

Esempio di dependency inversion

L'esempio si articola in 3 diverse implementazioni di un programma che rappresenta studenti e i loro voti, e consente di salvarli su disco in un formato testuale.

Caso 1: implementazione banale

Senza alcuna inversione delle dipendenze.

Main.java

```
package it.uniud.poo.dependency_inversion.base;

import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;
import java.time.LocalDate;

public class Main {

    public static void main(String[] args) throws Exception {
        Anagrafica elenco = new Anagrafica();
        Studente s1 = new Studente( "francesca", "rossi", "12345" );
        elenco.iscrivi(s1, LocalDate.parse("2017-01-01"));
        Studente s2 = new Studente( "giovanni", "bianchi", "23456" );
        elenco.iscrivi(s2, LocalDate.parse("2013-11-27") );

        s1.registraVoto( LocalDate.now(), "P00", 23, false );
        s1.registraVoto( LocalDate.parse( "2019-08-25"), "Metodi", 23, false );
        s1.registraVoto( LocalDate.parse( "2018-08-05"), "ASD", 23, false );

        s2.registraVoto( LocalDate.now(), "P00", 30, true );
        s2.registraVoto( LocalDate.parse( "2019-08-15"), "Metodi", 18, false );
        s2.registraVoto( LocalDate.parse( "2018-08-15"), "ASD", 28, false );
        s2.registraVoto( LocalDate.parse( "2017-08-15"), "Programmazione", 28, false );

        elenco.salva();

        System.out.println(elenco.toJson());
        System.out.println(elenco.toJsonVERO());
    }
}
```

versione didattica

versione che si utilizzerebbe normalmente (potrebbe avere problemi con le iterazioni) se un campo lo dichiaro transient non verrà serializzato

Quando eseguo il `main()` ottengo:

```
{"elencoStudenti": [ {"nome": "francesca", "cognome": "rossi",  
"matricola": "12345", "elencoEsami": [ {"corso": "ASD", "voto": 23,  
"lode": false, "data": "2018-08-05"}, {"corso": "Metodi", "voto": 23,  
"lode": false, "data": "2019-08-25"}, {"corso": "P00", "voto": 23,  
"lode": false, "data": "2019-11-25"} ]}, {"nome": "giovanni",  
"cognome": "bianchi", "matricola": "23456", "elencoEsami": [ {"corso":  
"Programmazione", "voto": 28, "lode": false, "data":  
"2017-08-15"}, {"corso": "ASD", "voto": 28, "lode": false, "data":  
"2018-08-15"}, {"corso": "Metodi", "voto": 18, "lode": false, "data":  
"2019-08-15"}, {"corso": "P00", "voto": 30, "lode": true, "data":  
"2019-11-25"} ]} ]}
```

Anagrafica.java

La seguente classe memorizza i vari studenti e espone dei metodi per produrne una rappresentazione in JSON.

Il metodo `toJson()` lo fa in maniera artigianale, che è quanto useremo in questo esempio. Il metodo `toJsonVERO()` è quello che userei per davvero per risolvere questo problema, sfruttando la libreria Gson (che occorre importare nel progetto, ad esempio inserendola come dipendenza nel file `pom.xml` usato da maven).

```
package it.uniud.poo.dependency_inversion.base;  
  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.io.UnsupportedEncodingException;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
/**  
 * gestisci elenco di studenti  
 */  
public class Anagrafica {  
    List<Studente> elencoStudenti = new ArrayList<>();  
    /**  
     * iscriviti lo studente e aggiungilo all'anagrafica  
     * @param s2  
     * @param dataIscrizione
```

```

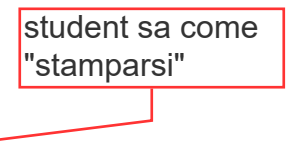
    */
    public void iscrivi( Studente s2, LocalDate dataIscrizione ) {
        elencoStudenti.add( s2 );
        s2.iscriviti( dataIscrizione );
    }

    public String toJson(){
        String tmp = elencoStudenti.stream()
            .map( st -> st.toJson() )
            .collect( Collectors.joining( "," ) );
        String ris = String.format("{ \"elencoStudenti\": [ %s ]}", tmp);
        return ris;
    }

    public void salva() throws FileNotFoundException,
        UnsupportedEncodingException {
        String res = toJson();
        PrintWriter writer = new PrintWriter("studenti.json", "UTF-8");
        writer.println( res );
        writer.close();
    }

    /**
     * @return una rappresentazione in json dell'istanza
     * fatta in maniera piu' efficiente e semplice
     */
    public String toJsonVERO() {
        GsonBuilder builder = new GsonBuilder();
        builder.setPrettyPrinting()
            .serializeNulls();
        // Una soluzione migliore sarebbe di evitare di usare l'exlude
        // ho fatto cosi' per motivi didattici
        Gson gson = builder.create();
        return gson.toJson( this );
    }
}

