Домашнее задание
по курсу “Конструирование программного обеспечения”
на тему
«Мультиагентная система управления рестораном»

Глоссарий

Посетитель ресторана – это человек, для обслуживания которого создается агент посетителя, который взаимодействует с самим посетителем через мобильное приложение или терминал в ресторане.

Повар – человек, задействованный в операциях по приготовлению блюд и напитков (характеризуется определенной почасовой оплатой, его простой также оплачивается, поэтому нежелателен), взаимодействует с управляющим агентом через сенсорный терминал на кухне.

Заказ – связан с конкретным посетителем, включает определенный набор блюд и напитков, выбранных из актуального меню ресторана с учетом возможности их приготовления за заданное время.

Счет – связан с определенным заказом, содержит список заказанных блюд и напитков и характеризуется итоговой суммой.

Меню – содержит динамический список блюд и напитков, который может изменяться в течение дня из-за исчерпания / поступления на склад некоторых ресурсов, загруженности поваров и / или кухонного оборудования.

Блюдо – присутствует в меню, готовится из продуктов согласно технологической карте процесса приготовления (по рецептуре, описанной в этой карте).

Напиток – это присутствующий в меню продукт, который как и блюдо приготавливается в ресторане согласно рецептуре из технологической карты (например, чай, кофе и т. д.), даже привозные в таре напитки, например кола в алюминиевых банках, требуют выполнения как минимум одной элементарной подготовительной операции извлечения из холодильника / со склада ресторана.

Ингредиент – продукт для приготовления конкретного блюда или напитка, объем которого измеряется в определенных единицах (штуках, килограммах (граммы являются их производными), литрах (миллилитры являются их производными)).

Ресурс – это людской ресурс (повар), единица кухонной техники (оборудование) или продукт, используемым для приготовления блюда / напитка в качестве ингредиента.

Процесс (рецептура приготовления блюда / напитка, описанная в технологической карте) – определенная последовательность элементарных операций над различными продуктами (ингредиентами в определенном объеме) с использованием людских ресурсов (поваров) и кухонного оборудования (печей, духовок и т. д.).

Операция (технологическая) – элементарное действие над определенным(и) продуктом(ами) на кухне (резка овощей, варка мяса, смешивание

ингредиентов, раскладывание приготовленных блюд в посуду и т. д.), для выполнения которого должен быть задействован максимум один повар и максимум одна единица кухонной техники (например, духовка, микроволновка и т. д., посуда сюда не входит), а также определенное время согласно рецептуре, указанной в технологической карте.

Введение

Мультиагентный подход – это способ построения компьютерной системы таким образом, чтобы она состояла из множества отдельных полуавтономных объектов, называемых агентами. Более формально, мультиагентная система (MAS) – это парадигма проектирования программного обеспечения, когда монолитная система разделена на несколько отдельных частей программного обеспечения, где каждая из них работает независимо, имеет свои собственные цели и алгоритм. Эти части программного обеспечения называются агентами. Таким образом, парадигма называется мультиагентной, потому что несколько агентов выполняют общую задачу, взаимодействуя друг с другом. Каждый агент действует от имени своего собственного пользователя или своего собственного представителя в реальном мире. И каждый агент пытается самостоятельно достичь конкретных целей пользователя, владельца или представляемого объекта. Даже если кажется, что, взятый по отдельности, каждый такой автономный фрагмент программного обеспечения должен быть проще, чем одно гигантское монолитное приложение, такого рода разделение приводит к проблемам и задачам, которые не имеют отношения к традиционному способу организации программных частей. Чаще всего одному агенту для достижения своей задачи приходится связываться и взаимодействовать с другим агентом, который, в свою очередь, пытается достичь своей собственной цели, решая некоторую задачу. Разработчик программного обеспечения должен предоставить всем таким агентам средства поиска и взаимодействия друг с другом.

Существуют различные типы агентов. Агенты могут быть как активными, так и пассивными. Пассивные агенты – это те, у кого нет конкретной цели или задачи, которую они постоянно пытаются выполнить. Они являются просто представлением некоторой среды, которая влияет на ситуацию и других агентов. Они обладают такими способностями как способность сотрудничать или вести переговоры. Разница в том, что они сами не инициируют новые соединения, транзакции или коммуникации. Активные агенты, напротив, выполняют определенную задачу и взаимодействуют со своей средой или другими агентами, постоянно находясь в поиске способа выполнения какой-либо активной задачи.

