Corso di Programmazione ad Oggetti – Anno 2013/2014
Corso di laurea in Ingegneria e Scienze Informatiche – Università di Bologna

Relazione relativa all'ingegnerizzazione di Beach Management System: un sistema per la gestione di uno stabilimento balneare

Componenti del gruppo: Alessandro Bagnoli.

1 Analisi del problema

E' stato realizzato un applicativo desktop per la gestione di uno stabilimento balneare. Viene gestita la richiesta di un solo tipo di utenza, ovvero il proprietario e/o gestore dello stabilimento.

Nello specifico un generico gestore deve essere in grado di:

- Modificare il numero di attrezzatura (lettini, sedie, etc.) di proprietà all'interno del suo stabilimento.
- Creare ed eventualmente modificare prenotazioni.
- Visualizzare tutte le prenotazioni effettuate con relativo periodo di permanenza e costo.
- Eliminare le prenotazioni pagate e scadute (ovvero quando il cliente ha pagato ed ha terminato il suo periodo di permanenza).
- Visualizzare una lista di prenotazioni non ancora saldate, i pagamenti ricevuti e quelli ancora da ricevere.
- Visualizzare un grafico riassuntivo con l'andamento degli incassi in base al periodo.
- Visualizzare le previsioni metereologiche dei prossimi cinque giorni, senza aver bisogno di uscire dalla applicazione stessa.

L'applicativo garantisce la persistenza dei dati dai vari utilizzatori salvando e caricando i dati interessati ad ogni chiusura e ad ogni apertura dell'applicativo (Il percorso di default di salvataggio e caricamento sarà la propria cartella utente, ma si potrà caricare e salvare i dati da/in un percorso arbitrario, selezionabile tramite l'apposita menu bar in alto).

Visti i requisiti sopra elencati, è stato necessario implementare un applicativo GUI based che si presenta con un primo pannello di login nel quale si dà la possibilità di registrarsi al servizio nel caso sia la prima volta che si utilizza l'applicazione, oppure di inserire la proprio login e password.
Una volta verificate queste ultime, all'utente si aprirà un pannello che darà la possibilità di accedere a tutte le funzionalità descritte.

