

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных**  **технологий** | **Кафедра**  **управления и информатики**  **в технических системах** |

**ЛАБОРАТОРНАЯ РАБОТА №1**  
ПО ДИСЦИПЛИНЕ

«ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТА | 2 | КУРСА | | БАКАЛАВРИАТА | ГРУППЫ | ИДБ-24-12 |
|  | | | *(уровень профессионального образования)* | |  | |

|  |
| --- |
| **Загородникова Дмитрия Олеговича** |
| *(ФИО)* |

ТЕМА РАБОТЫ

|  |
| --- |
| «ООП в Python. Обработка исключительных ситуаций. Форматы XML и JSON» |

|  |  |
| --- | --- |
| Направление: | 09.03.03 Прикладная информатика |

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Кайкова Юлия Викторовна |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

МОСКВА 2025

Содержание

[Описание работы 3](#_Toc209308893)

[Программная реализация 4](#_Toc209308894)

[Вывод 15](#_Toc209308895)

# Описание работы

**Цель работы**: Написать классы для предметной области интернет-магазина электроники

**Вариант**: 14

**Технологии**:

1. Язык программирования Python,
2. диаграмма классов UML,
3. форматы данных XML и JSON.

**Инструменты**:

1. Интегрированная среда разработки для языка программирования Python Visual Studio Code
2. Сервис draw.io (https://app.diagrams.net/) или аналог,

**Задача**:

1. спроектировать в сервисе draw.io диаграмму классов для выбранной предметной области (предложив свой вариант или выбрав один из представленных) и написать по ней код на Python в объектно-ориентированном стиле;
2. обработать встроенные и собственное исключения (чтобы при работе пользователя не допускать выхода из программы при возникновении ошибок), после чего актуализировать диаграмму классов;
3. спроектировать в текстовом редакторе структуру для хранения данных предметной области в форматах XML и JSON;
4. написать код на Python, реализующий считывание из файла и запись в файл по разработанной структуре в форматах XML и JSON;
5. подготовить отчет о выполненной лабораторной работе и подготовиться по вопросам к защите.

# Программная реализация

import json

import xml.etree.ElementTree as ET

from typing import List

# ИСКЛЮЧЕНИЯ

class ProductNotFoundError(Exception):

    """Ошибка: товар не найден"""

class CustomerNotFoundError(Exception):

    """Ошибка: покупатель не найден"""

class InsufficientStockError(Exception):

    """Ошибка: недостаточно товара на складе"""

# МОДЕЛИ

class Product:

    """Класс описывает товар в интернет-магазине"""

    def \_\_init\_\_(self, id: int, name: str, category: str, price: float, stock: int):

        self.id = id

        self.name = name

        self.category = category

        self.price = price

        self.stock = stock

    # обновление наличия

    def update\_stock(self, quantity: int) -> None:

        if self.stock + quantity < 0:

            raise InsufficientStockError(f"Недостаточно товара '{self.name}' на складе.")

        self.stock += quantity

    # объект в словарь (для json и xml)

    def to\_dict(self) -> dict:

        return vars(self)

    # словарь в объект(для json и xml)

    @staticmethod

    def from\_dict(data: dict) -> "Product":

        return Product(\*\*data)

class Customer:

    """Класс описывает покупателя в интернет-магазине"""

    def \_\_init\_\_(self, id: int, name: str, email: str, address: str):

        self.id = id

        self.name = name

        self.email = email

        self.address = address

    def to\_dict(self) -> dict:

        return vars(self)

    @staticmethod

    def from\_dict(data: dict) -> "Customer":

        return Customer(\*\*data)

class CartItem:

    """Класс описывает один элемент корзины в интернет-магазине (товар+кол-во)"""

    def \_\_init\_\_(self, product: Product, quantity: int):

        self.product = product

        self.quantity = quantity

    def subtotal(self) -> float:

        return self.product.price \* self.quantity

class ShoppingCart:

    """Класс описывает корзину в интернет-магазине (товар+кол-во)"""

    def \_\_init\_\_(self, customer: Customer):

        self.customer = customer

        self.items: List[CartItem] = []

    def add\_item(self, product: Product, quantity: int) -> None:

        for item in self.items:

            if item.product.id == product.id:

                item.quantity += quantity

                return

        self.items.append(CartItem(product, quantity))

    def remove\_item(self, product\_id: int) -> None:

        self.items = [i for i in self.items if i.product.id != product\_id]

    def total\_price(self) -> float:

        return sum(i.subtotal() for i in self.items)

class Order:

    """Класс описывает заказ в интернет-магазине"""

    def \_\_init\_\_(self, id: int, customer: Customer, items: List[CartItem]):

        self.id = id

        self.customer = customer

        self.items = items

        self.status = "Создан"

        self.total = sum(item.subtotal() for item in items)

    def to\_dict(self) -> dict:

        return {

            "id": self.id,

            "customer\_id": self.customer.id,

            "items": [{"product\_id": i.product.id, "quantity": i.quantity} for i in self.items],

            "status": self.status,

            "total": self.total

        }

    @staticmethod

    def from\_dict(data: dict, customers: List[Customer], products: List[Product]) -> "Order":

        customer = next(c for c in customers if c.id == data["customer\_id"])

        items = []

        for i in data["items"]:

            product = next(p for p in products if p.id == i["product\_id"])

            items.append(CartItem(product, i["quantity"]))

        order = Order(data["id"], customer, items)

        order.status = data.get("status", "Создан")

        order.total = data.get("total", sum(i.subtotal() for i in items))

        return order

class Payment:

    """Информация об оплате"""

    def \_\_init\_\_(self, order\_id: int, amount: float, method: str, status: str = "Ожидание"):

        self.order\_id = order\_id

        self.amount = amount

        self.method = method

        self.status = status

    def to\_dict(self) -> dict:

        return vars(self)

    @staticmethod

    def from\_dict(data: dict) -> "Payment":

        return Payment(\*\*data)

class Delivery:

    """Информация о доставке"""

    def \_\_init\_\_(self, order\_id: int, address: str, status: str = "Не отправлено"):

        self.order\_id = order\_id

        self.address = address

        self.status = status

    def to\_dict(self) -> dict:

        return vars(self)

    @staticmethod

    def from\_dict(data: dict) -> "Delivery":

        return Delivery(\*\*data)

class Review:

    """Отзыв покупателя о товаре"""

    def \_\_init\_\_(self, id: int, product\_id: int, customer\_id: int, rating: int, comment: str):

        self.id = id

        self.product\_id = product\_id

        self.customer\_id = customer\_id

        self.rating = rating  # Оценка от 1 до 5

        self.comment = comment

    def to\_dict(self) -> dict:

        return vars(self)

    @staticmethod

    def from\_dict(data: dict) -> "Review":

        return Review(\*\*data)

class Category:

    """Категория товаров"""

    def \_\_init\_\_(self, id: int, name: str, description: str = ""):

        self.id = id

        self.name = name

        self.description = description

    def to\_dict(self) -> dict:

        return vars(self)

    @staticmethod

    def from\_dict(data: dict) -> "Category":

        return Category(\*\*data)

# ---------- Менеджер магазина ----------

class StoreManager:

    """Главный класс управления интернет-магазином"""

    def \_\_init\_\_(self):

        self.products: List[Product] = []

        self.customers: List[Customer] = []

        self.categories: List[Category] = []

        self.orders: List[Order] = []

        self.payments: List[Payment] = []

        self.deliveries: List[Delivery] = []

        self.reviews: List[Review] = []

    # --- CRUD: товары ---

    def add\_product(self, product: Product) -> None:

        self.products.append(product)

    def find\_product(self, product\_id: int) -> Product:

        for p in self.products:

            if p.id == product\_id:

                return p

        raise ProductNotFoundError(f"Товар с ID {product\_id} не найден.")

    def update\_product(self, product\_id: int, \*\*kwargs) -> None:

        product = self.find\_product(product\_id)

        for k, v in kwargs.items():

            if hasattr(product, k):

                setattr(product, k, v)

    def remove\_product(self, product\_id: int) -> None:

        self.products = [p for p in self.products if p.id != product\_id]

    # --- Категории ---

    def add\_category(self, category: Category):

        self.categories.append(category)

    # --- CRUD: клиенты ---

    def add\_customer(self, customer: Customer) -> None:

        self.customers.append(customer)

    def find\_customer(self, customer\_id: int) -> Customer:

        for c in self.customers:

            if c.id == customer\_id:

                return c

        raise CustomerNotFoundError(f"Покупатель с ID {customer\_id} не найден.")

