OpenTraining Base

Progetto Ingegneria del Software 9 CFU
OpenTraining Base
23/11/2017

Carmelo Leonardi
Matr. W83000088

C.d.L di primo livello Informatica
Dipartimento di Matematica e Informatica
Università degli Studi di Catania

Indice

Carmelo Leonardi

Requisiti funzionali

Legenda: classi, attributi, metodi

OpenTraining Base è il software per un azienda che eroga corsi di formazione. Il sistema sistema in questione permette di registrare, modificare ed eliminare i dati relativi ai corsi, i docenti che vi insegnano e gli studenti che vi si iscrivono.
Un corso di formazione ha un id, un nome, una descrizione, una categoria, numero di ore, la data di inizio e la data di fine ed un costo. Ogni corso è strutturato in moduli. Ogni modulo ha un nome, un numero di ore ed un ssd (settore scientifico disciplinare).
Dei docenti registrati nel sistema sono noti, l'id, dati anagrafici (nome, cognome, codice fiscale, data di nascita), email, qualifica. Per uno studente: id, dati anagrafici, dati reddituali. La dichiarazione del reddito per lo studente è facoltativa. Lo studente può iscriversi solo ad un corso. Il docente può insegnare uno o più moduli e può coordinare un ed un solo corso.

Progettazione

Per realizzare OpenTrainig Base sono state usate 16 classi:

- Corso: classe che contiene le informazioni del corso e permette di aggiungere e rimuovere i moduli. Questa classe rappresenta l'elemento composto per il design pattern Composite;
- Modulo: classe che contiene informazioni sui moduli. Questa classe ha rappresenta l'elemento semplice per il design pattern Composite;
- CorsoFormazione: classe astratta che accomuna i metodi per Corso e Modulo e rappresenta gli elementi semplici e composti;
- CreatorCorsoFormazione: classe a cui è demandata la responsabilità di creare le istanze per Corso e Modulo;
- Anagrafica: per rappresentare i dati anagrafici dei docenti e studenti;
- Persona: classe astratta che accomuna i metodi e gli attributi per studenti e docenti;
- Docente: classe derivata da Persona, che contiene i dati dei docenti;
- Studente: classe derivata da Pesona che contiene i dati degli studenti
- CreatorPersona: interfaccia in cui vengono definiti i comportamenti per creare istanze di Studente e Docente. Si specializza il CreatorDocente e Creator Studente.
- CreatorDocente: classe a cui è demandata la responsabilità di creare oggetti di tipo Docente;
- CreatorStudente: classe a cui è demandata la responsabilità di creare oggetti di tipo Studente;
- Archivio: classe di tipo Singleton per contenere i dati deli studenti e dei docenti e fornirli all'interfaccia principale per consentirne la consultazione e manipolazione
- Reddito: interfaccia che implementai comportamenti per i dati reddituali dello studente
- DatiReddito: implementazione della classe reddito. Reddito e Dati reddito implemetano un comportamento opzionale per lo studente;
- OpenTrainingBase: interfaccia principale che fonisce un punto di accesso globale al sistema siffatto.
- Data: classe composta da tre interi (giorno, mese, anno).

Carmelo Leonardi

Il linguaggio di programmazione utilizzato è Java e come tools di sviluppo sono stati utilizzati IDE IntelliJ-Idea e ArgoUML per progettare i diagrammi UML.

Diagramma UML delle classi

Di seguito viene rappresentato il diagramma UML delle classi per avere una visione globale del sistema.

