Università degli Studi di Bergamo
Scuola di Ingegneria
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione
Corso di Laurea Magistrale in Ingegneria Informatica

Informatica III
Modulo di Programmazione

ASM-commerce

Prof. Angelo Gargantini

Progetto Asmeta di:

Emanuele Perico

Anno Accademico 2019-2020

Indice

1 Introduzione

Nel progetto *ASM-commerce*, scritto in linguaggio AsmetaL, sono stati utilizzati diversi costrutti tipici di questo linguaggio di programmazione, come: domini, funzioni e regole di transizione tra stati per poter modellizzare una macchina a stati astratta (ASM).
Lo scenario scelto per questo progetto è quello di un e-commerce: in particolare, si è voluto creare un piccolo esempio per la gestione dello stato di un ordine cliente da parte di un'azienda.

2 Programma

Il programma si occupa della gestione di un ordine cliente da parte di un'azienda, attraverso una macchina a stati.
Il programma ha l'obiettivo di gestire il corretto processo di un ordine: richiesta del codice identificativo del cliente, scelta di un articolo da ordinare, scelta del tipo di spedizione e invio della fattura.

2.1 Struttura

All'interno di questo progetto, sono presenti rispettivamente:

- Domains: *IdArticle*, *IdCustomer*, *Shipping*, *State*.
  I primi due domini sono definiti come sottodomini di **Integer**, mentre gli ultimi due sono di tipo **enum** per gestire rispettivamente il tipo di spedizione e lo stato in cui ci si trova attualmente.

- Functions: definite secondo tipi differenti.

  - dynamic controlled: *currArtID*, *currCustID*, *currState*, *msg*.
  - dynamic monitored: *input*, *shippingChoice*.
  - static: *isValidArtID*, *isValidCustID*.

  La differenza tra **dynamic** e **static** è dovuta al variare (o meno) del valore della funzione da uno stato all'altro. Inoltre, **controlled** indica le funzioni lette e scritte dall'ASM corrente, mentre **monitored** indica quelle lette dall'ASM corrente e scritte dall'ambiente dell'ASM (nel nostro caso, si tratta di input dell'utente).

- Transition rules: *r_insertCustID*, *r_checkCustID*, *r_choice*, *r_checkArtID*, *r_ordConf*, *r_ship*, *r_invExp*, *r_invCheap*, *r_Main*.

2.2 Macchina a stati

Il funzionamento della macchina a stati è il seguente:

- Il cliente dell'azienda fornisce il suo codice identificativo e:
  - se è corretto, può scegliere un articolo.
  - altrimenti viene richiesto di inserire un codice identificativo valido.
- Il cliente sceglie un articolo che vuole ordinare (inserendo il corrispondente codice identificativo) e:
  - se esiste, viene ordinato l'articolo scelto.
  - altrimenti viene richiesto di inserire un codice articolo valido.
- Il cliente sceglie il tipo di spedizione di cui vuole usufruire (veloce o lenta). In entrambi i casi, si viene portati in uno stato *INVOICE*, in cui l'unica cosa che cambia è il messaggio che viene dato in output verso il cliente ad indicare la scelta effettuata.
- Una volta emessa la fattura, si torna allo stato iniziale, in cui si potrà chiedere nuovamente il codice identificativo di un cliente.

La macchina a stati del progetto è visibile in Figura 1.

