Моделирование программы для управления данными о движении товаров на складе и эффективной обработки запросов на обновление информации в многопользовательской среде.  
  
Представим, что у вас есть система управления складом, где каждую минуту поступают запросы на обновление информации о поступлении товаров и отгрузке товаров.  
Наша задача заключается в разработке программы, которая будет эффективно обрабатывать эти запросы в многопользовательской среде, с использованием механизма мультипроцессорности для обеспечения быстрой реакции на поступающие данные.  
  
Создайте класс WarehouseManager - менеджера склада, который будет обладать следующими свойствами:

1. Атрибут data - словарь, где ключ - название продукта, а значение - его кол-во. (изначально пустой)
2. Метод process\_request - реализует запрос (действие с товаром), принимая request - кортеж.

Есть 2 действия: receipt - получение, shipment - отгрузка.  
а) В случае получения данные должны поступить в data (добавить пару, если её не было и изменить значение ключа, если позиция уже была в словаре)  
б) В случае отгрузки данные товара должны уменьшаться (если товар есть в data и если товара больше чем 0).  
  
3.Метод run - принимает запросы и создаёт для каждого свой параллельный процесс, запускает его(start) и замораживает(join).

**Пример работы:**

# Создаем менеджера склада  
manager = WarehouseManager()  
  
# Множество запросов на изменение данных о складских запасах  
requests = [  
    ("product1", "receipt", 100),  
    ("product2", "receipt", 150),  
    ("product1", "shipment", 30),  
    ("product3", "receipt", 200),  
    ("product2", "shipment", 50)  
]  
  
# Запускаем обработку запросов  
manager.run(requests)  
  
# Выводим обновленные данные о складских запасах  
print(manager.data)

**Вывод на консоль:**

{"product1": 70, "product2": 100, "product3": 200}

РЕШЕНИЕ:

class WarehouseManager:  
 def \_\_init\_\_(self): # В конструкторе \_\_init\_\_ инициализируется  
 # атрибут data как пустой словарь для хранения данных о запасах на складе.  
 self.data = {}  
  
 def process\_request(self, request): # Метод process\_request принимает запрос на изменение данных о запасах  
 product, action, quantity = request  
 if action == "receipt":  
 if product in self.data:  
 self.data[product] += quantity  
 else:  
 self.data[product] = quantity  
 elif action == "shipment":  
 if product in self.data and self.data[product] >= quantity:  
 self.data[product] -= quantity  
  
 def run(self, requests): # Метод run класса WarehouseManager принимает список запросов requests.  
  
 for request in requests:  
 self.process\_request(request)  
  
  
if \_\_name\_\_ == '\_\_main\_\_': # В блоке if \_\_name\_\_ == '\_\_main\_\_': создается экземпляр класса WarehouseManager.  
  
 # Создаем менеджера склада  
 manager = WarehouseManager()  
  
 # Затем создается список requests, который содержит запросы на изменение данных о складских запасах.  
 requests = [  
 ("product1", "receipt", 100),  
 ("product2", "receipt", 150),  
 ("product1", "shipment", 30),  
 ("product3", "receipt", 200),  
 ("product2", "shipment", 50)  
 ]  
  
 # Вызывается метод run менеджера склада с передачей списка запросов для обработки.  
 manager.run(requests)  
  
 # Выводим обновленные данные о складских запасах  
 print(manager.data)

ВЫВОД:

{'product1': 70, 'product2': 100, 'product3': 200}

Process finished with exit code 0