UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Report sul testing di un'applicazione Java

Antonio Bevilacqua - M63/162
Pasquale Boemio - M63/200
estate 2012

1 NOTE PRELIMINARI SUL PROGETTO

Dovendo testare un'applicazione in Java, nella fattispecie l'applicazione *Rubrica*, la primissima operazione effettuata è stata una grezza analisi del codice, il quale si è rivelato disordinato e in alcuni tratti apparentemente privo di formattazione. Inoltre quasi mai è rispettata la *camel notation*, fondamentale in Java per evitare confusione tra classi, metodi e istanze.

Per una più completa comprensione della logica di sviluppo adottata dal programmatore, gli autori hanno provveduto dunque a ripulire il codice, utilizzando come stile di indentazione il K&R *Unix style* comprensivo di alcune varianti. Altri aspetti negativi riscontrati nel listato, come ad esempio i nomi poco intuitivi assegnati alla maggior parte delle variabili o la totale assenza di commenti, non sono stati valutati in questa sede, sebbene pratiche simili siano fortemente scoraggiate durante la scrittura di codice.

Prima di procedere con le operazioni di testing, le entità coinvolte nel progetto sono state inoltre riorganizzate nel seguente modo:

- ✓ la classe `panelInserimento` è stata rimossa dal file di classe `Rubrica.java`, inserita in un file separato e rinominata `PanelInserimento`;
- ✓ la classe `framePri` è stata rimossa dal file di classe `Rubrica.java`, inserita in un file separato e rinominata `FramePri`.

Inoltre, predati dalla propria buona coscienza di programmatori, gli autori hanno provveduto ad eliminare altre *sporcature* che non sono state risparmiate nemmeno in fase preliminare.

- ✓ Gli oggetti `JFrame` e `JPanel` sono di tipo serializzabile. Pertanto, al fine di eliminare tutti i warning generati in fase di compilazione, tra le variabili membro delle classi

`PanelInserimento` e `FramePri` è stato aggiunto un campo denominato `serialVersionUID`, rappresentante un id autogenerato da assegnare alle istanze delle classi stesse.

- ✓ Tutte le variabili membro inutilizzate sono state eliminate senza pietà. In particolare, nella classe `PanelInserimento` il trattamento marziale è stato riservato alle variabili `private boolean extist`, `private int con`, `private int x`, `private String nameRub`.

- ✓ Alla variabile membro `private JLabel Congome` è stata sostituita, in uno slancio di patriottismo grammaticale, la variabile `private JLabel Cognome`.

1.1 FARE E NON FARE, ESISTE PROVARE!

Le operazioni di testing potrebbero essere definite ,in un certo senso, *esplorative*: non si ha idea di quello che si sta cercando, nemmeno si è sicuri che quel qualcosa esista davvero. Inoltre, nel caso di specie, l'applicazione è stata scritta in una sede diversa rispetto a quella in cui viene testata, e con essa non sono stati forniti requisiti di alcun tipo. Come sapere allora quando l'applicazione funziona *correttamente*, se non si conosce il funzionamento corretto che da essa ci si attende?

L'analisi dell'applicazione in esame ha attraversato tre *macrofasi* distinte e consequenziali.

1. In primo luogo si è proceduto alla valutazione statica della GUI, costruendo un automa che potesse modellare il comportamento del software in esame sintetizzandolo in termini di sequenze. Tale automa è stato utilizzato altresì per formulare una prima ipotesi sulle possibili funzionalità messe a disposizione dall'applicazione.

2. In base ai risultati ottenuti tramite l'FSM, si è stata scritta una *test suite* che fosse in grado di ripercorrere le sequenze definite in precedenza, al fine di verificare che il comportamento atteso dell'applicazione fosse coerente con il comportamento reale.

3. In ultima istanza è stato ispezionato staticamente il codice, con particolare enfasi sulle istruzioni che hanno dato luogo a casi di test positivi. Tecniche quali *Code reading*, *Code inspection* e *Debbugging* sono state applicate per scovare e infine correggere i bachi che affliggono la nostra applicazione.

Nei successivi paragrafi relativi alla costruzione delle sequenze di azioni poste sotto testing è stata adottata la seguente simbologia:

- ✓ Il simbolo $\mapsto$ indica l'avvio di una sequenza mediante esecuzione dell'applicazione.

- ✓ La dicitura `click(NomePulsante)` indica la pressione del pulsante `NomePulsante`.

- ✓ La dicitura [esegui azione] indica l'esecuzione di un'azione non atomica da parte dell'utente, come l'inserimento di alcuni valori testuali richiesti.

2 ESPLORAZIONE E SOLLECITAZIONE DELLA GUI

Come anticipato nel capitolo precedente, il primo passo per la ricerca di comportamenti anormali dell'applicazione è la comprensione di quali dovrebbero essere i suoi comportamenti normali. A tal fine, ci viene in aiuto il modello di automa a stati finiti.

L'automa a stati finiti, mostrato in figura 2.1 nella pagina seguente, racchiude nei nodi tutte le schermate che si presentano durante l'utilizzo del software, e negli archi le azioni necessarie a transitare da una schermata all'altra. In particolare, le schermate esaustivamente esplorate sono:

Pannello iniziale il quale viene visualizzato non appena si esegue l'applicazione;

Pannello creazione rubrica nel quale si inserisce il nome con cui si vuole creare una rubrica;

Pannello immissione info contatti che viene visualizzato dopo la creazione di una rubrica, utilizzato per immettere le informazioni di uno o più contatti;

Pannello inserimento nome rubrica nel quale si inserisce il nome di una rubrica della quale si vogliono visualizzare i contatti;

Pannello scorrimento contatti tramite il quale si visualizzano i contatti presenti in una rubrica;

Pannello ricerca con cui si esegue la ricerca di un certo contatto in una certa rubrica;

Pannello informazioni contatto nel quale si visualizzano le informazioni di uno specifico contatto.

2.1 CREAZIONE RUBRICA E INSERIMENTO CONTATTI

L'insieme di operazioni finalizzate alla creazione di una nuova rubrica e all'inserimento in essa di uno o più contatti riguarda essenzialmente due pannelli: *Pannello creazione rubrica* e *Pannello immissione info contatti*.

La sequenza testata in questo esempio comprende la creazione di una rubrica tramite l'inserimento del nome desiderato e il successivo inserimento di un contatto, per il quale vengono specificati tutti i campi richiesti dal form.

$\mapsto$ S_1 : `click(Crea rubrica)` $\rightarrow$ [inserisci nome rubrica] $\rightarrow$ `click(Invio)` $\rightarrow$ [compila form] $\rightarrow$ `click(add)` $\rightarrow$ `click(Save and back)`

L'applicazione risponde ad S_1 con la creazione di un file, il cui nome è quello precedentemente inserito per la rubrica, al cui interno si possono trovare 4 righe, ognuna contenente uno dei campi di contatto che sono stati richiesti nel form. Stesso risultato si ottiene creando una rubrica e inserendo più di un contatto: in tal caso, il file di rubrica conterrà le informazioni relative a tutti i contatti inseriti.