```



Studente.java

Nulla di particolare. Oltre ai metodi per manipolare gli oggetti di dominio c'è anche il metodo `toJson()` per produrre la versione serializzata dello studente e i suoi esami. Si noti che viene sfruttato un metodo `stream()` di `CarrieraUniversitaria`. Ho deciso di usare la programmazione funzionale invece di un iteratore.

```

package it.uniud.poo.dependency_inversion.base;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonElement;
import com.google.gson.JsonParser;
import com.google.gson.annotations.Expose;
import lombok.Getter;
import lombok.NonNull;
import lombok.Setter;

import java.time.LocalDate;
import java.util.stream.Collectors;

/**
 * rappresenta uno studente dell'universita'
 *
 * Stato astratto e concreto:
 * tupla di nome, cognome, matricola
 * la sua carriera (elenco esami fatti)
 */
public class Studente {
    @Getter
    private String nome;
    @Getter
    private String cognome;
    @Getter @Setter
    private String matricola;
    @Getter
    private CarrieraUniversitaria carriera;

    public Studente( @NonNull String nm, @NonNull String cogn,
        @NonNull String matr){
        nome = nm;
        cognome= cogn;
        carriera = new CarrieraUniversitaria();
        matricola = matr;
    }

    /**
     * iscrivi lo studente e setta la data di iscrizione
     * @param data data di iscrizione
     */
    public void iscriviti( @NonNull LocalDate data ){
        carriera.setDataIscrizione( data );
    }
}

```

```

    /**
     * registra un voto; lode solo se 30
     * @param data
     * @param corso
     * @param voto
     * @param lode
     */
    public void registraVoto( @NonNull LocalDate data,
        @NonNull String corso, int voto, boolean lode){
        if (lode && voto < 30){
            throw new RuntimeException( "non e' possibile "+
                "registrare la lode senza un 30" );
        }
        carriera.registraVoto(data, corso, voto, lode);
    }

    /**
     * @param dataLaurea
     * @param voto in 110
     */
    public void laureati( @NonNull LocalDate dataLaurea, int voto){
        carriera.registraLaurea(dataLaurea, voto);
    }

    /**
     * @return una rappresentazione in json dell'istanza
     */
    public String toJson(){
        String tmp = carriera.stream()
            .map( es -> es.toJson() )
            .collect( Collectors.joining( "," ) );
        String ris = String.format("{ \"nome\": \"%s\", \"cognome\": \"+
            \"%s\", \"matricola\": \"%s\", \"elencoEsami\": [\" +
                \"%s\"]\",
            nome, cognome, matricola, tmp);
        return ris;
    }
}

```

CarrieraUniversitaria.java

Niente di che: tutto come uno se l'aspetta.

```

package it.uniud.poo.dependency_inversion.base;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDate;
import java.util.*;
import java.util.stream.Stream;

/**
 * rappresenta l'elenco degli esami fatti e
 * la data di iscrizione e laurea (se fatta)
 * e se si e' laureato
 */
public class CarrieraUniversitaria {

    // INVARIANTE:
    // laureato==true sse dataLaurea != null
    @Setter
    private LocalDate dataIscrizione;
    @Setter
    private LocalDate dataLaurea;
    @Setter
    private boolean laureato;
    @Setter
    private int votoLaurea;
    @Setter
    private Set<Esame> esami = new TreeSet<>();

    /**
     * @param data
     * @param corso
     * @param voto
     * @param lode true solo se voto=30 TODO assicurarsi
     * che non crei duplicati (data, corso)
     */
    void registraVoto( LocalDate data, String corso, int voto, boolean lode ) {
        Esame es;
        es = new Esame( data, corso, voto, lode );
        esami.add( es );
    }

    public void registraLaurea( LocalDate dataLaurea, int voto ) {

```

```

        setLaureato( true );
        setDataLaurea( dataLaurea );
        setVotoLaurea( voto );
    }

    public Stream<Esame> stream() {
        return esami.stream();
    }
}

```

Esame.java

E infine l'ultimo tassello.

Deve implementare l'interfaccia Comparable in modo da poter essere inserito in un TreeSet.

```

package it.uniud.poo.dependency_inversion.base;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.Expose;

import java.time.LocalDate;

/**
 * rappresenta uno degli esami fatti
 * (data, nome corso, voto)
 * voto con/senza lode; se lode allora voto=30
 * IMMUTABILE
 */
public class Esame implements Comparable {
    private final LocalDate data;
    private final String corso;
    private final int voto;
    private final boolean lode;

    public Esame( LocalDate data, String corso, int voto, boolean lode ) {
        this.data = data;
        this.corso = corso;
        this.voto = voto;
        this.lode = lode;
    }

    @Override

```

```

public int compareTo( Object o ) {
    if (!(o instanceof Esame ))
        return -1;
    Esame es = (Esame) o;
    if (data.isBefore( es.data ))
        return -1;
    else if (data.isAfter( es.data ))
        return 1;
    else return (corso.compareTo( es.corso ));
}

/**
 * @return una rappresentazione in json dell'istanza
 */
public String toJson() {
    String dataRegistrazione = data.toString();
    String ris = String.format( "{\"corso\": \"%s\", \"voto\": \"+
    \"%d, \"lode\": %s, \"data\": \"%s\"}\",
        corso, voto, lode, dataRegistrazione );
    return ris;
}
}

```

Osservazioni

La prima osservazione è che ci sono delle responsabilità troppo estese in questi oggetti. Mi va bene che `Studente` “sappia” cosa fare per `laureati()` o per `registraVoto()`, dato che riguardano operazioni legate al dominio universitario. Non va tanto bene che ci siano operazioni come `toJson()` che di universitario non hanno nulla.