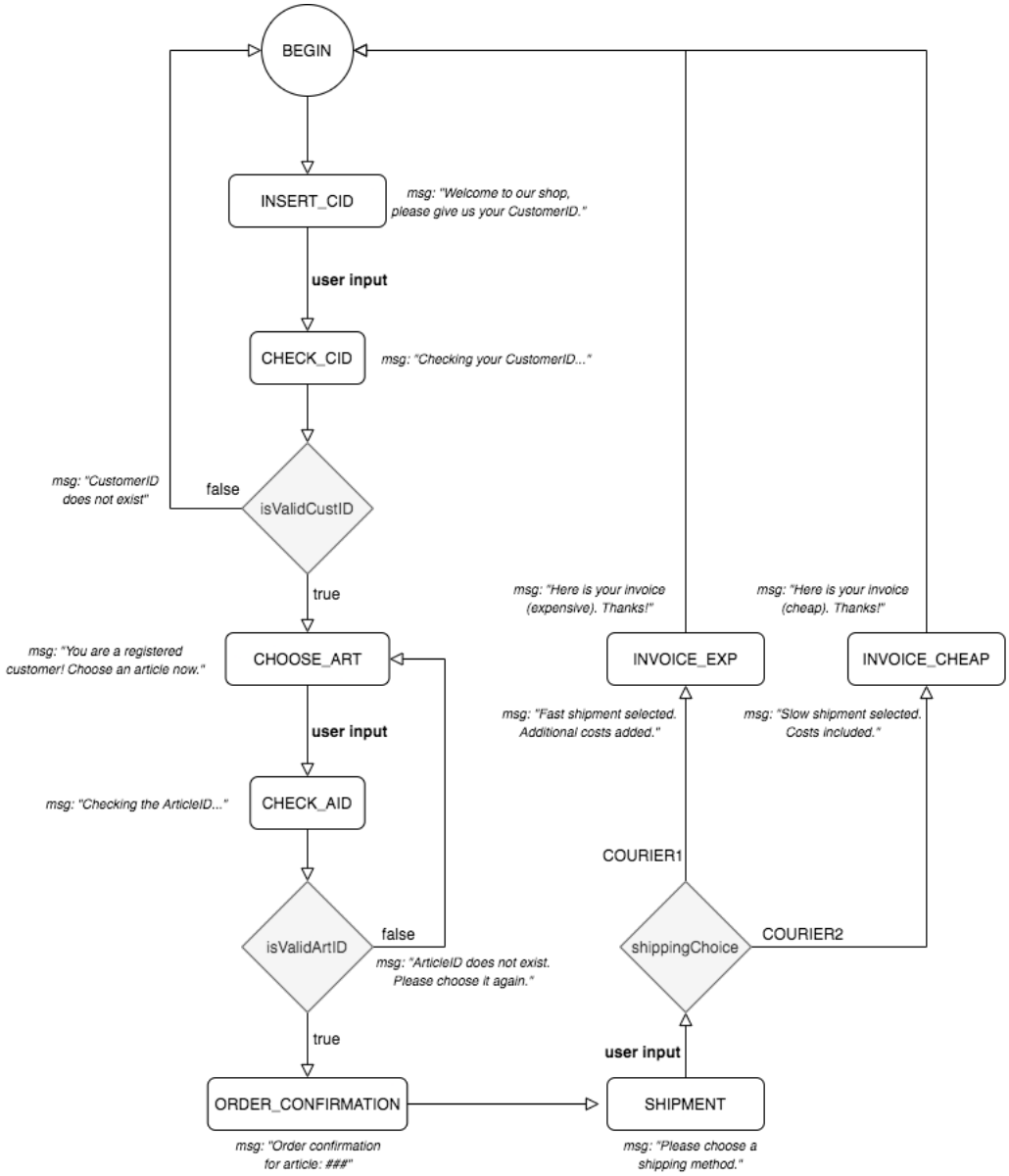

Figura 1: State machine di ASM-commerce

3 Costrutti utilizzati

Per lo sviluppo del progetto in considerazione, sono stati utilizzati i seguenti costrutti tipici del linguaggio AsmetaL:

- Block rules

```
par
  ...
endpar
```

- Costrutti if-else

```
if(isValidArtID(currArtID)) then
  ...
else
  ...
endif
```

- Definizione dello stato iniziale della macchina

```
default init s0:
  function currState = BEGIN
```

- Main rule

```
main rule r_Main =
  if(currState = BEGIN) then
    par
      currState := INSERT_CID
      msg := "Welcome to our shop, please give us
         your CustomerID."
    endpar
  else
    par
      r_insertCustID[]
```

```
11        r_checkCustID[]
12        r_choice[]
13        r_checkArtID[]
14        r_ordConf[]
15        r_ship[]
16        r_invExp[]
17        r_invCheap[]
18      endpar
19    endif
```

- Update rules (aggiornamento delle locazioni)

```
1 msg := "Checking the ArticleID..."
```

- Update rules (aggiornamento dello stato della macchina)

```
1 currState := SHIPMENT
```

- Variabili logiche

```
1 function isValidArtID($id in Integer) =
2   if(exist $a in IdArticle with $a = $id) then
3     true
4   else
5     false
6 endif
```

4 Caso d'uso: Animator

Attraverso l'animatore integrato AsmetaA, è possibile simulare e visualizzare la successione degli stati della macchina a stati del progetto.

