RELAZIONE PROGETTO PMO A.A 2020/2021

Carlo Ricchiuto,298320

SPECIFICA DEL SOFTWARE

Lo scopo del progetto è la realizzazione di un distributore automatico di bevande attraverso il linguaggio di programmazione c#
Il distributore automatico consentirà di:

- Visualizzare i prodotti presenti, con il loro prezzo, il loro codice di riconoscimento univoco e la loro disponibilità.
- Aggiungere credito in ogni momento (solo monete).
- Visualizzare il credito attuale.
- Acquistare, tramite codice di riconoscimento, il prodotto corrispondente a quel codice (se disponibile).
- Ritirare il resto.

STUDIO DEL PROBLEMA

PUNTI CRITICI

Dal progetto, sono nati diversi punti critici, elencati di seguito:

- La creazione della lista dei prodotti acquisiti da file e la loro leggibilità per il cliente.
- Lo sviluppo del menu di selezione delle operazioni da fare e la conseguente combinazione con gli altri metodi per utilizzare tutte le funzioni del distributore.
- Ricerca del prodotto una volta inserito il codice di riconoscimento.
- Leggibilità generale della console.

FRONTEGGIAMENTO

Creazione della lista dei prodotti acquisiti da file e la loro leggibilità per il cliente.
Al fine di rendere possibile l'acquisizione da file della lista dei prodotti con nome,prezzo,codice di riconoscimento e disponibilità, è stato necessario implementare una lista "prodotti"(ovviamente di tipo Prodotto, in modo che acquisisca gli attributi necessari) e un oggetto, sempre di tipo Prodotto, istanziato all'interno di un ciclo in modo da aggiungere ogni prodotto all'interno della lista. Ciò è reso possibile poiché all'interno del ciclo viene letto e acquisito ogni attributo di ogni prodotto attraverso una lettura stringa per stringa(che fosse di numeri o di lettere), riconosciuta opportunamente da un "-"(che divide un attributo dall'altro).

Lo sviluppo del menu di selezione delle operazioni da fare e la conseguente combinazione con gli altri metodi per utilizzare tutte le funzioni del distributore.
Per il fine di ottenere un menu adeguato, e leggibile a livello di codice, è stata presa la decisione di implementare uno switch che richiamasse i metodi a seconda di ciò che viene selezionato. Infatti a seconda della selezione del cliente sarà possibile effettuare le operazioni citate sulla "specifica del software". Inoltre il menu ha la possibilità di terminare il programma nel caso in cui nello stesso menu venga inserita una lettera. Nel caso in cui invece ciò avvenga in un metodo selezionato dal menu, il programma non si chiude, ma torna al menu iniziale.

Ricerca del prodotto una volta inserito il codice di riconoscimento.
Per cercare all'interno della lista il codice univoco corrispondente a un determinato prodotto, è stato necessario implementare un metodo che potesse acquisire dal cliente il numero da cercare e, tramite un ciclo e diverse operazioni di selezione, che escludesse sia un numero inesistente che un prodotto esaurito. In seguito se il prodotto viene trovato, i dati vengono passati sul metodo del tentativo di acquisto.

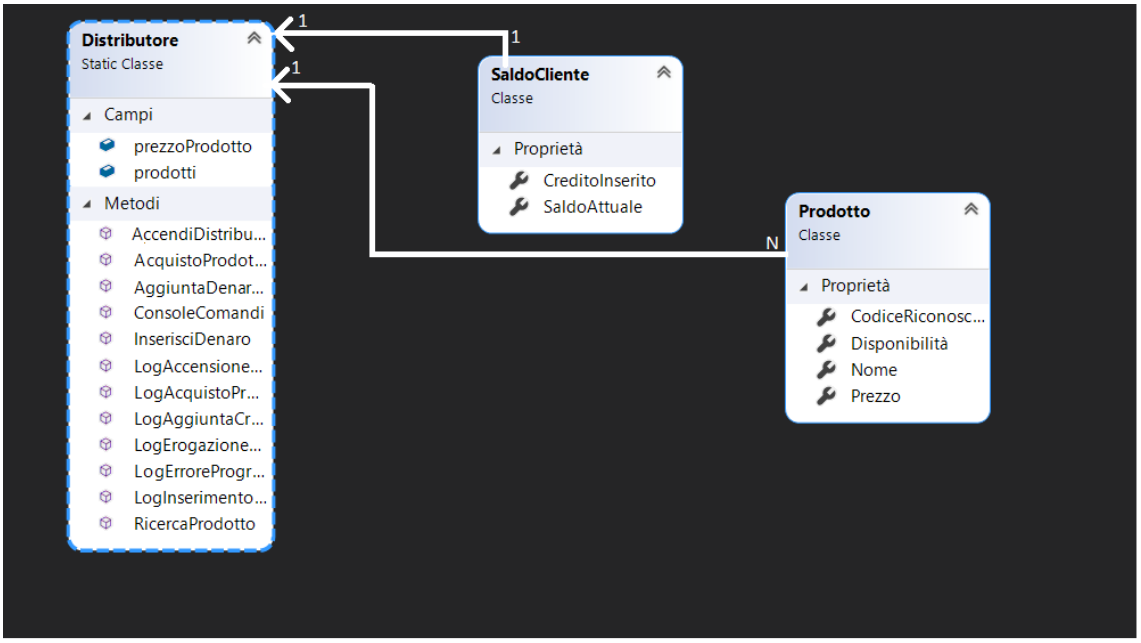
Leggibilità generale della console.
Per rendere la console leggibile, in quanto vi sono molte operazioni da fare, esiti, elementi da mostrare, è stato necessario modificare a seconda dell'operazione(o errore) il colore delle stringhe della console.
In particolare:

- Rosso per gli errori e per la richiesta del percorso del file.
- Verde per mostrare il menu dei prodotti a disposizione con i loro attributi.
- Blu per il menu iniziale.
- Giallo per il menu di inserimento monete.

- Magenta per la ricerca del prodotto.
- Diverse tonalità di giallo per il tentativo di acquisto.
- Grigio per mostrare il saldo attuale.
- Blu scuro per l’erogazione del resto.

SCELTE ARCHITETTURALI

DIAGRAMMA DELLE CLASSI



DESCRIZIONE DELL'ARCHITETTURA

AccendiDistributore

L'architettura del software è incentrata sulla classe statica "Distributore" e, in particolare, sul metodo "AccendiDistributore" . Attraverso il richiamo di questo metodo nel main infatti avviene l'accensione, l'acquisizione da file della lista dei prodotti e il menu di selezione con tutte le operazioni da fare.

```
//Metodo statico accendiDistributore
1 riferimento
public static void accendiDistributore()
{
    //Acquisizione percorso file Inventario.txt da console
    string filePath;
    //Colore del testo console: rosso
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("Benvenuto! prima di iniziare, digita qui il percorso per acquisire l'inventario:");
    //Inserimento percorso file e inizializzazione classe
    filePath = Console.ReadLine();
    SaldoCliente saldo = new SaldoCliente()
    {
        CreditoInserito = 0
    };
    try
    {
        Distributore.LogAccensioneSpegnimentoDistributore("Accensione"," riuscita!");
        var lines = File.ReadAllLines(filePath);
        //Colore del testo console: verde
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("I prodotti disponibili oggi sono:\nNome      Prezzo  Codice  Disponibilità");
        //Inizializzazione lista di tipo Prodotto da compilare
        Distributore.prodotti = new List<Prodotto>();
        //Acquisizione da file degli attributi di ogni prodotto
        foreach (var item in lines)
        {
            //Compilazione lista di ogni prodotto
            var elements = item.Split("-");
            Prodotto prodotto = new Prodotto()
            {
                Nome = elements[0],
                Prezzo = double.Parse(elements[1]),
                CodiceRiconoscimento = int.Parse(elements[2]),
                Disponibilità = int.Parse(elements[3])
            };
            prodotti.Add(prodotto);
            Console.WriteLine(prodotto.Nome + "      " + prodotto.Prezzo + "      " + prodotto.CodiceRiconoscimento + "       " + prodotto.Disponibilità);
        }
        //Richiamo metodo ConsoleComandi
        ConsoleComandi(saldo);

    }
    //catch per l'exception
    catch (FileNotFoundException)
    {
        Distributore.LogAccensioneSpegnimentoDistributore("Accensione"," fallita");
        Console.WriteLine("File di input non trovato, impossibile avviare programma");
        Distributore.LogErroreProgramma("Il programma è stato chiuso in quanto: è stato inserito un percorso file non valido.");
    }
}
```

All’interno del metodo avvengono tutte le operazioni fondamentali, inclusa la verifica del file di input tramite try-catch e il richiamo del metodo “ConsoleComandi”, anch’esso di una certa importanza.

