Employee Management System

Ralph Quinto

Patrick Roderman

October 4, 2015

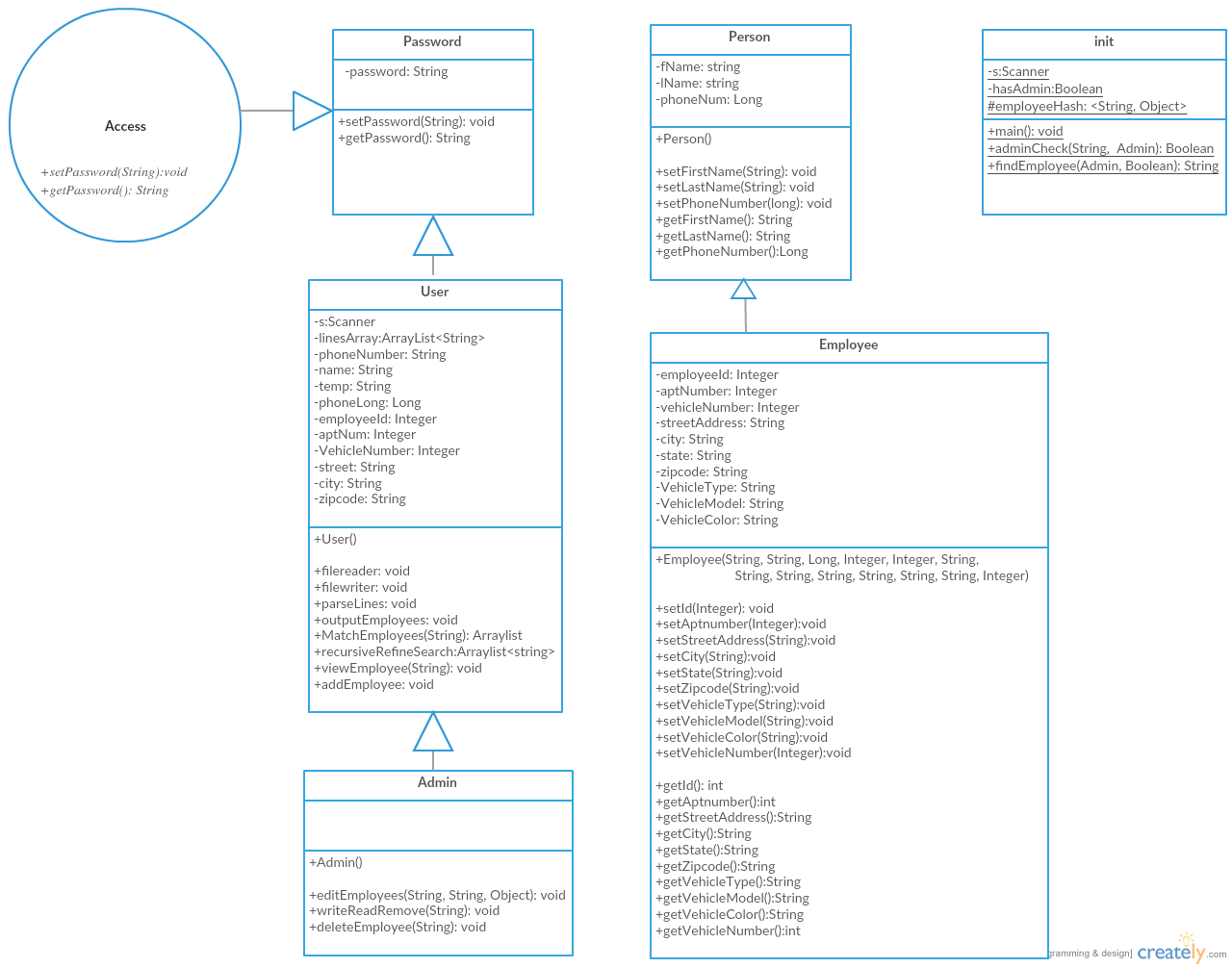
Introduction

For this project we were tasked to design and implement a program, which stores and reads employee details. The program should contain functionality to add, delete, edit and view employee details with different permissions for a user and an admin. A main goal for this project is to apply the object oriented programming principles we learned in class.

