به نام خدا

WAREHOUSE MANAGEMENT CLI

اعضای گروه:

- امیرعلی شهسواری
- شهروز خلیلی
- امیرعلی زلیخانی

لینک همگیت: https://hamgit.ir/amrlishah.۲۰۰۶/oop

عناوین

ماژول مدل[1]

کلاس آیتم[2]

در این بخش، کلاس «کالا» پیاده‌سازی شده که وظیفه‌ی نگهداری اطلاعات پایه‌ای هر محصول در انبار را دارد. ویژگی‌هایی مانند نام کالا، شناسه یکتا، تعداد موجودی، ابعاد فیزیکی (طول، عرض، ارتفاع) و نیازمندی‌های نگهداری کالا در این کلاس تعریف شده‌اند. در سازنده‌ی کلاس، امکان مقداردهی اولیه به این ویژگی‌ها فراهم شده و برای مواردی که ورودی داده نمی‌شود، مقادیر پیش‌فرض در نظر گرفته شده تا فرآیند ایجاد کالا در سیستم ساده‌تر باشد. متدی برای به‌روزرسانی تعداد موجودی کالا طراحی شده که هم افزایش و هم کاهش تعداد را پشتیبانی می‌کند، اما در صورت منفی شدن موجودی، پیام خطا نمایش داده می‌شود تا از بروز خطاهای منطقی جلوگیری شود.همچنین متدی جهت نمایش اطلاعات کامل کالا پیاده‌سازی شده که مشخصات کالا را به‌صورت منظم و قابل خواندن بازمی‌گرداند. این متد می‌تواند در نمایش‌های متنی، گزارش‌گیری و بررسی‌های دستی بسیار کاربردی باشد.طراحی کلاس به‌گونه‌ای انجام شده که امکان توسعه و افزودن ویژگی‌های بیشتر در آینده (مانند قیمت یا برند) وجود داشته باشد. همچنین اصول کپسوله‌سازی رعایت شده و کنترل خطاها در جای مناسب انجام شده است.

```
class Item:
    def __init__(self, name, SKU, quantity=0, dimensions=(0, 0, 0),
storage_requirements=None):
        self.name = name
        self.SKU = SKU
        self.quantity = int(quantity)
        self.dimensions = dimensions
        self.storage_requirements = storage_requirements if storage_requirements else
{}

    def update_quantity(self, quantity):
        new_quantity = self.quantity + int(quantity)
        if new_quantity < 0:
            raise ValueError("quantity cannot be negative")
```

[1] Model Module
[2] Class Item

```
        self.quantity = new_quantity

    def display_info(self):
        return (
            f"Item Info:\n"
            f"Name: {self.name}\n"
            f"SKU: {self.SKU}\n"
            f"quantity: {self.quantity}\n"
            f"Dimensions: {self.dimensions}\n"
            f"Storage Requirements: {self.storage_requirements}"
        )
```

کلاس جعبه ذخیره‌سازی[1]

کلاس «جایگاه ذخیره‌سازی» به‌عنوان واحدی از فضای فیزیکی انبار طراحی شده و وظیفه‌ی نگهداری کالاها را با رعایت محدودیت‌های محیطی بر عهده دارد. هر جایگاه دارای شناسه، ظرفیت کلی، بار فعلی، محدودیت‌های محیطی و فهرستی از کالاهای موجود است.در سازنده‌ی کلاس، مقادیر اولیه مانند شناسه، ظرفیت، بار فعلی و محدودیت‌ها تعیین می‌شوند و فهرست کالاها به‌صورت پیش‌فرض تهی است.متدی برای افزودن کالا به جایگاه پیاده‌سازی شده که ابتدا نوع شیء را بررسی می‌کند، سپس نیازمندی‌های نگهداری کالا را با محدودیت‌های محیطی جایگاه مقایسه کرده و در صورت عدم تطابق، پیام خطا صادر می‌شود. در صورتی که فضای کافی وجود داشته باشد، کالا به جایگاه افزوده شده یا مقدار آن به‌روزرسانی می‌شود و بار فعلی جایگاه نیز افزایش می‌یابد.همچنین متدی برای حذف کالا از جایگاه در نظر گرفته شده که در صورت موجود بودن کالا و کافی بودن تعداد آن، عملیات حذف را انجام می‌دهد و در صورت صفر شدن تعداد، آن کالا از فهرست حذف می‌شود. برای بررسی ظرفیت آزاد، متدی برای محاسبه و بازگرداندن فضای باقی‌مانده در جایگاه تعریف شده است.در پایان، متدی برای نمایش اطلاعات کامل جایگاه از جمله مشخصات کلی و فهرست کالاهای موجود فراهم شده که می‌تواند در نمایش‌های متنی و گزارش‌گیری مورد استفاده قرار گیرد.

```
class StorageBin:
    def __init__(self, bin_id, capacity, current_load=0, constraints=None):
        self.bin_id = bin_id
        self.capacity = int(capacity)
        self.current_load = int(current_load)
        self.constraints = constraints if constraints else {}
```

