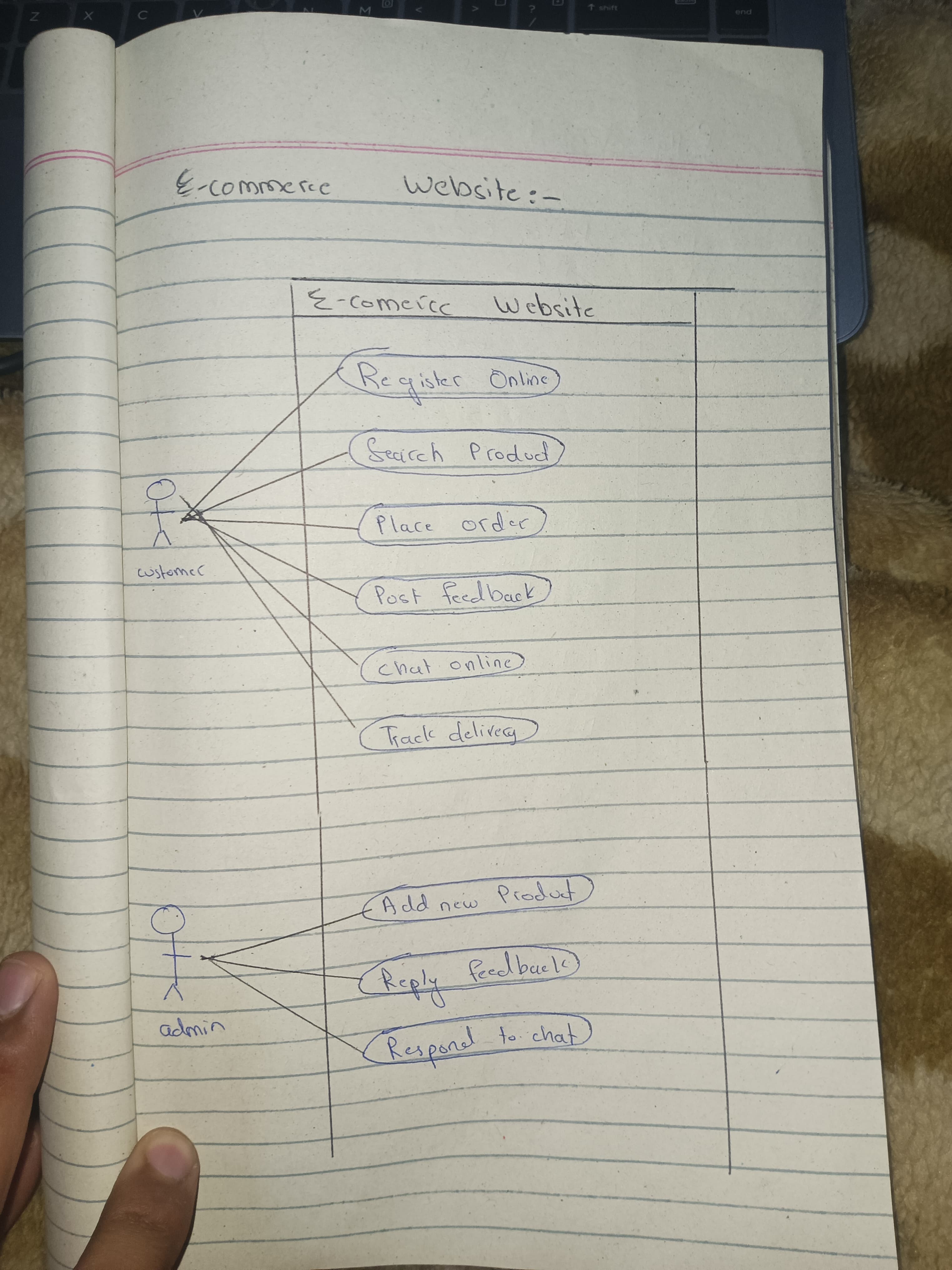
# Project title: (e commerce website)