ConsoleComandi

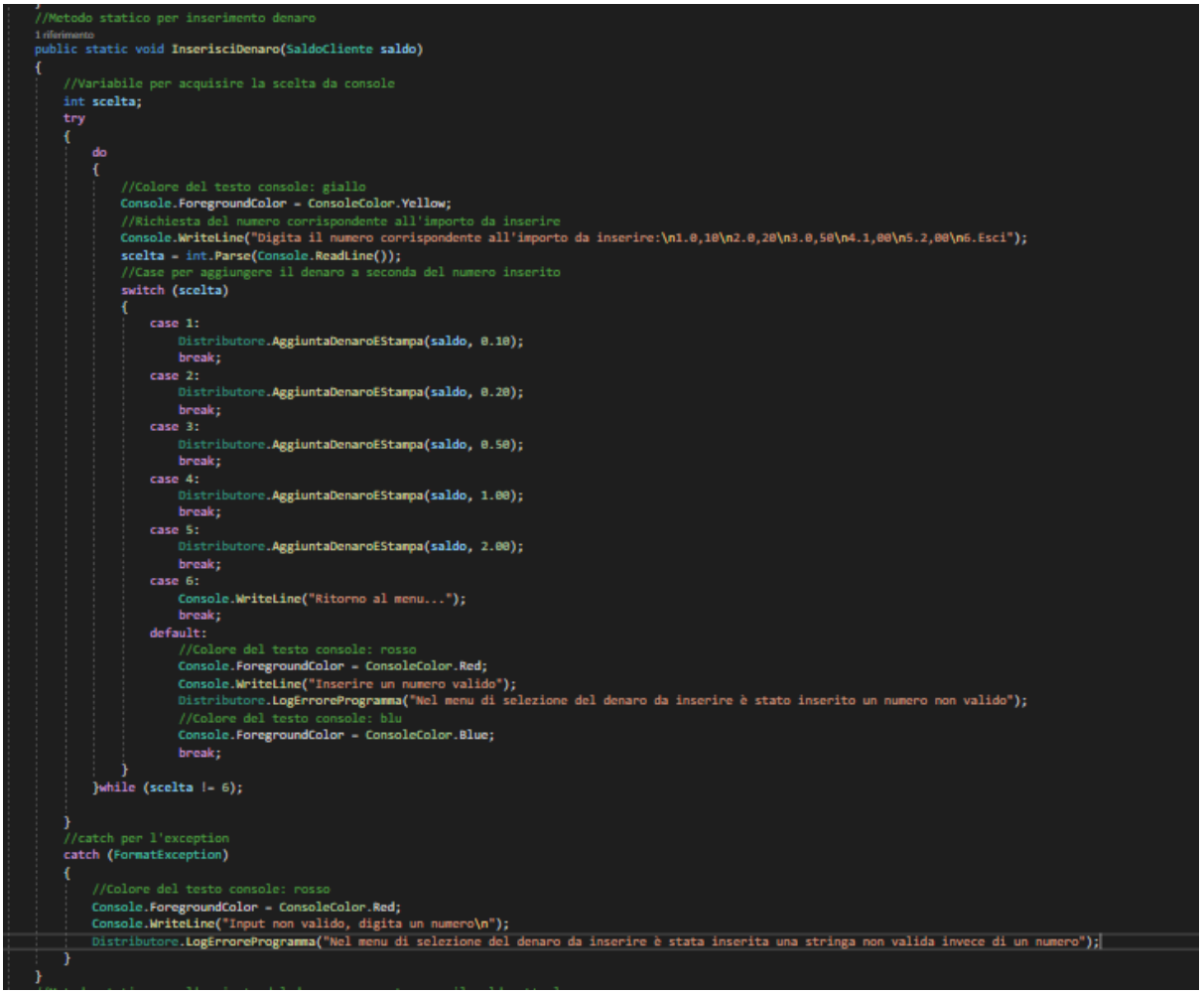
Il metodo “ConsoleComandi” include infatti tutti gli altri metodi necessari per la corretta esecuzione del programma.

```
//Metodo statico Console Comandi
1 riferimento
public static void ConsoleComandi(SaldoCliente saldo)
{

    int scelta;
    try
    {
        do
        {
            //Colore del testo console: blu
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine("\nScegli l'operazione da fare: \n1.Inserisci denaro\n2.Cerca il prodotto e acquista\n3.Visualizza il saldo attuale\n4.Ritira il resto e termina");
            scelta = int.Parse(Console.ReadLine());

            switch (scelta)
            {
                case 1:
                    //Caso uno per richiamare il metodo InserisciDenaro
                    InserisciDenaro(saldo);
                    break;
                case 2:
                    //Caso due per richiamare il metodo Ricerca prodotto
                    RicercaProdotto(saldo);
                    break;
                case 3:
                    //Caso tre per richiamare il metodo Ricerca prodotto
                    //Colore del testo console: grigio scuro
                    Console.ForegroundColor = ConsoleColor.DarkGray;
                    Console.WriteLine("Il tuo saldo attuale è di: " + String.Format("{0:0.00}", saldo.SaldoAttuale));
                    break;
                case 4:
                    //Caso quattro per richiedere il resto
                    //Colore del testo console: blu scuro
                    Console.ForegroundColor = ConsoleColor.DarkBlue;
                    Console.WriteLine("Erogazione resto in corso...\nResto erogato: " + String.Format("{0:0.00}", saldo.SaldoAttuale));
                    //Colore del testo console: bianco
                    Console.ForegroundColor = ConsoleColor.White;
                    Distributore.LogErogazioneResto(saldo.SaldoAttuale);
                    Distributore.LogAccensioneSpegnimentoDistributore("Spegnimento",".");
                    break;
                default:
                    //Colore del testo console: rosso
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Inserire un numero valido\n\n");
                    Distributore.LogErroreProgramma("Nel menu di selezione delle operazioni da fare è stato inserito un numero non valido");
                    break;
            }
        } while (scelta != 4);
    }
    //catch per l'exception
    catch (FormatException)
    {
        //Colore del testo console: rosso
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Input non valido, digita un numero.\nChiusura del programma...");
        Distributore.LogErroreProgramma("Il programma è stato chiuso in quanto nel menu di selezione delle operazioni da fare è stata inserita una stringa non valida invece di un numero.");
    }
}
```

