



POLITECNICO
MILANO 1863

Corso di Ingegneria del Software

Prof. Marco Brambilla

Prova Finale

Java e testing

Gruppo 2

Cappelletti Andrea | Primerano Matteo | Maglione Sandro

Anno accademico 2018-2019

In questo documento viene presentata la realizzazione del sistema per il controllo e monitoraggio del traffico cittadino in Java.

INDICE DEI CONTENUTI

- [Considerazioni e scelte](#)
- [Configurazione ed avvio](#)
- [Sistema Centrale](#)
- [Applicazione Mobile](#)
- [Centralina](#)
- [Testing](#)

Considerazioni e scelte

Inizialmente si pensava di integrare delle api che fornissero una mappa con i dati di traffico (come Google Maps), ma si è scelto di procedere senza l'utilizzo di queste ultime in quanto si è ritenuto opportuno mettersi in gioco e provare a realizzare l'intero sistema tramite gli strumenti a disposizione. Questa scelta è risultata molto costruttiva, sia a livello didattico che a livello di esperienza. Si è deciso di realizzare la mappa tramite una matrice bidimensionale e di mostrarla graficamente con JavaFx. Maggiori dettagli vengono discussi in seguito.

Configurazione e avvio

Parametri preimpostati

Di seguito la lista di tutti i parametri di configurazione impostati di default all'interno dell'intero sistema:

- Valore traffico rilevato da centraline
Valore massimo = **20**
Valore minimo = **1**
- Velocità AppMobile i.e. numero di step (pixel) che l'indicatore può percorrere in un ciclo
Valore massimo = **5**
Valore minimo = **1**
- Tempo di scadenza di una segnalazione prima di essere rimossa = **60000** (1 minuto)
- Tempo tra la visualizzazione di due notifiche uguali sulla stessa AppMobile = **30000** (30 secondi)
- Tempo tra un invio e il successivo dei dati sul traffico delle centraline al sistema centrale (valore reale proporzionale al flusso di traffico)
Tempo massimo = **20000** (20 secondi)
Tempo minimo = **3000** (3 secondi)

Istruzioni per l'uso

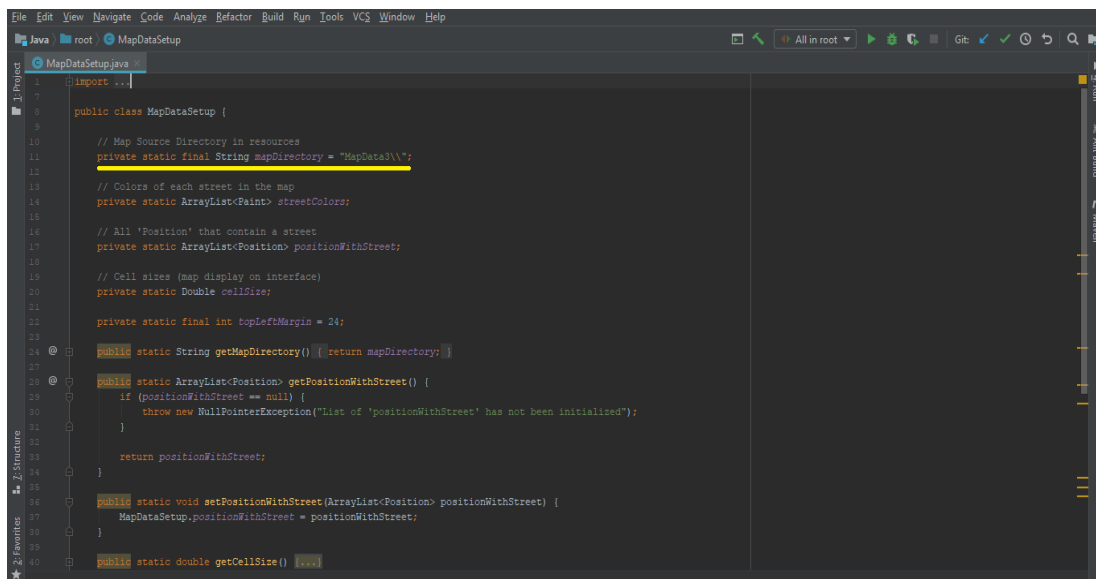
1. Indicare la mappa che si vuole visualizzare modificando la variabile *mapDirectory* all'interno della classe **MapDataSetup**.

Il percorso differisce in base al sistema operativo che si sta utilizzando.

