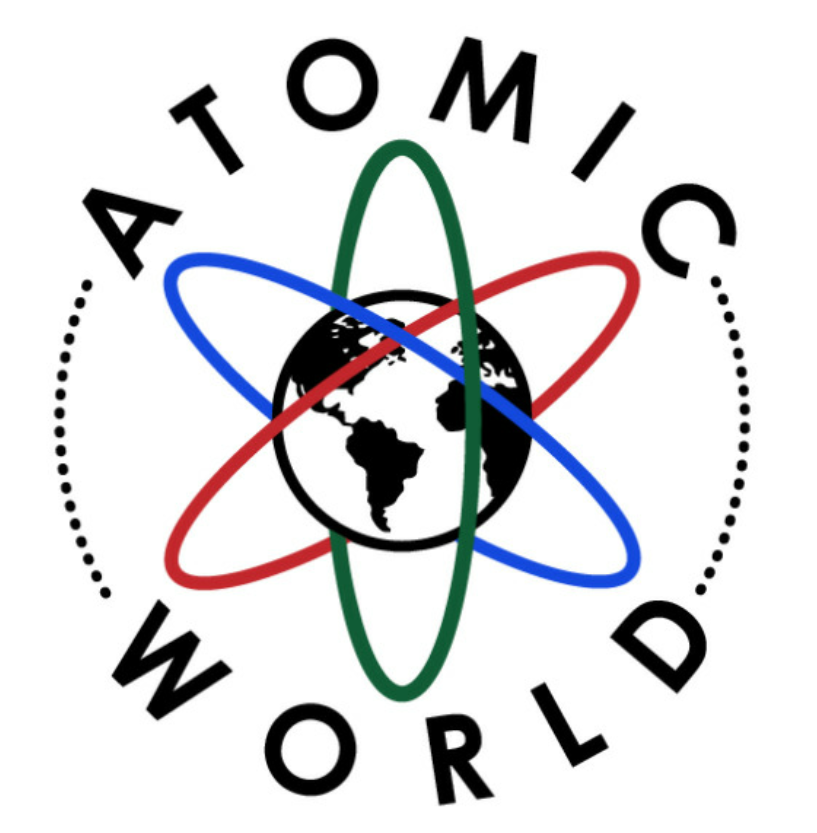
**Atomic World**

**Esposizione del problema:**

Creare un’applicazione con un’interfaccia grafica per la vendita di un particolare prodotto. Deve essere possibile visionare le foto dei prodotti e sceglierli; visualizzare il costo totale dei prodotti acquistati. Utilizzare almeno una finestra di dialogo.

**Requisiti:**

1. Analisi del problema  
   Introdurre il progetto; specificare come si è scelto di impostare l’interfaccia; tracciare UML; esporre ipotesi aggiuntive.

A questo punto produrre un documento delle specifiche per richiedere una consulenza nell'indirizzo grafica comunicazione riguardante, ad esempio:

- logo dell'azienda;

- font, colori , allineamenti, disposizione dei componenti;

- icone personalizzate (es. carrello, cestino, pulsanti vari);

- ritocchi di immagini o fotografici.

1. Disegno del’interfaccia grafica  
   Disegnare l'interfaccia; descrivere brevemente i contenitori ed i componenti usati; definire le proprietà delle componenti; scegliere un gestore di layout; riassumere in una tabella i nomi degli oggetti utilizzati nell’interfaccia grafica con la relativa classe.
2. Gestione degli eventi  
   Come interagisce l’utente con il programma? Cosa succede in seguito a particolari azioni dell’utente?
3. Programma

**Descrizione dell'impostazione del lavoro:**

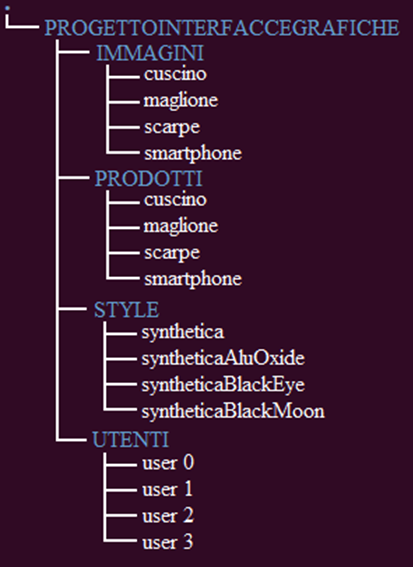
Il gruppo era formato da:

* Luca Miccioni: coordinatore, programmatore, aiuto-analista
* Filippo Mantovani: analista, UML
* Francesco Corbucci: aiuto-programmatore
* Luca Sosti: programmatore interfaccia grafica
* Alex Los: grafico

