

BusFinder

PROJECT REPORT

Wangdu Gyalpo (4415)

Mobile and Internet services

Date: 29th January 2010

Contents

Project Description

System Functionalities

Find bus

List bus stops

Find bus stop

Architecture

Managing Performance

Screen navigation management

Generic thread management

Application screenshots

Code structure

Package controller

Package model

Package ui

Package utils

Package main

Major technical problems and solutions

References

Project description:

This project is developed for the course Mobile and internet services of the Free University of Bolzano. Bus Finder is a mobile application that is aimed to help bolzanonians find the bus to reach to their destination. The application asks the user to either insert the destination or select from the bus stop list and returns a timetable of the bus which goes towards the destination he has selected. In addition to the timetable, the application informs the user the bus stop that he should get off. The result is bus name or number and the time the bus leaves starting from current time if the time is not specified during the query and finally the bus stop name that he should get down.

System Functionalities:

Find Bus

The main function of this mobile application is to find the bus for the user. The system has a very simple input screen with the source field, destination field and a time field. The source and destination fields takes in a address of Bolzano or the user can use the command to select bus stop name from the lists. In case the user types in the address the result will be given back only if there is the address in the file. In addition the user has to specify the time at which the user wants to take the bus. The user is shown the current time and the bus search will be done using the time incase the user didn't change it for a later time.

List bus stops

Another function for the user of this application is the list of bus stops. The user can select the option to list the bus stops. This option can be used by the user in case the user wants to quickly see the list of the bus stops.

Find Bus stop

This function can give the user the name of the bus stop and the bus lines available for that bus stop. The user has to input a street name and the system will find for the user the bus stop name that is on that street together with the bus lines.

This function is useful when the user is on some part of the city and the user knows the street name. So he can get the name of the bus stop and the name of the bus lines that pass by that bus stop.

Architecture

The application is composed of the main application and the Data Files.

The bus lines and bus stops data are stored in files. The Bus Finder application uses these files to do the operations. There is a bus stop file "busstops.txt" which has the list of all the bus stops and for each bus line there is a file referenced with its name.

1 CARDANO – VIA FAGO (LUNEDÌ-VENERDÌ) KARDAUN – FAGENSTR. (MONTAG-FREITAG)												
					S	S	dalle von	minuti Minuten	alle bis			
Cardano		6.50	7.10	7.20			7.35	.05 .20 .35 .50	18.35	18.50		Kardaun
Funivia del Renon		6.55	7.15	7.25			7.40	.10 .25 .40 .55	18.40	18.55		Rittner Seilbahn
Stazione 3		6.57	7.17	7.27	7.33	7.35	7.42	.12 .27 .42 .57	18.42	18.57		Bahnhof 3
P. Domenicani		6.59	7.19	7.29	7.35	7.37	7.44	.14 .29 .44 .59	18.44	18.59		Dominikanerplatz
Via Diaz 1	6.43	7.02	7.22	7.33	7.39	7.41	7.48	.18 .33 .48 .03	18.48	19.02		Diazstr. 1
Via Fago 1	6.45	7.04	7.24	7.36	7.42	7.44	7.51	.21 .36 .51 .06	18.51	19.04		Fagenstr. 1

Bolzano bus line (cardano to via Fago 1)

Managing performance

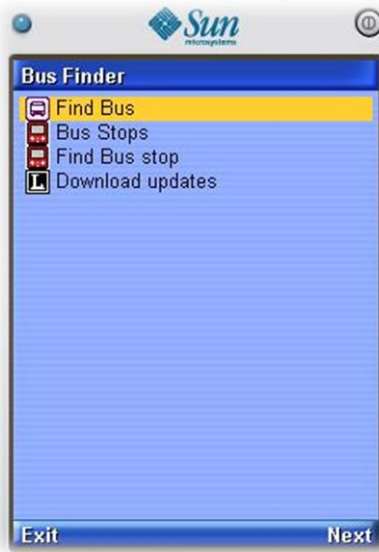
Screen navigation management

Since mobile applications are mainly navigating through the screens, one of the challenges is to handle the user interfaces navigations easy and convenient for the user. Particularly, the back navigation feature for the user to go back to the previous screen is to be handled well. Since, the displays that have been gone through by the user are still reachable in the application; pushing them inside a stack is the best way to retrieve them back when the user chooses to go back previous screen. Therefore, a controller should be made which handles the pushing of displayables in the stack and removing them also.

Generic thread management

There are many threads that are created for each of the operation by the user. It is mainly to inform the user that some process is running and to avoid hanging up the user interface. As these threads are created often, the generic thread management by one of the previous student of this course is useful in centralizing the thread management. Not only does it make it easier to create threads but it also has a listener which calls the proper function depending on the completion of the thread.