[1] Class Storage Bin

```
        self.items = {}

    def add_item(self, item):
        if not isinstance(item, Item):
            raise TypeError("Only instances of Item can be added to the bin.")

        # Check storage requirements
        for req, value in item.storage_requirements.items():
            if req in self.constraints and self.constraints[req] != value:
                raise ValueError(f"Item does not meet bin's {req} requirement")

        required_space = item.quantity
        if self.current_load + required_space > self.capacity:
            raise ValueError("Not enough space in the bin")

        if item.SKU in self.items:
            self.items[item.SKU]['quantity'] += item.quantity
        else:
            self.items[item.SKU] = {
                'quantity': item.quantity,
                'name': item.name,
                'dimensions': item.dimensions
            }

        self.current_load += required_space

    def remove_item(self, item):
        if not isinstance(item, Item):
            raise TypeError("Only instances of Item can be removed from the bin.")

        if item.SKU not in self.items:
            raise ValueError("Item not found in the bin")

        if self.items[item.SKU]['quantity'] < item.quantity:
            raise ValueError("Not enough items in the bin to remove")

        self.items[item.SKU]['quantity'] -= item.quantity
        self.current_load -= item.quantity

        if self.items[item.SKU]['quantity'] == 0:
            del self.items[item.SKU]

    def available_space(self):
        return self.capacity - self.current_load

    def display_bin_info(self):
        info = (
            f"Bin Info:\n"
```

```
            f"Bin ID: {self.bin_id}\n"
            f"Capacity: {self.capacity}\n"
            f"Current Load: {self.current_load}\n"
            f"Available Space: {self.available_space()}\n"
            f"Constraints: {self.constraints}\n"
            f"Items in Bin:\n"
        )

        for sku, details in self.items.items():
            info += (
                f"  - SKU: {sku}, Name: {details['name']}, "
                f"quantity: {details['quantity']}, Dimensions:
{details['dimensions']}\n"
            )

        return info
```

کلاس سفارش[1]

کلاس «سفارش» برای ثبت و پردازش درخواست‌های خروج کالا از انبار طراحی شده است. هر سفارش شامل شناسه یکتا، فهرست کالاهای درخواستی، تاریخ ثبت سفارش و وضعیت جاری آن است.در زمان ایجاد شیء، شناسه سفارش و فهرست کالاها به عنوان ورودی دریافت می‌شود. فهرست کالاها شامل یک نگاشت بین شناسه‌های کالا و تعداد مورد نیاز از هرکدام است. تاریخ ثبت سفارش به صورت خودکار بر اساس زمان جاری سیستم تعیین می‌شود و وضعیت اولیه‌ی سفارش به‌صورت «در حال پردازش» در نظر گرفته می‌شود.متدی برای پردازش سفارش در نظر گرفته شده که ابتدا بررسی می‌کند آیا کالاهای درخواستی به اندازه کافی در انبار موجود هستند یا نه. در صورت وجود موجودی کافی، کالاها از جایگاه‌های ذخیره‌سازی حذف و موجودی کل کاهش داده می‌شود. در پایان، وضعیت سفارش به «تکمیل شده» تغییر داده می‌شود. در صورت بروز هرگونه خطا در این فرآیند، پیام مناسبی تولید می‌شود.برای تغییر وضعیت سفارش، متدی با بررسی مقادیر مجاز وضعیت‌ها طراحی شده تا از وارد کردن مقادیر نادرست جلوگیری شود.در نهایت، متدی برای نمایش اطلاعات کامل سفارش طراحی شده که شامل شناسه، تاریخ، وضعیت جاری و فهرست کالاهای درخواست شده است. این اطلاعات می‌تواند در گزارش‌گیری یا بررسی وضعیت سفارشات مورد استفاده قرار گیرد.ساختار کلاس به‌گونه‌ای طراحی شده که کنترل خطا به‌درستی انجام شده و تعامل کامل با سیستم انبارداری فراهم باشد.