Dopo un'attenta analisi del problema presentatosi si è deciso di sviluppare l'applicazione simile all'ormai noto sito Amazon, per quanto riguarda la sua interfaccia utente. Abbiamo quindi proceduto all'individuazione dei principali punti di forza e delle caratteristiche principali del nostro programma, ovvero le varie interfacce, la gestione dell'utenza e dei prodotti forniti. Quindi, nel nostro specifico caso, l'applicativo ha il compito di far visualizzare tutti i prodotti disponibili nel "mercato" e il loro acquisto, da parte di un utente registrato ed autenticato, tutto racchiuso in un'interfaccia semplice ed intuitiva. Il concetto del nostro applicativo gira attorno al fatto che un qualsiasi utente si possa registrare ed autenticare in un qualsiasi momento per acquistare i prodotti che più desidera, sempre che abbia un saldo per coprire le spese. Inoltre è stato richiesto l'aiuto di grafici per quanto riguarda la maggior parte della grafica. Ad essi è stato chiesto di realizzare:

* logo dell'applicazione
* icona carrello
* icona aggiunta prodotto al carrello
* icona ricerca prodotto

**Premessa e spiegazione funzionamento:**

Organizzazione spazio di lavoro:

Oltre ai classici file e cartelle che compongono un progetto Java, all'interno della directory del progetto ne sono state aggiunte altre quattro:

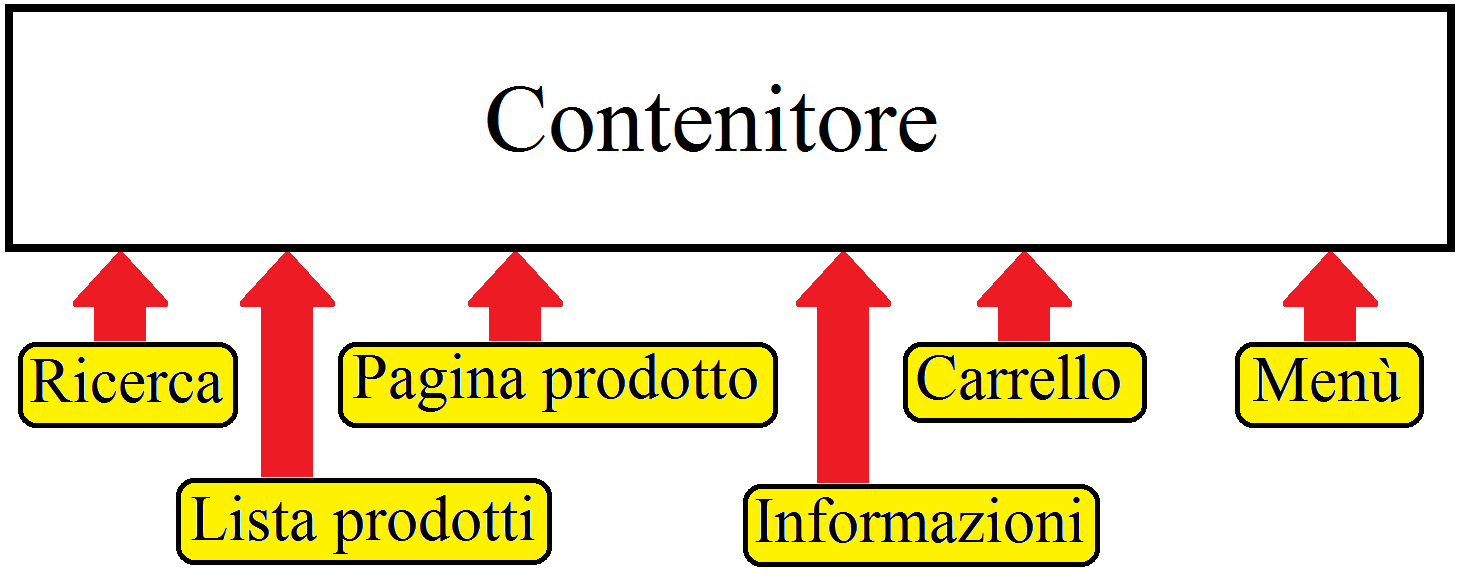
* Immagini: all'interno della quale si trovano tutte le immagini relative ai vari prodotti
* Prodotti: la quale contiene tutti i prodotti in vendita, successivamente verrà mostrato la struttura di ciascun file
* Style: cartella contenente tutti gli stili dell'applicazione, ovvero i vari look and feel, i quali ammontano a diciassette, tutti del tipo Synthetica
* Utenti: nella quale sono salvati tutti i dati relativi agli utenti registrati, anche quest'ultimo verrà descritto in seguito

Descrizione interfacce utilizzate:

Prima di passare alla descrizione è doveroso affermare che le interfacce sono state costruite a partire da quelle più esterne, per poi arrivare ai singoli componenti atomici. La disposizione di tali componenti è stata dettata sia da una scelta di buona visualizzazione sia da una realizzazione di un'interfaccia semplice da gestire ed utilizzare.