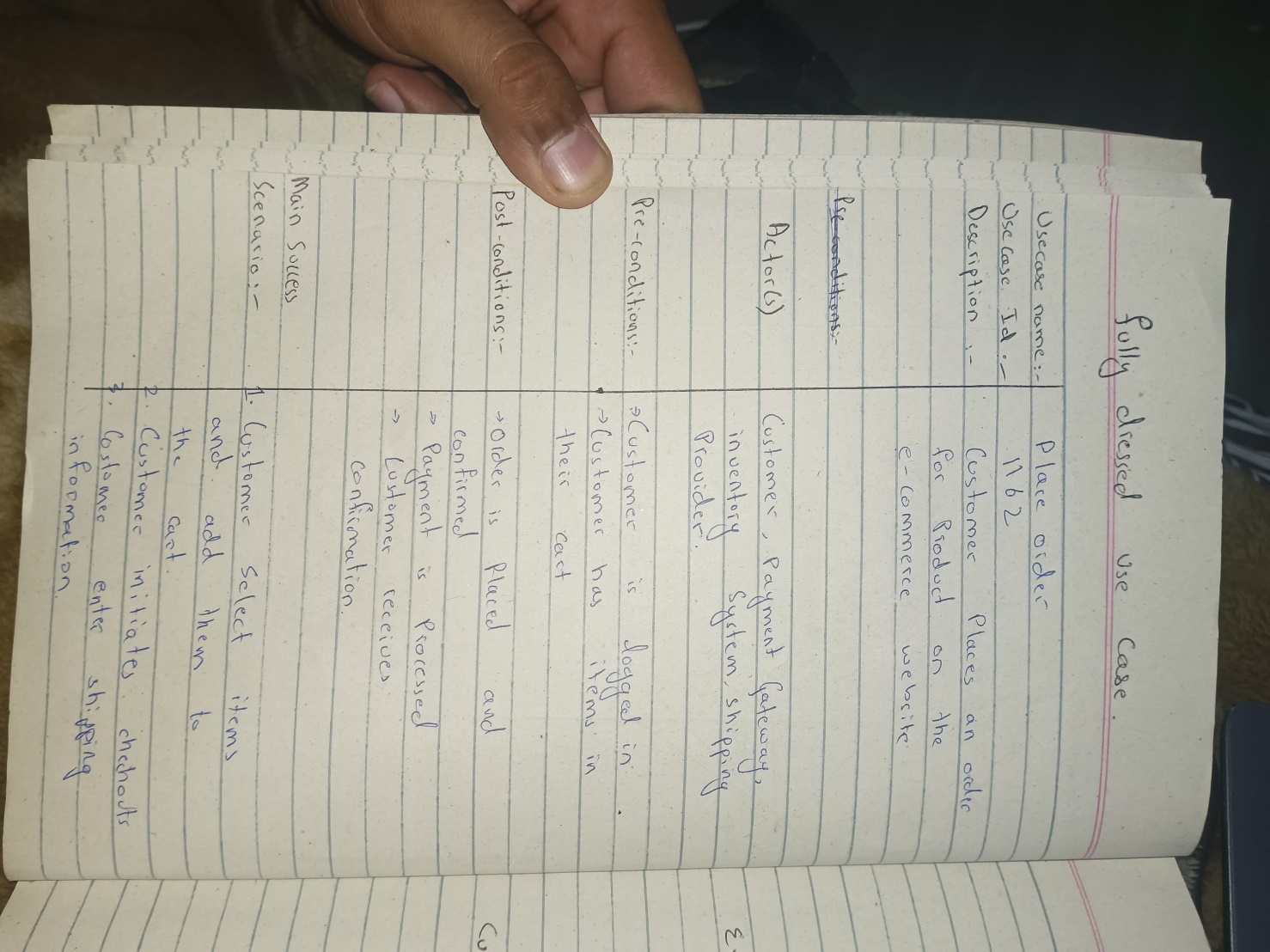
An e-commerce website is an online platform that enables businesses and individuals to buy and sell goods or services over the internet. These websites serve as digital storefronts where customers can browse products, add items to a virtual shopping cart, and complete purchases through secure payment processing. E-commerce websites have transformed the shopping experience by offering convenience, a wide selection of products, and 24/7 availability, allowing customers to shop from anywhere at any time