Infatti questo metodo richiama, tramite uno switch, ogni metodo necessario a seconda della richiesta del cliente, tranne per il caso in cui venga richiesto di mostrare il saldo attuale, essendo un’operazione semplice gestibile direttamente in quel metodo. Ovviamente anche nella console è presente un try-catch che chiude il programma in caso venga inserita una stringa non valida invece di un numero.

InserisciDenaro

Il metodo “InserisciDenaro” è stato relativamente semplice da implementare, in quanto è un semplice switch che a seconda del numero inserito, è possibile inserire una moneta, che va dagli 0,10 ai 2,00.



```
//Metodo statico per inserimento denaro
1 riferimento
public static void InserisciDenaro(SaldoCliente saldo)
{
    //Variabile per acquisire la scelta da console
    int scelta;
    try
    {
        do
        {
            //Colore del testo console: giallo
            Console.ForegroundColor = ConsoleColor.Yellow;
            //Richiesta del numero corrispondente all'importo da inserire
            Console.WriteLine("Digita il numero corrispondente all'importo da inserire:\n1.0,10\n2.0,20\n3.0,50\n4.1,00\n5.2,00\n6.Esci");
            scelta = int.Parse(Console.ReadLine());
            //Case per aggiungere il denaro a seconda del numero inserito
            switch (scelta)
            {
                case 1:
                    Distributore.AggiuntaDenaroEStampa(saldo, 0.10);
                    break;
                case 2:
                    Distributore.AggiuntaDenaroEStampa(saldo, 0.20);
                    break;
                case 3:
                    Distributore.AggiuntaDenaroEStampa(saldo, 0.50);
                    break;
                case 4:
                    Distributore.AggiuntaDenaroEStampa(saldo, 1.00);
                    break;
                case 5:
                    Distributore.AggiuntaDenaroEStampa(saldo, 2.00);
                    break;
                case 6:
                    Console.WriteLine("Ritorno al menu...");
                    break;
                default:
                    //Colore del testo console: rosso
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Inserire un numero valido");
                    Distributore.LogErroreProgramma("Nel menu di selezione del denaro da inserire è stato inserito un numero non valido");
                    //Colore del testo console: blu
                    Console.ForegroundColor = ConsoleColor.Blue;
                    break;
            }
        }while (scelta != 6);

    }
    //catch per l'exception
    catch (FormatException)
    {
        //Colore del testo console: rosso
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Input non valido, digita un numero\n");
        Distributore.LogErroreProgramma("Nel menu di selezione del denaro da inserire è stata inserita una stringa non valida invece di un numero");
    }
}
```

