**VDM ASSIGNMENT**

**Formal Methods in Software Engineering**

**(SE-313)**



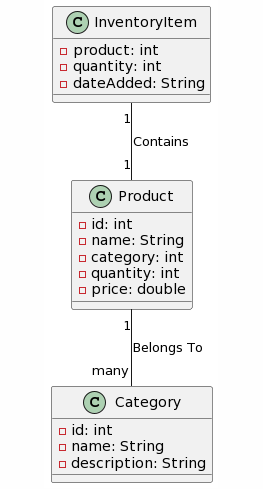
***Submitted to: Sir Mustafa Latif***

***Submitted by: Afra Rasheed (SE-21007)***

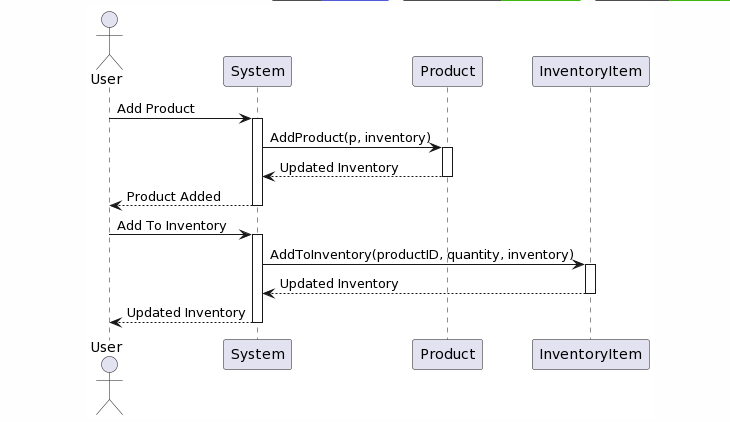
**INVENTORY MANAGEMENT SYSTEM**

The proposed inventory management system is a comprehensive and efficient solution designed to streamline and optimize the management of products within a business setting. This system is based on a formal specification using the Vienna Development Method (VDM). The core types involved in this project include "Product," representing the fundamental attributes of items such as ID, Name, Category, Quantity, and Price; "Category," encapsulating details like ID, Name, and Description; and "InventoryItem," comprising attributes such as Product, Quantity, and DateAdded. The system's functionality is driven by a set of well-defined operations, including the addition of products and categories, management of inventory quantities, retrieval of detailed product information, and the generation of comprehensive reports. Notably, the VDM specification emphasizes the importance of preconditions and postconditions for each operation, ensuring the system's reliability and consistency. This inventory management system offers a robust foundation for businesses seeking an organized and efficient approach to handling their product inventory.

***CLASS DIAGRAM:***

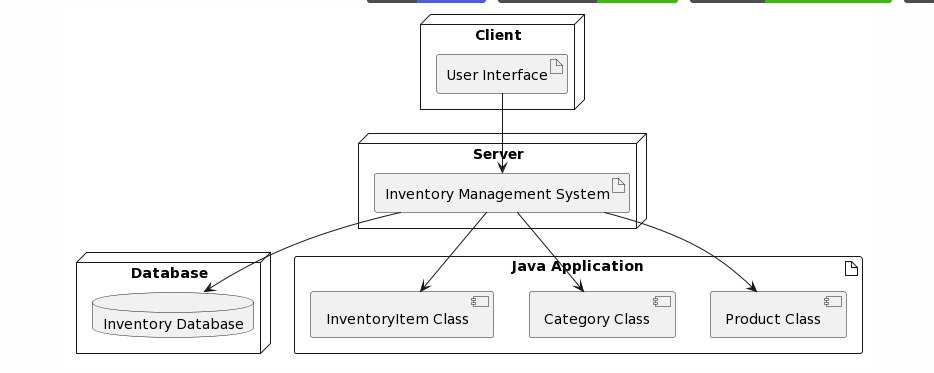
******

The provided class diagram illustrates a simplified representation of an Inventory Management System. It includes three main classes: `Product`, `Category`, and `InventoryItem`. Each class encapsulates relevant attributes denoted by private fields, such as `id`, `name`, `category`, `quantity`, and `price` for products, `id`, `name`, and `description` for categories, and `product`, `quantity`, and `dateAdded` for inventory items. The relationships between the classes are also depicted: a one-to-many relationship between `Product` and `Category` denoted by "Belongs To," indicating that a product belongs to one category, and a one-to-one relationship between `InventoryItem` and `Product` denoted by "Contains," signifying that an inventory item contains information about a specific product along with its quantity and the date it was added.

