|  |  |
| --- | --- |
| **Student Name:** | **LIU SHENGWEI** |
| **Student ID:** | **S1037766** |
| **Module Name:** | COMP1103 Software Development |
|  |  |
| **Lecturer** | **Mr. Loh** |
| **Submission Date:** | 15th, December, 2025 |
| **Assignment Report**  **Word Count (if applicable)）** |  |
| **Declaration**    **I declare that this assignment is my own work, unless otherwise referenced, as**  **defined by the LSBF policy on plagiarism. I have read the LSBF Student handbook.**    **Submitted by: ..........LIU SHENGWEI........... Date .....**15th, December, 2024**......** | |

# COMP1103 Software Development

# Assignment Cover Page & Declaration

Table of Contents

1) Analysis

2) Design

3) Development

3.1 Classes and Objects

3.1.1 Branch Class

3.1.2 Employee class

3.1.3 Warehouse Class

3.1.4 Supplier Class

3.1.5 Product Class

3.2 Records of SoftDev

3.2.1 Branch

3.2.2 Employee

3.2.3 Warehouse

3.2.4 Supplier

3.2.5 Product

4) Test Cases

4.1 Branch test case

4.2 Employee test case

4.3 Warehouse test case

4.4 Supplier test case

4.5 Product test case

5) Program Listings (with comments and explanations)

6) Critical Evaluation

6.1 Strengths

6.2 Weaknesses

6.3 Future Enhancement

# Analysis

Me: Hello, I'm the manager in charge of this project from SoftDev Pte Ltd. In order to complete the project better, I need more detailed information. Could you please introduce the current situation of the company and the problems that need to be solved?

Mr. John Doe: Our company mainly engages in the retail of electronic products and has branches all over the world with a large number of employees. However, currently, there is a lack of an internal management system, which makes it impossible to globally calculate the sales volume and costs, thus hindering the continuous development of our company. I hope you can help us.

Me: Okay. Based on your description, our system will have the functions of managing branches and managing employees.

Me: Regarding the sales part, we will manage electronic products. You should also have warehouses, right? By the way, do you also have suppliers? This also needs to be counted.

Mr. John Doe: Thank you for your reminder. I almost forgot that we have many suppliers. I also hope to grade them so that we can screen out high-quality suppliers.

Me: Alright, I've got it. As a commercial project, let's talk about the most important thing first. What's your budget?

Mr. John Doe: My initial budget is 50,000 Singapore dollars. I hope to get a workable system within one month.

Me: Then it should still be in time. Let me help you sort out the business logic. See if this is right. Employees are under branches, and warehouses are also associated with branches. Electronic products should be associated with warehouses, and theoretically, suppliers should also be associated with warehouses. In this way, with branches as the main body, all the logic can be connected. What do you think?

Mr. John Doe: Suppliers may correspond to multiple branches. It depends on the inventory specifically. If the inventory of a branch is insufficient, supplies will be prioritized to that branch.

Me: Okay. Then we can make one branch correspond to multiple suppliers. Are there any other problems?

Mr. John Doe: No problem. I wish you all the best and hope we have a pleasant cooperation.

Me：Wish us a pleasant cooperation.

# Design

|  |
| --- |
| Employee |
| * String EName * String EId * String EContact * String BId |
| + getEName(): String  + setEName(name: String)  + getEId(): String  + setEId(id: String)  + getEContact(): String  + setEContact(contact: String)  + getBId():String  + setBId(val: String) |

|  |
| --- |
| Warehouse |
| * String WId * String WName * String WAddress * String BId |
| + getWId (): String  + setWId(id: String)  + getWName(): String  + setWName(name: String)  + getWAddress(): String  + setWAddress(address: String)  + getBId():String  + setBId(val: String) |

|  |
| --- |
| Branch |
| * String BName * String BAddress * String BContact * String BId |
| + getBName(): String  + setBName(name: String)  + getBAddress(): String  + setBAddress(address: String)  + getBContact(): String  + setBContact(contact: String)  + getBId():String  + setBId(val: String) |

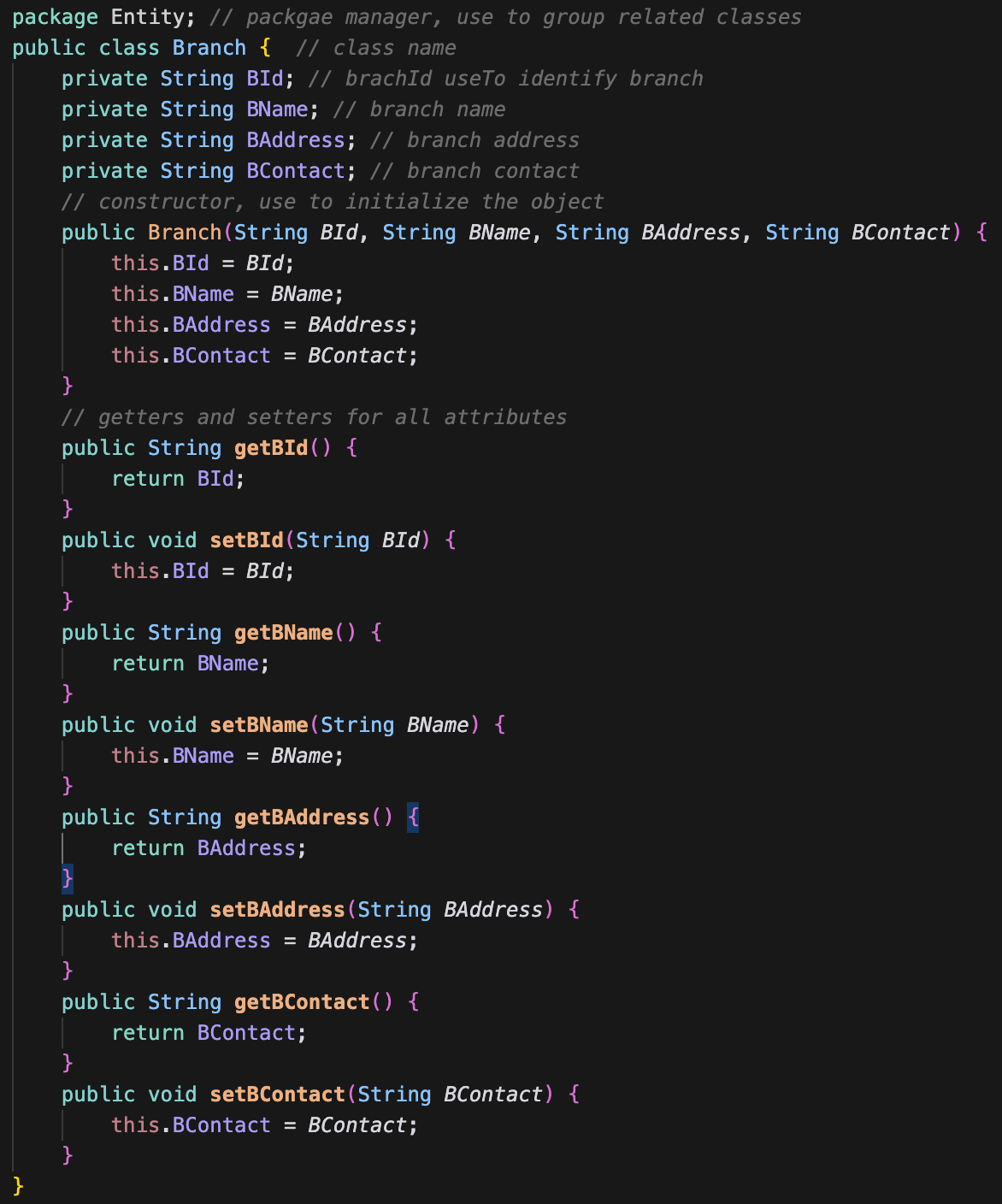
|  |
| --- |
| Product |
| * String PId * String PName * double PPrice * String WId |
| + getPId():String  + setPId(val: String)  + getPName(): String  + setPName(name: String)  + getPPrice(): double  + setPPrice (price: double)  + getWId (): String  + setWId(id: String) |

|  |
| --- |
| Supplier |
| * String SId * String SName * String SContact * String[] BIds |
| + getSId (): String  + setSId(id: String)  + getSName(): String  + setSName(name: String)  + getSContact(): String  + setSContact (contact: String)  + getBIds():String[]  + setBIds(ids: String[]) |

