Assignment 2
Analysis and Design Document

Student: Bolba Raluca
Group: 30235

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

Să se proiecteze și implementeze o aplicație pentru angajații unui magazin de cărți. Aplicația trebuie să aibe două tipuri de utilizatori : un utilizator obișnuit reprezentat de angajatul magazinului și un administrator care trebuie să furnizeze un username și o parolă pentru a utiliza aplicația. Informațiile despre utilizatori, cărți și vânzări vor fi stocate multiple fișiere XML. Să se utilizeze pattern-ul Model View Controller în proiectarea aplicației. Să se utilizeze design pattern-ul Factory Method pentru generarea rapoartelor.

1.2 Functional Requirements

Utilizatorul obișnuit (angajatul) trebuie să efectueze următoarele operații:

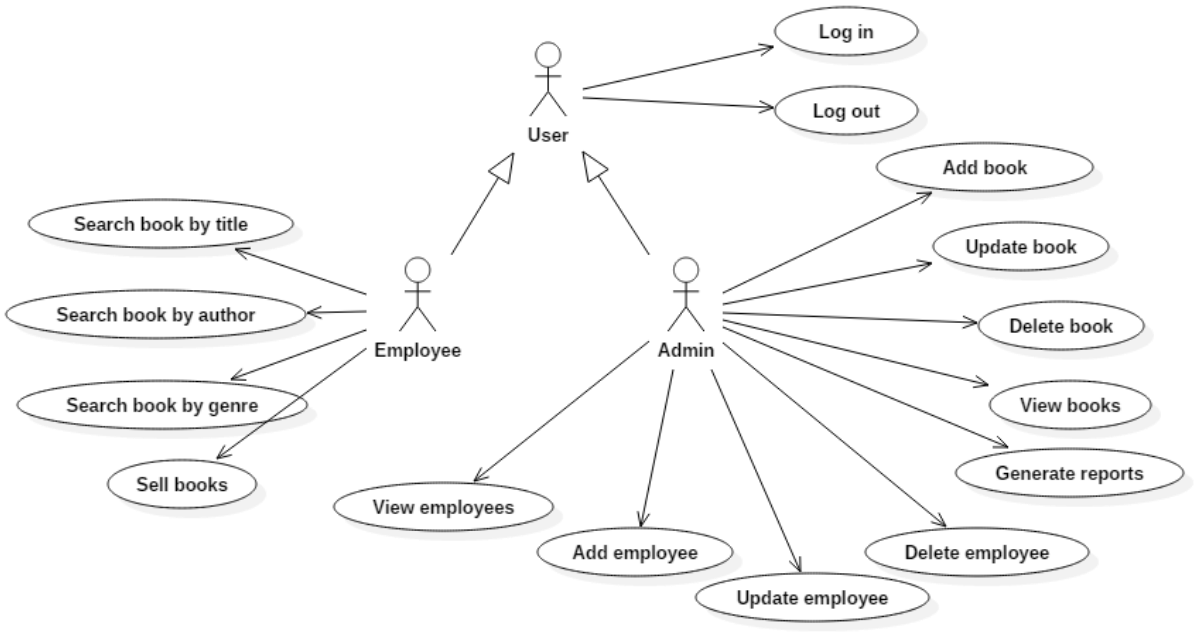
- Să caute cărți după gen, titlu sau autor
- Să vândă cărți

Administratorul trebuie să efectueze următoarele operații:

- CRUD pe cărți
- CRUD pe informațiile utilizatorilor
- Să genereze rapoarte într-un fișier *txt* sau *xml* cu cărțile care nu mai sunt pe stoc

2. Use-Case Model

Use case general:

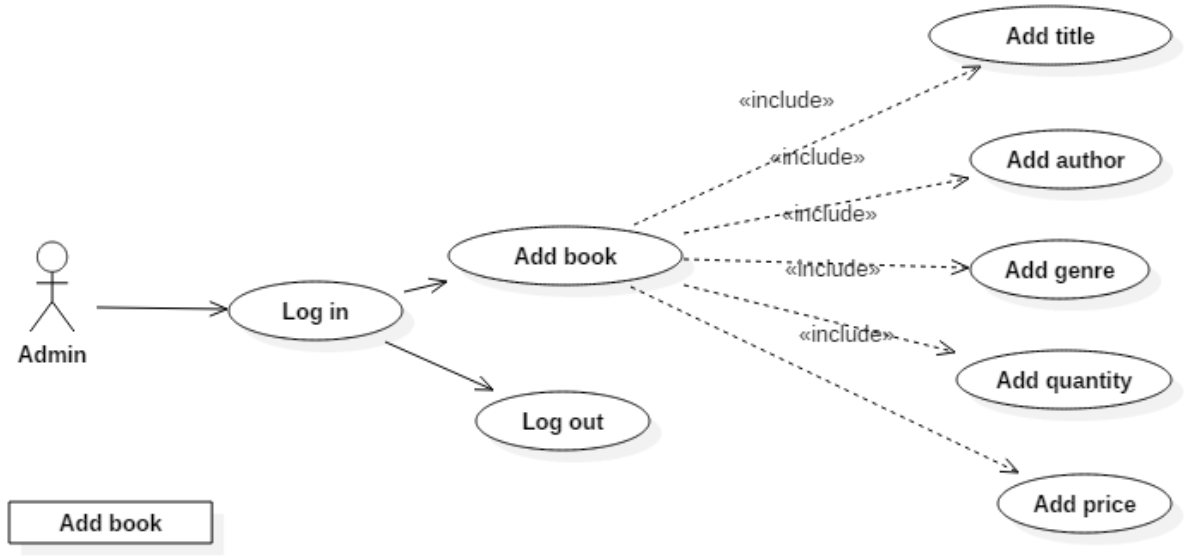


Use case: **Adăugare carte**
Level: User-goal
Primary actor: Administrator
Main success scenario:

1. Utilizatorul se autentifică la sistem
2. Administratorul alege să efectueze operația de adăugare carte
3. Administratorul introduce informațiile despre carte
    3.1. Administratorul introduce titlul cărții
    3.2. Administratorul introduce autorul cărții
    3.3. Administratorul introduce genul cărții
    3.4. Administratorul introduce numărul de cărți
    3.5. Administratorul introduce prețul cărții
4. Administratorul adaugă cartea în magazin
5. Administratorul se deconectează

Extensions:

1a. Datele de autentificare sunt incorecte
    1a1.Se revine la pasul 1
3a. Cartea există deja în magazin
    3a1. Se revine la pasul 3
3b. Numărul de exemplare este negativ
    3b1. Se revine la pasul 3.4
3c. Prețul este negativ
    3c1. Se revine la pasul 3.5

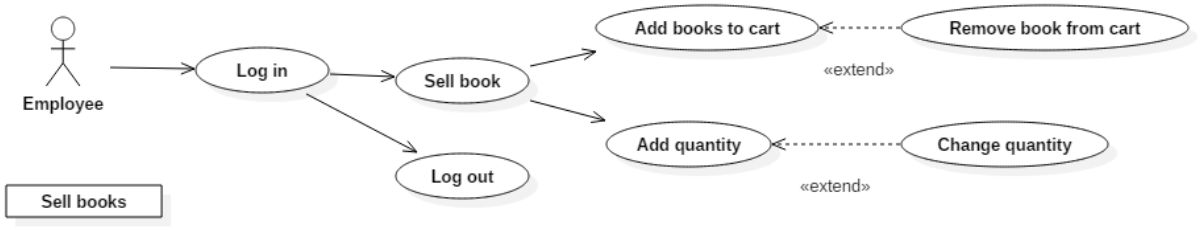


Use case: **Vânzare cărți**
Level: User-goal
Primary actor: Angajat
Main success scenario:

1. Utilizatorul se autentifică la sistem
2. Angajatul alege să efectueze operația de vânzare cărți
3. Angajatul adaugă în coș cărțile pe care vrea să le vândă
4. Angajatul finalizează vânzarea
    4.1. Coșul de cumpărături este golit
5. Angajatul se deconectează

Extensions:

1a. Datele de autentificare sunt incorecte
    1a1.Se revine la pasul 1
3a. Cărțile nu există pe stoc
    3a1. Se revine la pasul 3
3b. Numărul de exemplare dorit este negativ
    3b1. Se revine la pasul 3
3c. Numărul de exemplare dorit este mai mare decât numărul de exemplare existente pentru cartea respectivă
    3c1. Se revine la pasul 3
3d. Numărul de exemplare nu este cel dorit
    3d1. Se modifică numărul de exemplare
3e. Cărțile nu sunt cele dorite
    3e1. Cărțile nedorite se înlătură din coșul de cumpărături



3. System Architectural Design

3.1 Architectural Pattern Description

Pattern-ul architectural folosit pentru această aplicație este pattern-ul MVC (Model View Controller) care, așa cum sugerează numerele, structurează aplicația în 3 categorii : Model, View și Controller.

Partea de *Model* este responsabilă de păstrarea stării curente a aplicației sau a unui set de date. Pe lângă date, care sunt reprezentări ale conceptelor utilizate în aplicație, mai conține și logica care se aplica pe aceste date. Modelul nu are nici o informație cu privire la felul în care sunt reprezentate datele către utilizatori într-un mod grafic. Acesta oferă în schimb Controller-ului metode prin care starea curentă a aplicației poate fi accesată sau modificată. În cazul

acestei aplicații, modelul este reprezentat din clasele prezente în pachete *entities* (care conține clasele ce modelează conceptele întâlnite în acest domeniu, și anume : *Book, User, Admin, Employee, Sale, Bookstore, ShoppingCart*), *factory* (care conține clasele și interfața care se ocupă de implementarea design pattern-ului *Factory Method*) *și dataAcces* (care conține clasele ce se ocupă de citirea, scrierea sau modificarea datelor din model în fișiere xml sau txt).

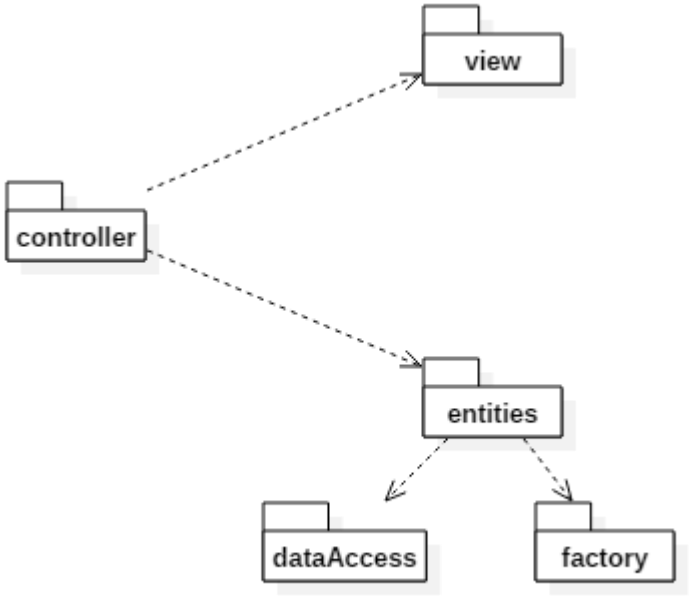
Partea de *View* se ocupă de reprezentarea într-o manieră grafică a datelor primite de la model. Pot exista de exemplu mai multe moduri de reprezentare pentru același set de date dintre care Controller-ul poate alege unul. În cazul acestei aplicații, partea de View corespunde pachetului *view* ce conține doar clasa *Window* ce reprezintă interfața grafică cu utilizatorul.

Partea de *Controller* este partea ce se ocupă de procesarea cererilor venite de la utilizator, în acest caz ascultătorii de evenimente pentru componentele din interfața grafică, iar datele primite de la utilizator sunt eventual modificate și transmise către partea de model pentru a modifica starea curentă a sistemului. Această parte mai are rolul de a păstra modelul și view-ul sincronizate. În cadrul acestei aplicații, partea de Controller revine pachetului *controller* format din clasele *Main* și *Controller*, cea din urmă conținând toate metodele și ascultătorii utilizate pentru a păstra modelul și view-ul sincronizate.