Esempio: si supponga di voler aggiungere la mappa della directory MapData3:

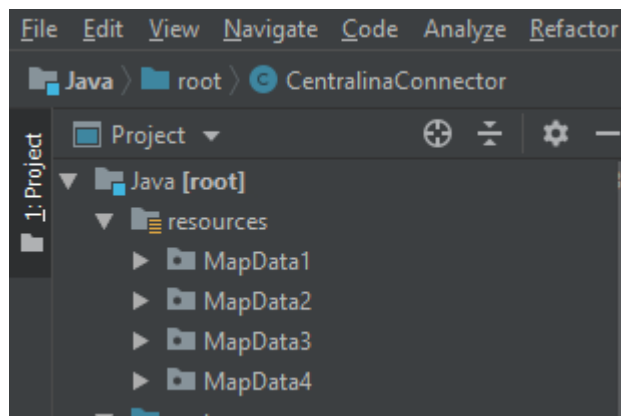
OSx: "MapData3/"

Windows: "MapData3\\"



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Java root MapDataSetup
MapDataSetup.java
1 import ...
2
3 public class MapDataSetup {
4
5     // Map Source Directory in resources
6     private static final String mapDirectory = "MapData3\\";
7
8     // Colors of each street in the map
9     private static ArrayList<Paint> streetColors;
10
11     // All 'Position' that contain a street
12     private static ArrayList<Position> positionWithStreet;
13
14     // Cell sizes (map display on interface)
15     private static Double cellSize;
16
17     private static final int topLeftMargin = 24;
18
19     @ public static String getMapDirectory() { return mapDirectory; }
20
21     @ public static ArrayList<Position> getPositionWithStreet() {
22         if (positionWithStreet == null) {
23             throw new NullPointerException("List of 'positionWithStreet' has not been initialized");
24         }
25         return positionWithStreet;
26     }
27
28     public static void setPositionWithStreet(ArrayList<Position> positionWithStreet) {
29         MapDataSetup.positionWithStreet = positionWithStreet;
30     }
31
32     public static double getCellSize() { ... }
33 }
```

Vengono rese disponibili quattro mappe nella cartella *resources*: MapData1, MapData2, MapData3, MapData4.

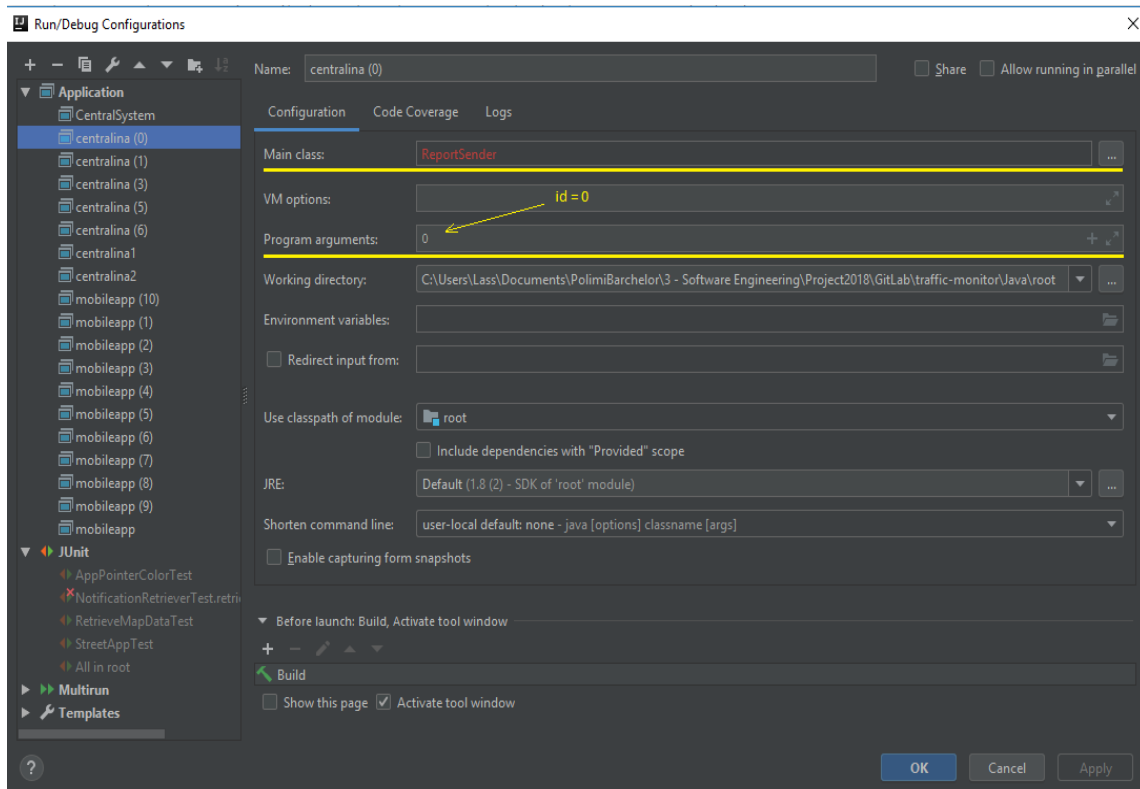


Nonostante ciò è sempre possibile aggiungere una configurazione a piacere seguendo lo standard utilizzato.

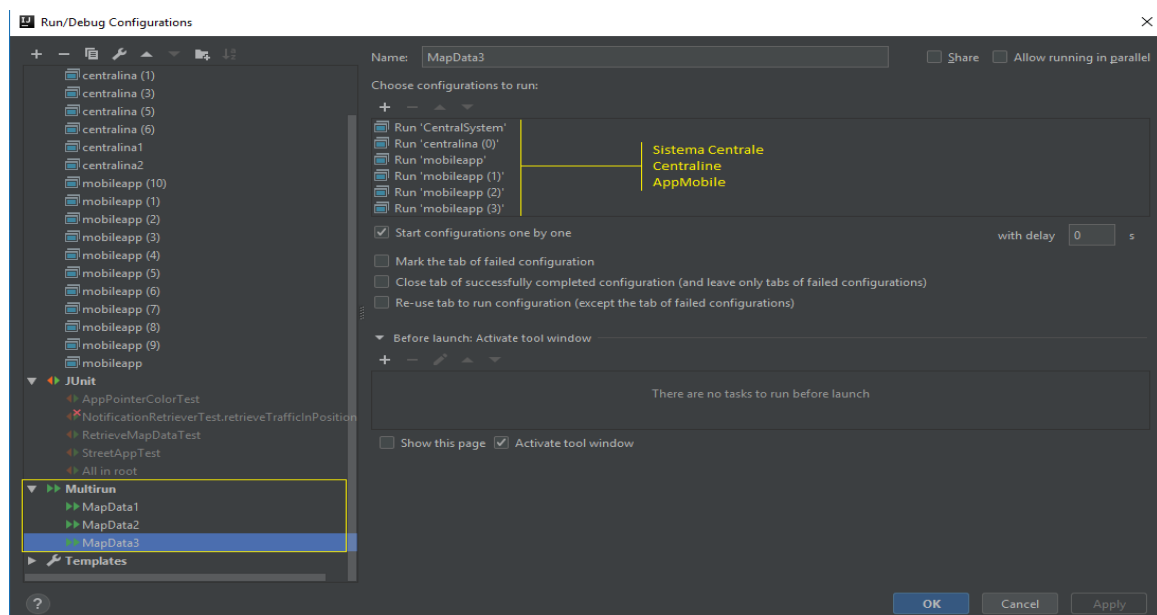
2. Ogni mappa necessita di un numero di centraline preimpostato. Ad ogni centralina dovrà essere assegnato uno specifico numero identificativo come parametro *args* al main di avvio. Di seguito la configurazione necessaria per ogni mappa presente:

MainClass Centralina: ReportSender

- MapData1:
Numero di centraline da istanziare: 2
id = 1, id = 5
- MapData2:
Numero di centraline da istanziare: 3
id = 0, id = 3, id = 6
- MapData3:
Numero di centraline da istanziare: 1
id = 0
- MapData4:
Numero di centraline da istanziare: 6
id = 0, id = 3, id = 6, id = 12, id = 16, id = 19