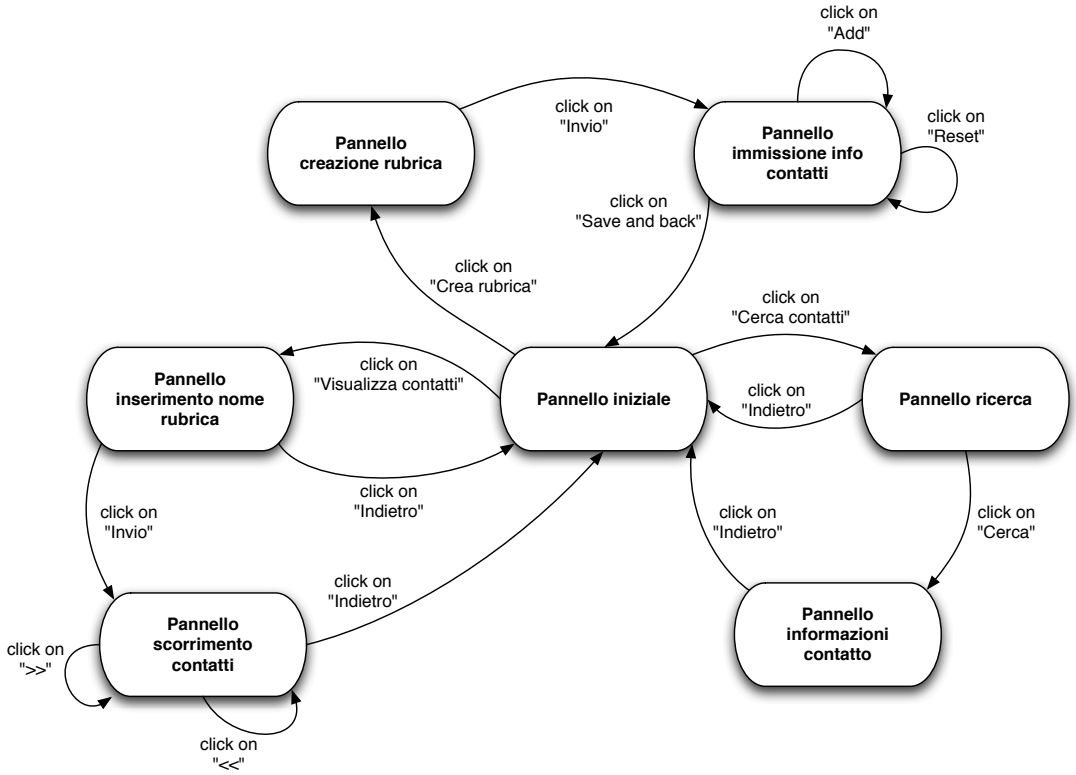

Figura 2.1: Diagramma a stati dell'interfaccia grafica

La sequenza appena studiata, con grande probabilità, ha lo scopo di generare una rubrica e inserirvi dei contatti. Questo set di operazioni è stato dunque trattato come una prima funzionalità messa a disposizione degli utenti.

Una più profonda valutazione della sequenza S_1 è stata portata avanti nei modi seguenti:

- $S_{1,NoName}$: `click(Crea rubrica)` → `click(Invio)` → [compila form] → `click(add)` → `click(Save and back)`
- $S_{1,NoCon}$: `click(Crea rubrica)` → [inserisci nome rubrica] → `click(Invio)` → `click(add)` → `click(Save and back)`
- $S_{1,NoAll}$: `click(Crea rubrica)` → `click(Invio)` → `click(add)` → `click(Save and back)`

In $S_{1,NoName}$ è stata creata una rubrica senza specificarne il nome. In questo caso, il programma risponde con un laconico

```
Errore in lettura File...
```

stampato sull'output di sistema, ma procede nell'avanzamento dei pannelli. Nessun file viene prodotto, come ci si aspettava, e l'aggiunta di un contatto tramite il tasto `add` genera anch'essa un messaggio di sistema:

`File inesistente...`

Il tasto `Save and back` ci riporta al pannello iniziale senza obiezioni.

La sequenza $S_{1,NoCon}$ risponde correttamente alla creazione di una rubrica (in questo caso ne specifichiamo il nome) con la consueta creazione del file. L'aggiunta di un contatto privo di informazioni non sembra disturbare l'applicazione, la quale si limita ad inserire all'interno del file delle righe vuote corrispondenti ai campi che sono stati lasciati non specificati. Ci si chiede a questo punto: nessun set minimo di informazioni è stato specificato per l'aggiunta di un contatto?

La sequenza $S_{1,NoAll}$ raccoglie gli aspetti inutili delle altre due sequenza, restituendo i messaggi di errore e non producendo alcun file.

Una veloce ispezione del tasto `Reset` mediante la sequenza S_{Reset} e innumerevoli sue derivate confermano che l'utente ha la possibilità di cancellare tutte le voci inserite nei box di testo prima di effettuare un inserimento:

↦ S_{Reset} : `click(Crea rubrica)` → [inserisci nome rubrica] → `click(Invio)` → [compila form] → `click(add)` → `click(Reset)` → [compila form] → `click(add)` → `click(Save and back)`

Un ultimo appunto riguarda la mancanza di un tasto `Indietro` in *Pannello creazione rubrica*. Non si è capito se tale mancanza sia intenzionale o frutto della distrazione, tuttavia si ritiene una crudeltà obbligare un utente a creare una rubrica qualora questi avesse cambiato idea.

In definitiva, i malfunzionamenti da imputare alla creazione della rubrica e all'aggiunta di contatti, qualora le funzionalità fossero state comprese correttamente, possono essere riassunti come segue:

- ✓ nessun errore viene segnalato al momento della creazione di una rubrica senza nome;
- ✓ nessun set minimo di informazioni è stato specificato per l'inserimento delle informazioni di contatto;
- ✓ non è presente un tasto per tornare da *Pannello creazione rubrica* al pannello iniziale.

Probabile che qualcuno al reparto sviluppo passerà un brutto quarto d'ora per questo.

2.2 VISUALIZZAZIONE CONTATTI

Il pulsante `Visualizza contatti` presente sul pannello iniziale dell'applicazione dovrebbe servire, se il nostro sesto senso non ci inganna, alla visualizzazione dei contatti. Premendolo, viene attivato *Pannello inserimento nome rubrica*. Le sequenze estrapolate per la valutazione di questa funzionalità coinvolgono dunque quanto visto nel paragrafo 2.1 a pagina 3 circa la creazione di una rubrica.

La prima sequenza da valutare è senza dubbio quella che si ritiene essere di utilizzo più frequente: un utente visualizza i contatti di una rubrica creata in precedenza.

↦ $S_2 : S_1$ → `click(Visualizza contatti)` → [inserisci nome rubrica] → `click(Invio)`

S_2 attiva *Pannello scorrimento contatti*, il quale si ritiene possa essere un pannello di navigazione. Alla sua prima attivazione, tale pannello mostra le informazioni relative al primo contatto inserito durante S_1.

Qualora fossero stati inseriti diversi contatti (più di uno), sequenze del tipo

$\mapsto$ $S_{2,NavNext} : S_2 \rightarrow$ `click(>>)`

$\mapsto$ $S_{2,NavPrev} : S_{2,NavNext} \rightarrow$ `click(<<)`

consentiranno all'utente di *navigare* attraverso i contatti precedentemente inseriti nella rubrica. Le informazioni vengono sempre visualizzate all'interno di *Pannello scorrimento contatti*, sotto forma di campi testuali che variano alla pressione dei tasti di navigazione.

La pressione del tasto `Indietro` riporta l'utente al pannello iniziale.

Ci troviamo probabilmente di fronte ad un'ennesima funzionalità resa disponibile dal nostro disastrato programma. Una volta creata una rubrica, e dopo avervi inserito dei contatti, la si può scorrere in modo sequenziale visualizzando tutte le informazioni in essa contenute.

In qualità di esperti cacciatori di malfunzionamenti, gli autori hanno notato che i pulsanti di navigazione in *Pannello scorrimento contatti* dovrebbero essere abilitati solo quando il numero di contatti lo consente. Ad esempio, a fronte della sequenza S_2 il tasto di navigazione >> risulta attivo, mentre il tasto << è inattivo. Scorrendo i contatti in avanti (si ipotizza che se ne siano inseriti diversi), il tasto << diventa cliccabile, ma non perde tale proprietà una volta che l'utente sia ritornato al primo contatto della rubrica. Ad eccezione di questo probabile baco, la visualizzazione dei contatti sembra funzionare correttamente.

