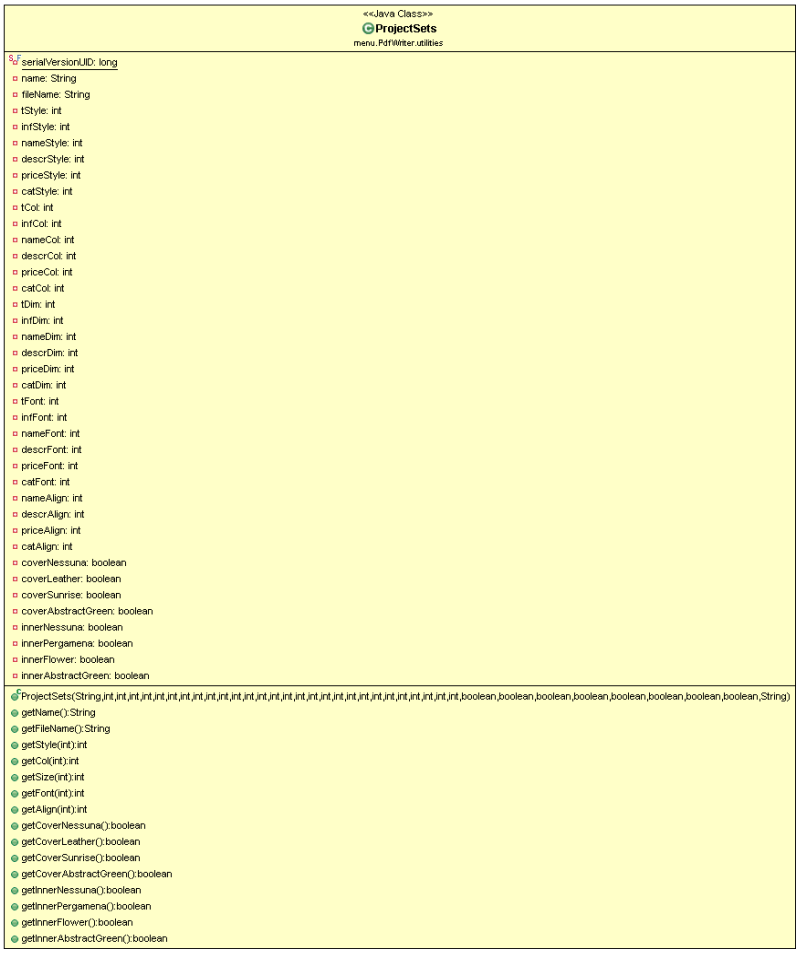
1. I setters della classe Settings, oltre al parametro che seleziona quale valore settare, non prendono in ingresso il valore che si desidera il campo assuma, ma un intero (che in scenario di esecuzione è il valore dell'indice della selezione di una JComBobox in *SettingsGUI*) in base al valore del quale settano il campo ad un valore contenuto in *SettingsConstants* o lanciano una eccezione *InvalidComboboxSelection*.
2. Dal momento che istanze di ProjectSets vengono create solo per la serializzazione di una certa configurazione di selezioni, e che essa viene passata al costruttore, i setters per questa classe non avrebbero motivo di comparire visto che, anche volendo aggiornare i valori contenuti nella struttura dati, il thread che li salva andrebbe a crearne una totalmente nuova da sovrascrivere al file corrente.
   La conseguenza è che la classe ProjectSets presenta come unici metodi il costruttore e i getters per i suoi campi.

Qualora il bottone premuto sia quello per il caricamento di una vecchia configurazione di selezioni, l'ActionListener CaricaSettingsListener crea e lancia un thread di tipo LoadThread, il quale mostra una finestra JFileChooser per la selezione del file da cui si desidera prelevare la struttura dati, e successivamente setta i vari campi della view SettingsController in maniera coerente con quanto letto da essa, riportando la configurazione dell'interfaccia grafica ai valori settati al momento del salvataggio.