AggiuntaEStampaDenaro

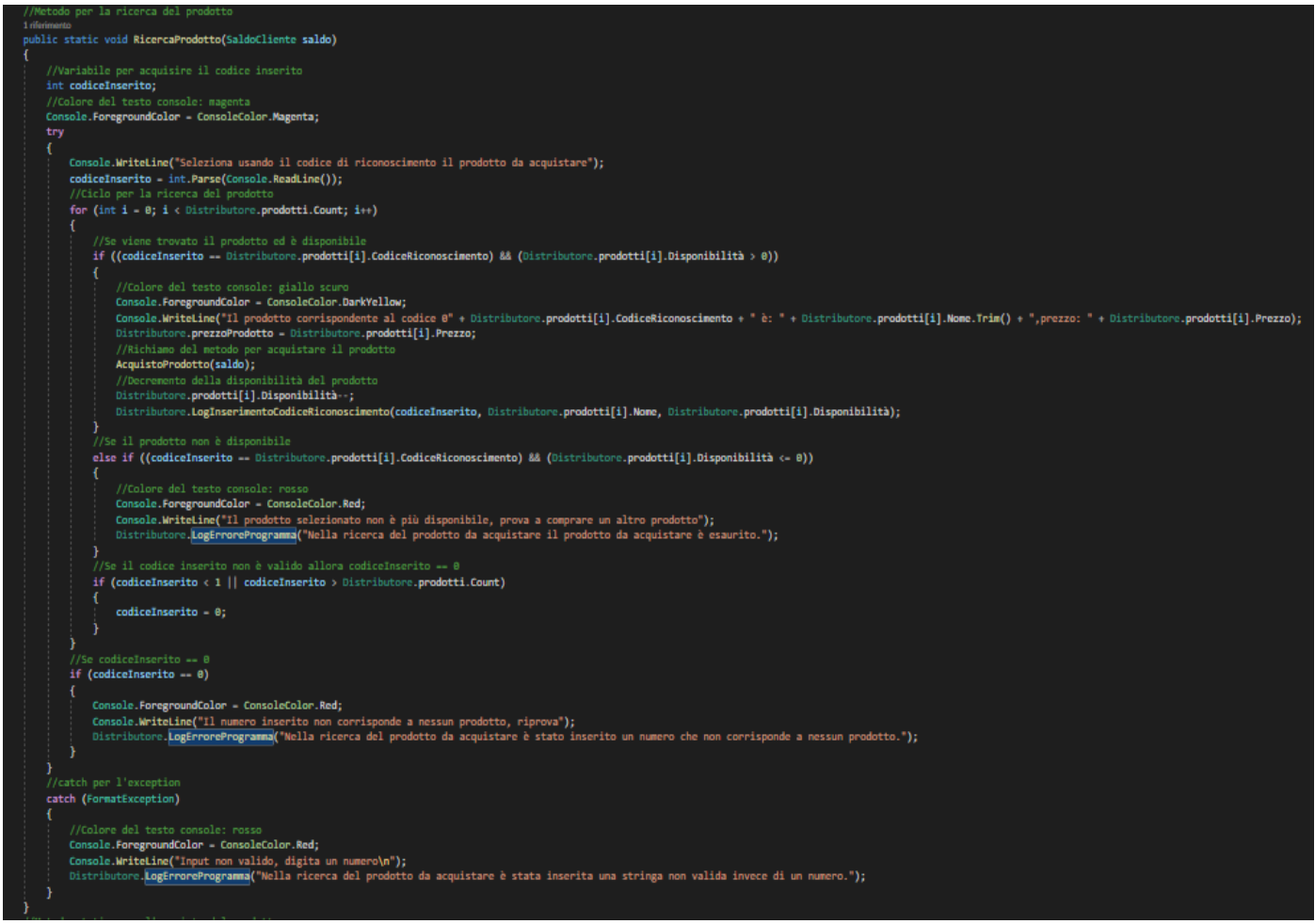
Il metodo “AggiuntaEStampaDenaro” fa semplicemente ciò che il nome del metodo indica:

```
//Metodo statico per l'aggiunta del denaro e per stampare il saldo attuale
5 riferimenti
public static void AggiuntaDenaroEStampa(SaldoCliente saldo, double creditoDaAggiungere)
{
    //Colore del testo console: grigio scuro
    Console.ForegroundColor = ConsoleColor.DarkGray;
    //Somma del credito inserito al saldo attuale
    saldo.CreditoInserito = creditoDaAggiungere;
    saldo.SaldoAttuale += saldo.CreditoInserito;
    Console.WriteLine("Credito aggiunto! Saldo attuale: " + String.Format("{0:0.00}", saldo.SaldoAttuale));
    Distributore.LogAggiuntaCredito(saldo.CreditoInserito, saldo.SaldoAttuale);
    //Colore del testo console: giallo
    Console.ForegroundColor = ConsoleColor.Yellow;
}
```

Il credito inserito prende il valore della variabile “creditoDaAggiungere” e viene aggiunta al saldo attuale.