Tra le tante sequenze testate, si riportano qui quelle considerate critiche. Per quanto riguarda la creazione di una rubrica innominata, la sequenza

$\mapsto$ $S_{2,NoName}$: $S_{1,NoName} \rightarrow$ `click(Visualizza contatti)` $\rightarrow$ [inserisci nome rubrica] $\rightarrow$ `click(Invio)`

non procede nell'apertura del pannello per la visualizzazione dei contatti, addirittura non sarebbe nemmeno realizzabile: quale nome inseriamo al momento della selezione della rubrica da visualizzare? Anche lasciando il box di testo bianco, premendo sul tasto `Invio` otteniamo la comparsa sul pannello del seguente messaggio:

```
Nome File non corretto.Riprova!
```

Tutto chiaro, se si tralascia la spaziatura.

Ultima sequenza proposta riguarda la visualizzazione di contatti fantasma, i cui campi non sono stati specificati:

$\mapsto$ $S_{2,NoCon} : S_{1,NoCon} \rightarrow$ `click(Visualizza contatti)` $\rightarrow$ [inserisci nome rubrica] $\rightarrow$ `click(Invio)`

In questo caso, la funzione di visualizzazione si comporta come atteso, ovvero offre all'utente la visualizzazione di contati i cui campi sono tutti vuoti. Tuttavia, questo probabile baco non risiede in questa sequenza ma in $S_{1,NoCon}$, come precedentemente indicato.

Riassumendo, l'unico baco che potrebbe essere presente nella funzionalità di visualizzazione dei contatti riguarda i tasti di navigazione, i quali non si comportano come il senso comune proporrebbe.

2.3 RICERCA CONTATTI

Tra i pulsanti presenti sul pannello iniziale ne troviamo uno chiamato `Cerca Contatti`. In nome del sospetto sopra ogni cosa, decidiamo di non fidarci della sua denominazione e procediamo con l'analisi esplorativa delle sue funzionalità.

Effettivamente, la pressione del tasto `Cerca Contatti` attiva *Pannello ricerca*, nel quale viene chiesto all'utente di inserire il nome di una rubrica e un campo a scelta tra nome, cognome, numero e indirizzo. Viene da pensare, dunque, che tale funzionalità effettui la ricerca all'interno di una specifica rubrica secondo un campo richiesto dall'utente. Non resta che costruire qualche sequenza per dissipare i nostri dubbi.

In prima istanza si verificano le condizioni di funzionamento supposto normale: l'utente crea una rubrica, inserisce alcuni contatti (ipotizzati distinti) e ne effettua la ricerca.

$\mapsto$ S_3 : S_1 $\rightarrow$ `click(Cerca Contatti)` $\rightarrow$ [inserisci nome rubrica e un campo di ricerca presente] $\rightarrow$ `click(Cerca)`

Inaspettatamente, dopo aver premuto il tasto `Cerca` viene attivato *Pannello informazioni contatto* il quale ci informa che l'applicazione non ha trovato alcuna corrispondenza, mediante il seguente messaggio:

```
Nessun contatto trovato!!
```

La visualizzazione della rubrica mediante la sequenza S_2 ci conferma comunque che il contatto è stato correttamente creato e salvato all'interno del file corrispondente. Questo strano comportamento sembrerebbe suggerire la presenza di un baco nella funzionalità di ricerca dei contatti. Approfondiamo maggiormente l'analisi del malfunzionamento applicando altre sequenze.

$\mapsto$ $S_{3,NoMatch}$: S_1 $\rightarrow$ `click(Cerca Contatti)` $\rightarrow$ [inserisci nome rubrica e un campo di ricerca non presente] $\rightarrow$ `click(Cerca)`

La sequenza $S_{3,NoMatch}$ effettua una ricerca tramite un campo sicuramente non presente all'interno della rubrica. Come risultato, tuttavia, otteniamo *Pannello informazioni contatto*, il quale ci mostra le informazioni del primo contatto inserito in rubrica.

Potremmo allora avanzare una prima ipotesi circa il funzionamento (o il malfunzionamento) della funzione di ricerca: questa, infatti, sembra funzionare *al contrario*. La ricerca di un contatto esistente non produce alcun risultato, mentre la ricerca di un contatto inesistente produce come risultato la prima voce presente in rubrica. A meno che l'intento del programmatore non fosse quello di implementare la funzione inversa della ricerca, si direbbe che questo sia un baco di proporzioni notevoli.

Altre sequenze poste sotto stress non mostrano deviazioni dal comportamento dell'applicazione fino ad ora osservato. Ad esempio, la ricerca di contatti in una rubrica senza nome tramite la sequenza

$\mapsto$ $S_{3,NoName} : S_{1,NoName} \rightarrow$ `clic(Cerca Contatti)` $\rightarrow$ [inserisci un campo di ricerca qualsiasi] $\rightarrow$ `click(Cerca)`

viene interrotta dal consueto messaggio

```
Nome File non corretto.Riprova!
```

L'ultima sequenza proposta riguarda la ricerca di contatti fantasma.

$\mapsto$ $S_{3,NoCon} : S_{1,NoCon} \rightarrow$ `clic(Cerca Contatti)` $\rightarrow$ [inserisci nome rubrica e un campo di ricerca] $\rightarrow$ `click(Cerca)`

La sequenza $S_{3,NoCon}$ ci mostra le informazioni su un contatto fantasma, ovvero con i campi tutti vuoti, avendo noi cercato un campo non vuoto.

In ultima analisi, l'unico baco presente nella funzione di ricerca sembra essere la stessa funzione di ricerca. I contatti presenti in rubrica non vengono inclusi nei risultati di una ricerca, mentre le ricerche di campi non presenti producono risultati non vuoti.

2.4 Tasto di uscita

L'ultimo tasto di cui è stato analizzato il comportamento è quello chiamato `Esci`. Gli autori si aspettano che questo tasto serva per uscire dall'applicazione, ma considerati i bachi trovati nei paragrafi precedenti questo non viene dato per scontato.

In effetti, la pressione del tasto `Esci` non produce alcun risultato, come mostrato in figura 2.2.

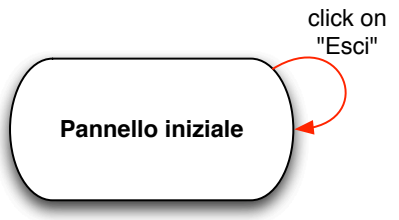

Figura 2.2: Baco sul tasto di uscita

Una possibilità è che quel pulsante sia stato creato nel pannello iniziale come riempitivo. Gli autori, tuttavia, sono stati più propensi a credere che la funzione di uscita di suddetto pulsante sia afflitto da un baco che ne altera il comportamento.

3 Analisi statica del codice

L'analisi fino ad ora effettuata ha avuto lo scopo di sintetizzare un set di possibili funzionalità messe a disposizione dall'applicazione. Tale sintesi è stata preliminarmente portata avanti con la sola osservazione dell'automa a stati finiti associato al software.

Per una comprensione più profonda della caratteristiche del programma in esame, se ne studia ora il codice, andando ad analizzare in dettaglio metodi, funzioni e strutture dati.

Tralasciando l'esoscheletro completamente errato su cui è stata costruita l'applicazione, si possono comunque identificare le strutture principali e i meccanismi funzionali complessivi cui gli autori sono interessati per eseguire procedure di testing.