Queste interazioni sono espresse dal seguente diagramma UML.


«Java Class»
SalvaSettingsListener
menu.PdfWriter.controllers
SalvaSettingsListener()
actionPerformed(ActionEvent):void
«Java Class»
SettingsController
menu.PdfWriter.controllers
model: Settings
SettingsController(Model,Settings,SettingsGUI)
«Java Class»
SaveThread
menu.PdfWriter.controllers
SaveThread()
run():void
«Java Class»
CaricaSettingsListener
menu.PdfWriter.controllers
CaricaSettingsListener()
actionPerformed(ActionEvent):void
«Java Class»
LoadThread
menu.PdfWriter.controllers
LoadThread()
run():void
~view
0..1


«Java Class»
SettingsGUI
menu.PdfWriter.view
serialVersionUID: long
contentPane: JPanel
tabbedPane: JTabbedPane
FileNameTab: JPanel
FileNameLB: JLabel
fileNameTF: JTextField
StyleTab: JPanel
StyleLB: JLabel
tltStyleLB: JLabel
infoStyleLB: JLabel
nameStyleLB: JLabel
descStyleLB: JLabel
priceStyleLB: JLabel
catStyleLB: JLabel
tltStyleCB: JComboBox<String>
infoStyleCB: JComboBox<String>
nameStyleCB: JComboBox<String>
descStyleCB: JComboBox<String>
priceStyleCB: JComboBox<String>
catStyleCB: JComboBox<String>
ColourTab: JPanel
colLB: JLabel
tltColLB: JLabel
infoColLB: JLabel
nameColLB: JLabel
descColLB: JLabel
priceColLB: JLabel
catColLB: JLabel
tltColCB: JComboBox<String>
infoColCB: JComboBox<String>
nameColCB: JComboBox<String>
descColCB: JComboBox<String>
priceColCB: JComboBox<String>
catColCB: JComboBox<String>
DimensionTab: JPanel
dimLB: JLabel
tltDimLB: JLabel
infoDimLB: JLabel
nameDimLB: JLabel
descDimLB: JLabel
priceDimLB: JLabel
catDimLB: JLabel
tltDimCB: JComboBox<String>
infoDimCB: JComboBox<String>
nameDimCB: JComboBox<String>
descDimCB: JComboBox<String>
priceDimCB: JComboBox<String>
catDimCB: JComboBox<String>
FontTab: JPanel
fontLB: JLabel
tltFontLB: JLabel
infoFontLB: JLabel
nameFontLB: JLabel
descFontLB: JLabel
priceFontLB: JLabel
catFontLB: JLabel
tltFontCB: JComboBox<String>
infoFontCB: JComboBox<String>
nameFontCB: JComboBox<String>
descFontCB: JComboBox<String>
priceFontCB: JComboBox<String>
catFontCB: JComboBox<String>
AlignTab: JPanel
alignLB: JLabel
nameAlignLB: JLabel
descAlignLB: JLabel
priceAlignLB: JLabel
catAlignLB: JLabel
nameAlignCB: JComboBox<String>
descAlignCB: JComboBox<String>
priceAlignCB: JComboBox<String>
catAlignCB: JComboBox<String>
ImageTab: JPanel
coverImgButtonGroup: SelectButtonGroup
crdbtnNessuna: ImgRadioButton
crdbtnLeather: ImgRadioButton
crdbtnSunrise: ImgRadioButton
crdbtnAbstractGreen: ImgRadioButton
innerImgButtonGroup: SelectButtonGroup
irdbtnNessuna: ImgRadioButton
irdbtnPergamena: ImgRadioButton
irdbtnFlower: ImgRadioButton
irdbtnAbstractGreen: ImgRadioButton
BTpanel: JPanel
btnAiuto: JButton
btnCreaPdf: JButton
btnSave: JButton
btnCarica: JButton
SettingsGUI()
addListeners(ActionListener,ActionListener,ActionListener,ActionListener):void
getFileName():String
getFileNameTF():JTextField
getCB(int,int):JComboBox<String>
getCoverImgButtonGroup():ButtonGroup
getInnerImgButtonGroup():ButtonGroup
getCrdbNessuna():JRadioButton
getCrdbLeather():JRadioButton
getCrdbSunrise():JRadioButton
getCrdbAbstractGreen():JRadioButton
getIrdbNessuna():JRadioButton
getIrdbPergamena():JRadioButton
getIrdbFlower():JRadioButton
getIrdbAbstractGreen():JRadioButton
setFileName(String):void
setCrdbNessuna(boolean):void
setCrdbLeather(boolean):void
setCrdbSunrise(boolean):void
setCrdbAbstractGreen(boolean):void
setIrdbNessuna(boolean):void
setIrdbPergamena(boolean):void
setIrdbFlower(boolean):void
setIrdbAbstractGreen(boolean):void


