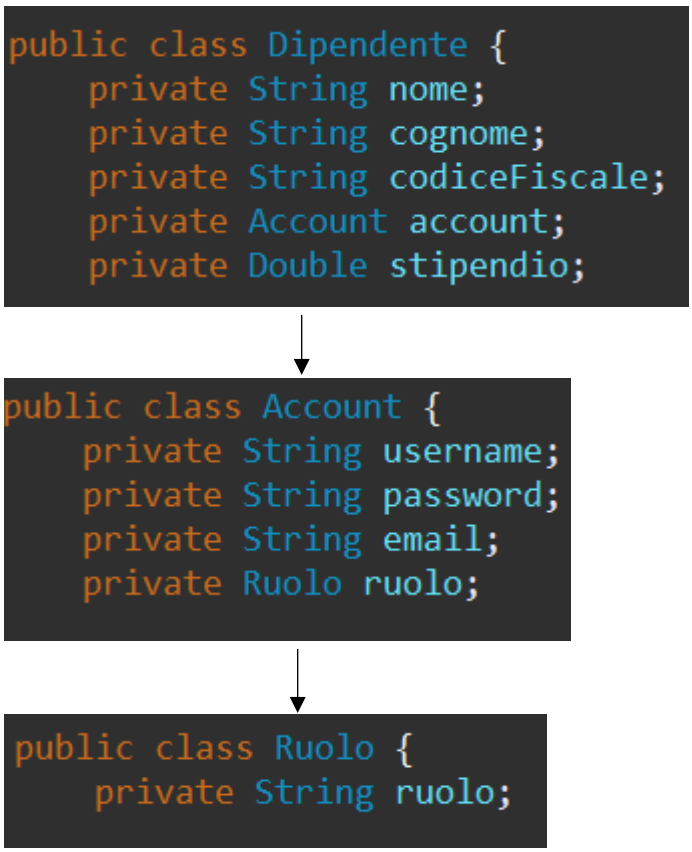
Programma gestione Database con ArrayList e relazione tra oggetti

Il nostro programma permette all'utente di creare un database di Dipendenti, grazie all'utilizzo di funzionalità per inserire dati con parametri preimpostati.
Oltre che all'inserimento il programma permette di modificare, cancellare, ricercare e visualizzare i dati contenuti nel database.
I dati che l'utente potrà inserire sono: Nome, Cognome, Codice Fiscale, Stipendio, Username, Password, e-mail, Ruolo.
Questi dati appartengono a tre classi diverse: Dipendente, Account, Ruolo.

Le classi Model

Le Classi Model contengono gli attributi che l'utente potrà inserire.
La Classe Dipendente contiene come dati: Nome, Cognome, Codice Fiscale, Stipendio e Account.
La Classe Account contiene come dati: Username, password, e-mail, Ruolo.
La Classe Ruolo contiene come dati: Ruolo.
Possiamo notare che la Classe Dipendente come anche la Classe Account hanno una relazione tra oggetti.
La Classe Account ha una relazione con la classe Ruolo, all'interno di Account abbiamo infatti una variabile di tipo Ruolo che mette in collegamento le due Classi senza l'utilizzo dell'ereditarietà.
Poi abbiamo una relazione tra Dipendente e Account che collega le classi allo stesso modo di Account e Ruolo, siccome Ruolo è collegato ad Account, Dipendente che conterrà Account, conterrà anche la classe Ruolo.

Esempio di messa in relazione



Tramite questo esempio possiamo notare come le classi siano messe in relazione attraverso l'utilizzo degli attributi del tipo della classe di appartenenza.

Le classi del package utility

Nel package Utility oltre alla Classe DipendenteCRUD e alla Classe Crittografia, troviamo l'interfaccia IDipendenteCRUD che contiene i metodi non implementati che saranno implementati nella classe DipendenteCRUD attraverso la keyword implements.
La Classe CRUD contiene le operazioni necessarie alla manipolazione dei dati:
Il repository dei dati sarà un'ArrayList.

