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'*.

All'interno della cartella punto1 è possibile trovare la cartella contenente i diagrammi UML.

-Secondo punto

Sono stati realizzati i diagrammi di sequenza di sistema dei seguenti casi d'uso: Iscrizione, Ricerca risorsa e Aggiunta risorsa; poi sono stati definiti i contratti per alcune delle operazioni di sistema in essi presenti e 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'*.

All'interno della cartella punto2 è possibile trovare le quattro sottocartelle distinte contenenti il codice completo, i contratti, i diagrammi SSD ed i diagrammi UML di Comunicazione.
I test si trovano in un package apposito nel codice completo.

-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()* 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.

All'interno della cartella punto3 è possibile trovare le due sottocartelle distinte contenenti il codice completo e le porzioni di codice utili per la discussione.
I test si trovano in un package apposito nel codice completo.

-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.

All'interno della cartella punto4 è possibile trovare le tre sottocartelle distinte contenenti il codice completo, i diagrammi UML e le porzioni di codice utili per la discussione.
I test si trovano in un package apposito nel codice completo.

-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 i pattern *'Preserve Whole Object'* e *'Introduce Parameter 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 i principii *'Pull Up Field'* e *'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.

All'interno della cartella punto5-versione definitiva è possibile trovare le due sottocartelle contenenti il codice completo definitivo a seguito dell'attività di refactoring di tutti i cinque punti della consegna e le porzioni di codice utili per la discussione.
I test si trovano in un package apposito nel codice completo.