    # --- Заказы ---

    def create\_order(self, customer\_id: int, cart: ShoppingCart) -> Order:

        customer = self.find\_customer(customer\_id)

        for item in cart.items:

            item.product.update\_stock(-item.quantity)

        order = Order(len(self.orders) + 1, customer, cart.items)

        self.orders.append(order)

        return order

    # --- Отзывы ---

    def add\_review(self, review: Review) -> None:

        self.reviews.append(review)

    # --- Оплата ---

    def add\_payment(self, payment: Payment) -> None:

        self.payments.append(payment)

    # --- Доставка ---

    def add\_delivery(self, delivery: Delivery) -> None:

        self.deliveries.append(delivery)

    # --- Сохранение / загрузка JSON ---

    def save\_to\_json(self, filename: str) -> None:

        data = {

            "products": [p.to\_dict() for p in self.products],

            "customers": [c.to\_dict() for c in self.customers],

            "categories": [c.to\_dict() for c in self.categories],

            "orders": [o.to\_dict() for o in self.orders],

            "payments": [p.to\_dict() for p in self.payments],

            "deliveries": [d.to\_dict() for d in self.deliveries],

            "reviews": [r.to\_dict() for r in self.reviews],

        }

        with open(filename, "w", encoding="utf-8") as f:

            json.dump(data, f, ensure\_ascii=False, indent=4)

    def load\_from\_json(self, filename: str) -> None:

        try:

            with open(filename, "r", encoding="utf-8") as f:

                data = json.load(f)

            self.products = [Product.from\_dict(p) for p in data.get("products", [])]

            self.customers = [Customer.from\_dict(c) for c in data.get("customers", [])]

            self.categories = [Category.from\_dict(c) for c in data.get("categories", [])]

            self.orders = [Order.from\_dict(o, self.customers, self.products) for o in data.get("orders", [])]

            self.payments = [Payment.from\_dict(p) for p in data.get("payments", [])]

            self.deliveries = [Delivery.from\_dict(d) for d in data.get("deliveries", [])]

            self.reviews = [Review.from\_dict(r) for r in data.get("reviews", [])]

        except FileNotFoundError:

            print(f"Файл {filename} не найден. Загружается пустой магазин.")

    def save\_to\_xml(self, filename: str):

        root = ET.Element("store")

        for key, items in [("products", self.products),

                           ("customers", self.customers),

                           ("categories", self.categories),

                           ("orders", self.orders),

                           ("payments", self.payments),

                           ("deliveries", self.deliveries),

                           ("reviews", self.reviews)]:

            section = ET.SubElement(root, key)

            for item in items:

                el = ET.SubElement(section, key[:-1])

                for k, v in item.to\_dict().items():

                    sub = ET.SubElement(el, k)

                    if isinstance(v, (dict, list)):

                        sub.text = json.dumps(v, ensure\_ascii=False)

                    else:

                        sub.text = str(v)

        tree = ET.ElementTree(root)

        tree.write(filename, encoding="utf-8", xml\_declaration=True)

    def load\_from\_xml(self, filename: str):

        try:

            tree = ET.parse(filename)

            root = tree.getroot()

            def parse\_section(tag):

                """Извлекает список элементов раздела XML в виде словарей"""

                items = []

                section = root.find(tag)

                if section is not None:

                    for el in section:

                        data = {child.tag: child.text for child in el}

                        items.append(data)

                return items

            def to\_int(value):

                try:

                    return int(value)

                except (TypeError, ValueError):

                    return value

            def to\_float(value):

                try:

                    return float(value)

                except (TypeError, ValueError):

                    return value

            # --- Загрузка объектов ---

            self.products = [

                Product(

                    id=to\_int(d.get("id")),

                    name=d.get("name", ""),

                    category=d.get("category", ""),

                    price=to\_float(d.get("price")),

                    stock=to\_int(d.get("stock")),

                )

                for d in parse\_section("products")

            ]

            self.customers = [

                Customer(

                    id=to\_int(d.get("id")),

                    name=d.get("name", ""),

                    email=d.get("email", ""),

                    address=d.get("address", ""),

                )

                for d in parse\_section("customers")

            ]

            self.categories = [Category.from\_dict(d) for d in parse\_section("categories")]

            # --- Восстановление заказов ---

            self.orders = []

            for d in parse\_section("orders"):

                order\_data = {

                    "id": to\_int(d.get("id")),

                    "customer\_id": to\_int(d.get("customer\_id")),

                    "items": json.loads(d.get("items", "[]")),

                    "status": d.get("status", "Создан"),

                    "total": to\_float(d.get("total", 0)),

                }

                try:

                    order = Order.from\_dict(order\_data, self.customers, self.products)

                    self.orders.append(order)

                except StopIteration:

                    print(f"Ошибка восстановления заказа ID {order\_data['id']}: клиент или товар не найден")