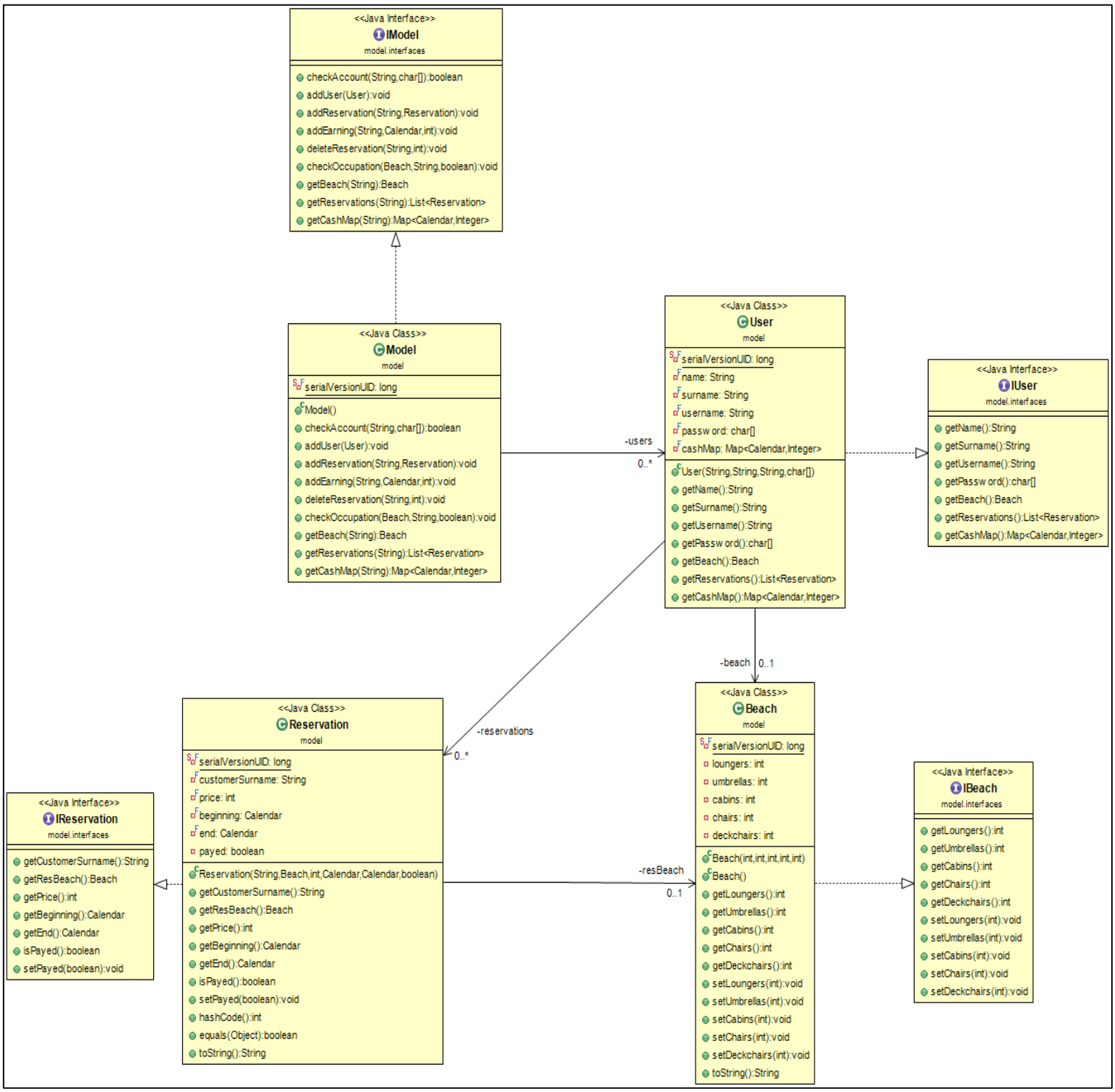
2 Progettazione architetturale

In questa fase è stata effettuata la progettazione dell'applicazione, prendendo le più importanti decisioni sull'organizzazione del lavoro, focalizzandosi su quali entità andassero introdotte e come metterle in relazione.

Per tutta la fase di progettazione è stato utilizzato il pattern architetturale MVC (Model – View – Controller) per garantire la netta separazione fra i diversi aspetti di presentazione e visualizzazione, di rappresentazione dei dati utilizzati, e di aspetti legati al funzionamento e alla logica dell'applicativo. La separazione sopra descritta consente un futuro riutilizzo e una facile manutenibilità del modello, delle viste o del controller nel scenario di applicazioni differenti.

Segue ora la presentazione degli aspetti più rilevanti relativi all'architettura del sistema.

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



Descrizione degli aspetti principali:

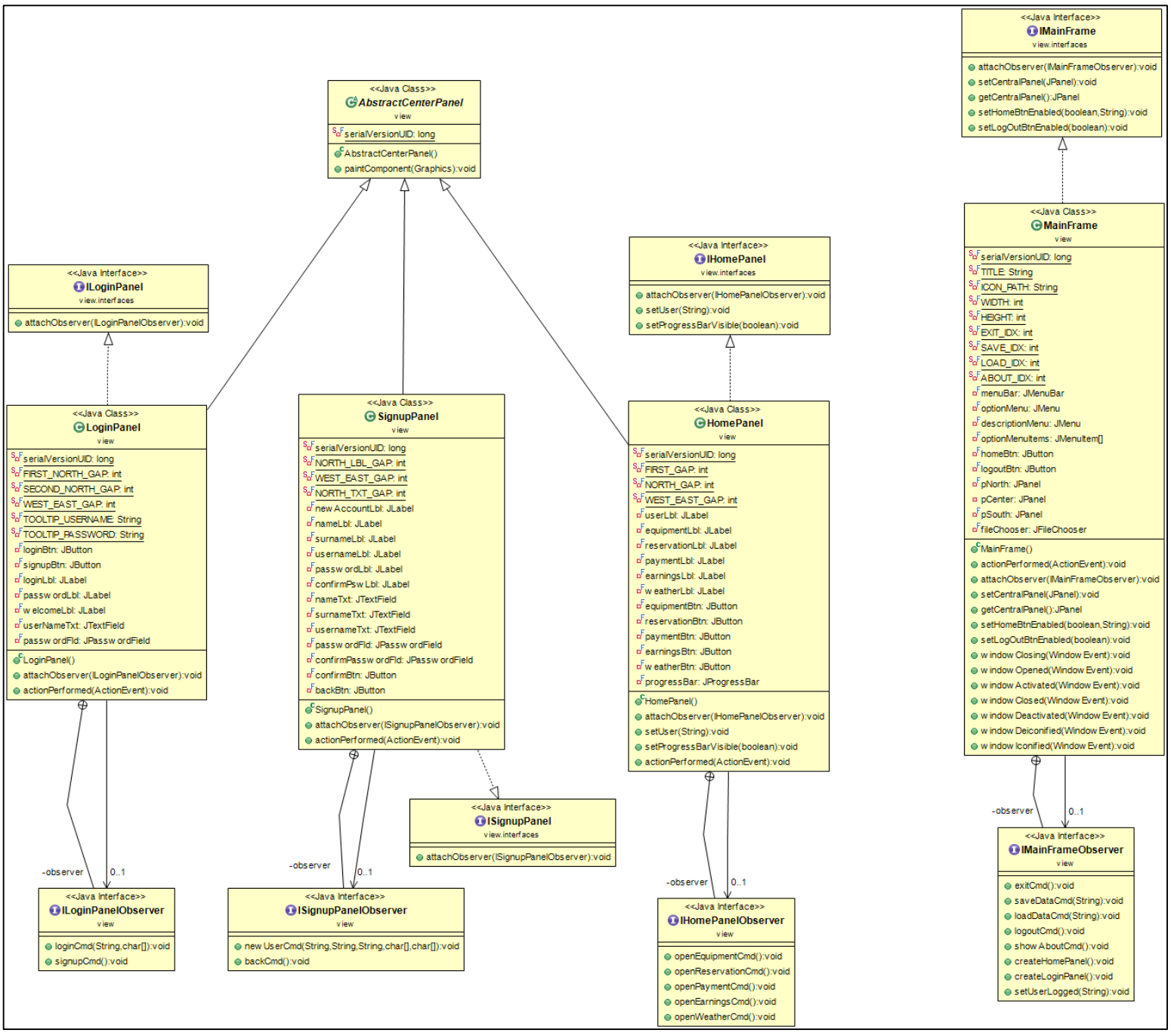
- **IModel e Model**: rappresentano rispettivamente l'interfaccia e l'implementazione del modello, ovvero il "contenitore" principale di tutti i dati necessari per l'applicazione. Model implementa anche l'interfaccia **Serializable** per garantire la persistenza dei dati.

- **IUser e User:** interfaccia e relativa implementazione che modella un generico utente, in grado di effettuare il login all'applicazione e utilizzare le funzionalità offerte dall'applicazione. Per tenere traccia degli incassi, si è deciso di utilizzare una mappa **TreeMap<Calendar, Integer>** in cui ogni entry corrisponde a **<data di incasso, incassi totali>**. Per far sì che il sistema sia in grado di garantire la persistenza dei dati, User implementa anche l'interfaccia **Serializable**.

- **IReservation e Reservation:** interfaccia e relativa implementazione che modella una generica prenotazione. Per memorizzare le prenotazioni si è scelta una semplice lista **List<Reservation>** e inoltre si è effettuato l'**Override** dei metodi **equals(Object obj)** e **hashCode()** per fare in modo di non avere più di una prenotazione con lo stesso cognome all'interno di una stessa gestione. Per far sì che il sistema sia in grado di garantire la persistenza dei dati, Reservation implementa anche l'interfaccia **Serializable**.

- **IBeach e Beach:** interfaccia e relativa implementazione che modella una generica spiaggia, appartenente ad un **User** o prenotata in una **Reservation**. Per far sì che il sistema sia in grado di garantire la persistenza dei dati, Beach implementa anche l'interfaccia **Serializable**.

Diagramma UML delle principali classi relative alla parte di vista dell'applicazione. In particolare il diagramma si concentra sulle viste relative al pannello di login, di registrazione e a quella del menu principale visualizzato una volta effettuato il login.

