**PROJECT 5 REPORT**

**INTRODUCTION**

This report presents a thorough examination of Java project 5 that makes use of advanced object-oriented programming (OOP) ideas. The project consists of 16 Java classes, each of which is designed to represent different scenarios in a medical transaction system. This report intends to outline how inheritance, polymorphism, encapsulation, composition, and abstraction are implemented in this project.

**INHERITANCE AND POLYMORPHISM**

Inheritance is a cornerstone of OOP and is prominently incorporated in the project. The subclasses Doctor, Patient, Receptionist, and Admin all extend the Person superclass, establishing a clear hierarchical structure. This strategy not only promotes code reusability but also improves maintainability by centralizing common characteristics and functions in the Person class. This commitment to the DRY (Don't Repeat Yourself) philosophy reduces repetition and promotes efficient code development.

Method overriding easily integrates polymorphism, another important OOP concept. To provide specialized string representations, each subclass (Doctor, Patient, Receptionist, and Admin) customizes the toString() method. As a result, various objects reply to the same method call in a unique way, customizing their output to their respective kinds. The toString() function of a Doctor object, for example, reveals specific facts about a doctor, but the toString() method of a Patient object reveals patient-specific information. This polymorphic behavior improves the flexibility and maintainability of the code.

**ENCAPSULATION**

Encapsulation, a key principle of OOP, is utilized the design of the project. Private fields are used sparingly to protect sensitive data such as names, contact information, and financial information. Public getter and setter methods provide restricted access to these fields, ensuring that data is only modified via predefined interfaces. This safeguard protects data from illegal external access or unintended change, enhancing data integrity and security.

**COMPOSITION**

Composition is effectively employed through the integration of instances of other classes within a class. For instance, the Transaction class encompasses instances of Doctor, Receptionist, and Patient. This composition approach facilitates the creation of more intricate objects by assembling simpler ones. It champions modularity, simplifying the management of individual components. Moreover, it encourages code reuse, as instances of classes can be leveraged across various contexts, enhancing code versatility.

