Задание №4. Многопоточное программирование.

«Эмулятор работы фабрики»

1. Постановка задачи

Напишите приложение, эмулирующее работу фабрики по сборке автомашин. Машина состоит из 3-х частей: кузов, двигатель и аксессуары. Машину надо собрать и отвезти на склад, откуда она поступает дилерам. Процесс работы фабрики показан на картинке:

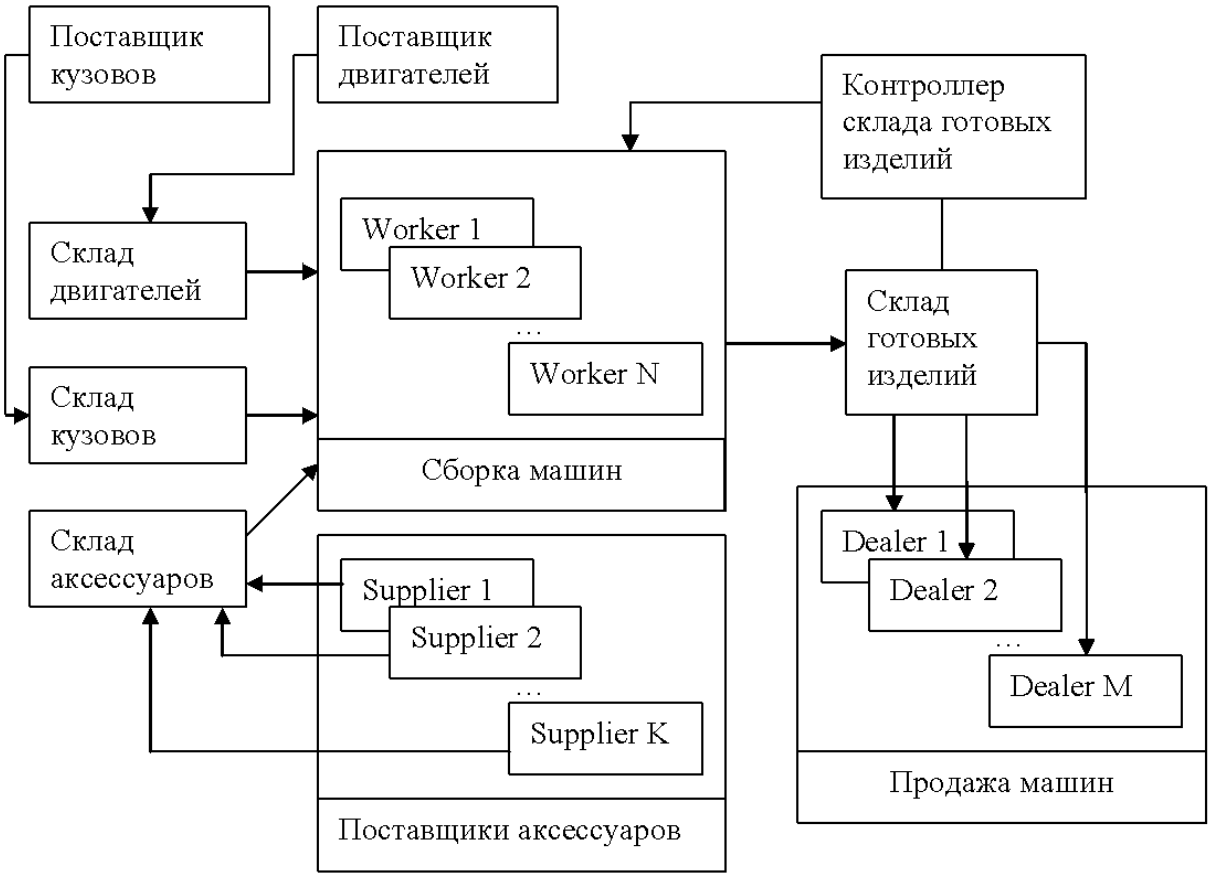

Рисунок 1 Схема работы фабрики по производству автомашин

2. Структура программы

/ru/nsu/ccfit/ФАМИЛИЯ/factory – пакет содержит основные классы программы
/ru/nsu/ccfit/ФАМИЛИЯ/threadpool – пакет для реализации пула потоков

3. Требования к программе

1. Все склады имеют определенный размер, который нельзя превышать. Размеры складов, количество сборщиков, поставщиков и дилеров задаются в конфигурационном файле. Приложение предоставляет графический интерфейс (библиотека Swing), где можно смотреть основные параметры работы фабрики и

контролировать процесс.

