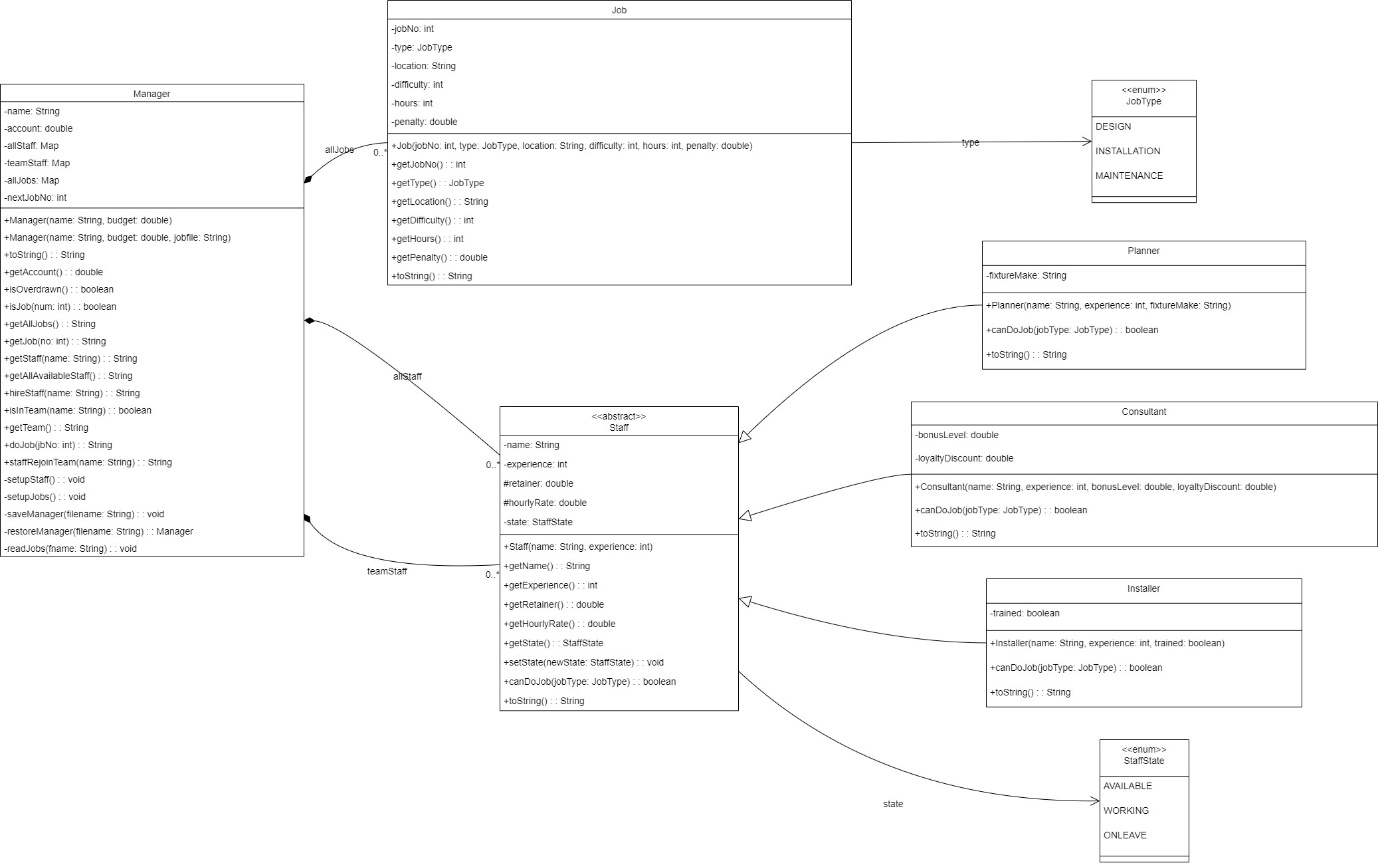
HISS System Design Document

# ****1. Introduction****

In this report the design and implementation of the HISS (Home Improvement Scheduling System) simulation is described. The system administers the finances of projects, personnel allocation and status of projects. In this document, the UML class diagram will be given and three major design decisions that influenced the overall design of the system will be discussed.