[1] Class Order

```python
from datetime import datetime

class Order:
    def __init__(self, order_id: str, items: dict):
        self.order_id = order_id
        self.items = items  # Dictionary with SKU as key and quantity as value
        self.order_date = datetime.now()
        self.order_status = "Pending"  # Default status

    def process_order(self, warehouse):
        """
        Processes the order by checking inventory and updating status
        Returns True if successful, False otherwise
        """
        # Check if all items are available in inventory
        for sku, quantity in self.items.items():
            if sku not in warehouse.inventory or warehouse.inventory[sku].quantity <
quantity:
                raise ValueError(f"Not enough inventory for item with SKU {sku}")

        # If all items are available, process the order
        try:
            for sku, quantity in self.items.items():
                # Create a temporary item for removal
                temp_item = Item("", sku, quantity)
                warehouse.remove_item_from_bins(temp_item)
                warehouse.inventory[sku].update_quantity(-quantity)

            self.update_status("Fulfilled")
            return True
        except Exception as e:
            raise RuntimeError(f"Error processing order: {str(e)}")

    def update_status(self, new_status: str):
        """
        Updates the order status
        Valid statuses: Pending, Fulfilled
        """
        valid_statuses = ["Pending", "Fulfilled"]
        if new_status not in valid_statuses:
            raise ValueError(f"Invalid status. Must be one of: {valid_statuses}")
        self.order_status = new_status

    def display_order_info(self):
        """
        Returns formatted string with order details
        """
```

```
        items_info = "\n".join([f"  - SKU: {sku}, quantity: {qty}" for sku, qty in
self.items.items()])
        return (
            f"Order Info:\n"
            f"Order ID: {self.order_id}\n"
            f"Order Date: {self.order_date}\n"
            f"Status: {self.order_status}\n"
            f"Items:\n{items_info}"
        )
```

کلاس تامین کننده[1]

کلاس «تأمین‌کننده» برای نگهداری اطلاعات مرتبط با تأمین‌کنندگان کالا طراحی شده است. هر شیء از این کلاس شامل نام تأمین‌کننده، اطلاعات تماس و فهرستی از کالاهایی است که توسط او تأمین می‌شود. در زمان ایجاد شیء، در صورت عدم ارائه‌ی فهرست کالاها، این لیست به‌صورت پیش‌فرض تهی در نظر گرفته می‌شود. این موضوع موجب انعطاف‌پذیری بیشتر در هنگام افزودن تأمین‌کنندگان جدید می‌گردد.متدی برای ثبت سفارش به تأمین‌کننده در نظر گرفته شده که به‌صورت نمادین عملیات سفارش را چاپ می‌کند و نشان‌دهنده تعامل با زنجیره تأمین است.همچنین متدی برای به‌روزرسانی اطلاعات تماس تأمین‌کننده پیاده‌سازی شده که امکان ویرایش اطلاعات را در زمان‌های بعد فراهم می‌سازد.برای مشاهده اطلاعات تأمین‌کننده، متدی طراحی شده که خروجی‌ای شامل نام، اطلاعات تماس و فهرست کالاهای تأمین‌شده تولید می‌کند و در نمایش گزارش‌ها یا بررسی داده‌ها قابل استفاده است.طراحی این کلاس ساده اما کاربردی است و ارتباط آن با دیگر اجزای سیستم (مانند سفارشات و انبارداری) به خوبی قابل توسعه خواهد بود.

```
class Supplier:
    def __init__(self, supplier_name, contact_details, items_supplied=None):
        self.supplier_name = supplier_name
        self.contact_details = contact_details
        self.items_supplied = items_supplied if items_supplied is not None else []

    def place_order(self):
        print(f"Order placed with supplier: {self.supplier_name}")

    def update_contact(self, new_contact_details):
        self.contact_details = new_contact_details
        print(f"Contact details updated for supplier: {self.supplier_name}")
```