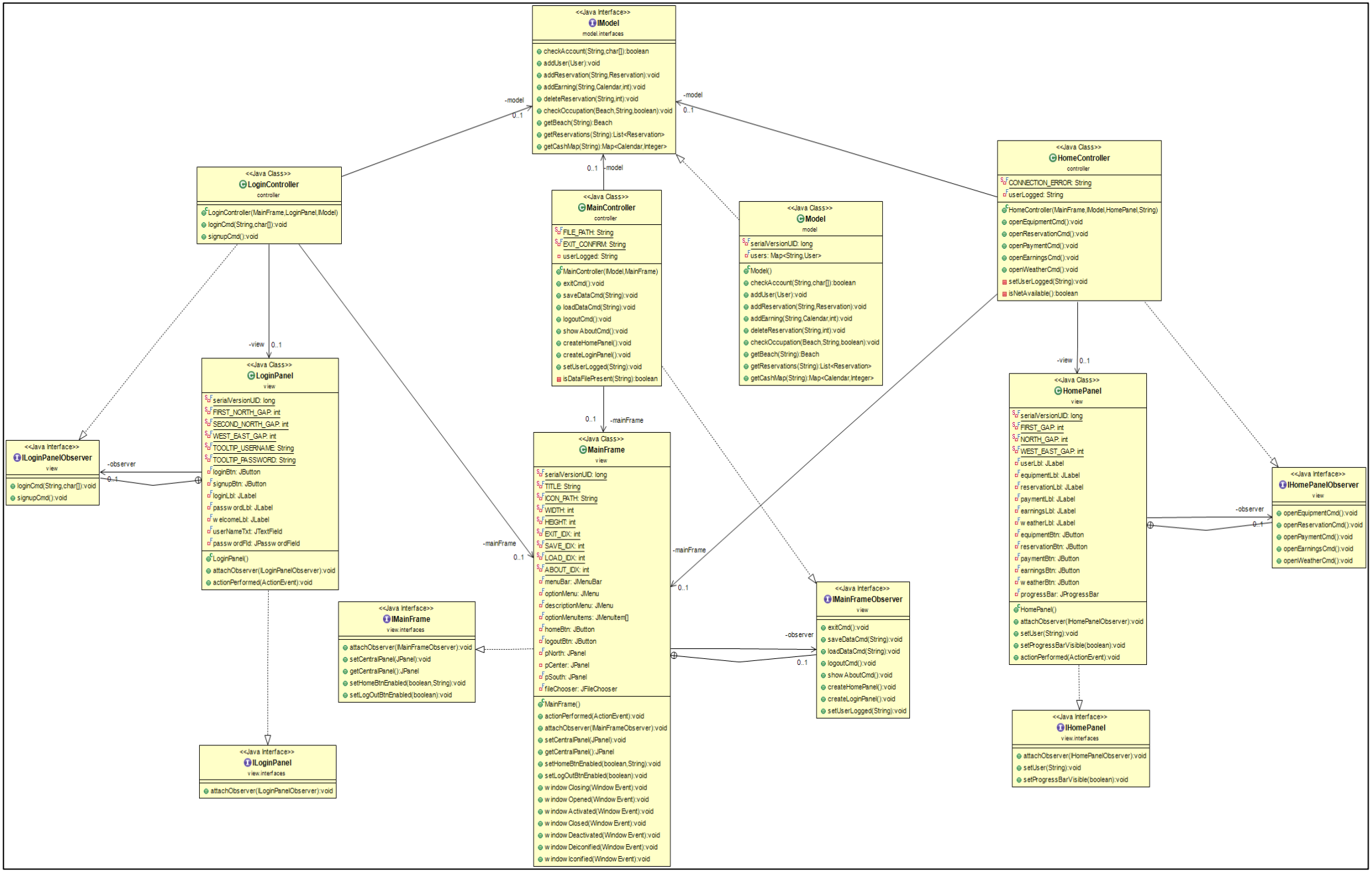


Descrizione degli aspetti principali:

- Dato che l'applicazione si compone di molteplici viste, si è deciso di implementare una view principale (l'interfaccia **IMainFrame**, implementata dalla classe **MainFrame**) in grado di:
  1. Presentare due piccoli pannelli, uno nella parte superiore e uno nella parte inferiore del frame. Il primo avrà il compito di tornare nella **HomePanel** a prescindere dalla vista in cui l'utente sta lavorando (tranne nel caso in cui mi trovi nella vista di login o di registrazione), mentre il secondo effettuerà il logout e tornerà quindi nella vista iniziale di login (**LoginPanel**).
  2. Supportare il load/unload della parte centrale della vista a seconda delle operazioni scelte dall'utente (funzionalità implementata dal metodo **setCentralPanel** di **IMainFrame**).
  3. Sfruttare una menu bar per alcune funzionalità, come il caricamento e il salvataggio dei dati navigando nel file system, o mostrare una vista di about.