# ****2. UML Class Diagram****

The architecture of the HISS system is pictured in the following UML Class Diagram. This chart shows the principal classes, their attributes and methods and the defining relationships among them which the sign of clear and modular design.



# ****3. Important Design Decisions****

**There were three main design choices that had to be made in order to achieve the robustness, maintainability and extensibility of the HISS system.**

## ****3.1. Abstract Staff Class to specialize the Role****

**That system makes use of an abstract Staff class representing the common basis of all personnel, and concrete subclasses (Planner, Consultant, Installer) that share common attributes and operations. Each subclass is specialized by overriding abstract methods including canDoJob() that determines their special ability and computes their particular hourlyRate and retainer. such as, a Planner does DESIGN jobs.**

**The alternative, having one Staff class and conditional logic (e.g. StaffType enum and then if-else statements), was taken into consideration. The downside of this, though, would be a less modular design and additional complexity of the Staff class, particularly with new roles. Entirely independent, unrelated staff classes would also bring about a lot of duplication in the code.**

**The design selected makes use of polymorphism, whereby the Manager can treat all the staff in a similar manner using the Staff type, which makes operations easier when dealing with collections. This enhances a high degree of code reusability, since common attributes and behaviour are centralized, and is more extensible, since new staff roles only need a new subclass. The result is a more maintainable, flexible and scalable system, and follows fundamental object-oriented principles.**

## ****3.2. Manager has a centralized Job and Staff Management****

**One of the main design choices was to close all the job and staff manipulation in the Manager class. The Manager keeps collections of Maps of allJobs and allStaff (and teamStaff), and directly manages operations such as loading jobs (readJobs()), hiring staff ( hireStaff()), and assigning jobs (doJob()).**

**Other options considered were the introduction of specific Service classes (e.g. JobService, StaffService) to separate these concerns. This would increase the abstraction unnecessarily on the scale of the current HISS simulation, though it would be beneficial to more complex, larger systems.**

**The idea of centralizing the management in Manager would be simple and efficient. It simplifies access to the data, based on the needs of O(1) average-time lookups by job number or staff name, which is essential to operate responsively, it uses HashMap. Such tight control is required to make the cohesion in the Manager class high, where it will be a main conductor of all activities in the project. This option provides a balance between design and practical considerations on the on-going project scope.**

## ****3.3. StaffState Enum-Based Staff Management by state****

**The operational state of any given staff is explicitly monitored through StaffState enumeration, which states AVAILABLE, WORKING and ONLEAVE. Manager methods update these states; hireStaff() (to WORKING), doJob() (to ONLEAVE upon completion), and staffRejoinTeam() (back to WORKING).**

**The other options were several boolean flags (e.g. isAvailable, isWorking) or implicit state based on the presence of staff in various collections. Boolean flags may cause invalid state combinations, and can be unwieldy when having many states. Implicit state inference may be unclear, and less evident.**

**The StaffState enum offers a better clarity, type safety and maintainability. It provides a directly understandable, self-documenting model of status, and provides type safety against invalid states, and requires explicit state transitions to robust system behavior. Staff eligibility conditions can be made tidy and dependable. The given design is rather effective at handling the staff lifecycle, which leads to a stable and bug-free codebase.**

# ****4. Conclusion****

The HISS system is designed keeping the object-oriented principles in mind where inheritance is used to model staff roles and explicit state management through enums. Manager class will act as the main controlling device of the human resource, as well as job assignment. The implementation performs well in regards to the functional requirements.

There is a slight incompatibility with GeneralTest.isInJobRange. This test requires job IDs in a certain lower range, whereas Manager.nextJobNo had to be set to 1000 to make sure that other important tests, like ManagementJobsDoing and ManagementGeneral, succeed properly. Since the external test files can not be changed, this one test failure is an inevitable artifact of the test environment and not a bug in the fundamental logic of the system. In general the system design and functionality is strong and compliant with the project requirements.