3. Bisogna avviare il sistema centrale per primo (classe **Main**.) In seguito si lanciano le centraline (classe **ReportSender**.) Infine, è possibile introdurre un numero di AppMobile arbitrario (classe **MobileAppInterface**.) A tale proposito consigliamo l'utilizzo del plugin [Multirun](#) per IntelliJ. Il plugin permette di lanciare con un solo click una serie di configurazioni diverse. Basta semplicemente creare le singole configurazioni e inserirle nella lista dell'unica configurazione Multirun. Il plugin si occuperà di lanciare in sequenza tutte le classi inserite.



Sistema centrale

Avvio

All'avvio del sistema, che avviene attraverso il *main()* presente nella classe **Main**, viene aperta la connessione RMI e viene reso disponibile uno stub della classe **SystemModuleConnector**, la quale implementa l'interfaccia **RemoteCentralSystemConnector**. Quest'ultima contiene i metodi necessari ai sotto-moduli (Centralina e AppMobile) per interagire con il sistema centrale, come ad esempio *sendReport()* e *RetrieveNotification()*.

Successivamente il sistema inizializza la mappa e i dati sulle strade. Queste informazioni vengono recuperate dalla cartella *resources* all'interno del progetto. Abbiamo incluso 4 diverse mappe in altrettante cartelle con nome *MapData[indice]*. La directory da cui recuperare i dati viene impostata all'interno della classe **MapDataSetup**, modificando opportunamente la variabile *mapDirectory*. La mappa consiste in una immagine .png denominata *map.png*. Il sistema processa l'immagine e restituisce una matrice dove ogni cella ha valore -1 se non esiste una strada in quel punto, oppure un valore intero che indica l'indice della strada corrispondente.

I dati sui nomi delle strade e la presenza delle centraline in esse sono recuperati dal file *streetData.txt*. Ogni riga contiene un primo carattere che indica la presenza o meno della centralina nella strada ('C' (centralina) ed 'E' (empty) rispettivamente), un separatore '|' ed il nome della strada.

Tutti i dati necessari al corretto funzionamento del sistema vengono salvati all'interno delle classi **MapManager** e **TrafficReportDatabase** all'avvio del sistema.

Una volta completati i settaggi preliminari viene lanciata l'interfaccia grafica.

Memorizzare informazioni di stato

D'ora in avanti il sistema si aggiorna ogni qual volta riceve una richiesta o un report da un sotto-modulo esterno. Esso provvederà a salvare opportunamente i dati sul traffico, aggiornando la variabile *savedTrafficData*, di tipo **StreetList**, all'interno della classe

TrafficReportDatabase.