Stiamo violando il principio di Singola Responsabilità.

Inoltre, le operazioni di dominio vanno bene che stiano in oggetti del dominio, ma operazioni tecnologiche non devono stare lì. Meglio posizionarle in altri oggetti, dei servizi.

E questo porta alla seconda variante.

Seconda variante: creazione di un servizio

L’idea è di realizzare una classe che offre il servizio di generazione di JSON, e le classi di dominio useranno questo servizio.

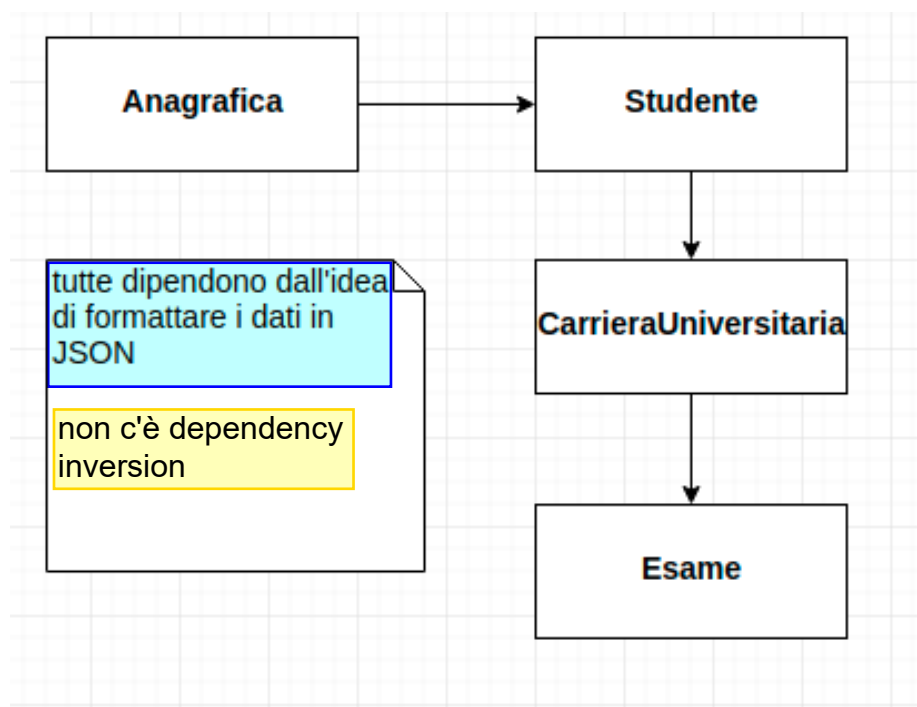


Figure 1: Diagramma delle classi per versione base

Main.java

Rispetto a prima c'è la generazione di un oggetto JSONizzatore a cui si fanno fare le operazioni di toJson() e salva().

```
package it.uniud.poo.dependency_inversion.servizio;

import java.time.LocalDate;

public class Main {

    public static void main(String[] args) throws Exception {
        Anagrafica elenco = new Anagrafica();

        Studente s1 = new Studente( "francesca", "rossi", "12345" );
        elenco.iscrivi(s1, LocalDate.parse("2017-01-01"));
        Studente s2 = new Studente( "giovanni", "bianchi", "23456" );
        elenco.iscrivi(s2, LocalDate.parse("2013-11-27" ));

        s1.registraVoto( LocalDate.now(), "P00", 23, false );
        s1.registraVoto( LocalDate.parse( "2019-08-25"), "Metodi", 23, false );
        s1.registraVoto( LocalDate.parse( "2018-08-05"), "ASD", 23, false );

        s2.registraVoto( LocalDate.now(), "P00", 30, true );
        s2.registraVoto( LocalDate.parse( "2019-08-15"), "Metodi", 18, false );
        s2.registraVoto( LocalDate.parse( "2018-08-15"), "ASD", 28, false );

        JSONizzatore js = JSONizzatore.getInstance();
        js.salva( elenco );
        System.out.println( js.toJson( elenco ));
    }
}
```

la capacità di salvare è delegata ad un'altra classe

Anagrafica.java

E quelle operazioni 'tecnologiche' vengono tolte dalle varie classi di dominio, che risultano quindi più pulite.

```
package it.uniud.poo.dependency_inversion.servizio;

import com.google.gson.annotations.Expose;
import lombok.Getter;
import lombok.Setter;
```

```

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * gestisci elenco di studenti
 */
public class Anagrafica {
    List<Studente> elenco = new ArrayList<>();

    /**
     * iscriviti lo studente e aggiungilo all'anagrafica
     * @param s2
     * @param dataIscrizione
     */
    public void iscriviti( Studente s2, LocalDate dataIscrizione ) {
        elenco.add( s2 );
        s2.iscriviti( dataIscrizione );
    }

    public Stream<Studente> stream() {
        return elenco.stream();
    }
}

