Progetto Gestione Dipendente

Il progetto si prefigge l'obbiettivo di gestire un Database di un azienda, i cui contenuti saranno le schede Utente dei vari dipendenti con annesse e-mail aziendale ,username ,password e ruolo in azienda;

Il gruppo per risolvere le richieste sostenute dal cliente ha suddiviso in 4 package il progetto:

-Model: in cui sono contenute le classi di archiviazione dati sensibili degli utenti;

-Controller: in cui vi è un menù per la gestione del database degli utenti;

gli utenti potranno essere inseriti nel database, modificati, o eliminati in base alle esigenze del cliente.

-View: in cui sarà gestita tutta la parte I/O del progetto, e della logica del progetto;

-Utility: in cui sarà gestita la gestione del database e l'encrypting della Password utente.

Per la gestione dell' I/O è stato usata una classe Singleton GestoreIO che comprende tutte le richieste di input dall'utente, e la gestione del popolamento dei dati sensibili.

Classe View

```
 1 package view;
 2 import java.util.*;
 3
 4 import model.Account;
 5 import model.Dipendente;
 6 import model.Ruolo;
 7 import utility.Crittografia;
 8 public final class GestoreIO implements iGestoreIO{
 9     private static GestoreIO instance=null;
10     private GestoreIO() {}
11     public static GestoreIO getInstance() {
17     public String leggiStringa(String suggerimento) {
22     public Integer leggiIntero(String suggerimento) {
34     public Double leggiDouble(String suggerimento) {
46     public void mascheraInserimento(Dipendente dip) {
60     public void mascheraModifica(Dipendente dip) {
97     public void stampaScheda(ArrayList<Dipendente> dipendente) {
111    public void stampaSchedaRuolo(ArrayList<Dipendente>dipendente,String ruolo){
132    public void stampaCSV(ArrayList<Dipendente> dipendente) {
137    public String menu(){
140
141 }
142
```

È stato deciso dal team di inizializzare la classe "GestoreIO" come "final" per evitare di poter apportare modifiche o di poter ereditare questa classe all'esterno.

È stata inoltre inserita un interfaccia a gestione della classe chiamata “iGestoreIO”

Interfaccia della Classe View

```java
package view;
import java.util.ArrayList;
import model.Dipendente;

public interface iGestoreIO {
    public String leggiStringa(String suggerimento);
    public Integer leggiIntero(String suggerimento);
    public Double leggiDouble(String suggerimento);
    public void mascheraInserimento(Dipendente dip);
    public void mascheraModifica(Dipendente dip);
    public void stampaScheda(ArrayList<Dipendente>dipendente);
    public void stampaCSV(ArrayList<Dipendente>dipendente);
}
```

L’interfaccia iGestoreIO contiene tutti i metodi principali di inserimento dati in input, popolamento dei dati sensibili e relative stampe, per quanto riguarda la stampa delle schede degli utenti è stata criptata la password dell’utente consentendo di visualizzare solo il primo e l’ultimo carattere.

```java
    @Override
    public void stampaScheda(ArrayList<Dipendente> dipendente) {
        for(int i=0;i<dipendente.size();i++) {
            System.out.println("Scheda "+(i+1));
            System.out.print("Dipendente:\n\t-Nome: "+dipendente.get(i).getNome()+
                    "\n\t-Cognome: "+dipendente.get(i).getCognome()+
                    "\n\t-Codice Fiscale: "+dipendente.get(i).getCodiceFiscale()+
                    "\nAccount:\n\t-Username: "+dipendente.get(i).getAccount().getUsername()+
                    "\n\t-Email: "+dipendente.get(i).getAccount().getEmail()+
                    "\n\t-Password: "+dipendente.get(i).getAccount().getPassword().charAt(0));
                    for(int j=1;j<=(dipendente.get(i).getAccount().getPassword().length()-1);j++)
                        System.out.print("*");
    System.out.println(dipendente.get(i).getAccount().getPassword().charAt((dipendente.get(i).getAccount().getPassword().length()-1))+
            "\nRuolo: "+dipendente.get(i).getAccount().getRuolo());
        }
    }
```

Questo sistema stamperà in formato scheda i vari utenti e consentirà la protezione della password come mostrato nell’esempio:

```
Scheda 1
Dipendente:
        -Nome: Vittorio
        -Cognome: Carannante
        -Codice Fiscale: dsds43
Account:
        -Username: vitt
        -Email: vi@mail.it
        -Password: 0*******C
Ruolo: admin
```