Nelle seguenti Figure, è possibile quindi vedere un esempio di esecuzione.

	Type	Functions	State 0	State 1	State 2	State 3
	C	currState	BEGIN	INSERT_CID	CHECK_CID	CHOOSE_ART
	C	msg		Welcome to our shop, please give us your CustomerID.	Checking your CustomerID...	You are a registered customer! Choose an article now.
	M	input		202	202	
	C	currCustID			202	202

Figura 2: State 0 - 3

	Type	Functions	State 4	State 5	State 6
	C	currState	CHECK_AID	ORDER_CONFIRMATION	SHIPMENT
	C	msg	Checking the ArticleID...	Order confirmation for article: 99001	Please choose a shipping method.
	M	input	99001	99001	99001
	C	currCustID	202	202	202
	C	currArtID	99001	99001	99001
	M	shippingChoice			COURIER1

Figura 3: State 4 - 6

	Type	Functions	State 7	State 8
	C	currState	INVOICE_EXP	BEGIN
	C	msg	Fast shipment selected. Additional costs added.	Here is your invoice (expensive). Thanks!
	M	input	99001	99001
	C	currCustID	202	202
	C	currArtID	99001	99001
	M	shippingChoice	COURIER1	COURIER1

Figura 4: State 7 - 8

Università degli Studi di Bergamo
Scuola di Ingegneria
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione
Corso di Laurea Magistrale in Ingegneria Informatica

Informatica III
Modulo di Programmazione

PhotographicLenses++

Prof. Angelo Gargantini

Progetto C++ di:

Emanuele Perico

Anno Accademico 2019-2020

Indice

1 Introduzione

Nel progetto *PhotographicLenses++*, scritto in linguaggio C++, sono stati utilizzati diversi costrutti tipici di questo linguaggio di programmazione, come per esempio: distruttori, enumerativi, ereditarietà a diamante, ereditarietà privata e metodi virtual.

Lo scenario scelto per questo progetto sono gli obiettivi fotografici: in particolare, facendo uso delle classi e dell'ereditarietà si è voluto creare un piccolo esempio per poter differenziare le diverse categorie esistenti.

2 Programma

Il programma si occupa della categorizzazione delle diverse tipologie di obiettivi fotografici presenti sul mercato.
Il programma ha l'obiettivo di creare un output ordinato in cui ogni obiettivo fotografico contenga i parametri che lo caratterizzano.
Inoltre, ogni obiettivo fotografico è caratterizzato anche da un prezzo (in dollari americani) che sarà convertito (in euro) con un metodo ereditato dalle varie classi e quindi diversificato a seconda dell'obiettivo fotografico.

2.1 Struttura

La classe *Lens* è astratta, dovuto al fatto che al suo interno è dichiarato almeno un metodo virtuale puro, ed è la classe "padre" da cui ereditano tutte le altre classi, in particolare le classi figlie *Wide*, *Normal* e *Zoom*, le quali dovranno implementare i metodi della classe astratta.
Wide è a sua volta la classe "padre" della classe *Fisheye*; invece, *Normal* e *Zoom* sono le classi "padri" della classe *Macro*: quest'ultima è quindi soggetta ad ereditarietà multipla.
Si noti la struttura ereditaria a diamante, dovuta alla classe *Macro* che eredita da due classi che hanno la medesima classe "padre". Entrambe comunque hanno l'ereditarietà di tipo **virtual** dalla classe *Lens*.
Nella sua interezza, quindi, il programma si compone di sei classi, oltre al file **main**.
La struttura del progetto è visibile in Figura 1.