1)Inserimento: Attraverso l'utilizzo del metodo "inserisci(Dipendente d)", richiamiamo il metodo "add" dell'ArrayList per popolare il nostro database di un elemento alla volta.

```
public boolean inserisci(Dipendente d) {
    listadip.add(d);
    return true;
```

2)Rimuovi: Attraverso l'utilizzo del metodo "cancella(int indice)", richiamiamo il metodo "remove" dell'ArrayList per rimuovere dal nostro database l'elemento che corrisponde all'indice passato in argomento.

```
public boolean cancella(int indice) {
    listadip.remove(indice);
    return true;
```

3)Lettura: Attraverso l'utilizzo del metodo "leggi(int indice)", richiamiamo il metodo "get(indice)" dell'ArrayList per avere in output l'elemento del nostro database che corrisponde all'indice passato in argomento.

```
@Override
public Dipendente leggi(int indice) {
    return listadip.get(indice);
}
```

Abbiamo implementato due metodi di ricerca tramite codice fiscale, di cui uno ci ritornerà il Dipendente stesso e l'altro l'indice della posizione del Dipendente nell'ArrayList.

```
public Dipendente ricercaCF() {
    String codFiscale=GestoreIO.getInstance().leggiStringa("Inserire CF: ");
    for(int i=0; i<listadip.size(); i++) {
        if(listadip.get(i).getCodiceFiscale().equalsIgnoreCase(codFiscale)) {
            return listadip.get(i);
        }
    }
    return null;
}

public Integer ricercaCFindice() {
    String codFiscale=GestoreIO.getInstance().leggiStringa("Inserire CF: ");
    for(Integer i=0; i<listadip.size(); i++) {
        if(listadip.get(i).getCodiceFiscale().equalsIgnoreCase(codFiscale)) {
            return i;
        }
    }
    return null;
```

Avvalendoci di questi due metodi possiamo poi compiere le operazioni di ricerca che di conseguenza ci permetteranno di effettuare le operazioni di modifica, cancella e di stampa che chiameremo in altri metodi nell'Avvio.

Nella Classe Crittografia implementeremo un singolo metodo con un ritorno di tipo stringa che sarà essa stessa una serie di caratteri generati casualmente che formerà la password che utilizzeremo nell'inserimento.

```
public String getPassword() {
    String password="";
    Random r=new Random();
    do {
        char lettera= (char) (r.nextInt(123));
        if((lettera>=48 && lettera<=52 )|| (lettera>=65 && lettera<=90) || (lettera>=97 && lettera <=122)) {
            password=password+lettera;
        }
    }while(password.length()<8);
    return password;
}
```

Le Classi del package view

Nel package view oltre alla Classe GestoreIO troviamo l'interfaccia IGestoreIO che contiene i metodi non implementati che saranno implementati nella classe GestoreIO attraverso la keyword implements.

La Classe GestoreIO contiene i metodi per inserire i valori in input, e visualizzare le stampe in output.

Tra i metodi della ClasseGestoreIO troviamo il metodo per la maschera d'inserimento e per la maschera di modifica.

```
public void mascheraInserimento(Dipendente d) {
    d.setNome(leggiStringa("Nome: "));
    d.setCognome(leggiStringa("Cognome: "));
    d.setCodiceFiscale(leggiStringa("CF: "));
    d.setStipendio(leggiDouble("Stipendio: "));
    d.setAccount(new Account());
    d.getAccount().setEmail(leggiStringa("E-Mail: "));
    d.getAccount().setUsername(leggiStringa("Username: "));
    d.getAccount().setPassword(Crittografia.getInstance().password());
    d.getAccount().setRuolo(new Ruolo());
    do {
        d.getAccount().getRuolo().setRuolo(leggiStringa("Ruolo: (guest) / (admin) "));
    }while(!d.getAccount().getRuolo().getRuolo().equalsIgnoreCase("guest")
        && !d.getAccount().getRuolo().getRuolo().equalsIgnoreCase("admin"));
}
```

La maschera d'inserimento ci permette di valorizzare ogni dato di tipo Dipendente attraverso l'utilizzo dei Setter. Come possiamo vedere nell'immagine sopra, inseriamo nell'attributo account, l'oggetto Account, così da poterne richiamare gli attributi e impostarli. La stessa procedura viene applicata per l'oggetto Ruolo che viene inserito nell'attributo ruolo di Account.

