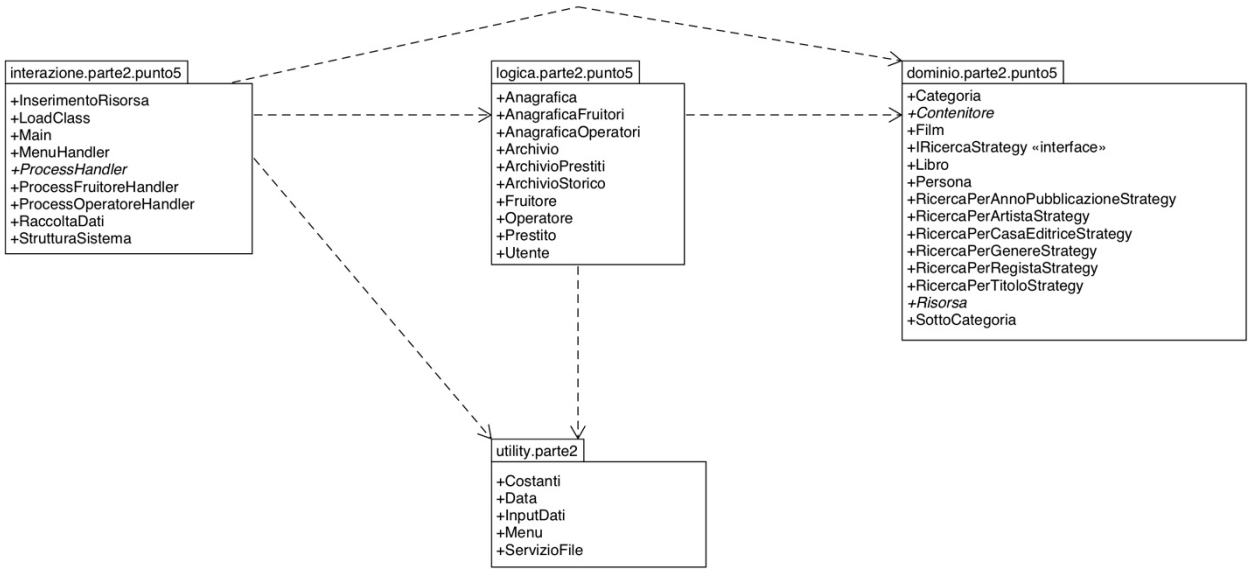
RELAZIONE SECONDA PARTE DEL PROGETTO

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.

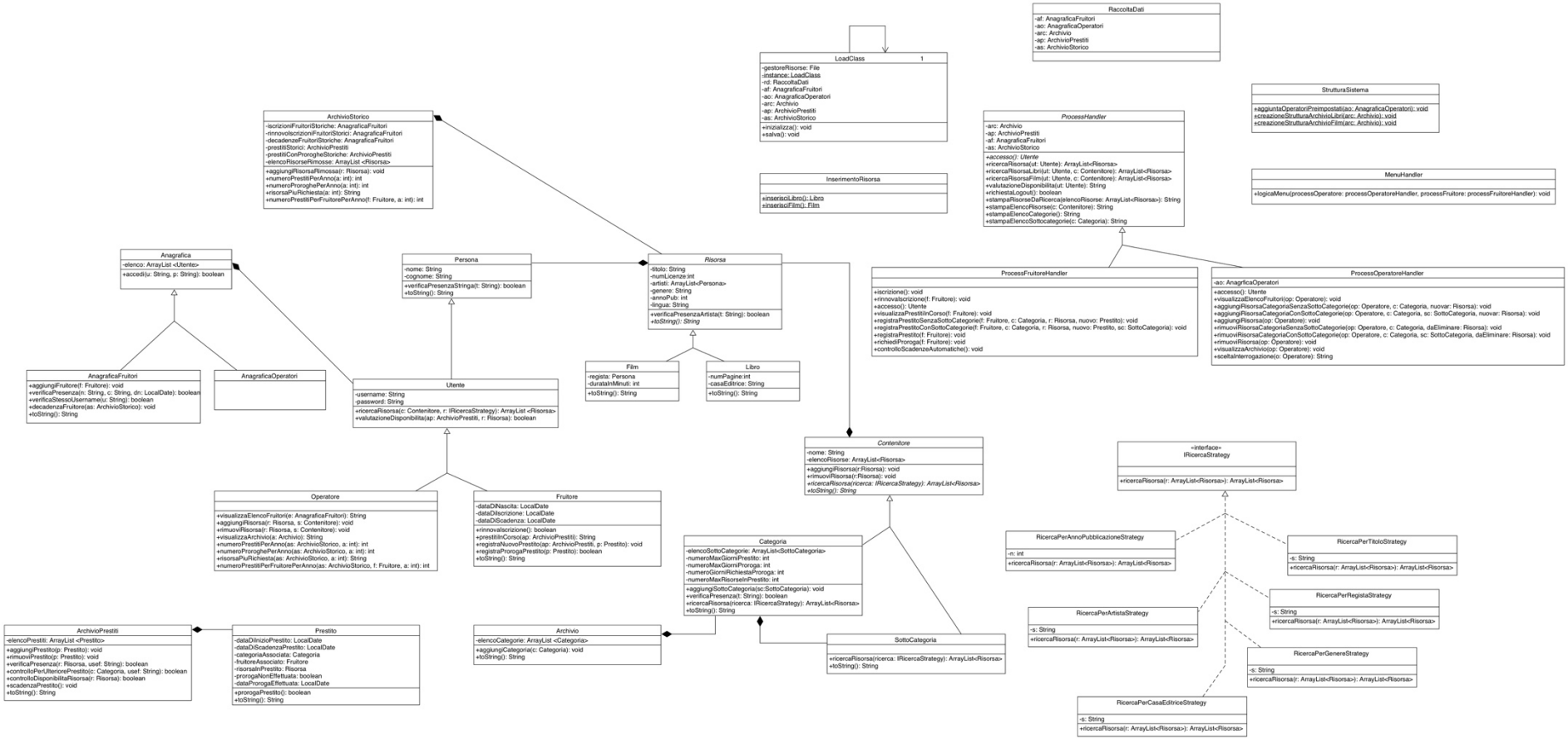
-Primo punto (realizzato per ultimo)

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 *ProcessFruitoreHandler* o *ProcessOperatoreHandler*, 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