- Sono state implementate tutta una serie di view (sempre tramite l'accoppiata interfaccia e relativa classe concreta) pensate per poter essere sostituite nel corpo centrale della **IMainFrame.** Esempi concreti mostrati dal diagramma sono: **ILoginPanel**, **IHomePanel** e **ISignupPanel.** Tutti i panelli che vengono posizionati al centro di **IMainFrame** estendono dalla classe astratta **AbstractCentraPanel**, la quale effettua **l'override** del metodo **paintComponent** della classe swing **JComponent**.
  L'override di questo metodo permette di dare uno sfondo personalizzato ai pannelli che estendono da **AbstractCenterPanel**.

Diagramma UML relativo all’interazione tra la parte di view, model e controller dell’applicazione, in particolare per la gestione del login di un generico utente nel sistema.



Descrizione degli aspetti principali:

- Il diagramma UML rappresenta l'interazione tra le diverse classi dovuta all'utilizzo del pattern MVC.

- Ogni classe della view presenta un'interfaccia observer che racchiude le intestazioni dei metodi necessari per la gestione degli eventi provenienti dalle view, e modificare di conseguenza le view stesse e i dati presenti nel modello.

- Questa interfaccia viene ogni volta implementata da un controller apposito, che definisce il comportamento di ogni metodo definito nell'interfaccia e che quindi si occupa effettivamente di gestire i cambiamenti sia sulla view che nel model, aggiornandoli tramite l'interazione dell'utente. In questo modo, l'unica parte ad avere un riferimento al model nella sua interezza è il controller.

- È presente un **MainController** il cui compito è gestire gli eventi provenienti dai componenti del **MainFrame** (comuni a tutte le viste): salvataggio/caricamento del modello su/da file a seguito dei click sui vari menu presenti nelle menu bar, mostrare la vista di about.