Per quanto riguarda l'inserimento della password, essa sarà impostata in base al metodo che richiameremo dalla Classe Crittografia, che ritornerà una stringa di caratteri casuali.

```
public void mascheraModifica(Dipendente d) {
s=leggiStringa("Username: ["+ d.getAccount().getUsername()+"]:");
if(s!="") {
    d.getAccount().setUsername(s);
}
s=leggiStringa("Password: ["+ d.getAccount().getPassword()+"]:\nVuoi generare una nuova password? (si) / (no)");
if (s.equalsIgnoreCase("si")) {
    d.getAccount().setPassword(Crittografia.getInstance().password());
}
do {
    s=leggiStringa("Ruolo: ["+ d.getAccount().getRuolo()+"]:\n(admin) / (guest)");
}while(!s.equalsIgnoreCase("admin") && !s.equalsIgnoreCase("guest"));
d.getAccount().getRuolo().setRuolo(s);
```

La maschera di modifica ci permette di modificare i vari attributi del dipendente ricercato attraverso il metodo di ricerca, mostrandoci i dati già presenti e dandoci la possibilità di alterarli scrivendo in input da console, oppure di lasciarli invariati premendo invio. Per la password e il ruolo, l'utente dovrà necessariamente scegliere se desidera generare una nuova password e se inserire un ruolo diverso da quello attuale.
Entrambi i metodi applicano un ritorno di tipo void, perché l'argomento passato è di tipo reference, ovvero contiene l'indirizzo di riferimento dell'oggetto richiamato, di conseguenza andremo ad applicare i settaggi e le modifiche sull'oggetto stesso.

Il Package controller

Il package controller contiene solo la Classe Avvio.
La Classe Avvio è responsabile dell'intera esecuzione del programma; come tale contiene il metodo main() statico in quanto può essere chiamato solo dalla Classe Avvio stessa.
Il metodo main() funge da starter e chiamerà tutti gli altri metodi necessari alle operazioni del programma.
Per la gestione del richiamo dei metodi utilizziamo la struttura switch-case, all'interno di un ciclo do-while.

```
public static void main(String[] args) {
    Dipendente d=null;
    boolean esci=false;
    do {
        GestoreIO.getInstance().menu();
        switch(GestoreIO.getInstance().leggiIntero("Inserisci quale operazione eseguire ")) {
        case 1:
            d=new Dipendente();
            GestoreIO.getInstance().mascheraInserimento(d);
            DipendenteCRUD.getInstance().inserisci(d);
            break;
        case 2:
            GestoreIO.getInstance().mascheraModifica(DipendenteCRUD.getInstance().ricercaCF());
            break;
        }
    }while(!esci);
```

Nel momento in cui il main verrà avviato, si avrà in output un menù numerato composto da tutte le operazioni disponibili, l'utente dovrà quindi inserire il numero dell'operazione che intende compiere, il numero dato in input chiamerà quindi il metodo corrispondente all'operazione scelta.
Terminata un'operazione il programma non cesserà, poiché la condizione verificata nel do-while sarà regolata da una variabile booleana che inizializzeremo a false e finche sarà tale il ciclo si ripetererà.
Il modo con cui l'utente può uscire dal programma è digitare il numero corrispondente all'operazione di uscita, che imposterà la variabile booleana a true, così che la condizione del ciclo do-while non sarà più verificata.