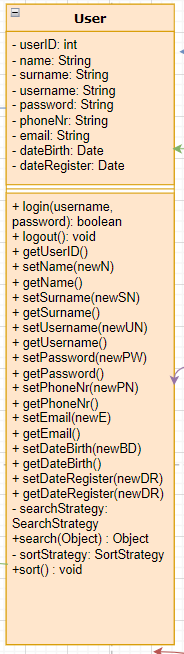
**Strategy Pattern**

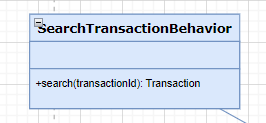
In the context of our cinema management system, the Strategy pattern is implemented through the SearchStrategy interface, which defines a common method search(Object) Object. This interface is then concretely implemented by several classes: SearchReportBehavior, SearchEmployeeBehavior, SearchMovieBehavior, and SearchTicketBehavior.

By using the Strategy pattern, we can dynamically assign search behaviors to various user roles within the system. For instance, an admin might be granted the SearchEmployeeBehavior to manage staff records, while a customer might use SearchMovieBehavior to find movies. This design allows for clear separation of concerns and promotes the open/closed principle, where new search behaviors can be added without modifying existing code. Furthermore, it enhances code maintainability and readability by encapsulating the search algorithms into distinct classes. This modular approach ensures that each search behavior can be independently developed, tested, and modified, leading to a robust and easily extendable system.

Things we have included in our class diagram:

 Added searchStraregy and sortStrategy into user class, meaning that a user has a searchingStrategy and sortingStrategy, which means that all the subclasses of user, in our case the user roles will have different searching and sorting methods they can perform.

 SearchStrategy interface that will have a generic search method depending on the type of search, more specific behaviour will be implemented in the concrete classes that will implement a SearchStretegy.



 These are concrete classes that will implement the SearchStrategy or SortStrategy and will be very specific, for one functionality only. This is the only way we can differentiate the actions a certain user can perform, for example only admin and manager will be able to sort the reports, and no other user levels.