Theory/Background

Java is an object oriented programming language. As such, the concepts of inheritance, encapsulation, polymorphism, and abstraction should be utilized. Encapsulation is used to isolate and grant limited access to data. In the scope of this project, the passwords are the key pieces of data that should be encapsulated. Additionally, the employee information has private access modifiers to further exemplify encapsulation. Inheritance is heavily used in object oriented programming to propagate methods from classes to their subclasses. In this lab, inheritance can be used to create common methods between the User and Admin classes. Polymorphism takes place when methods are overridden or overloaded, and it is inherent when utilizing abstraction. For instance, when extending an abstract class, it is required to override the body of its method. In this project, polymorphism can be used when implementing an interface for password methods.

Design



**Init Class**

The init class handles all the input and output. When methods originating from other classes are called, the input is used as the parameters. The methods inside this class are used to consolidate code that is used multiple times in the class to check admin privileges/search for a specified employee. Additionally, the employee LinkedHashMap resides inside this class, which holds each employee’s first name, last name and ID as the *key*, and the employee object itself as the *value*.

**Person Class**

The Person class contains the first name, last name, and phone number. The methods consist of setter and getter methods for these variables.

**Employee Class**

This class contains the rest of the employee attributes. The methods consist of setter and getter methods for these variables.

**Access Interface**

The purpose of the Access Interface is to force the Password class to implement the set and get abstract methods.

**Password Abstract Class**

The Password class is created as an abstract class in order to ensure that the password methods are declared in both User and Admin. This creates separate encapsulated Strings for passwords for each respective user type.

**User Class**

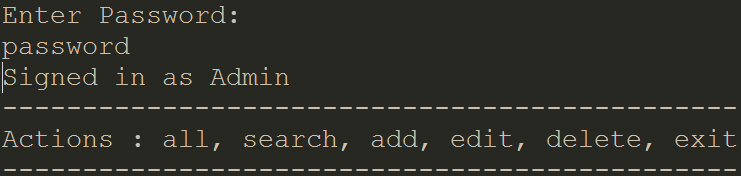
This class contains base methods that are utilized by the User and also inherited by the Admin class. The constructor sets the User password to “csc250”.

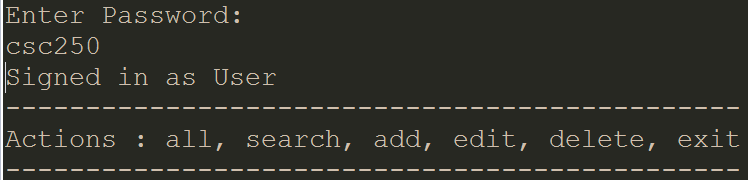
**Admin Class**

This class contains the unique methods: editEmployee(), and deleteEmployee(), which modify an employee entry and delete an employee entry respectively. The constructor sets the User password to “password”.

