Courier Management System

Coding Task 7

Exception Handling

(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,thow & throws keyword usage)

Define the following custom exceptions and throw them in methods whenever needed . Handle all the excpetionsin main method,

1. TrackingNumberNotFoundException :throw this exception when user try to withdraw amount or transfer amount to another acco

2. InvalidEmployeeIdException throw this exception when id entered for the employee not existing in the system

---

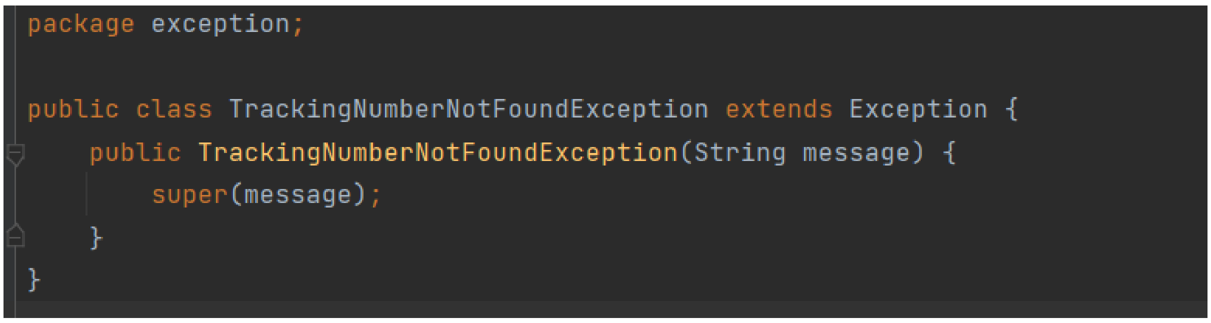
This task focuses on handling user-defined exceptions in Java to ensure that a program can gracefully handle unexpected scenarios. It demonstrates the use of checked exceptions by defining two custom exceptions:

1. **TrackingNumberNotFoundException**
2. **InvalidEmployeeIdException.**

These exceptions are thrown under specific conditions and handled properly using try-catch blocks and appropriate exception propagation with throws keyword. By integrating these custom exceptions in real-time courier tracking and employee management systems, the application becomes more robust, user-friendly, and error-resilient.

Task 7.1 TrackingNumberNotFoundException

This custom checked exception is used to indicate that the tracking ID provided by the user does not exist in the system. When a user tries to get the order status using a tracking number that is not present in the database, this exception is thrown to inform the user that the requested order could not be found. This ensures that invalid tracking operations are caught and handled properly, maintaining application stability and providing meaningful feedback.



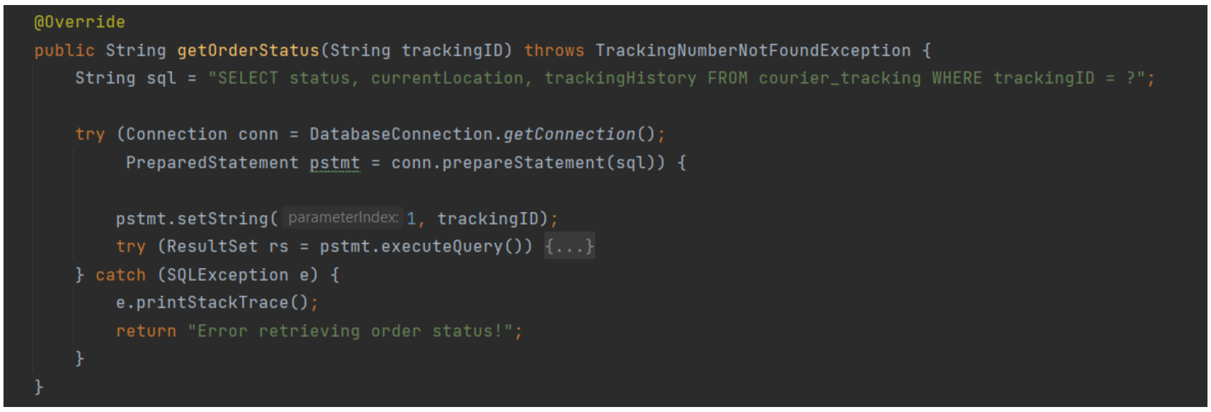
```java
package exception;

public class TrackingNumberNotFoundException extends Exception {
    public TrackingNumberNotFoundException(String message) {
        super(message);
    }
}
```

Usage

getOrderStatus(String trackingID)

This method retrieves the current order status, location, and tracking history based on the tracking ID provided by the user. It executes a SQL query to check if the tracking ID exists in the database. If found, it returns detailed tracking information. If not found, it throws a TrackingNumberNotFoundException, informing the user that the ID is invalid or not available in the system.