            # --- Оплаты ---

            self.payments = [

                Payment(

                    order\_id=to\_int(d.get("order\_id")),

                    amount=to\_float(d.get("amount")),

                    method=d.get("method", ""),

                    status=d.get("status", "Ожидание"),

                )

                for d in parse\_section("payments")

            ]

            # --- Доставки ---

            self.deliveries = [

                Delivery(

                    order\_id=to\_int(d.get("order\_id")),

                    address=d.get("address", ""),

                    status=d.get("status", "Не отправлено"),

                )

                for d in parse\_section("deliveries")

            ]

            # --- Отзывы ---

            self.reviews = [

                Review(

                    id=to\_int(d.get("id")),

                    product\_id=to\_int(d.get("product\_id")),

                    customer\_id=to\_int(d.get("customer\_id")),

                    rating=to\_int(d.get("rating")),

                    comment=d.get("comment", ""),

                )

                for d in parse\_section("reviews")

            ]

            print(f"Данные успешно загружены из файла '{filename}'")

        except FileNotFoundError:

            print(f"Файл '{filename}' не найден.")

        except ET.ParseError:

            print(f"Ошибка чтения XML-файла '{filename}'. Проверьте структуру XML.")

# ---------- Пример использования ----------

if \_\_name\_\_ == "\_\_main\_\_":

    store = StoreManager()

    store.add\_category(Category(1, "Ноутбуки"))

    store.add\_category(Category(2, "Телефоны"))

    store.add\_category(Category(3, "Аксессуары"))

    # --- Добавляем товары и покупателей ---

    store.add\_product(Product(1, "Ноутбук Lenovo", "Ноутбуки", 85000.0, 5))

    store.add\_product(Product(2, "Смартфон Samsung", "Телефоны", 55000.0, 10))

    store.add\_product(Product(3, "Наушники Sony", "Аксессуары", 12000.0, 15))

    store.add\_customer(Customer(1, "Тимофей Мальцев", "fala@gmail.com", "Москва, ул. Ташкентская 32/156"))

    store.add\_customer(Customer(2, "Олег Мандриков", "penza777@mail.ru", "Пенза, ул. Пушкина"))

    # --- Обновляем товар ---

    store.update\_product(3, price=11000.0, stock=20)

    print("\nОбновленный товар:", store.find\_product(3).to\_dict())

    # --- Создаем корзину ---

    cart = ShoppingCart(store.find\_customer(1))

    cart.add\_item(store.find\_product(1), 1)

    cart.add\_item(store.find\_product(2), 2)

    # --- Создаем заказ ---

    try:

        order = store.create\_order(1, cart)

        print(f"\nЗаказ №{order.id} успешно создан. Сумма: {order.total} руб.")

    except (ProductNotFoundError, CustomerNotFoundError, InsufficientStockError) as e:

        print("Ошибка при создании заказа:", e)

    # --- Оплата заказа ---

    payment = Payment(order\_id=order.id, amount=order.total, method="Карта")

    payment.status = "Оплачено"

    store.add\_payment(payment)

    print("\nИнформация об оплате:", payment.to\_dict())

    # --- Доставка заказа ---

    delivery = Delivery(order\_id=order.id, address=order.customer.address)

    delivery.status = "Отправлено"

    store.add\_delivery(delivery)

    print("\nИнформация о доставке:", delivery.to\_dict())

    store.add\_review(Review(1, 1, 1, 5, "Отличный ноутбук!"))

    # --- Сохраняем и загружаем данные ---

    store.save\_to\_json("store.json")

    store.save\_to\_xml("store.xml")

    print("\nДанные сохранены в файлы store.json и store.xml.")

    # --- Пробуем загрузить обратно ---

    print("\nЗагрузка данных из JSON:")

    store2 = StoreManager()

    store2.load\_from\_json("store.json")

    for p in store2.products:

        print(p.to\_dict())

    print("\nЗагрузка данных из XML:")

    store3 = StoreManager()

    store3.load\_from\_xml("store.xml")

    for p in store3.products:

        print(p.to\_dict())

        # --- Удаляем товар ---

    print("\nДо удаления:")

    for p in store3.products:

        print(p.to\_dict())

    store3.remove\_product(2)

    print("\nСписок товаров после удаления:")

    for p in store3.products:

        print(p.to\_dict())