```

Studente.java

```

package it.uniud.poo.dependency_inversion.servizio;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.Expose;
import lombok.Getter;
import lombok.NonNull;
import lombok.Setter;

import java.time.LocalDate;
import java.util.stream.Collectors;

/**
 * rappresenta uno studente dell'universita'

```

```

*
* Stato astratto e concreto:
* tupla di nome, cognome, matricola
* la sua carriera (elenco esami fatti)
*/
public class Studente {
    @Getter
    private String nome;
    @Getter
    private String cognome;
    @Getter @Setter
    private String matricola;
    @Getter
    private CarrieraUniversitaria carriera;

    public Studente( @NonNull String nm, @NonNull String cogn, @NonNull String matr){
        nome = nm;
        cognome= cogn;
        carriera = new CarrieraUniversitaria();
        matricola = matr;
    }

    /**
     * iscriviti lo studente e setta la data di iscrizione
     * @param data data di iscrizione
     */
    public void iscriviti( @NonNull LocalDate data ){
        carriera.setDataIscrizione( data );
    }

    /**
     * registra un voto; lode solo se 30
     * @param data
     * @param corso
     * @param voto
     * @param lode
     */
    public void registraVoto( @NonNull LocalDate data, @NonNull String corso, int voto, boolean lode ){
        if (lode && voto < 30){
            throw new RuntimeException( "non e' possibile registrare la lode senza un 30" );
        }
        carriera.registraVoto(data, corso, voto, lode);
    }

    /**
     * @param dataLaurea

```

```

        * @param voto in 110
        */
        public void laureati( @NonNull LocalDate dataLaurea, int voto){
            carriera.registraLaurea(dataLaurea, voto);
        }
    }
}

```

CarrieraUniversitaria.java

```

package it.uniud.poo.dependency_inversion.servizio;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import lombok.Data;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDate;
import java.util.Set;
import java.util.TreeSet;
import java.util.stream.Stream;

/**
 * rappresenta l'elenco degli esami fatti e
 * la data di iscrizione e laurea (se fatta)
 * e se si e' laureato
 */
public class CarrieraUniversitaria {

    // INVARIANTE:
    // laureato==true sse dataLaurea != null
    @Setter @Getter
    private LocalDate dataIscrizione;
    @Setter @Getter
    private LocalDate dataLaurea;
    @Setter @Getter
    private boolean laureato;
    @Setter @Getter
    private int votoLaurea;
    @Setter @Getter
    private Set<Esame> esami = new TreeSet<>();
}

```

```

    /**
     * @param data
     * @param corso
     * @param voto
     * @param lode true solo se voto=30 TODO assicurarsi che non crei duplicati (data, corso)
     */
    void registraVoto( LocalDate data, String corso, int voto, boolean lode ) {
        Esame es;
        es = new Esame( data, corso, voto, lode );
        esami.add( es );
    }

    public void registraLaurea( LocalDate dataLaurea, int voto ) {
        setLaureato( true );
        setDataLaurea( dataLaurea );
        setVotoLaurea( voto );
    }

    public Stream<Esame> stream() {
        return esami.stream();
    }
}

```

Esame.java

```

package it.uniud.poo.dependency_inversion.servizio;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.Expose;
import lombok.Getter;

import java.time.LocalDate;

/**
 * rappresenta uno degli esami fatti
 * (data, nome corso, voto)
 * voto con/senza lode; se lode allora voto=30
 * IMMUTABILE
 */
public class Esame implements Comparable {
    @Getter
    private final LocalDate data;
    @Getter
    private final String corso;
}

```

```

@Getter
private final int voto;
@Getter
private final boolean lode;

public Esame( LocalDate data, String corso, int voto, boolean lode ) {
    this.data = data;
    this.corso = corso;
    this.voto = voto;
    this.lode = lode;
}

@Override
public int compareTo( Object o ) {
    if (!(o instanceof Esame))
        return -1;
    Esame es = (Esame) o;
    if (data.isBefore( es.data ))
        return -1;
    else if (data.isAfter( es.data ))
        return 1;
    else return (corso.compareTo( es.corso ));
}
}

```

JSONizzatore.java

E infine la classe del servizio. Si noti come essa contenga metodi `toJson()` per i vari oggetti del dominio. Infatti è il JSONizzatore che “conosce” come produrre la versione JSON degli oggetti, non gli oggetti.

```

package it.uniud.poo.dependency_inversion.servizio;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import lombok.Getter;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.stream.Collectors;