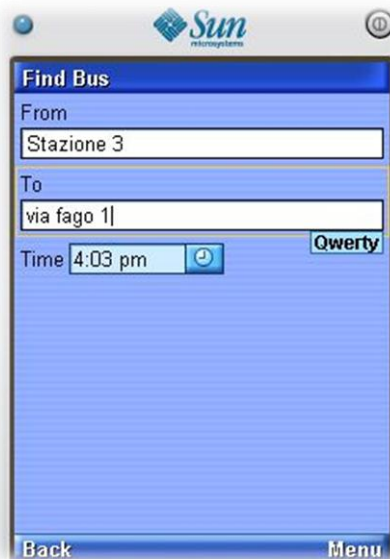
Application screen shots



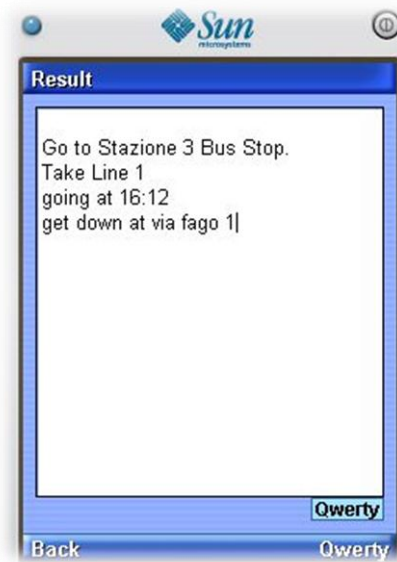
Main screen of the application



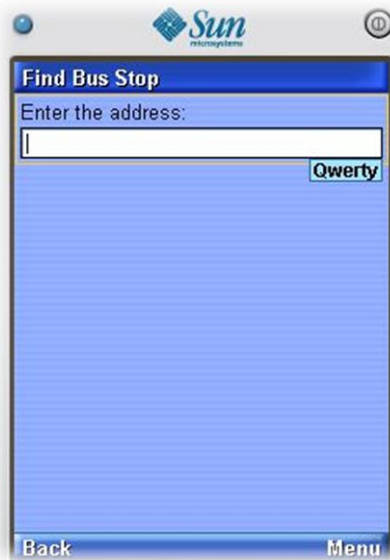
progress screen



Find bus user interface



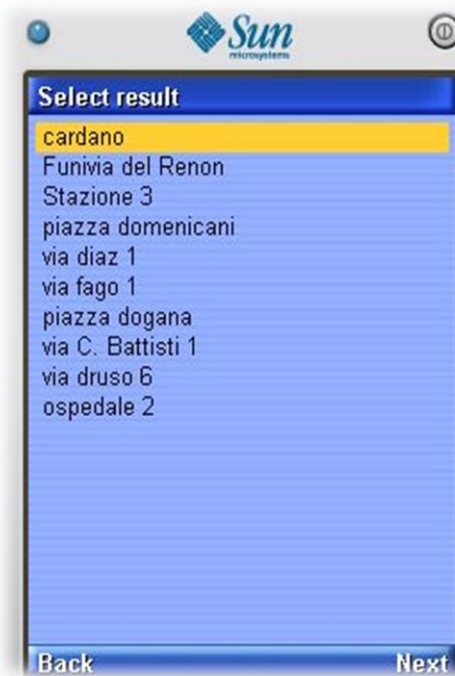
result of the bus search



Find bus stop screen



text field item command



Bus stops list screen

Code structure

Depending on the functions of the classes, they are grouped into different packages. They are described below with their roles and the main classes of the packages. Although the packages might imply the MVC design pattern, it is not fully MVC. It is mainly because some of the controlling logic need to be integrated with the user interface logic. However, the different logics has been separated as much as possible.

Package controller

This package acts as a controller for the application and therefore all the business logic are implemented in this package.

The main class of this package is the BusFinder.java class which consumes the user inputs from the user interface and invokes the file reader classes (BusStopFileReader.java & BusLineFileReader.java) to complete the operation. Since these classes handle the operation in the background, all of them are threads. The other two classes in this package are the Abstract class ExecutableTask.java and the TaskProcessListener.java. These two abstract classes have the responsibility to handle the multiple threads of the application. I have found this design used by a previous student of Mobile and Internet services lecture.

Package model

There are two javabean style classes that represents the model of the application. They are the BusStop.java and BusLine.java. As you can imagine, there are private attributes and the getters and setters for these attributes.

These two classes are instantiated during the search and the queries are done on them. Of course, at any time, there can be more than one instances of these classes. However, the implementation has been made in a way that it will never create all the instances for all the bus stops at any given time. This is achieved by filtering right after reading the file and creating only the instances that fits the input while the file is read. The idea here is not to overload the application's process.

Package ui

All the user interface classes; screens, forms, lists etc are grouped into this package. It is the package which forms the uppermost layer that the user can interact with.

The main classes of this package are the MainUI.java; which is a list and is the entry point for the application, the BusFinderUI.java; which is a form with some input fields for the user to search for the bus, and the BusStopFinderUI.java; which is again a form with one input field for the user to type in the address.

Package utils

The features that are used in the application which is general and invoked often are inside the utils package. The ImageUtil.java has only one static function loadImage(String imageName) which loads the image. The Constants.java holds all the constants which are in the application. Then there is ItemList.java which is utility for creating a List screen and it is used for displaying dynamic results as a list. Finally all the parsing of file and consequently creating the instances of javabeans object are all in the FileManager.java class.

Package main

The main package holds the BusFinderMIDlet.java class which is the entry point for the application. The midlet when started immediately hands off the control to the MainUI.java class which is then waiting for the user selection of functions.

Major technical problems and solutions

General constrained programming

Since applications developed for mobile devices has limited operating environment, it was always a problem to understand during the development how much resources and processing power it would use. There are best practices which might be useful to take care of the programming part. However, it is difficult to adapt since we are used to full blown features of desktop application.

Data representation and storage

Initially the idea was to use both file and record store for the bus stops and bus line data. However, the queries to the record store is quite complex and therefore, it was a good decision to make the queries on the instances created from the data. Meanwhile, the records that are entered to the record store would be taken from the files, it was a redundant job to store them into record store and create instances from them. Therefore, directly reading from the file and then creating the objects needed was a better solution.

Bus time queries

The time schedule of the bus was first stored as double value but trying a simple operation like if the user input time is before or after certain time was way too difficult. So, to use the java calendar object was far easier to do this operation and therefore, the start time and end time of the bus line at a certain bus stop was converted into java calendar object. This was a good solution for this problem.

References

Internet & mobile service lecture <http://www.inf.unibz.it/~ricci/MS/index.html>