Formal Requirement Specification
Online Book Store
Phase-II

Submitted in partial fulfillment of the requirements of the degree of
Master of Software Engineering

Vamsi Krishna Mummaneni
CIS 895 – MSE Project
Kansas State University

Committee Members
Dr. Torben Amtoft (Major Professor).
Dr. Dan Andresen.
Dr. Mitchell L. Neilsen.
model OnlineBookstore

-- Classes

class User
attributes
UserID: string
password: string
LoginStatus: string

operations
Verifylogin (): Boolean
end

class Administrator < User
attributes
AdminID: string
password: string
Name: string
email: string
phoneNo: integer

operations
addCategory(): Boolean
deleteCategory:Boolean
addMember():Boolean
deleteMember():Boolean
addBook():Boolean
deleteBook():Boolean
addCCtype():Boolean
deleteCCtype():Boolean
end

class Customer < User
attributes
customerID:string
password:string
Name:string
address:string
email:string
phoneno:integer
CCInfo: string

operations
register(): Boolean
login(): Boolean
updateProfile(customerID: string, Name:string, address:string, email:string,
phoneno: integer, CCInfo: string):Boolean
pre: Customer.allInstances.customerID->includes (customerID)
post: Customer.allInstances.customerID = user.allInstances.customerID@pre
post: Customer.allInstances.Name =user.allInstances->select(C:Customer | C.customerID <>customerID).Name@pre->includes (Name)
post: Customer.allInstances.email = Customer.allInstances->select(C:Customer | C.customerID<>customerID).email@pre->includes(email)
post: Customer.allInstances.address = Customer.allInstances->select(C:Customer | C.customerID<>customerID).address@pre->includes(address)
post: Customer.allInstances.phoneno = Customer.allInstances->select(C:Customer | C.customerID<>customerID).phoneno@pre->includes(phoneno)
post: Customer.allInstances.CCInfo = Customer.allInstances->select(C:Customer | C.customerID<>customerID).CCInfo @pre->includes(CCInfo)
end

class Category
attributes
categoryID:integer
categoryName:string

operations
categoryName:categoryName

operation getCategoryBooks(bookID:int):Set(Book) = 
  Book.allInstances->select(b:Book| b.bookID = bookID)
end

class Book
attributes
bookID:integer
categoryID:integer
bookName:string
authorName:string
notes:string
price:float
imageurl:string
producturl:string
rating:int

operations
getBook():Boolean
end

class ShoppingCart
attributes
orderID:integer
customerID:integer
price:float

operations
addCart():Boolean
deleteCart():Boolean
updateCart():Boolean
end
class BookOrder
attributes
orderID: integer
customerID: integer
price: float
quantity: integer
operations
placeOrder(BO: BookOrder): Boolean
pre: BO.User. Verifylogin (BO.User. UserID, BO.User.password)=true
pre: BookOrder.allInstances->excludes(BO)
post: Customer.allInstances -> forAll(C: Customer | C.customerID=BO.customerID
implies BO.orderID = C.orderID)
post: BookOrder.allInstances.orderID =
BookOrder.allInstances.orderID@pre->includes(BO.orderID)
end

class Search
attributes
bookTitle: String
categoryID: integer
operations
getBookset(): Boolean
end

class AdvSearch
attributes
bookTitle: String
categoryID: integer
bookAuthor: String
bookLowCost: float
bookHighCost: float
operations
getBooksetbyAdv(): Boolean
end

class BookSet
attributes
bookID: Int
bookName: String
end

-- Assosiations

--Each book should belong to only one Category
association bookCategory between
Category[1] role subCategory
Book[1..*] role allBook
end

--Each BooksOrder should contain atleast one Book
association BooksOrderHasBook between
BooksOrder[1] role theBookOrder
Book[1..*] role theBook
end
--Each BooksOrder should belong to exactly one Customer
association CustomerHasOrder between
Customer[1] role belongstocustomer
BooksOrder[0..*] role thecustomerbook
end

--Each Shopping cart should belong to only one Customer
association CustomerrelatedtoShoppingCart between
Customer[1] role thecustomer
ShoppingCart[0..*] role thecart
end

--Each shoppingcart should have atleast one BooksOrder
association ShoppingCartHasOrder between
ShoppingCart[1] role thebookCart
BooksOrder[1..*] role theorder
end

--Each search should result some bookset
association searchhassomebookset between
Search[1] role thesearch
BookSet[0..*] role thesearchset
end

--Each Advsearch should result some bookset
association Advsearchhassomebookset between
AdvSearch[1] role theAdvsearch
BookSet[0..*] role theAdvsearchset
end

--Constraints

-- Each user should have different userID
context User
inv distinctuserID:
User.allInstances -> forAll(user1, user2 |user1 <> user2 implies user1.userID <> user2.userID)

-- Each Book should belong to exactly one Category
context Category
inv BookHasoneCategory:
Category.allInstances -> forAll(C1, C2 |C1<>C2 & & C1. getCategoryBooks ->
includes(book) implies C2. getCategoryBooks -> excludes(book))

--Each Book should have a different bookID
context Book
inv DistinctBookID:

--The OrderID for each Order must be different
context BookOrder
inv DistinctOrderID:
BookOrder.allInstances -> forAll(BO1, BO2 | BO1 <> BO2 implies BO1.orderID<> BO2.orderID)

--Each BookOrder should have some books
context BookOrder
inv BookOrderHasbooks
self.contains -> notEmpty()

-- Each BookOrder belongs to exactly one customer
context BookOrder
inv OrdertoOneCustomer
Order.allInstances -> forAll (BO1, BO2 | BO1.orderID <> BO2.orderID implies BO1.customerID <> BO2.customerID)

-- Quantity should always be a positive value
context BookOrder
inv BookOrder Positive
self.quantity > 0

--Each Shopping cart belongs to only one customer
context ShoppingCart
inv CarthasOneCustomer
ShoppingCart.allInstances -> forAll (SC1, SC2 | SC1.orderID <> SC2.orderID implies SC1.customerID <> SC2.customerID)

--In search Low price should be less than High Price
context AdvSearch
inv PriceCompare
self. bookLowCost < self. bookHighCost

-- Price should always be a positive value
context Book
inv BookPricePositive:
self.price > 0