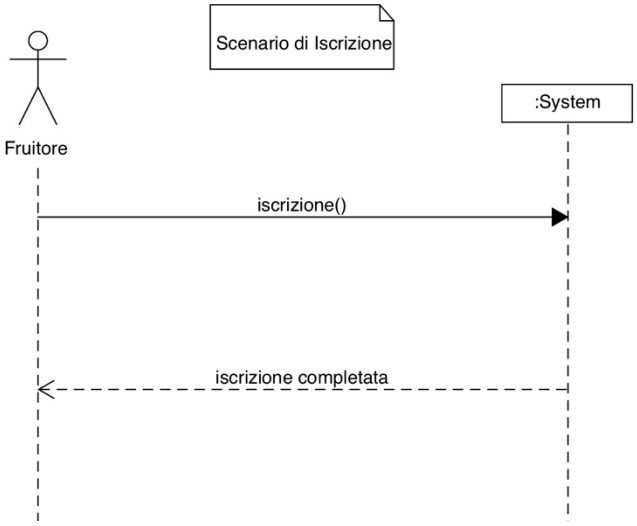
-Diagramma delle classi



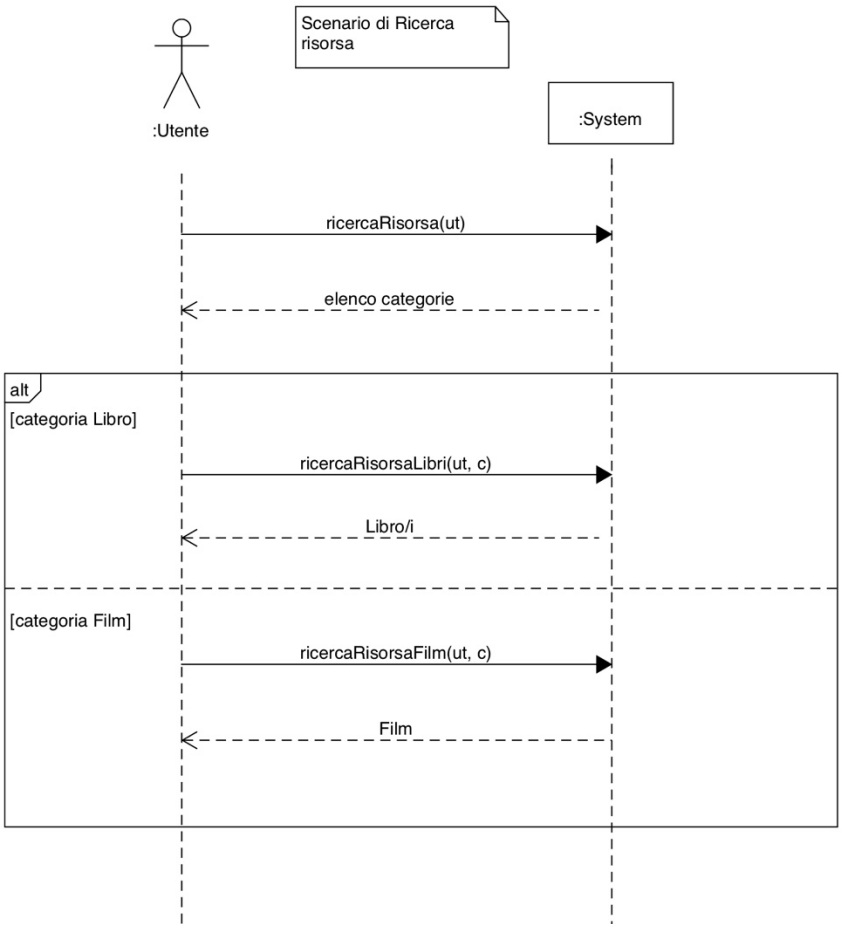
-Secondo punto

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

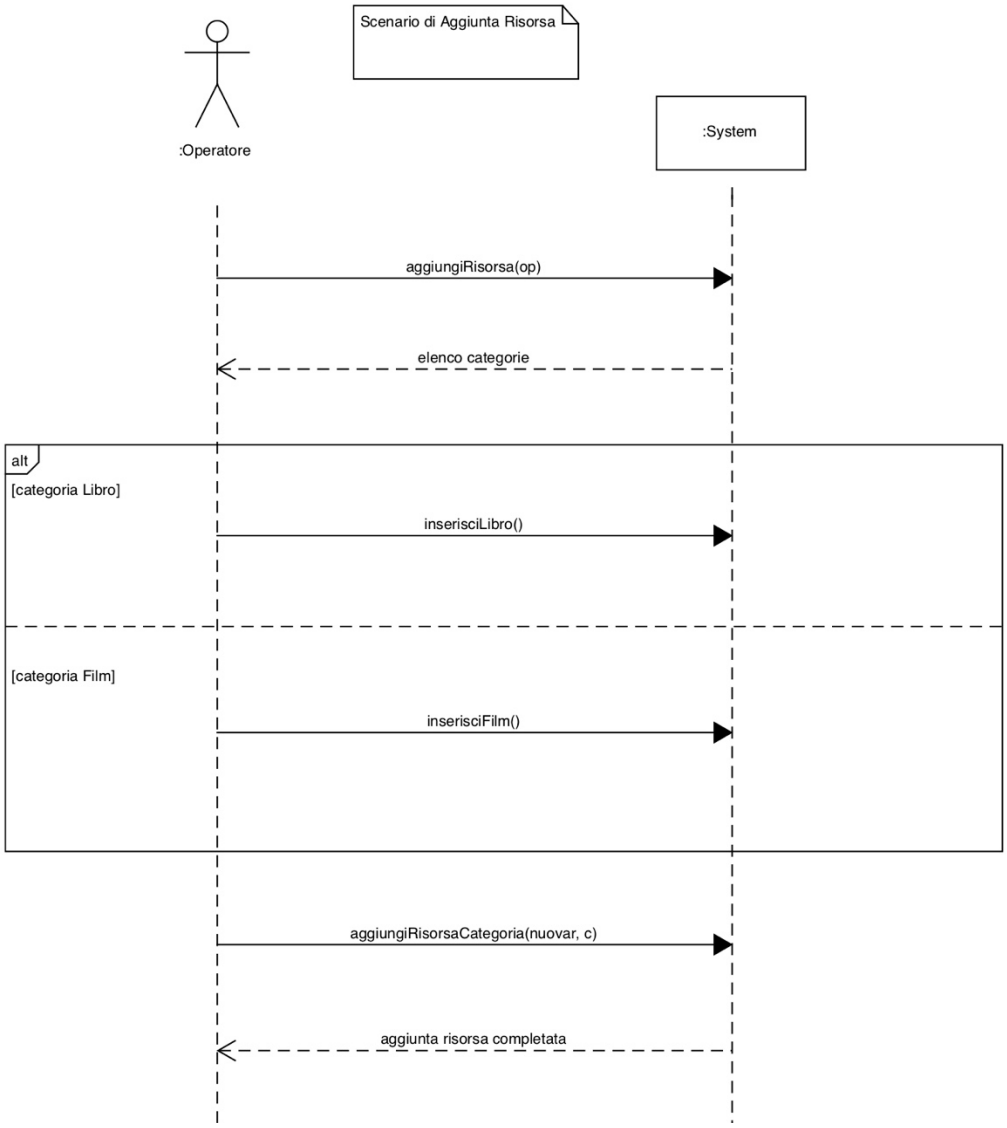
-SSD Iscrizione



-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 effettuato l'accesso e ha dichiarato di voler proseguire nell'aggiunta della risorsa.

Post-condizioni: - è stata creata una nuova istanza lib di Libro con tutti gli attributi inizializzati.