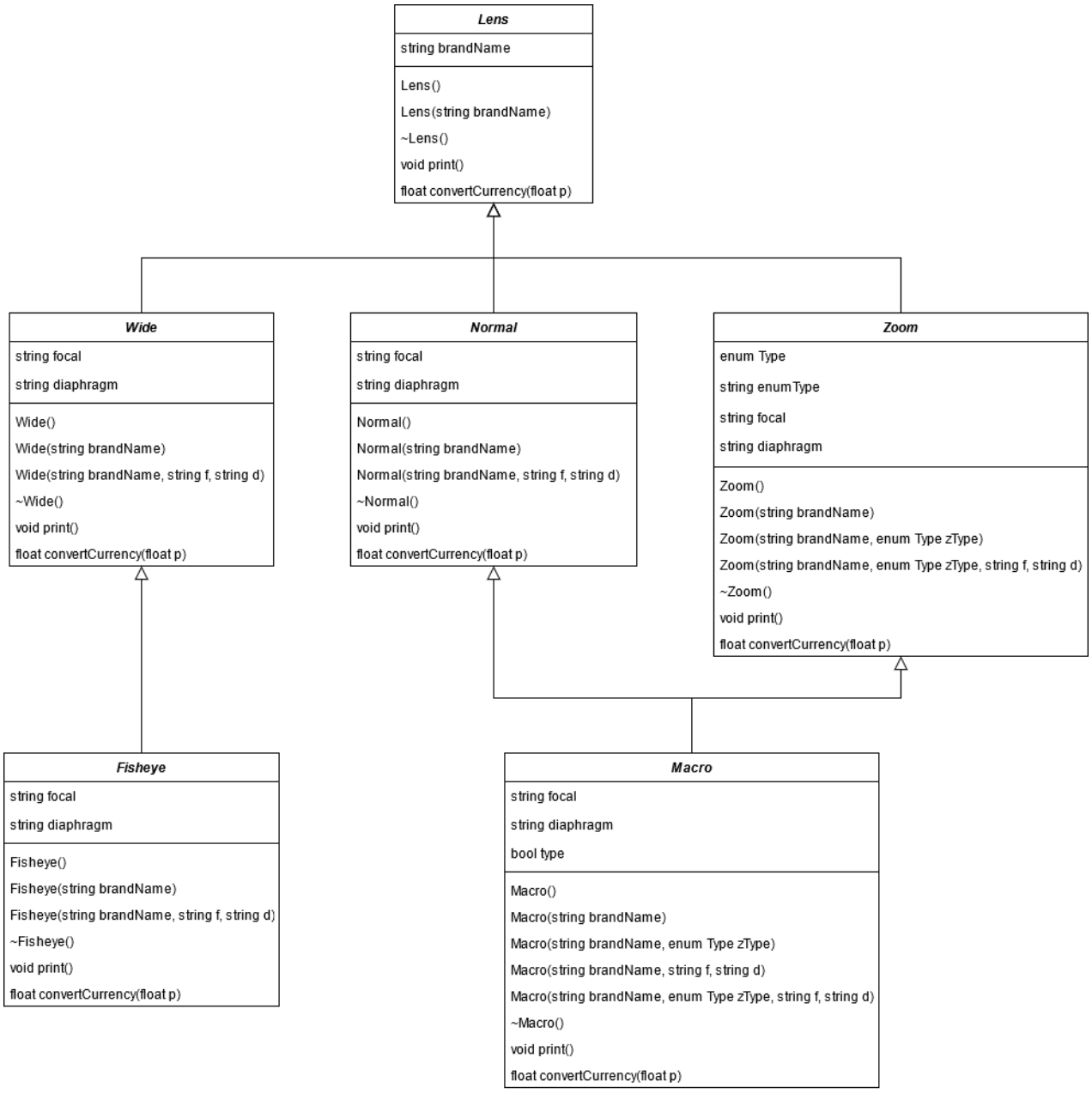

Figura 1: Class diagram di PhotographicLenses++

3 Costrutti utilizzati

Per lo sviluppo del progetto in considerazione, sono stati utilizzati i seguenti costrutti tipici del linguaggio C++:

- Costruttore con initialization list

```
Lens::Lens(string brandName):brandName(brandName){
  Lens::brandName = brandName;
  cout<<"Lens";
}
```

- Distruttori

```
Lens::~Lens(){
}
```

- Enumerativi

```
enum Type {
  zoom = 0,
  telephoto = 1,
  supertelephoto = 2
};
```

- Ereditarietà multipla (usando virtual per l'eredità a diamante)

```
class Macro: public Normal, public Zoom {
        public:
        string focal;
        string diaphragm;
        // false (0) > Normal, true (1) > Zoom
        bool type = false;
        Macro();
        Macro(string brandName);
        Macro(string brandName, enum Type zType);
```

```
10          Macro(string brandName, string f, string d);
11          Macro(string brandName, enum Type zType,
12          string f, string d);
13          ~Macro();
14          void print();
15          float convertCurrency(float p);
16  };
```

- Ereditarietà privata (si può accedere ai campi solo con metodi in *Lens*)

```
1  class Zoom: private virtual Lens {
2  public:
3    enum Type {zoom=0, telephoto=1,
4                 supertelephoto=2};
5    string enumType;
6    string focal;
7    string diaphragm;
8    Zoom();
9    Zoom(string brandName);
10   Zoom(string brandName, enum Type zType);
11   // etc...
12 };
```

- Metodo pure virtual

```
1  virtual float convertCurrency(float p) = 0;
```

- Overriding di metodi virtual

```
1  float Normal::convertCurrency(float p){
2    cout<<"Currency (USD to EUR) using method from
3             class Normal: "<<p/1.20<<" EUR."<<endl;
4    return p/1.20;
5  }
```

4 Output

Una volta creati diversi oggetti rappresentativi delle varie classi definite in questo progetto, il file **main** restituirà a schermo un output come il seguente.

```
Object 1 generated as: Lens > Normal
Brand1 lens is a normal.
Focal: Unknown
Diaphragm: Unknown

Object 2 generated as: Lens > Zoom
Brand2 lens is a zoom.
Type: Unknown
Focal: Unknown
Diaphragm: Unknown

Object 3 generated as: Lens > Normal
Brand3 lens is a normal.
Focal: 50mm
Diaphragm: f/1.8
Currency (USD to EUR) using method from
   class Normal: 74.9917 EUR.

Object 4 generated as: Lens > Zoom
Brand4 lens is a zoom.
Type: telephoto
Focal: 100-400mm
Diaphragm: f/4.5-6.3
Currency (USD to EUR) using method from
   class Zoom: 714.158 EUR.
```

```
Object 5 generated as: Lens > Normal > Macro
Brand5 lens is a normal.
Focal: 105mm
Diaphragm: f/2.8
It is a macro.
Currency (USD to EUR) using method from
   class Macro: 754.992 EUR.

Object 6 generated as: Lens > Zoom > Macro
Brand6 lens is a zoom.
Type: zoom
Focal: 17-70mm
Diaphragm: f/2.8-4
It is a macro.
Currency (USD to EUR) using method from
   class Macro: 349.992 EUR.

Object 7 generated as: Lens > Wide
Brand7 lens is a wide-angle.
Focal: 28mm
Diaphragm: f/2.8
Currency (USD to EUR) using method from
   class Wide: 76.6583 EUR.

Object 8 generated as: Lens > Wide > Fisheye
Brand8 lens is a wide-angle and fish-eye.
Focal: 16mm
Diaphragm: f/2.8
Currency (USD to EUR) using method from
   class Fisheye: 866.658 EUR.
```

Università degli Studi di Bergamo
Scuola di Ingegneria
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione
Corso di Laurea Magistrale in Ingegneria Informatica

Informatica III
Modulo di Programmazione

SWM: ScalaWarehouseManager

Prof. Angelo Gargantini

Progetto Scala di:
Emanuele Perico

Anno Accademico 2019-2020

Indice

1 Introduzione

Nel progetto *ScalaWarehouseManager*, scritto in linguaggio Scala, sono stati utilizzati diversi costrutti tipici di questo linguaggio di programmazione, come per esempio: ereditarietà multipla, operazioni sulle collezioni (**filter**, **fold**, **foreach**, **map**, **reduce**), programmazione orientata alle espressioni e traits.
Lo scenario scelto per questo progetto è quello della gestione di un magazzino: in particolare, si è voluto creare un piccolo esempio per poter inserire, cercare ed eliminare degli articoli di merce in stock.

