Algoritmos e Estruturas de Dados

Gestão de uma Biblioteca

Turma 4 - Grupo B

José Peixoto	200603103	ei12134@fe.up.pt
Paulo Faria	201204965	ei12135@fe.up.pt
Pedro Moura	201306843	up201306843@fe.up.pt

7 de Novembro de 2014

Conteúdo

Resumo

Foi-nos proposto o desenvolvimento de uma aplicação em C++ para gestão de uma biblioteca que abrangesse livros, leitores, funcionários e supervisores. A biblioteca teria necessariamente que permitir, de forma controlada, o empréstimo de livros a leitores e gestão de ambos pelos funcionários. Os empréstimos, quando possíveis, teriam um período máximo de duração de 7 dias para cada livro, podendo o leitor incorrer numa multa caso não o devolvesse dentro deste prazo. Os supervisores estariam individualmente a cargo de uma equipa composta por funcionários.

1 Descrição da solução implementada

Após um período de interpretação do guião que nos foi fornecido, determinámos, através de esquemas, a relação entre as classes necessárias para implementar o pedido recorrendo a UML. Decidimos que a classe *L*ibrary teria um papel basilar ao armazenar e manipular as estruturas de dados essenciais à gestão de uma biblioteca, nomeadamente contentores de livros, leitores e funcionários. Estabelecemos uma relação de hierarquia de classes entre pessoas, leitores e funcionários, sendo estes dois últimos subclasses da classe abstrata *P*erson. Considerámos inapropriado a criação de uma subclasse *S*upervisor em relação à classe *E*mployee, por se tratar de uma solução quase forçada que recorre a uma relação hierárquica entre classes que, afinal, não diferem significativamente. Como um supervisor é sempre à partida um empregado, permitimos a sua promoção, recorrendo a um booleano *s*upervisor, membro dado da classe *E*mployee.

Na conceptualização de empréstimos, foi necessária a classe *B*orrow que associaria a cada empréstimo um livro, um leitor e um empregado responsável pela operação. Ainda a propósito da criação e gestão correta de um empréstimo, sentimos a necessidade de recorrer a uma classe que nos permitisse representar a data de forma suficientemente flexível para determinar facilmente as diferenças entre duas datas (empréstimo e entrega do livro).

2 Lista de casos de utilização da aplicação

A aplicação permite a criação, edição e remoção de livros, leitores, funcionários e empréstimos. É possível a visualização simplificada e ordenação de livros, leitores e funcionários. Na gestão de empréstimos, a aplicação também admite o cálculo de multa, caso o leitor não devolva o livro dentro do prazo estipulado.

3 Relato das principais dificuldades encontradas no desenvolvimento do trabalho

Mesmo tendo investido algum tempo no planeamento inicial, encontrámos algumas dificuldades na definição de uma representação ideal dos conceitos pedidos e que permitisse de igual forma satisfazer as operações requeridas. Fomos, portanto, de forma iterativa, reformulando as nossas abordagens a cada problema/incompatibilidades que foram surgindo. Tivemos também que transpor vários problemas no uso do software de gestão de repositórios *G*it, por não termos conhecimentos que nos permitissem fluxos de trabalho paralelos, sem conflitos, aquando da junção do código.

4 Indicação do esforço dedicado por cada elemento do grupo

4.1 Notas em grupo

Tentámos desde o início, trabalhando em grupo, fazer o esboço das classes e separação de tarefas de forma equitativa. Fomos trabalhando ora por turnos, ora por conferência via *S*kype.