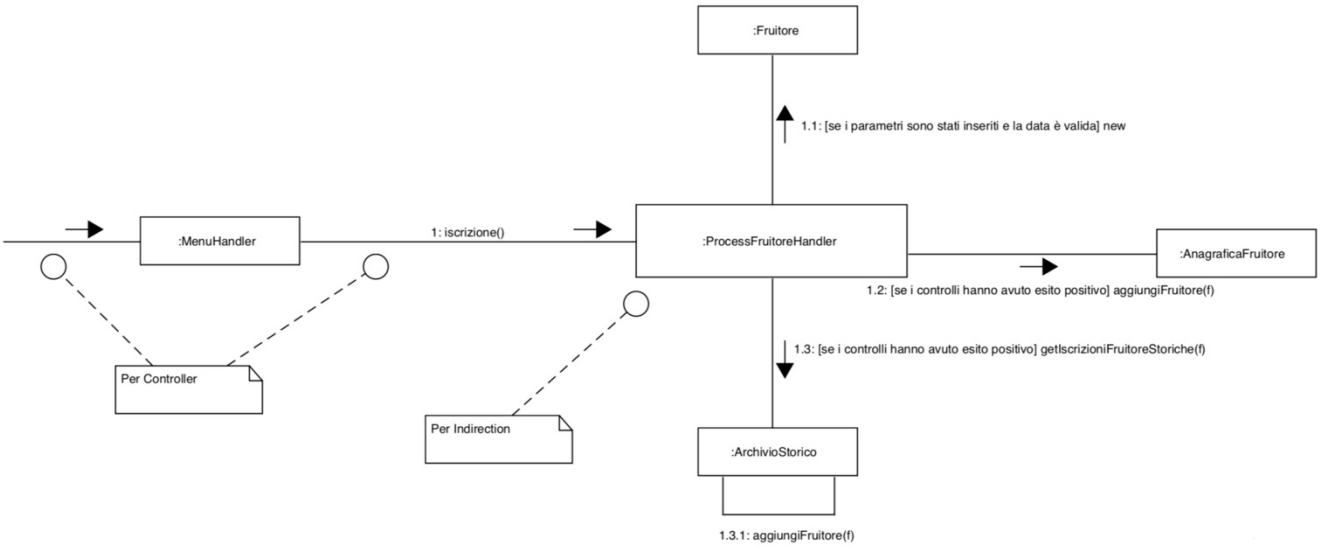
Contratto CO4: aggiungiRisorsaCategoria

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.

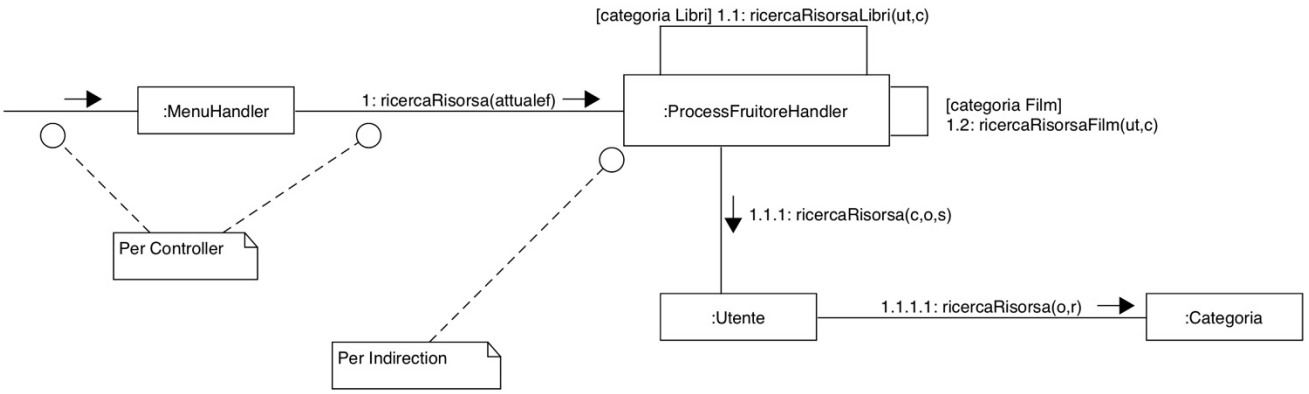
Post-condizioni: -la risorsa r è stata aggiunta all'elenco delle risorse della categoria.

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 processi relativi 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'*.

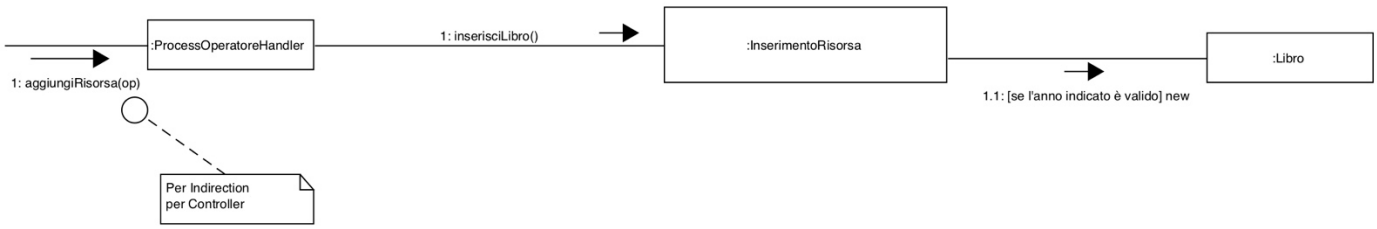
-Diagramma di comunicazione iscrizione



-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 LSP 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;
```

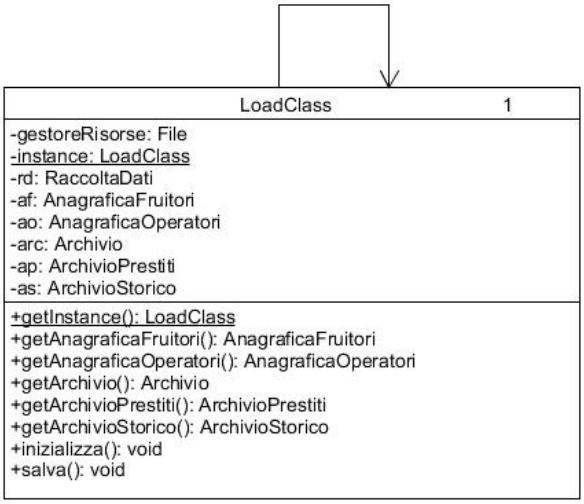
```
            case 2:o = InputDati.leggiStringa(Costanti.INS_COGNOME_AUTORE_LIBRO);
                   s = Categoria.RIC_PER_AUTORE_I;
                   break;

            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.

