**RELAZIONE SECONDA PARTE DEL PROGETTO**

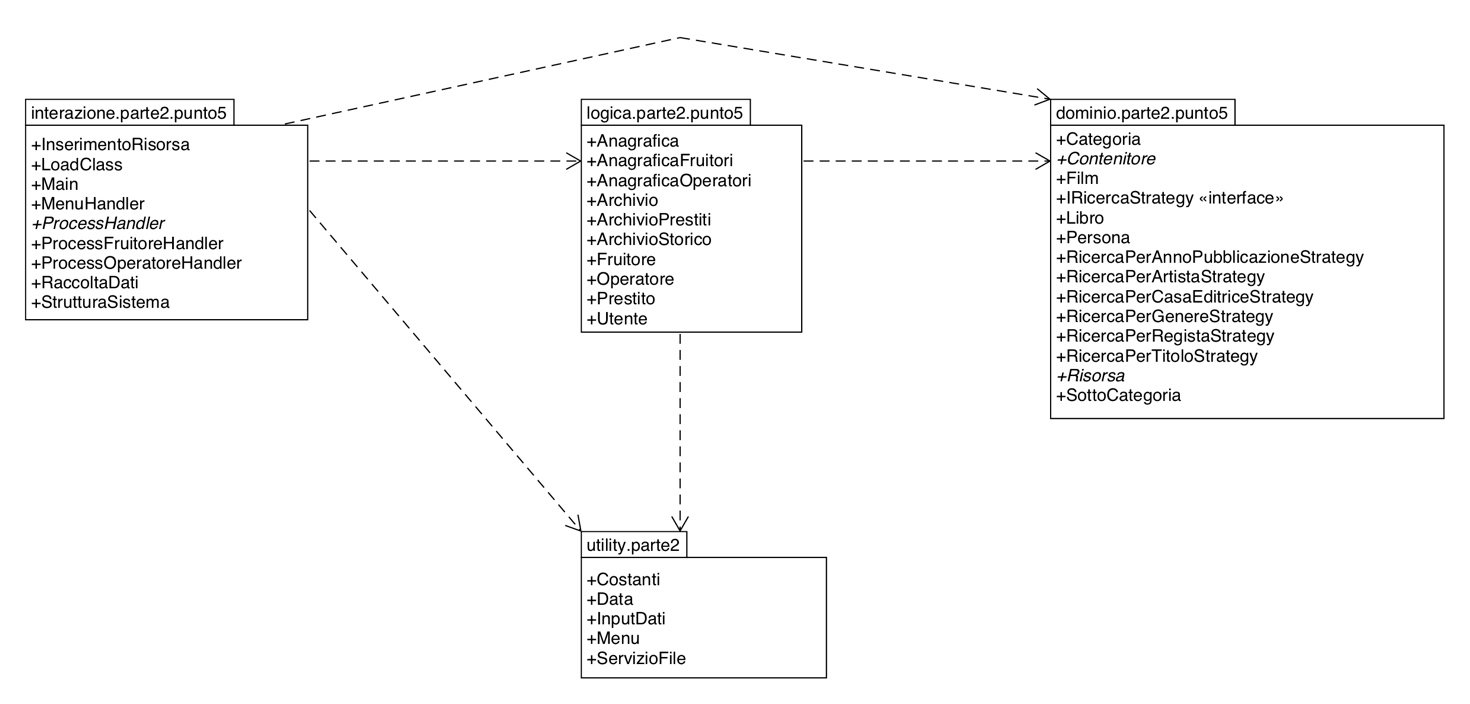
**All’interno della cartella di consegna del progetto vi sono due sottocartelle: una contenente il codice della versione precedente comprensivo di test e l’altra il codice rifattorizzato distinto secondo i vari punti indicati dalla traccia.**

**-Primo punto (realizzato per ultimo)**

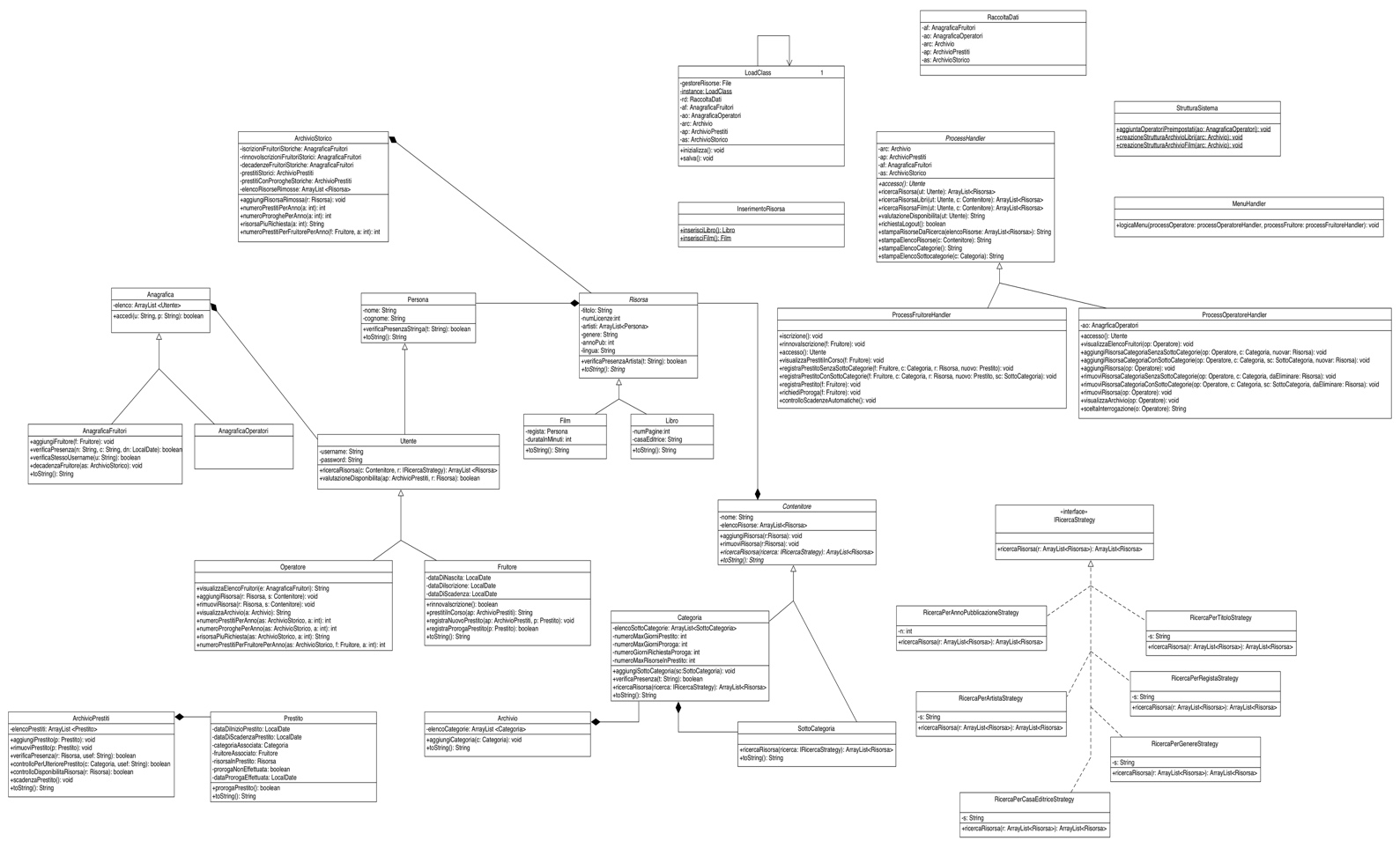
Come si può evincere dai diagrammi dei package e delle classi, il sistema è stato progettato in modo che l’interazione con l’utente avvenga solo nelle classi appartenenti al package *‘interazione’* secondo una strutturazione progressiva passando dapprima per il *MenuHandler* ed in seguito per lo specifico *ProcessFruitore* o *ProcessOperatore*, il quale permette la comunicazione con il package *‘logica’* favorendo la scomposizione e la differenziazione delle operazioni che comportano il dialogo con l’utente da quelle applicative secondo il principio di separazione modello-vista. Le dipendenze si propagano successivamente in un unico verso fino ad arrivare al package *‘dominio’*, che contiene gli elementi costitutivi su cui è stato realizzato il sistema, evitando così legami ciclici ed accoppiamenti di oggetti non UI con oggetti UI.