RicercaProdotto

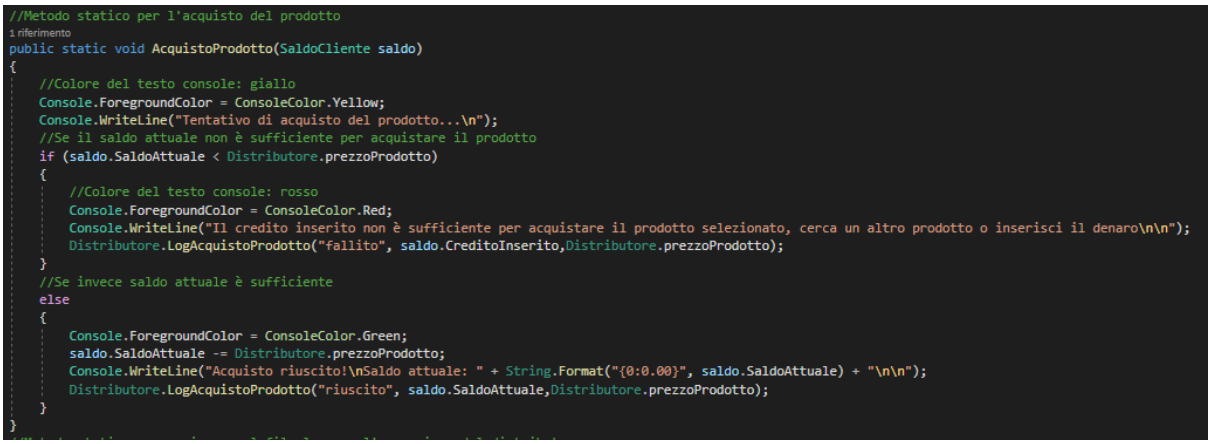
Il metodo “RicercaProdotto” si occupa di scorrere la lista e cercare il prodotto corrispondente al numero (con eventuali messaggi di errore nel momento dell’inserimento da tastiera da parte del cliente o del prodotto esaurito o codice inesistente) e passare al metodo “AcquistoProdotto”, attraverso la variabile pubblica per tutti i metodi “prezzoProdotto”, il prezzo del prodotto trovato, in modo da tentare l’acquisto



```
//Metodo per la ricerca del prodotto
1 riferimento
public static void RicercaProdotto(SaldoCliente saldo)
{
    //Variabile per acquisire il codice inserito
    int codiceInserito;
    //Colore del testo console: magenta
    Console.ForegroundColor = ConsoleColor.Magenta;
    try
    {
        Console.WriteLine("Seleziona usando il codice di riconoscimento il prodotto da acquistare");
        codiceInserito = int.Parse(Console.ReadLine());
        //Ciclo per la ricerca del prodotto
        for (int i = 0; i < Distributore.prodotti.Count; i++)
        {
            //Se viene trovato il prodotto ed è disponibile
            if ((codiceInserito == Distributore.prodotti[i].CodiceRiconoscimento) && (Distributore.prodotti[i].Disponibilità > 0))
            {
                //Colore del testo console: giallo scuro
                Console.ForegroundColor = ConsoleColor.DarkYellow;
                Console.WriteLine("Il prodotto corrispondente al codice 0" + Distributore.prodotti[i].CodiceRiconoscimento + " è: " + Distributore.prodotti[i].Nome.Trim() + ",prezzo: " + Distributore.prodotti[i].Prezzo);
                Distributore.prezzoProdotto = Distributore.prodotti[i].Prezzo;
                //Richiamo del metodo per acquistare il prodotto
                AcquistoProdotto(saldo);
                //Decremento della disponibilità del prodotto
                Distributore.prodotti[i].Disponibilità--;
                Distributore.LogInserimentoCodiceRiconoscimento(codiceInserito, Distributore.prodotti[i].Nome, Distributore.prodotti[i].Disponibilità);
            }
            //Se il prodotto non è disponibile
            else if ((codiceInserito == Distributore.prodotti[i].CodiceRiconoscimento) && (Distributore.prodotti[i].Disponibilità <= 0))
            {
                //Colore del testo console: rosso
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("Il prodotto selezionato non è più disponibile, prova a comprare un altro prodotto");
                Distributore.LogErroreProgramma("Nella ricerca del prodotto da acquistare il prodotto da acquistare è esaurito.");
            }
            //Se il codice inserito non è valido allora codiceInserito == 0
            if (codiceInserito < 1 || codiceInserito > Distributore.prodotti.Count)
            {
                codiceInserito = 0;
            }
        }
        //Se codiceInserito == 0
        if (codiceInserito == 0)
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Il numero inserito non corrisponde a nessun prodotto, riprova");
            Distributore.LogErroreProgramma("Nella ricerca del prodotto da acquistare è stato inserito un numero che non corrisponde a nessun prodotto.");
        }
    }
    //catch per l'exception
    catch (FormatException)
    {
        //Colore del testo console: rosso
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Input non valido, digita un numero\n");
        Distributore.LogErroreProgramma("Nella ricerca del prodotto da acquistare è stata inserita una stringa non valida invece di un numero.");
    }
}
```