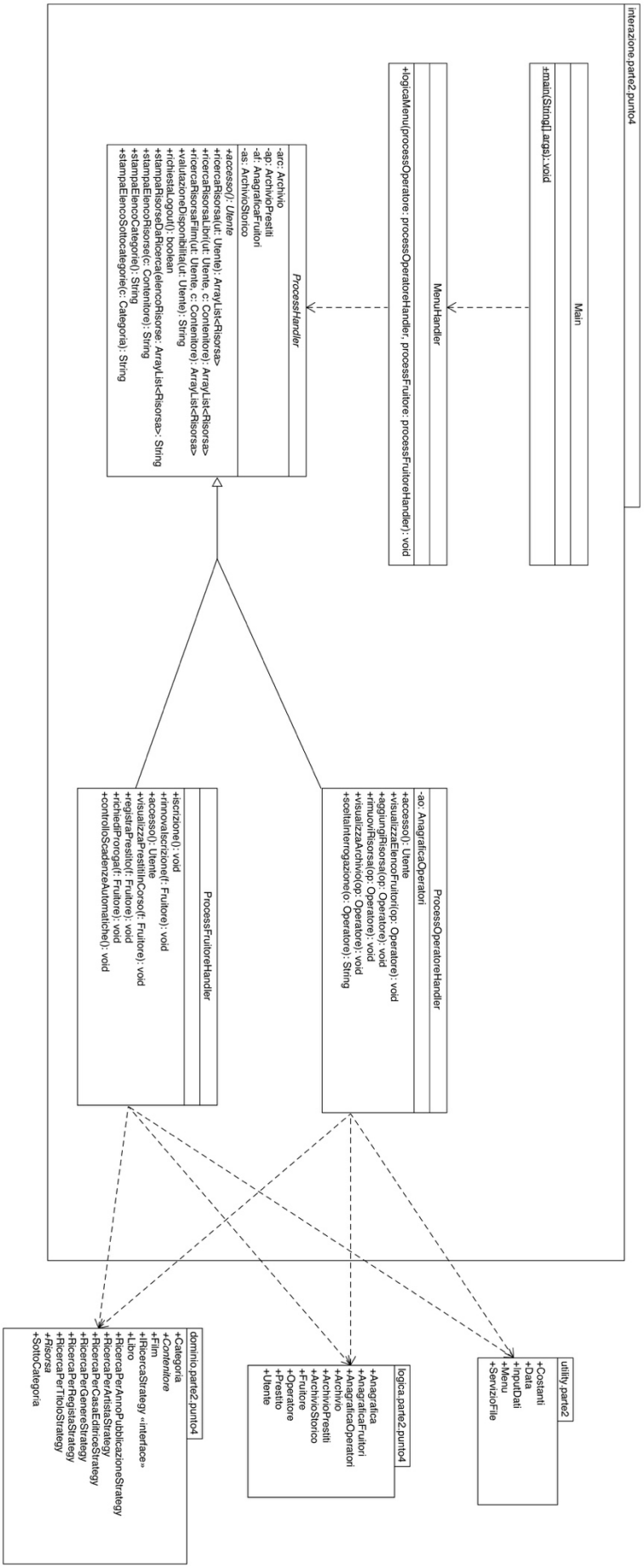
-Quarto punto

Per fare in modo di avere un solo esemplare di *Loadclass*, deputato al corretto reperimento e caricamento dei dati, è stato applicato il pattern *'Singleton'* definendo l'attributo statico *instance*, il costruttore e il metodo *getInstance()* e andando poi a rifattorizzare le chiamate dei metodi nel *main*.
La rivisitazione del codice secondo le linee guida espresse nei punti precedenti ha poi permesso allo stesso tempo l'applicazione dei pattern '*Strategy'* e *'Facade'*. Il primo riguarda le classi finalizzate alle varie tipologie di ricerca e fa in modo che ognuna di esse sia contenuta in una classe separata (al cui nome è stato aggiunto il suffisso -Strategy), la quale implementa l'interfaccia *IRicercaStrategy* con la firma del metodo interessato, mentre il secondo si focalizza sulla creazione dello strato aggiuntivo dei *Process*, il cui compito è proprio quello di smistare le richieste entranti agli specifici oggetti distinguendo dapprima i fruitori dagli operatori e coordinando successivamente l'interazione con i package logica e dominio.

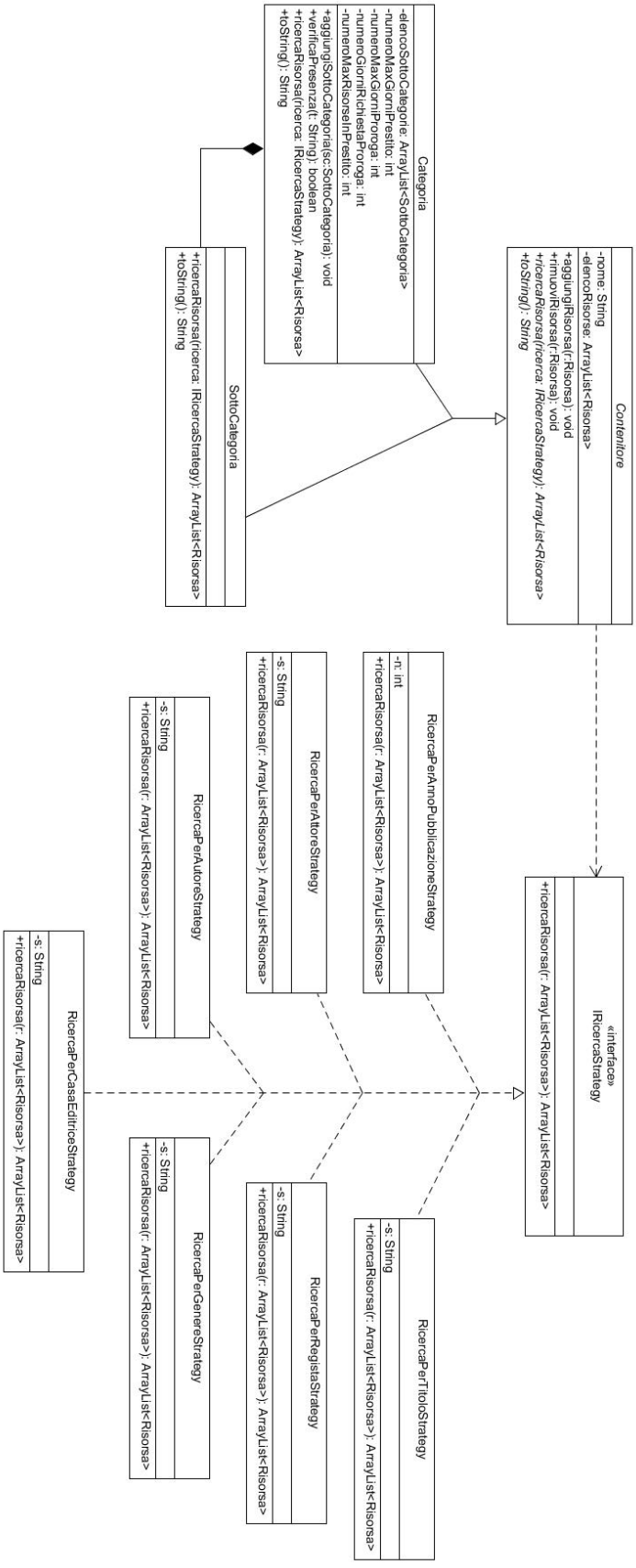
-Diagramma delle classi per il pattern Singleton