Un chiaro esempio a supporto di quanto indicato può essere dedotto dalle varie operazioni di stampa a video che vengono distinte nella composizione della stringa da visualizzare, all’interno delle classi di *‘logica’* e di *‘dominio’* attraverso il metodo *toString()*, e nella presentazione all’utente nelle classi di *‘interazione’*.

**-Diagramma dei package**

****

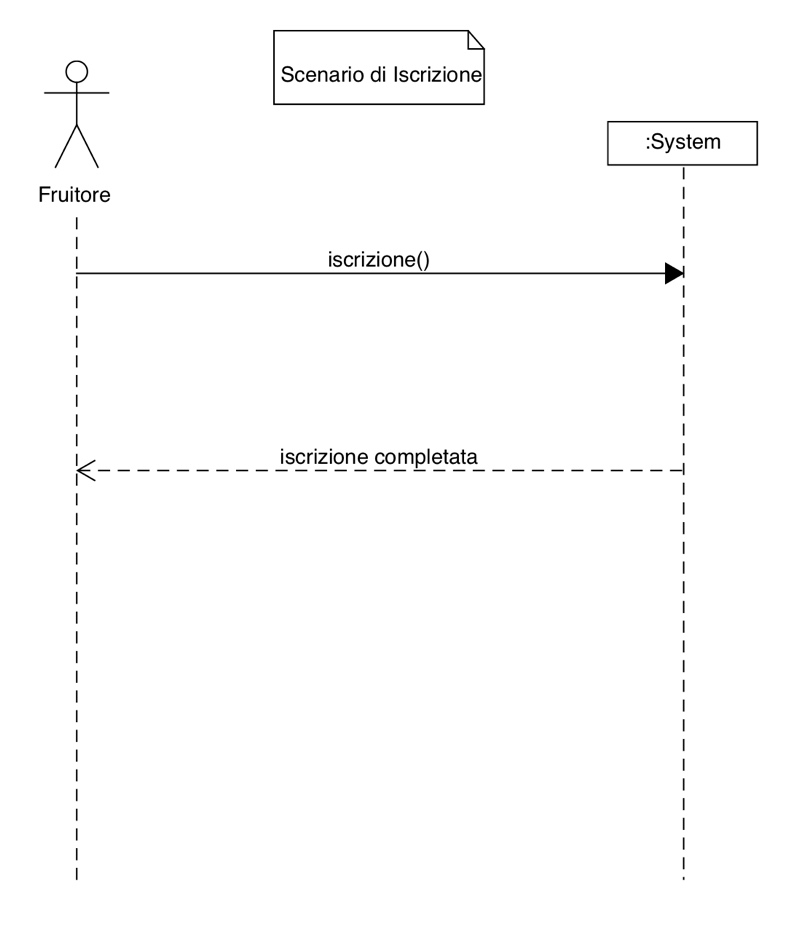
**-Diagramma delle classi**



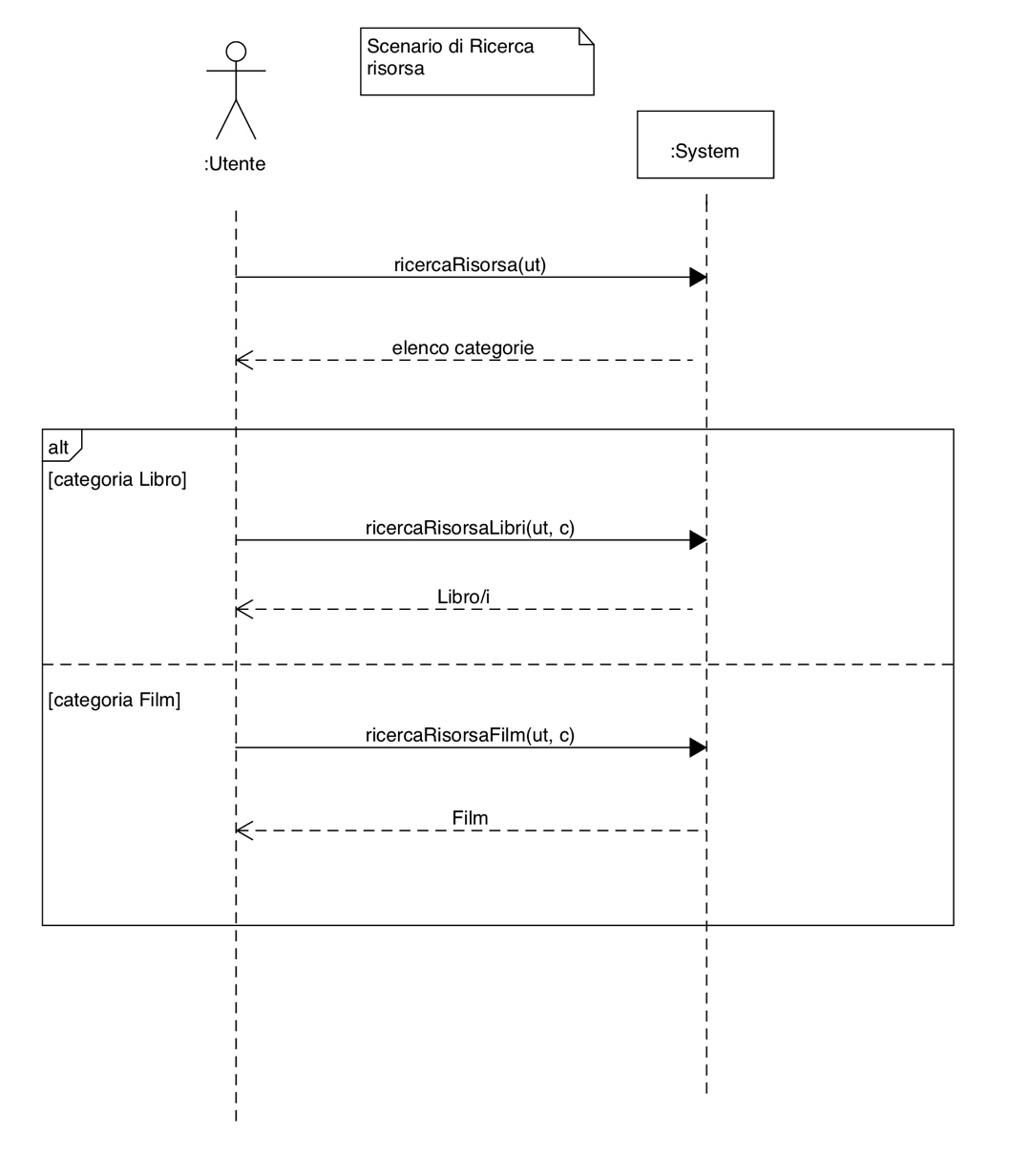
**-Secondo punto**

Sono stati realizzati i diagrammi di sequenza di sistema dei seguenti casi d’uso: Iscrizione, Ricerca risorsa e Aggiunta risorsa.

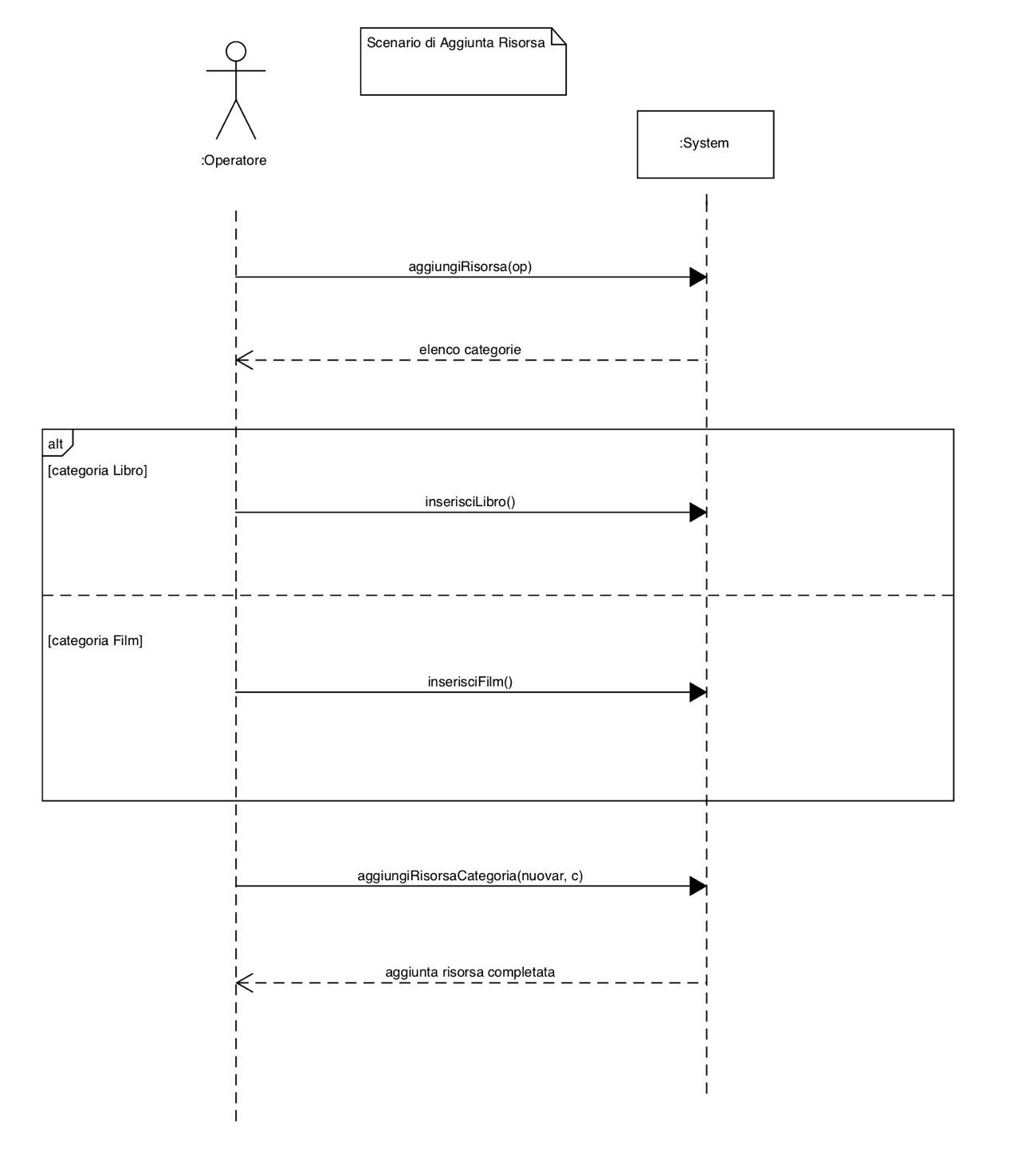
**-SSD Iscrizione**



**-SSD Ricerca risorsa**



**-SSD Aggiunta Risorsa**



Poi sono stati definiti i contratti per alcune delle operazioni di sistema presenti negli SSD:

**Contratto CO1: iscrizione**

**Operazione**: iscrizione()

**Riferimenti**: caso d’uso: Iscrizione

**Pre-condizioni**: nessuna.

**Post-condizioni:** - è stata creata un’istanza f di Fruitore.

- l’istanza f è stata aggiunta all’AnagraficaFruitori af.

- l’istanza f è stata aggiunta alle iscrizioni storiche dei fruitori dell’ArchivioStorico

as.

**Contratto CO2: ricercaRisorsa**

**Operazione**: ricercaRisorsa(ut: Utente)

**Riferimenti**: caso d’uso: Ricerca risorsa

**Pre-condizioni**: l’utente ha effettuato l’accesso.

**Post-condizioni**: -nessuna.

**Contratto CO3: inserimentoRisorsa**

**Operazione**: inserisciLibro() (ad es.)

**Riferimenti**: caso d’uso: Aggiunta Risorsa

**Pre-condizioni**: l’operatore ha dichiarato di voler proseguire nell’aggiunta della risorsa.

**Post-condizioni:** - è stata creata una nuova istanza lib di Libro.

- gli attributi di lib sono stati inizializzati.

**Contratto CO4: aggiungiRisorsaCategoria**

**Operazione**: aggiungiRisorsaCategoria(r: Risorsa, s: Categoria)

**Riferimenti**: caso d’uso: Aggiunta Risorsa

**Pre-condizioni:** - è stata creata un’istanza di Risorsa.

- è stato possibile reperire la Categoria di appartenenza della Risorsa indicata.