Le interfacce grafiche principali utilizzate nel progetto sono state:

* Un'interfaccia di contenimento, chiamata GUI\_Overview, che rappresenta la finestra principale che conterrà poi tutti i suoi sottopannelli e barre principali. Essa è anche usata come mezzo di comunicazione tra i vari pannelli, come mostra la figura sottostante
* Un pannello, chiamato GUI\_SearchPanel, usato come header fisso dell'applicazione. Al suo interno troviamo tutti quei componenti destinati alla ricerca più specifica dei prodotti, come la categoria o il nome
* Un altro pannello di nome GUI\_ProductsList dove vengono mostrati i risultati di ricerca del pannello precedente, ovvero dove verranno visualizzati tutti i prodotti che soddisfano i criteri di ricerca
* Un pannello chiamato GUI\_Product nel quale verrà mostrato il prodotto selezionato dal pannello appena descritto. Tutti i dati del prodotto sono riportati in questo pannello. Da qui è inoltre possibile aggiungere al carrello il prodotto
* Un pannello chiamato GUI\_Info dove si possono revisionare tutti i dati dell'utente autenticato
* Un pannello chiamato GUI\_Cart che contiene tutti gli oggetti aggiunti al carrello dall'utente
* Una barra di menù chiamata GUI\_MenuBar dalla quale vi si può accedere, registrare o uscire dal programma

Schema comunicazione pannelli:

Ognuno dei componenti interni (componenti gialli) al contenitore quando ha la necessità di comunicare con un altro componente fa una chiamata al contenitore comunicandogli con chi deve comunicare e cosa devi dirgli. Facciamo un esempio pratico: se dal pannello di ricerca l'utente immette i dati relativi la ricerca di determinati prodotti e vuole visualizzarli, il pannello stesso dirà al contenitore di far visualizzare il pannello dei risultati, la lista prodotti, indicandogli tutti i prodotti forniti dal risultato della propria ricerca.

Struttura file:

* Prodotti (descrizione campi dei file prodotti):
  + Nome: il nome del prodotto, che sarà poi quello visualizzato
  + Categoria: categoria del prodotto, dalla quale si costruiranno tutte le categorie per la ricerca
  + Descrizione: descrizione del prodotto
  + Prezzo: prezzo del prodotto, riportato a float
* Utenti (descrizione campi dei file utenti):
  + Nome
  + Cognome
  + Email: email dell'utente, che servirà poi per autenticarsi. Ogni utente ha la propria email e quindi non possono esistere più utenti con la stessa
  + Password: password dell'utente, che anch'essa servirà per l'autenticazione. Inoltre si era pensato ad un sistema di crittografia nel salvataggio della password ma, avendo poche conoscenze a riguardo, non preservando l'aspetto della sicurezza per un progetto solo client e per la mancanza di tempo si è pensato di evitarlo
  + Saldo: denaro dell'utente, servirà poi per l'acquisto dei prodotti

Gestione eventi:

L'utente è in grado di interagire con il programma tramite l'interfaccia grafica. Tutte le azioni che esso può compiere, tra cui click, ridimensionamenti e quant'altro, vengono gestite tramite i listener. Quelli utilizzati sono stati:

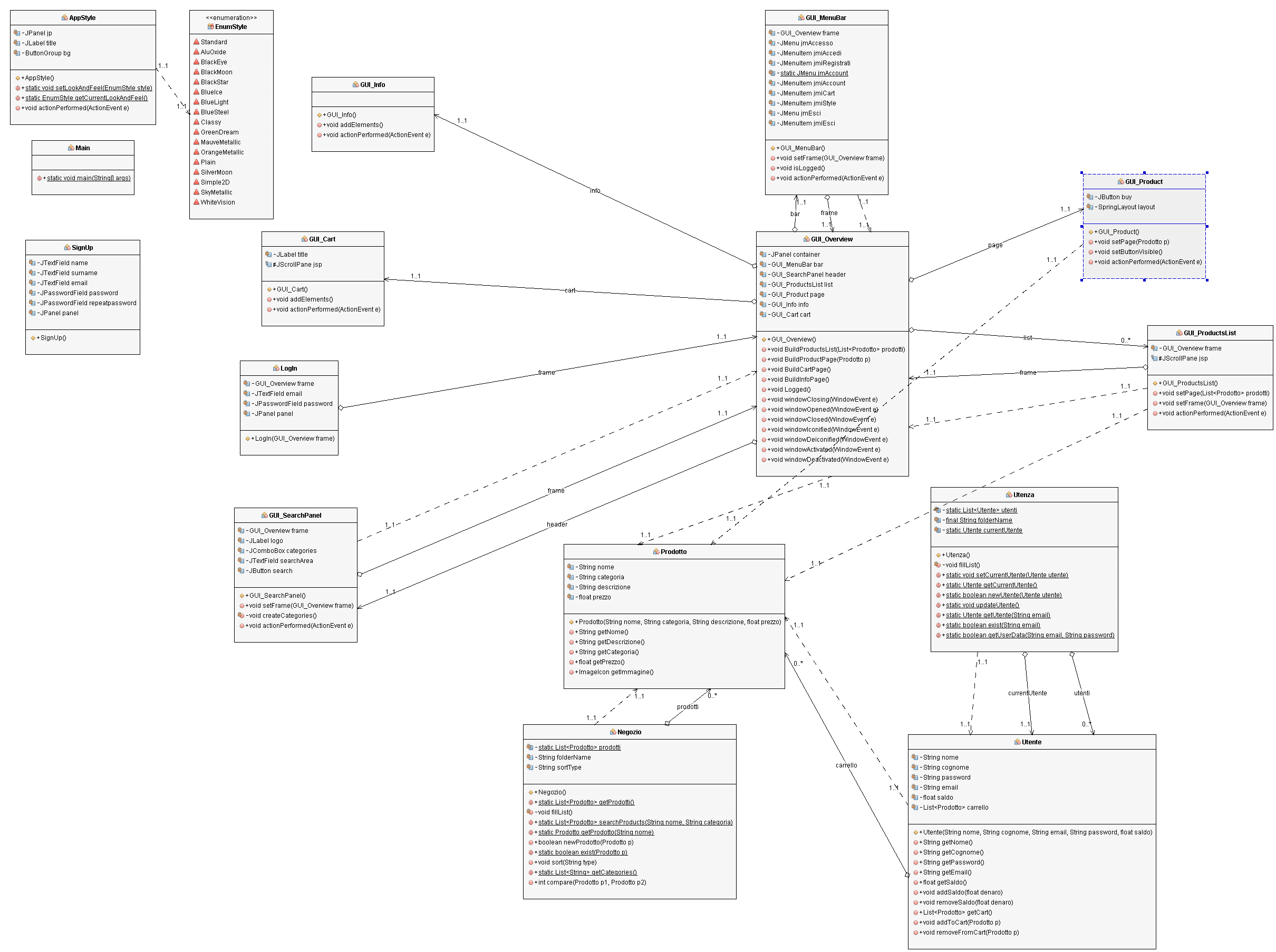
* ActionListener: gestore eventi relativi a tutti i click su quei componenti come bottoni o elementi delle barre dei menù
* WindowListener: gestore eventi relativi chiusura delle finestre. Ne è stato sfruttato soltanto uno, quello relativo la chiusura effettiva della finestra