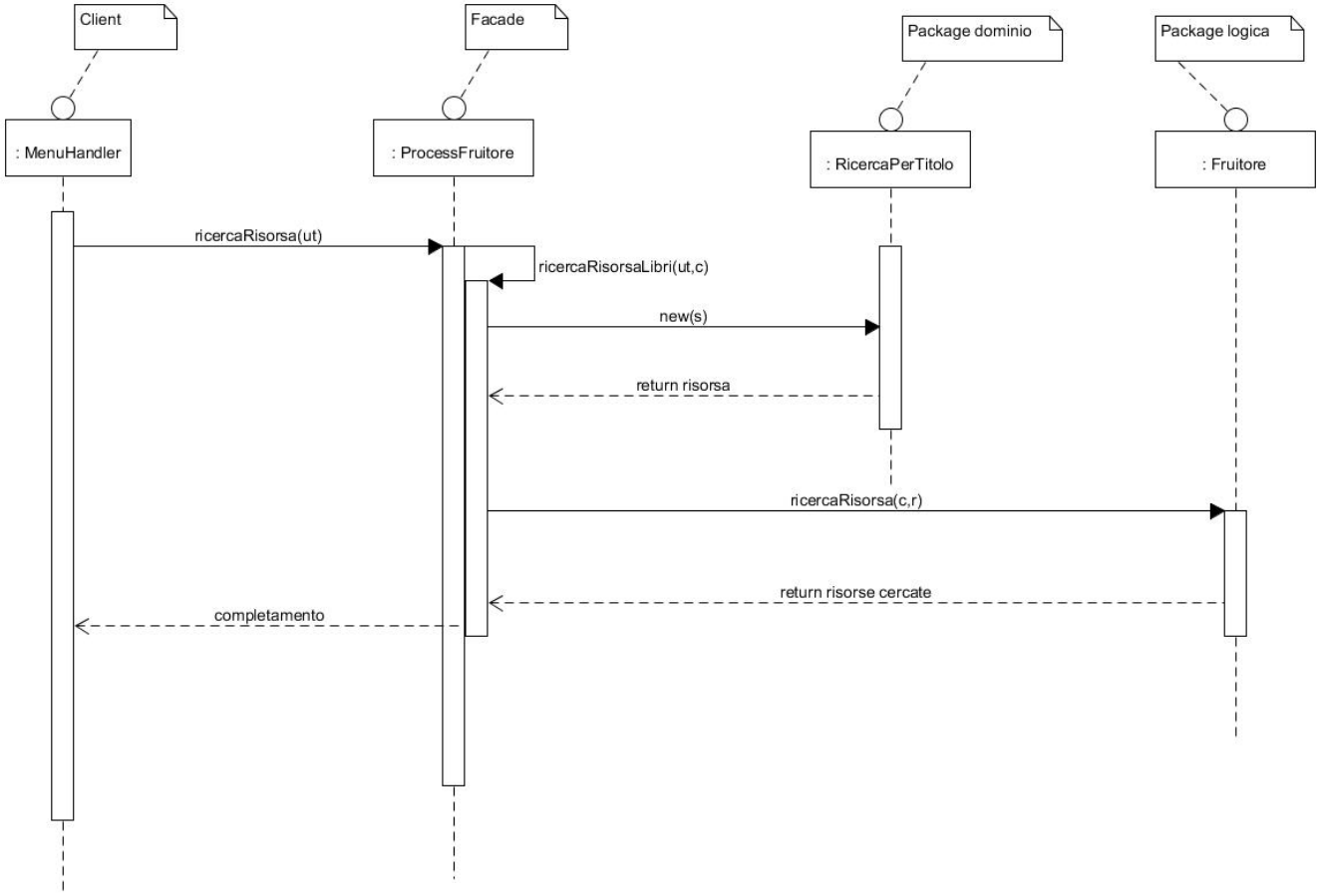
-Diagramma delle classi per il pattern Facade



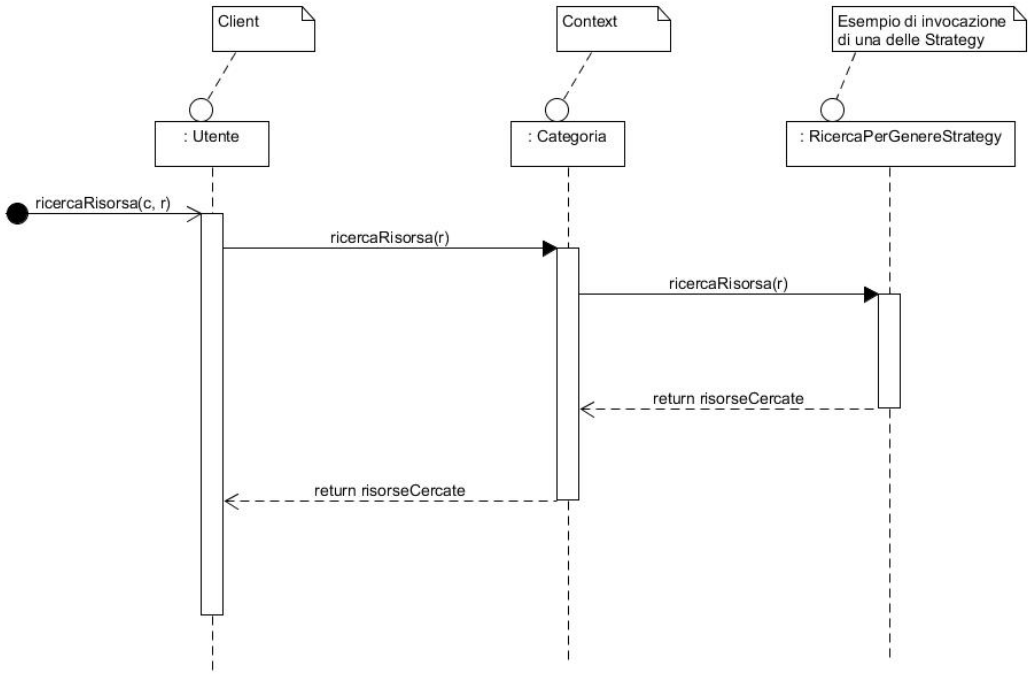
-Diagramma delle classi per il pattern Strategy



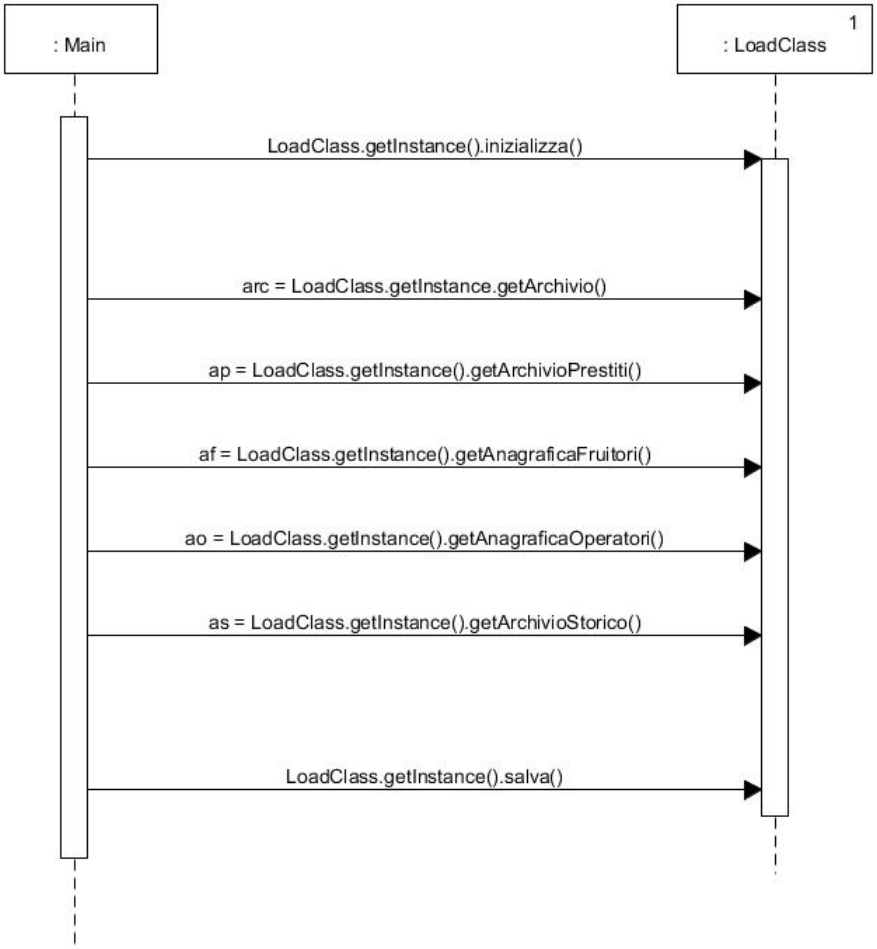
-Diagramma di sequenza per il pattern Facade