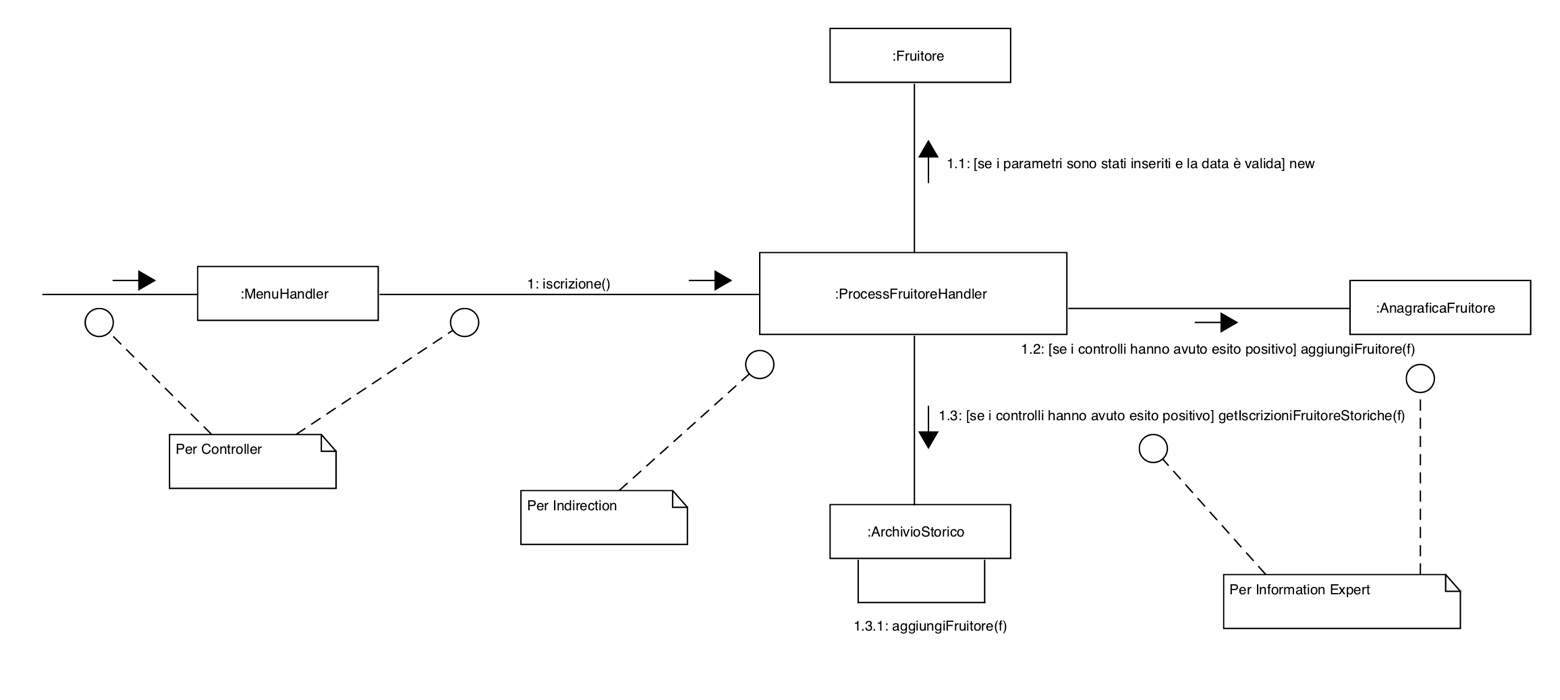
**Post-condizioni**: -nessuna.

Infine i diagrammi di comunicazione mostrano l’applicazione di alcuni dei pattern GRASP.

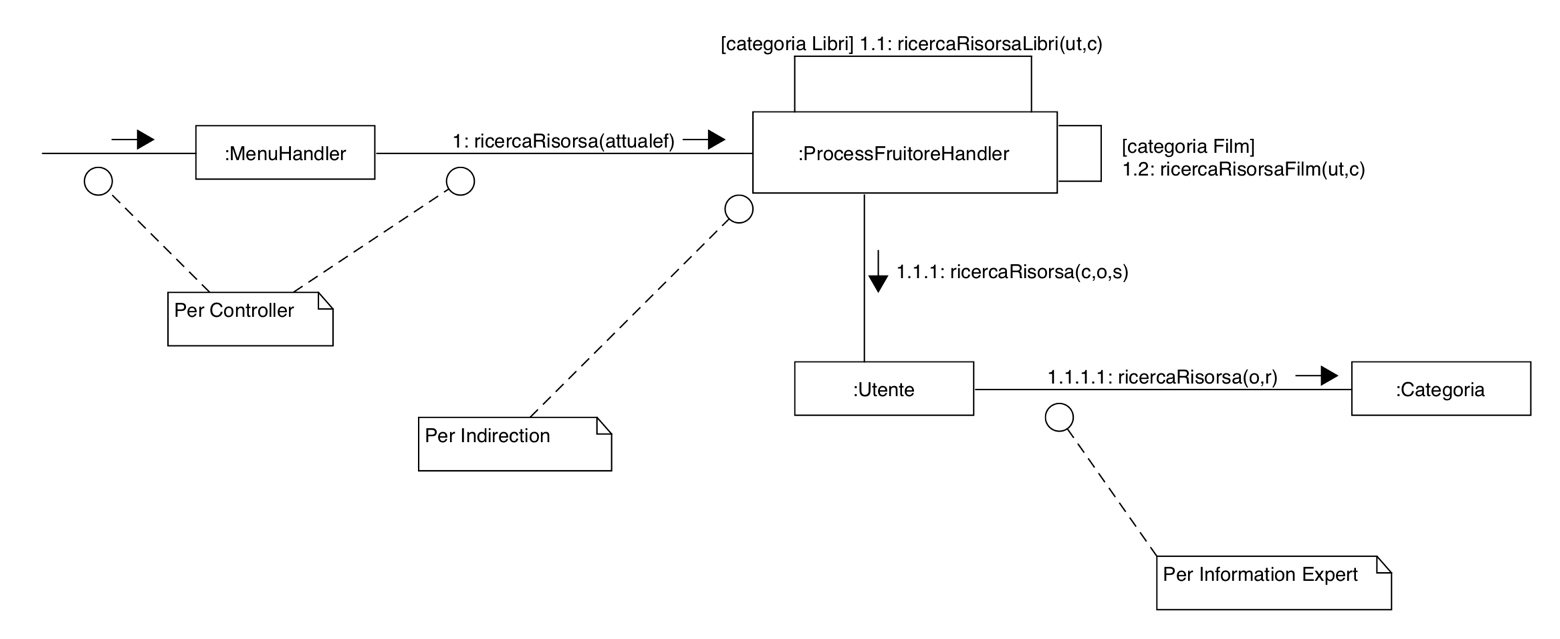
È stato applicato il pattern *‘Pure Fabrication’* con l’introduzione della classe *LoadClass* che fa da tramite tra il *Main* e la logica dell’applicazione: essa si occupa di operazioni quali l’inizializzazione delle istanze al momento della creazione del file, il reperimento delle istanze da file e il salvataggio su file.

