James Osibogun

UML Class Diagrams

SNHU

Professor Manning

III. Software Design

General Responsibility Assignment Software Patterns recommends that software designers follow these principles to design and analyze software (SNHU).

  Another important concept in object-oriented programming

is inheritance, which is acquiring the traits of one's predecessors. Inheritance uses and modifies

another class's public variables and methods (the superclass). The subclass is a specialized version of the more general superclass, an is-a relationship.

  Polymorphism is writing code using a superclass that can be

executed for any subclass object. With object-oriented programming, you focus

on what the program will manipulate.

**Encapsulation**is the process of combining all an object's attributes and methods into a single package. **Information hiding**is the concept that other classes should not alter

an object's characteristics—only the methods of an object's class should have

that privilege. Five of its integral components: are classes, objects, polymorphism, inheritance, and encapsulation.

  Classes are the basic building blocks of object-oriented programming. A class's fields, or instance variables, hold data, and the instance of a class possesses classes and objects, three essential features of object-oriented languages are polymorphism, inheritance, and encapsulation.

  The definition of a class is a set of program statements that

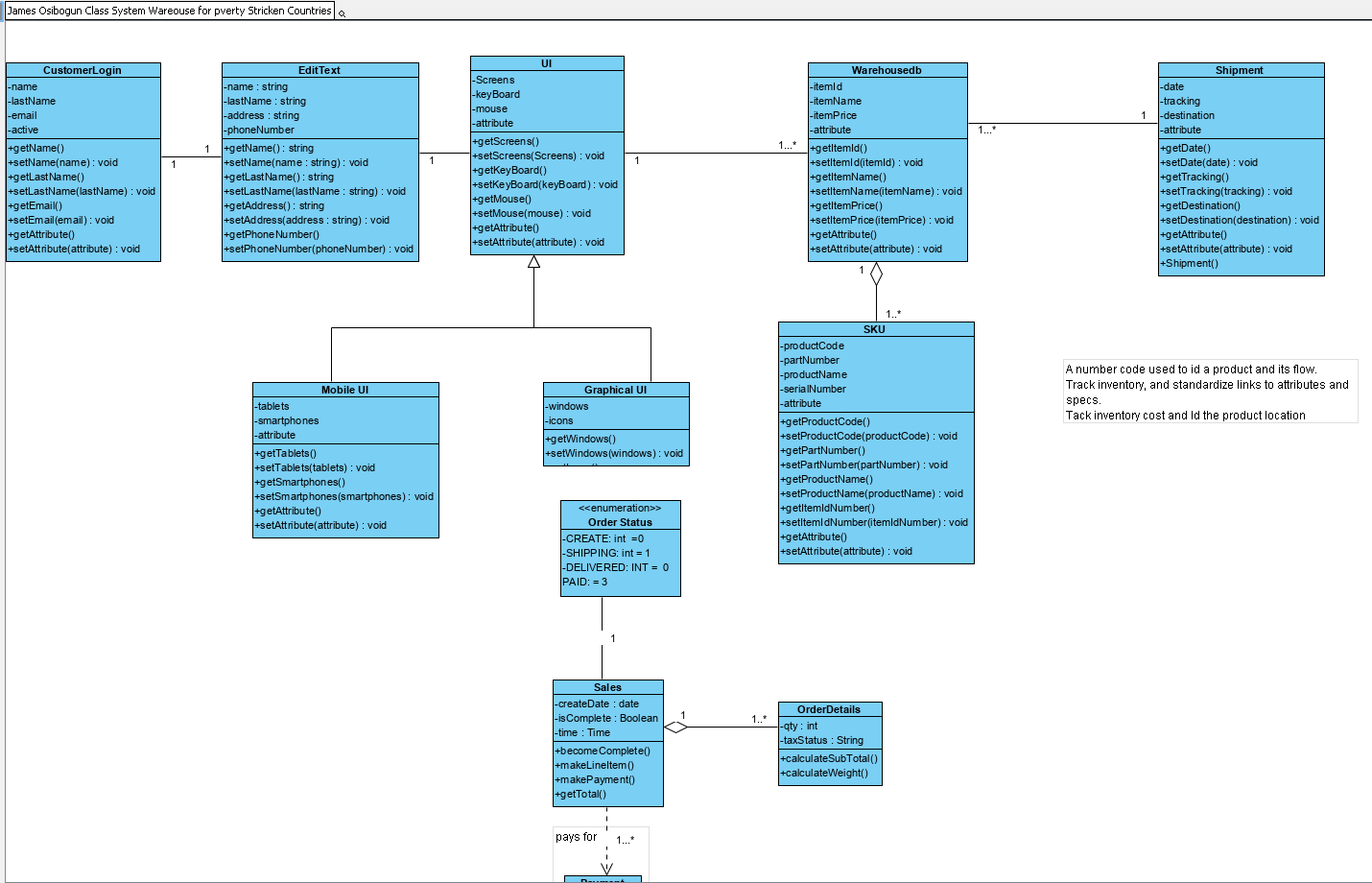
tell you the characteristics of the class's objects and the methods that can be applied to its objects. A class contains a name, data, and methods (Farrell,2013). Programmers use