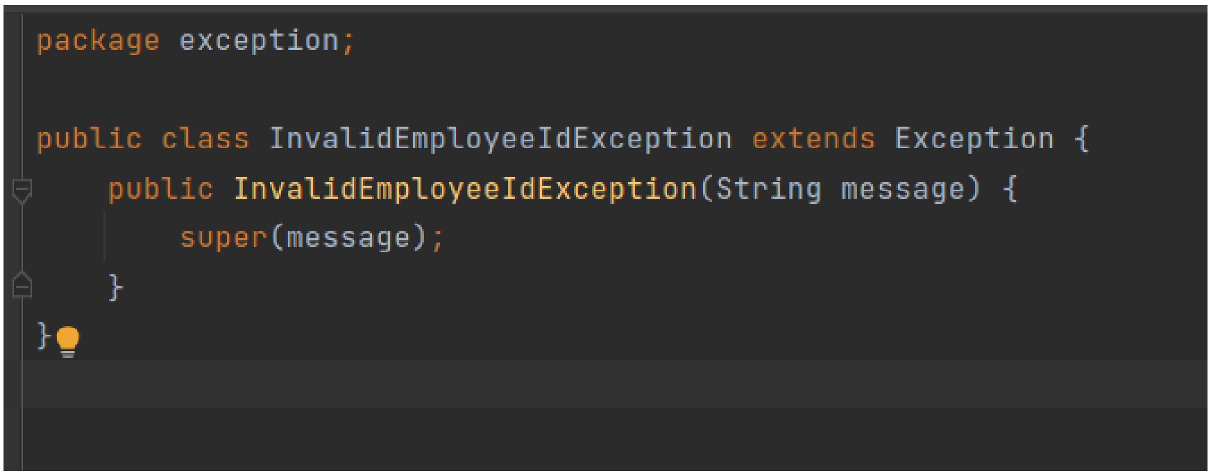
```
@Override
public String getOrderStatus(String trackingID) throws TrackingNumberNotFoundException {
    String sql = "SELECT status, currentLocation, trackingHistory FROM courier_tracking WHERE trackingID = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setString( parameterIndex: 1, trackingID);
        try (ResultSet rs = pstmt.executeQuery()) {...}
    } catch (SQLException e) {
        e.printStackTrace();
        return "Error retrieving order status!";
    }
}
```

Task 7.2 InvalidEmployeeIdException

The InvalidEmployeeIdException is thrown when an invalid employee ID is used in the system, particularly in cases where employee-related data is being retrieved. This custom exception helps validate the integrity of employee records by making sure operations are only performed on existing employees. If the provided ID is not found in the employee table, this exception alerts the system and avoids potential data inconsistencies.



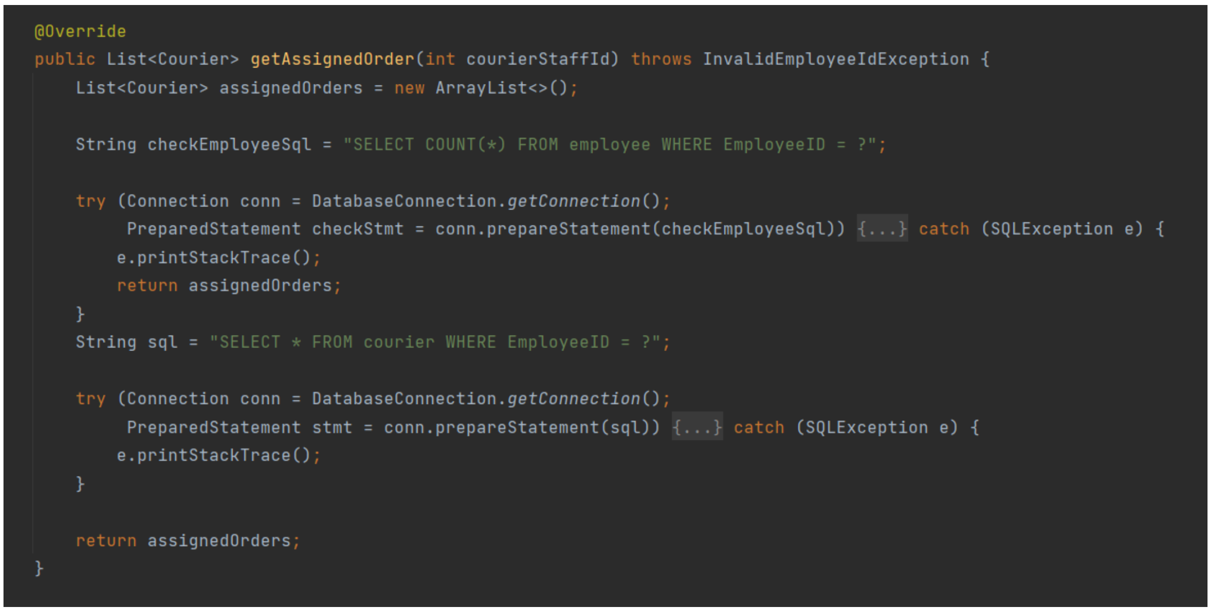
```
package exception;

public class InvalidEmployeeIdException extends Exception {
    public InvalidEmployeeIdException(String message) {
        super(message);
    }
}
```

Usage

getAssignedOrder(int courierStaffId)

This method fetches all orders assigned to a particular courier staff member using their employee ID. Before fetching the orders, it validates whether the employee exists in the system. If not, it throws an InvalidEmployeeIdException, indicating that the ID is invalid. This ensures operations are executed only for valid employees, enhancing the accuracy of the courier assignment process.



```java
@Override
public List<Courier> getAssignedOrder(int courierStaffId) throws InvalidEmployeeIdException {
    List<Courier> assignedOrders = new ArrayList<>();

    String checkEmployeeSql = "SELECT COUNT(*) FROM employee WHERE EmployeeID = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement checkStmt = conn.prepareStatement(checkEmployeeSql)) {...} catch (SQLException e) {
        e.printStackTrace();
        return assignedOrders;
    }
    String sql = "SELECT * FROM courier WHERE EmployeeID = ?";

    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {...} catch (SQLException e) {
        e.printStackTrace();
    }

    return assignedOrders;
}
```

Conclusion

The implementation of custom exceptions like TrackingNumberNotFoundException and InvalidEmployeeIdException enhances the maintainability and user interaction of Java applications. These exceptions provide specific, user-friendly messages for known error scenarios and help developers manage errors gracefully. With appropriate exception handling using try, catch, finally, throw, and throws, this approach ensures a robust and professional standard of programming that leads to reliable enterprise-level systems.