La classe *GestoreMenu* è stata ridenominata in *MenuHandler* e sono state create tre nuove classi: *ProcessHandler*, *ProcessFruitoreHandler* e *ProcessOperatoreHandler* che riguardano rispettivamente i processi comuni a *Fruitore* e *Operatore*, i processi relativi al *Fruitore* e i processirelativi all’*Operatore*. Tutto ciò è stato fatto applicando il pattern *‘Controller’*: la classe *MenuHandler* si occupa dell’interazione con l’utente ed è a conoscenza solo di esso, delega poi il lavoro alle classi *ProcessFruitoreHandler* o *ProcessOperatoreHandler*, che costituiscono uno strato aggiuntivo tra *MenuHandler* e la logica dell’applicazione, formando dunque delle classi indirezione secondo il pattern *‘Indirection’*.

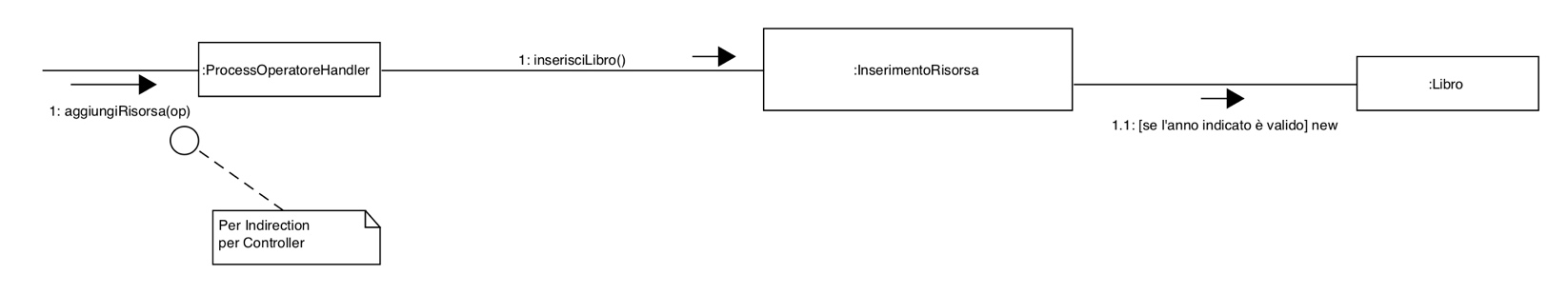
**-Diagramma di comunicazione iscrizione**



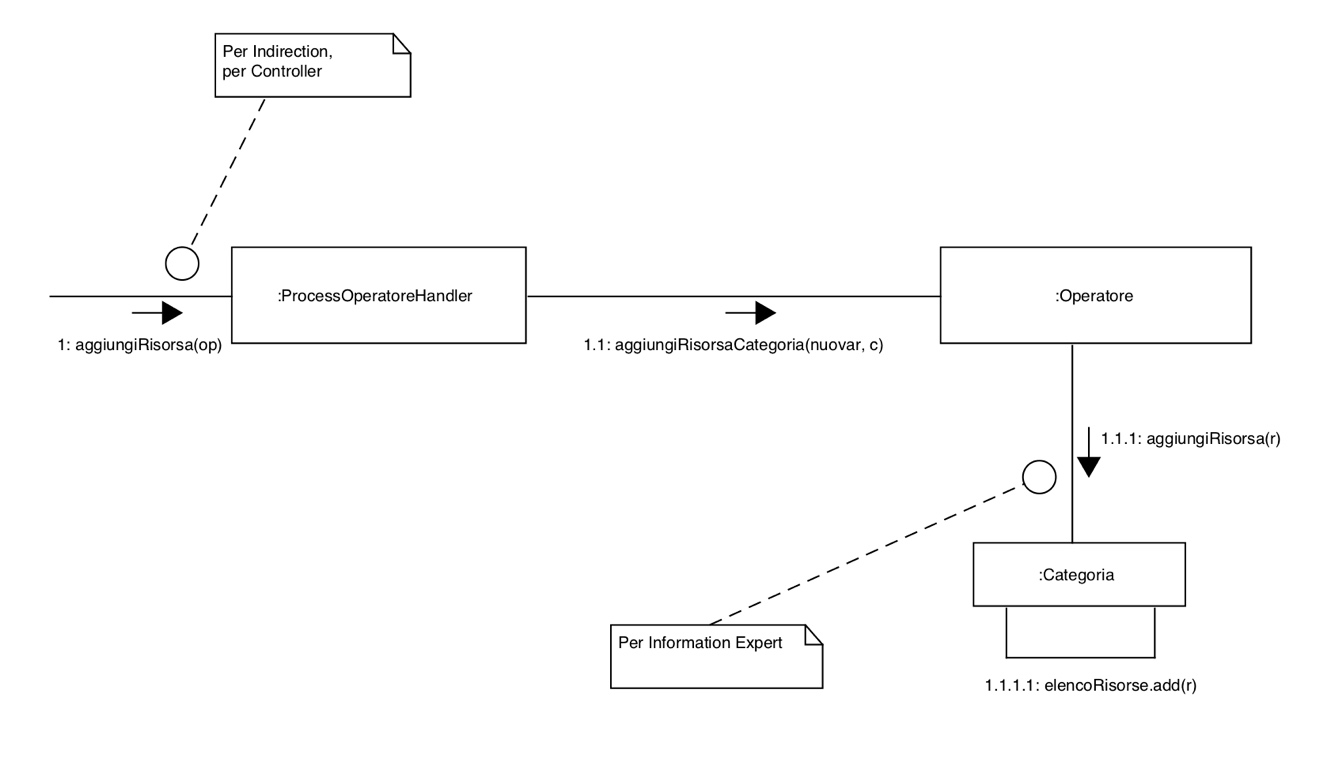
**-Diagramma di comunicazione ricercaRisorsa**



**-Diagramma di comunicazione inserimentoRisorsa**



**-Diagramma di comunicazione di aggiungiRisorsaCategoria**



**All’interno della cartella punto2 è possibile trovare la sottocartella contenente il codice completo comprensivo di test.**

**-Terzo punto**

In accordo con i due principi SRP ed OCP è stata creata l’interface *Ricerca* per il metodo generico di *ricercaRisorsa()* poi implementato dettagliatamente dalle varie classi deputate a realizzare in maniera più specifica le numerose tipologie di ricerca, riducendo di fatto le responsabilità attribuite alle diverse classi e favorendo un approccio distribuito nella suddivisione del codice.

Sono poi stati spostati sia i metodi *stampaElencoRisorse()* e *stampaElencoSottocategorie()* da Categoria sia il metodo *stampaElencoCategorie()* da *Archivio* a *ProcessHandler*, in quanto fautori di operazioni attinenti alla semplice stampa delle informazioni e non direttamente coinvolti nella logica relativa allo strato di dominio; conseguentemente, volendo uniformare lo stile, il nome del metodo *ricercaRisorsaFormatoStringa() in ProcessHandler* è stato modificato in *stampaRisorseDaRicerca()* ed un’analoga ridenominazione di metodi ha interessato poi la classe Operatore.