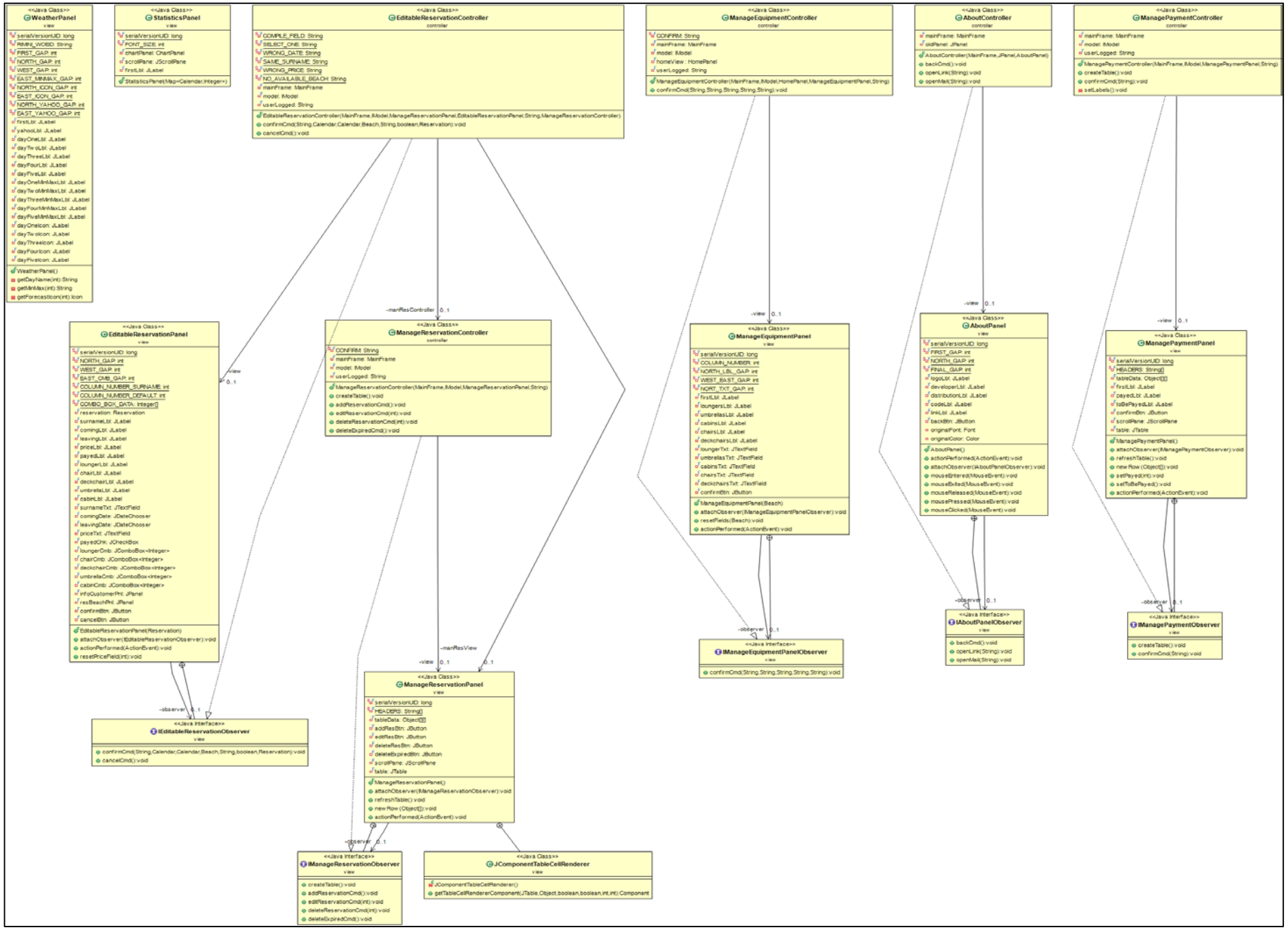
3 Organizzazione dei Package

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

- **controller**: contiene i sorgenti che incapsulano il comportamento dei controller relativi al MainFrame e agli altri pannelli.
    - controller.AboutController: controller associato alla schermata di about dell'applicazione
    - controller.EditableReservationController: controller associato alla schermata in cui si crea una nuova prenotazione o se ne modifica una esistente.
    - controller.HomeController: controller associato alla schermata home che si presenta subito dopo aver effettuato il login.
    - controller.LoginController: controller associato alla schermata di login.
    - controller.MainController: controller associato al frame dell'applicazione.
    - controller.ManageEquipmentController: controller associato alla schermata in cui l'utente modifica il quantitativo della proprio attrezzatura.
    - controller.ManagePaymentController: controller associato alla schermata in cui l'utente controlla i pagamenti da ricevere e li setta come pagati.
    - controller.ManageReservationController: controller associato alla schermata in cui l'utente controlla tutte le prenotazioni presenti.
    - controller.SignupController: controller associato alla schermata in cui un nuovo utente può registrarsi.
- **exceptions**: contiene l'insieme delle classi relative ad alcune particolari eccezioni che possono verificarsi nell'applicazione e che vengono opportunamente gestite.
    - exceptions.AlreadyExistsException: generata quando si tenta di registrare un nuovo utente con lo stesso username di uno già presente, o quando si tenta di inserire una nuova prenotazione con un cognome già presente.
    - exceptions.OccupationException: generata quando si tenta di prenotare un numero indisponibile di attrezzattura, o quando si tenta di modificare l'attrezzatura con un numero inferiore di quello attualmente prenotato.
    - exceptions.UserNotFoundException: generata quando l'username con cui si sta cercando di loggarsi non esiste.
- **main**: contiene la sola classe Main del programma.
    - main.Main: contiene il metodo main, punto di ingresso del programma.