[1] Class Supplier

```
    def display_supplier_info(self):
        return (
            f"Supplier Info:\n"
            f"Name: {self.supplier_name}\n"
            f"Contact Details: {self.contact_details}\n"
            f"Items Supplied: {', '.join(self.items_supplied) if self.items_supplied
else 'None'}"
        )
```

کلاس مدیریت انبار[1]

کلاس «مدیریت انبار» به‌عنوان هسته‌ی اصلی سیستم طراحی شده و وظیفه‌ی هماهنگی بین تمام اجزای سیستم از جمله کالاها، جایگاه‌های ذخیره‌سازی، سفارش‌ها و تأمین‌کنندگان را بر عهده دارد.

در سازنده‌ی کلاس، چهار بخش اصلی به‌صورت پیش‌فرض تعریف می‌شود:

- فهرست جایگاه‌ها به‌صورت یک نگاشت،
- موجودی انبار به‌صورت یک نگاشت از شناسه کالا به کالا،
- فهرست سفارش‌ها به‌صورت یک لیست،
- فهرست تأمین‌کنندگان به‌صورت یک لیست.

برای افزودن جایگاه و تأمین‌کننده، متدهایی طراحی شده‌اند که علاوه بر بررسی نوع داده ورودی، از ثبت تکراری نیز جلوگیری می‌کنند و در صورت موفقیت، پیام مناسبی چاپ می‌شود.متدی برای دریافت کالا از تأمین‌کننده پیاده‌سازی شده که کالا را به جایگاه مورد نظر اضافه کرده و موجودی کلی انبار را به‌روزرسانی می‌کند. در صورت بروز خطا (مانند محدودیت فضا یا ناسازگاری شرایط نگهداری)، پیام هشدار صادر می‌شود.پردازش سفارش از طریق متدی انجام می‌شود که ابتدا بررسی می‌کند آیا همه کالاهای سفارش به اندازه کافی موجود هستند یا نه. در صورت تأیید، کالاها از جایگاه‌ها حذف شده، موجودی کاهش یافته، و وضعیت سفارش به حالت «تکمیل شده» تغییر می‌یابد.برای گزارش‌گیری، متدی طراحی شده که دو بخش اصلی را پوشش می‌دهد: خلاصه‌ای از موجودی کالاها در انبار و وضعیت کلی جایگاه‌های ذخیره‌سازی از نظر بار فعلی و فضای آزاد.متد دیگری برای جست‌وجوی