**Analisi classi:**

Scomposizione classi per argomenti:

* Gestione prodotti
* Gestione utenti
* Stile
* Interfacce
* Main

UML:



1 - UML generato automaticamente tramite il plugin easyUML generator

1. Gestione prodotti:

* Classe prodotto: questa classe è il modello principale del prodotto, ovvero dove vengono definite tutte le sue caratteristiche e tutte le sue azioni
  + Attributi (tutti i suoi attributi sono già stati esaminati):
    - String nome
    - String categoria
    - String descrizione
    - float prezzo
  + Metodi:
    - Prodotto(String nome, String categoria, String descrizione, float prezzo): metodo costruttore della classe che inizializza ed imposta un valore agli attributi della classe
    - String getNome(): metodo getter che ritorna il nome del prodotto
    - String getCategoria(): metodo getter che ritorna la descrizione del prodotto
    - String getDescrizione(): metodo getter che ritorna la descrizione del prodotto
    - float getPrezzo(): metodo getter che ritorna il prezzo del prodotto
    - ImageIcon getImmagine(): metodo getter che ritorna l’immagine del prodotto a partire dal suo nome. Infatti i file all’interno della cartella immagini hanno gli stessi nomi di quelli nella cartella prodotti
* Classe Negozio: classe che gestisce tutti i prodotti disponibili nel mercato. Per la realizzazione di questo ci siamo serviti di una lista gestita dinamicamente da Java (List e ArrayList). La classe implementa l'interfaccia Comparator per l'implementazione dell'ordinamento della lista interna alla classe. Essa ha molti metodi statici in quanto la si è voluta rendere una classe accessibile da chiunque
  + Attributi:
    - static List<Prodotto> prodotti: la lista è dove vengono contenuti effettivamente tutti i prodotti disponibili. Essa è statica perché vi si può accedere da qualsiasi classe e per il corretto funzionamento di alcuni metodi interni alla classe. Si è scelto di utilizzare una lista e non un’array perché la sua gestione è sicuramente più complessa con una grande base di dati
    - final String folderName: questo attributo, praticamente una costante, serve per riferirsi alla cartella del progetto contenente i prodotti
    - String sortType: indica il tipo di ordinamento da applicare alla lista di prodotti, quando viene richiamato il metodo opportuno
  + Metodi:
    - Negozio(): costruttore della classe che inizializza la lista e la riempe, tramite il metodo fillList()
    - static List<Prodotto> getProdotto(): metodo getter che ritorna la lista di prodotti
    - void fillList(): metodo che per ogni file all'interno della cartella contenente i prodotti crea un Prodotto che immetterà poi nella lista. Se la cartella non esiste verrà opportunamente creata
    - static List<Prodotto> searchProducts(String nome, String categoria): metodo utilizzato per la costruzione dei risultati di ricerca. Se il nome e la categoria passate al metodo corrispondono con un certo prodotto allora esso sarà aggiunto alla lista dei risultati da visualizzare
    - static Prodotto getProdotto(String nome): metodo che restituisce un prodotto conoscendo soltanto il suo nome. La ricerca si ferma al primo riscontro
    - boolean newProdotto(Prodotto p): metodo che crea un nuovo prodotto e ne crea il relativo file contenente i suoi dati, oltre ad essere aggiunto nella lista corrente. Il tutto avviene se e soltanto se il Prodotto non è già esistente
    - static boolean exist(Prodotto p): metodo che controlla se un determinato prodotto esiste già, confrontando il suo nome con quelli presenti nella lista
    - void sort(String type): metodo che ordina la lista secondo il criterio specificato secondo il parametro passatogli. Esso può essere alfabetico, crescente o decrescente
    - static List<Prodotto> getCategories(): metodo che, a partire dalle categorie di tutti i prodotti, prese una sola volta, costruisce e restituisce una lista con quest'ultimi valori
    - int compare(Prodotto p1, Prodotto p2): sovraccarico del metodo compare dell'interfaccia Comparator. Ha la funzione di confrontare due prodotti secondo il loro prezzo, crescente e decrescente