Results/Sample Outputs

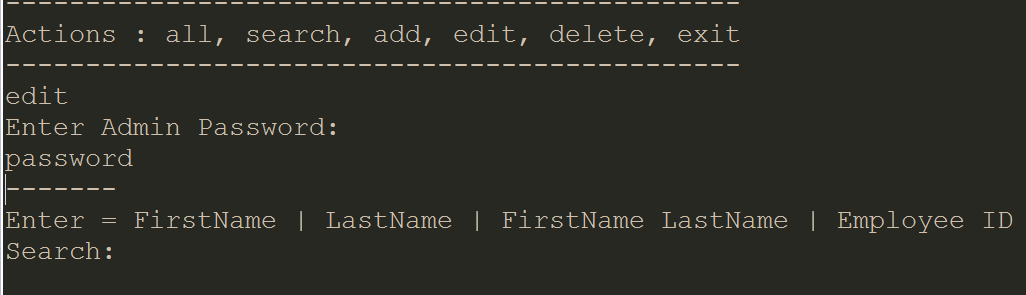
**Admin/User Password:**





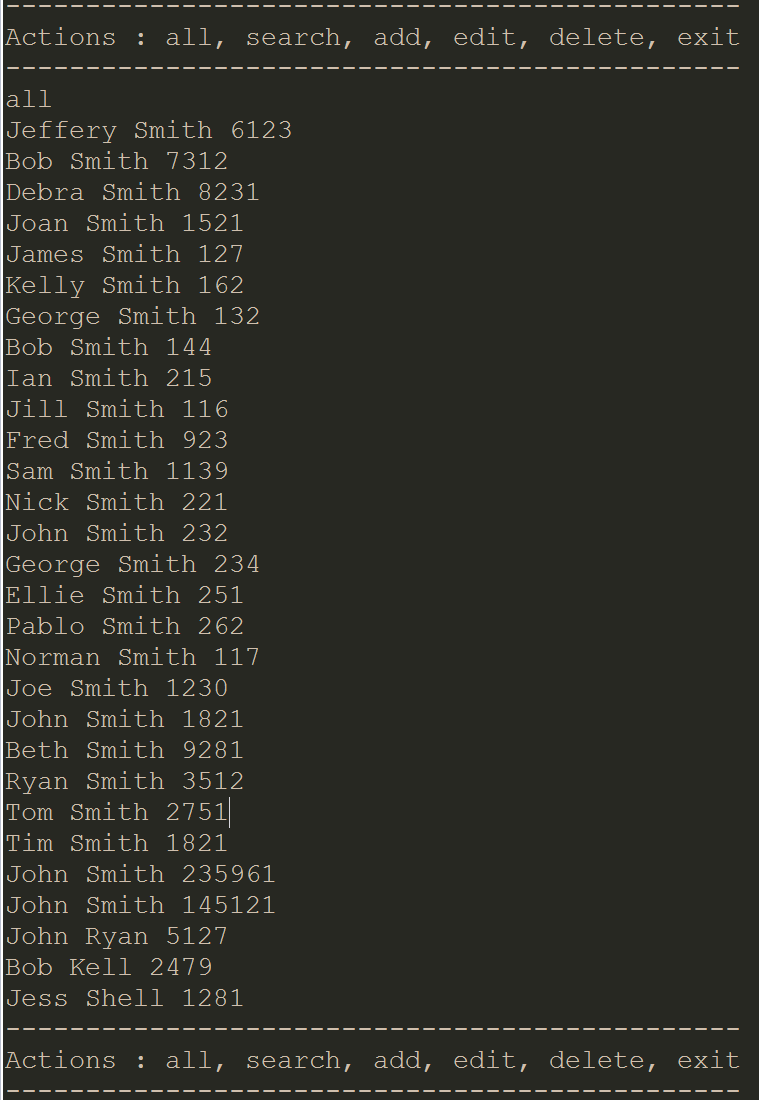
*Admin Check/Privilege*

(must enter the admin password to access edit and delete as user)



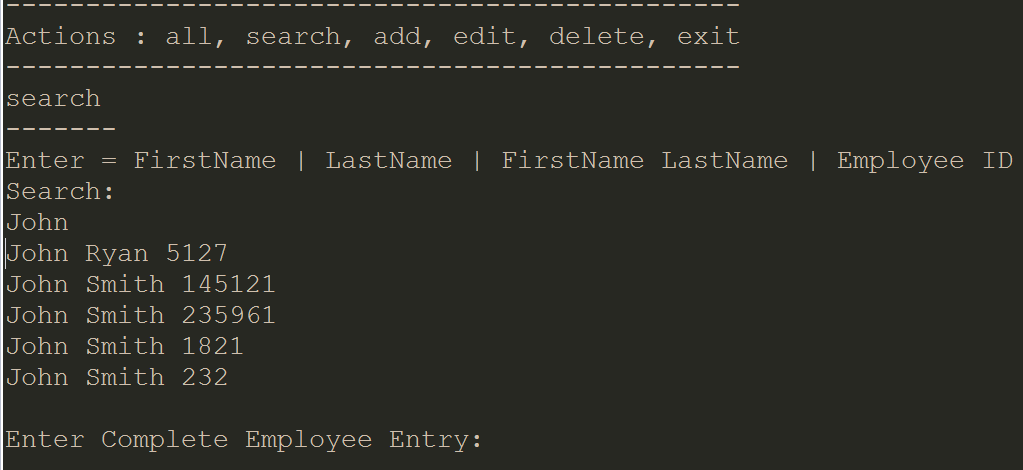
**All:**

Displays all keys in employee HashMap (FirstName LastName EmployeeId)