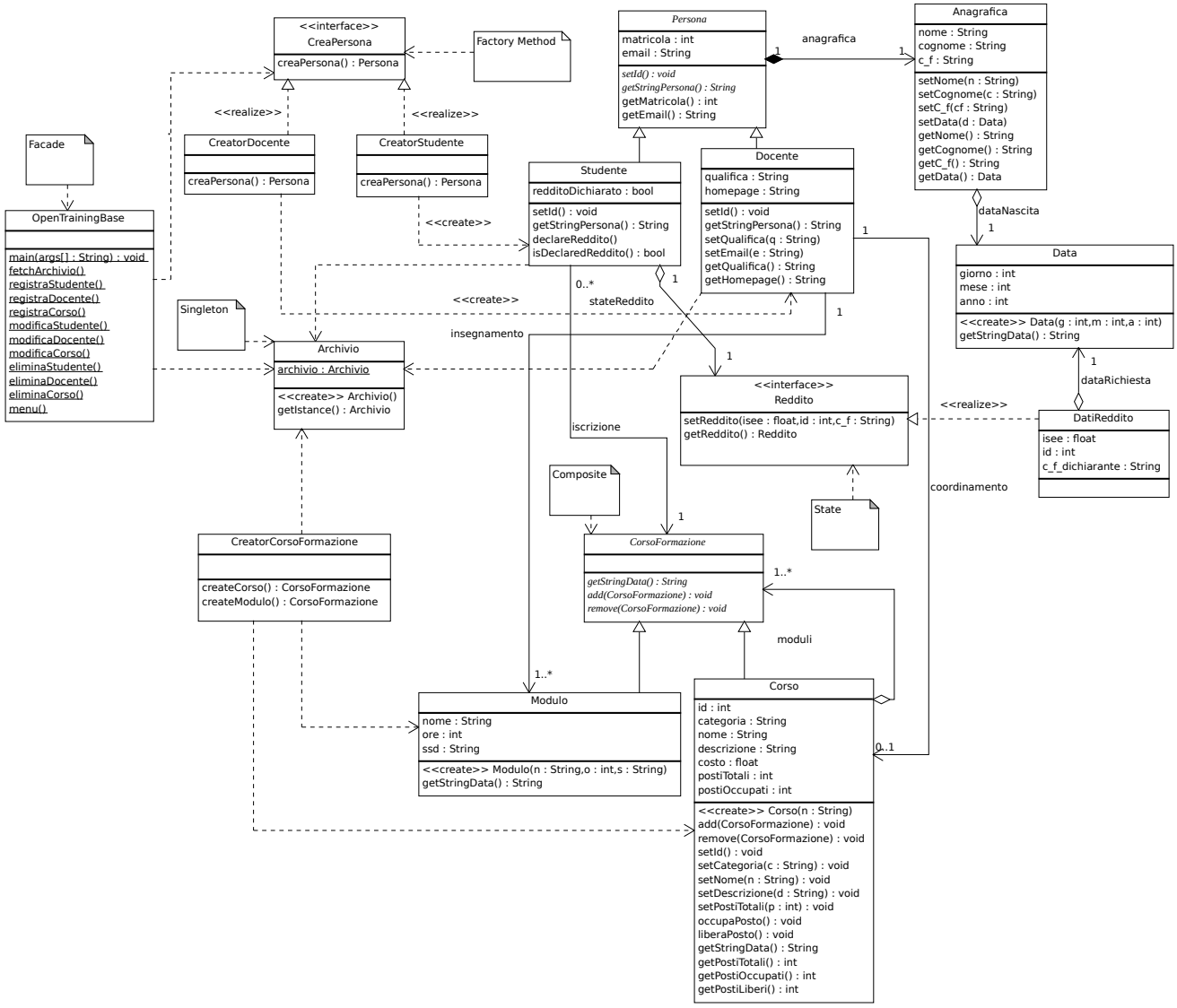
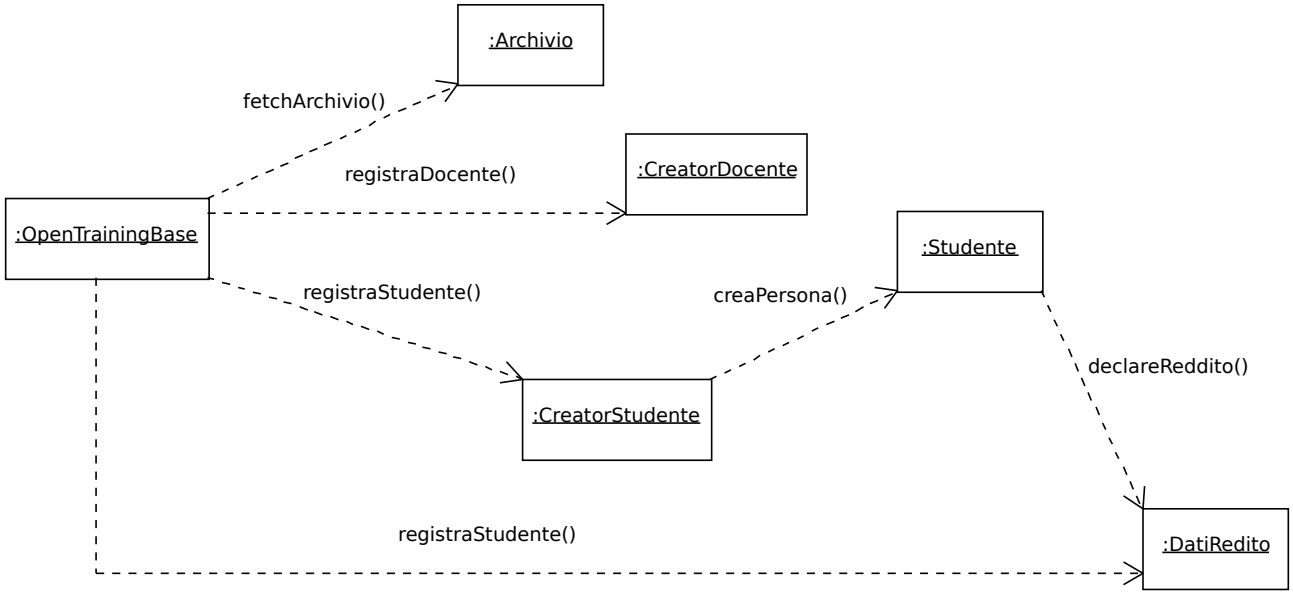