***SEQUENCE DIAGRAM:***

The provided sequence diagram depicts the interaction between an actor, denoted as "User," and a system composed of the main components: `Product` and `InventoryItem`. The sequence starts with the user initiating an "Add Product" operation. Upon this request, the system is activated, and the user sends a message to the system. The system, in turn, communicates with the `Product` component, specifically invoking the `AddProduct` operation, which results in an update to the inventory. After the interaction with the `Product` component is complete, the system deactivates and sends a confirmation message back to the user, indicating that the product has been successfully added.

The sequence then continues with the user initiating an "Add To Inventory" operation. The system is once again activated, and this time, it communicates with the `InventoryItem` component, invoking the `AddToInventory` operation. As a result, the inventory is updated, and the system sends a confirmation message back to the user, indicating that the inventory has been successfully updated. The system then deactivates, concluding the sequence.

***DEPLOYMENT DIAGRAM:***

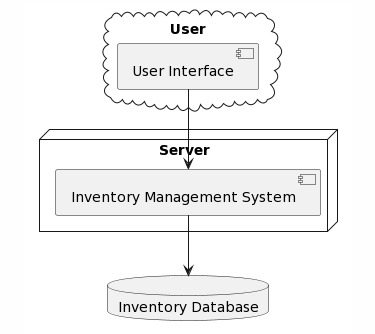
The provided deployment diagram represents the deployment structure of an Inventory Management System. The system is divided into three main components: the "Client," the "Server," and the "Database." Each of these components is represented by nodes.

The "Client" node hosts the "User Interface" artifact, indicating that the user interface is deployed on the client side. The "Server" node hosts the "Inventory Management System" artifact, suggesting that the core functionalities and business logic of the system are deployed on the server side. The "Database" node represents the "Inventory Database," signifying the storage of data related to the inventory.

The "Java Application" artifact encompasses three classes: "Product Class," "Category Class," and "InventoryItem Class." These classes likely contain the implementation of the product, category, and inventory item entities, respectively, and are grouped together as part of the Java application.

The arrows between the nodes and artifacts denote the dependencies and interactions between them. Specifically, the "User Interface" interacts with the "Inventory Management System," which, in turn, interacts with the "Inventory Database," "Product Class," "Category Class," and "InventoryItem Class."

***COMPONENT DIAGRAM:***

******

The provided component diagram illustrates the high-level components and their relationships in an Inventory Management System. The "User" is represented by a cloud, and it interacts with the "User Interface" component. This component is responsible for presenting information to the user and receiving user input. The "Server" node hosts the core logic of the system, represented by the "[Inventory Management System]" component. This component likely includes functionalities related to managing products, categories, and inventory items, as well as handling user requests. The "Inventory Database" is represented as a separate database component. This component is responsible for storing and managing data related to the inventory, such as product details, categories, and inventory item information.

The arrows between the components denote the interactions and dependencies between them. Specifically, the "User Interface" communicates with the "[Inventory Management System]" component, and the "[Inventory Management System]" component interacts with the "Inventory Database.

***VDM SPECIFICATION:***

types

ProductID = nat;

CategoryID = nat;

Quantity = nat;

Product = record

id: ProductID;

name: seq of char;

category: CategoryID;

quantity: Quantity;

price: real;

end

Category = record

id: CategoryID;

name: seq of char;

description: seq of char;

end

InventoryItem = record

product: ProductID;

quantity: Quantity;

dateAdded: seq of char; -- Consider using a proper date type in a real implementation

end

operations

AddProduct: Product \* set of Product -> set of Product

AddProduct(p, inventory) ==

inventory union {p}

AddCategory: Category \* set of Category -> set of Category

AddCategory(c, categories) ==

categories union {c}

AddToInventory: ProductID \* Quantity \* set of InventoryItem -> set of InventoryItem

AddToInventory(productID, quantity, inventory) ==

let item = inventory & {i | i.product = productID} in

if item = {}

then

inventory union {mk\_InventoryItem(productID, quantity, "current\_date")}

else

inventory munion {item |-> mk\_InventoryItem(productID, item.quantity + quantity, "current\_date")}

RemoveFromInventory: ProductID \* Quantity \* set of InventoryItem -> set of InventoryItem

RemoveFromInventory(productID, quantity, inventory) ==

let item = inventory & {i | i.product = productID} in

if item = {}

then

inventory

else

let newQuantity = item.quantity - quantity in

if newQuantity > 0

then

inventory munion {item |-> mk\_InventoryItem(productID, newQuantity, "current\_date")}

else

inventory \ {item}

***JAVA Code & Test Cases:***

import java.util.HashSet;

import java.util.Set;

class Product {

private int id;

private String name;

private int category;

private int quantity;

private double price;

public Product(int id, String name, int category, int quantity, double price) {

this.id = id;

this.name = name;

this.category = category;

this.quantity = quantity;

this.price = price;