[1] Class Warehouse

سریع کالا در موجودی طراحی شده که با استفاده از شناسه کالا، شیء مربوطه را بازمی‌گرداند یا در صورت عدم وجود، مقدار تهی ارائه می‌دهد.در نهایت، متدی جهت حذف مقدار مشخصی از یک کالا از جایگاه‌های مختلف تعریف شده که به‌صورت تدریجی و از جایگاه‌های دارای موجودی، عملیات حذف را انجام می‌دهد. اگر امکان حذف تمام مقدار خواسته‌شده وجود نداشته باشد، پیام خطا صادر می‌شود.

```
class Warehouse:
    def __init__(self):
        self.storage_bins = {}
        self.inventory = {}
        self.orders = []
        self.suppliers = []

    def add_storage_bin(self, storage_bin):
        if not isinstance(storage_bin, StorageBin):
            raise TypeError("Only instances of StorageBin can be added to warehouse.")
        if storage_bin.bin_id in self.storage_bins:
            raise ValueError(f"Storage bin with ID {storage_bin.bin_id} already
exists.")
        self.storage_bins[storage_bin.bin_id] = storage_bin
        print(f"Storage bin {storage_bin.bin_id} added successfully.")

    def add_supplier(self, supplier):
        if not isinstance(supplier, Supplier):
            raise TypeError("Only instances of Supplier can be added to warehouse.")
        self.suppliers.append(supplier)
        print(f"Supplier {supplier.supplier_name} added successfully.")

    def receive_shipment(self, item, bin_id):
        if not isinstance(item, Item):
            raise TypeError("Only instances of Item can be received.")

        if bin_id not in self.storage_bins:
            raise ValueError(f"Storage bin {bin_id} not found.")

        storage_bin = self.storage_bins[bin_id]

        try:
            storage_bin.add_item(item)

            # Update inventory
            if item.SKU in self.inventory:
                self.inventory[item.SKU].update_quantity(item.quantity)
            else:
                self.inventory[item.SKU] = item
```

```
            print(f"Shipment received and added to bin {bin_id} successfully.")
        except Exception as e:
            print(f"Error receiving shipment: {str(e)}")

    def fulfill_order(self, order):
        if not isinstance(order, Order):
            raise TypeError("Only instances of Order can be processed.")

        # Check inventory for all items
        for sku, quantity in order.items.items():
            if sku not in self.inventory or self.inventory[sku].quantity < quantity:
                print(f"Not enough inventory for item with SKU {sku}")
                return False

        # Process the order
        try:
            for sku, quantity in order.items.items():
                item_to_remove = Item("", sku, quantity)
                self.remove_item_from_bins(item_to_remove)
                self.inventory[sku].update_quantity(-quantity)

            order.update_status("Fulfilled")
            self.orders.append(order)
            print("Order fulfilled successfully.")
            return True
        except Exception as e:
            print(f"Error fulfilling order: {str(e)}")
            return False

    def generate_inventory_report(self):
        report = "Inventory Report:\n\n"

        # Inventory summary
        report += "Inventory Summary:\n"
        for sku, item in self.inventory.items():
            report += f"  - SKU: {sku}, Name: {item.name}, quantity: {item.quantity}\n"
        # Storage bins status
        report += "\nStorage Bins Status:\n"
        for bin_id, storage_bin in self.storage_bins.items():
            report += f"  - Bin ID: {bin_id}, Capacity: {storage_bin.capacity}, "
            report += f"Current Load: {storage_bin.current_load}, "
            report += f"Available Space: {storage_bin.available_space()}\n"

        return report

    def find_item(self, sku):
        return self.inventory.get(sku, None)
```

```
    def remove_item_from_bins(self, item):
        if not isinstance(item, Item):
            raise TypeError("Only instances of Item can be removed.")

        remaining_quantity = item.quantity

        for bin_id, storage_bin in self.storage_bins.items():
            if remaining_quantity <= 0:
                break

            if item.SKU in storage_bin.items:
                available_quantity = storage_bin.items[item.SKU]['quantity']
                quantity_to_remove = min(available_quantity, remaining_quantity)

                temp_item = Item("", item.SKU, quantity_to_remove)
                storage_bin.remove_item(temp_item)
                remaining_quantity -= quantity_to_remove

        if remaining_quantity > 0:
            raise ValueError(f"Could not remove all items. {remaining_quantity} items
not found in bins.")
```

ماژول اصلی[1]

تابع دریافت محموله[2]

در این بخش از کد در ابتدا تابعی تغریف کردیم که اطلاعات اولیه‌ی کالا شامل نام، شناسه و تعداد را از کاربر دریافت و تعداد ورودی را به عدد صحیح تبدیل می‌کند. در صورت صحیح بودن ورودی، شی‌ای از کالا ساخته و سپس بر اساس شناسه جایگاه مورد نظر، از طریق متد مربوطه در سیستم انبار، عملیات دریافت کالا انجام می‌شود. در پایان، پیام موفقیت یا خطا به کاربر نمایش داده می‌شود.

```
import datetime as dt
import models

def receive_shipment_cli():
    print("Welcome to the Shipment Management System")

    item_name = input("Enter the item name: ")
    item_id = input("Enter the item ID: ")
    item_quantity = input("Enter the item quantity: ")
    try:
        item_quantity = int(item_quantity)
    except ValueError:
        print("Invalid quantity. Please enter a number.")
        return

    dimensions = (0, 0, 0)
    storage_requirements = {}
    unique_object_name = f"{item_name}_{item_id}"

    item_object = models.Item(item_name, item_id, item_quantity, dimensions,
storage_requirements)

    if 'items_dict' not in globals():
        global items_dict
        items_dict = {}

    items_dict[unique_object_name] = item_object
```

[1] Main Module
[2] Def Receive Shipment Cli

```
    bin_id = input("Enter the storage bin ID to receive the shipment: ")

    try:
        models.warehouse.receive_shipment(item_object, bin_id)
        print(f"Shipment for {item_name} successfully received and stored in bin
{bin_id}.")
    except Exception as e:
        print(f"Error receiving shipment: {str(e)}")
```

تابع پردازش سفارش[1]