4.1.1 José Peixoto

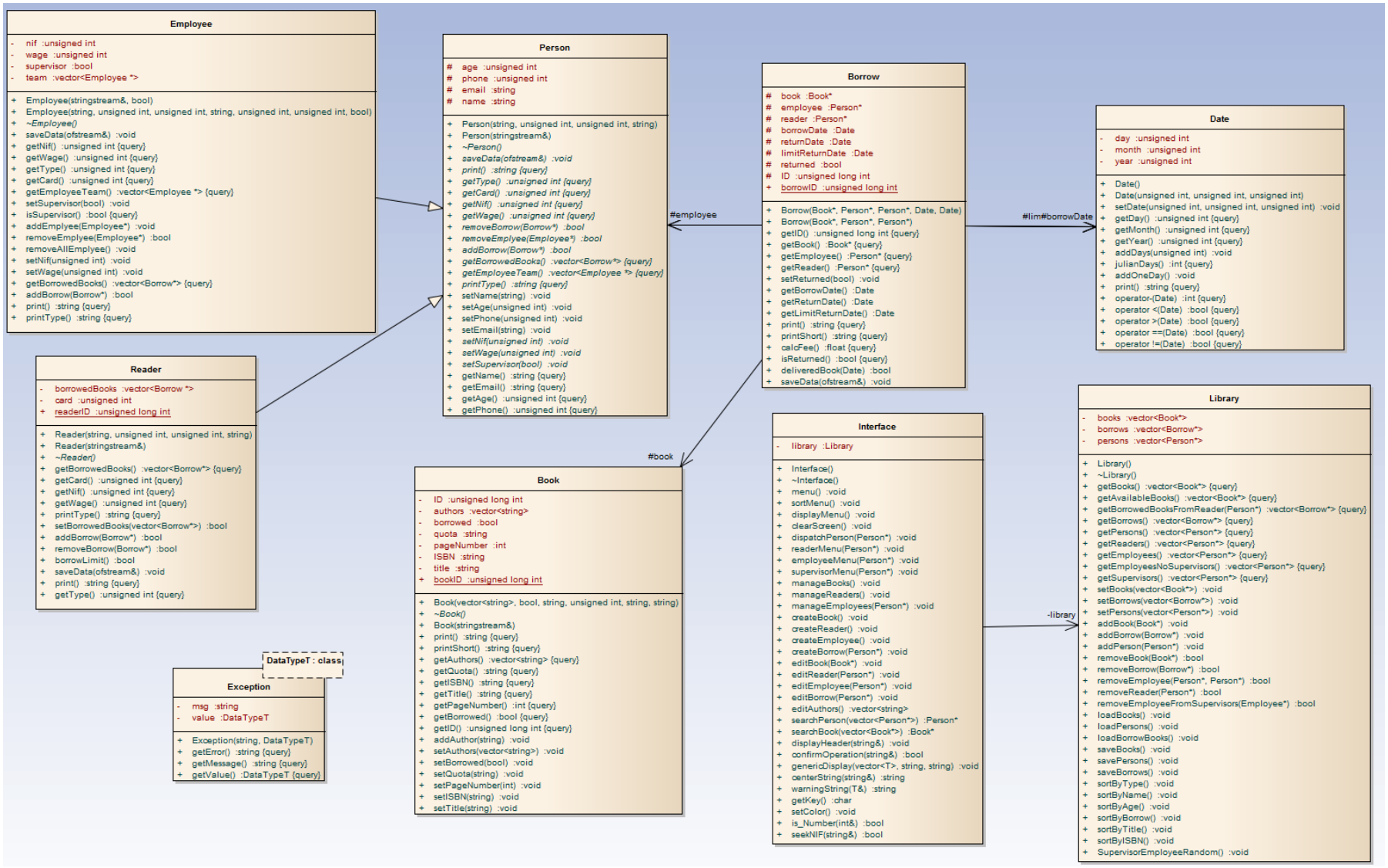
Tal como os restantes membros do grupo, participei desde o início na esquematização de uma solução do que nos fora pedido. Depois das tarefas divididas, estive mais envolvido nas classes *Library, Reader, Algorithms e Interface*, colaborando também, sempre que necessário, nas classes a cargo dos meus colegas. Além disso assumi, voluntariamente, a concretização do relatório final, incluindo as autoavaliações redigidas pelos restantes colegas.

4.1.2 Paulo Faria

Colaborei sobretudo nas classes *Interface, Book, Library*, mas tal como os meus colegas, sempre que o era necessário, também dei uma ajuda em outras classes.

4.1.3 Pedro Moura

Estive mais envolvido nas classes *Data, Exception, Borrow, Persons* e na leitura e escrita de ficheiros, colaborando também em outras partes do programa, quando necessário.


