Note: Please **SUBMIT** each question individually before ending the exam to receive score.
Note: This is a monitored test.

TIME REMAINING
0:36:06

End Exam

eCommerce Store Design Problem

Owing to your impressive technical skills you are hired as the founding CTO of the next big ecommerce startup. Your first job as the CTO is to follow the system design requirements, and build an initial version of the system's functionality based on those requirements, using Object Oriented Design patterns.

System Design Requirements:

The system should allow:

- Having items from many different vendors.
- Having many different items from each vendor and thousands of different items in total.
- Having multiple types of users with the ability to perform different tasks as follows:
  - normal_user: These users can view and buy items.
  - store_manager: These users can do everything a normal_user can do, and can also add new items to an existing vendor.
  - admin: These users can do everything a store manager can do, and can also create new vendors and new users in the system.
- Creating new users.
- Logging in existing users.
- Creating new vendors.
- Adding new items to vendors.
- Viewing items (all items, all items from a vendor, or details of a specific item using a unique identifier).
- Buying items, while keeping track of the available stock for each item.

Following OOD and keeping the above requirements in mind, you should create classes and make use of object-oriented principles as you see fit. Feel free to make assumptions (writing them down as a part of your code), and extend the requirements if you want.

Your program must have some testing code that shows that the above requirements are met. There is no need to save any data to the database or file system, you can hard code test data if needed. Your final code is expected to be well-commented and readable.

COMPLETE

Your Response

Status

Your response has been submitted. You will receive your grade after all steps are complete and your response is fully assessed.