2. Каждый сборщик, поставщик и дилер должен работать в отдельном потоке. Для синхронизации и ожидания событий необходимо использовать мониторы синхронизации (notify(), notifyAll(), wait()). Наличие процедуры ожидания в виде цикла автоматически ведет к непринятию задания. Каждая деталь - это отдельный объект. Хранить просто количество изделий/деталей нельзя - необходимо хранить непосредственно объекты. Каждый объект должен иметь уникальный идентификатор для отслеживания.
3. Потоки, которые представляют поставщиков деталей, поставляют одну деталь раз в N миллисекунд. Если какой-то склад деталей полон, то поставщик ожидает освобождения места для деталей (используя методы wait(), notify()). Скорость работы поставщиков определяется 3-мя ползунками (для каждого типа деталей). Должно отображаться кол-во деталей на каждом из складов в текущий момент и кол-во деталей, произведенных поставщиками (для поставщиков аксессуаров можно общий показывать).
4. Потоки, которые представляют дилеров, запрашивают со склада готовой продукции 1 машину в M миллисекунд. Скорость запрашивания машин можно регулировать ползунком в интерфейсе окна. Интерфейс также должен показывать кол-во произведенных машин (вообще) и кол-во машин на складе в данный момент. При отправке машины дилеру информация о покупке должна писаться в лог работы фабрики (в файл) в виде строки:
   <Time>: Dealer <Number>: Auto <ID> (Body: <ID>, Motor: <ID>, Accessory: <ID>)
   Включение/отключение лога контролируется с помощью специального параметра в конфигурационном файле.
5. Поток контроллера склада готовой продукции просыпается при любом отправке машины со склада продукции. Он анализирует состояние склада и передает запрос на изготовление новых машин (в случае необходимости) на фабрику.
6. На фабрике работает несколько потоков (сборщиков) в рамках ThreadPool. Задачами для ThreadPool являются запросы на создание новых машин (от контроллера склада готовых изделий). При выполнении такой задачи сборщик должен взять по одной детали, необходимой для сборки машины, с соответствующих складов. Если на складе нет нужной детали, то поток ждет поставки. Собирая новую машину, рабочий создает новый объект и с помощью всех необходимых объектов, представляющих детали. После этого объект отправляется на склад готовой продукции. Если склад полон, то рабочий ждет освобождения места для новой машины. Интерфейс должен отображать, сколько всего было сделано машин и сколько задач еще ждут исполнителя (в очереди задач ThreadPool).
7. Конфигурационный файл должен предоставлять настройки для задания вместимости всех складов и количестве всех типов потоков. Примерный список параметров в конфигурационном файле (просьба использовать свои имена):
   StorageBodySize=100
   StorageMotorSize=100
   StorageAccessorySize=100
   StorageAutoSize=100
   AccessorySuppliers=5
   Workers=10
   Dealers=20

LogSale=true

8. Информацию о потоках и концепции ThreadPool можно найти на страничках:
    - http://ccfit.nsu.ru/~rylov/java_lections/index.html (Лекции №8, №9)
    - http://ccfit.nsu.ru/~deviv/ (ООП- >Java ->Документация-> Thinking in Java)

9. Особенности реализации

1. Классы объектной модели (деталь, склад, поставщик, сборщик и т.д.) не должны зависеть от библиотеки графического интерфейса SWING.
2. Пул потоков (ThreadPool) нужно реализовать в отдельном пакете. ThreadPool должен выполнять абстрактные задачи и не зависеть от реализации (задача на сборку автомашины в фабрике).
3. Программа должна корректно завершаться. При получении сигнала о закрытии окна все потоки должны корректно прерываться, и лог-файл закрываться.