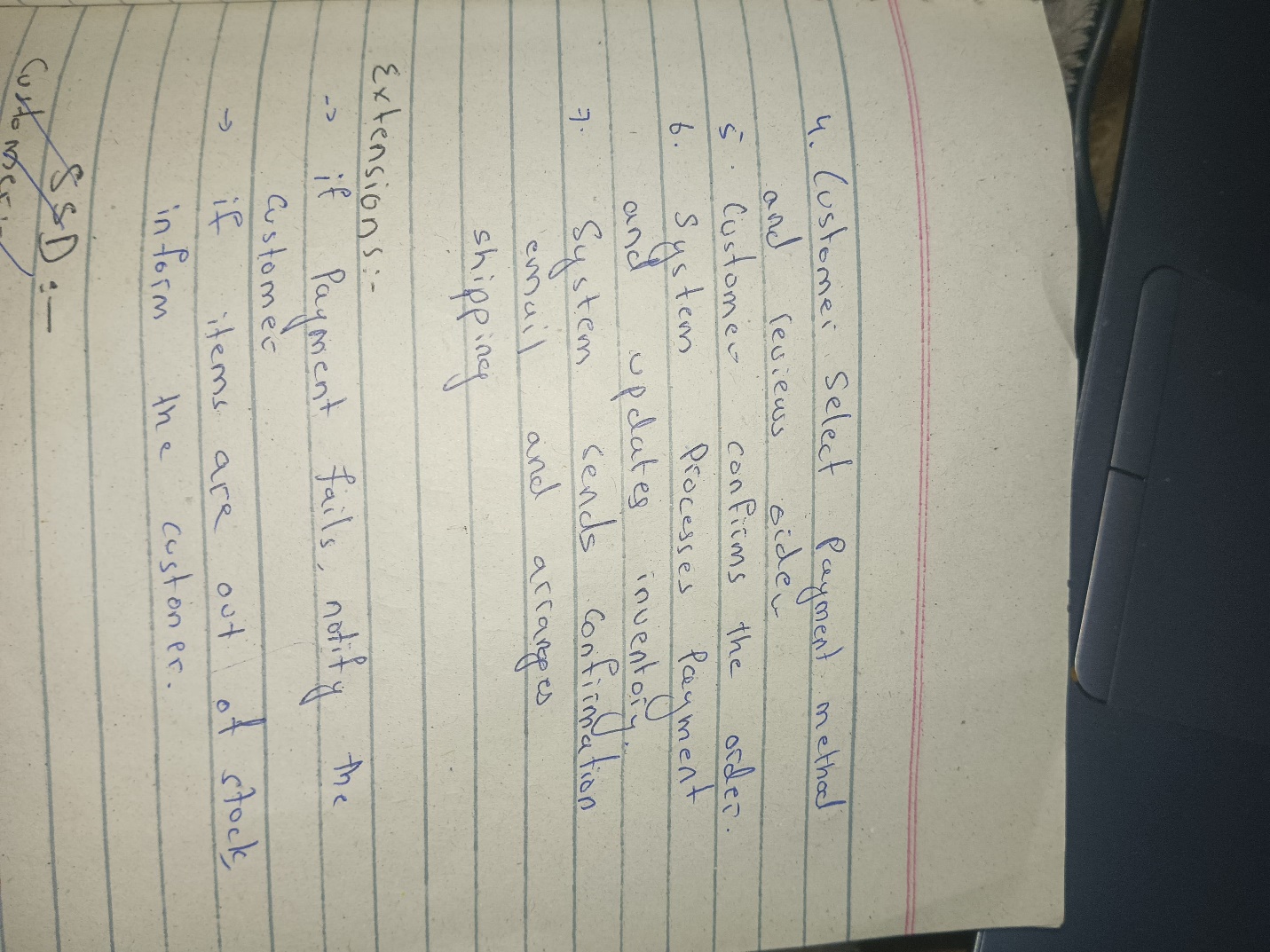
# Use case diagram:



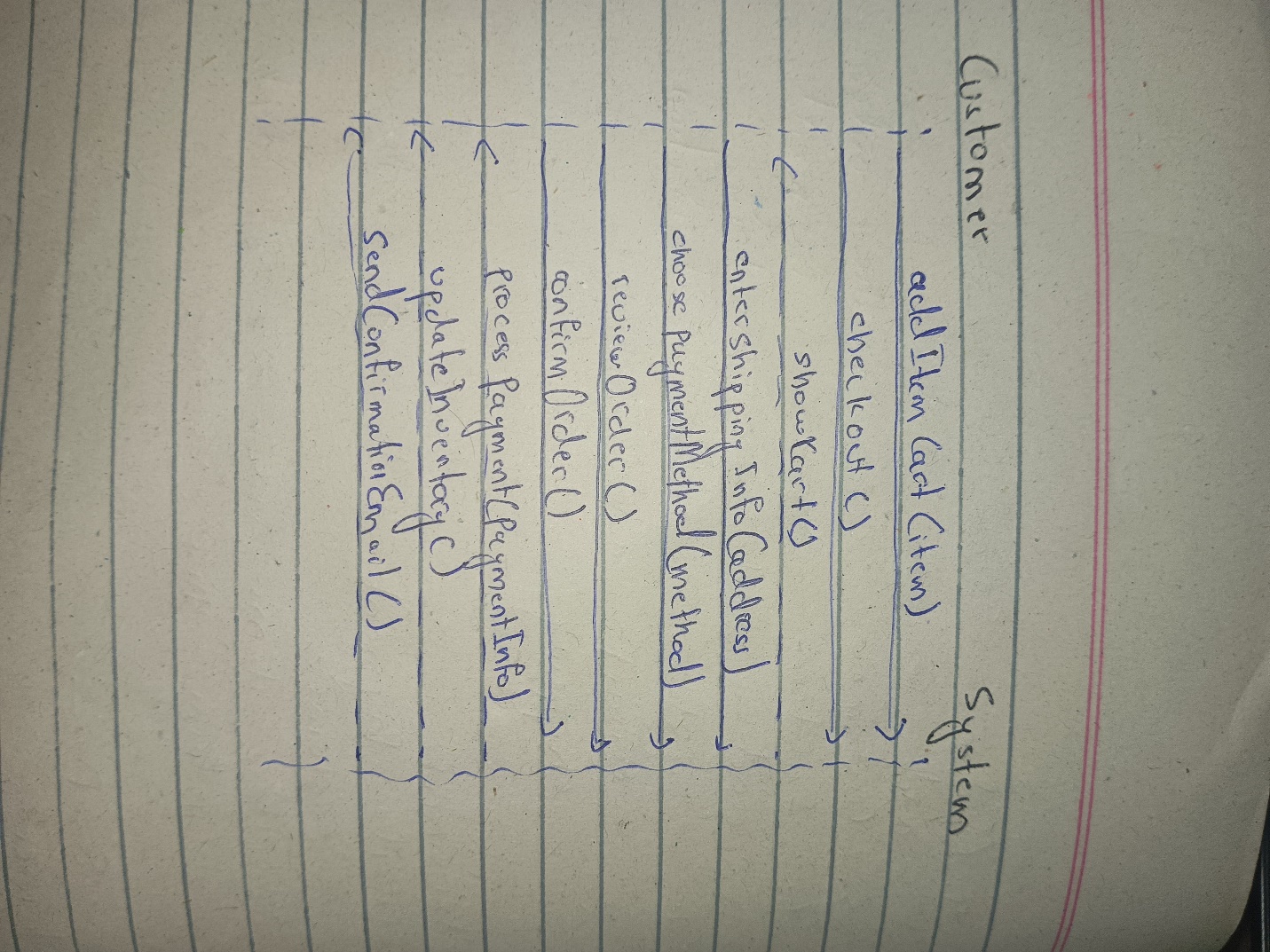
# Fully dresses use case :

The fully dressed use case diagram is for the place order use case.