Diagramma di collaborazione degli oggetti

Il seguente diagramma illustra la collaborazione tra gli oggetti OpenTraining base che richiede l'istanza Archivio e le classi CreatorDocente, CreatorStudente, e DatiReddito.
OpenTrainingBase reperisce l'istanza di Archivio contentente le strutture che memorizzano i dati dei corsi, studenti e docenti.
Per registrare un nuovo studente viene ricevuta l'istanza da parte di CretorStudente e creato lo studente possiamo memorizzare i dati reddituali



Design pattern

Sono stati usati i seguenti design pattern:

- **Factory method:** permette di delegare la creazione di un oggetto ad un altro oggetto. Tale design pattern è stato utilizzato per creare le istanze di Studente e Docente;
- **Singleton**: Permette di creare una sola istanza per una classe e fornisce un punto di accesso globale per tale istanza. E' stato utilizzato per creare l'archivio dati di Studente, Docente e Corsi;
- **Facade**: Permette di avere un punto di accesso globale per il sistema, mettendo a disposizione un interfaccia utente che interagisce con le altre interfacce. E' usato per la classe OpenTrainingBase
- **Composite**: permette di comporre ad albero oggetti semplici e non. Dispone di una classe di base Component che si specializza in Leaf per gli oggetti semplici e Composite per gli oggetti composti. E' stato usato per comporre i Corsi ed i Moduli

- **State**: permette ad un oggetto di variare il suo comportamento quando cambia un suo stato interno. Il design pattern indicato è stato usato per memorizzare i dati reddituali dello Studente qualora dichiara il reddito

Factory Method

Classe	Ruolo
Persona	Product
Studente	ConcretePoduct
Docente	ConcreteProduct
CreatorPersona	Creator
CreatorStudente	ConcreteCreator
CreatorDocente	ConcreteCreator
OpenTrainingBase	Client

Singleton

Classe	Ruolo
Archivio	Singleton
OpenTrainingBase	Client

Facade

Classe	Ruolo
OpenTrainingBase	Facade

Composite

Classe	Ruolo
CorsoFormazione	Component
Corso	Composite
Modulo	Leaf
OpenTrainingBase	Client

State

Classe	Ruolo
Cliente	Context/Client
Reddito	State
DatiReddito	ConcreteState
OpenTrainingBase	Client

