Prj_5_StoreInventory Analysis and Design Document

Version 1.0

27 February, 2017

Developed by:

Student Name: Muresan Andreea

Group, year: 936 , III

Version History

Version	Description of Change	Author	Date
V01	Initial/Modification of ….	Student Name	27 February, 2017

Contents

Analysis and design Document

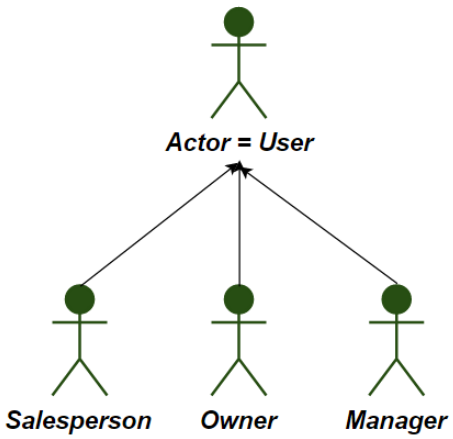
1 Functional Requirements

List the functional requirements (FR) of the system.

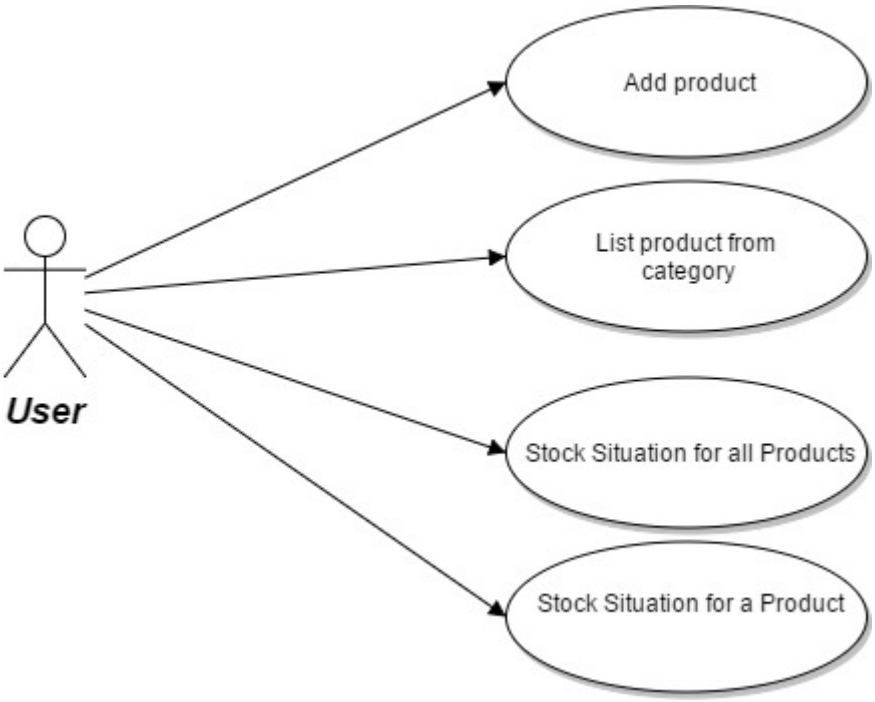
Section/ Requirement ID	Requirement Definition
FR1	The system shall add a new Product. The product has the following fields: code:int, name:string, quantity:int, supplier:string. The product information input will be specified in the console and the products will be saved in a locally stored txt file. The output will be on the console.
FR2	The system shall list the Products from a Category. The user inputs a string which is the Chosen category and the system must return a list containing all products from that category. The input is a locally stored txt file from which the system will output the products. The output will be on the console screen.
FR3	The system shall present stock situation for all products. The system must return a list with all products and their quantities and respective information. The list items will be taken from a locally stored list and the output shall be processed in the console.
FR4	The system shall present stock situation for a specified product. The user will specify a string which will be the product for which he queries the information. After that the system will output the stock situation (quantity) for that product. The list items will be taken from a locally stored list and the output shall be processed in the console.

2 Actors

A use case defines the interactions between external actors and the system under consideration to accomplish a goal. Actors must be able to make decisions, and for our product potential actors are people that interact with the system. Likely actors are: counter salesperson, manager, owner.