}

// Getters and setters (omitted for brevity)

@Override

public String toString() {

return "Product{id=" + id + ", name='" + name + "', category=" + category +

", quantity=" + quantity + ", price=" + price + '}';

}

}

class Category {

private int id;

private String name;

private String description;

public Category(int id, String name, String description) {

this.id = id;

this.name = name;

this.description = description;

}

// Getters and setters (omitted for brevity)

@Override

public String toString() {

return "Category{id=" + id + ", name='" + name + "', description='" + description + "'}";

}

}

class InventoryItem {

private int product;

private int quantity;

private String dateAdded;

public InventoryItem(int product, int quantity, String dateAdded) {

this.product = product;

this.quantity = quantity;

this.dateAdded = dateAdded;

}

// Getters and setters (omitted for brevity)

@Override

public String toString() {

return "InventoryItem{product=" + product + ", quantity=" + quantity + ", dateAdded='" + dateAdded + "'}";

}

}

public class InventoryManagementSystem {

private Set<Product> inventory = new HashSet<>();

private Set<Category> categories = new HashSet<>();

private Set<InventoryItem> items = new HashSet<>();

public Set<Product> addProduct(Product product) {

inventory.add(product);

return inventory;

}

public Set<Category> addCategory(Category category) {

categories.add(category);

return categories;

}

public Set<InventoryItem> addToInventory(int productID, int quantity) {

for (InventoryItem item : items) {

if (item.getProduct() == productID) {

item.setQuantity(item.getQuantity() + quantity);

return items;

}

}

InventoryItem newItem = new InventoryItem(productID, quantity, "current\_date");

items.add(newItem);

return items;

}

public Set<InventoryItem> removeFromInventory(int productID, int quantity) {

for (InventoryItem item : items) {

if (item.getProduct() == productID) {

int newQuantity = item.getQuantity() - quantity;

if (newQuantity > 0) {

item.setQuantity(newQuantity);

} else {

items.remove(item);

}

return items;

}

}

return items;

}

public String viewInventory() {

StringBuilder inventoryDetails = new StringBuilder("Inventory:\n");

for (Product product : inventory) {

inventoryDetails.append(product).append("\n");

}

return inventoryDetails.toString();

}

public Product viewProductDetails(int productID) {

for (Product product : inventory) {

if (product.getId() == productID) {

return product;

}

}

return null;

}

public Set<Product> updateProductDetails(int productID, double newPrice, int newQuantity) {

for (Product product : inventory) {

if (product.getId() == productID) {

product.setPrice(newPrice);

product.setQuantity(newQuantity);

return inventory;

}

}

return inventory;

}

public String generateReport() {

StringBuilder report = new StringBuilder("Inventory Report:\n");

for (Product product : inventory) {

report.append(product).append("\n");

}

return report.toString();

}

public Set<Category> viewCategory() {

return categories;

}

public Set<Category> removeCategory(int categoryID) {

categories.removeIf(category -> category.getId() == categoryID);

return categories;

}

public static void main(String[] args) {

InventoryManagementSystem inventorySystem = new InventoryManagementSystem();

// Test case 1: Add Product to Inventory

Product product1 = new Product(1, "Laptop", 1, 10, 1200.0);

inventorySystem.addProduct(product1);

// Test case 2: Add Category

Category category1 = new Category(1, "Electronics", "Electronic devices");

inventorySystem.addCategory(category1);

// Test case 3: Add to Inventory

inventorySystem.addToInventory(1, 5);

// Test case 4: Remove from Inventory

inventorySystem.removeFromInventory(1, 3);

// Test case 5: View Inventory

System.out.println(inventorySystem.viewInventory());

// Test case 6: View Product Details

System.out.println("Product Details: " + inventorySystem.viewProductDetails(1));

// Test case 7: Update Product Details

inventorySystem.updateProductDetails(1, 1300.0, 15);

// Test case 8: Generate Report

System.out.println(inventorySystem.generateReport());

// Test case 9: View Category

System.out.println("Categories: " + inventorySystem.viewCategory());

// Test case 10: Remove Category

inventorySystem.removeCategory(1);

// Display results

System.out.println("Updated Inventory: " + inventorySystem.inventory);

System.out.println("Updated Categories: " + inventorySystem.categories);

}

}