-Diagramma di sequenza per il pattern Strategy



-Diagramma di sequenza per il pattern Singleton



-Codice successivo al refactoring:

```
public class LoadClass {
       // In accordo con l'applicazione del pattern Singleton, è stato creato l'attributo
       // instance, il costruttore è vuoto ed è stato indicato il metodo statico getInstance()
        private static LoadClass instance;
        ...
        private LoadClass() {};

        public static LoadClass getinstance() {
               if(instance == null) {
                      instance = new LoadClass();
               }
               return instance;
        }
        ...
}

public class Main {
       public static void main(String[] args) {
        // E' stato applicato il pattern Singleton sulla LoadClass
        // andando a definire i getInstance() per reperire l'unica istanza presente
        LoadClass.getinstance().inizializza();

        Archivio arc = LoadClass.getinstance().getArchivio();
        ArchivioPrestiti ap = LoadClass.getinstance().getArchivioPrestiti();
        AnagraficaFruitori af = LoadClass.getinstance().getAnagraficaFruitori();
        AnagraficaOperatori ao = LoadClass.getinstance().getAnagraficaOperatori();
        ArchivioStorico as = LoadClass.getinstance().getArchivioStorico();

        ProcessOperatoreHandler processOperatore = new ProcessOperatoreHandler(arc, ap,
                                                                                af, ao, as);

         ProcessFruitoreHandler processFruitore = new ProcessFruitoreHandler(arc, ap, af, as);
         ...
       }
}
```

-Codice precedente al refactoring:

```
public class Main {
  public static void main(String[] args) {
       LoadClass loadclass = new LoadClass();
        Loadclass.inizializza();

      ProcessOperatoreHandler processOperatore = new
                                          ProcessOperatoreHandler(Loadclass.getArchivio(),
                        Loadclass.getArchivioPrestiti(), Loadclass.getAnagraficaFruitori(),
                        Loadclass.getAnagraficaOperatori(), Loadclass.getArchivioStorico());

      ProcessFruitoreHandler processFruitore = new
                                           ProcessFruitoreHandler(Loadclass.getArchivio(),
                        Loadclass.getArchivioPrestiti(), Loadclass.getAnagraficaFruitori(),
                        Loadclass.getArchivioStorico());
       ...
       }
}
```

All'interno della cartella punto4 è possibile trovare la sottocartella contenente il codice completo comprensivo di test.

-Quinto punto

Il primo pattern di refactoring applicato è l'*'Extract Method'* sui metodi *aggiungiRisorsa()* e *rimuoviRisorsa()* della classe *ProcessOperatoreHandler*, e *registraPrestito()* della classe *ProcessFruitoreHandler* andando a suddividere l'operazione complessa in sottounità distinte contenute nella medesima classe in modo da semplificare la struttura e l'organizzazione del codice migliorandone la leggibilità.
Successivamente, seguendo il pattern *'Preserve Whole Object'*, i vari oggetti responsabili della raccolta e conservazione dei dati (*AnagraficaFruitori, ArchivioPrestiti*, *ArchivioStorico*, …), che venivano trasmessi al *Main* dalla *LoadClass* e poi passati come parametri ai *Process* per il corretto funzionamento del sistema, sono stati trattenuti all'interno della *RaccoltaDati*, da cui invece erano in precedenza immediatamente prelevati, rimandando tale operazione nei *Process*, i quali possono così accettare un singolo oggetto come parametro, che contenga le varie tipologie di raccolta dei dati. Ciò ha permesso di ridurre le dipendenze tra queste classi ed il *Main*, preservando altresì l'integrità e l'incapsulamento dell'oggetto *RaccoltaDati*.
E' stato inoltre applicato il pattern *'Replace Data Value With Object'* sugli attributi autore e attore delle classi *Libro* e *Film*, che sono stati unificati per mezzo della classe *Persona*, la quale contiene anche il metodo necessario per la verifica della presenza di una stringa all'interno del nome o del cognome. Sono poi stati sostituiti dall'attributo artisti direttamente nella classe *Risorsa* ed è stato introdotto l'apposito *getArtisti()*, semplificando così la trattazione delle diverse categorie, in accordo con il principio *'Pull Up Method'*, e compattando allo stesso tempo le classi *RicercaPerAutoreStrategy* e *RicercaPerAttoreStrategy* nella singola classe *RicercaPerArtistaStrategy*.
Infine, come previsto dal pattern *'Introduce Foreign Method'*, per poter personalizzare al meglio la classe statica *LocalDate*, introducendo metodi più specifici e dettagliati sulla base del contesto del programma e al tempo stesso più leggibili, si è deciso di creare la classe *Data* (contenuta nel package '*utility.parte2*') contenente le varie operazioni che interessano la manipolazione delle date, di conseguenza sono state modificate tutte le invocazioni che riguardano la *Data* nelle classi *AnagraficaFruitori, ArchivioPrestiti, ArchivioStorico, Fruitore, Prestito, ProcessFruitoreHandler* e *ProcessOperatoreHandler.*