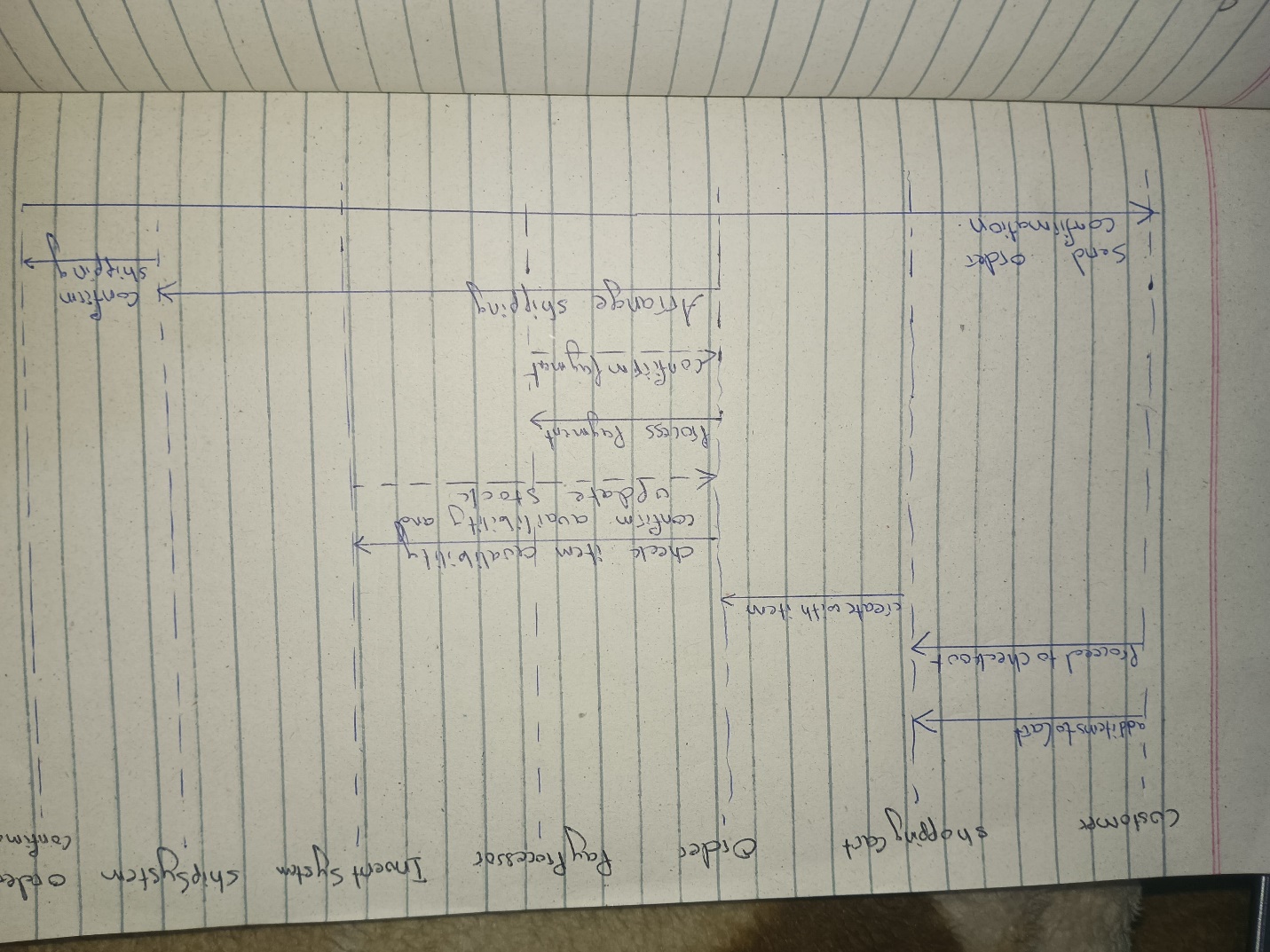




# SYSTEM SEQUENC DIAGRAM:



# COMMUNICATION DIAGRAM:



**GRASP RULES:**

**Controller**:

**Customer** acts as a controller in the system. It is responsible for initiating key tasks like adding items to the cart, checking out, providing shipping information, choosing payment methods, and confirming orders.

In a layered architecture, the OrderService (in our earlier example) would be the *Controller* for handling the business logic related to the order.

**Information Expert**:

**ShoppingCart** is the *Information Expert* for calculating the total cost of items. Since it contains the list of items, it is the most appropriate class to calculate the total price of items in the cart.

**Order** is responsible for confirming itself and managing its own details like the items included, total amount, shipping address, and payment status. It is the class with the most information about the order.

**InventorySystem** is responsible for checking the availability of an item, as it holds the necessary information regarding inventory levels.

**Low Coupling**:

The classes like **PaymentProcessor**, **ShippingSystem**, and **OrderConfirmation** demonstrate low coupling. Each class has a well-defined role and interacts with other parts of the system only through specific, well-defined methods (e.g., processPayment(), arrangeShipping(), sendEmail()).

**Order** and **PaymentProcessor** are loosely coupled; the Order doesn't need to know how payments are processed internally, just that it can call the processPayment() method.

**High Cohesion**:

Each class has a focused responsibility. For example, **ShoppingCart** is only concerned with managing items in the cart, **PaymentProcessor** handles payments, and **ShippingSystem** deals with arranging shipping. This ensures that each class has high cohesion, meaning that all its responsibilities are closely related.

**Creator**:

**Customer** is responsible for creating an instance of **Order**. This is based on the principle that a class should be responsible for creating an instance of another class if it contains or closely uses that class.

Similarly, **ShoppingCart** creates items or manages them since it aggregates multiple items.

# COMMUNICATION DIAGRAM:

