```java
package lab6;

import java.io.*;
import java.util.*;
/**
 * Print out employee menu and store to file
 * @author alexb
 *
 */
public class EmployeeMenu implements Serializable{
    public Map<Integer, Employee> empMap;
    public Map<Employee, Integer> gradeMap;
    private static File file;
    /**
     * Default EmployeeEmnu constructor
     */
    public EmployeeMenu() {
        // *Redo with TreeMap
        empMap= new HashMap<Integer, Employee>() ;
        gradeMap = new HashMap<Employee, Integer>();
    }
    /**
     * Saves to file
     * @param em Employee Map
     * @param gm Grade Map
     * @throws FileNotFoundException
     * @throws IOException
     */
    void save(Map<Integer, Employee> em, Map<Employee, Integer> gm) throws FileNotFoundException, IOException {
        ObjectOutputStream oos1 = new ObjectOutputStream(new FileOutputStream("Employee.dat"));
        oos1.writeObject(em);
        oos1.close();

        ObjectOutputStream oos2 = new ObjectOutputStream(new FileOutputStream("Employee.dat"));
        oos2.writeObject(gm);
        oos2.close();
    }
    /**
     * Gets employee map from file
     * @return returns employee map
     * @throws FileNotFoundException file not found
     * @throws IOException cant read it in
     * @throws ClassNotFoundException class not found
     */
    public Map<Integer, Employee> getEmployee() throws FileNotFoundException, IOException, ClassNotFoundException {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream("Employee.dat"));
        empMap = (Map<Integer, Employee>) ois.readObject();
        ois.close();
        return empMap;
    }
    /**
     * Gets employee performace
     * @return returns performace map
```

```
54      * @throws FileNotFoundException file not found
55      * @throws IOException cant read it in
56      * @throws ClassNotFoundException class not found
57      */
58     public Map<Employee, Integer> getPerformance() throws FileNotFoundException, IOException,
   ClassNotFoundException {
59         ObjectInputStream ois = new ObjectInputStream(new FileInputStream("Employee.dat"));
60         gradeMap = (Map<Employee, Integer>) ois.readObject();
61         ois.close();
62         return gradeMap;
63     }
64     /**
65      * Runs the map save
66      * @param empMen Employee Menu
67      * @throws FileNotFoundException file not found
68      * @throws ClassNotFoundException class not found
69      * @throws IOException cant read it in
70      */
71     public void run(EmployeeMenu empMen) throws FileNotFoundException, ClassNotFoundException,
   IOException {
72         file = new File("Employee.dat");
73 
74         if (file.length() == 0)
75         {
76             System.out.println("File is empty, generated empty map");
77             empMen = new EmployeeMenu();
78         }
79         else
80         {
81             System.out.println("Filling map with data from file.");
82             empMen.empMap = getEmployee();
83             empMen.gradeMap = getPerformance();
84         }
85         Scanner scan = new Scanner(System.in);
86 
87         boolean quit = false;
88         while(!quit) {
89             System.out.println("Employee List: \nHit 1 to add an employee.  \nHit 2 to update
   an employee\n"
90                     + "Hit 3 to remove an employee.  \nHit 4 to display all employees.\n"
91                     + "Hit 5 to quit.");
92             int get = scan.nextInt();
93             switch (get) {
94             case 1:
95                 empMen.add(empMap, gradeMap);
96                 break;
97             case 2:
98                 empMen.update(empMap, gradeMap);
99                 break;
100            case 3:
101                empMen.remove(empMap, gradeMap);
102                break;
103            case 4:
104                empMen.display(gradeMap);
105                break;
106            case 5:
107                try {
```

```
108                     save(empMen.empMap, empMen.gradeMap);
109                     scan.close();
110                 } catch (FileNotFoundException e) {
111                     // TODO Auto-generated catch block
112                     e.printStackTrace();
113                 } catch (IOException e) {
114                     // TODO Auto-generated catch block
115                     e.printStackTrace();
116                 }
117                 quit  =  true;
118                 break;
119             default:
120                 System.out.println("Invalid choice");
121                 break;
122             }// End Switch
123         }// End While loop
124 
125     }
126     /**
127      * Add employee to employee map and grade map
128      * @param em Employee map
129      * @param gm Grade map
130      */
131     public void add(Map<Integer, Employee> em, Map<Employee, Integer> gm) {
132         Scanner scan = new Scanner(System.in);
133 
134         System.out.println("Please enter ID");
135         int empID = scan.nextInt();
136 
137                 System.out.println("Please enter last name");
138                 String lName = scan.next();
139                 System.out.println("Please enter first name");
140                 String fName = scan.next();
141                 System.out.println("Please enter performance");
142                 int performance = scan.nextInt();
143 
144                 Employee hold = new Employee(fName, lName, empID);
145 
146                 em.put(empID, hold);
147                 gm.put(hold, performance);
148     }
149     /**
150      * Updates employee in grade and employee map
151      * @param em employee map
152      * @param gm grade map
153      */
154     public void update(Map<Integer, Employee> em, Map<Employee, Integer> gm) {
155         Scanner scan = new Scanner(System.in);
156         System.out.println("Enter the ID of the employee you wish to modify");
157         int empId = scan.nextInt();
158         if (!em.containsKey(empId))
159             System.out.println("No such employee exist");
160         else {
161             System.out.println("Please enter new performance");
162             int performance = scan.nextInt();
163             Employee hold = em.get(performance);
164             gm.replace(hold, performance);
```

```
165         }
166     }
167     /**
168      * removes an employee from the grade and employee map
169      * @param em employee map
170      * @param gm grade map
171      */
172     public void remove(Map<Integer, Employee> em, Map<Employee, Integer> gm) {
173         Scanner scan = new Scanner(System.in);
174         System.out.println("Please enter the ID you wish to remove");
175         int empID = scan.nextInt();
176         Employee remover = em.get(empID);
177         if(gm.containsKey(remover)) {
178             System.out.println("Found ID to removed");
179             gm.remove(remover);
180             em.remove(empID);
181         }
182         else {
183             System.out.println("Person not found");
184         }
185     }
186     /**
187      * Sort and display grades
188      * @param gm grade map
189      */
190     public void display(Map<Employee, Integer> gm) {
191         Set<Employee> empList = gm.keySet();
192         ArrayList<Employee> arrListEmp = new ArrayList<Employee>();
193         Iterator<Employee> iterArr = gm.keySet().iterator();
194         while(iterArr.hasNext())
195         {
196             Employee e = iterArr.next();
197             arrListEmp.add(e);
198         }
199         Collections.sort(arrListEmp, (o1, o2) -> o1.compareTo(o2));
200         for(Employee e: arrListEmp)
201             System.out.println(e.toString() + " " + gm.get(e));
202     }
203 }
204
```