Classe Avvio

La classe avvio del package controller è utilizzata per gestire l'intera logica del progetto e contiene il menù per la gestione dell'intero progetto

```java

public class Avvio {

    public static void main(String[] args) {
        Dipendente d = null;

        boolean flag = true;

        String cf;

        do {

            switch(GestoreIO.getInstance().leggiIntero(GestoreIO.getInstance().menu())) {
            case 1:
                d = new Dipendente();
                GestoreIO.getInstance().mascheraInserimento(d);
                Crud.getInstance().inserisci(d);
                break;
            case 2:
                if(!(Crud.getInstance().leggi().isEmpty())) {
                    cf = GestoreIO.getInstance().leggiStringa("Inserisci il codice fiscale del dipendente da modificare: ");
                    if(Crud.getInstance().ricerca(cf) != null) {
                        System.out.println(Crud.getInstance().ricerca(cf));
                        GestoreIO.getInstance().mascheraModifica(Crud.getInstance().leggi(Crud.getInstance().leggi().indexOf(Crud.getInstance().ricerca(cf))));
                    } else {
                        System.out.println("Non Ã¨ stato trovato alcun dipendente con questo codice fiscale");
                    }
                } else {
                    System.out.println("Nessun Dipendente prensente nel DB");
                }
                break;
            case 3:
                if(!(Crud.getInstance().leggi().isEmpty())) {
                    cf = GestoreIO.getInstance().leggiStringa("Inserisci il codice fiscale del dipendente da cancellare: ");
                    if(Crud.getInstance().ricerca(cf) != null) {
                        System.out.println(Crud.getInstance().ricerca(cf));
                        Crud.getInstance().cancella(Crud.getInstance().leggi().indexOf(Crud.getInstance().ricerca(cf)));
                    } else {
                        System.out.println("Non Ã¨ stato trovato alcun dipendente con questo codice fiscale");
                    }
                } else {
                    System.out.println("Nessun Dipendente prensente nel DB");
```

```java
            case 4:
                if(!(Crud.getInstance().leggi().isEmpty())) {
                    cf = GestoreIO.getInstance().leggiStringa("Inserisci il codice fiscale del Dipendente da ricercare: ");
                    if(Crud.getInstance().ricerca(cf) != null) {
                        System.out.println(Crud.getInstance().ricerca(cf));
                    } else {
                        System.out.println("Non Ã¨ stato trovato alcun Dipendente con questo codice fiscale");
                    }
                } else {
                    System.out.println("Nessun Dipendente prensente nel DB");
                }
                break;
            case 5:
                if(!(Crud.getInstance().leggi().isEmpty())) {
                    GestoreIO.getInstance().stampaScheda(Crud.getInstance().leggi());
                } else {
                    System.out.println("Nessun Dipendente prensente nel DB");
                }
                break;
            case 6:
                if(!(Crud.getInstance().leggi().isEmpty())) {
                    GestoreIO.getInstance().stampaCSV(Crud.getInstance().leggi());
                } else {
                    System.out.println("Nessun Dipendente prensente nel DB");
                }
                break;
            case 7:
                if(!(Crud.getInstance().leggi().isEmpty())) {
                String ruolo = GestoreIO.getInstance().leggiStringa("Ricerca Dipendenti di ruolo: ");
                GestoreIO.getInstance().stampaSchedaRuolo(Crud.getInstance().leggi(), ruolo);
                } else {
                    System.out.println("Nessun Dipendente prensente nel DB");
                }
                break;
            case 8:
                flag = false;
                break;
            default:
                System.out.println("Inserire un numero compreso nel MenÃ¹");
                break;
            }

```

Come mostrato dalle immagini abbiamo una serie di scelte ricevute dal metodo della vista menù

```
138@ public String menu(){
139     return "Menù:\n1)Inserimento\n2)Modifica\n3)Cancella\n4)Ricerca per codice fiscale\n5)Stampa Scheda\n6)Stampa CSV\n7)Stampa per ruolo in formato Scheda\n8)Exit\n";
140 }
```

un menù composto da 7 scelte più la scelta di chiusura del programma .

Le classi model

Le classi model mostrano una panoramica dei dati sensibili degli utenti che si andranno ad inserire nel database, Le classi model create sono Dipendente,Account e Ruolo pur non usufruendo dell'ereditarietà sono messi in relazione grazie alla creazione dell'attributo interno alla classe Dipendente "account"

```
public class Dipendente {

    private String nome;
    private String cognome;
    private String codiceFiscale;
    private Double stipendio;
    private Account account;
```

```
public class Account {

    private String username;
    private String password;
    private String email;
    private Ruolo ruolo;
```

E grazie all' attributo "ruolo" della classe account abbiamo una relazione anche con la terza classe model(Ruolo)

```
public class Ruolo {

    private String ruolo;
```

Naturalmente con tutti i vari attributi inizializzati con private ogni classe model ha dei metodi "Setter & Getter" per poter inserire e in caso modificare i vari parametri.

Le classi Utility

All'interno del package utility sono inserite le classi per la creazione randomica della password e per la gestione del ArrayList che conterrà tutti gli utenti.

Per la creazione della password randomica è stato utilizzato la classe "Random" della libreria "java.util" che ogni volta che sarà chiamato nextInt restituirà un numero randomico come mostrato dall'esempio dell'utilizzo in caso della password

```
Random r =new Random();
do {
char casuale= (char) (r.nextInt(123));
```

All'interno delle parentesi limitiamo il rande di numeri con fattori 0(incluso)-123(escluso).

Per poter inserire questi numeri randomici all'interno di una stringa utilizziamo una variabile di supporto di tipo carattere "casuale" e grazie al casting a char prendendo determinati range di numeri nella tabella di codici ascii andremo a inserire nella nostra stringa la password

```
do {
char casuale= (char) (r.nextInt(123));
if((casuale>=48 && casuale<=57)||(casuale>=65&& casuale<=90)||(casuale>=97&& casuale<=122)){
    password=password+casuale;

}

}while(password.length()<8);
```

Il team ha deciso che le password generate non dovranno contenere più di 8 caratteri.

Nella classe Crud è presente la dichiarazione e inizializzazione dell'ArrayList dei Dipendenti e i relativi metodi della gestione di inserimento(add),cancellazione(remove), richiamo di singoli dipendenti(get(index)) o dell'intero ArrayList(ritorno del puntatore all'ArrayList) e in fine un metodo di ricerca codice fiscale all'interno dell'ArrayList che ci permette di ricercare un singolo dipendente.

```
41    @Override
42    public Dipendente ricerca(String cf) {
43
44        for(int i=0;i<dipendente.size();i++) {
45            if(dipendente.get(i).getCodiceFiscale().equalsIgnoreCase(cf)) {
46                return dipendente.get(i);
47            }
48        }
49        return null;
50    }
```

La ricerca restituirà il dipendente trovato, oppure null se non avrà trovato alcun dipendente.