題目：

模擬一個美食外送的伺服端功能，他會收到顧客訂單、餐廳收單、外送員接單的請求，過程中可能會產生很多例外，依據不同狀況寫到 log。

設計並拋出 Checked Exception（業務邏輯錯誤）。處理並記錄 Unchecked Exception（程式碼邏輯或環境錯誤）。正確使用 Log4j 2 的不同 日誌級別（INFO, WARN, ERROR）。

程式、執行畫面及其說明：

直接把程式的說明都寫在註解。沒有把完整程式都放上來，只擷取重要的片段。

package org.example.model;

```
public enum OrderStatus {  11 usages
    PENDING,     // 等待餐廳取餐  2 usages
    ACCEPTED,    // 訂單已被餐廳接受  2 usages
    PICKED_UP,   // 外送員已取餐  2 usages
    DELIVERED    // 餐點已送達  1 usage
}
```

package org.example.service;
public class DeliveryService

```
// 檢查是否在營業時間內
private void checkOpenClose(Order order) throws BusinessException {  1 usage
    int hour = order.getOrderHour();
    if(hour < order.getRestaurantOpenHour() || hour >= order.getRestaurantCloseHour()) {
        throw new BusinessException(
                "目前非 " + order.getRestaurantName() + " 營業時間（營業時間: " +
                        order.getRestaurantOpenHour() + "-" + order.getRestaurantCloseHour() +
                        "，訂單時間: " + hour + "）。"
        );
    }
    logger.info( message: "餐廳 {} 營業中，可接單。", order.getRestaurantName());
}
```

```java
// 檢查餐廳和顧客距離是否合理
private void checkDeliveryDistance(Order order) throws BusinessException {  1 usage
    double distance = Math.sqrt(Math.pow(order.getRestaurantX() - order.getCustomerX(), 2)
            + Math.pow(order.getRestaurantY() - order.getCustomerY(), 2));
    if (distance > MAX_DISTANCE) {
        throw new BusinessException(
                // 距離顯示到小數點後一位就好
                String.format("餐廳 %s 與顧客 %s 距離 %.1f km 超出合理外送距離。",
                        order.getRestaurantName(), order.getCustomerName(), distance)
        );
    }
    String distanceStr = String.format("%.1f", distance);  // 距離顯示到小數點後一位就好
    logger.info( message: "餐廳 {} 與顧客 {} 距離 {} km，在合理範圍內。",
            order.getRestaurantName(), order.getCustomerName(), distanceStr);
}
```

```java
// 餐廳接單
public void acceptOrder(Order order) throws BusinessException {  1 usage
    logger.info( message: "餐廳 {} 正在接受訂單 {}。", order.getRestaurantName(), order.getOrderId());

    checkOpenClose(order);          // 檢查營業時間
    checkDeliveryDistance(order);  // 檢查距離

    if(order.getStatus() != OrderStatus.PENDING) {
        throw new BusinessException("只能接 PENDING 狀態的訂單。");
    }

    order.setStatus(OrderStatus.ACCEPTED);
    // 訂單狀態有更新就在日誌中紀錄
    logger.info( message: "訂單 {} 狀態變更為 {}。", order.getOrderId(), order.getStatus());
}
```

```java
// 外送員取餐
public void pickupOrder(Order order) {  1 usage
    try {
        // 訂單要先被餐廳接受才能被外送員取餐
        if(order.getStatus() != OrderStatus.ACCEPTED) {
            throw new BusinessException("訂單尚未被餐廳接單，無法取餐。");
        }

        order.setStatus(OrderStatus.PICKED_UP);
        // 訂單狀態有更新就在日誌中紀錄
        logger.info( message: "{} 狀態變更為 {}。", order.getOrderId(), order.getStatus());

    } catch (BusinessException e) {
        logger.warn( message: "{}", e.getMessage());
    }
}
```

```java
// 訂單送達
public void deliverOrder(Order order) {  1 usage
    try {
        // 訂單要先被外送員取餐才能送餐
        if(order.getStatus() != OrderStatus.PICKED_UP) {
            throw new BusinessException("訂單尚未被取餐，無法送達。");
        }

        order.setStatus(OrderStatus.DELIVERED);
        // 訂單狀態有更新就在日誌中紀錄
        logger.info( message: "訂單 {} 狀態變更為 {}。", order.getOrderId(), order.getStatus());

    } catch (BusinessException e) {
        logger.warn( message: "{}", e.getMessage());
    }
}
```