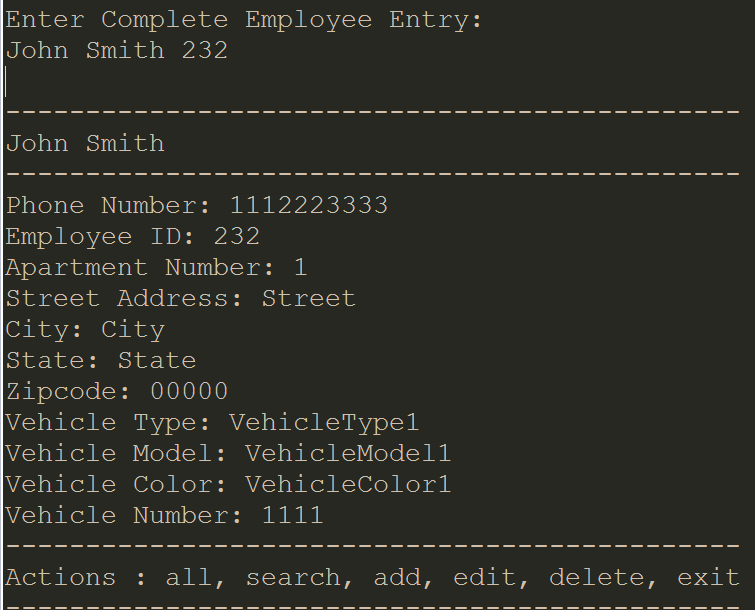


**Search:**

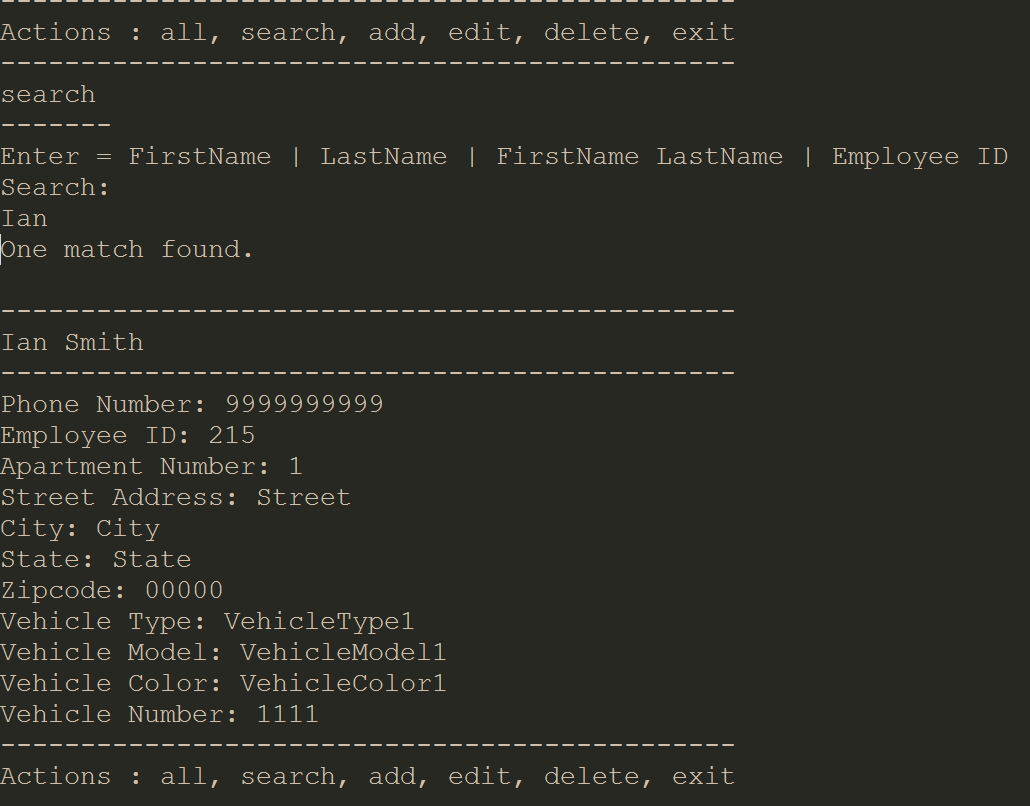
Compares String input to the Keys in HashMap and returns matches



Prompts user to enter full entry/key.



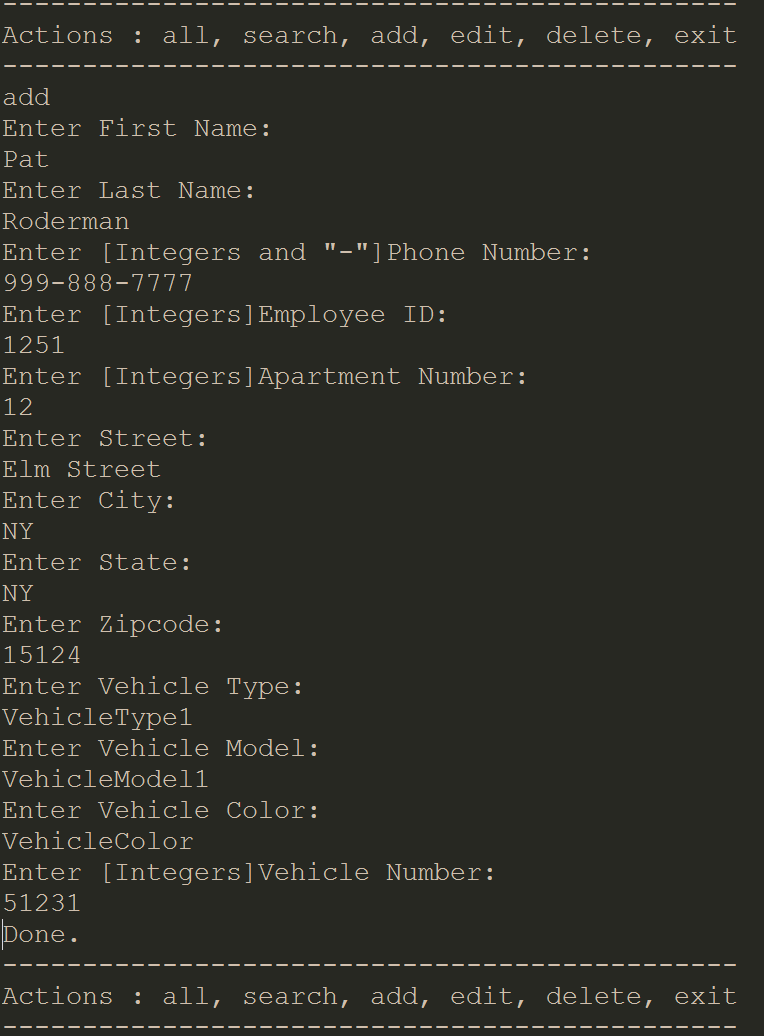
If there's only one matching employee, displays that employees info.