Frammenti di codice

OpenTrainingBase.java

```
public class OpenTrainingBase { // Facade
  private static Archivio archivio = Archivio.getIstance();
  private static int  scelta;
  private static Scanner input = new Scanner(System.in);
  private static ArrayList<Studente> studenti;
  private static ArrayList<Docente> docenti;
  private static ArrayList<Corso> corsi;
  public static void main(String args[]) {
    fetchArchivio();
    menu();
  }
  public static void fetchArchivio() {
    studenti = archivio.getStudenti();
    docenti= archivio.getDocenti();
    corsi = archivio.getCorsi();
  }
  public static void printStudenti(){
    if(studenti.size()==0){
      System.out.println("Non ci sono studenti");
    }
    else{
      for(Studente s : studenti){
        //stampa dati studente
      }
    }
  }
  public static void printDocenti(){
    if(docenti.size()==0){
      System.out.println("Non ci sono studenti");
    }
    else{
      for(Docente d : docenti){
        //stampa dati docente
      }
    }
  }
```

```java
public static void printCorsi(){
  if(corsi.size()==0){
    System.out.println("Non ci sono corsi");
  }
  else{
    for(Corso c : corsi){
      //stampa dati corsi
    }
  }
}
public static void menu() {
  do {
    System.out.println("Digitare per la scelta\n");
    System.out.println("1) -- Visualizza archivio studenti");
    System.out.println("2) -- Visualizza archivio docenti");
    System.out.println("3) -- Visualizza archivio corsi");
    System.out.println("4) -- Inserisci studente");
    System.out.println("5) -- Inserisci docente");
    System.out.println("6) -- Inserisci corso");
    System.out.println("7) -- Iscrivi studente ad un corso");
    System.out.println("8) -- Coordinamento docente");
    System.out.println("9) -- Elimina studente");
    System.out.println("10) -- Elimina docente");
    System.out.println("11) -- Elimina corso");
    System.out.println("0) -- Per terminare");
    System.out.print("Inserisci scelta: ");
    scelta = Integer.parseInt(input.nextLine());
    int idC;
    switch(scelta){
      case 1:
        printStudenti();
        break;
      case 2:
        printDocenti();
        break;
      case 3:
        printCorsi();
        break;
      case 4:
        registraStudente();
        break;
      case 5:
        registraDocente();
        break;
      case 6:
        registraCorso();
        break;
      case 7:
        System.out.println("Id Studente");
        int idS = Integer.parseInt(input.nextLine());
        System.out.println("Id Corso");
        idC = Integer.parseInt(input.nextLine());
```

```
          iscrizione(idS,idC);
          break;
        case 8:
          System.out.println("Id Docente");
          int idD = Integer.parseInt(input.nextLine());
          System.out.println("Id Corso");
          idC = Integer.parseInt(input.nextLine());
          setCoordinamento(idD,idC);
          break;
        case 9:
          eliminaStudente();
          break;
        case 10:
          eliminaDocente();
          break;
        case 11:
          eliminaCorso();
          break;
        default:
          System.out.println("Ricarico menu");
      }
    }while(scelta !=0);
  }
  public static void registraStudente() {
    String nome, cognome, c_f, dataNascita, email;
    String risp;
    CreatorPersona cS = new CreatorStudente();
    Persona stud = cS.creaPersona();
    stud.setId();
    System.out.println("Nome: ");
    nome=input.nextLine();
    System.out.println("Cognome: ");
    cognome=input.nextLine();
    System.out.println("Codice fiscale: ");
    c_f=input.nextLine();
    System.out.println("Data Nascita: ");
    dataNascita=input.nextLine();
    System.out.println("email: ");
    email=input.nextLine();
    stud.setAnagrafica(nome,cognome,c_f,dataNascita);
    stud.setEmail(email);
    System.out.println("Dichiarare reddito : Sì/No");
    risp=input.nextLine();
    Studente s= (Studente)stud;
    if(risp=="Sì") {
      s.setRedditoDichiarato(true);
      s.declareReddito();
      float isee=0;
      int id;
      String cf, data;
      System.out.println("id: ");
      id=Integer.parseInt(input.nextLine());
```

```
      System.out.println("isee: ");
      isee=Float.parseFloat(input.nextLine());
      System.out.println("c_f dichiarante: ");
      cf=input.nextLine();
      System.out.println("Data Nascita: ");
      data=input.nextLine();
      s.getStateReddito().setReddito(isee,id,cf,data);
    }
    else
      s.setRedditoDichiarato(false);
    studenti.add(s);
  }
```