Il codice si compone di tre classi:

- ✓ la classe `PanelInserimento`, la quale estende la classe `JPanel` e contiene tutte le variabili membro utilizzate durante l'elaborazione, nonchè i metodi necessari all'elaborazione stessa;
- ✓ la classe `FramePri`, la quale estende la classe `JFrame` e contiene, tra sue variabili membro, un oggetto di tipo `PanelInserimento`, che viene creato e posizionato al momento della chiamata al costruttore di `FramePri`;
- ✓ la classe `Rubrica`, la quale contiene un oggetto di tipo `FramePri` da creare al momento dell'esecuzione del programma (la classe `Rubrica` contiene il metodo `main`).

Salta subito all'occhio che la totalità dei metodi *sensibili* sono localizzati nella classe `PanelInserimento`. Di conseguenza, ci si aspetta in buona misura di localizzare al suo interno i bug classificati nel capitolo precedente.

La classe `PanelInserimento` contiene al suo interno variabili membro molto eterogenee, ma è facile osservare che tutti gli elementi dell'interfaccia grafica ne fanno parte. Inoltre, nella classe sono presenti i seguenti metodi:

- ✓ `public PanelInserimento()`, erculeo costruttore che alloca tutte le componenti di tipo `JLabel`, `JButton` e `JTextField`, vi associa un nome e le aggiunge al pannello in fase di creazione, abilitando solo quelle necessarie alla visualizzazione dell'indice (ovvero il pannello iniziale);
- ✓ `public void actionPerformed(ActionEvent evt)`, metodo che riceve in ingresso un evento (ad esempio, il click del mouse su di un tasto) e a seconda del nome dell'evento chiamante esegue un'azione piuttosto che un'altra.
- ✓ `void letturaFile(String nomeFile)`, metodo che apre in lettura un file il cui nome è uguale a `nomeFile`.
- ✓ `void cerca(String stringRicerca,String nomeFile)`, metodo che ricerca all'interno di un file il cui nome è uguale a `nomeFile` la stringa `stringRicerca`;
- ✓ `void stampaRisultati(int nume,String nomeFile)`, metodo che preleva da un file il cui nome è uguale a `nomeFile` delle stringhe prelevate a partire da una certa riga `nume`;
- ✓ `void stampa()`, metodo che imposta il valore di alcune variabili membro di tipo `JLabel` secondo le stringhe prelevate da un buffer, anch'esso variabile membro;
- ✓ `void reset()`, metodo che imposta il valore del testo di alcune variabili membro di tipo `JTextField` uguale ad una stringa nulla;

- ✓ `void pannelloCerca(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;
- ✓ `void pannelloRisultati(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;
- ✓ `void pannelloInput(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;
- ✓ `void pannelloOutput(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;
- ✓ `void pannelloRubIns(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;
- ✓ `void pannelloRubStp(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;
- ✓ `void pannelloIndex(boolean att)`, metodo che imposta la visibilità di alcune variabili membro al valore `att`;

Soffermiamoci un istante sul metodo `actionPerformed`. Esso ha una struttura che grosso modo può essere schematizzata come mostrato nel listato 1.

```
public void actionPerformed(ActionEvent evt) {
     ...
     String command = evt.getActionCommand();
     ...
     if(command.equals("<<")) {
          ...
     }
     if(command.equals(">>")) {
          ...
     }
     if(command.equals("indietro")) {
          ...
     }
     if(command.equals("add")) {
          ...
     }
     if(command.equals("invio")) {
          ...
     }
     if(command.equals("invio ")) {
          ...
     }
     if(command.equals("reset")) {
          ...
     }
     if(command.equals("save & back")) {
          ...
     }
     if(command.equals("cerca")) {
          ...
     }
```

```
        if(command.equals("esci")) {
                ...
        }
        if(command.equals("crea rubrica")) {
                ...
        }
        if(command.equals("visualizza contatti")) {
                ...
        }
        if(command.equals("cerca contatti")) {
                ...
        }
}
```

Listato 1: Il metodo di risposta agli eventi

Il compito di questo metodo è quello di rispondere agli eventi di click che avvengono sui pulsanti dell'interfaccia grafica. A seconda del `command` passato dall'evento corrente verrà eseguito un blocco di codice corrispondente al bottone premuto. Per sapere quindi cosa succede quando si preme un tasto dell'interfaccia grafica, e per scrivere i test case corrispondenti alle sequenze di nostro interesse, occorre esaminare il codice di questo metodo, valutare come vengono trattate le variabili membro e in che modo vengono richiamati gli altri metodi.

Nel capitolo successivo verranno individuati i metodi chiamati durante l'esecuzione delle sequenze corrispondenti alle funzionalità dell'applicazione, e per ogni sequenza verrà implementato un test case che possa coprire al meglio il codice sollecitato dalla sequenza stessa.

4 TESTING AUTOMATICO CON UISPEC4J

I metodi automatici per il testing dell'applicazione sono stati implementati tramite il framework UISpec4J, il quale estende JUnit e supporta la gestione delle interfacce grafiche. La classe `PanelInserimentoTest`, inclusa all'interno del pacchetto `is2.rubrica.test`, raccoglie i metodi di test qui riportati.

4.1 METODI DI VERIFICA DEI PANNELLI

Come asserito nel capitolo precedente, il frame principale dell'applicazione contiene al suo interno *tutti* i pulsanti, *tutti* i campi di testo e *tutte* le etichette che verranno utilizzati dall'utente nel corso dell'esecuzione del programma. A seconda del pannello in cui si trova l'utente, tuttavia, l'applicazione si preoccupa di mostrare solo gli elementi di interesse. Per questo motivo, sono stati implementati dei metodi di test che possano verificare la correttezza di tali impostazioni a fronte di ogni transizione (ovvero di ogni evento), dopo aver fatto corrispondere le variabili della classe agli elementi presenti sui pannelli dell'interfaccia grafica. In particolare:

- ✓ il metodo `private void pannelloIndexVisible(boolean att)` nel listato 2 nella pagina seguente verifica se il pannello iniziale sia visibile o meno;

- ✓ il metodo `private void pannelloCercaVisible(boolean att)` nel listato 3 verifica se *Pannello ricerca* sia visibile o meno;
- ✓ il metodo `private void pannelloRisultatiVisible(boolean att)` nel listato 4 verifica se *Pannello informazioni contatto* sia visibile o meno;
- ✓ il metodo `private void pannelloInputVisible(boolean att)` nel listato 5 nella pagina successiva verifica se *Pannello immissione info contatti* sia visibile o meno;
- ✓ il metodo `private void pannelloOutputVisible(boolean att)` nel listato 6 nella pagina seguente verifica se *Pannello scorrimento contatti* sia visibile o meno;
- ✓ il metodo `private void pannelloRubInsVisible(boolean att)` nel listato 7 nella pagina successiva verifica se *Pannello creazione rubrica* sia visibile o meno;
- ✓ il metodo `private void pannelloRubStpVisible(boolean att)` nel listato 8 nella pagina seguente verifica se *Pannello immissione info contatti* sia visibile o meno.