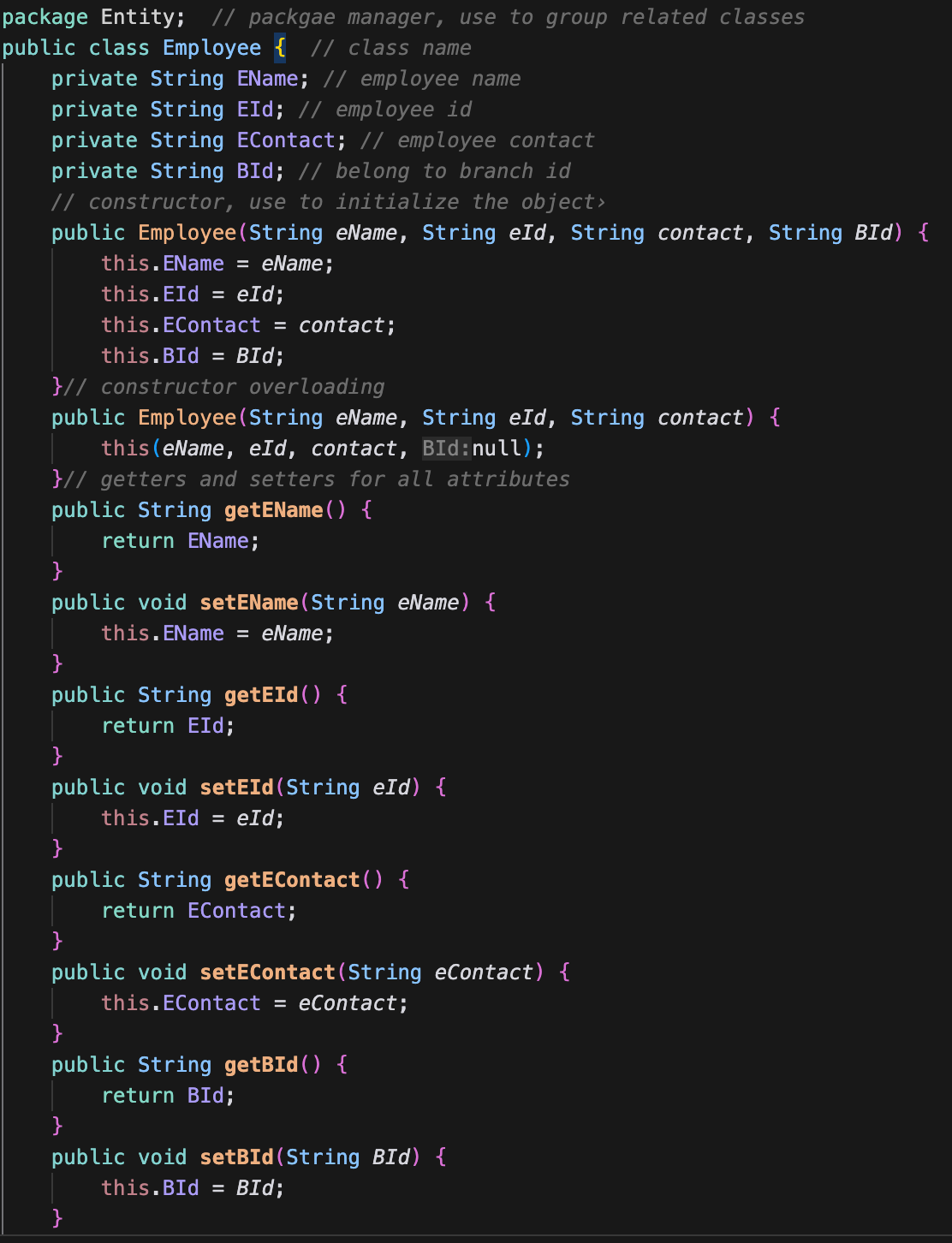
# Development

## 3.1 Classes and objects

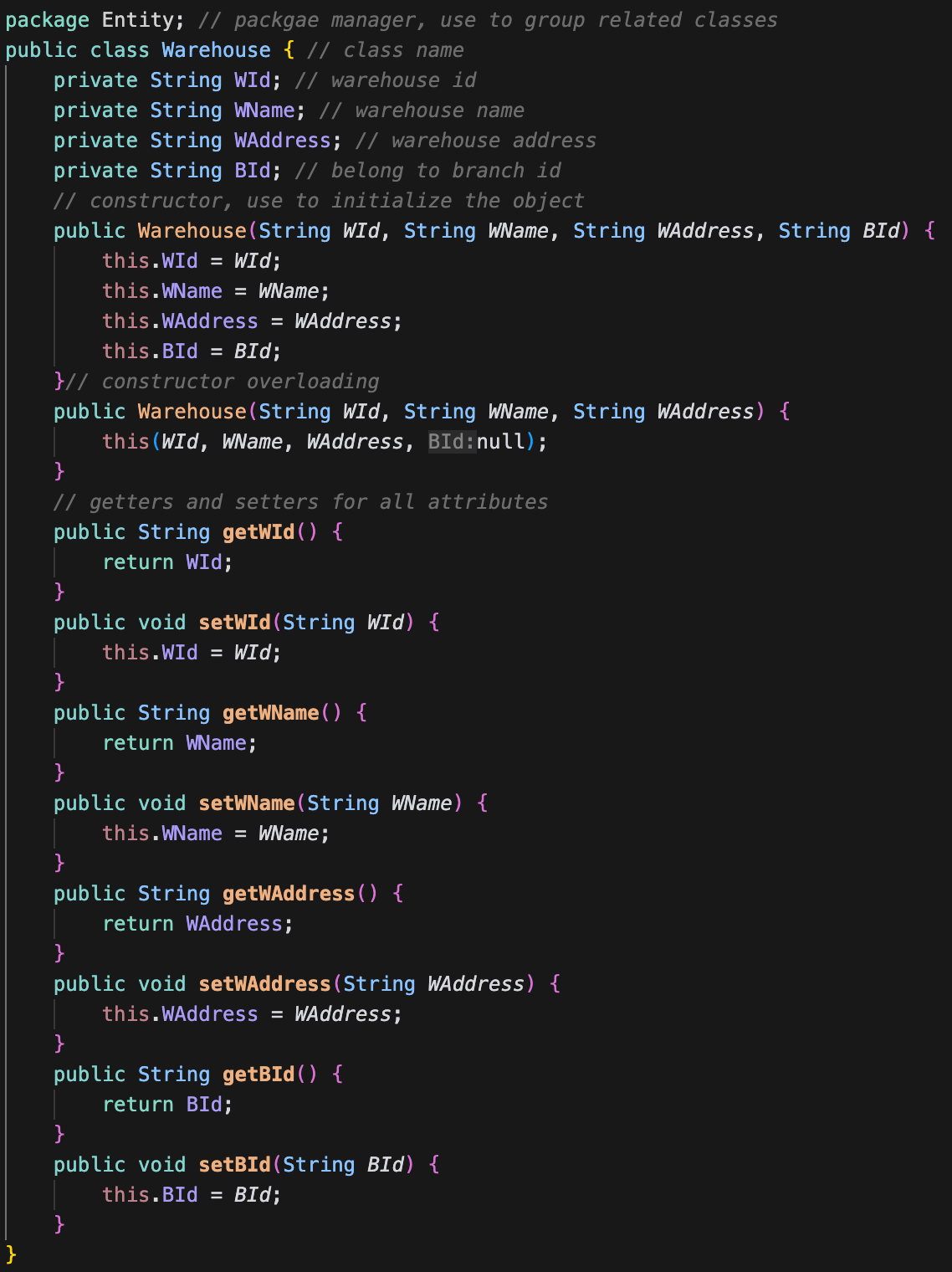
### 3.1.1 Branche Class



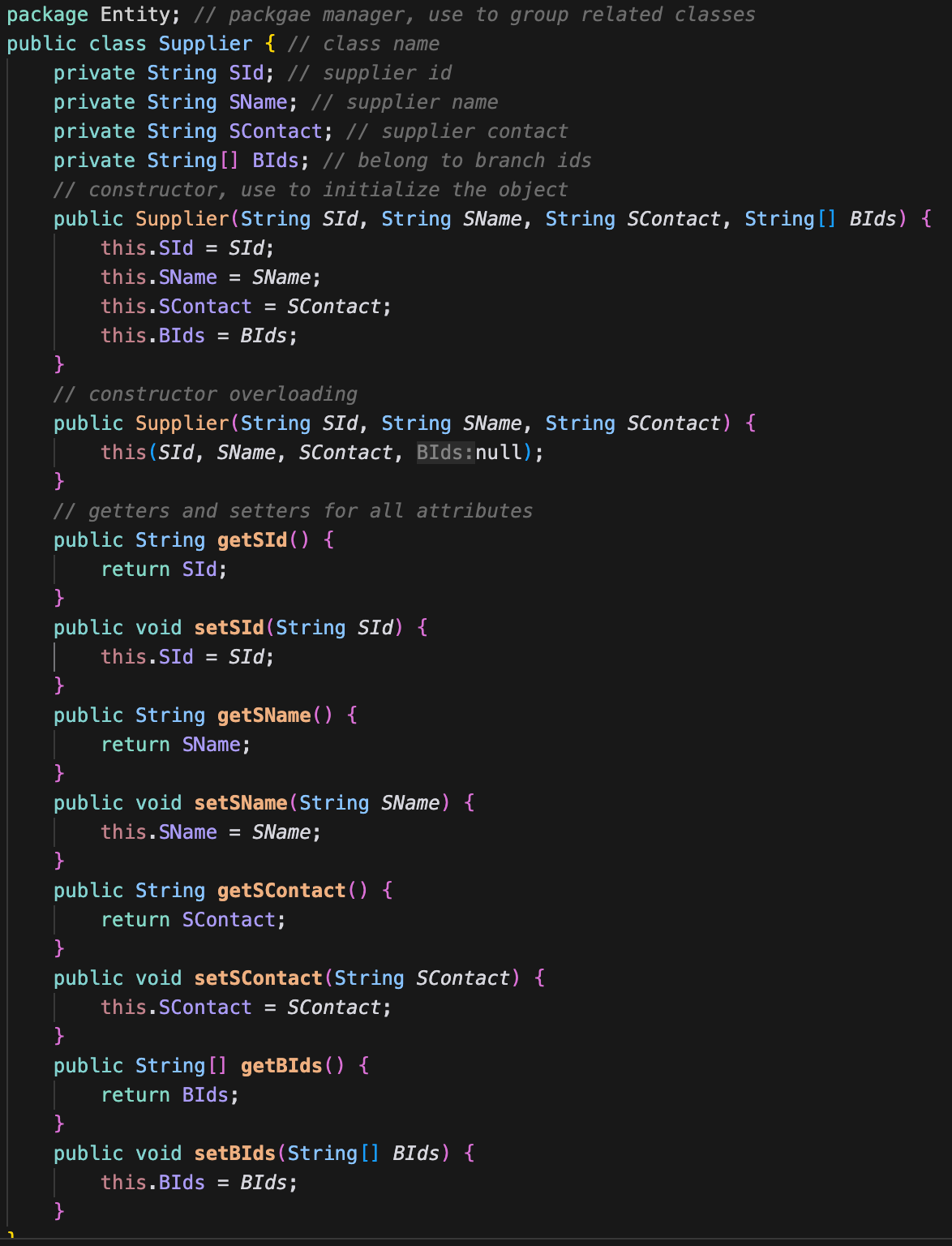
### 3.1.2 Employee Class



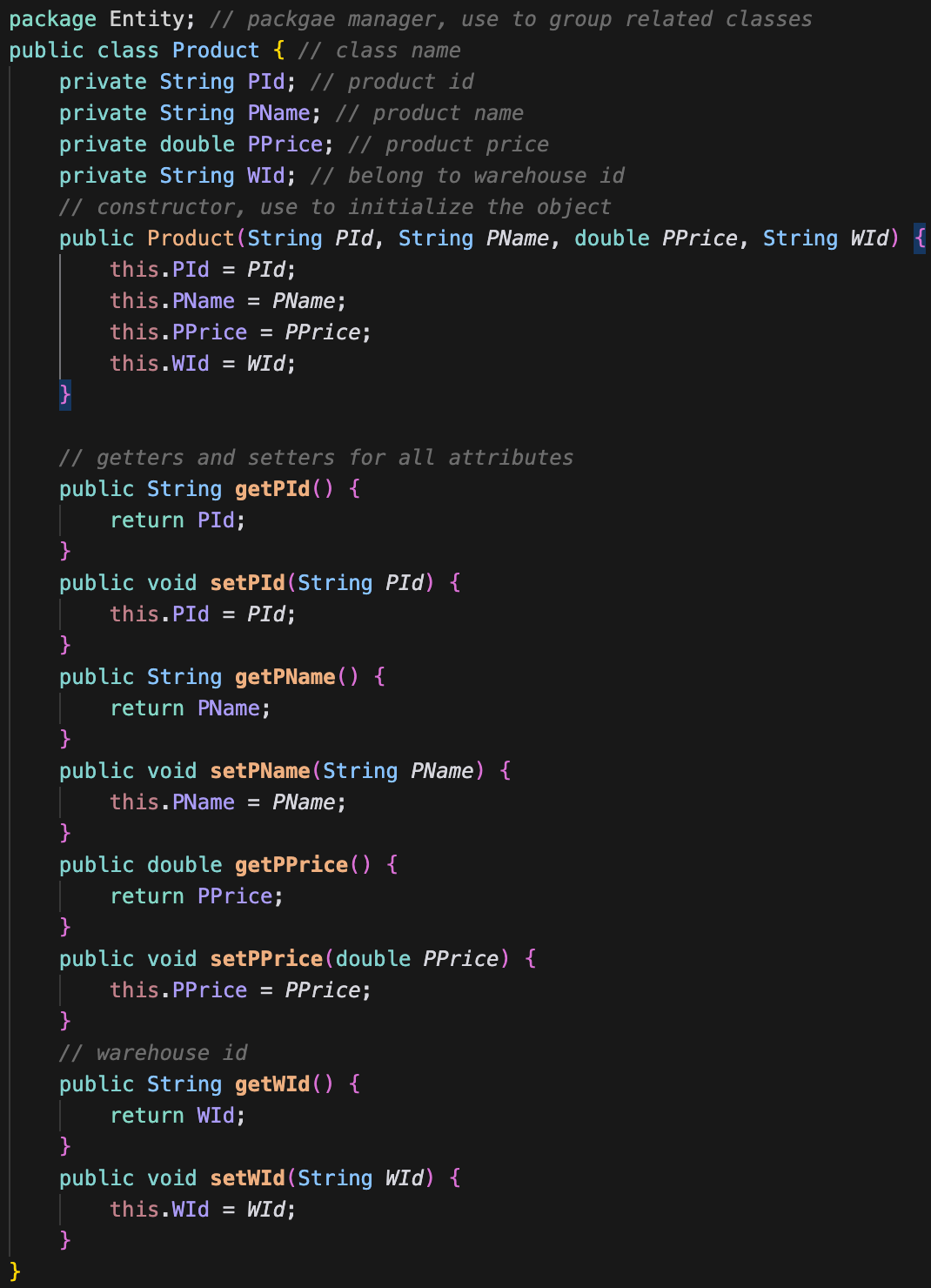