Secondo quanto espresso dal principio OCP è stata inoltre modificata da ‘protected’ a ‘private’ la visibilità di alcuni attributi presenti nelle classi *Categoria, Risorsa, ProcessHandler* ed *Anagrafica* con l’introduzione dei rispettivi metodi getters.

In seguito, dopo un’attenta analisi, si è stabilito di non poter applicare ulteriormente il pattern SRP sulle restanti classi del progetto, poiché ciò avrebbe comportato lo scorporo di operazioni fondamentali per le diverse entità che le espongono, con una conseguente complicazione di codice che potrebbe poi apparire carente in termini di concisione.

Un’ultima valutazione si è voluta invece soffermare sulla piena aderenza al LSP per quanto riguarda le classi *Fruitore* ed *Operatore*, le quali ereditano da *Utente*, e *Libro* e *Film*, che realizzano *Risorsa*; pur non sovrascrivendo alcun metodo della classe padre, potrebbero tuttavia sorgere delle complicanze concettuali in merito alle conversioni di tipo presenti ad esempio in alcuni metodi delle classi che implementano *Ricerca* ed in *AnagraficaFruitori*. Tali operatori sono giustificabili tenendo presente che i metodi in cui ha efficacia la loro azione fanno riferimento ad oggetti specifici di una delle sottoclassi indicate e sono invocabili solo attraverso una procedura apposita che vincola espressamente l’uso dell’oggetto richiesto. In alternativa si sarebbe potuto pensare di introdurre, ad esempio, le firme dei metodi interessati nella classe *Risorsa*, con la conseguenza tuttavia di dover implementare queste operazioni sia in *Libro* che in *Film*; tale progettazione, coerente con il ‘design for uniformity’ del pattern *Composite* della GoF, appare però in contrasto con la possibilità di mantenere un codice semplice, snello e chiaro, ed è per questo che non è stata applicata.

Si è invece deciso di applicare tale pattern alle classi *Categoria* e *Sottocategoria* poiché quest’ultima, in quanto sottoclasse della prima, erediterebbe dei metodi che non potrebbe tuttavia utilizzare secondo la logica del programma e dovrebbe dunque sollevare delle eccezioni. Si è perciò pensato di annullare il legame esistente tra le due classi e di farle dipendere entrambe da una classe astratta *Contenitore*, la quale preserva gli attributi ed i metodi comuni.

**-Codice successivo al refactoring:**

public interface Ricerca {

ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> r);

}

public class RicercaPerAnnoPubblicazione implements Ricerca, Serializable {

private static final long serialVersionUID = 1L;

private int n;

public RicercaPerAnnoPubblicazione(String s){

this.n = Integer.parseInt(s);

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(ris.getAnnoPub() == n)

risorseCercate.add(ris);

}

return risorseCercate;

}

}

public class RicercaPerAttore implements Ricerca, Serializable{

private static final long serialVersionUID = 1L;

private String s;

public RicercaPerAttore(String s){

this.s = s;

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(((Film)ris).getAttore().toLowerCase().indexOf((s).toLowerCase()) > -1)

risorseCercate.add(ris);

}

return risorseCercate;

}

}

public class RicercaPerAutore implements Ricerca, Serializable{

private static final long serialVersionUID = 1L;

private String s;

public RicercaPerAutore(String s){

this.s = s;

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(((Libro)ris).getAutore().toLowerCase().indexOf((s).toLowerCase()) > -1)

risorseCercate.add(ris);

}

return risorseCercate;

}

}

public class RicercaPerCasaEditrice implements Ricerca, Serializable {

private static final long serialVersionUID = 1L;

private String s;

public RicercaPerCasaEditrice(String s){

this.s = s;

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris)

{

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(((Libro)ris).getCasaEditrice().equalsIgnoreCase(s))

risorseCercate.add(ris);

}

return risorseCercate;

}

}

public class RicercaPerGenere implements Ricerca, Serializable{

private static final long serialVersionUID = 1L;

private String s;

public RicercaPerGenere(String s){

this.s = s;

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(ris.getGenere().equalsIgnoreCase(s))

risorseCercate.add(ris);

}

return risorseCercate;

}

}

public class RicercaPerRegista implements Ricerca, Serializable{

private static final long serialVersionUID = 1L;

private String s;

public RicercaPerRegista(String s){

this.s = s;

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(((Film)ris).getRegista().toLowerCase().indexOf((s).toLowerCase())>-1)

risorseCercate.add(ris);

}

return risorseCercate;

}

}

public class RicercaPerTitolo implements Ricerca, Serializable{

private static final long serialVersionUID = 1L;

private String s;

public RicercaPerTitolo(String s){

this.s = s;

}

public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoris.size(); i++){

Risorsa ris = elencoris.get(i);

if(ris.getTitolo().toLowerCase().indexOf((s).toLowerCase()) > -1)

risorseCercate.add(ris);

}

return risorseCercate;

}

}

/\*

\* E' stata creata la classe astratta Contenitore che raccoglie i metodi

\* comuni alle due sottoclassi Categoria e SottoCategoria, nel rispetto

\* del principio di Liskov.

\*/

public abstract class Contenitore implements Serializable{

private static final long serialVersionUID = 1L;

private String nome;

private ArrayList <Risorsa> elencoRisorse;

public Contenitore(String n){

this.nome = n;

}

public void inizializzaElencoRisorse(){

elencoRisorse = new ArrayList <Risorsa> ();

}

public String getNome(){

return nome;

}

public ArrayList <Risorsa> getElencoRisorse(){

return elencoRisorse;

}

public Risorsa getRisorsa(String t){

for(int i = 0; i < elencoRisorse.size(); i++){

Risorsa r = elencoRisorse.get(i);

if(r.getTitolo().equalsIgnoreCase(t))

return r;

}

return null;

}

public void aggiungiRisorsa(Risorsa r){

elencoRisorse.add(r);

}

public void rimuoviRisorsa(Risorsa r){

elencoRisorse.remove(r);

}

public abstract ArrayList <Risorsa> ricercaRisorsa(Ricerca ricerca);

public abstract String toString();

}

/\*

\* La classe Categoria estende la classe astratta Contenitore nel

\* rispetto del principio di Liskov.