AcquistoProdotto

Qui di seguito è presente il metodo sopracitato “AcquistoProdotto”, in cui viene tentato l’acquisto del prodotto, e a seconda del saldo ne consegue un esito positivo o negativo, a seconda se il credito sia sufficiente o meno per quel prodotto selezionato



```
//Metodo statico per l'acquisto del prodotto
1 riferimento
public static void AcquistoProdotto(SaldoCliente saldo)
{
    //Colore del testo console: giallo
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("Tentativo di acquisto del prodotto...\n");
    //Se il saldo attuale non è sufficiente per acquistare il prodotto
    if (saldo.SaldoAttuale < Distributore.prezzoProdotto)
    {
        //Colore del testo console: rosso
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Il credito inserito non è sufficiente per acquistare il prodotto selezionato, cerca un altro prodotto o inserisci il denaro\n\n");
        Distributore.LogAcquistoProdotto("fallito", saldo.CreditoInserito,Distributore.prezzoProdotto);
    }
    //Se invece saldo attuale è sufficiente
    else
    {
        Console.ForegroundColor = ConsoleColor.Green;
        saldo.SaldoAttuale -= Distributore.prezzoProdotto;
        Console.WriteLine("Acquisto riuscito!\nSaldo attuale: " + String.Format("{0:0.00}", saldo.SaldoAttuale) + "\n\n");
        Distributore.LogAcquistoProdotto("riuscito", saldo.SaldoAttuale,Distributore.prezzoProdotto);
    }
}
```

Log

Per la gestione dei log è stato necessario implementare diversi metodi, uno per ogni tipo di operazione, in quanto le operazioni effettuate sono molto diverse tra loro e non è stato possibile ottenere un numero limitato di metodi.

```
//Metodo statico per scrivere sul file log per l'accensione del distributore
3 riferimenti
public static void LogAccensioneSpegnimentoDistributore(string statoDistributore, string esitoLog)...
//Metodo statico per scrivere sul file log l'aggiunta del credito
1 riferimento
public static void LogAggiuntaCredito(double creditoInseritoLog, double saldoAttualeLog)...
//Metodo statico per scrivere sul file log l'inserimento del codice di riconoscimento con stampa delle informazioni sul prodotto
1 riferimento
public static void LogInserimentoCodiceRiconoscimento(int codiceLog,string nomeProdottoLog,int nuovaDisponibilitàLog)...
//Metodo statico per scrivere sul file log l'esito dell'acquisto
2 riferimenti
public static void LogAcquistoProdotto(string esitoAcquistoLog, double saldoAttualeLog, double prezzoProdottoLog)...
//Metodo statico per scrivere sul file log il resto erogato
1 riferimento
public static void LogErogazioneResto(double restoErogatoLog)...
//Metodo statico per scrivere sul file log l'eventuale motivo della chiusura prematura del programma
4 riferimenti
public static void LogErroreProgramma(string esitoLog)...
```