```

```

/**
 * Servizio deputato alla generazione del json e al suo salvataggio e stampa
 */
public class JSONizzatore {

```

factory per non rendere pubblico il costruttore

```

    @Getter
    public final static JSONizzatore instance = new JSONizzatore();
    private Gson gson;

```

```

    public JSONizzatore() {
        GsonBuilder builder = new GsonBuilder();
        builder.setPrettyPrinting()
            .serializeNulls()
            .excludeFieldsWithoutExposeAnnotation();
        gson = builder.create();
    }

```

overload di toJson

```

/**
 * @param anagrafica
 * @return una rappresentazione in json dell'istanza
 */
    public String toJson( Anagrafica anagrafica ) {
        String tmp = anagrafica.stream()
            .map( st -> toJson( st ) )
            .collect( Collectors.joining( "," ) );
        String ris = String.format( "{\\"elencoStudenti\\": [ %s ]}", tmp );
        return ris;
    }

```

```


/**
 * @return una rappresentazione in json dell'istanza
 */
    public String toJson( Studente st){
        String ris = String.format("{\\"nome\\": \\"%s\\",\\"cognome\\": \\"%s\\", \\"matricola\\": \\"%s\\", \\"elencoEsami\\": [\" +
            " %s ]}",
            st.getNome(), st.getCognome(), st.getMatricola(),
            toJson( st.getCarriera() ));
        return ris;
    }

```

```

    public String toJson( CarrieraUniversitaria carriera){
        String tmp = carriera.stream()
            .map( es -> toJson( es ) )
            .collect( Collectors.joining( "," ) );
    }

```

```

        String ris = String.format("{\"dataIscrizione\": %s, \"dataLaurea\": %s,\" +
            \"votoLaurea\": %s, \"elencoEsami\": [%s]}\",
            carriera.getDataIscrizione(),
            carriera.getDataLaurea(),
            carriera.getVotoLaurea(),tmp);
        return ris;
    }

    private String toJson( Esame es ) {
        String dataRegistrazione = es.getData().toString();
        String ris = String.format( "{\"corso\": \"%s\", \"voto\": \"+
            \" %d, \"lode\": %s, \"data\": \"%s\"}\",
            es.getCorso(), es.getVoto(), es.isLode(), dataRegistrazione );
        return ris;
    }

    /**
     * Salva in un file la stringa json
     * @param anagrafica
     * @throws FileNotFoundException
     * @throws UnsupportedEncodingException
     */
    public void salva( Anagrafica anagrafica ) throws FileNotFoundException, UnsupportedEncod
        String res = toJson( anagrafica );
        PrintWriter writer = new PrintWriter( "studenti.json", "UTF-8" );
        writer.println( res );
        writer.close();
    }

    /**
     * @return una rappresentazione in json dell'istanza fatta in
     * maniera piu' efficiente e semplice
     */
    public String toJsonVERO( Anagrafica anagrafica ) {
        GsonBuilder builder = new GsonBuilder();
        builder.setPrettyPrinting()
            .serializeNulls();
        // Una soluzione migliore sarebbe di evitare di usare l'exclude
        // ho fatto cosi' per motivi didattici
        Gson gson = builder.create();
        return gson.toJson( anagrafica );
    }
}

```

Osservazioni

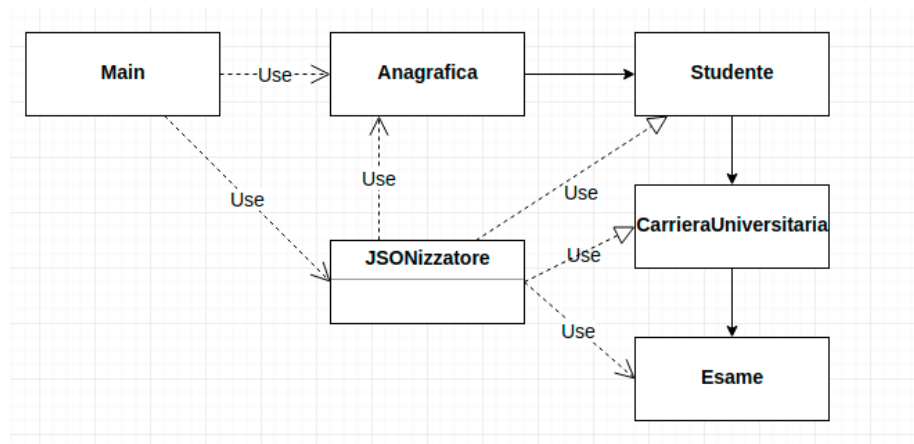


Figure 2: Diagramma delle classi per versione con servizio

A questo punto abbiamo incorporato le responsabilità relative alla trasformazione in JSON in una classe a parte, e dato modo al Main di poterla usare.

E il beneficio maggiore che ne traiamo è che le singole classi di dominio sono indipendenti dalla tecnologia. Dovessimo cambiare la tecnologia di persistenza (ad es. passare a un db relazionale), dovremmo cambiare solo parti della classe JSONizzatore.