Агенты в таких системах обычно обладают следующими характеристиками:

1. Они децентрализованы, что означает, что они не являются частью единого монолитного приложения и могут запускаться независимо и удаленно.

2. Агенты автономны. Они способны выполнять свою задачу без внешнего контроля. Они самостоятельно взаимодействуют с агентами, которых считают необходимыми, и самостоятельно делают все, что в их силах, для достижения своих целей.

3. Агенты не должны иметь глобального представления о модели, что означает, что они должны быть в состоянии действовать в терминах своих локальных знаний, которые могут представлять только часть системы. Это дает агентам возможность работать, когда глобальная система слишком велика, и упрощает реализацию агента.

Реальные примеры использования многоагентной системы включают компьютерные игры (каждый персонаж может быть реализован с помощью отдельного агента). В настоящее время он широко используется в кинопроизводстве для имитации массовой толпы в батальных сценах.

В данном ДЗ необходимо создать мультиагентную систему для управления ресурсами высокотехнологичного ресторана. В этой модели ресурсы ресторана могут быть разделены на агентов, которые будут общаться и взаимодействовать друг с другом, чтобы доставлять заказы посетителям как можно быстрее. Каждый агент может знать способы оптимизации процесса выполнения своей задачи. Если все они эффективно выполняют свою работу, эффективность всей системы повышается.

Сценарий обслуживания посетителя пиццерии из реальной жизни

Рассмотрим процессы, которые функционирует в типовом высокотехнологичном ресторане ежедневно.

Человек делает заказ на веб-сайте пиццерии или в мобильном приложении, приложение рисует картинку пиццы, движущейся по конвейерной ленте. Каждая ступень этого конвейера представляет собой этап в процессе выпечки и доставки пиццы. Клиент может легко отслеживать статус своего заказа и получать информацию о том, сколько времени ему придется ждать, поскольку есть индикатор, который показывает приблизительное время до готовности заказа. И этот показатель постоянно обновляется. Этот простой факт получения информации о статусе успокаивает клиента и дает ему ощущение контроля над процессом.

С другой стороны, есть персонал ресторана, который готовит и оформляет заказы. Обычно у них на кухне установлено специальное устройство. Оно очень похоже на кассовый аппарат, который устанавливается в кассе для ведения бухгалтерского учета. Но вместо печати кассовых чеков устройство на кухне печатает (или выводит на сенсорный экран) заказы – список блюд, включенных в заказ. В очень популярных ресторанах, где поток посетителей огромен, эта машина постоянно печатает один заказ за другим. Администратор кухни должен убедиться, что все эти заказы распределены между поварами и в итоге поданы посетителям.

Возникает вопрос: каков правильный способ распределения этих заказов? Мы можем просто сделать простой циклический алгоритм. Он просто назначает все

задачи в порядке их создания всем работникам кухни в циклическом порядке. В случае ресторана заказ – это задача, а повар – это работник. Такой способ решения задач гарантирует, что все они в итоге будут выполнены. Но время ожидания для каждой отдельной задачи не обязательно является минимальным. Например, можно было бы изменить порядок выполнения задач, чтобы общее время обработки было меньше. В целом это можно сформулировать следующим образом:
При наличии нескольких задач и нескольких работников каждому работнику должна быть назначена любая задача. Для каждой задачи запрашивается определенная стоимость, которая может варьироваться в зависимости от работника, назначенного для этой задачи. Все задачи должны быть распределены между работниками таким образом, чтобы общие затраты были минимальными.
Эта проблема может быть решена с помощью нескольких алгоритмов, но можно утверждать, что в условиях постоянно поступающих заказов администратор кухни может выполнить все необходимые вычисления самостоятельно. Это означает, что некоторым клиентам приходится ждать дольше, чем это могло бы быть в случае автоматического распределения задач.
Более того, решения задачи распределения заказов недостаточно. Необходимо не только учитывать индивидуальный темп работы каждого повара, но и принимать во внимание тот факт, что несколько поваров не могут одновременно использовать одно и то же кухонное устройство. Это означает, что мы должны отслеживать очередь входящих задач для устройств таким же образом, как мы это делаем с поварами. Но если мы действительно отслеживаем всех участников процесса приготовления и точно знаем очередь их задач, мы можем легко предсказать время, оставшееся до того, как конкретный заказ будет подан клиенту.
Таким образом, мы можем организовать систему, показывающую клиенту точный статус его заказа и точное время его готовности.