public class Main {

```java
public static void main(String[] args) {
    DeliveryService service = new DeliveryService();
    // 建立訂單
    List<Order> orders = new ArrayList<>();
    // 訂單 001 : 正常訂單
    orders.add(new Order(
            orderId: "001", customerName: "Andy", restaurantName: "McDonalds",
            openHour: 1000, closeHour: 2100, orderHour: 1300,
            restaurantX: 2, restaurantY: 3, customerX: 5, customerY: 6));
    // 訂單 002 : 超出距離
    orders.add(new Order(
            orderId: "002", customerName: "Edward", restaurantName: "KFC",
            openHour: 1000, closeHour: 2100, orderHour: 1800,
            restaurantX: 0, restaurantY: 0, customerX: 15, customerY: 15));
    // 訂單 003 : 非營業時間
    orders.add(new Order(
            orderId: "003", customerName: "Walter", restaurantName: "Subway",
            openHour: 1000, closeHour: 2100, orderHour: 2200,
            restaurantX: 1, restaurantY: 1, customerX: 8, customerY: 5));
```

```java
// 依序處理每一筆訂單
for (Order order : orders) {
    try {
        service.acceptOrder(order);  // 餐廳接受訂單
    } catch (BusinessException e) {
        logger.warn( message: "Business Exception: {}", e.getMessage());
    }
    service.pickupOrder(order);   // 外送員取餐
    service.deliverOrder(order);  // 送達
}

logger.info("所有訂單處理完成。");
```

```
10:33:51.267 [main] INFO  org.example.service.DeliveryService - 餐廳 McDonalds 正在接受訂單 001。
10:33:51.269 [main] INFO  org.example.service.DeliveryService - 餐廳 McDonalds 營業中，可接單。
10:33:51.277 [main] INFO  org.example.service.DeliveryService - 餐廳 McDonalds 與顧客 Andy 距離 4.2 km，在合理範圍內。
10:33:51.277 [main] INFO  org.example.service.DeliveryService - 訂單 001 狀態變更為 ACCEPTED。
10:33:51.277 [main] INFO  org.example.service.DeliveryService - 001 狀態變更為 PICKED_UP。
10:33:51.278 [main] INFO  org.example.service.DeliveryService - 訂單 001 狀態變更為 DELIVERED。
10:33:51.278 [main] INFO  org.example.service.DeliveryService - 餐廳 KFC 正在接受訂單 002。
10:33:51.278 [main] INFO  org.example.service.DeliveryService - 餐廳 KFC 營業中，可接單。
10:33:51.278 [main] WARN  Main - Business Exception: 餐廳 KFC 與顧客 Edward 距離 21.2 km 超出合理外送距離。
10:33:51.278 [main] WARN  org.example.service.DeliveryService - 訂單尚未被餐廳接單，無法取餐。
10:33:51.279 [main] WARN  org.example.service.DeliveryService - 訂單尚未被取餐，無法送達。
10:33:51.279 [main] INFO  org.example.service.DeliveryService - 餐廳 Subway 正在接受訂單 003。
10:33:51.287 [main] WARN  Main - Business Exception: 目前非 Subway 營業時間（營業時間: 1000-2100，訂單時間: 2200）。
10:33:51.288 [main] WARN  org.example.service.DeliveryService - 訂單尚未被餐廳接單，無法取餐。
10:33:51.288 [main] WARN  org.example.service.DeliveryService - 訂單尚未被取餐，無法送達。
10:33:51.289 [main] INFO  Main - 所有訂單處理完成。
```

執行結果如上，訂單 001 為正常訂單、訂單 002 超出距離、訂單 003 在非營業時間。

參考資料與使用工具及比例（包含 AI）：

上課講義、AI(大約 80%，功能都是自己想出來的，但有用 AI 輔助設計架構及程式碼)