### 3.1.3 Warehouse Class



### 3.1.4 Supplier Class

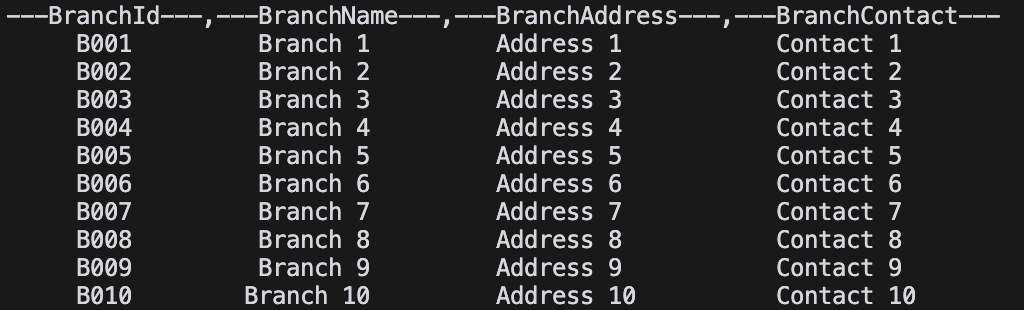


### Product Class

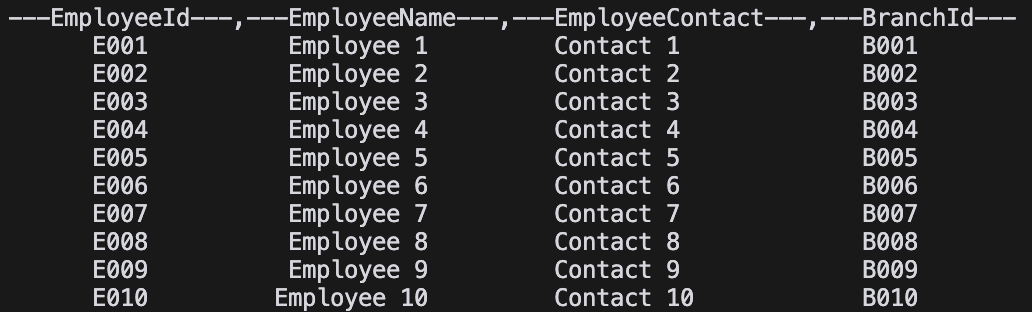


## 3.2 Record of SoftDev

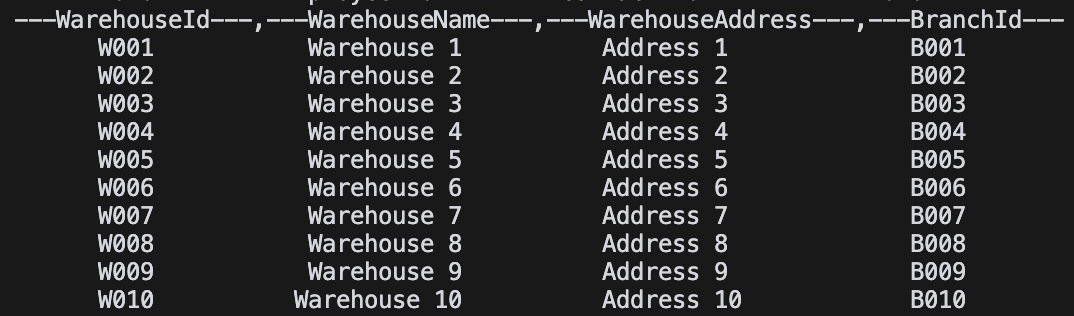
### 3.2.1 Branch Records



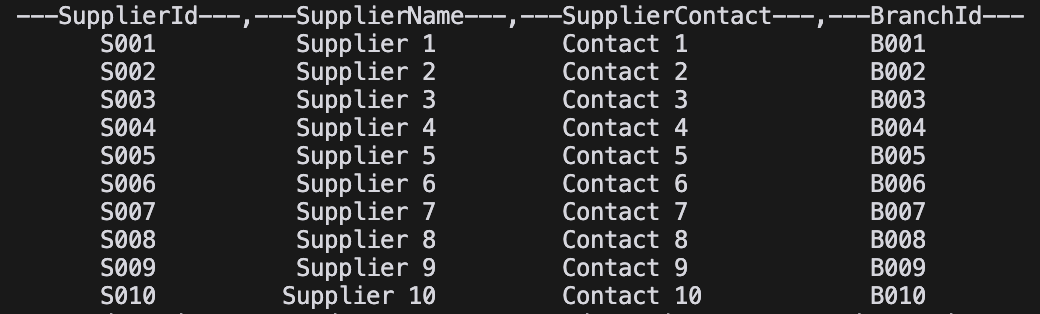