Questo insieme di metodi possono essere considerati come *di diagnostica*, a supporto quindi dei veri e propri metodi di test che verranno presentati nel paragrafo successivo.

```
private void pannelloIndexVisible(boolean att) {
     Window window = getMainWindow();

     assertEquals(att, window.getTextBox("index").isVisible());
     assertEquals(att, window.getButton("Crea rubrica").isVisible());
     assertEquals(att, window.getButton("Visualizza contatti").isVisible());
     assertEquals(att, window.getButton("Cerca Contatti").isVisible());
     assertEquals(att, window.getButton("esci").isVisible());
     assertEquals(att, window.getTextBox("                              ").
           isVisible());
}
```

Listato 2: Verifica del pannello principale

```
private void pannelloCercaVisible(boolean att) {
     Window window = getMainWindow();

     assertEquals(att, window.getTextBox("testoStp").isVisible());
     assertEquals(att, window.getTextBox("inNameRub").isVisible());
     assertEquals(att, window.getTextBox("infoCerca").isVisible());
     assertEquals(att, window.getTextBox("inCerca").isVisible());
     assertEquals(att, window.getButton("Cerca").isVisible());
}
```

Listato 3: Verifica del pannello di ricerca

```
private void pannelloRisultatiVisible(boolean att) {
     Window window = getMainWindow();

     assertEquals(att, window.getTextBox("risultati").isVisible());
}
```

Listato 4: Verifica del pannello risultati

```
private void pannelloInputVisible(boolean att) {
     Window window = getMainWindow();

     assertEquals(att, window.getTextBox("                Nome").isVisible());
     assertEquals(att, window.getTextBox("inNome").isVisible());
     assertEquals(att, window.getTextBox("                Cognome").isVisible());
     assertEquals(att, window.getTextBox("inCognome").isVisible());
     assertEquals(att, window.getTextBox("                Numero Casa").isVisible
           ());
     assertEquals(att, window.getTextBox("inNumero").isVisible());
     assertEquals(att, window.getTextBox("                Indirizzo").isVisible())
           ;
     assertEquals(att, window.getTextBox("inIndirizzo").isVisible());
     assertEquals(att, window.getButton("Add").isVisible());
     assertEquals(att, window.getButton("Reset").isVisible());
     assertEquals(att, window.getButton("Save & Back").isVisible());
}
```

Listato 5: Verifica del pannello immissione info contatti

```
private void pannelloOutputVisible(boolean att) {
     Window window = getMainWindow();

     assertEquals(att, window.getTextBox("nomeLabel").isVisible());
     assertEquals(att, window.getTextBox("cognomeLabe").isVisible());
     assertEquals(att, window.getTextBox("numeroLabel").isVisible());
     assertEquals(att, window.getTextBox("indirizzoLabel").isVisible());
     assertEquals(att, window.getButton(">>").isVisible());
     assertEquals(att, window.getButton("<<").isVisible());
}
```

Listato 6: Verifica del pannello scorrimento contatti

```
private void pannelloRubInsVisible(boolean att) {
     Window window = getMainWindow();

     assertFalse(window.getTextBox("testoStp").isVisible());
     assertEquals(att, window.getTextBox("testoIns").isVisible());
     assertEquals(att, window.getTextBox("inNameRub").isVisible());
     assertEquals(att, window.getButton("Invio").isVisible());
     assertFalse(window.getButton("Invio ").isVisible());
     assertEquals(att, window.getButton("Indietro").isVisible());
}
```

Listato 7: Verifica del pannello creazione rubrica

```
private void pannelloRubStpVisible(boolean att) {
     Window window = getMainWindow();

     assertFalse(window.getTextBox("testoIns").isVisible());
```

```
        assertEquals(att, window.getTextBox("testoStp").isVisible());
        assertEquals(att, window.getTextBox("inNameRub").isVisible());
        assertEquals(att, window.getButton("Invio ").isVisible());
        assertFalse(window.getButton("Invio").isVisible());
}
```

Listato 8: Verifica del pannello immissione contatti

4.2 TEST CASES E SEQUENZE

VISUALIZZAZIONE DEI PANNELLI

I primi metodi di testing servono a verificare che tutti gli elementi dell'interfaccia grafica si comportino come atteso durante la transizione dal pannello principale a un altro pannello, e viceversa.

Le sequenze di ogni metodo sono indicate nei commenti preliminari. In particolare:

- ✓ nel listato 9 viene testata la transizione da *Pannello iniziale* a *Pannello ricerca*, e ritorno;
- ✓ nel listato 10 nella pagina seguente viene testata la transizione da *Pannello iniziale* a *Pannello inserimento nome rubrica*, e ritorno;
- ✓ nel listato 11 nella pagina successiva viene testata la transizione da *Pannello iniziale* a *Pannello creazione rubrica*.

```
/**
 * Testing path
 * PANNELLO INIZIALE -> click(Cerca Contatti) -> PANNELLO RICERCA -> click(
       Indietro) -> PANNELLO INIZIALE
 */
public void testGoToPannelloCerca() {
      // parto da index
      pannelloIndexVisible(true);

      // vado in cerca contatti
      getMainWindow().getButton("Cerca Contatti").click();

      // ci sono davvero?
      pannelloIndexVisible(false);
      pannelloCercaVisible(true);

      // vado via
      getMainWindow().getButton("Indietro").click();

      // me ne sono andato?
      pannelloIndexVisible(true);
      pannelloCercaVisible(false);

}
```

Listato 9: Verifica dell'attivazione del pannello cerca

```
/**
 * Testing path
 * PANNELLO INIZIALE -> click(Visualizza contatti) -> PANNELLO INSERIMENTO NOME
       RUBRICA -> click(Indietro) -> PANNELLO INIZIALE
 */
public void testGoToVisualizzaContatti() {
     // parto da index
     pannelloIndexVisible(true);

     // vado in visualizza contatti
     getMainWindow().getButton("Visualizza contatti").click();

     // ci sono davvero?
     pannelloIndexVisible(false);
     pannelloRubStpVisible(true);

     // vado via
     getMainWindow().getButton("Indietro").click();

     // me ne sono andato?
     pannelloIndexVisible(true);
     pannelloRubStpVisible(false);

}
```

Listato 10: Verifica dell'attivazione del pannello visualizzazione contatti

```
/**
 * Testing path
 * PANNELLO INIZIALE -> click(Crea Rubrica) -> PANNELLO CREAZIONE RUBRICA -> click
       (Indietro) -> PANNELLO INIZIALE
 */
public void testGoToCreazioneRubrica(){
     // parto da index
     pannelloIndexVisible(true);

     // vado in crea rubrica
     getMainWindow().getButton("Crea rubrica").click();

     // ci sono davvero?
     pannelloRubInsVisible(true);
     pannelloIndexVisible(false);

     // vado via
     getMainWindow().getButton("Indietro").click();

     // me ne sono andato?
     pannelloRubInsVisible(false);
     pannelloIndexVisible(true);
}
```

Listato 11: Verifica dell'attivazione del pannello creazione rubrica

CREAZIONE RUBRICA E INSERIMENTO CONTATTI

I seguenti metodi di test verificano il comportamento del codice durante la creazione di una rubrica. Nella fattispecie:

- ✓ il metodo 12 si occupa di verificare il comportamento della sequenza $S_{1,NoName}$, ovvero la creazione di una rubrica priva di nome;
- ✓ il metodo 13 testa la sequenza S_1 per la creazione di una rubrica il cui nome inserito sia valido;
- ✓ il metodo 14 nella pagina seguente si occupa di verificare il corretto funzionamento dei meccanismi di inserimento dei contatti.
- ✓ il metodo 15 a pagina 18 crea due rubriche con il medesimo nome per valutare la presenza di un messaggio di errore.

```
/**
 * Testing if the user submit a void name into Creazione Rubrica
 */
public void testVoidNameCreazioneRubrica(){
     // parto da index
     pannelloIndexVisible(true);

     // vado in creazione rubrica
     getMainWindow().getButton("Crea rubrica").click();

     // setto un campo nullo a inNameRub
     getMainWindow().getTextBox("inNameRub").setText("");
     getMainWindow().getButton("Invio").click();

     // non mi sono mosso da creazione rubrica
     pannelloRubInsVisible(true);
     pannelloInputVisible(false);

     // controllo che l'errore venga stampato
     assertEquals("Nome File non corretto.Riprova                          ",
           getMainWindow().getTextBox("testoIns").getText());

     // vado via
     getMainWindow().getButton("Indietro").click();
}
```