Archivio.java

```
public class Archivio {
  private static Archivio archivio;
  private ArrayList<Studente> studenti;
  private ArrayList<Docente> docenti;
  private ArrayList<Corso> corsi;
  private Archivio() {
    studenti = new ArrayList<Studente>();
    docenti = new ArrayList<Docente>();
    corsi = new ArrayList<Corso>();
  }
  private String[] splitData(String line) {
    String [] arrSplit = line.split(" - ");
    return arrSplit;
  }
  private void fetchData() {
   //lettura file di Studenti, Docenti, Corsi
  }
  public static Archivio getIstance() {
    if(archivio==null){
      archivio = new Archivio();
    }
    return archivio;
  }
  public void insertStudente(String [] record){
    CreatorPersona pc = new CreatorStudente();
    Persona stud = pc.creaPersona();
    stud.setDatiPersona(record);
    studenti.add((Studente)stud);
  }
  public ArrayList<Corso> getCorsi() {
    return corsi;
  }
  public ArrayList<Studente> getStudenti() {
    return studenti;
  }
  public ArrayList<Docente> getDocenti() {
    return docenti;
```

```
  }
}
```

Corso.java

```
public class Corso extends CorsoFormazione {
  private int id;
  private String categoria;
  private String nome;
  private String descrizione;
  private float costo;
  private int postiTotali;
  private int postiOccupati;
  private ArrayList<CorsoFormazione> moduli;
  public void setId() { /*id è autogenerato */}

  public Corso(String nome, String categoria) {
    this.nome=nome;
    this.categoria=categoria;
    moduli = new ArrayList<CorsoFormazione>();
  }
  public void add(CorsoFormazione c) {
    moduli.add(c);
  }
  public void remove( CorsoFormazione c) {
    moduli.remove(c);
  }
```

Modulo.java

```
public class Modulo extends CorsoFormazione {
  private String nome;
  private int ore;
  private String ssd;
  public Modulo(String n, int o, String s) {
  }
  @Override
  public String getStringData() {
  return "";
  }
  @Override
  public void add(CorsoFormazione c){}
  @Override
  public void remove( CorsoFormazione c){}
}
```

CreatorStudente.java

```
public class CreatorStudente implements CreatorPersona {
  @Override
  public Persona creaPersona() {
```

```
    return new Studente();
  }
}
```

CreatorDocente.java

```
public class CreatorDocente implements CreatorPersona {
  @Override
  public Persona creaPersona() {
    return new Docente();
  }
}
```

Studente.java

```
import com.sun.org.apache.regexp.internal.RE;
public class Studente extends Persona {
  private boolean redditoDichiarato;
  private CorsoFormazione iscrizione;
  private Reddito stateReddito;
  @Override
  public void setId() {
    //genera l'id dello studente
  }
  @Override
  public String getStringPersona() {
    return "";
  }
  @Override
  public void setDatiPersona(String []data){
    // assegna i data del vettore data agli attibuti della classe
  }
  public void setRedditoDichiarato(boolean f){
    redditoDichiarato=f;
  }
  public void declareReddito() {
    if(redditoDichiarato){
      stateReddito = new DatiReddito();
    }
  }
  public Reddito getStateReddito(){
    return stateReddito;
  }
  public boolean isDeclaredReddito() {
    return redditoDichiarato ? true : false;
  }
}
```

DatiReddito.java

```
import java.util.Vector;
public class DatiReddito implements Reddito {
  private float isee;
  private int id;
  private String c_f_dichiarante;
  public Data dataRichiesta;
  @Override
  public void setReddito(float isee, int id, String c_f, String d){
    this.isee=isee;
    this.id=id;
    c_f_dichiarante=c_f;
    String []arrData =d.split("/");
    dataRichiesta=new
Data(Integer.parseInt(arrData[0]),Integer.parseInt(arrData[3]),Integer.parseInt(ar
rData[2]));
  }
  @Override
  public Reddito getReddito(){
    return this;
  }
}
```