### 3.2.2 Employee Records



### 3.2.3 Warehouse Records



### 3.2.4 Supplier Records

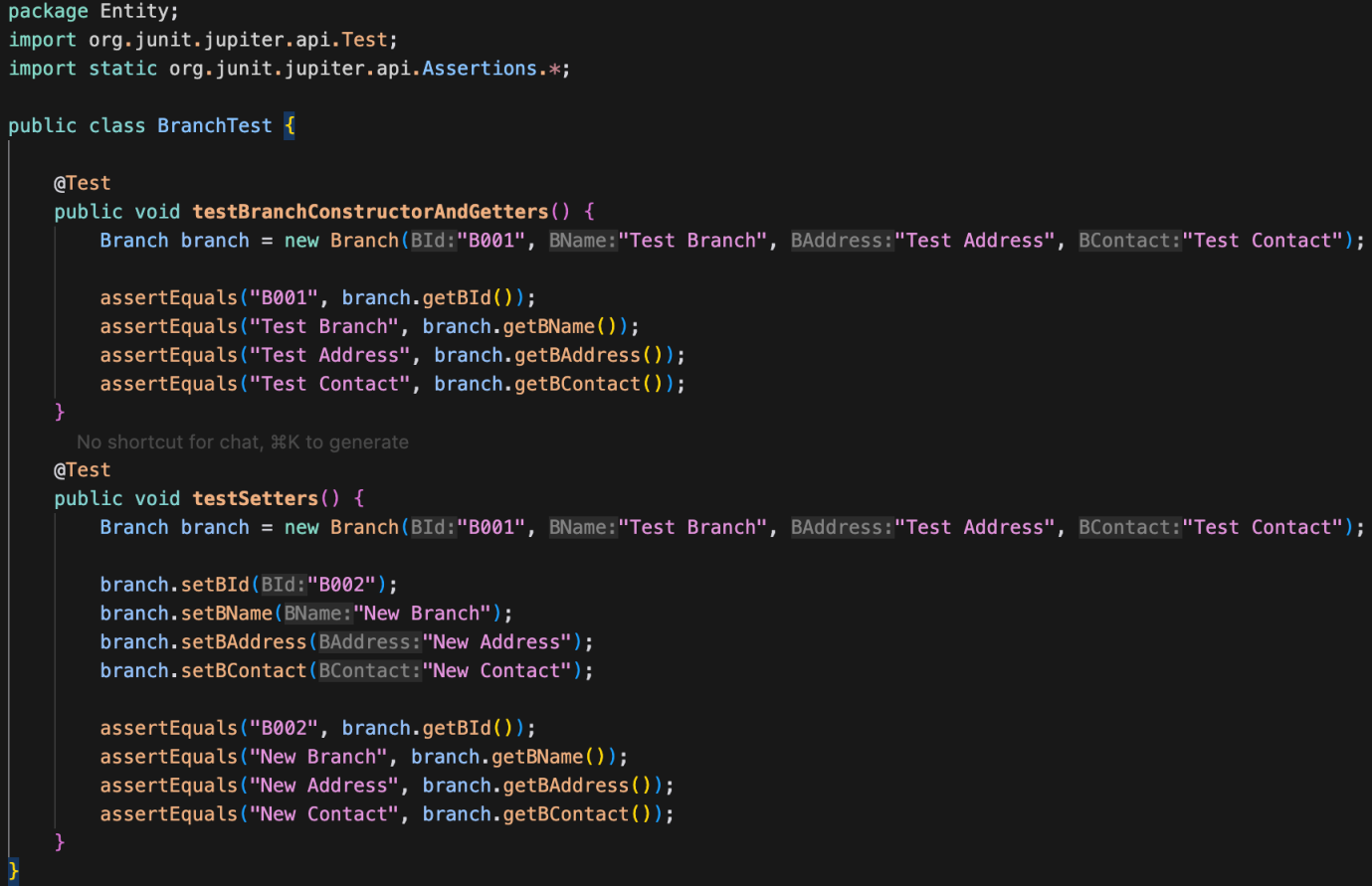


### 3.2.5 Product Records

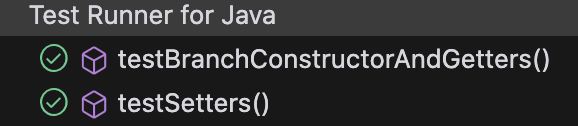


# Test Case

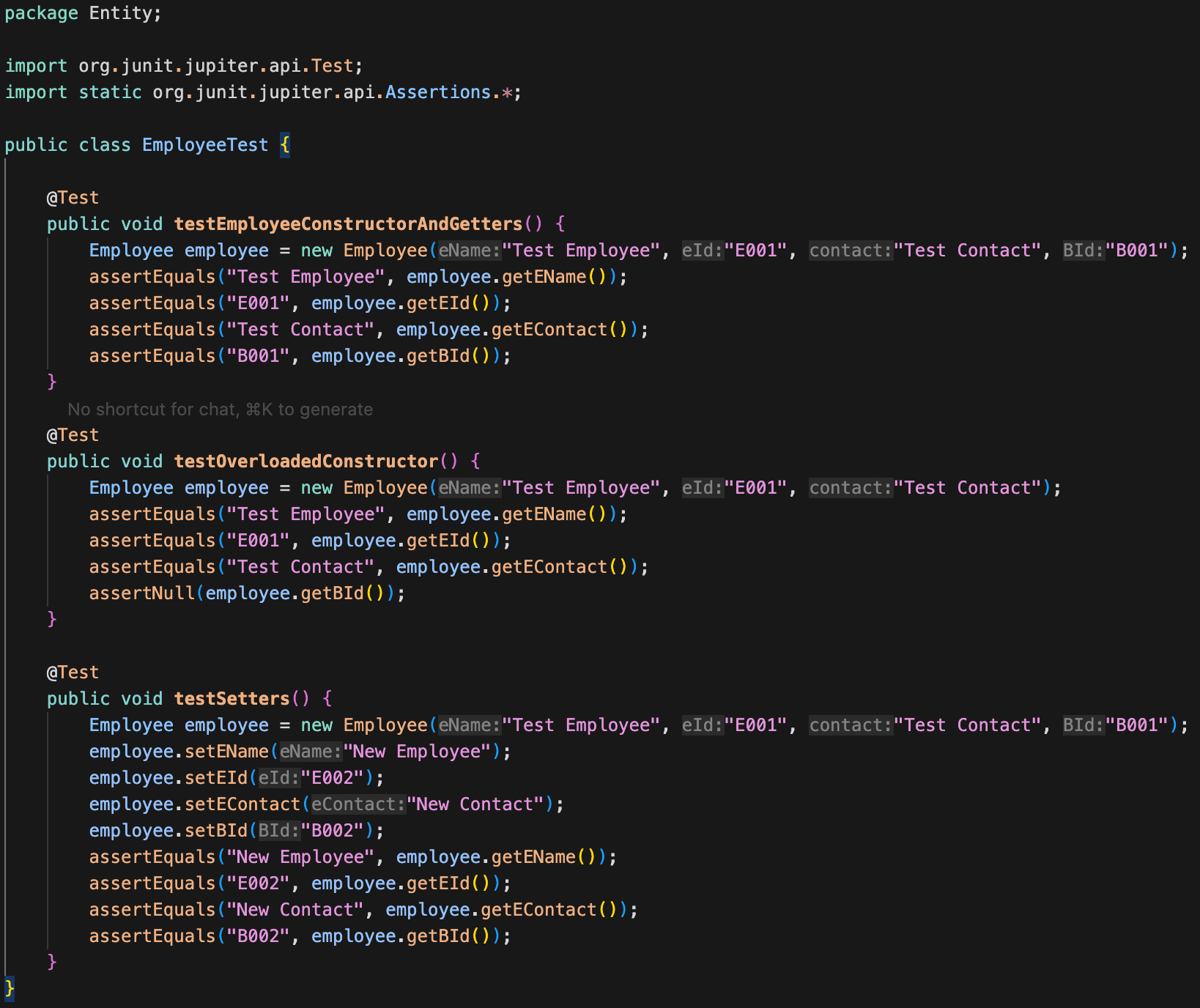
## 4.1 Branch test case



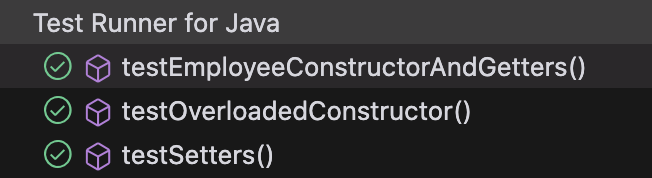
Ouput:



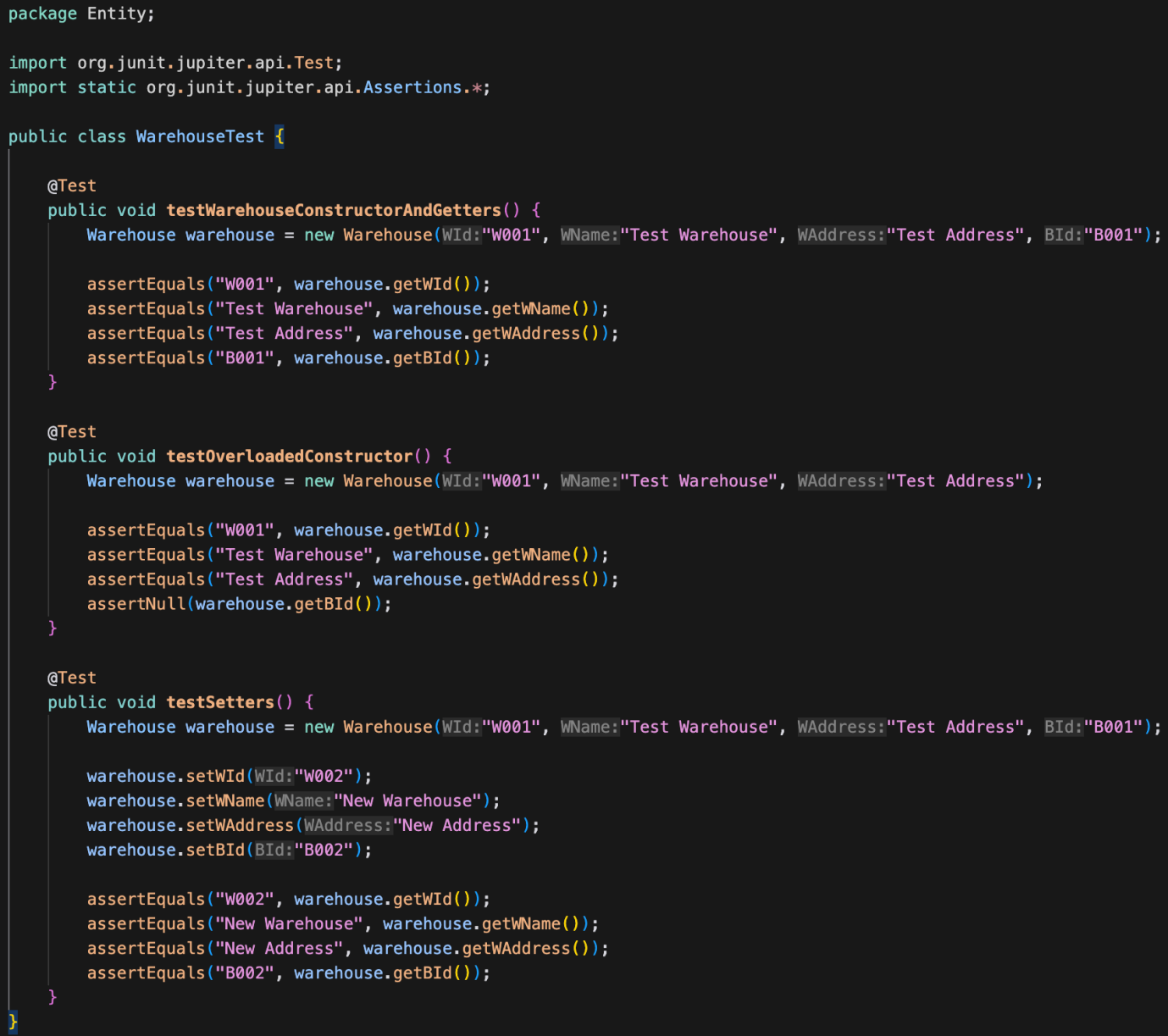
## 4.2 Employee test case



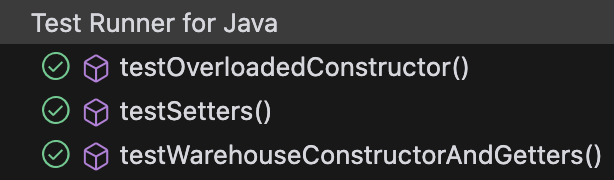
Output:



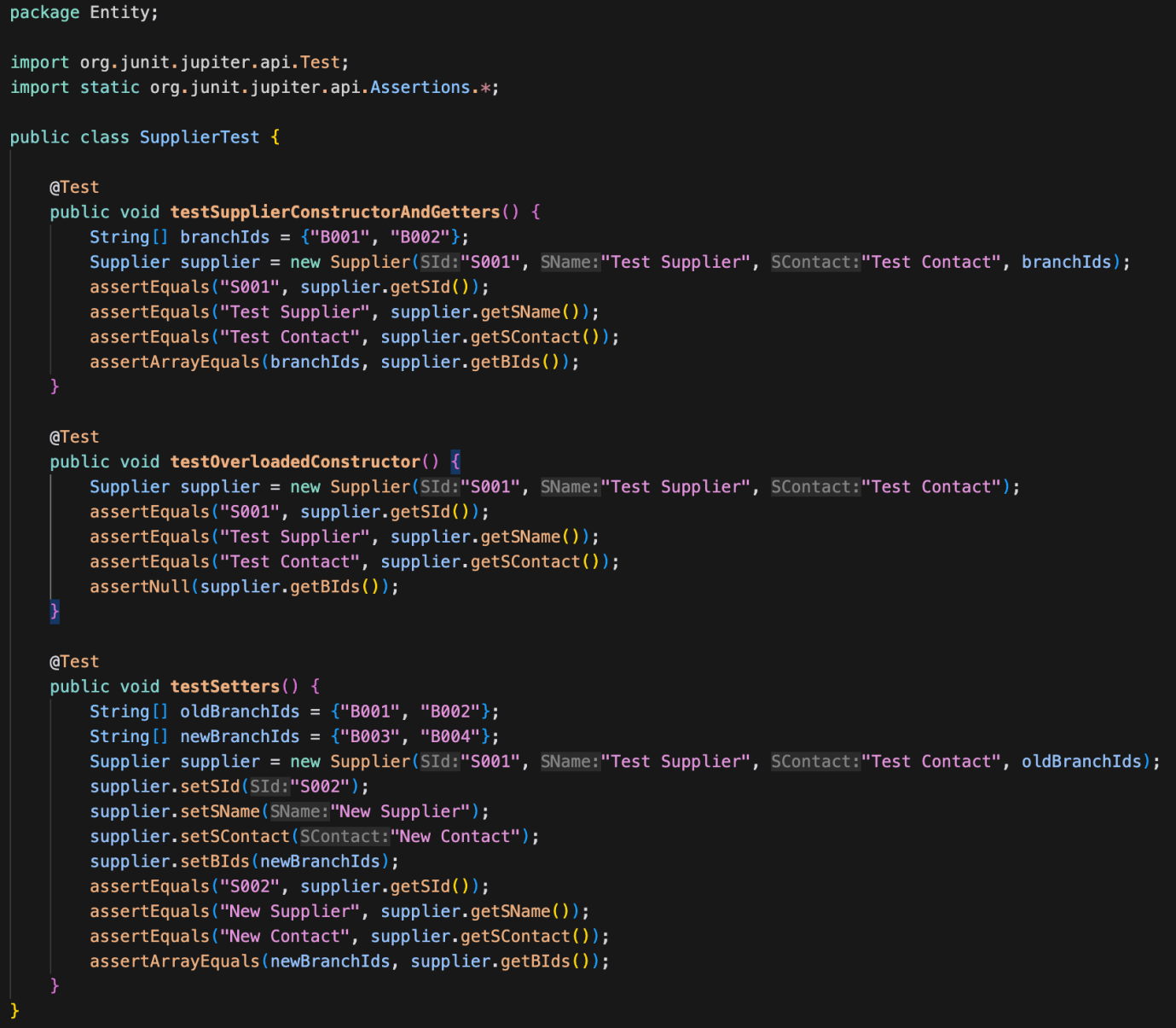
## 4.3 Warehouse test case



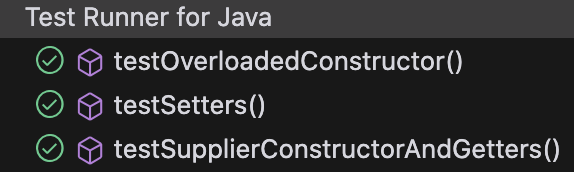
Output:



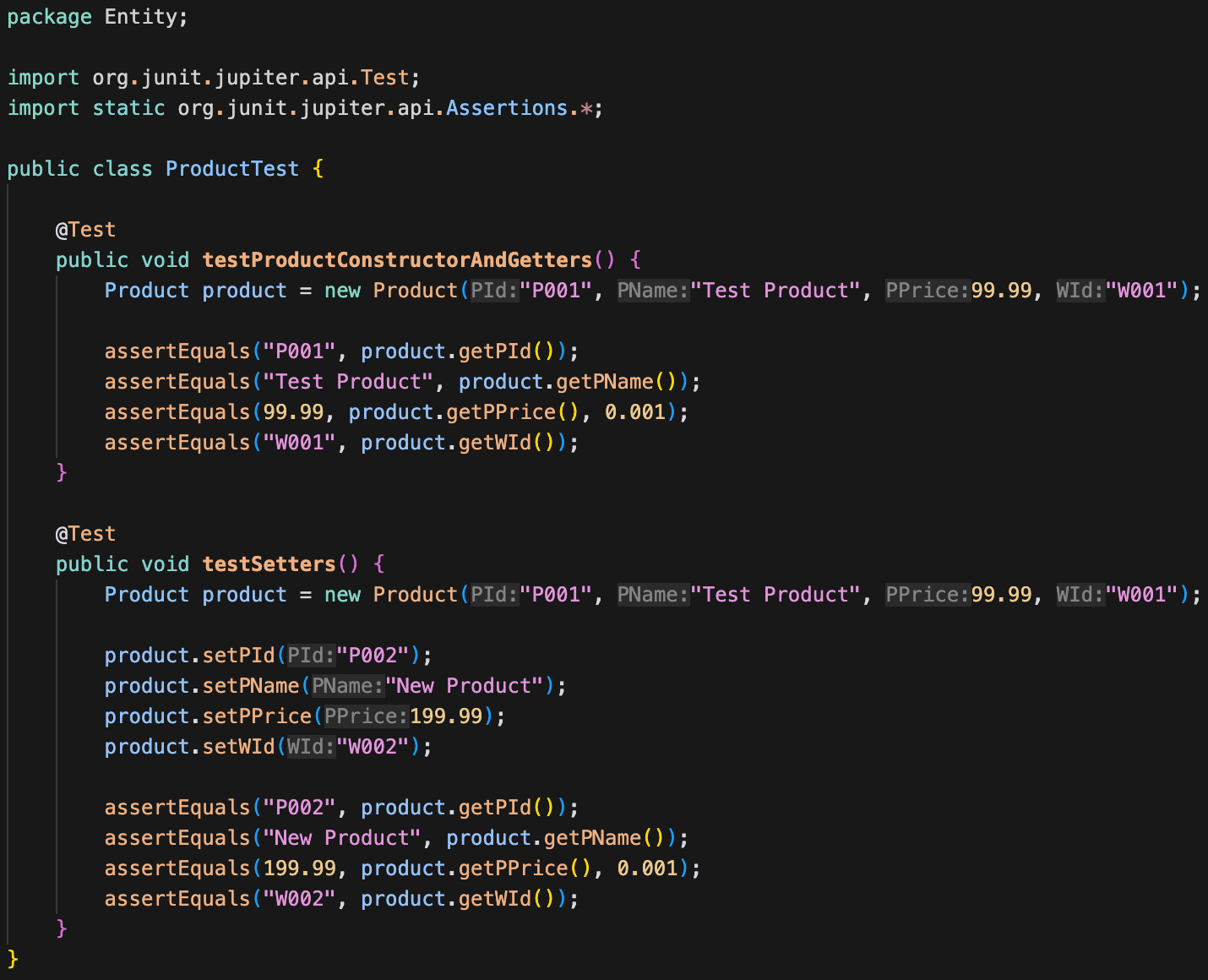
## 4.4 Supplier test case



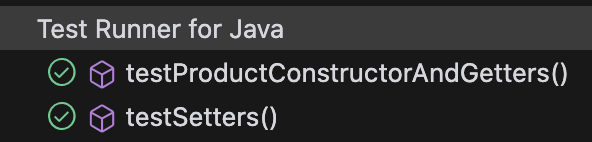
Output



## 4.5 Product test case

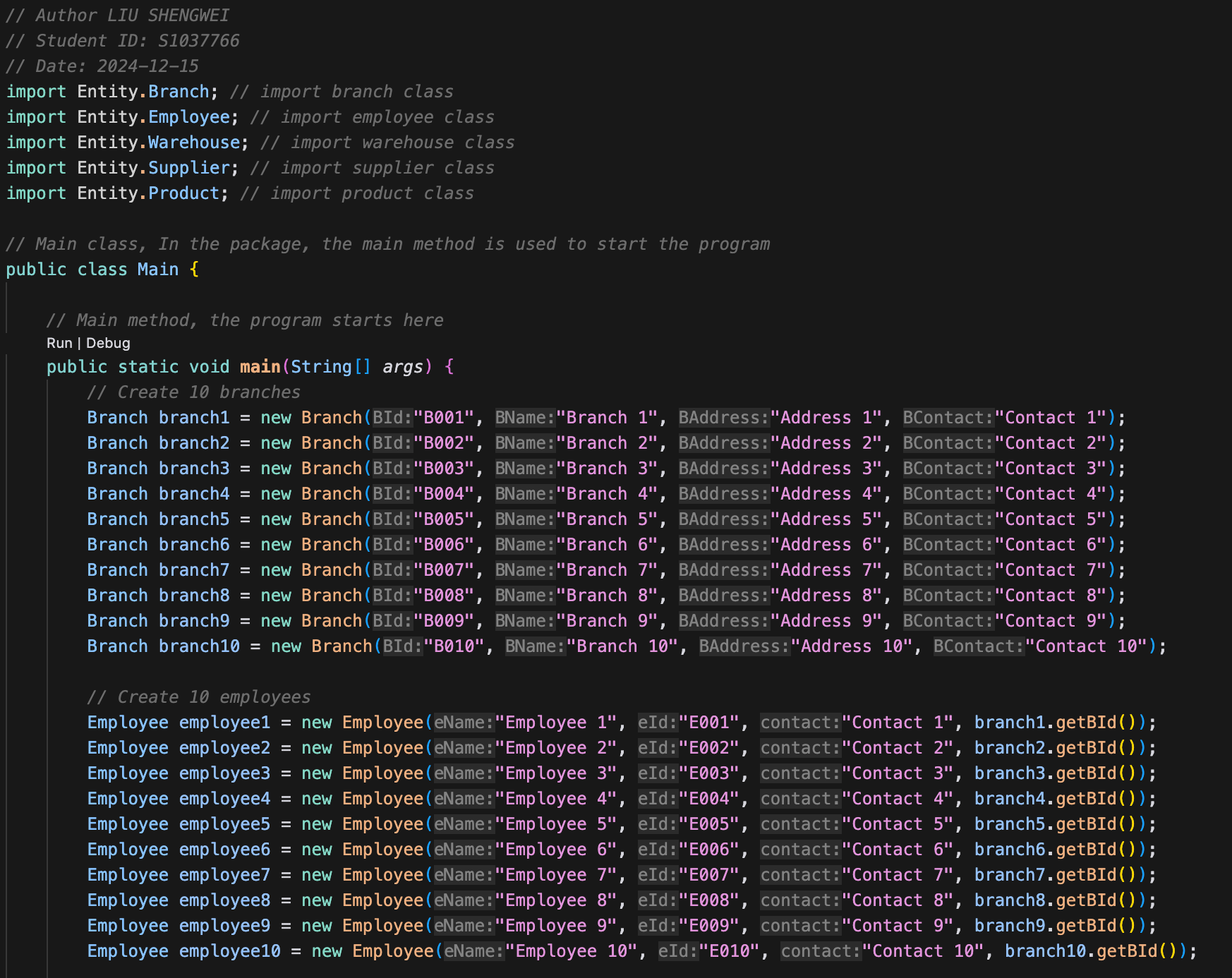


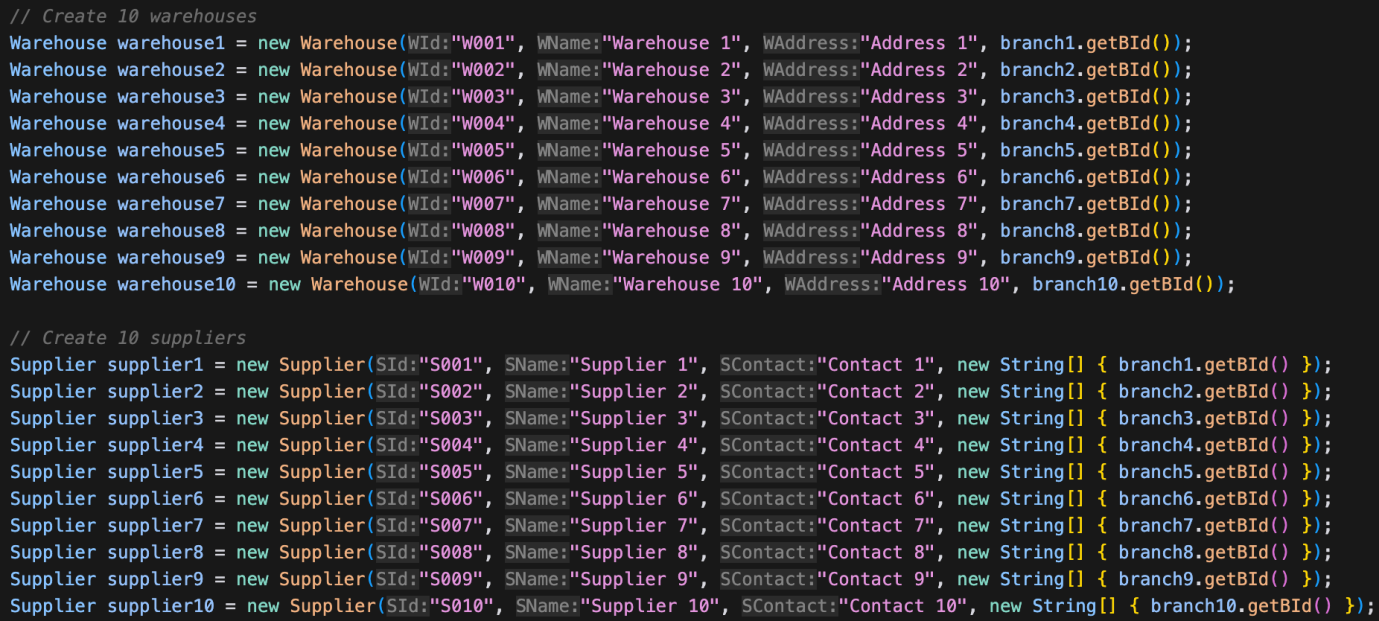
Output:

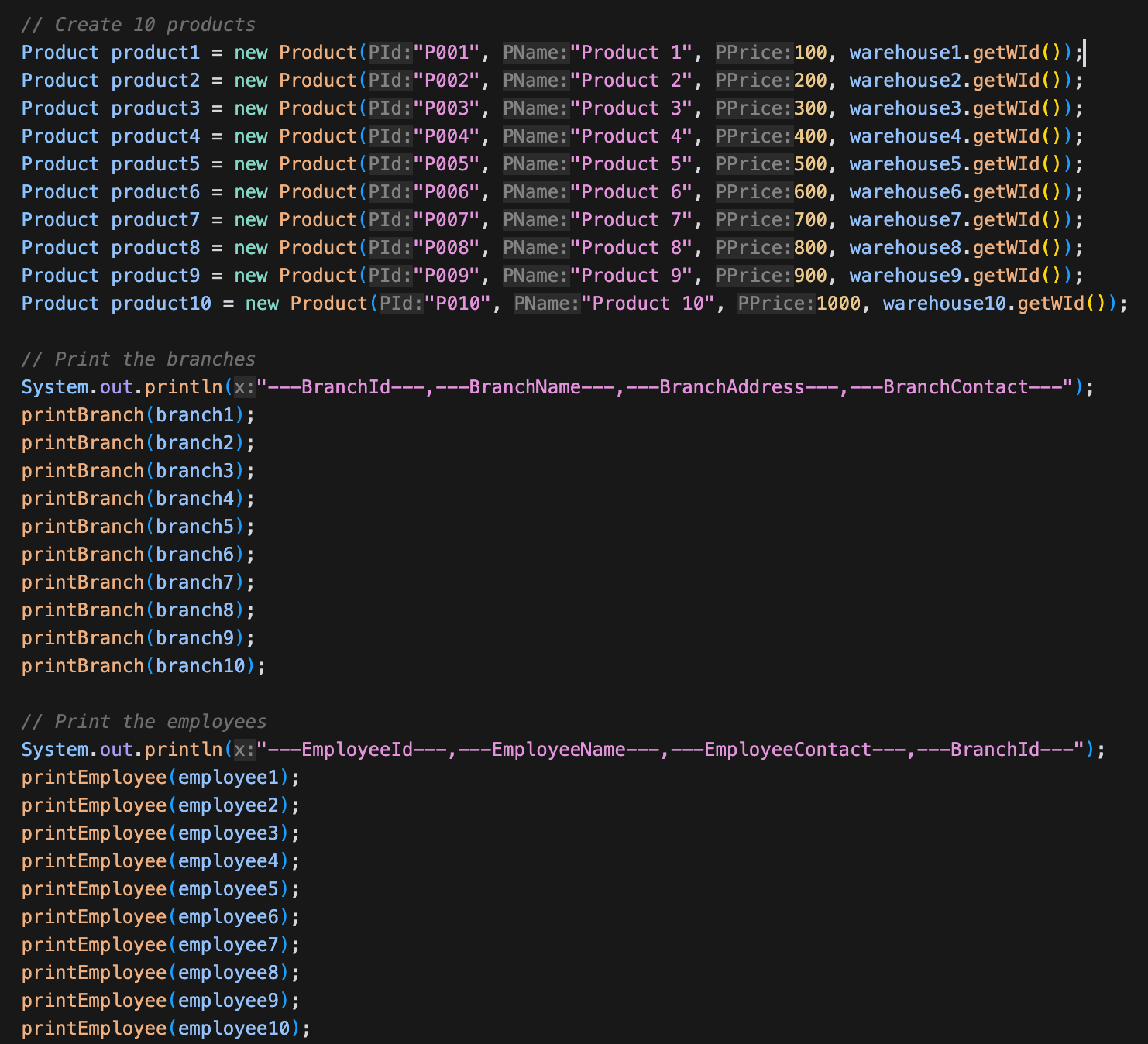


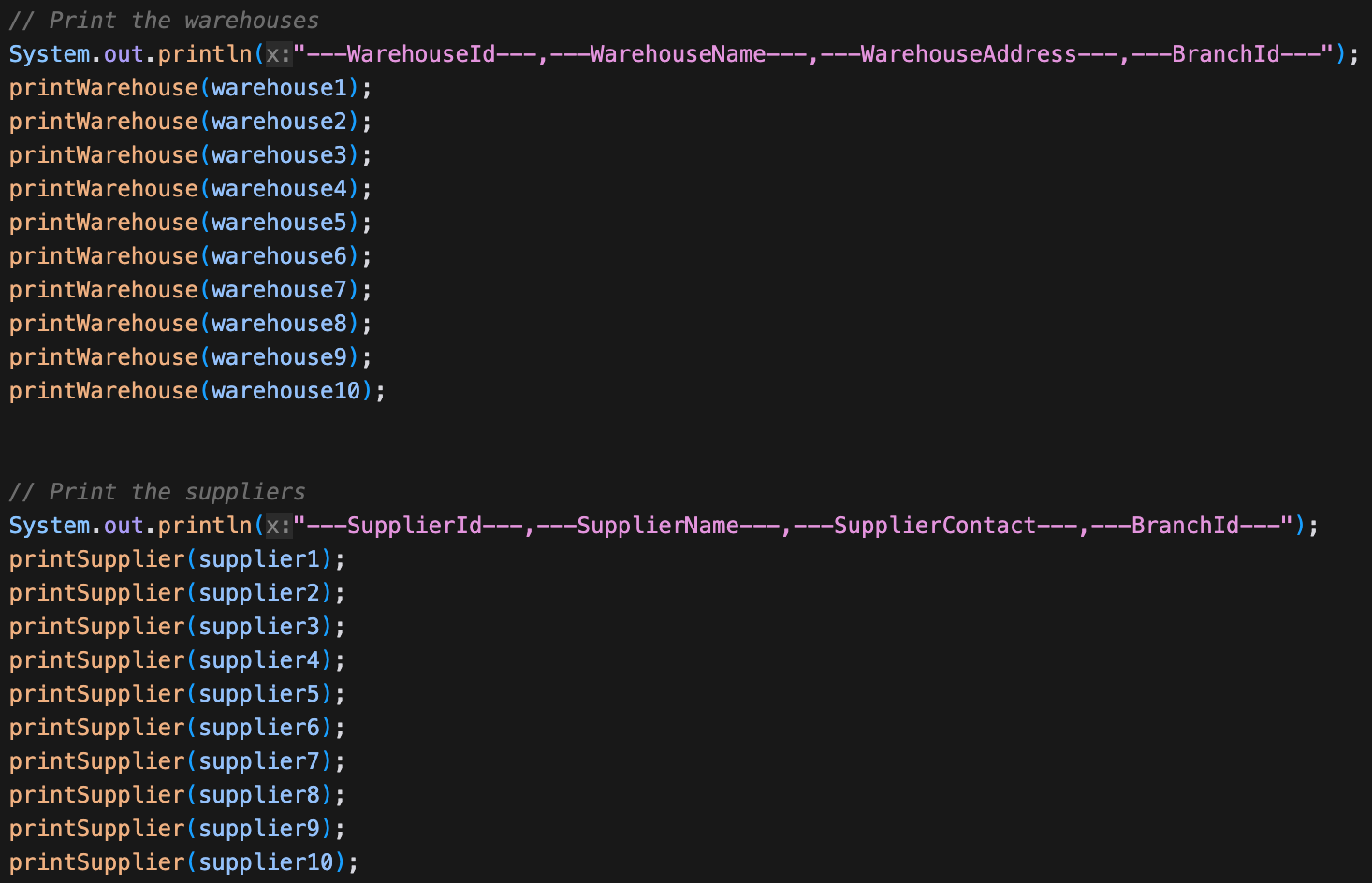
# Program Listings(with coments and explanations)

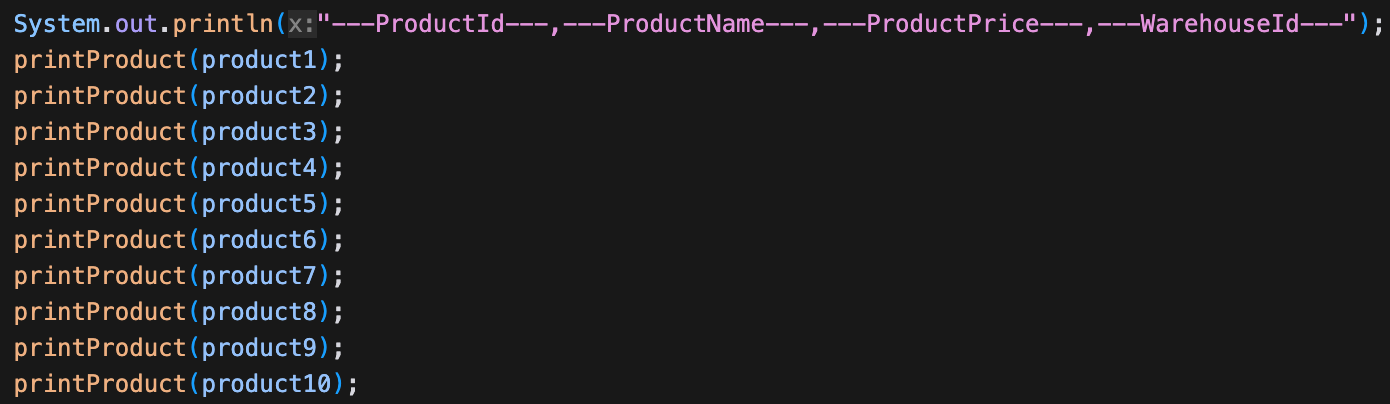
All the individual classes have already been explained. Next, I'll explain the main program.