Si notino le dipendenze tra le classi riportate nel diagramma:

- le classi di dominio tra loro sono dipendenti (ad es. **Studente** dipende da **CarrieraUniversitaria**, ma non da **Anagrafica**), ma questa dipendenza se i tipi di dati astratti sono ben realizzati è solo legata a dover conoscere il nome del tipo e le sue operazioni. Poco grave.
- La classe **JSONizzatore** dipende dalle classi di dominio, dato che ha bisogno nei suoi metodi `toJson()` di manipolare lo stato di ciascun oggetto (ad es. deve estrarre il `nome` di **Studente**).
- La classe **Main** è l'unica che dipende da **JSONizzatore**. In altri termini è l'unica che incapsula tale conoscenza, ed è l'unica che dovremo cambiare se volessimo cambiare tecnologia di persistenza e di serializzazione.

Terza versione: con l'interfaccia

In questa terza versione creiamo un'interfaccia per il servizio, anzi per i servizi.

Main.java

```
package it.uniud.poo.dependency_inversion.interfaccia;

import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;
import java.time.LocalDate;

public class Main {

    public static void main(String[] args) throws Exception {
        Serializzatore ser = preparaSerializzatore();
        GestorePersistenza gp = preparaPersistenza();
        faiIlLavoro( ser, gp );
    }

    private static void faiIlLavoro( Serializzatore ser,
        GestorePersistenza gp )
        throws FileNotFoundException, UnsupportedEncodingException {
        Anagrafica elenco = new Anagrafica();

        Studente s1 = new Studente( "francesca", "rossi", "12345" );
        elenco.iscrivi(s1, LocalDate.parse("2017-01-01"));
        Studente s2 = new Studente( "giovanni", "bianchi", "23456" );
        elenco.iscrivi(s2, LocalDate.parse("2013-11-27" ));

        s1.registraVoto( LocalDate.now(), "P00", 23, false );
        s1.registraVoto( LocalDate.parse( "2019-08-25"), "Metodi", 23, false );
        s1.registraVoto( LocalDate.parse( "2018-08-05"), "ASD", 23, false );

        s2.registraVoto( LocalDate.now(), "P00", 30, true );
        s2.registraVoto( LocalDate.parse( "2019-08-15"), "Metodi", 18, false );
        s2.registraVoto( LocalDate.parse( "2018-08-15"), "ASD", 28, false );
        gp.salva( elenco );
        System.out.println( ser.serializza( elenco ));
    }

    private static Serializzatore preparaSerializzatore() {
        Serializzatore ser = JSONizzatore.getInstance();
        Serializzatore ser2 = GeneratoreDiToString.getInstance();
        return ser2;
    }

    private static GestorePersistenza preparaPersistenza() {
        GestorePersistenza gp = JSONizzatore.getInstance();
    }
}
```

scomposizione delle
funzionalità: serializzazione e
salvataggio operati da due
oggetti separati
(concettualmente, non è detto
che lo siano fisicamente)

```

        GestorePersistenza gp2 = GeneratoreDiToString.getInstance();
        return gp2;
    }
}

```

Si noti come abbiamo deciso di delegare al metodo `prepara()` le decisioni relative a quali serializzatori e quali gestori di persistenza utilizzare. Il resto del `Main()` è indipendente da queste decisioni.

Le classi di dominio non sono cambiate.

Interfaccia Serializzatore.java

```

package it.uniud.poo.dependency_inversion.interfaccia;

import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public interface Serializzatore {
    /**
     * @param anagrafica
     * @return un JSON che rappresenta tutta l'anagrafica
     */
    String serializza( Anagrafica anagrafica );

    /**
     * @param st
     * @return una rappresentazione dello studente
     */
    String serializza( Studente st );

    /**
     * @param carriera
     * @return una rappresentazione della carriera di uno studente
     */
    String serializza( CarrieraUniversitaria carriera );

    /**
     * @param es
     * @return una rappresentazione di un esame
     */
    String serializza( Esame es );
}

```

Interfaccia GestorePersistenza.java

```
package it.uniud.poo.dependency_inversion.interfaccia;

import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;

public interface GestorePersistenza {
    /**
     * sala in un file la rappresentazione dell'anagrafica
     * @param anagrafica
     * @throws FileNotFoundException se non si puo' aprire il file
     * @throws UnsupportedEncodingException se l'encoding non e' supportato
     */
    void salva( Anagrafica anagrafica ) throws FileNotFoundException,
        UnsupportedEncodingException;
}
```

