|  |  |  |  |
| --- | --- | --- | --- |
| **学生学号** |  | **实验课成绩** |  |



**学 生 实 验 报 告 书**

|  |  |
| --- | --- |
| **实验课程名称** | Java语言程序设计D |
| **开 课 学 院** | 计算机科学与技术学院 |
| **指导教师姓名** |  |
| **学 生 姓 名** |  |
| **学生专业班级** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2021 | —— | 2022 | 学年 | 第 | 一 | 学期 |

课程名称：Java语言程序设计D

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验项目名称 | 实验 2：基本语句及容器的使用 | | | 实验成绩 |  |
| 实验者 |  | 专业班级 |  | 实验日期 | 2021年10月26日 |
| 1. 实验分析与设计（可加页）    1. 实验内容描述（问题描述）   根据上面的描述，完成 products 工程中 StockManager 类的设计，具体要求如下：  ⚫ 实现 printProductDetails 方法，该方法可以遍历产品的容器，并通过调用每个 Product 对象的 toString 方法来打印出所有产品的详细信息。  ⚫ 实现 findProduct 方法，这个方法应该在容器中搜索与其参数 ID 相匹配的 ID 的产品。 如果能找到匹配的产品，就返回这个产品；如果找不到匹配的产品，就返回 null。  ⚫ 实现 numberInStock 方法，这个方法在容器中找到 ID 匹配的产品，然后返回该产品当 前的数量。如果找不到 ID 匹配的产品，则返回 0。  ⚫ 实现 delivery 方法，它根据给定的 ID 找到产品，然后调用产品的 increaseQuantity 方法， 为该产品增加指定的库存数量。  ⚫ 实现 printLowStockProducts 方法，该方法能够打印出所有库存水平低于给定值（作为参 数传递给方法）的产品的详情。  ⚫ 修改 addProduct 方法，使得新产品的 ID 如果与已有产品列表的 ID 相同就无法加入。  ⚫ 实现 findProductByName 方法，这个方法能够根据产品的名称（而不是 ID）在容器中 搜索相匹配的产品。如果能找到匹配的产品，就返回这个产品；如果找不到匹配的产品， 就返回 null。  完成上述设计任务后，可以通过 StockDemo 类中的代码测试 StockManager 类是否工作正常   * 1. 实验基本原理与设计（包括实验方案设计，实验手段的确定，实验步骤等，用硬件逻辑或者算法描述）   Product:  /\*\*  \* Model some details of a product sold by a company.  \*  \* @author David J. Barnes and Michael Kölling.  \* @version 2016.02.29  \*/  public class Product  {  // An identifying number for this product.  private int id;  // The name of this product.  private String name;  // The quantity of this product in stock.  private int quantity;  /\*\*  \* Constructor for objects of class Product.  \* The initial stock quantity is zero.  \* @param id The product's identifying number.  \* @param name The product's name.  \*/  public Product(int id, String name)  {  this.id = id;  this.name = name;  quantity = 0;  }  /\*\*  \* @return The product's id.  \*/  public int getID()  {  return id;  }  /\*\*  \* @return The product's name.  \*/  public String getName()  {  return name;  }  /\*\*  \* @return The quantity in stock.  \*/  public int getQuantity()  {  return quantity;  }  /\*\*  \* @return The id, name and quantity in stock.  \*/  public String toString()  {  return id + ": " +  name +  " stock level: " + quantity;  }  /\*\*  \* Restock with the given amount of this product.  \* The current quantity is incremented by the given amount.  \* @param amount The number of new items added to the stock.  \* This must be greater than zero.  \*/  public void increaseQuantity(int amount)  {  if(amount > 0) {  quantity += amount;  }  else {  System.out.println("Attempt to restock " +  name +  " with a non-positive amount: " +  amount);  }  }  /\*\*  \* Sell one of these products.  \* An error is reported if there appears to be no stock.  \*/  public void sellOne()  {  if(quantity > 0) {  quantity--;  }  else {  System.out.println(  "Attempt to sell an out of stock item: " + name);  }  }  }  StockManager:  import java.util.ArrayList;  /\*\*  \* Manage the stock in a business.  \* The stock is described by zero or more Products.  \*  \* @author (your name)  \* @version (a version number or a date)  \*/  public class StockManager  {  // A list of the products.  private ArrayList<Product> stock;  /\*\*  \* Initialise the stock manager.  \*/  public StockManager()  {  stock = new ArrayList<>();  }  /\*\*  \* Add a product to the list.  \* @param item The item to be added.  \*/  public void addProduct(Product item)  {  boolean isUnique = true;  for(Product it : stock){  if(it.getID() == item.getID()){  isUnique = false;  break;  }  }  if(isUnique){  stock.add(item);  }  }    /\*\*  \* Receive a delivery of a particular product.  \* Increase the quantity of the product by the given amount.  \* @param id The ID of the product.  \* @param amount The amount to increase the quantity by.  \*/  public void delivery(int id, int amount)  {  if(amount <= 0) return;  Product product = null;  for(Product it : stock){  if(it.getID() == id){  product = it;  break;  }  }  if(product == null) return;  product.increaseQuantity(amount);  }    /\*\*  \* Try to find a product in the stock with the given id.  \* @return The identified product, or null if there is none  \* with a matching ID.  \*/  public Product findProduct(int id)  {  for(Product it : stock){  if(id == it.getID()){  return it;  }  }  return null;  }    /\*\*  \* Locate a product with the given ID, and return how  \* many of this item are in stock. If the ID does not  \* match any product, return zero.  \* @param id The ID of the product.  \* @return The quantity of the given product in stock.  \*/  public int numberInStock(int id)  {  for(Product it : stock){  if(id == it.getID()){  return it.getQuantity();  }  }  return 0;  }  /\*\*  \* Print details of all the products.  \*/  public void printProductDetails()  {  stock.forEach(it -> {System.out.println(it.toString());});  }  public void printLowStockProducts(final int lowLimit){  stock.forEach(it -> {  if(it.getQuantity() < lowLimit){  System.out.println(it.toString());  }  });  }  public Product findProductByName(final String name){  for(Product it : stock){  if(it.getName().equals(name)){  return it;  }  }  return null;  }  }  StockDemo:  /\*\*  \* Demonstrate the StockManager and Product classes.  \* The demonstration becomes properly functional as  \* the StockManager class is completed.  \*  \* @author David J. Barnes and Michael Kölling.  \* @version 2016.02.29  \*/  public class StockDemo  {  // The stock manager.  private StockManager manager;  /\*\*  \* Create a StockManager and populate it with a few  \* sample products.  \*/  public StockDemo()  {  manager = new StockManager();  manager.addProduct(new Product(132, "Clock Radio"));  manager.addProduct(new Product(37, "Mobile Phone"));  manager.addProduct(new Product(23, "Microwave Oven"));  }    /\*\*  \* Provide a very simple demonstration of how a StockManager  \* might be used. Details of one product are shown, the  \* product is restocked, and then the details are shown again.  \*/  public void demo()  {  // Show details of all of the products.  manager.printProductDetails();  // Take delivery of 5 items of one of the products.  manager.delivery(132, 5);  manager.printProductDetails();  }    /\*\*  \* Show details of the given product. If found,  \* its name and stock quantity will be shown.  \* @param id The ID of the product to look for.  \*/  public void showDetails(int id)  {  Product product = getProduct(id);  if(product != null) {  System.out.println(product.toString());  }  }    /\*\*  \* Sell one of the given item.  \* Show the before and after status of the product.  \* @param id The ID of the product being sold.  \*/  public void sellProduct(int id)  {  Product product = getProduct(id);    if(product != null) {  showDetails(id);  product.sellOne();  showDetails(id);  }  }    /\*\*  \* Get the product with the given id from the manager.  \* An error message is printed if there is no match.  \* @param id The ID of the product.  \* @return The Product, or null if no matching one is found.  \*/  public Product getProduct(int id)  {  Product product = manager.findProduct(id);  if(product == null) {  System.out.println("Product with ID: " + id +  " is not recognised.");  }  return product;  }  /\*\*  \* @return The stock manager.  \*/  public StockManager getManager()  {  return manager;  }  }  Test:  public class Test {  public static void main(String[] args) {  StockDemo stockDemo = new StockDemo();  stockDemo.demo();  stockDemo.sellProduct(5);  stockDemo.sellProduct(37);  stockDemo.showDetails(5);  stockDemo.showDetails(37);  stockDemo.getManager().delivery(132, 100);  stockDemo.getManager().findProductByName("Mobile Phone");  stockDemo.getManager().printLowStockProducts(100);  }  }   * 1. 主要仪器设备及耗材   电脑 | | | | | |

|  |
| --- |
| 1. 实验调试与结果分析（可加页）    1. 调试过程（包括调试方法描述、实验数据记录、实验现象记录，实验过程发现的问题等）   进行正常的程序编写，解决了vsc的bug后过程一切正常。     * 1. 实验结果及分析（包括结果描述、实验现象分析、影响因素讨论、综合分析和结论等）        * 1. 实验小结、建议及体会   Java的操作相较于Kotlin还是太笨重了，Kotlin的语法糖非常好用。 |