«Java Class»
ProjectSets
menu.PdfWriter.utilities
serialVersionUID: long
name: String
fileName: String
tStyle: int
infStyle: int
nameStyle: int
descrStyle: int
priceStyle: int
catStyle: int
tCol: int
infCol: int
nameCol: int
descrCol: int
priceCol: int
catCol: int
tDim: int
infDim: int
nameDim: int
descrDim: int
priceDim: int
catDim: int
tFont: int
infFont: int
nameFont: int
descrFont: int
priceFont: int
catFont: int
nameAlign: int
descrAlign: int
priceAlign: int
catAlign: int
coverNessuna: boolean
coverLeather: boolean
coverSunrise: boolean
coverAbstractGreen: boolean
innerNessuna: boolean
innerPergamena: boolean
innerFlower: boolean
innerAbstractGreen: boolean
ProjectSets(String,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,boolean,boolean,boolean,boolean,boolean,boolean,boolean,boolean,String)
getName():String
getFileName():String
getStyle(int):int
getCol(int):int
getSize(int):int
getFont(int):int
getAlign(int):int
getCoverNessuna():boolean
getCoverLeather():boolean
getCoverSunrise():boolean
getCoverAbstractGreen():boolean
getInnerNessuna():boolean
getInnerPergamena():boolean
getInnerFlower():boolean
getInnerAbstractGreen():boolean

Se l'evento di avvenuta pressione si ha, infine, sul bottone di aiuto, l'ActionListener preposto mostra una finestra che presenta un breve messaggio che dovrebbe fornire all'utente le minime informazioni necessarie per un corretto uso dell'applicazione in questa fase di esecuzione.

Si è deciso di usare classi che estendessero Thread per far compiere le operazioni non strettamente inerenti la gestione dell'interfaccia grafica a flussi di controllo diversi da quello deputato alla sola GUI, evitando, così, di appesantirlo affidandogli compiti che esulassero dalla sua competenza.

Per la scrittura del menù in formato PDF si è fatto uso della libreria gratuita esterna al JDK iText.

6 Testing

builder:
Lo scenario di utilizzo previsto è il seguente:

- All'apertura dell'applicazione sarà subito possibile scegliere la lingua preferita tra italiano ed inglese. E' possibile visualizzare alcune piccole informazioni riguardanti l'applicazione e procedere con l'utilizzo di essa.
- La nuova schermata richiede l'inserimento di una directory VALIDA dove salvare il menù tramite JFileChooser. Non è possibile lasciare il campo vuoto o inserire un path non valido in quanto vengono gestite tutte le opportune eccezioni. Diversa è la questione in caso si scelga l'opzione del path standard in quando la directory scelta sarà automaticamente il desktop.
- Successivamente si presenta una schermata più complessa, che presenta diversi campi da compilare. Una volta compilate le varie aree di testo è necessario inserire i giorni lavorativi dell'attività. Appena si spunta il giorno sarà possibile, tramite combo box, inserire l'orario di inizio e fine servizio. Se si desidera si può anche inserire una descrizione dell'attività (opzionale). Anche qui non è possibile commettere errori: lasciando un campo non compilato (fatta eccezione per la descrizione) si genera un eccezione. Per poter proseguire è previsto l'inserimento di almeno un giorno lavorativo.
- Segue la schermata di gestione delle vivande, divisa in 3 spazi: l'inserimento, la rimozione e l'importazione (non implementata). Per poter inserire la vivanda:
  - Scegliere dalla comboBox la tipologia della vivanda.
  - Inserire il rispettivo nome.
  - Inserire il prezzo secondo formato ##,##.
  - (Opzionalmente) Inserire la descrizione/ingredienti.
  - Procedere con l'inserimento.

  Vengono gestite anche qui tutte le possibili eccezioni. In particolare:
  - Nessuna tipologia inserita.
  - Nome e/o prezzo non inseriti.
  - Formato del prezzo non valido.

  Per poter rimuovere la vivanda:
  - Inserire il rispettivo nome.
  - Procedere con la rimozione.

  La rimozione può dar luogo a due esiti: la vivanda viene trovata e quindi rimossa oppure la vivanda cercata non esiste nella struttura.
  Tramite il tasto LISTA sarà possibile avere un riassunto tabellare delle vivande finora inserite.
  Ogni singola schermata dispone del tasto AIUTO per ogni evenienza e dubbio che possa sorgere nell'utilizzatore.

PdfWriter:
Lo scenario di utilizzo previsto è il seguente:

- Dopo aver popolato la struttura dati deputata alla memorizzazione della lista dei cibi e delle informazioni relative all'esercizio, all'utente viene proposta una GUI con sette pannelli.
- Inserito un nome per il progetto nel primo pannello (unico campo il cui settaggio è indispensabile per il proseguimento della creazione del PDF), l'utente dovrà settare attraverso le JComboBoxes i valori prescelti per le varie impostazioni della veste grafica e selezionare, se desiderata, una immagine di sfondo tramite i JRadioButtons dell'ultimo pannello.
- In qualunque momento l'utente ha la possibilità di salvare in un documento in fromato .dat le selezioni attuali delle JComBoboxes e dei JRadioButtons, e il nome del progetto, o di caricare selezioni già salvate in precedenza tramite i pulsanti "Salva" e "Carica".
- Una volta soddisfatto dei settaggi prescelti, l'utente può completare l'operazione tramite il tasto "Crea PDF".
  A questo punto nella destinazione impostata nella prima schermata lanciata dall'applicativo viene creato il documento contenente il menù definitivo.

Sono stati effettuati test tramite JUnit per le classi Settings e ProjectSets, ma trattandosi di test su metodi getter, setter e costruttori, non si ritiene siano particolarmente significativi.

Per quanto riguarda, invece, la sezione di test effettuata "manualmente" dal programmatore, ci si è assicurati che ogni componente della GUI si presentasse come immaginato e si comportasse come richiesto, lanciando eccezioni qualora le richieste fossero scorrette e mostrando le opportune finestre di dialogo per guidare l'utente verso un utilizzo proprio dell'applicativo.

Per quel che riguarda l'interfaccia grafica, i test hanno verificato principalmente che essa fornisca un corretto controllo dei settaggi del modello, e che le selezioni effetuate su di essa vengano serializzate senza errori per il salvataggio/caricamento.

