The University of British Columbia

Electrical and Computer Engineering

EECE 571G Blockchain Software Engineering

**Midterm**

**Sample**

Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Time limit: 150 minutes

1. **What you need:**  
   Your laptop.  
   The hardhat and/or foundry.
2. **What you can access during the exam:**  
   You can feel free to access the Internet and any online materials as well as the sample code on the GitHub repository uploaded to UBC Canvas.
3. **What you cannot do:**  
   You cannot share your answer to other students. Being caught collaborating (i.e., sending or receiving answers) will result in falling the exam directly.
4. **What do you need to submit at the end of the midterm:**  
   You need to upload this document with your student name, student number, as well as your smart contract file(s), the test file(s), and the screenshot of the testing results to UBC Canvas.*You are required to work on a decentralized application (Dapp) for the Online Ethazon Shop. The use case of the service is simple. A customer can make an online order and then confirm the order. A customer needs to pay Ethers when puts the order. The paid Ethers are saved in the smart contract. If the customer has not yet confirmed, the Ethers can be refunded to the customer upon the customer’s request.*

***You can assume:*** *1. Every customer can have at most one active order in the system. The customer cannot make another order until she/he has confirmed the shipment or canceled her/his existing order.*

*2. Every order has the same price, for example 0.1 Ethers.*

*3. Customer needs to provide his/her name and mailing address when putting an order.*

*4. The order can always be canceled as long as the customer has not yet confirmed the order.*

*5. You do not need to save the historical records for the customer.*

*6. You do not need to consider the products availability (products are always in-stock).****Hint:*** *You need only one smart contract to serve all customers by using mapping. As you do not need to save the historical records for the customers, you can define a****struct****called EthazonOrder as follows:*

*struct EthazonOrder {*

*bool isValidEthazonOrder;*

*string customerName;*

*string shippingAddress;*

*bool hasConfirmed;*

*}*

*When a customer makes the online order, the isValidEthazonOrder should be set true. When a customer confirms the order, the hasConfirmed should be set true. When a customer cancels the order, the isValidEthazonOrder should be set false and you can reuse the struct again in the next booking. Similarly, if both isValidEthazonOrder and hasConfirmed are true, you can reuse the struct again for his/her next orders. If isValidEthazonOrder is true but hasConfirmed is false, the person cannot make another order.*

***Your need to test:***

1. *A customer cannot make the order if either the customerName or shippingAddress is empty.*
2. *A customer cannot confirm if the order is not valid (when isValidEthazonOrder is false).*
3. *A customer cannot cancel the order if the customer has confirmed the order.*
4. *A customer cannot make another order before she/he has confirmed or canceled the existing order.*
5. *A customer cannot make an order if he/she does not send enough ether to the smart contract.*
6. *If everything is OK, a customer should create an order successfully.*
7. *A customer should receive the money when she/he cancel the order successfully.*
8. *A customer can confirm the order if everything is good.*
9. *The smart contract should have the correct balance of Ethers if an order has been made.*

***Marking Rubrics***

Using the modifier [10]

Function of make the booking [10]

Function of check in [10]

Function of cancel the booking and get the refund [15]

Each test is 5 points [5' x 9 = 45]

Screenshot of the testing results [10]