-Codice successivo al refactoring:

```
public class Data {
	//foreign method, should be in LocalDate
	public static boolean confrontaDate(LocalDate d1, LocalDate d2) {
		return d1.isEqual(d2);
	}

	//foreign method, should be in LocalDate
	public static boolean verificaDataSuccessiva(LocalDate d) {
		return LocalDate.now().isAfter(d);
	}

	//foreign method, should be in LocalDate
	public static boolean verificaDataPrecedente(LocalDate d) {
		return LocalDate.now().isBefore(d);
	}

	//foreign method, should be in LocalDate
	public static boolean verificaDataCoincidente(LocalDate d) {
		return LocalDate.now().equals(d);
	}

	//foreign method, should be in LocalDate
	public static LocalDate aumentaNumeroAnni(LocalDate d, int numeroAnni) {
		return d.plusYears(numeroAnni);
	}
	//foreign method, should be in LocalDate
	public static LocalDate aumentaNumeroGiorni(LocalDate d, int numeroGiorni) {
		return d.plusDays(numeroGiorni);
	}

	//foreign method, should be in LocalDate
	public static LocalDate diminuisciNumeroGiorni(LocalDate d, int numeroGiorni) {
		return d.minusDays(numeroGiorni);
	}

	//foreign method, should be in LocalDate
	public static LocalDate aumentaDataAttualeNumeroAnni(int numeroAnni) {
		return LocalDate.now().plusYears(numeroAnni);
	}

	//foreign method, should be in LocalDate
	public static LocalDate aumentaDataAttualeNumeroGiorni(int numeroGiorni) {
		return LocalDate.now().plusDays(numeroGiorni);
	}

	//foreign method, should be in LocalDate
	public static LocalDate diminuisciDataAttualeNumeroGiorni(int numeroGiorni) {
		return LocalDate.now().minusDays(numeroGiorni);
	}

	//foreign method, should be in LocalDate
	public static LocalDate diminuisciDataAttualeNumeroAnni(int numeroAnni) {
		return LocalDate.now().minusYears(numeroAnni);
	}

	//foreign method, should be in LocalDate
	public static LocalDate getDataAttuale() {
		return LocalDate.now();
	}

	//foreign method, should be in LocalDate
	public static LocalDate getDataImpostata(int anno, int mese, int giorno) {
		return LocalDate.of(anno, mese, giorno);
	}

	//foreign method, should be in LocalDate
	public static int getNumeroAnniDataAttuale() {
		return LocalDate.now().getYear();
	}
```

```
	//foreign method, should be in LocalDate
	public static int getNumeroAnniData(LocalDate d) {
		return d.getYear();
	}

	//foreign method, should be in LocalDate
	public static String getDataFormattata(LocalDate d) {
		DateTimeFormatter formatter = DateTimeFormatter.ofPattern(Costanti.FORMATO_DATA);
		return d.format(formatter);
	}
}

public class Film extends Risorsa implements Serializable{
	...
	//Il tipo dell'attributo e' diventato Persona in seguito all'applicazione del pattern
	//'Replace Data Value With Object'.
	private Persona regista;
	...
	public Film(String titolo, int licenze, String genere, int annoPub, String lingua,
			Persona reg, ArrayList <Persona> a, int dm){

	    super(titolo, licenze, a, genere, annoPub, lingua);
	     ...
	}
	...
	public String toString(){
	    ...
	    StringBuffer artisti = new StringBuffer();

	    for(int i = 0; i < getArtisti().size(); i++){
			artisti.append(getArtisti().get(i).toString());
			if(i < getArtisti().size()-1)
				artisti.append(", ");
	     }

	    ris.append(String.format(DESCRIZIONE_FILM, getTitolo(), regista.toString(), artisti,
				getGenere(), getLingua(), durataInMinuti, getAnnoPub(), getNumLicenze()));
	    ...
	}
}

public class Libro extends Risorsa implements Serializable{
	...
	public Libro(String titolo, int licenze, String genere, int annoPub, String lingua,
			ArrayList <Persona> a, int np, String ce){

	     super(titolo, licenze, a, genere, annoPub, lingua);
	      ...
	}
	...
	public String toString(){
	   ...
	   StringBuffer artisti = new StringBuffer();

	  for(int i = 0; i < getArtisti().size(); i++){
			artisti.append(getArtisti().get(i).toString());
			if(i < getArtisti().size()-1)
				artisti.append(", ");
	  }

	  ris.append(String.format(DESCRIZIONE_LIBRO, getTitolo(), artisti, numPagine,
				getAnnoPub(), casaEditrice, getLingua(), getGenere(), getNumLicenze()));
	   ...
	}
}
```

```
public class LoadClass implements Serializable {
	...
	public RaccoltaDati getRaccoltaDati(){
		return rd;
	}

	public void inizializza() {
	  ...
	   if (!caricamentoRiuscito){
		...
		rd = new RaccoltaDati(af, ao, arc, ap, as);
		...
	  }
	}
	...
}

public class Main {
	public static void main(String[] args) {
		...
		RaccoltaDati rd = LoadClass.getinstance().getRaccoltaDati();

		//Applicazione dei pattern di Refactoring 'Preserve Whole Object':
		//'al costruttore dei due process viene passato
		//l'intero oggetto RaccoltaDati.
		ProcessOperatoreHandler processOperatore = new ProcessOperatoreHandler(rd);
		ProcessFruitoreHandler processFruitore = new ProcessFruitoreHandler(rd);
		...
	}
}

/*
 * E' stata introdotta la classe Persona applicando il pattern di Refactoring
 * 'Replace Data Value With Object'.
 */
public class Persona implements Serializable {
	...
	private String nome;
	private String cognome;

	public static final String DESCRIZIONE_PERSONA = "%s  %s";

	public Persona(String nome, String cognome){
		this.nome = nome;
		this.cognome = cognome;
	}

	public String getNome(){
		return nome;
	}

	public String getCognome(){
		return cognome;
	}

	public boolean verificaPresenzaStringa(String t){
		if((nome.toLowerCase().indexOf((t).toLowerCase()) > -1) ||
		    (cognome.toLowerCase().indexOf((t).toLowerCase()) > -1) ||
		      ((nome + " " + cognome).toLowerCase().indexOf((t).toLowerCase()) > -1))
				return true;
		else
		   return false;
	}

	public String toString(){
		StringBuffer ris = new StringBuffer();
		ris.append(String.format(DESCRIZIONE_PERSONA, getNome(), getCognome()));
		return ris.toString();
	}
}
```

```
public abstract class Risorsa implements Serializable{
	...
	//Con l'introduzione della classe Persona, l'attributo e' diventato comune
	//ad entrambe le sottoclassi quindi e' stato applicato il pattern di
	//Refactoring 'Pull Up Field'.
	private ArrayList <Persona> artisti;
	...
	public Risorsa(String t, int lic, ArrayList <Persona> a, String g, int ap, String l){
		...
		this.artisti = a;
		...
	}
	...
	//Applicazione del pattern di Refactoring 'Pull Up Method'.
	public ArrayList <Persona> getArtisti(){
		return artisti;
	}
	...
	public boolean verificaPresenzaArtista(String t){
		for(int i = 0; i < artisti.size(); i++){
			Persona p = artisti.get(i);
			if(p.verificaPresenzaStringa(t))
				return true;
		}
		return false;
	}
	...
}

public class RicercaPerRegistaStrategy implements IRicercaStrategy, Serializable{
	...
	public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris) {
		...
		for(int i = 0; i < elencoris.size(); i++){
			...
			if(((Film)ris).getRegista().verificaPresenzaStringa(s))
				...
		}
		...
	}
}

/*
 * Le due classi RicercaPerAttoreStrategy e RicercaPerAutoreStrategy sono
 * diventate un'unica classe in seguito all'introduzione della classe Persona
 */
public class RicercaPerArtistaStrategy implements IRicercaStrategy, Serializable {
	...
	public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris) {
		...
		for(int i = 0; i < elencoris.size(); i++) {
			...
			if(ris.verificaPresenzaArtista(s))
				...
		}
		...
	}
}

public abstract class ProcessHandler implements Serializable {
	...
	//Applicazione dei pattern di Refactoring 'Preserve Whole Object':
	//al costruttore viene passato l'intero oggetto RaccoltaDati.
	public ProcessHandler(RaccoltaDati rd){
		this.arc = rd.getArchivio();
		this.ap = rd.getArchivioPrestiti();
		this.af = rd.getAnagraficaFruitori();
		this.as = rd.getArchivioStorico();
	}
	...
}
```

```
public class ProcessFruitoreHandler extends ProcessHandler implements Serializable {
	...
	//Applicazione dei pattern di Refactoring 'Preserve Whole Object':
	//al costruttore viene passato l'intero oggetto RaccoltaDati.
	public ProcessFruitoreHandler(RaccoltaDati rd){
		super(rd);
	}
	//Metodo introdotto in seguito all'applicazione del pattern di Refactoring 'Extract
	//Method'al metodo registraPrestito.
	public void registraPrestitoSenzaSottoCategorie(Fruitore f, Categoria c, Risorsa r,
                                                        Prestito nuovo) {
		...
	}
	//Metodo introdotto in seguito all'applicazione del pattern di Refactoring 'Extract
	// Method'al metodo registraPrestito.
	public void registraPrestitoConSottoCategorie(Fruitore f, Categoria c, Risorsa r,
                                                      Prestito nuovo, SottoCategoria sc) {
		...
	}
	//Metodo modificato sulla base dell'applicazione del pattern di Refactoring
	//'Extract Method'.
	public void registraPrestito(Fruitore f) {
		 ...
	}
	...
}

public class ProcessOperatoreHandler extends ProcessHandler implements Serializable {
	...
	//Applicazione dei pattern di Refactoring 'Preserve Whole Object':
	//al costruttore viene passato l'intero oggetto RaccoltaDati.
	public ProcessOperatoreHandler(RaccoltaDati rd){
		super(rd);
		this.ao = rd.getAnagraficaOperatori();
	}
	...
	//Metodo introdotto in seguito all'applicazione del pattern di Refactoring 'Extract
	// Method'al metodo aggiungiRisorsa.
	public void aggiungiRisorsaCategoriaSenzaSottoCategorie(Operatore op, Categoria c,
                                                                Risorsa nuovar){
		...
	}
	//Metodo introdotto in seguito all'applicazione del pattern di Refactoring 'Extract
	// Method'al metodo aggiungiRisorsa.
	public void aggiungiRisorsaCategoriaConSottoCategorie(Operatore op, Categoria c,
                                                              SottoCategoria sc, Risorsa nuovar){
		...
	}
	//Metodo modificato sulla base dell'applicazione del pattern di Refactoring
	//'Extract Method'.
	public void aggiungiRisorsa(Operatore op){
		...
	}
	//Metodo introdotto in seguito all'applicazione del pattern di Refactoring 'Extract
	// Method'al metodo rimuoviRisorsa.
	public void rimuoviRisorsaCategoriaSenzaSottoCategorie(Operatore op, Categoria c,
                                                               Risorsa daEliminare){
		...
	}
	//Metodo introdotto in seguito all'applicazione del pattern di Refactoring 'Extract
	// Method' al metodo rimuoviRisorsa.
	public void rimuoviRisorsaCategoriaConSottoCategorie(Operatore op, Categoria c,
                                                             SottoCategoria sc, Risorsa daEliminare){
		...
	}
	//Metodo modificato sulla base dell'applicazione del pattern di Refactoring
	//'Extract Method'.
	public void rimuoviRisorsa(Operatore op){
		 ...
	}
	...
}
```