در ادامه در تابع مربوط به پردازش سفارش از طریق رابط کاربری ، ابتدا شناسه سفارش دریافت می‌شود، سپس کاربر می‌تواند چندین کالا و تعداد هرکدام را وارد کند. پس از پایان ورود، در صورت وجود کالا، یک شیء از سفارش ساخته شده و به سیستم مدیریت انبار ارسال می‌شود تا پردازش شود. در صورت موفقیت، اطلاعات سفارش نمایش داده می‌شود، و در صورت خطا، پیام مناسب به کاربر اعلام می‌گردد.

```
def process_order_cli():
    print("Welcome to the Order Processing System")

    order_id = input("Enter the order ID: ")
    order_items = {}

    while True:
        item_id = input("Enter item ID (or type 'done' to finish): ")
        if item_id.lower() == 'done':
            break
        quantity = input(f"Enter quantity for item {item_id}: ")
        try:
            quantity = int(quantity)
        except ValueError:
            print("Invalid quantity. Please enter a number.")
            continue

        order_items[item_id] = quantity

    if not order_items:
        print("No items in the order. Order processing cancelled.")
        return

    order = models.Order(order_id, order_items)
```

[1] Def Process Order Cli

```
    try:
        result = models.warehouse.fulfill_order(order)
        print("Order processed successfully.")
        print(f"Order ID: {order.order_id}")
        print("Items:")
        for item_id, quantity in order.items.items():
            print(f"  - {item_id}: {quantity}")
    except Exception as e:
        print("Order processed unsuccessfully.")
        print(f"Order processing failed: {str(e)}")
```

تابع گزارش گیری[1]

تابع گزارش گیری از موجودی انبار وظیفه دارد با فراخوانی متد مخصوص از کلاس مدیریت انبار، گزارشی جامع از وضعیت فعلی انبار را نمایش دهد. این گزارش شامل موجودی کالاها (بر اساس شناسه و تعداد) و همچنین وضعیت جایگاه‌های ذخیره‌سازی (از نظر ظرفیت، بار فعلی و فضای آزاد) است. در صورت بروز خطا هنگام تولید گزارش، پیامی به کاربر نمایش داده می‌شود تا از مشکل مطلع شود.

در تابع افزودن جایگاه جدید انبار ، کاربر می‌تواند یک جایگاه جدید به سیستم انبار اضافه کند. ابتدا شناسه و ظرفیت جایگاه از کاربر گرفته می‌شود و در صورت معتبر بودن مقدار ظرفیت، وارد مرحله بعد می‌شود. سپس کاربر می‌تواند محدودیت‌های محیطی مانند دما یا رطوبت را وارد کند (به صورت کلید–مقدار). این اطلاعات به‌صورت یک نگاشت ذخیره می‌شوند. در نهایت، جایگاه با اطلاعات واردشده ساخته شده و به سیستم انبار اضافه می‌شود. در صورت بروز هر گونه خطا در فرآیند اضافه‌سازی (مثلاً تکراری بودن شناسه یا ناهماهنگی نوع داده‌ها)، پیام هشدار چاپ می‌شود.

```
def generate_report_cli():
    print("Generating warehouse inventory report...\n")

    try:
        report = models.warehouse.generate_inventory_report()
        print(report)
    except Exception as e:
        print("Error generating report.")
        print(f"Details: {str(e)}")

def add_storage_bin_cli():
```

[1] Def Generate Report Cli

```
    print("Add a New Storage Bin")

    bin_id = input("Enter bin ID: ")
    while True:
        capacity_input = input("Enter bin capacity: ")
        try:
            capacity = int(capacity_input)
            break  # اگر ظرفیت صحیح بود، از حلقه خارج می‌شود
        except ValueError:
            print("Invalid capacity. Please enter a valid number.")
            return  # برگشت به منو در صورت وارد کردن مقدار غیر عددی

    cons_input = input("Enter environmental requirements (key1=value1,key2=value2,...):
")
    constraints = {}

    try:
        for pair in cons_input.split(','):
            if '=' in pair:
                key, value = pair.split('=')
                constraints[key.strip()] = value.strip()
    except Exception:
        print("Invalid format for environmental requirements. Skipping them.")
        constraints = {}

    new_bin = models.StorageBin(bin_id, capacity, constraints=constraints)

    try:
        models.warehouse.add_storage_bin(new_bin)
    except Exception as e:
        print(f"Error adding storage bin: {str(e)}")
```