2 Programma

Il programma si occupa della gestione di diversi articoli in stock in un magazzino.
Il programma ha l'obiettivo di gestire il corretto inserimento di nuovi articoli, ricercare un determinato articolo richiesto, eliminare un determinato articolo e gestire i prezzi di ogni articolo, eventualmente anche cambiandoli.

2.1 Struttura

All'interno di questo progetto, sono presenti rispettivamente:

- Traits: *Article*, *Logger*, *StockControl*.
  Sono simili alle interfacce di Java 8 e vengono utilizzati per condividere campi e metodi tra le classi.

- Classes: *Item*, *WarehouseDB*.
  Se si deve far uso dell'ereditarietà multipla, bisogna fare attenzione a come vengono ereditate le classi astratte o i traits: si può ereditare al massimo una classe astratta, mentre non c'è limite sull'eredità dei traits. Bisogna far uso delle keywords **extends** (per la prima classe/trait ereditata) e **with** (per le successive classi/trait ereditate).

- Object: *Main*.
  Si tratta di un Singleton, ovvero una classe che può avere una sola istanza.

La classe *Item* eredita da *Article* i suoi campi, mentre da *Logger* il metodo **log** che serve per visualizzare l'output (un esito relativo all'operazione svolta).
La classe *WarehouseDB* salva tutti gli **Item** in un **Array** e implementa i metodi definiti nel trait *StockControl*.
Ovviamente, i dati non sono salvati in modo persistente, ma vengono salvati solamente durante il runtime.
La struttura del progetto è visibile in Figura 1.