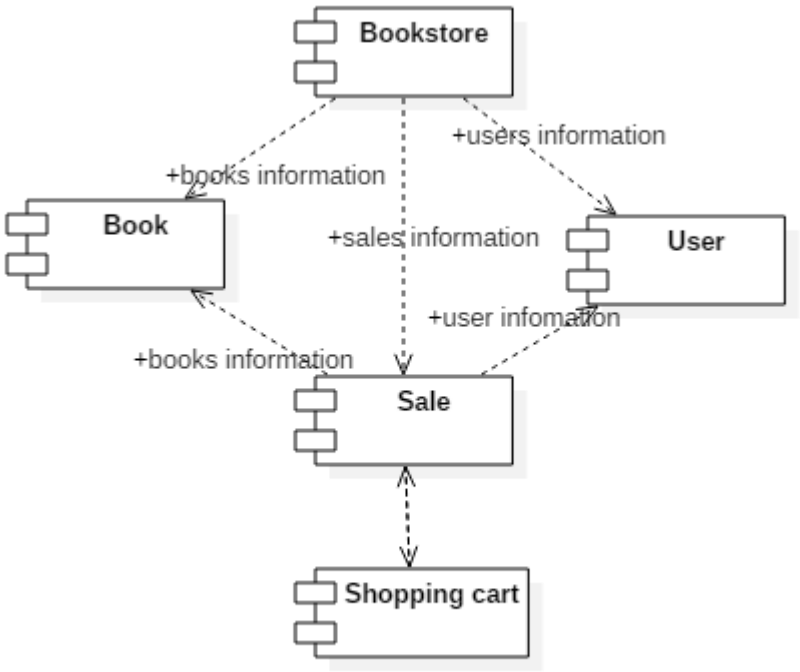
3.2 Diagrams

Diagramă de pachete

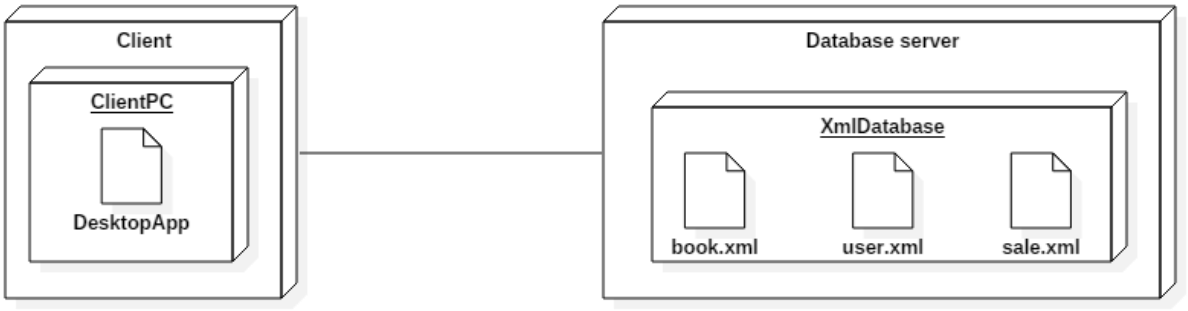
Pachetele corespund pattern-ului architectural MVC conform următoarelor: partea de View corespunde pachetului *view*, partea de Controller corespunde pachetului *controller* iar partea de Model este formată din pachetele *entities, dataAccess și factory.*



Diagramă de componente

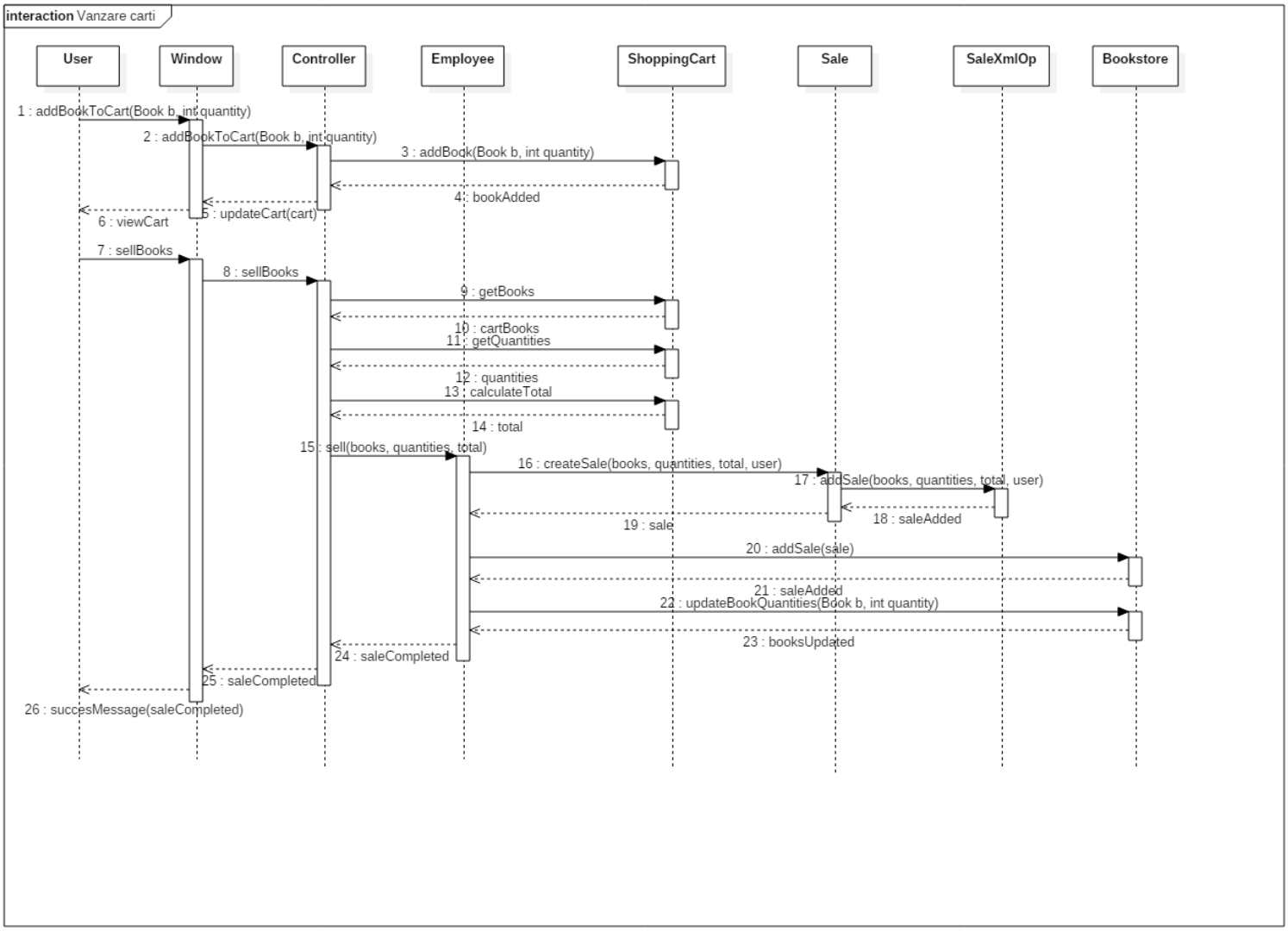


Deployment diagram



4. UML Sequence Diagrams

Diagramă de secvență pentru *Vânzare cărți.*



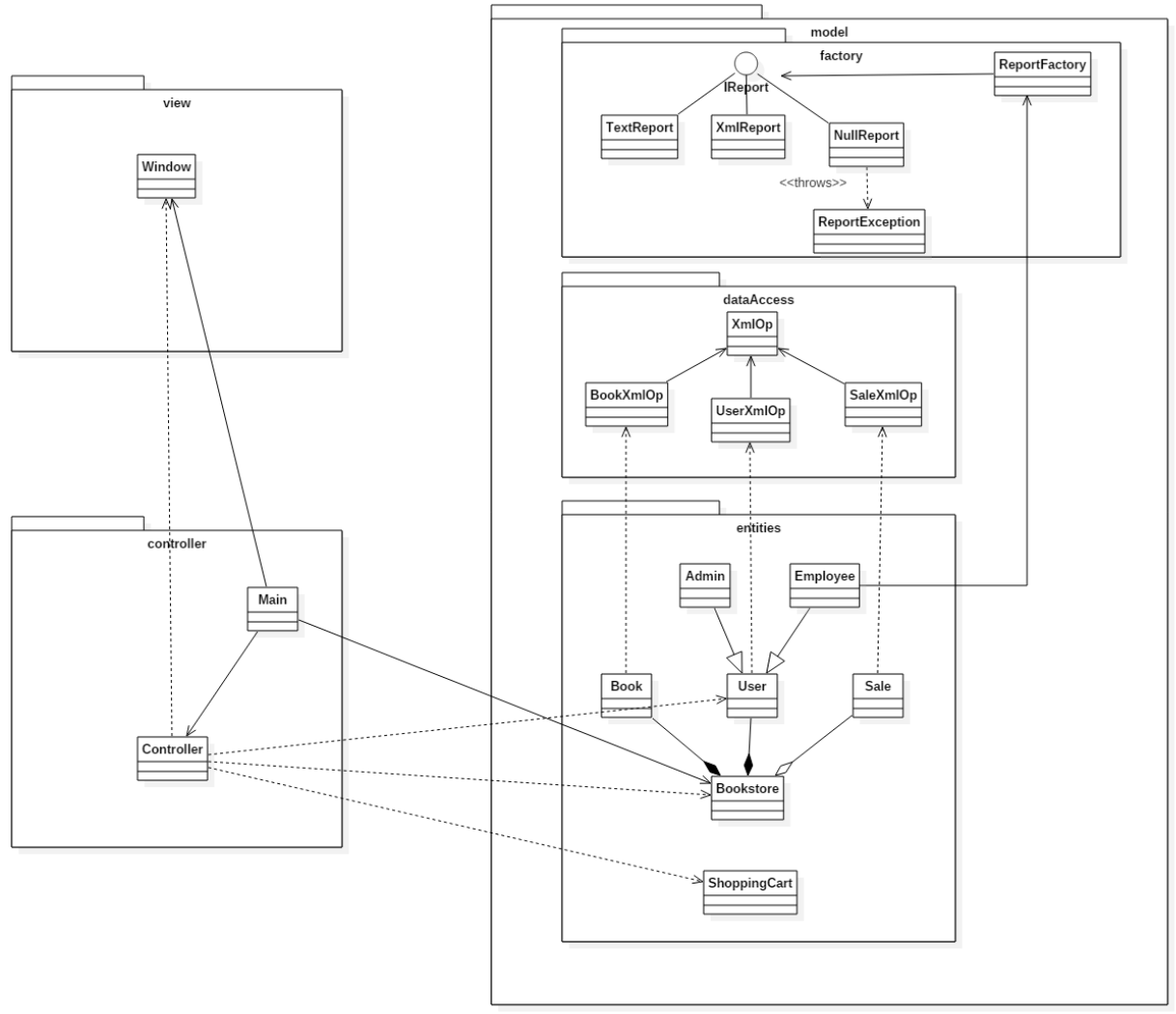
5. Class Design

5.1 Design Patterns Description

Design pattern-ul utilizat în această aplicație este *Factory Method*, un pattern creational care facilitează creare unor obiecte într-un mod indirect. Mai concret, se definește o interfață pentru crearea unui obiect, dar subclasele vor decide ce clasă trebuie instanțiată. Modul de creare a obiectelor este transparent astfel pentru utilizator întrucât el se raportează la o interfață, nu la implementarea propriu-zisă.

5.2 UML Class Diagram

În cazul acestei aplicații, pattern-ul este utilizat pentru generarea unui raport a cărților din bibliotecă care nu mai sunt pe stock, fie într-un fișier xml, fie în unul txt. Interfața pentru crearea unui obiect de tip raport este *IReport*, care este implementată de clasele concrete *TextReport*, *XmlReport și NullReport*. Clasa din urmă generează o excepție de tip *ReportException* în cazul în care ajunge să fie instanțiată, adică atunci când se dorește crearea unui raport de alt tip decât xml sau txt. Clasa ”creatoare”, adică cea care decide ce obiect trebuie creat este clasa *ReportFactory*. Dacă se dorește de exemplu crearea unui raport de tip xml, metoda *getReport(”XMLREPORT”)* din ReportFactory va fi apelată și va returna un obiect de tip *IReport* care este o instanță a clasei *XmlReport*.