La classe **StreetList** contiene un'ArrayList di oggetti di tipo **Street**. Ogni variabile all'interno della lista contiene le informazioni sul traffico corrispondenti sotto forma di un numero intero. Nel caso la strada sia coperta da una centralina, la lista contiene un oggetto di tipo **StreetCovered** e la variabile *trafficAmount* corrisponde al valore ricevuto dalla centralina esterna corrispondente.

Nel caso la strada non sia coperta da una centralina, la lista contiene un oggetto di tipo **StreetEmpty**, con al suo interno una lista di **AppReport**. Ogni segnalazione ricevuta dall'AppMobile viene salvata all'interno di questa lista e il valore del *trafficAmount* corrisponde alle dimensioni della lista stessa.

La classe **TrafficReportDatabase** ha poi al suo interno un timer, sviluppato utilizzando la classe [Timer](#), che provvederà periodicamente ad aggiornare la lista di segnalazioni, eliminando eventuali segnalazioni scadute.

Invio notifiche ai sistemi esterni

L'invio delle notifiche avviene attraverso la classe **NotificationRetriever**.

Ogni AppMobile invierà periodicamente una richiesta di notifica al sistema centrale tramite RMI, segnalando la propria posizione attuale. Il sistema recupera la lista delle strade presenti in un intorno fisso dalla posizione dell'AppMobile e il corrispettivo dato di traffico attuale.

Successivamente questa lista viene filtrata eliminando tutte le strade con valore di traffico inferiore

ad una data soglia. Infine, la lista di strade rimaste viene ritornata all'AppMobile che provvederà a visualizzare le notifiche ricevute.

Mostrare lo stato del sistema (Interfaccia grafica)

L'interfaccia grafica è sviluppata utilizzando la libreria [JavaFX](#).

Viene mostrata la mappa sulla sinistra dello schermo e i dati sul traffico sulla destra.

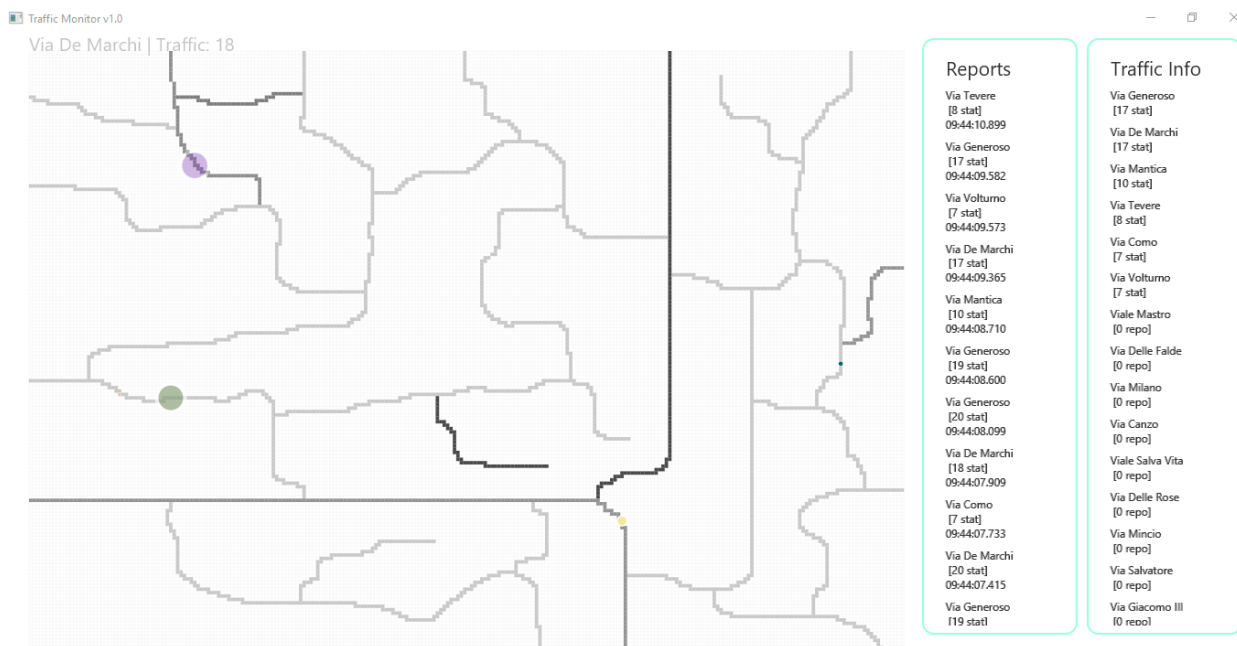
I dati sul traffico consistono in due liste:

- La lista (sulla destra) mostra il valore di traffico attuale per ciascuna *Street* ordinato a partire dalla strada più trafficata;
- La lista (sulla sinistra) mostra l'arrivo dei report dai vari sotto-moduli in tempo reale.

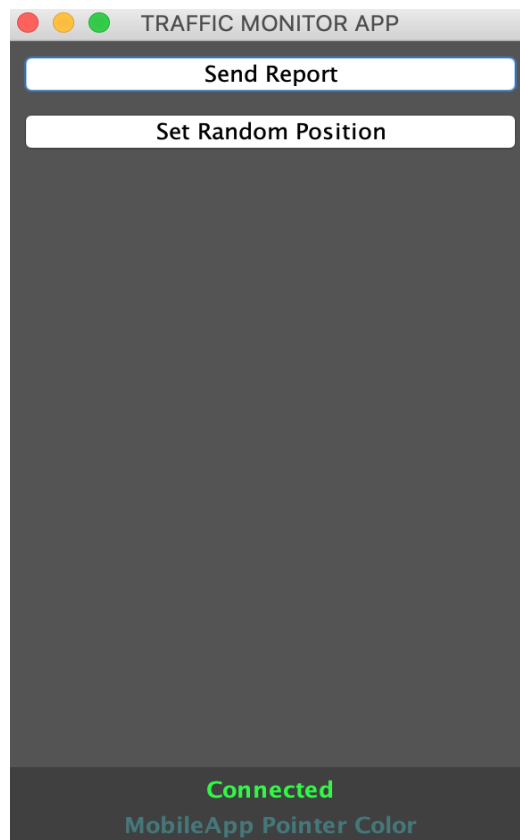
La mappa mostra tutte le strade e la posizione delle varie AppMobile.