Listato 12: Verifica sulla creazione di una rubrica senza nome

```
/**
      * Testing if the user submit a valid name into Creazione Rubrica
      */
     public void testValidNameCreazioneRubrica(){
          // parto da index
          pannelloIndexVisible(true);

          // vado in creazione rubrica
          getMainWindow().getButton("Crea rubrica").click();

          // setto inNameRub
          getMainWindow().getTextBox("inNameRub").setText(PanelInserimentoTest.
                thisTestRubName + "_no_items");
          getMainWindow().getButton("Invio").click();
```

```
            // controllo che abbia creato il file
            assertTrue(new File(PanelInserimentoTest.thisTestRubName + "_no_items").
                  exists());

            // vado in aggiungi contatto
            pannelloRubInsVisible(false);
            pannelloInputVisible(true);

            // torno indietro
            getMainWindow().getButton("Save & Back").click();

            // me ne sono andato?
            pannelloInputVisible(false);
            pannelloIndexVisible(true);

}
```

Listato 13: Verifica sulla creazione di una rubrica con come corretto

```
/**
 * Testing if the user inserts contacts (check the add and the reset operations)
 * @throws FileNotFoundException
 */
public void testAddContacts() throws Exception{
     // parto da index
     pannelloIndexVisible(true);

     // vado in creazione rubrica
     getMainWindow().getButton("Crea rubrica").click();

     // setto inNameRub
     getMainWindow().getTextBox("inNameRub").setText(PanelInserimentoTest.
           thisTestRubName);
     getMainWindow().getButton("Invio").click();

     String[] contatto = new String[] {"nome", "cognome", "numero", "indirizzo"};
     // inserisco un contatto
     getMainWindow().getTextBox("inNome").setText(contatto[0]);
     getMainWindow().getTextBox("inCognome").setText(contatto[1]);
     getMainWindow().getTextBox("inNumero").setText(contatto[2]);
     getMainWindow().getTextBox("inIndirizzo").setText(contatto[3]);

     getMainWindow().getButton("Add").click();

     // inserisco un altro contatto
     getMainWindow().getTextBox("inNome").setText(contatto[0]);
     getMainWindow().getTextBox("inCognome").setText(contatto[1]);
     getMainWindow().getTextBox("inNumero").setText(contatto[2]);
     getMainWindow().getTextBox("inIndirizzo").setText(contatto[3]);

     getMainWindow().getButton("Add").click();

     // controllo che siano stati inseriti i contatti correttamente
     BufferedReader fileReader = new BufferedReader(new FileReader(new File(
           PanelInserimentoTest.thisTestRubName)));
     String linea;
     int index=0;
     while((linea = fileReader.readLine()) != null)
          assertEquals(contatto[index++ % 4], linea);
```

```
        // inserisco di nuovo il contatto per testare il reset dei campi
        getMainWindow().getTextBox("inNome").setText(contatto[0]);
        getMainWindow().getTextBox("inCognome").setText(contatto[1]);
        getMainWindow().getTextBox("inNumero").setText(contatto[2]);
        getMainWindow().getTextBox("inIndirizzo").setText(contatto[3]);

        getMainWindow().getButton("Reset").click();

        // controllo che i campi siano stati resettati
        assertEquals("", getMainWindow().getTextBox("inNome").getText());
        assertEquals("", getMainWindow().getTextBox("inCognome").getText());
        assertEquals("", getMainWindow().getTextBox("inNumero").getText());
        assertEquals("", getMainWindow().getTextBox("inIndirizzo").getText());

        // torno indietro
        getMainWindow().getButton("Save & Back").click();
}
```

Listato 14: Verifica sulle funzioni di inserimento contatti

```
/**
 * Testing if the user submit a duplicate name into Creazione Rubrica
 */
public void testDuplicateNameCreazioneRubrica(){
        // parto da index
        pannelloIndexVisible(true);

        // vado in creazione rubrica
        getMainWindow().getButton("Crea rubrica").click();

        // controllo che gia‘ esista il file
        assertTrue(new File(PanelInserimentoTest.thisTestRubName).exists());

        // setto un campo nullo a inNameRub
        getMainWindow().getTextBox("inNameRub").setText(PanelInserimentoTest.
                thisTestRubName);
        getMainWindow().getButton("Invio").click();

        // non mi sono mosso da creazione rubrica
        pannelloRubInsVisible(true);
        pannelloInputVisible(false);

        // controllo che l'errore venga stampato
        assertEquals("File giÃ  esistente. Riprova                          ",
                getMainWindow().getTextBox("testoIns").getText());

        // vado via
        getMainWindow().getButton("Indietro").click();
}
```

Listato 15: Verifica sull'uso di un nome replicato per la rubrica

FUNZIONI DI RICERCA

Le funzioni di ricerca dei contatti sono state anch'esse testate mediante metodi automatici.

- ✓ il metodo 16 controlla il comportamento dell'applicazione per la sequenza $S_{2,NoName}$, ovvero nel caso in cui nel pannello di ricerca di contatti si lasci vuoto il campo del nome della rubrica in cui cercare;
- ✓ il metodo 17 effettua una visualizzazione inserendo un nome scorretto della rubrica in cui cercare;
- ✓ il metodo 18 nella pagina seguente visualizza dei contatti presenti in una rubrica e verifica che tutto proceda correttamente;
- ✓ il metodo 19 a pagina 21 si occupa di effettuare una ricerca di contatti lasciando vuoto il campo rubrica;
- ✓ il metodo 20 a pagina 21 effettua una ricerca di contatti inserendo un nome rubrica non valido;
- ✓ il metodo 21 a pagina 22 ricerca un contatto sicuramente non presente in rubrica;
- ✓ il metodo 22 a pagina 22 esegue una ricerca di un contatto sicuramente presente in rubrica.

```
/**
 * Testing if user insert a void Rub name in Visualizza contatti
 */
public void testVoidNameVisualizzaContatti() {
     // parto da index
     pannelloIndexVisible(true);

     // vado in visualizza contatti
     getMainWindow().getButton("Visualizza contatti").click();

     // setto un campo nullo a inNameRub
     getMainWindow().getTextBox("inNameRub").setText("");
     getMainWindow().getButton("Invio ").click();

     // non mi sono mosso da creazione rubrica
     pannelloRubStpVisible(true);
     pannelloOutputVisible(false);

     // controllo che l'errore venga stampato
     assertTrue(getMainWindow().getTextBox("Nome File non corretto.Riprova!
                                ").isVisible());

     // vado via
     getMainWindow().getButton("Indietro").click();

}
```

Listato 16: Verifica sulla ricerca senza il nome rubrica

```
/**
 * Testing if user insert an invalid Rub name in Visualizza contatti
 */
```

```
public void testInvalidNameVisualizzaContatti() {
      // parto da index
      pannelloIndexVisible(true);

      // vado in visualizza contatti
      getMainWindow().getButton("Visualizza contatti").click();

      // setto un campo nullo a inNameRub
      getMainWindow().getTextBox("inNameRub").setText("INVALID_INVALID_INVALID");
      getMainWindow().getButton("Invio ").click();

      // non mi sono mosso da creazione rubrica
      pannelloRubStpVisible(true);
      pannelloOutputVisible(false);

      // controllo che l'errore venga stampato
      assertTrue(getMainWindow().getTextBox("File non esistente. Riprova!
                                    ").isVisible());

      // vado via
      getMainWindow().getButton("Indietro").click();

}
```