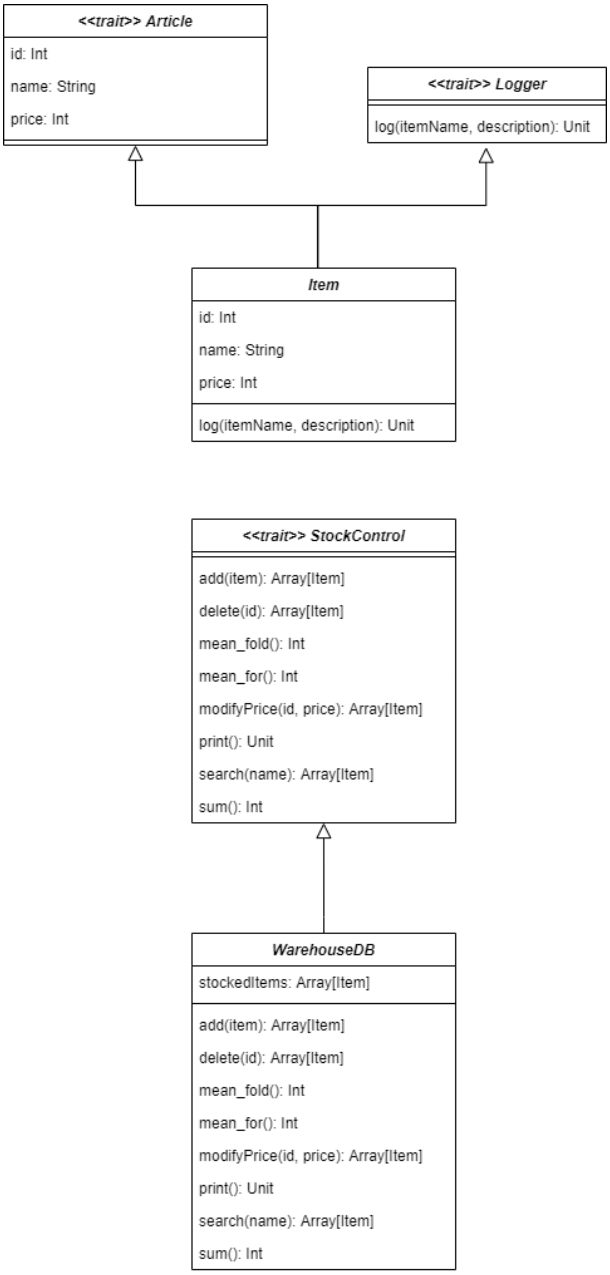

Figura 1: Class diagram di ScalaWarehouseManager

3 Costrutti utilizzati

Per lo sviluppo del progetto in considerazione, sono stati utilizzati i seguenti costrutti tipici del linguaggio Scala:

- Ereditarietà multipla

```
class Item(itemId: Int, itemName: String,
        itemPrice: Int) extends Logger with Article {
  def log(itemName: String, description: String):
    Unit = println(itemName + ": " + description)
  var id: Int = itemId
  var name: String = itemName
  var price: Int = itemPrice
}
```

- Operazioni sulle collezioni: **filter**

```
stockedItems = stockedItems.filter(x => x.id != id)
```

- Operazioni sulle collezioni: **fold**

```
def mean_fold(): Int = stockedItems.foldLeft(0)
  ((x,y) => x + y.price)/stockedItems.length
```

- Operazioni sulle collezioni: **foreach**

```
stockedItems.foreach(i => println(i.id + " - " +
                i.name + " - " + i.price))
```

- Operazioni sulle collezioni: **map-reduce**

```
def sum(): Int = stockedItems
                .map(x => x.price)
                .reduce((a,b) => a + b)
```

- Programmazione orientata alle espressioni

```
1 val removeItem: Option[Item] =
2 stockedItems.find(x => x.id == id)
3 removeItem match {
4   case None => println("Error: item with id " +
5     id + " was not found.")
6   case Some(found) => found.log(found.name,
7     "was deleted.")
8 }
```

Si noti che **removeItem** è di tipo **Option**[**T**]: un contenitore per nessun o un elemento di tipo **T**. Un **Option**[**T**] può essere un oggetto **Some**[**T**] oppure **None** (a rappresentare la mancanza di un valore).

- Traits

```
1 trait Article {
2   var id: Int
3   var name: String
4   var price: Int
5 }
```

4 Output

Una volta creati diversi oggetti rappresentativi degli articoli in stock e le relative operazioni da eseguire, il file **Main** restituirà a schermo un output come il seguente.

```
article1: was added correctly.
-----------------------------------------------
Stock availability (ID - Name - Price)
1 - article1 - 600
Sum of the items' prices in stock: 600
-----------------------------------------------
```

```
article2: was added correctly.
Error: item with id 1 already exists in stock.
  => article3 not added.
article4: was added correctly.
article5: was added correctly.
article: was added correctly.
article: was added correctly.
-----------------------------------------------
Stock availability (ID - Name - Price)
1 - article1 - 600
2 - article2 - 900
4 - article4 - 100
5 - article5 - 70
6 - article - 1200
7 - article - 1400
Sum of the items' prices in stock: 4270
-----------------------------------------------
article5: was deleted.
-----------------------------------------------
Stock availability (ID - Name - Price)
1 - article1 - 600
2 - article2 - 900
4 - article4 - 100
6 - article - 1200
7 - article - 1400
Sum of the items' prices in stock: 4200
-----------------------------------------------
Number of available items with name article: 2
article: id 6 is available in stock.
article: id 7 is available in stock.
-----------------------------------------------
```

```
Changing price to item 2...
Old price: 900
New price: 800
Price for item 2 changed correctly.
-----------------------------------------------
Stock availability (ID - Name - Price)
1 - article1 - 600
2 - article2 - 800
4 - article4 - 100
6 - article - 1200
7 - article - 1400
Sum of the items' prices in stock: 4100
-----------------------------------------------
Average price (using fold): 820
Average price (using for): 820
-----------------------------------------------
```