-Codice precedente al refactoring:

```
public class Film extends Risorsa implements Serializable {
	...
	private String regista;
	private ArrayList <String> attore_i;
	...
	public String getAttore(){
		StringBuffer att= new StringBuffer();

		for(int i = 0; i < attore_i.size(); i++) {
		att.append(attore_i.get(i));

		if(i < attore_i.size()-1)
				att.append("-");
		}

		return att.toString();
	}
	...
	public String toString(){
		...
		String att = getAttore();
		ris.append(String.format(DESCRIZIONE_FILM, getTitolo(), regista, att,
		getGenere(), getLingua(), durataInMinuti, getAnnoPub(), getNumLicenze()));
		...
	}
}

public class Libro extends Risorsa implements Serializable {
	...
	private ArrayList <String> autore_i;
	...
	public String getAutore(){
		StringBuffer aut = new StringBuffer();

		for(int i = 0; i < autore_i.size(); i++) {
			aut.append(autore_i.get(i));
			if(i < autore_i.size()-1)
					aut.append("-");
		}
		return aut.toString();
	}
	...
	public String toString(){
		...
		String aut = getAutore();
		ris.append(String.format(DESCRIZIONE_LIBRO, getTitolo(), aut, numPagine,
				getAnnoPub(), casaEditrice, getLingua(), getGenere(), getNumLicenze()));
		...
	}
}

public class LoadClass {
	...
	public void inizializza() {
		...
		if (!caricamentoRiuscito) {
			System.out.println(Costanti.MSG_NO_FILE);
			af = new AnagraficaFruitori();
			ao = new AnagraficaOperatori();
			arc = new Archivio();
			ap = new ArchivioPrestiti();
			as = new ArchivioStorico();

			StrutturaSistema.aggiuntaOperatoriPreimpostati(ao);
			StrutturaSistema.creazioneStrutturaArchivioLibri(arc);
			StrutturaSistema.creazioneStrutturaArchivioFilm(arc);
		}
	}
```

```
	public AnagraficaFruitori getAnagraficaFruitori() {
		return af;
	}

	public AnagraficaOperatori getAnagraficaOperatori() {
		return ao;
	}

	public Archivio getArchivio() {
		return arc;
	}

	public ArchivioPrestiti getArchivioPrestiti() {
			return ap;
	}

	public ArchivioStorico getArchivioStorico() {
		return as;
	}
	...
}

public class Main {
	public static void main(String[] args) {
	  ...
	  Archivio arc = LoadClass.getinstance().getArchivio();
	  ArchivioPrestiti ap = LoadClass.getinstance().getArchivioPrestiti();
	  AnagraficaFruitori af = LoadClass.getinstance().getAnagraficaFruitori();
	  AnagraficaOperatori ao = LoadClass.getinstance().getAnagraficaOperatori();
	  ArchivioStorico as = LoadClass.getinstance().getArchivioStorico();

	  ProcessOperatoreHandler processOperatore = new ProcessOperatoreHandler(arc, ap, af,
	                                                                          ao, as);

	   ProcessFruitoreHandler processFruitore = new ProcessFruitoreHandler(arc, ap, af, as);
	    ...
	}
}

public abstract class Risorsa implements Serializable {
	...
	public Risorsa(String t, int lic, String g, int ap, String l){
		this.titolo = t;
		this.numLicenze = lic;
		this.genere = g;
		this.annoPub = ap;
		this.lingua = l;
	}
	...
}

public class RicercaPerRegistaStrategy implements IRicercaStrategy, Serializable {
	...
	public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris) {
		...
		for(int i = 0; i < elencoris.size(); i++) {
			...
			if(((Film)ris).getRegista().toLowerCase().indexOf((s).toLowerCase()) > -1)
				...
		}
		...
	}
}

public class RicercaPerAttoreStrategy implements IRicercaStrategy, Serializable {
	...
	public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris) {
		...
		for(int i = 0; i < elencoris.size(); i++) {
			...
			if(((Film)ris).getAttore().toLowerCase().indexOf((s).toLowerCase()) > -1)
				...
```

```
            }
            ...
      }
}

public class RicercaPerAutoreStrategy implements IRicercaStrategy, Serializable {
      ...
      public ArrayList <Risorsa> ricercaRisorsa(ArrayList <Risorsa> elencoris) {
            ...
            for(int i = 0; i < elencoris.size(); i++) {
                  ...
                  if(((Libro)ris).getAutore().toLowerCase().indexOf((s).toLowerCase()) > -1)
                        ...
            }
            ...
      }
}

public abstract class ProcessHandler implements Serializable {
      ...
      public ProcessHandler(Archivio arc, ArchivioPrestiti ap, AnagraficaFruitori af,
                              ArchivioStorico as){
            this.arc = arc;
            this.ap = ap;
            this.af = af;
            this.as = as;
      }
      ...
}

public class ProcessFruitoreHandler extends ProcessHandler implements Serializable {
      ...
      public ProcessFruitoreHandler(Archivio arc, ArchivioPrestiti ap, AnagraficaFruitori af,
                                    ArchivioStorico as) {
            super(arc, ap, af, as);
      }
      ...
      public void registraPrestito(Fruitore f) {
            ...
      }
      ...
}

public class ProcessOperatoreHandler extends ProcessHandler implements Serializable {
      ...
      public ProcessOperatoreHandler(Archivio arc, ArchivioPrestiti ap, AnagraficaFruitori af
                                    , AnagraficaOperatori ao, ArchivioStorico as){
            super(arc, ap, af, as);
            this.ao = ao;
      }
      ...
      public void aggiungiRisorsa(Operatore op){
            ...
      }
      public void rimuoviRisorsa(Operatore op){
            ...
      }
      ...
}
```

All'interno della cartella punto5-versione definitiva è possibile trovare la sottocartella contenente il codice completo definitivo a seguito dell'attività di refactoring di tutti i cinque punti della consegna comprensivo di test.