JSONizzatore.java

```
package it.uniud.poo.dependency_inversion.interfaccia;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import lombok.Getter;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.stream.Collectors;

/**
 * Servizio deputato alla generazione del json e al suo salvataggio e stampa
 */
public class JSONizzatore implements Serializzatore , GestorePersistenza {

    @Getter
    public final static JSONizzatore instance = new JSONizzatore();
    private Gson gson;

    public JSONizzatore() {
        GsonBuilder builder = new GsonBuilder();
        builder.setPrettyPrinting()
            .serializeNulls()
            .excludeFieldsWithoutExposeAnnotation();
    }
}
```

```

        gson = builder.create();
    }

    /**
     * @param anagrafica
     * @return una rappresentazione in json dell'istanza
     */
    @Override
    public String serializza( Anagrafica anagrafica ) {
        String tmp = anagrafica.stream()
            .map( st -> serializza( st ) )
            .collect( Collectors.joining( "," ) );
        String ris = String.format( "{\n\"elencoStudenti\": [ %s ]}", tmp );
        return ris;
    }

    /**
     * @return una rappresentazione in json dell'istanza
     */
    @Override
    public String serializza( Studente st ){
        String ris = String.format("{\n\"nome\": \"%s\", "+
            "\"cognome\": \"%s\", \"matricola\": \"%s\", \"elencoEsami\": [\" +
            " %s ]}",
            st.getNome(), st.getCognome(), st.getMatricola(),
            serializza( st.getCarriera() ));
        return ris;
    }

    @Override
    public String serializza( CarrieraUniversitaria carriera ){
        String tmp = carriera.stream()
            .map( es -> serializza( es ) )
            .collect( Collectors.joining( "," ) );
        String ris = String.format("{\n\"dataIscrizione\": %s, \"dataLaurea\": %s, \" +
            "\"votoLaurea\": %s, \"elencoEsami\": [%s]}",
            carriera.getDataIscrizione(),
            carriera.getDataLaurea(),
            carriera.getVotoLaurea(), tmp);
        return ris;
    }

    @Override
    public String serializza( Esame es ) {
        String dataRegistrazione = es.getData().toString();
        String ris = String.format( "{\n\"corso\": \"%s\", \"voto\": "+

```

```

        "%d, \\"lode\\": %s, \\"data\\": \\"%s\\\"",
        es.getCorso(), es.getVoto(), es.isLode(), dataRegistrazione );
    }
    return ris;
}

/**
 * Salva in un file la stringa json
 * @param anagrafica
 * @throws FileNotFoundException
 * @throws UnsupportedEncodingException
 */
@Override
public void salva( Anagrafica anagrafica ) throws
    FileNotFoundException, UnsupportedEncodingException {
    String res = serializza( anagrafica );
    PrintWriter writer = new PrintWriter( "studenti.json", "UTF-8" );
    writer.println( res );
    writer.close();
}

/**
 * @return una rappresentazione in json dell'istanza fatta in
 * maniera piu'
 * efficiente e semplice
 */
public String toJsonVERO( Anagrafica anagrafica ) {
    GsonBuilder builder = new GsonBuilder();
    builder.setPrettyPrinting()
        .serializeNulls();
    // Una soluzione migliore sarebbe di evitare di usare l'exlude
    // ho fatto cosi' per motivi didattici
    Gson gson = builder.create();
    return gson.toJson( anagrafica );
}
}

```

Generatore di ToString.java

Si tratta di un meccanismo alternativo al JSON, in realtà stupido e inutile, utile solo per far vedere come lo si possa creare.

```

package it.uniud.poo.dependency_inversion.interfaccia;

import com.google.gson.Gson;

```

```

import com.google.gson.GsonBuilder;
import lombok.Getter;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.stream.Collectors;

/**
 * Servizio deputato alla generazione del json e al suo salvataggio e stampa
 */
public class GeneratoreDiToString implements Serializzatore, GestorePersistenza {

    @Getter
    public final static GeneratoreDiToString instance = new GeneratoreDiToString();

    /**
     * @param anagrafica
     * @return una rappresentazione in json dell'istanza
     */
    @Override
    public String serializza( Anagrafica anagrafica ) {
        String tmp = anagrafica.stream()
            .map( st -> serializza( st ))
            .collect( Collectors.joining( "," ) );
        String ris = String.format( "{ \"elencoStudenti\": [ %s ]}", tmp );
        return ris;
    }

    /**
     * @return una rappresentazione in json dell'istanza
     */
    @Override
    public String serializza( Studente st ){
        String ris = String.format( "%s %s", st.toString(),
            serializza( st.getCarriera() ));
        return ris;
    }

    @Override
    public String serializza( CarrieraUniversitaria carriera ){
        String tmp = carriera.stream()
            .map( es -> serializza( es ) )
            .collect( Collectors.joining( "," ) );
        String ris = String.format( "%s %s", carriera.toString(), tmp);
        return ris;
    }
}

```



```

@Override
public String serializza( Esame es ) {
    String dataRegistrazione = es.getData().toString();
    String ris = String.format( "%s %s", es.toString(), dataRegistrazione);
    return ris;
}

/**
 * Salva in un file la stringa json
 * @param anagrafica
 * @throws FileNotFoundException
 * @throws UnsupportedEncodingException
 */
@Override
public void salva( Anagrafica anagrafica ) throws
FileNotFoundException, UnsupportedEncodingException {
    String res = serializza( anagrafica );
    PrintWriter writer = new PrintWriter( "studenti.json", "UTF-8" );
    writer.println( res );
    writer.close();
}

/**
 * @return una rappresentazione in json dell'istanza fatta in
maniera
piu' efficiente e semplice
 */
public String toJsonVERO( Anagrafica anagrafica ) {
    GsonBuilder builder = new GsonBuilder();
    builder.setPrettyPrinting()
        .serializeNulls();
    // Una soluzione migliore sarebbe di evitare di usare l'exclude
    // ho fatto cosi' per motivi didattici
    Gson gson = builder.create();
    return gson.toJson( anagrafica );
}
}