Prodotto

La classe prodotto contiene i get e set di tutti gli attributi del prodotto

```
namespace FoodDispenser
{
    4 riferimenti
    class Prodotto
    {
        //Attributo per il nome
        4 riferimenti
        public string Nome
        {
            get;
            set;
        }
        //Attributo per il prezzo
        4 riferimenti
        public double Prezzo
        {
            get;
            set;
        }
        //Attributo per il codice di riconoscimento
        5 riferimenti
        public int CodiceRiconoscimento
        {
            get;
            set;
        }
        //Attributo per la disponibilità
        6 riferimenti
        public int Disponibilità
        {
            get;
            set;
        }
    }
}
```

SaldoCliente

SaldoCliente contiene i get e set per gli attributi che riguardano il credito inserito e il saldo attuale.

```
namespace FoodDispenser
{
    7 riferimenti
    class SaldoCliente
    {
        //Attributo per il credito inserito
        5 riferimenti
        public double CreditoInserito
        {
            get;
            set;
        }
        //Attributo per il saldo attuale
        10 riferimenti
        public double SaldoAttuale
        {
            get;
            set;
        }

    }
}
```

Program

La classe program contiene semplicemente la cancellazione del file log dell'eventuale esecuzione precedente per crearne uno nuovo e il richiamo del metodo statico "AccendiDistributore"

```
namespace FoodDispenser
{
    0 riferimenti
    class Program
    {
        0 riferimenti
        static void Main()
        {
            //Cancellazione del file log precedente
            File.Delete("Log.txt");
            //Richiamo del metodo statico accendiDistributore
            Distributore.accendiDistributore();
        }
    }
}
```

Inventario.txt

Il file di testo "Inventario.txt" contiene un elenco di prodotti con i loro attributi, divisi tramite il carattere "-", riconosciuto come split tra le stringhe nel momento in cui viene acquisita la lista dei prodotti.

```
1  CocaCola-2,5    -01-3
2  Sprite  -2,5    -02-2
3  Tuc -1,4    -03-3
4  Twix    -1,8    -04-3
5  Fanta   -2,5    -05-1
6  Smarties-1,7    -06-3
7  Patatine-1,5    -07-2
8  Acqua   -1,2    -08-3
9  Mikado  -1,6    -09-2
```

Questo è un esempio di una semplice esecuzione con l'acquisto di un prodotto, in particolare "Coca cola".

```
Benvenuto! prima di iniziare, digita qui il percorso per acquisire l'inventario:
C:\Users\lallo\OneDrive\Desktop\università\Progetto pmo\FoodDispenser\Inventario.txt
I prodotti disponibili oggi sono:
Nome      Prezzo  Codice  Disponibilità
CocaCola     2,5     1       3
Sprite       2,5     2       2
Tuc          1,4     3       3
Twix         1,8     4       3
Fanta        2,5     5       1
Smarties     1,7     6       3
Patatine     1,5     7       2
Acqua        1,2     8       3
Mikado       1,6     9       2

Scegli l'operazione da fare:
1.Inserisci denaro
2.Cerca il prodotto e acquista
3.Visualizza il saldo attuale
4.Ritira il resto e termina
1
Digita il numero corrispondente all'importo da inserire:
1.0,10
2.0,20
3.0,50
4.1,00
5.2,00
6.Esci
5
Credito aggiunto! Saldo attuale: 2,00
Digita il numero corrispondente all'importo da inserire:
1.0,10
2.0,20
3.0,50
4.1,00
5.2,00
6.Esci
3
Credito aggiunto! Saldo attuale: 2,50
Digita il numero corrispondente all'importo da inserire:
1.0,10
2.0,20
3.0,50
4.1,00
5.2,00
6.Esci
6
Ritorno al menu...
```

```
Scegli l'operazione da fare:
1.Inserisci denaro
2.Cerca il prodotto e acquista
3.Visualizza il saldo attuale
4.Ritira il resto e termina
2
Seleziona usando il codice di riconoscimento il prodotto da acquistare
1
Il prodotto corrispondente al codice 01 è: CocaCola,prezzo: 2,5
Tentativo di acquisto del prodotto...

Acquisto riuscito!
Saldo attuale: 0,00



Scegli l'operazione da fare:
1.Inserisci denaro
2.Cerca il prodotto e acquista
3.Visualizza il saldo attuale
4.Ritira il resto e termina
4
Erogazione resto in corso...
Resto erogato: 0,00
```

DOCUMENTAZIONE SULL'UTILIZZO

Consegno la cartella contenente tutti i file necessari all'esecuzione del programma con annesso il file di testo "Inventario.txt"(importante per il percorso da inserire: Progetto pmo\FoodDispenser\Inventario.txt).

USE CASES CON RELATIVO SCHEMA UML