1. Gestione utenti:
   * classe Utente: classe di base per la gestione dell'utenza. Modello per la definizione di un utente
     + Attributi (tutti i suoi attributi sono già stati esaminati):
       - String nome
       - String cognome
       - String password
       - String email
       - float saldo
       - List<Prodotto> carrello
     + Metodi:
       - Utente(String nome, String cognome, String password, float saldo): metodo costruttore della classe che imposta i valori ai vari attributi ed inizializza il carrello
       - String getNome(): metodo getter che ritorna il nome dell'utente
       - String getCognome(): metodo getter che ritorna il cognome dell'utente
       - String getPassword(): metodo getter che ritorna la password dell'utente
       - float getSaldo(): metodo getter che ritorna il saldo dell'utente
       - void addSaldo(float denaro): aggiunge il denaro specificato al saldo dell'utente
       - void removeSaldo(float denaro): rimuove il denaro specificato dal saldo dell'utente
       - List<Prodotto> getCart(): metodo getter che ritorna il carrello dell'utente
       - void addToCart(Prodotto p): metodo che aggiunge un prodotto al carrello, e quindi alla lista, dell'utente
       - void removeFromCart(Prodotto p): metodo che rimuove un prodotto dal carrello, e quindi dalla lista, dell'utente
   * classe Utenza: classe che gestisce tutti gli utenti registrati e quello correntemente autenticato. Essa ha molti metodi statici in quanto la si è voluta rendere una classe accessibile da chiunque
     + Attributi:
       - static List<Utente> utenti: lista contenete tutti gli utenti registrati all'applicazione
       - static final String folderName: come per la classe Negozio, l'attributo serve per riferirsi alla cartella contenente gli utenti
       - static Utente currentUtente: rappresenta l'utente che ha effettuato l'accesso
     + Metodi:
       - Utenza(): costruttore della classe che, come quello della classe Negozio, inizializza la lista utenti e la riempie, tramite il metodo fillList()
       - void fillList(): metodo che per ogni file all'interno della cartella contenente gli utenti crea un utente che immetterà poi nella lista. Se la cartella non esiste verrà opportunamente creata
       - static void setCurrentUtente(Utente utente): imposta l'utente che ha effettuato l'accesso
       - static void getCurrentUtente(): metodo che ritorna l'utente che ha effettuato l'accesso
       - static boolean newUtente(Utente utente): metodo che crea un nuovo utente e ne crea il relativo file contenente i suoi dati, oltre ad essere aggiunto nella lista corrente. Il tutto avviene se e soltanto se l'utente non è già esistente
       - static void update Utente(): metodo che aggiorna il contenuto del file di un determinato utente. Più nello specifico è utilizzato per andare a riscrivere tutti i prodotti del carrello aggiunti in fase d'esecuzione
       - static Utente getUtente(String email): metodo che ritorna l'utente con l'email corrispondente a quella passatagli come parametro. La ricerca si ferma al primo riscontro
       - static boolean exist(String email): metodo che controlla se un determinato utente esiste già, confrontando la sua email con quelle presenti nella lista
       - static boolean getUserData(Stringe mail, String password): metodo utilizzato per l'autenticazione. Se l'email e la password inserite, e quindi passati come parametri al metodo, corrispondono con almeno l'email e la password di un utente interno alla lista allora il logIn avrà successo
2. Stile:
   * classe EnumStyle: classe contenente un singolo enumeratore, utilizzato per riferirsi più facilmente ai Look And Feel disponibili per l'applicazione nel momento in cui vengono impostati
   * classe AppStyle: classe utilizzata per cambiare dinamicamente il look and feel dell'applicazione. Rappresenta una finestra a parte ed implementa l'interfaccia ActionListener
     + Attributi:
       - JPanel jp: il pannello nel quale vengono aggiunti tutti i suoi componenti
       - JLabel title: etichetta utilizzata per impostare il titolo della finestra
       - ButtonGroup bg: oggetto utilizzato per racchiudere tutti i radioButton e permettere la selezione esclusiva di uno di essi
     + Metodi:
       - AppStyle(): costruttore della classe inserisce il titolo nel pannello, aggiunge tutti i radioButton nel pannello e nel gruppo di bottoni e tutte le specifiche per la finestra, come il layout, la visibilità e la sua chiusura
       - static void setLookAndFeel(EnumStyle style): metodo che cambia il look and feel corrente dell'applicazione, se lo trova
       - EnumStyle getCurrentLookAndFeel(): metodo che ritorna il look and feel correntemente salvato nel file relativo (style.txt)
       - void actionPerformed(ActionEvent e): metodo sovraccaricato per la gestione degli eventi. Come per il metodo precedente, cambia il look and feel dell'applicazione andando a salvare il look and feel selezionato nel proprio file
3. Interfacce