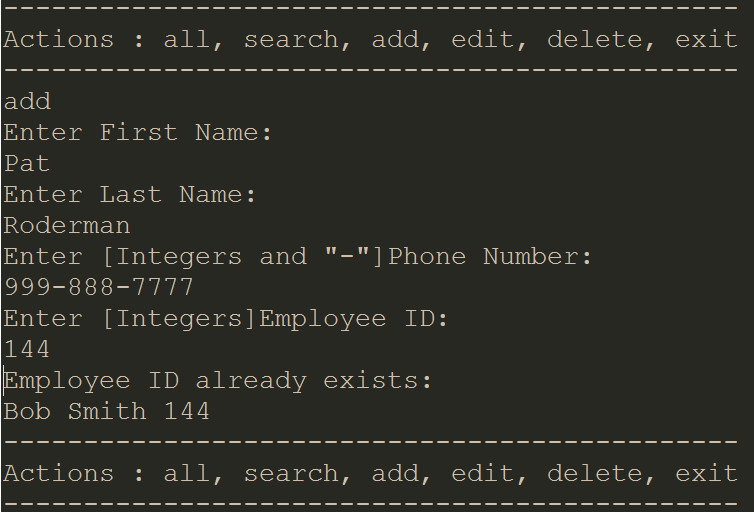
**Add:**

Loops through all necessary Employee Information for user input.

If given a wrong entry type, a message will appear notifying you, and setting a default value(0 for numbers, NA for Strings)

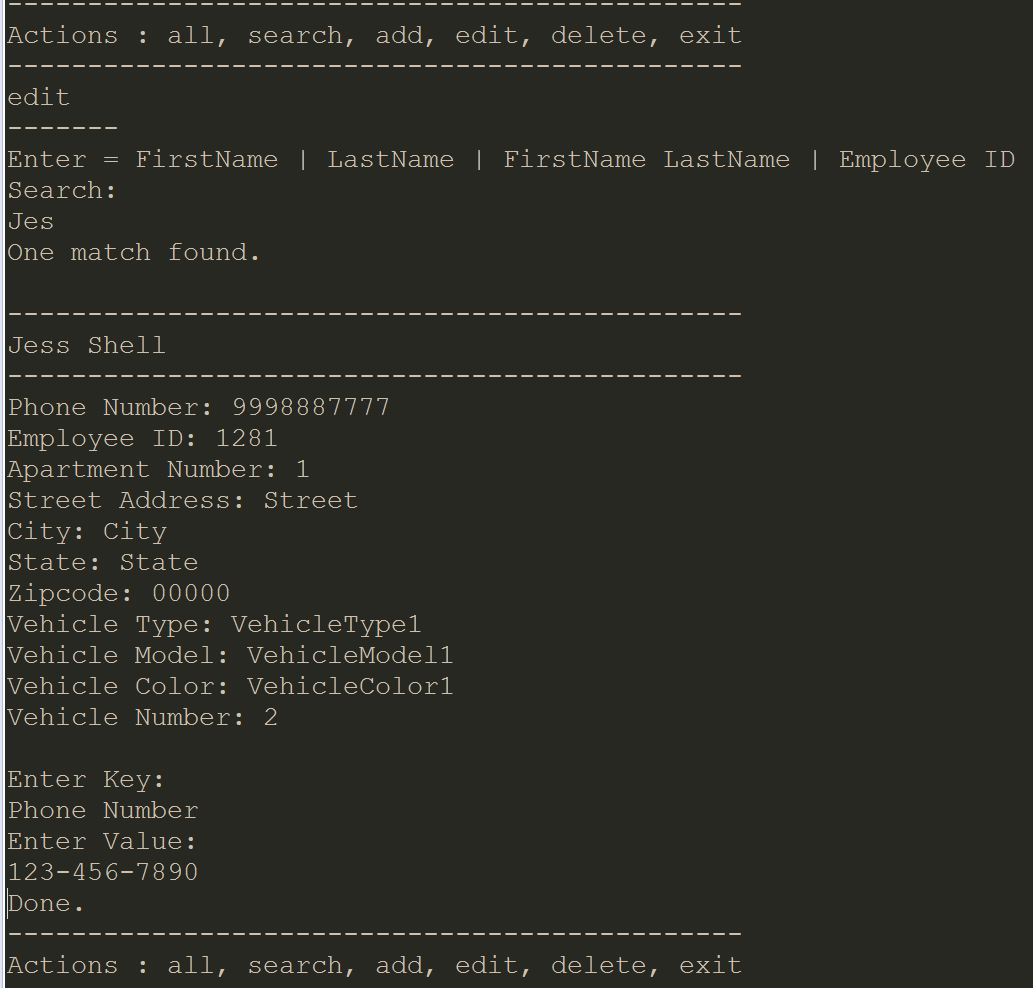


The employee ID is unique per employee, so if the ID you enter is found within the entries of employees, it will let you know which employee has that ID already and cancel the “add” action.