Listato 17: Verifica sulla visualizzazione con nome rubrica errato

```
/**
 * Testing if the user lists the contacts
 */
public void testListContacts() {
      // parto da index
      pannelloIndexVisible(true);

      // vado in visualizza contatti
      getMainWindow().getButton("Visualizza contatti").click();

      // setto un campo nullo a inNameRub
      getMainWindow().getTextBox("inNameRub").setText(PanelInserimentoTest.
            thisTestRubName);
      getMainWindow().getButton("Invio ").click();

      // vado in output
      pannelloRubStpVisible(false);
      pannelloOutputVisible(true);

      // mi faccio un giretto tra i contatti :)
      assertFalse(getMainWindow().getButton("<<").isEnabled());
      assertTrue(getMainWindow().getButton(">>").isEnabled());

      getMainWindow().getButton(">>").click();

      assertTrue(getMainWindow().getButton("<<").isEnabled());
      assertFalse(getMainWindow().getButton(">>").isEnabled());

      getMainWindow().getButton("<<").click();

      assertFalse(getMainWindow().getButton("<<").isEnabled());
      assertTrue(getMainWindow().getButton(">>").isEnabled());

      // vado via
```

```
        getMainWindow().getButton("Indietro").click();
}
```

Listato 18: Verifica visualizzazione contatti

```
/**
 * Testing if user insert a void Rub Name into Cerca contatti
 */
public void testVoidNameCerca() {
        // parto da index
        pannelloIndexVisible(true);

        // vado in cerca contatti
        getMainWindow().getButton("Cerca Contatti").click();

        // setto un campo nullo a inNameRub
        getMainWindow().getTextBox("inNameRub").setText("");
        getMainWindow().getButton("Cerca").click();

        // non mi sono mosso da cerca contatti
        pannelloCercaVisible(true);
        pannelloRisultatiVisible(false);

        // controllo che l'errore venga stampato
        assertEquals("Nome File non corretto.Riprova!                          ",
                getMainWindow().getTextBox("testoStp").getText());

        // vado via
        getMainWindow().getButton("Indietro").click();
}
```

Listato 19: Verifica sul nome rubrica vuoto per la ricerca contatti

```
/**
 * Testing if user insert an invalid Rub Name into Cerca contatti
 */
public void testInvalidNameCerca() {
        // parto da index
        pannelloIndexVisible(true);

        // vado in cerca contatti
        getMainWindow().getButton("Cerca Contatti").click();

        // setto un campo invalido a inNameRub
        getMainWindow().getTextBox("inNameRub").setText("INVALID_INVALID_INVALID");
        getMainWindow().getButton("Cerca").click();

        // non mi sono mosso da cerca contatti
        pannelloCercaVisible(true);
        pannelloRisultatiVisible(false);

        // controllo che l'errore venga stampato
        assertEquals("File non esistente. Riprova!                          ",
                getMainWindow().getTextBox("testoStp").getText());

        // vado via
        getMainWindow().getButton("Indietro").click();
}
```

Listato 20: Verifica sul nome rubrica errato per la ricerca contatti

```
/**
 * Testing if user insert an invalid contact name into Cerca contatti
 */
public void testInvalidContactNameCerca() {
      // parto da index
      pannelloIndexVisible(true);

      // vado in cerca contatti
      getMainWindow().getButton("Cerca Contatti").click();

      // setto un campo invalido a inNameRub
      getMainWindow().getTextBox("inNameRub").setText(PanelInserimentoTest.
            thisTestRubName);
      getMainWindow().getTextBox("inCerca").setText("INVALID_INVALID_INVALID");
      getMainWindow().getButton("Cerca").click();

      // vado in risultati
      pannelloCercaVisible(false);
      pannelloRisultatiVisible(true);

      // controllo che l'errore venga stampato
      assertEquals("Nessun contatto trovato!!", getMainWindow().getTextBox("
            risultati").getText());

      // vado via
      getMainWindow().getButton("Indietro").click();

}
```

Listato 21: Verifica sul campo errato per la ricerca contatti

```
/**
 * Testing if user insert an valid contact name into Cerca contatti
 */
public void testValidContactNameCerca() {
      // parto da index
      pannelloIndexVisible(true);

      // vado in cerca contatti
      getMainWindow().getButton("Cerca Contatti").click();

      // setto un campo invalido a inNameRub
      getMainWindow().getTextBox("inNameRub").setText(PanelInserimentoTest.
            thisTestRubName);
      getMainWindow().getTextBox("inCerca").setText("nome");
      getMainWindow().getButton("Cerca").click();

      // vado in risultati
      pannelloCercaVisible(false);
      pannelloRisultatiVisible(true);

      // controllo che l'errore non venga stampato
      assertFalse(getMainWindow().getTextBox("risultati").getText().equals("Nessun
            contatto trovato!!"));
```

```
        // vado via
        getMainWindow().getButton("Indietro").click();
}
```

Listato 22: Verifica di una corretta ricerca contatti

FUNZIONE DI USCITA

La funzione di uscita è stata testata tramite un unico metodo il cui scopo è controllare appunto che il tasto `Esci` funzioni correttamente. L'analisi preliminare dell'automa ha comunque evidenziato che non vi è modo di uscire dall'applicazione, quindi il seguente metodo di test è tristemente destinato a fallire.

```
/**
 * Testing exit button
 * uispec4j doesn't have a way to test if a non modal window is not disposable.
 * So we tests that, after a click on "esci" button, we can't get a window object.
 * This test can be successful, but it can't fail.
 * However this kind of test seems works for our interests
 */
public void testExit() {
        // parto da index
        pannelloIndexVisible(true);

        getMainWindow().getButton("esci").click();
        assertFalse(getMainWindow().toString().startsWith("org.uispec4j.Window"));
}
```

Listato 23: Verifica sulla funzione di uscita

5 ISPEZIONE DEL CODICE E BUGFIXING

Osservando i metodi di test falliti e quelli andati a buon fine, appare chiaro che i bachi rilevati dall'analisi dell'automa a stati finiti erano effettivamente presenti all'interno del codice. Il passo successivo per la rimozione dei difetti trovati comporta dunque l'analisi del codice e la riscrittura delle righe possedute dal male.

Iniziando dal semplice, si osserva il funzionamento del metodo di uscita per verificare come mai non effettua l'uscita dal programma.

Il metodo costruttore del pannello di indice contiene le righe:

```
public panelInserimento() {
        ...
        exit = new JButton("esci");
        ...
}
```

Andando ad esaminare il metodo `actionPerformed()` troviamo:

```
public void actionPerformed(ActionEvent evt) {
        ...
```

```
      String command = evt.getActionCommand();
      command = command.toLowerCase();
      ...
      if(command.equals("Esci")) {
            System.exit(0);
      }
      ...
}
```

Il gestore di azioni non verificherà mai la condizione riportata nel listato, a causa della precedente conversione in caratteri minuscoli. Per allineare quindi il carattere maiuscolo del pulsante di uscita con tutti gli altri presenti sui pannelli, e per interagire correttamente con la sua pressione, cambiamo il codice di `actionPerformed()` in questo modo:

```
public void actionPerformed(ActionEvent evt) {
      ...
      if(command.equals("esci")) {
            System.exit(0);
      }
      ...
}
```

Tale cambiamento nel codice risolve il baco: l'evento di pressione viene riconosciuto e il sistema risponde con la chiusura del frame aperto.