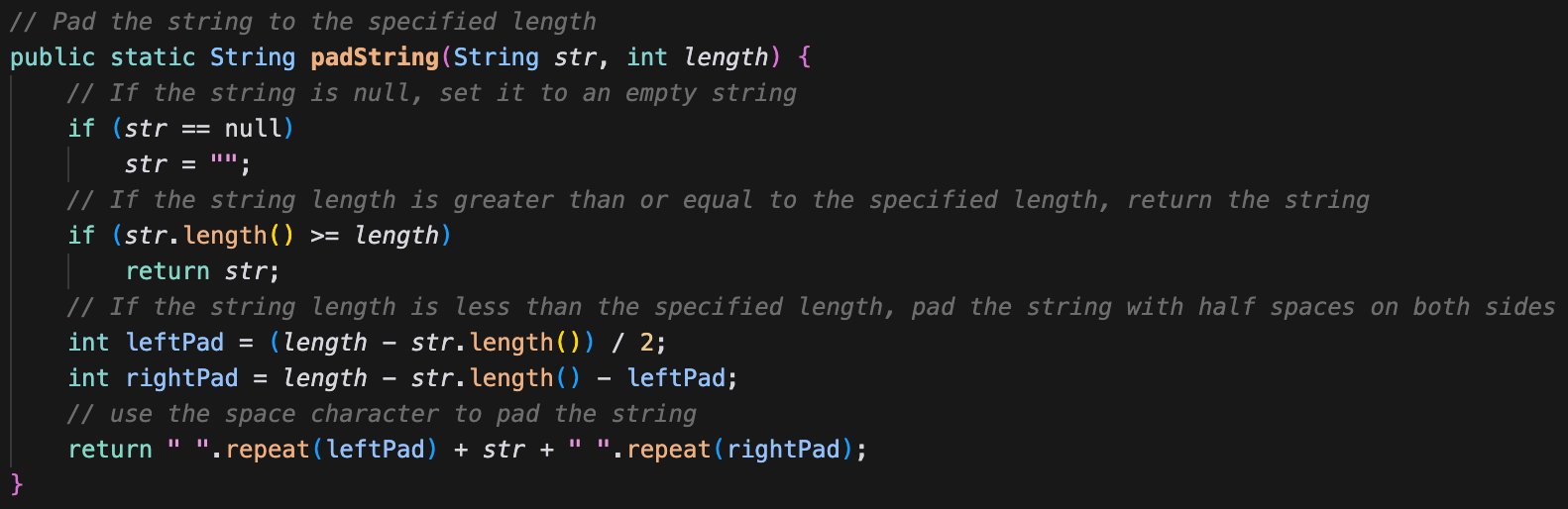












# 6. Critical Evaluation

## 6.1 Strengths

* Clear code structure that follows basic object-oriented principles, with each entity class having single responsibility and reasonable property and method definitions
* High test coverage, with complete unit tests written for each entity class, including constructors, getters/setters, and other basic functionalities
* Consistent naming conventions and comprehensive code comments, enhancing code readability and maintainability

## 6.2 Weaknesses

* Lack of data validation –

Currently, there is a significant lack of data validation in the system. For input parameters such as ID format, price range, contact format, etc., there is no effective validation process. This deficiency can easily lead to data integrity issues, allowing the system to accept data that does not conform to format specifications or logical requirements, thereby interfering with normal business processes and the accuracy of data analysis, and bringing a heavy burden to data quality maintenance.

* Weakness in Entity Class Relationships

The relationship architecture between entity classes is overly simplistic, maintained only through ID strings and lacking strong type constraints. This weakness poses a hidden danger to data consistency. In complex operations such as data update and associated query, it is extremely easy to trigger data association disorders and frequent occurrence of inconsistency errors, greatly weakening the reliability and stability of system data and increasing the complexity and error probability of data management.

* Absence of Serializable Interface
* The system does not implement the Serializable interface, which may cause potential obstacles during the data persistence process. When performing data storage, transmission, and recovery operations, it is highly likely to trigger a series of compatibility problems and errors, such as inability to accurately restore objects and data loss or damage, seriously threatening data persistence.

## 6.3 Future Enhancement

1. Suggest adding data validation layer:

public void setBId(String BId) {

if (BId == null || !BId.matches("B\\d{3}")) {

throw new IllegalArgumentException("Invalid Branch ID format");

}

this.BId = BId;

}

2.Consider adding business logic layer:

public class BranchService {

private List<Branch> branches;

public void addBranch(Branch *branch*) {

*// Business logic validation*

if (isBranchExists(branch.getBId())) {

throw new BusinessException("Branch already exists");

}

branches.add(branch);

}

}

3. Suggest implementing domain model relationships:

public class Branch {

private List<Employee> employees;

private List<Warehouse> warehouses;

public void addEmployee(Employee *employee*) {

if (employee.getBId().equals(this.getBId())) {

employees.add(employee);

}

}

}

These improvements would make the project more robust and complete. Additionally, consider adding:

 Exception handling mechanism

 Logging functionality

 Data persistence layer

 Transaction management

 Access control system

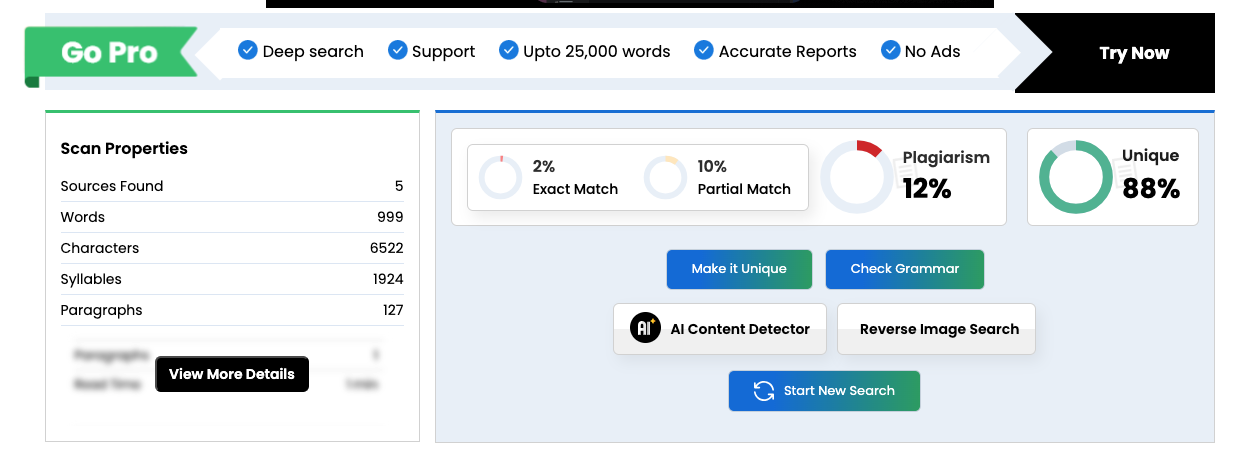
These additions would make the project more suitable for enterprise-level applications.

# 7.References

1.Horstmann, C.S. (2019) *Big Java: Early Objects, 7e Abridged Bound Print Companion with Wiley E-Text Reg Card Set*.

2.Horstmann, C. (2021) *Core Java, Volume I: Fundamentals, 12e*. Addison-Wesley Professional.

# 8. Plagiarism Checker



# 9. Presentation

This project is a short-term solution to the problems encountered by Mr. John Doe. It is dedicated to helping the client solve the issues. Currently, it is in the stage of design and simple development. We use classes to simulate the current business relationships. We have established the "Branch" class based on the client's branches.

To manage the employees under the branches, we have set up the "Employee" class. To manage the warehouses of the branches, we have created the "Warehouse" class. To manage the suppliers, we have established the "Supplier" class. And to manage the products sold, we have created the "Product" class. Many of these classes are related to the "Branch" class. Products will have connections with the "Warehouse" class.

If the class relationships are successfully established, we will proceed to the development of the next step of business logic. Finally, we will carry out the development, testing and launch of the Graphical User Interface (GUI).