\*/

public class Categoria extends Contenitore implements Serializable{

private ArrayList <SottoCategoria> elencoSottoCategorie;

private int numeroMaxGiorniPrestito;

private int numeroMaxGiorniProroga;

private int numeroGiorniRichiestaProroga;

private int numeroMaxRisorseInPrestito;

. . .

public Categoria(String n, int numPres, int numMaxPro, int numRiPro, int numRis){

super(n);

//il restante corpo del metodo e’ rimasto invariato

}

public void inizializzaElencoSottoCategorie(){

//corpo del metodo rimasto invariato

}

public int getNumeroMaxGiorniPrestito(){

//corpo del metodo rimasto invariato

}

public int getNumeroMaxGiorniProroga(){

//corpo del metodo rimasto invariato

}

public int getNumeroGiorniRichiestaProroga(){

//corpo del metodo rimasto invariato

}

public int getNumeroMaxRisorseInPrestito(){

//corpo del metodo rimasto invariato

}

public ArrayList <SottoCategoria> getElencoSottoCategorie(){

//corpo del metodo rimasto invariato

}

public void aggiungiSottoCategoria(SottoCategoria sc){

//corpo del metodo rimasto invariato

}

public boolean verificaPresenza(String t){

//corpo del metodo rimasto invariato

}

//Il metodo ricercaRisorsa è stato sistemato nel rispetto di OCP, ora si

//appoggia all'interface Ricerca

public ArrayList <Risorsa> ricercaRisorsa(Ricerca ricerca)

{

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

if(elencoSottoCategorie == null){

risorseCercate = ricerca.ricercaRisorsa(getElencoRisorse());

}

else{

for(int i = 0; i < elencoSottoCategorie.size(); i++){

SottoCategoria sc = elencoSottoCategorie.get(i);

risorseCercate.addAll(sc.ricercaRisorsa(ricerca));

}

}

return risorseCercate;

}

public String toString(){

//corpo del metodo rimasto invariato

}

}

/\*

\* La classe SottoCategoria estende la classe astratta Contenitore nel

\* rispetto del principio di Liskov.

\*/

public class SottoCategoria extends Contenitore implements Serializable{

. . .

public SottoCategoria(String ns){

super(ns);

inizializzaElencoRisorse();

}

//Il metodo ricercaRisorsa è stato sistemato nel rispetto di OCP, ora si

//appoggia all'interface Ricerca

public ArrayList <Risorsa> ricercaRisorsa(Ricerca ricerca){

return ricerca.ricercaRisorsa(getElencoRisorse());

}

public String toString(){

//corpo del metodo rimasto invariato

}

}

public abstract class Risorsa implements Serializable{

//L'attributo è stato reso private nel rispetto di OCP.

private String titolo;

. . .

}

public class Anagrafica implements Serializable{

. . .

//L'attributo elenco è stato reso private nel rispetto di OCP.

//Nella sottoclasse AnagraficaFruitore viene utilizzato il getter per accedere

//a questo attributo.

private ArrayList <Utente> elenco;

. . .

}

public abstract class ProcessHandler implements Serializable{

. . .

//Gli attributi arc e ap sono stati resi private nel rispetto di OCP.

//Gli attributi af e as sono stati spostati dalle sottoclassi alla superclasse

//in quanto comuni ad entrambi e sono stati resi private nel rispetto di OCP.

//Nelle sottoclassi ProcessFruitoreHandler e ProcessOperatoreHandler vengono

//utilizzati i getter per accedere a questi attributi.

private Archivio arc;

private ArchivioPrestiti ap;

private AnagraficaFruitori af;

private ArchivioStorico as;

public ProcessHandler(Archivio arc, ArchivioPrestiti ap){

this.arc = arc;

this.ap = ap;

this.af = af;

this.as = as;

}

public Archivio getArchivio(){

return arc;

}

public ArchivioPrestiti getArchivioPrestiti(){

return ap;

}

public AnagraficaFruitori getAnagraficaFruitori(){

return af;

}

public ArchivioStorico getArchivioStorico(){

return as;

}

. . .

//Metodo modificato in seguito alle modifiche fatte sul metodo ricercaRisorsa in

// Categoria e all'introduzione dell'interface Ricerca e delle classi che la

//implementano.

public ArrayList <Risorsa> ricercaRisorsaLibri(Utente ut, Contenitore c){

int numScelta = InputDati.leggiIntero(Costanti.AVVIO\_RICERCA\_LIBRI,

Costanti.NUM\_MINIMO, Costanti.NUM\_MASSIMO\_RICERCA);

String s = "";

Ricerca r = null;

switch(numScelta){

case 1: s = InputDati.leggiStringa(Costanti.INS\_PAROLA\_TITOLO\_RISORSA);

r = new RicercaPerTitolo(s);

break;

case 2: s = InputDati.leggiStringa(Costanti.INS\_COGNOME\_AUTORE\_LIBRO);

r = new RicercaPerAutore(s);

break;

case 3: s = InputDati.leggiStringa(Costanti.INS\_GENERE\_RISORSA);

r = new RicercaPerGenere(s);

break;

case 4: s = Integer.toString(InputDati.leggiIntero(Costanti.INS\_ANNOPUB\_RISORSA));

r = new RicercaPerAnnoPubblicazione(s);

break;

case 5: s = InputDati.leggiStringa(Costanti.INS\_CASAED\_LIBRO);

r = new RicercaPerCasaEditrice(s);

break;

}

return ut.ricercaRisorsa(c,r);

}

//Metodo modificato in seguito alle modifiche fatte sul metodo ricercaRisorsa in

// Categoria e all'introduzione dell'interface Ricerca e delle classi che la

//implementano.

public ArrayList <Risorsa> ricercaRisorsaFilm(Utente ut, Contenitore c){

int numScelta = InputDati.leggiIntero(Costanti.AVVIO\_RICERCA\_FILM,

Costanti.NUM\_MINIMO, Costanti.NUM\_MASSIMO\_RICERCA);

String s = "";

Ricerca r = null;

switch(numScelta){

case 1: s = InputDati.leggiStringa(Costanti.INS\_PAROLA\_TITOLO\_RISORSA);

r = new RicercaPerTitolo(s);

break;

case 2: s = InputDati.leggiStringa(Costanti.INS\_COGNOME\_REGISTA\_FILM);

r = new RicercaPerRegista(s);

break;

case 3: s = InputDati.leggiStringa(Costanti.INS\_COGNOME\_ATTORE\_FILM);

r = new RicercaPerAttore(s);

break;

case 4: s =Integer.toString(InputDati.leggiIntero(Costanti.INS\_ANNOPUB\_RISORSA));

r = new RicercaPerAnnoPubblicazione(s);

break;

case 5: s = InputDati.leggiStringa(Costanti.INS\_GENERE\_RISORSA);

r = new RicercaPerGenere(s);

break;

}

return ut.ricercaRisorsa(c, r);

}

. . .

}

**-Codice precedente al refactoring:**

public class Categoria implements Serializable{

private String nomeCategoria;

protected ArrayList <Risorsa> elencoRisorse;

private ArrayList <SottoCategoria> elencoSottoCategorie;

private int numeroMaxGiorniPrestito;

private int numeroMaxGiorniProroga;

private int numeroGiorniRichiestaProroga;

private int numeroMaxRisorseInPrestito;

. . .

public static final String RIC\_PER\_TITOLO = "titolo";

public static final String RIC\_PER\_AUTORE\_I = "autore\_i";

public static final String RIC\_PER\_GENERE = "genere";

public static final String RIC\_PER\_ANNOPUB = "annoPub";

public static final String RIC\_PER\_CASAED = "casaEditrice";

public static final String RIC\_PER\_REGISTA = "regista";

public static final String RIC\_PER\_ATTORE\_I = "attore\_i";

public Categoria(String n, int numPres, int numMaxPro, int numRiPro, int numRis){

this.nomeCategoria = n;

this.numeroMaxGiorniPrestito = numPres;

this.numeroMaxGiorniProroga = numMaxPro;

this.numeroGiorniRichiestaProroga = numRiPro;

this.numeroMaxRisorseInPrestito = numRis;

}

public Categoria(){

}

public void inizializzaElencoRisorse(){

. . .