Employee
- nif :unsigned int
- wage :unsigned int
- supervisor :bool
- team :vector<Employee *>
+ Employee(stringstream&, bool)
+ Employee(string, unsigned int, unsigned int, string, unsigned int, unsigned int, bool)
+ ~Employee()
+ saveData(ofstream&) :void
+ getNif() :unsigned int {query}
+ getWage() :unsigned int {query}
+ getType() :unsigned int {query}
+ getCard() :unsigned int {query}
+ getEmployeeTeam() :vector<Employee *> {query}
+ setSupervisor(bool) :void
+ isSupervisor() :bool {query}
+ addEmplyee(Employee*) :void
+ removeEmplyee(Employee*) :bool
+ removeAllEmplyee() :void
+ setNif(unsigned int) :void
+ setWage(unsigned int) :void
+ getBorrowedBooks() :vector<Borrow*> {query}
+ addBorrow(Borrow*) :bool
+ print() :string {query}
+ printType() :string {query}
Person
age :unsigned int
phone :unsigned int
email :string
name :string
+ Person(string, unsigned int, unsigned int, string)
+ Person(stringstream&)
+ ~Person()
+ saveData(ofstream&) :void
+ print() :string {query}
+ getType() :unsigned int {query}
+ getCard() :unsigned int {query}
+ getNif() :unsigned int {query}
+ getWage() :unsigned int {query}
+ removeBorrow(Borrow*) :bool
+ removeEmplyee(Employee*) :bool
+ addBorrow(Borrow*) :bool
+ getBorrowedBooks() :vector<Borrow*> {query}
+ getEmployeeTeam() :vector<Employee *> {query}
+ printType() :string {query}
+ setName(string) :void
+ setAge(unsigned int) :void
+ setPhone(unsigned int) :void
+ setEmail(string) :void
+ setNif(unsigned int) :void
+ setWage(unsigned int) :void
+ setSupervisor(bool) :void
+ getName() :string {query}
+ getEmail() :string {query}
+ getAge() :unsigned int {query}
+ getPhone() :unsigned int {query}
Borrow
book :Book*
employee :Person*
reader :Person*
borrowDate :Date
returnDate :Date
limitReturnDate :Date
returned :bool
ID :unsigned long int
+ borrowID :unsigned long int
+ Borrow(Book*, Person*, Person*, Date, Date)
+ Borrow(Book*, Person*, Person*)
+ getID() :unsigned long int {query}
+ getBook() :Book* {query}
+ getEmployee() :Person* {query}
+ getReader() :Person* {query}
+ setReturned(bool) :void
+ getBorrowDate() :Date
+ getReturnDate() :Date
+ getLimitReturnDate() :Date
+ print() :string {query}
+ printShort() :string {query}
+ calcFee() :float {query}
+ isReturned() :bool {query}
+ deliveredBook(Date) :bool
+ saveData(ofstream&) :void
Date
- day :unsigned int
- month :unsigned int
- year :unsigned int
+ Date()
+ Date(unsigned int, unsigned int, unsigned int)
+ setDate(unsigned int, unsigned int, unsigned int) :void
+ getDay() :unsigned int {query}
+ getMonth() :unsigned int {query}
+ getYear() :unsigned int {query}
+ addDays(unsigned int) :void
+ julianDays() :int {query}
+ addOneDay() :void
+ print() :string {query}
+ operator-(Date) :int {query}
+ operator <(Date) :bool {query}
+ operator >(Date) :bool {query}
+ operator ==(Date) :bool {query}
+ operator !=(Date) :bool {query}
#employee
#lim#borrowDate
Reader
- borrowedBooks :vector<Borrow *>
- card :unsigned int
+ readerID :unsigned long int
+ Reader(string, unsigned int, unsigned int, string)
+ Reader(stringstream&)
+ ~Reader()
+ getBorrowedBooks() :vector<Borrow*> {query}
+ getCard() :unsigned int {query}
+ getNif() :unsigned int {query}
+ getWage() :unsigned int {query}