تابع افزودن تامین کننده[1]

تابع افزودن تامین کننده جدید برای ثبت یک تأمین‌کننده جدید در سیستم طراحی شده است. ابتدا نام تأمین‌کننده از کاربر دریافت می‌شود. سپس اطلاعات تماس مانند تلفن، ایمیل یا سایر مشخصات، به‌صورت یک رشته‌ی جداشده با ویرگول از کاربر گرفته شده و به یک نگاشت کلید–مقدار تبدیل می‌شود.در صورت ورود قالب نامعتبر برای اطلاعات تماس، این بخش نادیده گرفته می‌شود و تأمین‌کننده بدون اطلاعات تماس ثبت می‌گردد.در پایان، شیء تأمین‌کننده ساخته شده و به فهرست تأمین‌کنندگان سیستم انبار افزوده می‌شود. در

[1] Def Add Supplier Cli

صورت موفقیت، پیام ثبت موفق نمایش داده شده و در صورت بروز خطا (مانند خطاهای نوع داده یا مشکلی در سازنده کلاس)، پیام هشدار چاپ خواهد شد.

```
def add_supplier_cli():
    print("Add a New Supplier")
    name = input("Enter supplier name: ")
    contact_input = input("Enter contact details (e.g., phone=123,
email=abc@example.com): ")
    contact_details = {}

    try:
        for pair in contact_input.split(','):
            if '=' in pair:
                key, value = pair.split('=')
                contact_details[key.strip()] = value.strip()
    except Exception:
        print("Invalid format for contact details. Skipping them.")
        contact_details = {}

    try:
        new_supplier = models.Supplier(name, contact_details)
        models.warehouse.add_supplier(new_supplier)
    except Exception as e:
        print(f"Failed to add supplier: {str(e)}")
```

تابع اصلی[1]

تابع اصلی[2] به عنوان نقطه‌ی شروع اجرای برنامه عمل می‌کند و سیستم مدیریت انبار را مقداردهی اولیه می‌نماید. در ابتدا، یک شیء از کلاس مدیریت انبار ایجاد شده و یک جایگاه پیش‌فرض با ظرفیت بالا به آن اضافه می‌شود تا آزمایش سیستم از همان ابتدا ممکن باشد.سپس منوی اصلی سیستم به‌صورت متنی چاپ می‌شود و کاربر می‌تواند از بین شش گزینه‌ی موجود یکی را انتخاب کند:

1. دریافت کالا: اجرای تابع مربوط به ثبت ورود کالا به انبار.
2. پردازش سفارش: اجرای فرآیند ثبت و پردازش سفارشات مشتریان.
3. گزارش‌گیری: نمایش خلاصه‌ای از موجودی کالاها و وضعیت جایگاه‌ها.

[1] Def Main
[2] main

4. افزودن جایگاه جدید: ثبت یک جایگاه ذخیره‌سازی جدید با مشخصات دلخواه.
5. افزودن تأمین‌کننده: ثبت مشخصات تأمین‌کننده جدید در سیستم.
6. خروج از سیستم: پایان اجرای برنامه.

ورودی کاربر بررسی می‌شود و در صورت ورود عددی نامعتبر، پیام خطا چاپ شده و از کاربر درخواست می‌شود تا گزینه‌ی صحیحی انتخاب کند.

```
def main():
    models.warehouse = models.Warehouse()

    default_bin = models.StorageBin("1BIN", 1000)
    models.warehouse.add_storage_bin(default_bin)

    menu = """
Warehouse Management CLI
------------------------
1. Receive Shipment
2. Process Order
3. Generate Inventory Report
4. Add Storage Bin
5. Add Supplier
6. Exit
"""
    while True:
        print(menu)
        choice = input("Select an option (1-6): ")

        if choice == "1":
            receive_shipment_cli()
        elif choice == "2":
            process_order_cli()
        elif choice == "3":
            generate_report_cli()
        elif choice == "4":
            add_storage_bin_cli()
        elif choice == "5":
            add_supplier_cli()
        elif choice == "6":
            print("Exiting the CLI. Goodbye!")
            break
        else:
            print("Invalid option. Please select a valid choice (1-6).")
```