}

public void inizializzaElencoSottoCategorie(){

. . .

}

public String getNome(){

. . .

}

public int getNumeroMaxGiorniPrestito(){

. . .

}

public int getNumeroMaxGiorniProroga(){

. . .

}

public int getNumeroGiorniRichiestaProroga(){

. . .

}

public int getNumeroMaxRisorseInPrestito(){

. . .

}

public ArrayList <Risorsa> getElencoRisorse(){

. . .

}

public ArrayList <SottoCategoria> getElencoSottoCategorie(){

. . .

}

public Risorsa getRisorsa(String t){

. . .

}

public void aggiungiRisorsa(Risorsa r){

. . .

}

public void rimuoviRisorsa(Risorsa r){

. . .

}

public void aggiungiSottoCategoria(SottoCategoria sc){

. . .

}

public boolean verificaPresenza(String t){

. . .

}

public ArrayList <Risorsa> ricercaRisorsaInElenco(Object o, String cr){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

for(int i = 0; i < elencoRisorse.size(); i++){

Risorsa r = elencoRisorse.get(i);

switch(cr){

case RIC\_PER\_TITOLO:

if(r.getTitolo().toLowerCase().indexOf(((String)o).toLowerCase())>-1)

risorseCercate.add(r);

break;

case RIC\_PER\_AUTORE\_I:

if(((Libro)r).getAutore().toLowerCase().indexOf(((String)o).toLowerCase())>-1)

risorseCercate.add(r);

break;

case RIC\_PER\_GENERE: if(r.getGenere().equalsIgnoreCase((String)o))

risorseCercate.add(r);

break;

case RIC\_PER\_ANNOPUB: if(r.getAnnoPub() == (Integer)o)

risorseCercate.add(r);

break;

case RIC\_PER\_CASAED:

if(((Libro)r).getCasaEditrice().equalsIgnoreCase((String)o))

risorseCercate.add(r);

break;

case RIC\_PER\_REGISTA:

if(((Film)r).getRegista().toLowerCase().indexOf(((String)o).toLowerCase())>-1)

risorseCercate.add(r);

break;

case RIC\_PER\_ATTORE\_I:

if(((Film)r).getAttore().toLowerCase().indexOf(((String)o).toLowerCase())>-1)

risorseCercate.add(r);

break;

}

}

return risorseCercate;

}

public ArrayList <Risorsa> ricercaRisorsa(Object o, String comeRicercare){

ArrayList <Risorsa> risorseCercate = new ArrayList <>();

if(elencoSottoCategorie == null){

risorseCercate = ricercaRisorsaInElenco(o, comeRicercare);

}

else{

for(int i = 0; i < elencoSottoCategorie.size(); i++){

SottoCategoria sc = elencoSottoCategorie.get(i);

risorseCercate.addAll(sc.ricercaRisorsaInElenco(o,comeRicercare));

}

}

return risorseCercate;

}

public String stampaElencoRisorse(){

. . .

}

public String stampaElencoSottocategorie() {

. . .

}

public String toString(){

. . .

}

}

public class SottoCategoria extends Categoria implements Serializable{

private String nomeSottoC;

. . .

public SottoCategoria(String ns){

super();

this.nomeSottoC= ns;

inizializzaElencoRisorse();

}

public String getNome(){

return nomeSottoC;

}

public String toString(){

. . .

}

}

public abstract class Risorsa implements Serializable{

protected String titolo;

. . .

}

public abstract class ProcessHandler implements Serializable{

protected Archivio arc;

protected ArchivioPrestiti ap;

. . .

}

public class Anagrafica implements Serializable{

protected ArrayList <Utente> elenco;

. . .

}

public abstract class ProcessHandler implements Serializable{

protected Archivio arc;

protected ArchivioPrestiti ap;

public ProcessHandler(Archivio arc, ArchivioPrestiti ap){

this.arc = arc;

this.ap = ap;

}

public ArrayList <Risorsa> ricercaRisorsaLibri(Utente ut, Categoria c){

int numScelta = InputDati.leggiIntero(Costanti.AVVIO\_RICERCA\_LIBRI,

Costanti.NUM\_MINIMO, Costanti.NUM\_MASSIMO\_RICERCA);

Object o = null;

String s = "";

switch(numScelta){

case 1: o = InputDati.leggiStringa(Costanti.INS\_PAROLA\_TITOLO\_RISORSA);

s = Categoria.RIC\_PER\_TITOLO;

break;

case 2: o = InputDati.leggiStringa(Costanti.INS\_COGNOME\_AUTORE\_LIBRO);

s = Categoria.RIC\_PER\_AUTORE\_I;

break;

case 3: o = InputDati.leggiStringa(Costanti.INS\_GENERE\_RISORSA);

s = Categoria.RIC\_PER\_GENERE;

break;

case 4: o = InputDati.leggiIntero(Costanti.INS\_ANNOPUB\_RISORSA);

s = Categoria.RIC\_PER\_ANNOPUB;

break;

case 5: o = InputDati.leggiStringa(Costanti.INS\_CASAED\_LIBRO);

s = Categoria.RIC\_PER\_CASAED;

break;

}

return ut.ricercaRisorsa(c, o, s);

}

public ArrayList <Risorsa> ricercaRisorsaFilm(Utente ut, Categoria c){

int numScelta = InputDati.leggiIntero(Costanti.AVVIO\_RICERCA\_FILM,

Costanti.NUM\_MINIMO, Costanti.NUM\_MASSIMO\_RICERCA);

Object o = null;

String s = "";

switch(numScelta){

case 1: o = InputDati.leggiStringa(Costanti.INS\_PAROLA\_TITOLO\_RISORSA);

s = Categoria.RIC\_PER\_TITOLO;

break;

case 2: o = InputDati.leggiStringa(Costanti.INS\_COGNOME\_REGISTA\_FILM);

s = Categoria.RIC\_PER\_REGISTA;

break;

case 3: o = InputDati.leggiStringa(Costanti.INS\_COGNOME\_ATTORE\_FILM);

s = Categoria.RIC\_PER\_ATTORE\_I;

break;

case 4: o = InputDati.leggiIntero(Costanti.INS\_ANNOPUB\_RISORSA);

s = Categoria.RIC\_PER\_ANNOPUB;

break;

case 5: o = InputDati.leggiStringa(Costanti.INS\_GENERE\_RISORSA);

s = Categoria.RIC\_PER\_GENERE;

break;

}

return ut.ricercaRisorsa(c, o, s);

}

. . .

}

**All’interno della cartella punto3 è possibile trovare la sottocartella contenente il codice completo comprensivo di test.**