Per quanto concerne il testing della classe PDF_MWriter, che è la più importante di questa sezione, i test principali si sono concentrati sulla serializzazione in InputStream e OutputStream dei dati per la scrittura dei PDF, in modo da assicurarsi che gli stream venissero chiusi dopo ogni loro apertura, e la formattazione finale del PDF, che deve rimanere coerente indipendentemente dal numero di pagine, di categorie di cibi, di cibi per ogni categoria, e di righe di descrizione per ogni cibo.
Quest'ultimo aspetto è stato particolarmente impegnativo, perché gestire valori disomogenei (come le dimensioni delle lettere nella pagina del documento) in maniera precisa non è semplice, e si è dovuto ricorrere a qualche stratagemma per avere una funzionalità che, alla fine, fosse in grado di gestire da 0 a 21* pagine di cibi di tutte le categorie, con lunghezze variabili per le descrizioni.
I test sono stati effettuati tramite classi di main per velocizzare la parte di settaggio dei modelli, specialmente per numeri elevati di cibi, ma le classi usate non sono state inserite nella versione finale del progetto.

*Questi i valori estremi per i test effettuati, ma si ritiene di poter proiettare la cosa ad un numero grande a piacere

7 Note finali

Giacomo Rossi:

Il processo di sviluppo è stato a spirale all'inizio, in quanto dopo 10/15 ore di lavoro si è deciso di implementare l'applicazione utilizzando il patter MVC. La relativa correzione non ha causato problemi ed è stata eseguita in tempi brevi. Giunti quasi alla fine dell'applicazione si è deciso di introdurre il supporto multilingua tramite la classe Text e i relativi file di testo in formato .properties contenuti nella cartella language. E' stato necessario quindi sostituire tutte le frasi dell'applicazione con il relativo comando `getBundle().getString("")` in modo tale da leggere le stringhe direttamente dal file.

Una volta raggiunto il limite delle ore di progetto si è pensato di inserire una funzione di import con l'obiettivo di poter importare da un vecchio menù generato l'intera lista di vivande. Avendo però raggiunto il limite di ore non è stata possibile implementare la funzione.

Raffaele Gori:

La pianificazione iniziale del progetto ha subito qualche modifica in corso d'opera, in quanto durante la scrittura della prima classe affrontata, PDF_MWriter, che è il motore della parte dell'applicazione da me curata, ci si è trovati ad un bivio sulla la scelta della politica di personalizzazione della vesta grafica:

da un lato si poteva scegliere di avere una struttura rigida, in cui le personalizzazioni possibili fossero minime, se non addirittura nulle, mentre dall'altro c'era una politica di personalizzazione che lasciasse all'utente un margine di manovra molto maggiore, e alla fine, nonostante le problematiche più complesse derivanti, si è optato per la seconda ipotesi.

A questo punto sono stati necessari un secondo model ed una view dedicata ai relativi settaggi, oltre che una modifica della classe scrittrice, ancora a livello embrionale, per permetterle di gestire le due strutture dati e di leggere valori di variabili al posto di costanti nei metodi di scrittura dei paragrafi.

Verso la fine del progetto, inoltre, si è deciso, dato l'elevato numero di configurazioni necessarie per la totale personalizzazione, di introdurre la possibilità di salvare e caricare gli eventuali settaggi, per permettere all'utente una loro più agevole gestione, senza dover impostare ogni volta manualmente i vari parametri richiesti.

Questo torna particolarmente utile in uno scenario verosimile in cui si voglia modificare l'elenco delle vivande di un menù senza alterarne la veste grafica, perché fornisce, a patto di aver salvato i settaggi relativi, una certezza quasi matematica che ciò avverrà, dato che riduce a zero le possibilità di sviste ed errori umani.

Ogni componente del gruppo ha sviluppato la propria parte in modo indipendente, dato che non erano presenti parti in comune. Una volta completate è bastato unire le due parti nel punto in cui finisce la raccolta dati e inizia il settaggio della grafica del menù.

Non ci sono stati problemi di interazioni tra i membri del gruppo.

FONTI:

http://www.jtattoo.net
http://docs.oracle.com/javase/7/docs/api/
http://stackoverflow.com
http://docs.oracle.com/javase/tutorial/
http://itextpdf.com/
http://tips4java.wordpress.com/2008/11/09/select-button-group/