Le classi sotto riportate estendono il componente che devono rappresentare ed implementano quasi tutte l'ascoltatore di eventi.

* + classe GUI\_Overview: come già detto questa è la classe che funge da contenitore per le altre, estende JFrame ed implementa WindowListener
    - Attributi:
      * JPanel container: pannello che effettivamente contiene gli altri
      * GUI\_MenuBar bar: principale barra dei menù dell'applicazione
      * GUI\_SearchPanel header: intestazione del programma, subito sotto la barra dei menù, è il pannello di ricerca
      * GUI\_ProductsList list: pannello utilizzato per la visualizzazione dei risultati di ricerca
      * GUI\_Product page: pannello nel quale vengono visualizzate le informazioni relative ad un prodotto selezionato
      * GUI\_Info info: pannello nel quale vengono visualizzate le informazioni relative all'utente che ha effettuato l'accesso
      * GUI\_Cart cart: pannello utile alla visualizzazione del carrello dell'utente che ha effettuato l'accesso
    - Metodi:
      * GUI\_Overview(): costruttore della classe nel quale vengono instanziati i pannelli sopra citati, aggiunti al pannello principale e impostati alcune caratteristiche della finestra, tra cui il layout (la disposizione dei pannelli nella finestra è nell'asse verticale, dato dal BoxLayout)
      * void BuildProductsList(List<Prodotto> prodotti): metodo che costruisce la pagina relativa la visualizzazione dei risultati di ricerca, secondo una lista passatagli come parametro. I seguenti quattro metodi, insieme a questo, rappresentano effettivamente il mezzo di comunicazione tra i vari pannelli, oltre a nascondere quelli che non devono essere visualizzati
      * void BuildProductPage(Prodotto p): metodo che costruisce la pagina del prodotto passatogli come parametro
      * void BuildCartPage(): metodo che costruisce la pagina relativa al carrello dell'utente che ha effettuato l'accesso. Da qui si può anche rimuovere un elemento dal proprio carrello
      * void BuildInfoPage(): metodo che costruisce la pagina relativa alle informazione dell'utente che ha effettuato l'accesso
      * void Logged(): metodo che rende visibili tutti quei componenti utilizzabili sono da un utente autenticato, come ad esempio il pulsante di aggiunta al carrello, o altri nella barra dei menù
      * windowClosing(WindowEvent e): gestore di eventi dell'interfaccia WindowListener che chiede la conferma dell'uscita tramite opportuna finestra di dialogo. Gli altri gestori non vengono citati in quanto non sono stati utilizzati
  + classe GUI\_MenuBar: questa classe rappresenta la barra dei menù dell'applicazione, estende la classe JMenuBar ed implementa ActionListener
    - Attributi:
      * GUI\_Overview frame: serve per riferirsi esplicitamente all'oggetto che rappresenta la finestra principale, ovvero il mezzo di comunicazione con altri pannelli. Per le altre classi che presentano questo stesso attributo il comportamento è identico
      * JMenu jmAccesso: elemento della barra grazie al quale vi si trovano tutte le vie per accedere come utente o registrarvi all'applicazione
      * JMenuItem jmiAccedi: elemento interno a jmAccesso, serve per accedere come utente
      * JMenuItem jmiRegistrati: elemento interno a jmAccesso, serve per registrarsi all'applicazione
      * static JMenu jmAccount: elemento della barra grazie al quale vi si trovano tutte le vie per la gestione del proprio account
      * JMenuItem jmiAccount: elemento interno a jmAccount, serve per la visualizzazione delle informazioni relative il proprio account
      * JMenuItem jmiCart: elemento interno a jmAccount, serve per la visualizzazione del carrello del proprio utente
      * JMenuItem jmiStyle: elemento interno a jmAccount, serve per cambiare il look and feel dell'applicazione
      * JMenu jmEsci: elemento della barra grazie al quale si può uscire dall'applicazione. L'azione rimane tale quale a quella descritta per la gestione della chiusura della finestra
      * JMenuItem jmiEsci: elemento interno a jmEsci, serve per uscire effettivamente dall'applicazione
    - Metodi:
      * GUI\_MenuBar(): costruttore della classe che aggiunge tutti gli attributi sopra riportati alla barra, oltre ad aggiungergli un azione ed un ascoltatore
      * void setFrame(GUI\_Overview frame): metodo setter per impostare il riferimento alla finestra principale. Le classi che presentano questo stesso metodo si comportano nel medesimo modo
      * void isLogged(): metodo che rende visibile tutti quei componenti con i quali l'utente verificato potrà interagire
      * void actionPerformed(ActionEvetnt e): gestore eventi relative l'uscita dal programma, l'accesso, la registrazione e la visualizzazione delle informazioni dell'utente autenticato
  + classe GUI\_SearchPanel: come già descritto sopra la classe ha il compito di permettere all'utente di inserire i dati per la ricerca dei prodotti. Anch'essa estende JPanel ed implementa ActionListener
    - Attributi:
      * GUI\_Overview frame
      * JLabel logo: il logo dell'applicazione
      * JComboBox categories: menù a tendina nel quale vengono poste tutte le categorie di tutti i prodotti, oltre ad una categoria generica, ovvero vuota che indica tutte le categorie
      * JTextField searchArea: area di testo nel quale immettere il nome del prodotto da cercare, lasciando vuoto questo campo indica che si vogliono cercare tutti i prodotti
      * JButton search: bottone che inizia la ricerca dei prodotti secondo i criteri specificati. Alternativamente la ricerca si può avviare anche premendo invio mentre la casella di testo sopra riportata è selezionata
    - Metodi:
      * GUI\_SearchPanel(): costruttore della classe che crea inizialmente le categorie ed aggiunge i componenti al pannello
      * void setFrame(GUI\_Overview frame)
      * void setCategories(): per tutti i prodotti all'interno della lista statica dei prodotti verrà ricercata la sua categoria ed aggiunta alla lista. Se una categoria è già esistente non verrà aggiunta
      * actionPerformed(ActionEvent e): gestore di eventi per quanto riguarda la ricerca. Il pannello mostrerà i risultati di ricerca se i prodotti che danno un riscontro positivo con la ricerca sono superiori a zero
  + classe ProductsList: classe utile alla visualizzazione dei risultati di ricerca ottenuti con il metodo appena descritto. Essa estende JPanel ed implementa ActionListener
    - Attributi:
      * GUI\_Overview frame
      * JScrollPane jsp: barra di scorrimento, protected per rendere l'accesso semplificato dalle altre classi
    - Metodi:
      * GUI\_ProductsList(): costruttore della classe che imposta il suo layout
      * void setPage(List<Prodotto> prodotti): metodo che aggiunge al pannello tanti bottoni quanti sono i prodotti nella lista, che rappresentano i risultati della ricerca effettuata
      * void setFrame(GUI\_Overview frame)
      * actionPerformed(ActionEvent e): gestore di eventi relativi la visualizzazione di uno specifico prodotto. In base a quale si selezioni si avrà una pagina mostrata differente
  + classe GUI\_Product: classe nella quale viene visualizzato nello specifico il contenuto di un prodotto. Come le altre, essa estende JPanel ed implementa ActionListener
    - Attributi:
      * JButton buy: bottone che serve per l'aggiunta al carrello del prodotto correntemente visualizzato. Questo bottone rimarrà nascosto finché un utente non accederà
      * SpringLayout layout: layout della classe
    - Metodi:
      * GUI\_Product(): costruttore della classe che imposta il suo layout
      * void setPage(Prodotto p): metodo che aggiunge al pannello tutte quei componenti adatti alla visualizzazione delle informazioni del prodotto passatogli come parametro
      * void setButtonVisible(): metodo che rende visibile il bottone per l'aggiunta al carrello e che sottintende l'autenticazione verificata un utente
      * actionPerformed(ActionEvent e): gestore eventi per l'acquisto del prodotto corrente. Esso sarà poi aggiunto al carrello dell'utente solo se i soldi nel saldo dell'utente sono sufficienti
  + classe GUI\_Info: classe nella quale vengono visualizzate tutte le informazioni relative ad un utente, può essere considerata una classe di riepilogo. Essa estende JPanel ed implementa ActionListener
    - Attributi:
      * Questa classe non presenta attributi
    - Metodi:
      * GUI\_Info(): costruttore della classe che imposta il suo layout e viene impostata la sua dimensione. Inoltre vengono aggiunti tutti i suoi elementi
      * void addElements(): metodo che aggiunge tutti i componenti per la corretta visualizzazione dei dati di un utente. Questo avviene solo se un utente ha effettuato l'accesso
      * void actionPerformed(ActionEvent e): gestore eventi della classe. Serve per gestire l'aggiunta di denaro all'account utente
  + classe GUI\_Cart: classe relativa la visualizzazione del carrello dell'utente. All'interno è inoltre possibile rimuovere prodotti dal carrello. Essa estende JPanel ed implementa ActionListener
    - Attributi:
      * JLabel title: il titolo del pannello
      * JScrollPane jsp: barra di scorrimento, protected per rendere l'accesso semplificato dalle altre classi
    - Metodi:
      * GUI\_Cart(): come per la precedente classe questo metodo costruttore imposta il suo layout. Inoltre vengono aggiunti tutti i suoi elementi
      * void addElements(): metodo che aggiunge tutti i componenti per la corretta visualizzazione dei prodotti interni al carrello dell'utente. Questo avviene solo se un utente ha effettuato l'accesso
      * actionPerformed(ActionEvent e): gestore di eventi che rimuove un prodotto dal carrello e restituisce la somma di denaro prelevata all'utente
  + classe LogIn: classe impiegata per far accedere un utente, richiamabile dalla barra dei menu
    - Attributi:
      * GUI\_Overview frame;
      * JTextField email: campo di autenticazione, email
      * JPasswordField password: campo di autenticazione, password
      * JPanel panel: pannello che fa da contenitore per gli elementi della classe
    - Metodi:
      * LogIn(GUI\_Overview frame): metodo costruttore che imposta il valore dell'attributo frame, aggiunge tutti i componenti relativi l'autenticazione al pannello ed una finestra di dialogo per l'effettiva autenticazione. Se i dati inseriti sono corretti allora il logIn avrà successo, altrimenti verrà comunicato il fallimento grazie ad un'altra finestra di dialogo
  + classe SignUp: classe utilizzata per la registrazione di un utente
    - Attributi:
      * JTextField name: campo nome
      * JTextField surname: campo cognome
      * JTextField email: campo email
      * JPasswordField password: campo password
      * JPasswordField repeatPassword: campo ripetizione password
      * JPanel panel: pannello di contenimento
    - Metodi:
      * SignUp(): metodo costruttore che aggiunge al pannello tutti gli elementi sopra riportati. Il pannello sarà poi aggiunto ad una finestra di dialogo adatta alla registrazione dell'utente. Per far si che ciò accada correttamente si deve inserire un nome e un cognome di almeno cinque caratteri, una password ed una password ripetuta di almeno cinque caratteri ed uguali tra di loro. Se ciò fallisce verrà comunicato opportunamente tramite finestre di dialogo il perché di tutto ciò