- **model**: contiene i sorgenti che implementano la parte di model dell'applicazione.
    - model.Beach: classe che incapsula l'entità spiaggia.
    - model.Model: classe che incapsula che il modello e le relative collezioni per memorizzare i dati necessari al funzionamento.
    - model.Reservation: classe che incapsula l'entità prenotazione.
    - model.User: classe che incapsula l'entità utente, colui che utilizza l'applicazione.
- **model.interfaces**: contiene le interfacce che vengono implementate dalle classi concrete presenti nel package model precedentemente descritto. Ad ogni classe corrisponde la relativa interfaccia.
- **view**: contiene tutte le viste dell'applicazione.
    - view.AboutPanel: pannello che mostra i crediti dell'applicazione
    - view.AbstractCenterPanel: classe astratta da cui tutti i pannelli estendono, per far si chè abbiano uno sfondo diverso da quello di default.
    - view.EditableReservationPanel: pannello in cui si procede con la creazione o con la modifica di una prenotazione.
    - view.HomePanel: pannello che viene mostrato quando è stato effettuato con successo il login. Mostra le funzionalità accessibili all'utente.
    - view.LoginPanel: il primo pannello mostrato all'avvio dell'applicativo, in cui è possibile effettuare il login o decidere di registrare un nuovo account.
    - view.MainFrame: la finestra dell'applicazione. Tutte gli altri pannelli che formano il package view vengono posti al centro di questo frame quando necessario.
    - view.ManageEquipmentPanel: pannello in cui l'utente modifica il numero della propria attrezzatura.
    - view.ManagePaymentPanel: pannello in cui vengono mostrate le prenotazioni ancora non saldate, il denaro ricevuto e quello ancora da ricevere.
    - view.ManageReservationPanel: pannello in cui vengono mostrate tutte le prenotazioni. Da qui è possibile procedere con la creazione di una nuova prenotazione, modificarne una esistente, o cancellarle.
    - view.SignupPanel: pannello in cui viene effettuata la registrazione di un nuovo utente.
    - view.StatisticsPanel: pannello in cui viene mostrato un grafico a barre con l'andamento temporale degli incassi dell'utente.
    - view.WeatherPanel: pannello in cui vengono mostrate le previsioni metereologiche dei prossimi cinque giorni. Nello specifico vengono mostrate le temperature minime/massime e un'icona che rappresenta la situazione meteo prevista (tutti i dati vengono presi direttamente da YahooWeather tramite un'apposita libreria).
- **view.interfaces**: contiene le interfacce che vengono implementate dalle classi concrete presenti nel package view precedentemente descritto.

Diagramma UML delle altre classi di view e controller non mostrate precedentemente, in cui si nota ancora una volta l'utilizzo del pattern MVC e observer.



4 Librerie esterne utilizzate

Vengono ora descritte le librerie di terze parti utilizzate nella realizzazione del progetto:

- jcalendar-1.4.jar: utilizzata per una migliore organizzazione e visualizzazione della data al momento della creazione/modifica di una prenotazione in EditableReservationPanel.
- jfreechart-1.0.19.jar: utilizzata per la creazione del grafico presente in StatisticsPanel.
- yahoo-weather-java-api-1.2.0.jar: utilizzata per ottenere i dati del meteo che vengono poi interpretati e visualizzati da WeatherPanel.
- Tutte le altre librerie presenti sono dipendenze delle librerie sopra descritte.

5 Testing

Il testing dell'applicazione è stato effettuato passo passo durante la scrittura del codice. Ogni funzionalità dopo essere stata implementata veniva testata nell'immediato, e poi si procedeva col debug fornito da eclipse per trovare eventuali errori. Una volta terminata la corretta implementazione di tutte le funzionalità (avvenuta in ambiente Windows 8.1), il programma è stato testato anche in ambiente Linux (più precisamente Ubuntu 14.04 LTS), evidenziando solo qualche problema dal punto di vista dei layout utilizzati per la collocazione dei componenti grafici, che sono poi stati prontamente corretti per permettere una visualizzazione ottimale anche su questo sistema operativo. Si è infine sistemato il codice grazie all'ausilio dei plugin consigliati PMD, Checkstyle e Findbugs.

6 Note finali

Il processo di sviluppo dell'applicazione ha seguito uno sviluppo del tutto lineare, la fase di progettazione iniziale ha permesso di procedere con l'implementazione di tutte le funzionalità originariamente pensate senza particolari intoppi. Il risultato finale è in linea con le mie personali aspettative e probabilmente fornirò l'applicativo al gestore dello stabilimento balneare presso cui ho lavorato per due anni per avere feedback più dettagliati e per migliorarlo in futuro con, ad esempio, la visualizzazione del layout degli ombrelloni proposta dal Prof. Viroli al momento della proposta di progetto; funzione purtroppo non implementata per mancanza di tempo residuo.