Your response

```
from abc import ABC, abstractmethod


# Custom Exceptions for Error Handling
# These exceptions can be customized in future
```

```python
# As of now they are left empty
class InvalidIDException(Exception):
    pass


class OutOfStockException(Exception):
    pass


# Assuming actual data is stored in a database
# We can create Object Relational Mappings (ORM) to relations in database
class Model(ABC):
    # data is class attribute to store dummy data of model.
    # actual data is assumed to be in a databas
    # Each child class should have their own class attributes of _id and data
    # to prevent conflicts
    data = {}
    # class attribute for generating unique ids
    _id = 0

    @abstractmethod
    def __init__(self):
        pass

    @classmethod
    def get_unique_id(cls) -> int:
        cls._id += 1
        return cls._id

    @classmethod
    def get_all(cls):
        return cls.data

    @classmethod
    def get_by_id(cls, id):
        if id in cls.data:
            return cls.data[id]
        else:
            raise InvalidIDException(f"{id} is invalid")

    # method for querying data by specific field
    @classmethod
    def find_by(cls, **kwargs):
        results = []
        for item in cls.data.values():
            for key, val in kwargs.items():
                if item.__dict__[key] == val:
                    results.append(item)
                    continue
        return results

    @classmethod
    def add_new(cls, item):
        item.id = cls.get_unique_id()
        cls.data[item.id] = item

    # override equals operator
    def __eq__(self, other: object) -> bool:
        return self.id == other.id

    def __repr__(self) -> str:
        return self.__str__()


# A vendor can be a comapany that wants to sell items on our platform
# for example, Coca-Cola, Ikea, MSI etc
class Vendor(Model):
    data = {}
    _id = 0

    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"vendor#{self.id}: {self.name}"


# An item is a commodity.
class Item(Model):
    data = {}
```

```
    _id = 0

    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"item#{self.id}: {self.name}"


# One Vendor can Sell many different Items
# And one Items can be sold by many different Vendors
# This many to many relationship is maintained by VendorItem object
# It also stores the price at which the vendor sells that item
# and its remaining stock
# Note that user can only buy VendorItem, and not the Item entity
class VendorItem(Model):
    data = {}
    _id = 0

    def __init__(self, vendor_id, item_id, price, stock):
        self.vendor = Vendor.get_by_id(vendor_id)
        self.item = Item.get_by_id(item_id)
        self.price = price
        self.stock = stock

    def __str__(self):
        return f"""
                    {self.item} sold by {self.vendor}
                    price: {self.price} stock: {self.stock}"""


# Order object to keep track of bought items
class Order(Model):
    data = {}
    _id = 0

    def __init__(self, user, vendor_item) -> None:
        self.user = user
        self.vendor_item = vendor_item
        self.price = self.vendor_item.price

    def __str__(self) -> str:
        return f"""
                order_id: {self.id}
                buyer: {self.user}
                bought_item: {self.vendor_item}"""


# An Abstract User Class
# It has a log in method
# Each child of User class has specific actions associated with it
# These actions are normally handled in Controllers in MVC design pattern
# For demonstrational purpose, these actions are implemented in the Model class itself
# Each action first asserts if a user is logged in or not
# In case user has not logged in before triggering an action,
# an assertion Exception will be thrown
class User(Model):
    _role = None

    def __init__(self, username, password):
        self.username = username
        self.password = password
        self.logged_in = False

    def log_in(self, username, password):
        if self.username == username and self.password == password:
            self.logged_in = True
            print(f"\n{self.username} logged in successfully. Role: {self._role}")

    def __str__(self) -> str:
        return f"""{self.username}"""


class NormalUser(User):
    data = {}
    _id = 0
    _role = "normal"

    def view_items_by_vendor(self, vendor_name) -> list[VendorItem]:
        vendors = Vendor.find_by(name=vendor_name)
```

```
        return VendorItem.find_by(vendor=vendors[0])

    def view_items_by_name(self, item_name) -> list[VendorItem]:
        assert self.logged_in
        results = []
        items = Item.find_by(name=item_name)
        for item in items:
            vendor_items = VendorItem.find_by(item=item)
            results.append(vendor_items)
        return results

    def view_all_items(self) -> list[VendorItem]:
        assert self.logged_in
        return VendorItem.get_all()

    def view_item_by_id(self, item_id) -> VendorItem:
        assert self.logged_in
        return VendorItem.get_by_id(item_id)

    def buy_item(self, vendor_item: VendorItem):
        assert self.logged_in
        if vendor_item.stock > 0:
            vendor_item.stock -= 1
            new_order = Order(self, vendor_item)
            Order.add_new(new_order)
            return new_order
        else:
            raise OutOfStockException("item out of stock")


class StoreManager(NormalUser):
    data = {}
    _id = 0
    _role = "manager"

    def add_item(self, item):
        assert self.logged_in
        Item.add_new(item)

    def add_vendor_item(self, vendor_id, item_id, price, stock):
        assert self.logged_in
        vendor_item = VendorItem(vendor_id, item_id, price, stock)
        VendorItem.add_new(vendor_item)


class Admin(StoreManager):
    data = {}
    _id = 0
    _role = "admin"

    def add_vendor(self, vendor):
        assert self.logged_in
        Vendor.add_new(vendor)

    def add_normal_user(self, normal_user):
        assert self.logged_in
        NormalUser.add_new(normal_user)

    def add_manager(self, manager):
        assert self.logged_in
        StoreManager.add_new(manager)


if __name__ == "__main__":
    # Add admin to database
    admin = Admin("admin", "admin")
    Admin.add_new(admin)

    # Admin Must be logged in before performing operations
    admin.log_in("admin", "admin")

    # Admin can add users to platform
    admin.add_normal_user(NormalUser("hammad", "123"))
    print(f"\nadmin added {NormalUser.get_by_id(1)} to platform")
    # Admin can add store manager to platform
    admin.add_manager(StoreManager("ali", "123"))
    print(f"\nadmin added {StoreManager.get_by_id(1)} to platform")

    # Admin can add items to platform
    admin.add_item(Item("Chair"))
    admin.add_item(Item("RTX 3090"))
```

```
admin.add_item(item("RTX 3090"))
print(f"\nadmin added following items:\n{Item.get_all().values()}")

# Admin can add vendors to platform
admin.add_vendor(Vendor("Ikea"))
admin.add_vendor(Vendor("MSI"))
print(f"\nadmin added following vendors:\n{Vendor.get_all().values()}")

# Manager can add new items to vendor's store
results = StoreManager.find_by(username="ali")
manager: StoreManager = results[0]
manager.log_in("ali", "123")
# adding Chair to Ikea (ID of Ikea: 1, ID of Chair: 1)
manager.add_vendor_item(1, 1, 1000, 10)
# adding RTX to Ikea
manager.add_vendor_item(1, 2, 500000, 1)
# adding RTX to MSI
manager.add_vendor_item(2, 2, 40000, 1)
print(
    f"\n{manager.username} added following VendorItems:\n{VendorItem.get_all().values()}"
)

# User can browse and buy items
results = NormalUser.find_by(username="hammad")
user: NormalUser = results[0]
user.log_in("hammad", "123")

# Browsing items by Ikea
print(f"\n{user.username} is browsing items sold by Ikea")
ikea_items = user.view_items_by_vendor("Ikea")
print(ikea_items)

# Browsing all items in platform
print(f"\n{user.username} is browsing all the vendor items in the platform")
all_items = user.view_all_items()
print(all_items.values())

# Browsing item by id
id = 2
try:
    print(f"\n{user.username} searched for vendor item with id {id}")
    specific_vendor_item = user.view_item_by_id(id)
    print(specific_vendor_item)

    # User can buy an item
    print(f"\n{user.username} is buying {specific_vendor_item}")
    user.buy_item(specific_vendor_item)
    print("\nAll Orders placed in Platform")
    print(Order.get_all().values())
except InvalidIDException:
    print("vendor item with given id does not exist")
except OutOfStockException:
    print(f"the requested item is out of stock")
```

Code Execution Result

```
admin logged in successfully. Role: admin

admin added hammad to platform

admin added ali to platform

admin added following items:
dict_values([item#1: Chair, item#2: RTX 3090])
```