3 Use cases – diagram



3.1 Use case number 1 (Add product)

Actors: User
Description: Adds a product to list
Precondition: Product p is valid with valid members
Postcondition: Local List of products has one more object, in the txt file we have one more valid entry.

Use case number 2 (List product from category)

Actors: User
Description: List a product with a certain category
Precondition: Category category is valid/nonempty string
Postcondition: List outputted to console and message if list is empty

Use case number 3 (Stock situation for all products)

Actors: User
Description: Lists all products and their information/members
Precondition: System is valid
Postcondition: : List outputted to console and message if list is empty

Use case number 4 (Stock situation for a product)

Actors: User
Description: Lists situation from one product which is given by user
Precondition: String product is valid/nonempty
Postcondition: : List outputted to console and message if list is empty

User action	System response

1. Add product	Persists product in local list
2. List product from category	Display list on console
3. Stock situation for all products	Display list on console
4. Stock situation for one product	Display list on console

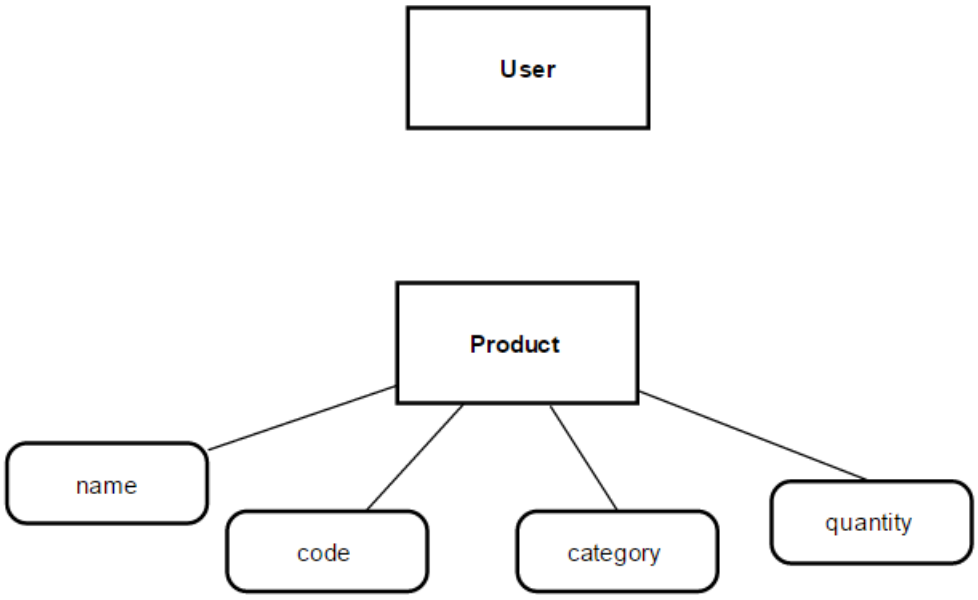
Exceptions: no exceptions

4 Analysis

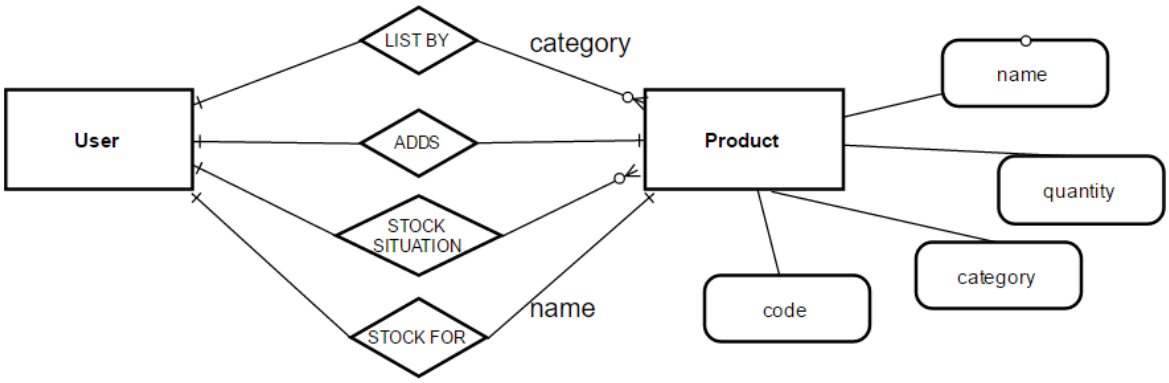
The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished. This phase defines the problem that the customer is trying to solve. The deliverable result at the end of this phase is a requirement document. Ideally, this document states in a clear and precise fashion what is to be built. This analysis represents the ``what'' phase.

4.1 Entities

The entities used to model the business need of the application are Product and User. User is not an implemented type, it is merely used to exemplify the application usability and business logic.



4.2 Relations between entities

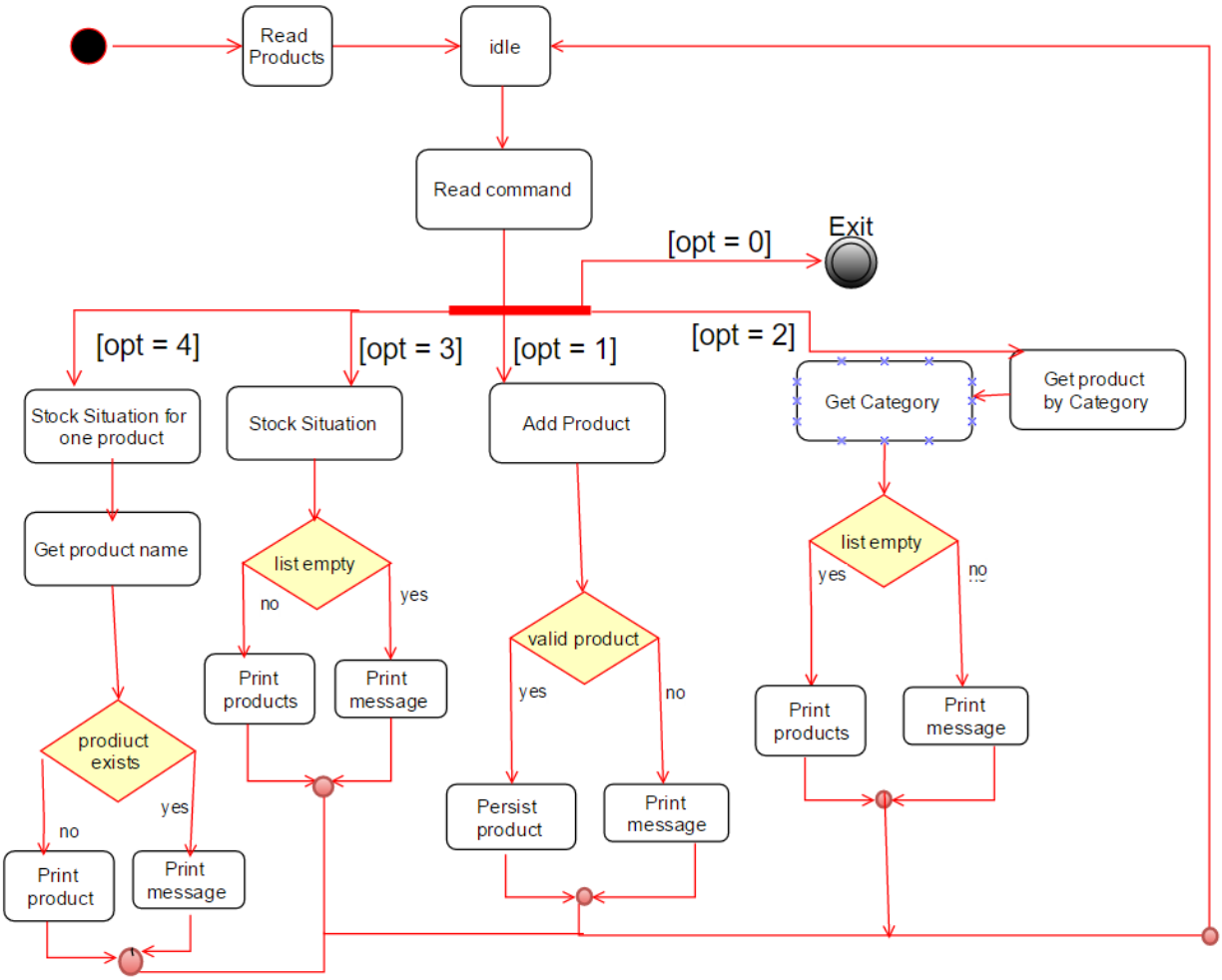


4.3 Attributes

Class	Attribute Name	Type	Description
Product	Name	String	Default value: "_" Valid state: must not be empty string, must contain at least one character. Meaning: Represents the name of the product. It is unique among other products and it uniquely defines a product.
	Code	Int	Default value: 0 Valid state: must be integer, positive number and respect integer limits according to program. Meaning: Represents a code that uniquely defines the product.
	Quantity	Int	Default value: 0 Valid state: must be integer, positive number and respect integer limits according to program. Meaning: Represents the stock of the current product meaning the quantity of product available.
	Category	String	Default value: "_" Valid state: must not be empty string, must contain at least one character. Meaning: Represents the category of a product. This may be seen as the immediate upper container, the containing set. A category may contain one or more products.

4.4 System behavior

System behavior is illustrated in the following UML Activity Diagram. Activities represent events triggered by the system and they are exemplified with the following scenario:



4.4.1

Use case 1

Normal behavior: Add product and persist it locally

Exceptions: If product invalid, prints message to output

Use case 2

Normal behavior: Outputs list of products

Exceptions: If product list empty, prints message

Use case 3

Normal behavior: Outputs list of products

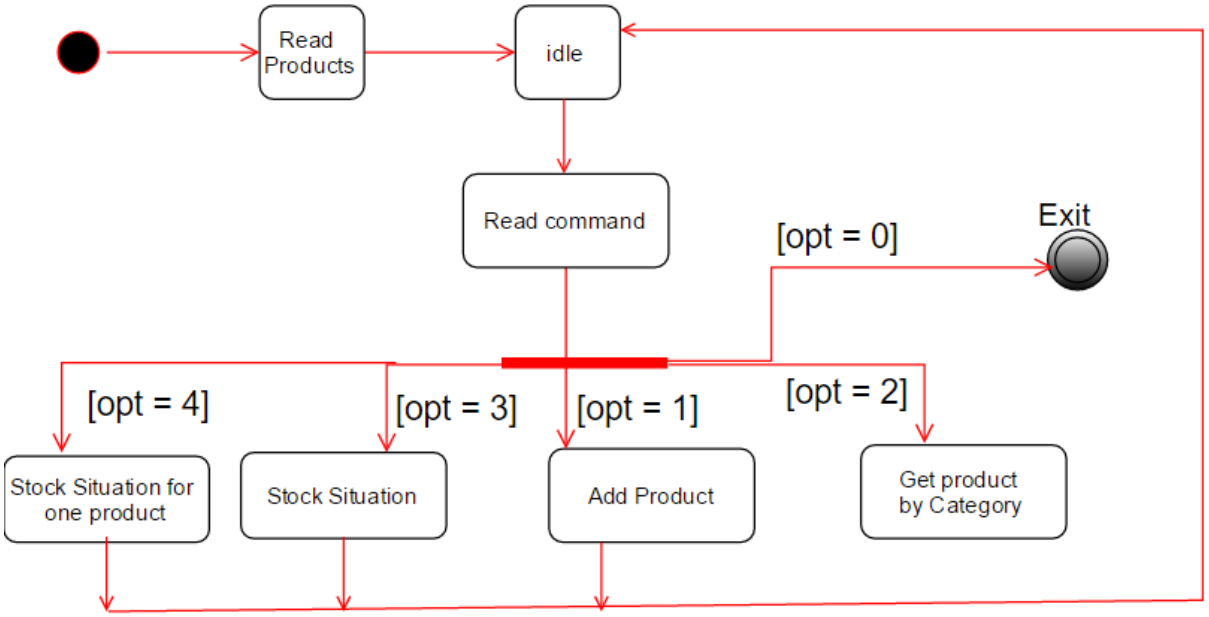
Exceptions: If product list empty, prints message

Use case 4

Normal behavior: Outputs product
Exceptions: If product list empty, prints message

4.5 System events

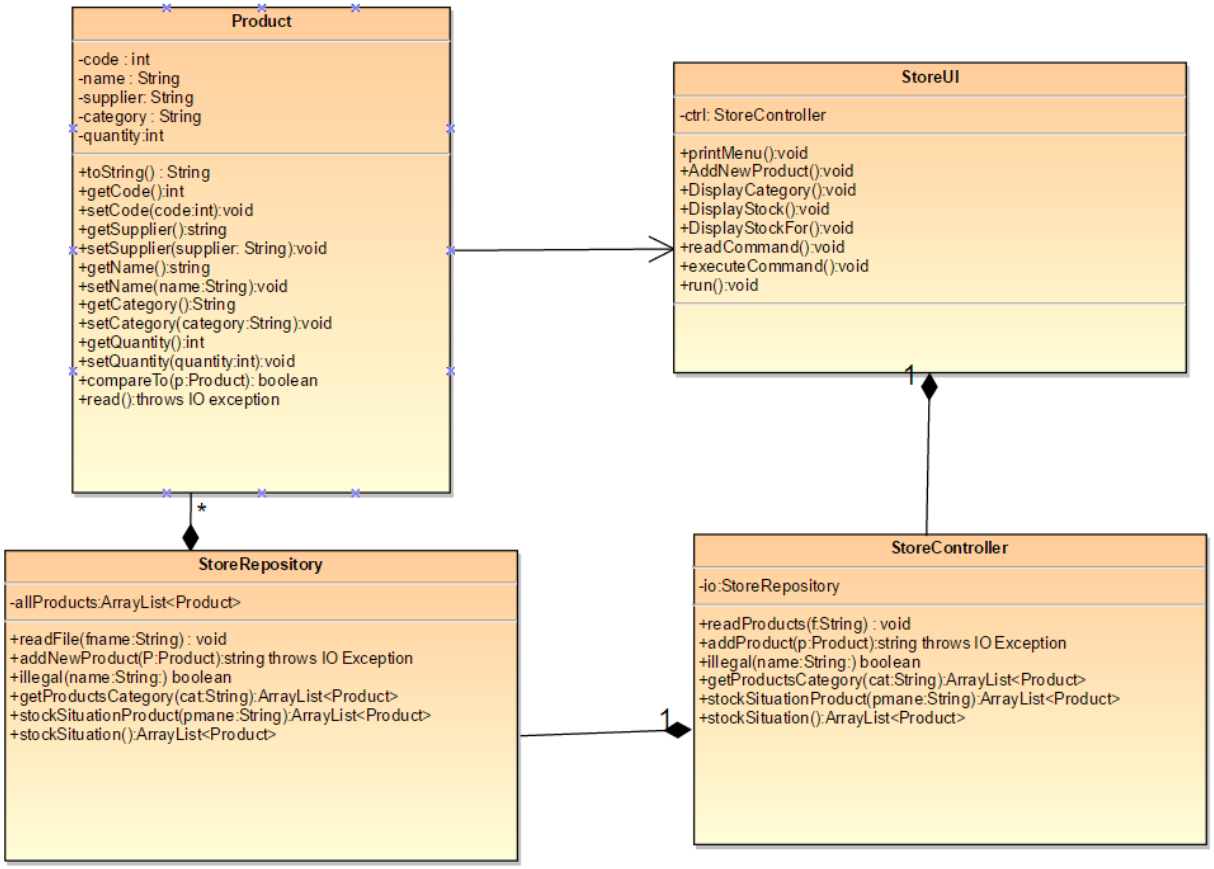
The following activity diagram reveals the main internal events of the system as well as heir triggers. The main triggers for the read commands and subsequent commands are the user and the user input. The Read Products event is triggered automatically whenever the system starts. It represents the loading of local data into memory. The system is under no stress from outside events.



5 Design

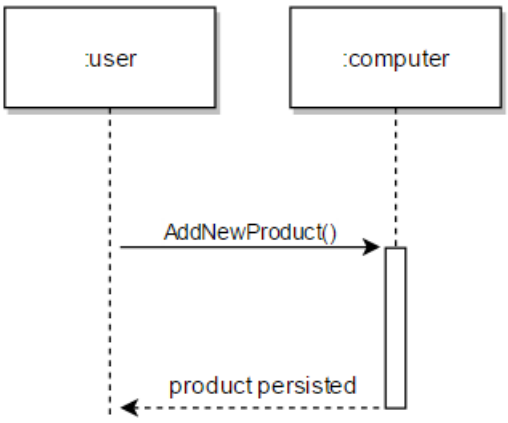
Software design documentation may be reviewed or presented to allow constraints, specifications and even requirements to be adjusted prior to computer programming. Redesign may occur after review of a programmed simulation or prototype. It is possible to design software in the process of programming, without a plan or requirement analysis, but for more complex projects this would not be considered feasible.

5.1 Class diagram

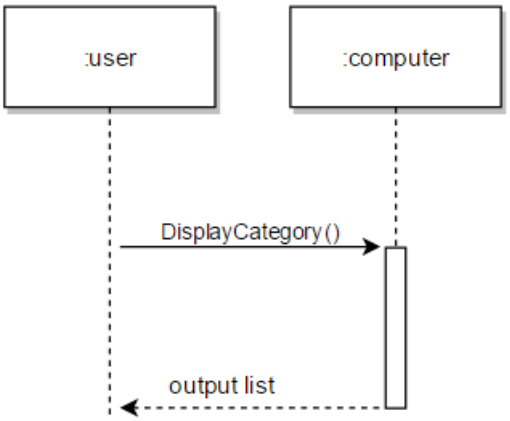


5.2 Sequence diagrams (for each use case)

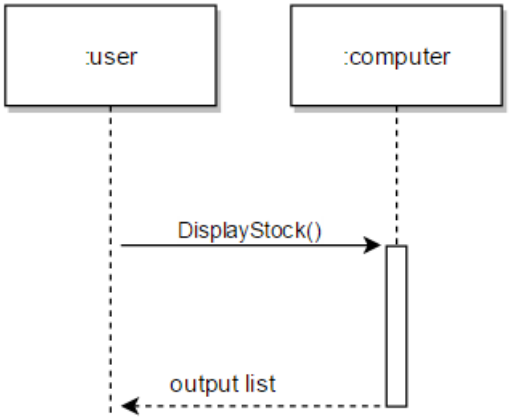
Use case 1:



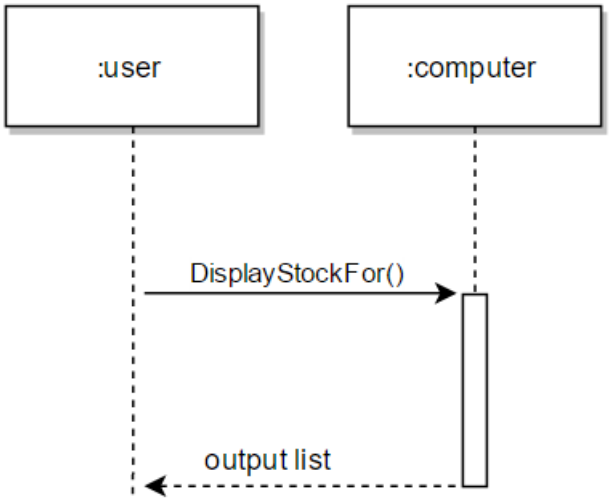
Use case 2:



Use case 3:



Use case 4:



5.3 GRASP

In the program implementation the controller pattern may be used, specifically the MODEL- VIEW-CONTROLLER pattern.

The controller pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario. A controller object is a non-user interface object responsible for receiving or handling a system event.

In our program the MVC – pattern is implemented by the following classes

- Controller: StoreController
- Model:Product
- View:StoreUI

As a consequence of using the MVC pattern the program may generally have high cohesion. High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of low coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Polymorphism has not been required so far.