6. Data Model

Datele acestei aplicații au fost păstrate în fișiere *XML*. Mai precis, pentru entitățile *Book, User și Sale* s-a folosit câte un fișier xml separat.

Fișierul *XML* care păstrează datele referitoare la cărți are ca rădăcină nodul *bookstore,* iar copiii acestui nod sunt cărțile care se află în bibliotecă. Fiecare carte (ce are ca nod *book*) are

un atribut de identificator asociat, pentru a facilita modificarea sau ștergerea unei astfel de componente. Copiii nodului *book* sunt *title, author, genre, quantity, price* care reprezintă de fapt informațiile despre o carte și care au asociate valori de tip text, după cum se poate observa în imaginea următoare:

```
▼<bookstore>
  ▼<book id="1">
      <title>Jane Eyre</title>
      <author>Charlotte Bronte</author>
      <genre>Romance</genre>
      <quantity>80</quantity>
      <price>50.0</price>
    </book>
  ▼<book id="2">
      <title>Memoirs of a Geisha</title>
      <author>Arthur Golden</author>
      <genre>Historical</genre>
      <quantity>20</quantity>
      <price>49.9</price>
    </book>
  ▼<book id="3">
      <title>Singur pe lume</title>
      <author>Hector Mallot</author>
      <genre>Novel</genre>
      <quantity>0</quantity>
      <price>10.0</price>
    </book>
  ▼<book id="4">
      <title>Poezii</title>
      <author>Mihai Eminescu</author>
      <genre>Poezie</genre>
      <quantity>9</quantity>
      <price>15.0</price>
    </book>
  </bookstore>
```

Pentru fișierul care păstrează datele de vânzările effectuate, structura este asemănătoare, atâta doar că un nod de tip *selling* are ca și copii o listă de cărți (*books* cu nodurile copii *book*), cu anumite detalii despre acestea, cât și cantitatea cumpărată, prețul (*totalprice)* total al vânzării, angajatul (*employee* împreună cu id-ul acestuia și care are ca nod copil doar numele angajatului) care a efectuat vânzarea și data (*date)* la care s-a efectuat aceasta. Structura arborescentă populată cu anumite date poae fi observată în continuare:

```
<sellings>
  <selling id="1">
    <books>
      <book>
        <title>Memoirs of a Geisha</title>
        <author>Arthur Golden</author>
        <genre>Historical</genre>
        <quantity>1</quantity>
        <unitprice>49.9</unitprice>
      </book>
    </books>
    <totalprice>49.9</totalprice>
    <date>2016-04-10</date>
    <employee id="1">
      <username>raluca</username>
    </employee>
  </selling>
  <selling id="2">
    <books>
      <book>
        <title>Jane Eyre</title>
        <author>Charlotte Bronte</author>
        <genre>Romance</genre>
        <quantity>1</quantity>
        <unitprice>50.0</unitprice>
      </book>
    </books>
    <totalprice>50.0</totalprice>
    <date>2016-04-10</date>
    <employee id="1">
      <username>raluca</username>
    </employee>
  </selling>
```

Pentru fișierul ce păstrează datele despre utilizatorii aplicației, acesta are ca rădăcină nodul *userList* ce are ca noduri fii utilizatorii *(user*), fiecare cu un atribut de identificator, și cu o listă de noduri fii are reprezintă informațiile referitoare la aceștia, în acest caz: numele de utilizator, parola și tipul utilizatorului (administrator/employee).

```
<userList>
  <user id="1">
    <username>raluca</username>
    <password>maria</password>
    <type>employee</type>
  </user>
  <user id="2">
    <username>admin</username>
    <password>admin</password>
    <type>administrator</type>
  </user>
</userList>
```

7. System Testing

Pentru testarea aplicației aceasta s-a rulat și s-a verificat dacă operațiile sunt realizate în mod corect. De exemplu pentru operația de autentificare s-a verificat dacă datele introduse sunt cele prezente și în baza de date, iar în caz contrar s-a afișat un mesaj de eroare.

8. Bibliography

https://dzone.com/articles/design-patterns-factory
http://code.tutsplus.com/tutorials/mvc-for-noobs--net-10488
http://www.mkyong.com/tutorials/java-xml-tutorials/