+ printType() :string {query}
+ setBorrowedBooks(vector<Borrow*>) :bool
+ addBorrow(Borrow*) :bool
+ removeBorrow(Borrow*) :bool
+ borrowLimit() :bool
+ saveData(ofstream&) :void
+ print() :string {query}
+ getType() :unsigned int {query}
#book
Book
- ID :unsigned long int
- authors :vector<string>
- borrowed :bool
- quota :string
- pageNumber :int
- ISBN :string
- title :string
+ bookID :unsigned long int
+ Book(vector<string>, bool, string, unsigned int, string, string)
+ ~Book()
+ Book(stringstream&)
+ print() :string {query}
+ printShort() :string {query}
+ getAuthors() :vector<string> {query}
+ getQuota() :string {query}
+ getISBN() :string {query}
+ getTitle() :string {query}
+ getPageNumber() :int {query}
+ getBorrowed() :bool {query}
+ getID() :unsigned long int {query}
+ addAuthor(string) :void
+ setAuthors(vector<string>) :void
+ setBorrowed(bool) :void
+ setQuota(string) :void
+ setPageNumber(int) :void
+ setISBN(string) :void
+ setTitle(string) :void
Interface
- library :Library
+ Interface()
+ ~Interface()
+ menu() :void
+ sortMenu() :void
+ displayMenu() :void
+ clearScreen() :void
+ dispatchPerson(Person*) :void
+ readerMenu(Person*) :void
+ employeeMenu(Person*) :void
+ supervisorMenu(Person*) :void
+ manageBooks() :void
+ manageReaders() :void
+ manageEmployees(Person*) :void
+ createBook() :void
+ createReader() :void
+ createEmployee() :void
+ createBorrow(Person*) :void
+ editBook(Book*) :void
+ editReader(Person*) :void
+ editEmployee(Person*) :void
+ editBorrow(Person*) :void
+ editAuthors() :vector<string>
+ searchPerson(vector<Person*>) :Person*
+ searchBook(vector<Book*>) :Book*
+ displayHeader(string&) :void
+ confirmOperation(string&) :bool
+ genericDisplay(vector<T>, string, string) :void
+ centerString(string&) :string
+ warningString(T&) :string
+ getKey() :char
+ setColor() :void
+ is_Number(int&) :bool
+ seekNIF(string&) :bool
-library
Library
- books :vector<Book*>
- borrows :vector<Borrow*>
- persons :vector<Person*>
+ Library()
+ ~Library()
+ getBooks() :vector<Book*> {query}
+ getAvailableBooks() :vector<Book*> {query}
+ getBorrowedBooksFromReader(Person*) :vector<Borrow*> {query}
+ getBorrows() :vector<Borrow*> {query}
+ getPersons() :vector<Person*> {query}
+ getReaders() :vector<Person*> {query}
+ getEmployees() :vector<Person*> {query}
+ getEmployeesNoSupervisors() :vector<Person*> {query}
+ getSupervisors() :vector<Person*> {query}
+ setBooks(vector<Book*>) :void
+ setBorrows(vector<Borrow*>) :void
+ setPersons(vector<Person*>) :void
+ addBook(Book*) :void
+ addBorrow(Borrow*) :void
+ addPerson(Person*) :void
+ removeBook(Book*) :bool
+ removeBorrow(Borrow*) :bool
+ removeEmployee(Person*, Person*) :bool
+ removeReader(Person*) :bool
+ removeEmployeeFromSupervisors(Employee*) :bool
+ loadBooks() :void
+ loadPersons() :void
+ loadBorrowBooks() :void
+ saveBooks() :void
+ savePersons() :void
+ saveBorrows() :void
+ sortByType() :void
+ sortByName() :void
+ sortByAge() :void
+ sortByBorrow() :void
+ sortByTitle() :void
+ sortByISBN() :void
+ SupervisorEmployeeRandom() :void
DataTypeT : class
Exception
- msg :string
- value :DataTypeT
+ Exception(string, DataTypeT)
+ getError() :string {query}
+ getMessage() :string {query}
+ getValue() :DataTypeT {query}