**ABSTRACTION**

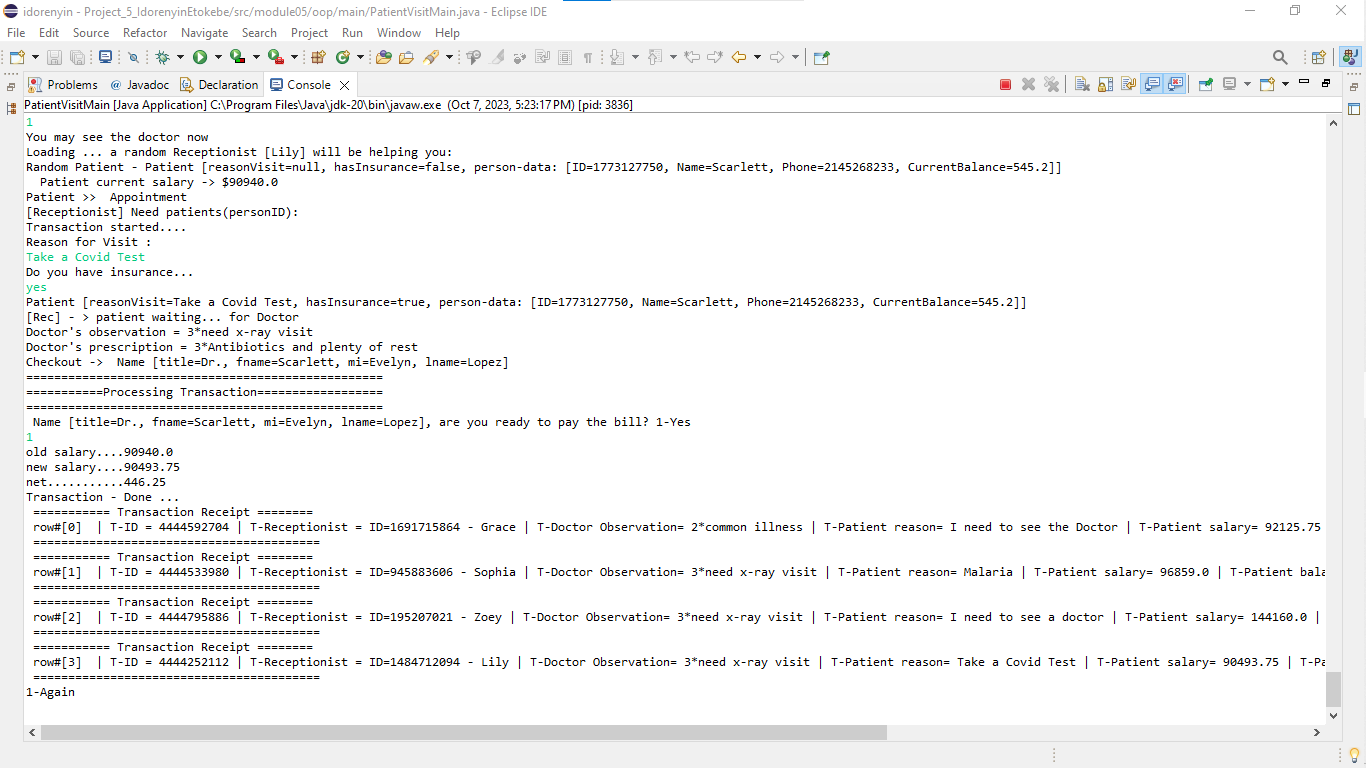
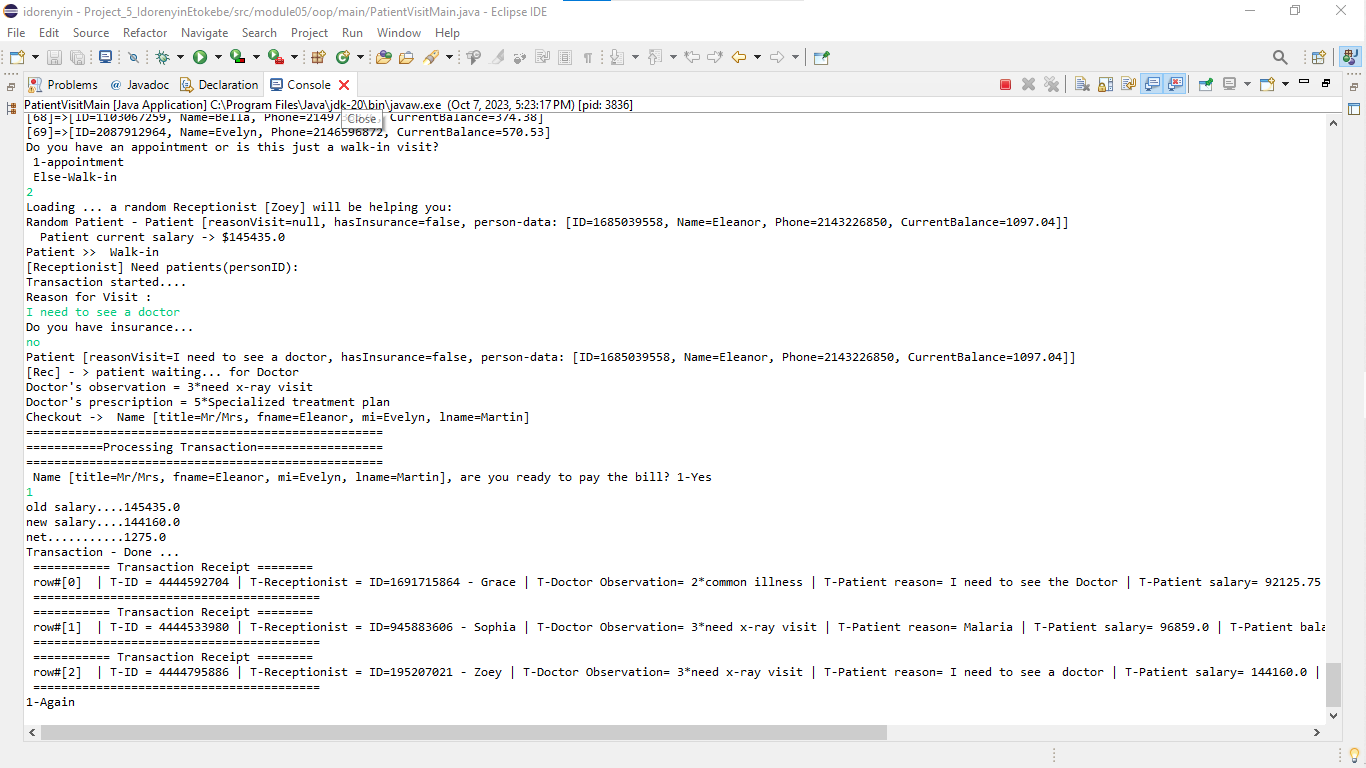
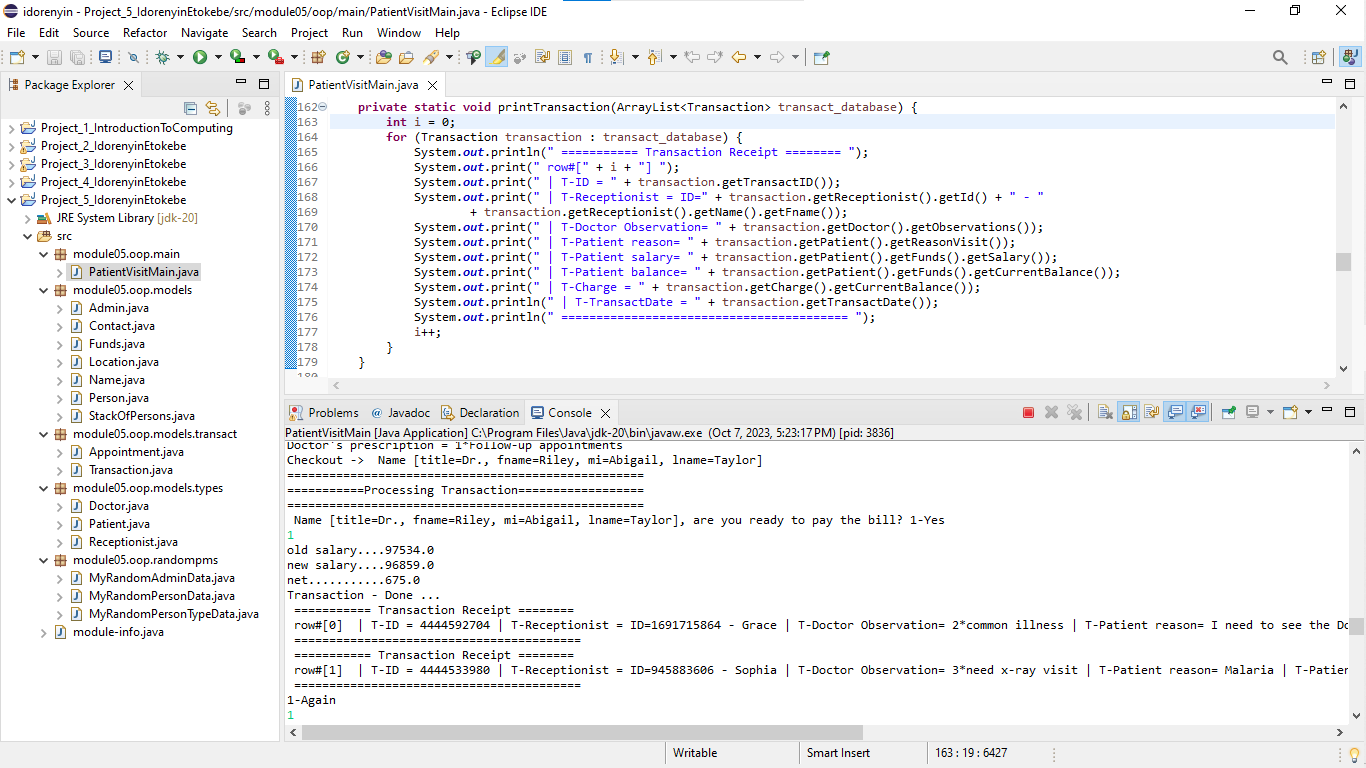
Abstraction is showcased through utility classes like MyRandomAdminData, MyRandomPersonData, and MyRandomPersonTypeData. These classes encapsulate intricate logic for generating random data, providing a high-level interface to access various types of data without revealing the underlying implementation details. This level of abstraction streamlines the process of obtaining random data, enhancing code readability and maintainability.

**CONCLUSION**

In summary, the Java project leverages a spectrum of OOP principles to forge a well-organized and easily maintainable codebase. By harnessing inheritance, polymorphism, encapsulation, composition, and abstraction, the project not only models a medical transaction system but also serves as an exemplary illustration of how OOP fortifies code organization, reusability, and readability.

The implementation of these principles demonstrates a profound comprehension of fundamental OOP concepts and exemplifies their practical application in software development.

**SCREENSHOTS**

****