Le strade assumono un colore con gradazione scura proporzionale alla quantità di traffico presente. È possibile riconoscere il nome di ogni strada tramite una label di testo in alto a sinistra della mappa. Questa si aggiorna ogni qual volta il mouse passa sopra una strada, indicandone il nome e il dato di traffico.

Le AppMobile possono essere riconosciute da degli indicatori colorati (ogni AppMobile ha un proprio colore specifico.) Questi indicatori vengono mostrati ogni qual volta un'applicazione interagisce con il sistema centrale (per una segnalazione o per il recupero delle notifiche.)



Applicazione mobile



Avvio interfaccia grafica

L'interfaccia grafica dell'applicazione mobile è stata sviluppata utilizzando la libreria grafica swing di Java all'interno della classe `MobileAppInterface`. Essa consiste in due `Button` e due `TextLabel`. Il primo pulsante permette l'invio di una segnalazione di traffico al sistema centrale, mentre il secondo permette di modificare casualmente la propria posizione all'interno della mappa. Le due `TextLabel` permettono rispettivamente di segnalare la corretta connessione dell'applicazione al sistema centrale e il colore dell'indicatore sulla mappa all'interno dell'interfaccia del sistema centrale.

Invio segnalazioni di traffico

L'utente può segnalare una condizione di traffico nella propria posizione cliccando sul tasto *SendReport*. Viene inviata la posizione attuale dell'applicazione mobile al sistema centrale tramite RMI. Il sistema si occuperà di salvare opportunamente il report ricevuto. L'applicazione mobile deve aspettare un periodo di tempo preimpostato prima dell'invio di una seconda segnalazione, in modo da evitare lo spamming di report al sistema centrale.

Ricezione notifiche da sistema centrale

L'applicazione mobile implementa un timer all'interno della classe `UpdatingAppTimer`. La durata del timer è proporzionale alla velocità di movimento, calcolata all'interno della classe `MobileAppMovement`. Allo scadere del timer, l'applicazione si collega con il sistema centrale

inviando la propria posizione attuale. Il compito del sistema è processare l'informazione ricevuta e ritornare una lista di strade in cui vi è traffico. Sfruttando i dati ricevuti dalle centraline e eventuali segnalazioni da altri utenti, salvati all'interno della classe **TrafficReportDatabase**, il sistema recupera una lista di strade con il corrispettivo traffico in un raggio fisso dalla posizione ricevuta, sotto forma di oggetti di tipo **StreetList**. L'applicazione mobile provvederà a mostrare un pop-up (*JOptionPane*) con le varie notifiche ricevute.