```

Osservazioni

Il Main dipende stavolta non dall'implementazione dei servizi, ma da una loro astrazione, Serializzatore e GestorePersistenza. E le loro implementazioni, come prima, dipendono dalle classi di dominio. Rispetto alla versione iniziale abbi-

la Dependency Inversion non è a costo zero: infatti diventa gradualmente più complicato e complesso rispetto alla prima versione, però si guadagna enormemente nella manutenzione e nel testing

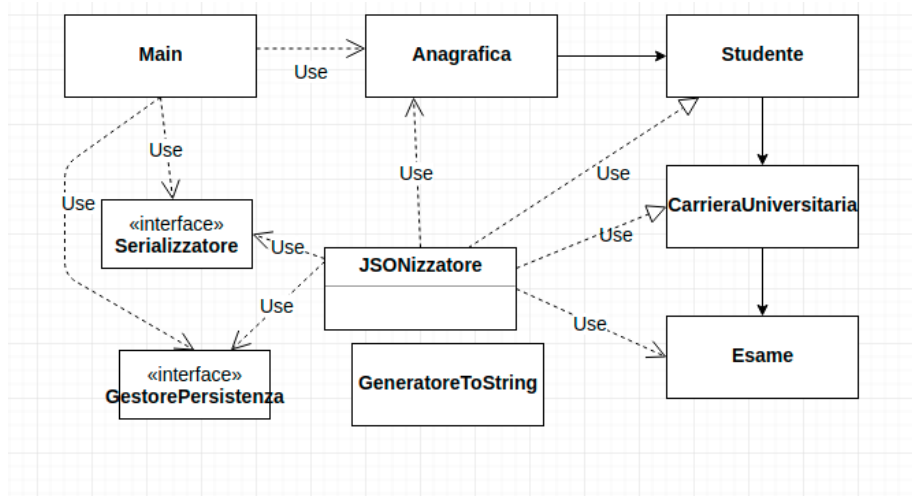


Figure 3: Diagramma delle classi per versione con interfacce

amo **invertito le dipendenze**. Questo consente di **cambiare l'implementazione dei servizi senza dover modificare nulla al Main**, eccetto la parte in cui si configura quale servizio si vuole usare. In questo esempio questa parte è un metodo di Main; nella realtà si possono usare dei **dependency frameworks** a cui far fare questo lavoro, e **quindi anche questa responsabilità non ricade al Main, ma viene delegata ad altri**.

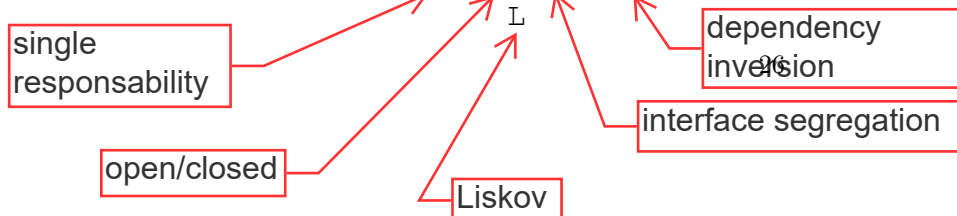
Inoltre creando dei tipi generali per i servizi (le **interfacce**) **abbiamo creato il modo per facilmente estendere il comportamento di questo programma**. Se ci dovesse servire una altro serializzatore (ad es in XML) dovremmo creare una nuova classe che implementa **Serializzatore** e configurarne l'utilizzo dentro la parte di Main dedicata alla preparazione.

Le due interfacce espongono metodi per serializzare e per salvare su disco, rispettivamente. E questi metodi devono venir implementati dalle classi che implementano le interfacce e i relativi servizi.

Si noti come le due ultime classi implementino le due interfacce: ciascuna classe implementa entrambe le interfacce. È stato fatto di proposito per **segregare le interfacce**: invece di averne una monolitica (che per istanziare devo usare classi altrettanto monolitiche) mi conviene averne tante e piccole come funzionalità (che poi posso implementare con una classe, 1:1 con l'interfaccia o 1:n).

in questo esempio abbiamo visto: - applicazione del principio di singola responsabilità - applicazione del principio di aperti alle estensioni e chiusi alle modifiche - applicazione del principio di inversione delle dipendenze - applicazione del principio di segregazione delle interfacce

ovvero la S, la O, la I e la D di SOLID.



manca il concetto di **DEPENDENCY INJECTION** per approfondire il disaccoppiamento