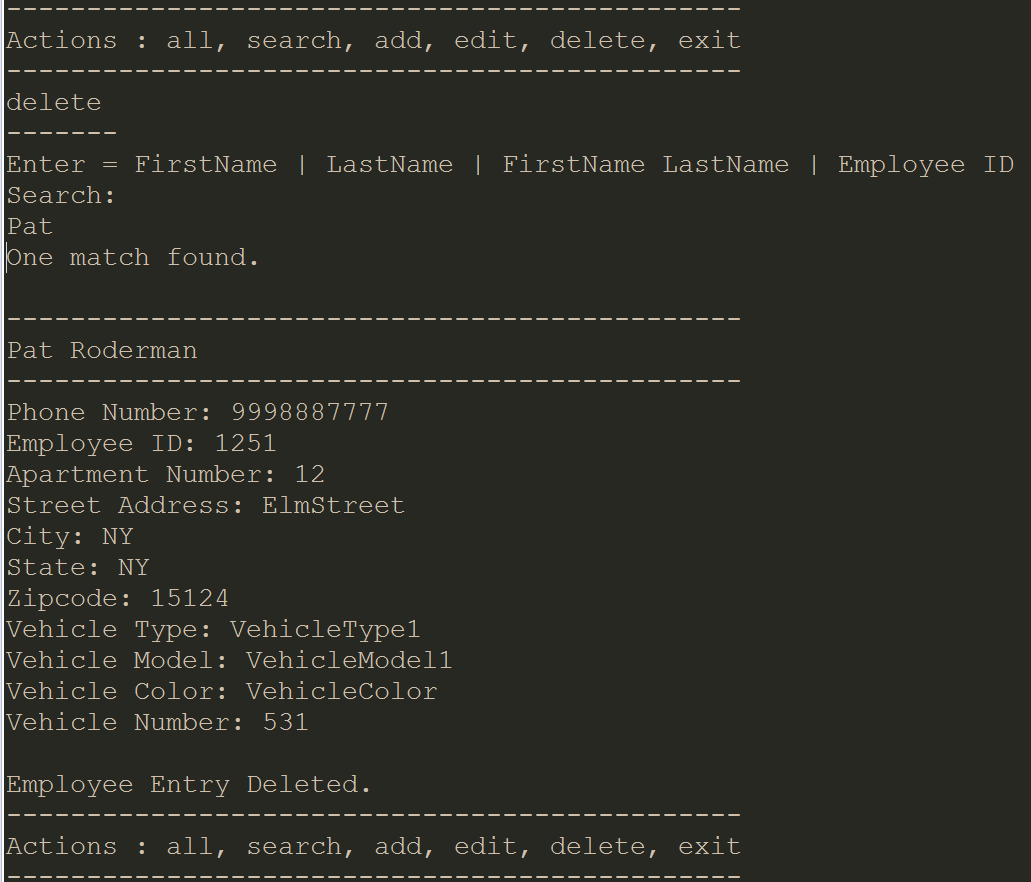
**Edit:**

Uses same methods as Search to get the employee you want to edit. Then prompts user for the Key and Value for the piece of information you want to change. You must enter a valid Key(not case sensitive / compares to lowercase), and a new value, or it will let you know and cancel the edit.



**Delete:**

Uses same methods as Search to get the employee you want to edit and then deletes the entry.



Conclusion

Utilizing object oriented programming principles of encapsulation, inheritance, abstraction and polymorphism, we created a functioning employee management system. From this implementation, we learned the importance of modular code and OOP. We met all the project requirements and produced a working employee management system.

Team Member’s Contributions

Patrick Roderman

Created the Init, Person, Employee, User, Admin Classes.

Ralph Quinto

Filled in the UML diagram and made the Access Interface and Password Abstract Class.

References

We referenced the Java Documentation when needed, to complete the assignment.

Program codes

***Submitted Separately***