1. Main
   * classe Main: classe il cui unico scopo è quello di avviare il programma
     + Attributi:
       - Questa classe non presenta attributi
     + Metodi:
       - static void main(String[] args): main nel quale viene impostato un look and feel iniziale, vengono instanziate le due classi statiche principali, ovvero Negozio e Utenza e viene instanziata l'interfaccia grafica

Tabella riassuntiva componenti grafici:

|  |  |  |  |
| --- | --- | --- | --- |
| Classe | Tipo | Nome componente | Layout (se pannello) |
| GUI\_Overview | GUI\_Overview | - | - |
| GUI\_Overview | JPanel | container | BoxLayout |
| GUI\_Overview | GUI\_MenuBar | bar | - |
| GUI\_Overview | GUI\_SearchPanel | header | FlowLayout |
| GUI\_Overview | GUI\_ProductsList | list | FlowLayout |
| GUI\_Overview | GUI\_Product | page | SpringLayout |
| GUI\_Overview | GUI\_Info | info | BoxLayout |
| GUI\_Overview | GUI\_Cart | cart | BoxLayout |
| GUIMenuBar | GUI\_MenuBar | - | - |
| GUI\_MenuBar | JMenu | jmAccesso | - |
| GUI\_MenuBar | JMenuItem | jmiAccedi | - |
| GUI\_SearchPanel | GUI\_SearchPanel | - | FlowLayout |
| GUI\_SearchPanel | JLabel | logo | - |
| GUI\_SearchPanel | JComboBox | categories | - |
| GUI\_SearchPanel | JTextField | searchArea | - |
| GUI\_SearchPanel | JButton | search | - |
| GUI\_ProductsList | GUI\_ProductsList | - | BoxLayout |
| GUI\_ProductsList | JScrollPane | jsp | - |
| GUI\_Product | GUI\_Product | - | SpringLayout |
| GUI\_Product | JButton | buy | - |
| GUI\_Product | SpringLayout | layout | SpringLayout |
| GUI\_Info | GUI\_Info | - | BoxLayout |
| GUI\_Cart | GUI\_Cart | - | BoxLayout |
| GUI\_Cart | JLabel | title | - |
| GUI\_Cart | JScrollPane | jsp | - |
| LogIn | LogIn | - | - |
| LogIn | JTextField | email | - |
| LogIn | JPasswordField | password | - |
| LogIn | JPanel | panel | FlowLayout |
| SignUp | SignUp | - | - |
| SignUp | JTextField | name | - |
| SignUp | JTextField | surname | - |
| SignUp | JTextField | email | - |
| SignUp | JPasswordField | password | - |
| SignUp | JPasswordField | repeatPassword | - |
| SignUp | JPanel | panel | FlowLayout |

**Conclusioni**:

Il lavoro svolto dal gruppo è stato realizzato in circa tre settimane, alternando il lavoro fatto a casa con quello fatto a scuola. Ognuno ha contribuito per lo sviluppo dell'applicativo fino al raggiungimento della soluzione. Un ringraziamento anche ai grafici che ci hanno dato una mano.

Partecipanti:

***Corbucci Francesco***

***Los Alex***

***Mantovani Filippo***

***Miccioni Luca***

***Sosti Luca***