    # --- Проверим обработку ошибок ---

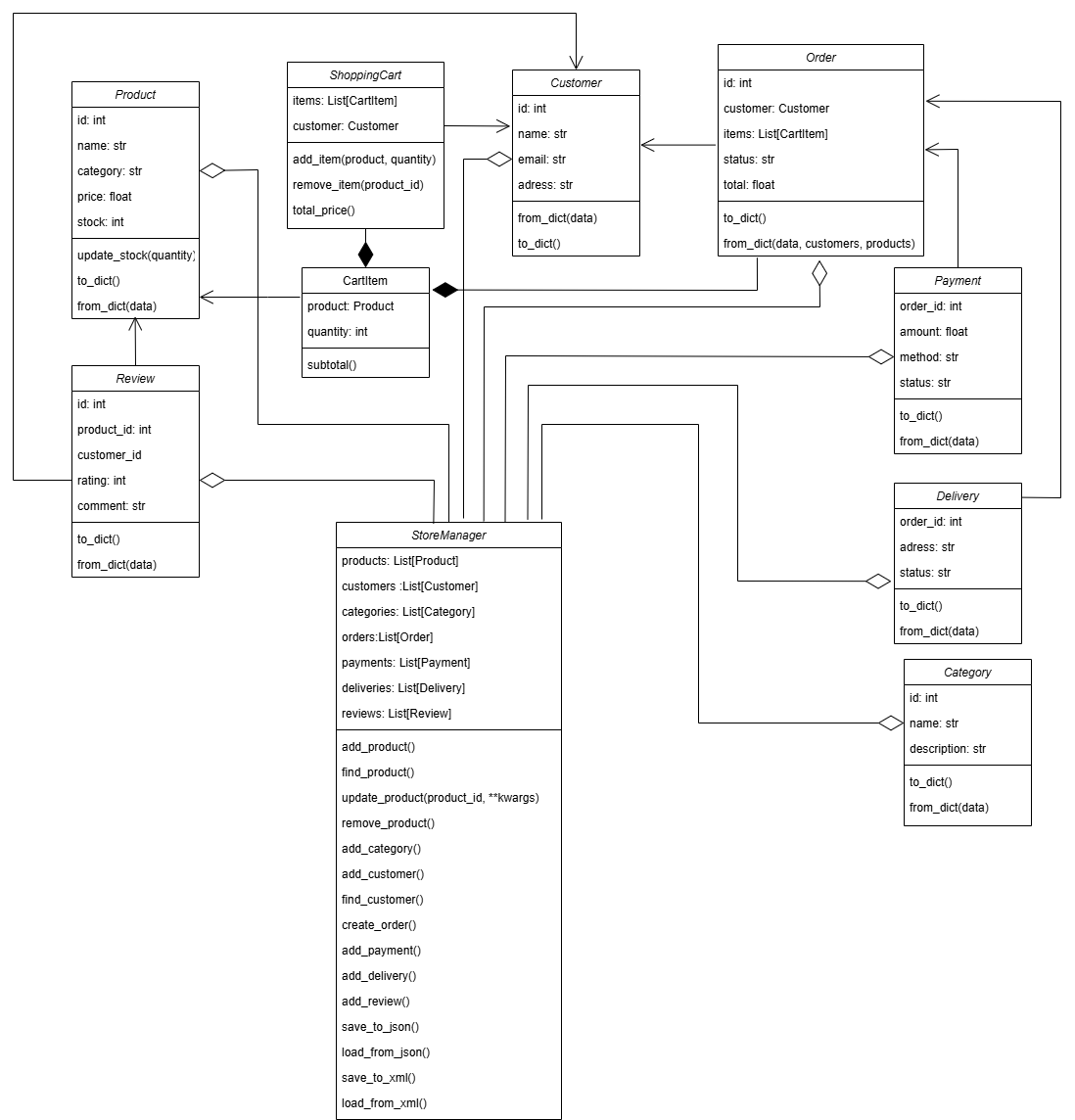
    print("\nПроверка исключений:")

    try:

        store.find\_product(99)

    except ProductNotFoundError as e:

        print("Ошибка:", e)



*Диаграмма классов UML*

# Вывод

В лабораторной работе я освоил навык работы с диаграммами классов UML и теперь могу строить собственные диаграммы. Такой подход делает понимание связей между сущностями гораздо легче. Также я попрактиковался в написании кода с ООП, обработкой исключений и записью/чтению данных из файлов формата XML и JSON.

Ссылка на репозиторий Git: https://github.com/ZAGAIQ/ProgLabStankin.git