Proseguendo, ci siamo occupati del baco relativo alla creazione di una rubrica priva di nome. Analizzando il codice salta subito agli occhi il gestore dell'evento della pressione del tasto **Invio**. Esso si comporta come segue:

```
...
if(command.equals("invio")) {
      test=true;
      if(nameRub.equals(""))
            test=true;
      exFile = new File(nameRub);
      if( ((exFile).exists()) || (test==false)) {
            inNameRub.setText("");
                  if(test==false)
                        //nome del file non corretto
                  else
                        //file esistente
            ...
      ...
...
```

La variabile booleana `test` ha lo scopo di informarci circa il campo inserito come nome per la rubrica. Nel codice mostrato, `test` viene settata vera qualora il nome del file sia vuoto. Tuttavia, pochi righi più sotto troviamo la verifica di una condizione proprio su `test`, che qualora fosse falsa dovrebbe essere di allerta per un nome non corretto (addirittura proprio per un nome vuoto). La correzione di questo baco si ottiene banalmente riscrivendo:

```
...
if(command.equals("invio")) {
      test = true;
      if(nameRub.equals(""))
            test = false;
      ...
```

```
...
```

Dopo aver sistemato il codice, l'applicazione si comporta come previsto, interrompendo la sequenza precedentemente indicata nel caso in cui l'utente non abbia inserito alcun nome per la rubrica in fase di creazione.

Stesso procedimento è stato applicato per la funzione di ricerca all'interno della rubrica, che come illustrato sembra funzionare all'inverso.

Le seguenti istruzioni

```
...
cerca = new int[100];
cerca(stringa,nameRub);
i=0;
if(cerca[0]==0) {
      do {
            stampaRisultati(cerca[i]-1,nameRub);
            i++;
      } while(cerca[i]!=0);
} else {
      risCerca="Nessun contatto trovato!!";
}
...
```

ci informano che i risultati vengono stampati nel caso in cui il primo valore dell'array di interi `cerca` sia diverso da zero. Esaminiamo quindi la funzione (che molto sgraziatamente ha avuto attribuito lo stesso nome dell'array) `cerca` per valutare se vi possa essere annidato il baco di cui siamo alla ricerca.

Il blocco di codice di interesse esegue le seguenti istruzioni:

```
...
linea = lettura.readLine();
if(linea != null) {
      if(linea.equals(stringRicerca)) {
            cerca[i]=num;
            i++;
      }
}
...
```

La variabile `lettura` rappresenta un buffer di lettura aperto sul file di rubrica, che nel nostro caso è stato correttamente creato, mentre la variabile di conteggio `i` parte correttamente dal valore zero. Ne consegue che il baco non è in questo codice, in quanto il riempimento del vettore `cerca` avviene correttamente.

Tornano al gestore di eventi, l'ovvia soluzione è quella di cambiare la condizione per la stampa dei risultati in:

```
...
if(cerca[0] != 0) {
      do {
            stampaRisultati(cerca[i]-1,nameRub);
            i++;
      } while(cerca[i]!=0);
} else {
      risCerca="Nessun contatto trovato!!";
```

```
}
...
```

Proseguendo nelle operazioni di fixing dei bachi, si riporta l'analisi effettuata sul codice relativo ai tasti di navigazione. Le istruzioni preposte alla disabilitazione del tasto di navigazione incriminato sono le seguenti:

```
...
if(command.equals("<<")) {
      if(num>0) {
            num--;
      avanti.setEnabled(true);
}
if(num<0)
      indietro.setEnabled(false);
...
```

che cambiate in:

```
...
if(command.equals("<<")) {
      if(num>0) {
            num--;
      avanti.setEnabled(true);
}
if(num == 0)
      indietro.setEnabled(false);
...
```

eliminano ogni fastidioso residuo grafico dal pannello di navigazione.

Come ultima modifica funzionale apportata al codice, si riporta l'abilitazione del tasto **indietro** anche nel pannello di creazione rubrica, tramite la modifica del metodo `void pannelloRubIns(boolean att)`:

```
void pannelloRubIns(boolean att) {
      ...
      this.back.setVisible(att);
}
```

6 CodeCover e risultati di copertura

L'esecuzione della test suite contenete i metodi presentati nel capitolo precedente è stata analizzata tramite il tool automatico CodeCover, il quale consente di valutare la percentuale di copertura del codice sotto testing, la correlazione tra i metodi di testing implementati e diverse altre metriche inerenti al testing.

Si riporta in questa sede l'analisi di correlazione tramite l'immagine 6.1 nella pagina successiva. Si nota che la sovrapposizione tra i metodi impiegati per il testing è notevole, e tuttavia inevitabile. Difatti, ogni test case deve richiamare i metodi di attivazione e disattivazione dei pannelli per poter funzionare correttamente.

Il report complessivo sulla copertura del codice sotto testing mostra come risultato finale il 96%.

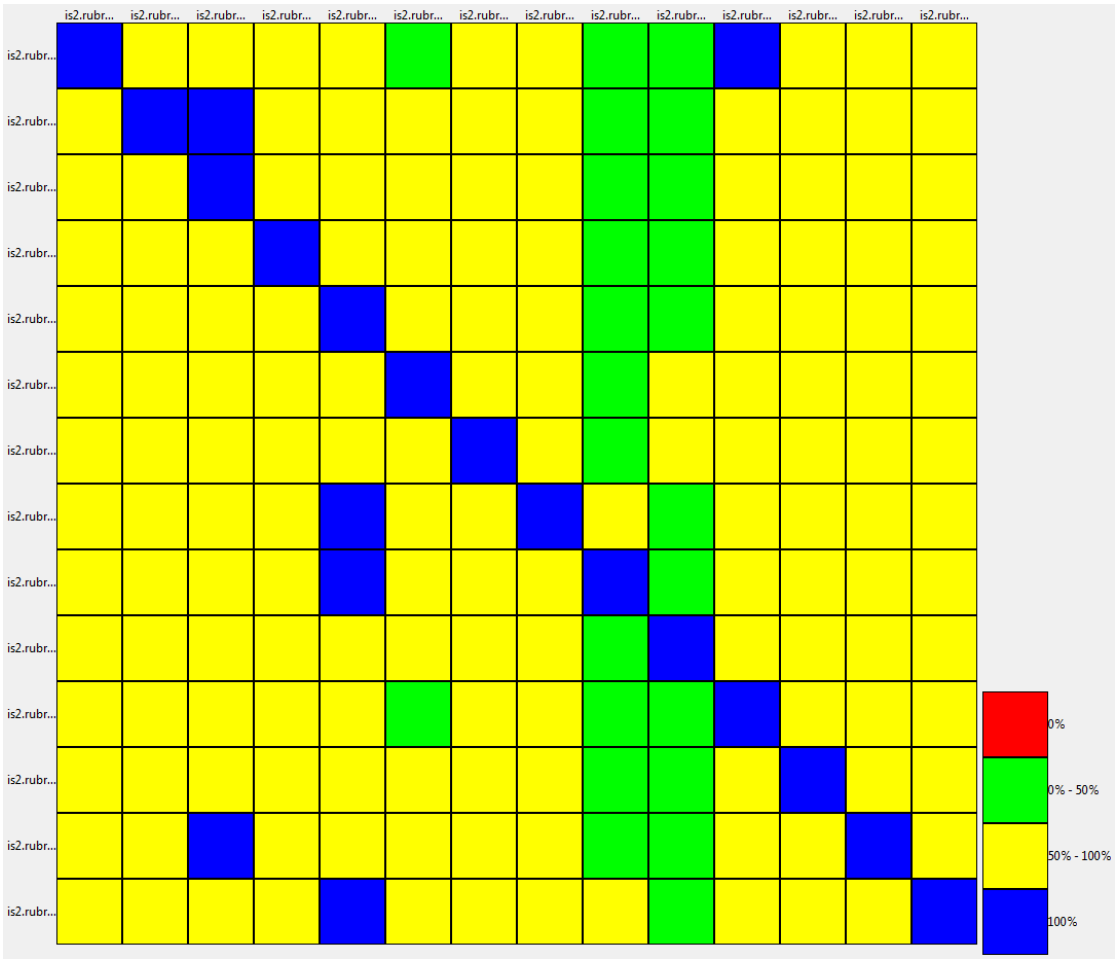

Figura 6.1: Correlazione tra i metodi di testing