Centralina

La divisione in classi all'interno della centralina rispecchia gli obiettivi che il modulo centralina deve raggiungere: la generazione dei report di traffico, la temporizzazione degli invii al sistema centrale e l'invio dei report vengono gestiti rispettivamente dalle classi **ReportCreator**, **TimerManager** e **CentralinaConnector**.

La classe **ReportSender** invece contiene solo metodi statici e fa da supporto alle altre tre, in particolare contiene il *main()* che si occupa di istanziare tutte le classi definendo anche l'id e quindi la strada sulla quale la centralina opera.

All'interno del **TimerManager** abbiamo il metodo *updateTimerCounter(int lastTrafficAmount)* che si occupa fra l'altro di aggiornare il tempo che deve trascorrere prima dell'invio del prossimo report: una semplice funzione definisce il valore in *ms* proporzionalmente all'ultima intensità di traffico rilevata nella via, più il traffico è intenso minore sarà l'attesa per la prossima rilevazione.

La temporizzazione è affidata ad un oggetto Timer in *backgroundTimerHandler()*. Dentro a **ReportCreator** viene simulato l'andamento del traffico con un valore che viene casualmente incrementato o decrementato di un certo numero (anch'esso casuale) di unità. Questa simulazione è affidata al metodo statico *retriveTrafficData* che viene richiamato da *trafficGenerator* anche qui attraverso un oggetto Timer. Questo è indipendente dal timer in **TimerManager**, cosa necessaria per evitare dipendenze fra l'istante di invio del report e quello di generazione (continua) del traffico.

CentralinaConnector apre la connessione con il sistema centrale in fase di creazione della centralina e contiene il metodo *retrieveTrafficInCentralina()*. Questo viene richiamato dal server centrale ogni qualvolta un'applicazione mobile invia una segnalazione di traffico da una strada in cui è presente anche una centralina. Viene così verificata la segnalazione fatta dall'utente.

Testing

Sono stati implementati 69 casi di test JUnit. Lo scopo è testare le funzionalità dei singoli metodi, appurando il loro corretto funzionamento sotto diverse configurazioni. Al momento della consegna, tutti i 69 test restituiscono un risultato positivo, come mostrato nell'immagine seguente. Ciò garantisce che i metodi e le classi testate funzionano come previsto.