a class diagram to illustrate class features. The purposes of many methods

contained in a class can be divided into three categories: set plans, get

strategies, and work methods (Farrell,2013). By the end of this report, the chart will explain how the chart flows. The report paper will use polymorphism, encapsulation, and inheritance.

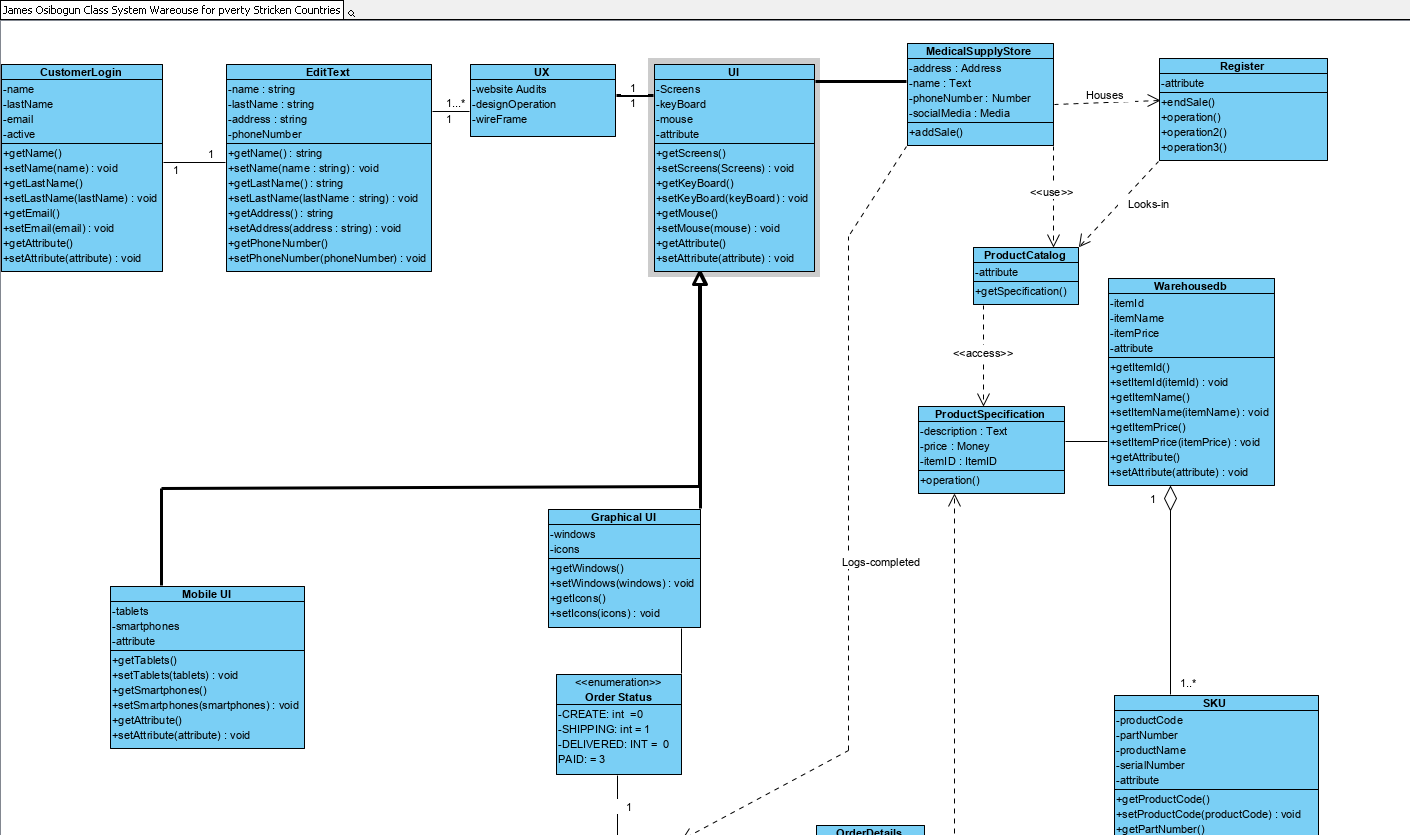
UML Class Diagrams:



A picture containing text, clock

Description automatically generated

**\*Updated UML Class diagram**



Diagram

Description automatically generated

**SDLC & Conclusion**

Software development life cycle and describes how a development team uses the diagrams to code the system. A well-defined process is a key to building excellent software and is integral to the software development lifecycle. It ensures fast and quality delivery.

 We are implementing case stories, sequence models, use case diagrams, and class diagrams for the project. Prevalent software development models include Waterfall, Spiral, feature-driven, Agile, Lean, Rapid Application Development, and [Scaled Agile Framework (SAFe)](https://www.netsolutions.com/insights/everything-about-scaled-agile-framework/). The products are medical grade and must be organized in the database and at the warehouse. The databases can be created using MongoDB, PopSQL, SQL from Microsoft, or Docker. However, whatever database program is used must be documented in the report, and the stakeholders must have a developer to take over the program and familiarize themselves with it.

SDLC can assist you in easing your product’s entire software development journey. Here’s how the SDLC process adds value to your software development ventures:

* Faster time to market
* High-Quality Software
* Facilitates management control — whether it is a small or a large project, the SDLC model offers project management control to help them visualize and track development progress
* It brings the development team on the same page.
* It Breaks existing silos among the teams.
* A clearer perspective of roles and responsibilities —

The client defines a problem that needs to be solved, which further forms the basis for finalizing the requirements. The design phase of the software development life cycle focuses on [creating software architecture](https://www.netsolutions.com/insights/why-software-architecture-matters-to-build-scalable-solutions/), prototypes, and [user experience design](https://www.netsolutions.com/experience-design). I suggest that stakeholders get familiar with all the developers and upper management. I could make frequent trips to the company or make many conference calls until it is completed.

The UI/UX team builds a [prototype version](https://www.netsolutions.com/insights/poc-vs-mvp-vs-prototype/) of the software to validate its look and flow of design elements. The concept is realized after an agreed meeting between stakeholders and owners. The DevOps teams can deliver(deliverables) in one go or provide in segments.

The DevOps team can be designated as someone or a team for testing. Logic testing or black box and white box testing are authorized. Also, use the whiteboard in the office to conduct desk checks. The entire software or a part of it goes into the production environment phase after developing, testing, fixing, retesting, and validating. If you follow the Agile SDLC process, deployment could be the launching of MVP and other features.

The common types of maintenance include:

* Corrective Maintenance
* Perfective Maintenance
* Documentation of Requirements: We created an excel sheet that clearly and comprehensively listed all the Soaq team’s requirements.
* Proof of Concept (POC): A proof of concept was created to validate the idea’s technical feasibility.

The diagrams will be used as a basis for future changes or modifications.

Reference:

Larman, C. (2012). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development for Capella University (3rd ed.). Pearson Learning Solutions. <https://mbsdirect.vitalsource.com/books/9781256882763>

[**SHARMA**](https://www.netsolutions.com/insights/author/rajnish_kumar_sharma/)**,R.K (2022, AUG 10).** SDLC: What it is, Software Development Processes & Best Practices. Retrieved on December 21, 2022, from

<https://www.netsolutions.com/insights/software-development-life-cycle/>

n.a(n.d).Inheritance and Polymorphism. Retrieved on December 27, 2022, from <http://www.cs.utsa.edu/~cs3443/notes/chapter10/ch10.html>

Farrell, J (2013). Programming logic and design. Cengage Learning