Общие требования к системе:

1. Компонентами системы должны быть агенты.
2. Добавить новый компонент в систему должно быть легко. Если возникает необходимость в предоставлении услуги, которая ранее не предоставлялась, и для этого потребуется добавление нового компонента (определенного типа агента), это не должно приводить к переписыванию всей программной системы.
3. Система должна обеспечивать выполнение заказов посетителей организованно и оперативно. Ситуация, когда определенный заказ не выполняется, потому что все повара постоянно принимают новые, более простые заказы, не должна иметь место.
4. Посетителю должно быть сообщено приблизительное время ожидания заказа.

Основываясь на требованиях выше, необходимо сформулировать более детальные требования к программному продукту:

1. Отдельные агенты должны взаимодействовать друг с другом путем вызовов методов друг друга посредством отправки сообщений.
2. Отдельные агенты должны поддерживать методы обнаружения других агентов определенного типа.
3. Агенты для решения определенных задач должны уметь автоматически выбирать других агентов, которые им требуются, из списка обнаруженных агентов.
4. В случае задержки выполнения агента управляющий агент должен автоматически обнаружить этот факт и предпринять меры по задействованию нового агента в уже запущенные процессы (опционально).
5. Агенты должны контролировать процесс обслуживания посетителя с момента оформления заказа и до момента его выдачи.
6. Агенты должны быть способны отвечать на запросы, касающиеся приблизительного времени принятой ими в обработку задачи (например, выполнения заказа, процесса по приготовлению конкретного блюда или операции в рамках процесса).

Типы агентов MAC

Поскольку система должна быть построена с применением мультиагентного подхода, необходимо определить типы агентов, которые должны присутствовать в системе. Для этого нужно рассмотреть жизненный цикл заказа.

Предполагается, что посетитель делает заказ с помощью мобильного или веб-приложения, которое является точкой непрерывного взаимодействия с ним. Необходимо рассчитать и предоставить пользователю (вывести в лог-файл) приблизительное время готовности заказа. При мультиагентном подходе некоторые / все объекты (сущности) являются агентами. Итак, приложение – первый тип агента. Пусть это будет агент посетителя, его функциональность такова:

• сформировать меню, в котором скрыты элементы (блюда и напитки), которые по какой-либо причине в данный момент не могут быть поданы / приготовлены;
• вывести это меню посетителю на экран для создания заказа (опционально);
• отслеживать процесс выполнения заказа, чтобы актуализировать время его ожидания.

Созданный заказ сам по себе является отдельной сущностью, поэтому он может быть представлен как отдельный агент. Этот агент заказа должен:

• «знать» все отдельные пункты заказа;
• «знать» процесс приготовления каждого отдельного элемента заказа (блюда / напитка);
• связаться с другими агентами, чтобы убедиться, что заказ в итоге будет выполнен;

• собрать информацию от других агентов, чтобы рассчитать предполагаемое время ожидания и предоставить эту информацию агенту посетителя.

Обычно в таких системах необходимо иметь возможность применять специальные алгоритмы управления заказами, сокращая общее время, необходимое для их обслуживания. Чтобы достичь этого, можно создать умных агентов по оформлению заказов, которые вели бы переговоры с другими агентами, пытаясь таким образом сотрудничать и достигать лучших результатов. Однако наличие очень сложных агентов нецелесообразно в ситуациях, когда можно избежать перегрузки агентов функциональностью. Вместо этого можно ввести другой тип агента – управляющего агента. Его цель – находить и отслеживать всех агентов по заказу и управлять ими с помощью определенного алгоритма. Такой подход дает несколько преимуществ. Алгоритм управления заказами не привязан к агентам заказа. Управляющий агент может быть заменен другим агентом без необходимости что-либо делать с другими частями системы.

Типовой подход к реализации МАС на основе *Java*-фреймворка *Jade*

Необходимо создавать агенты, расширяя некоторый базовый класс родительского агента. Этот базовый класс сам по себе является полностью функционирующим агентом, который ничего не делает. Чтобы добавить некоторую функциональность агентам, необходимо реализовать “поведение”. Это классы, которые расширены из базового класса агента. Поведение должно обеспечивать реализацию двух методов – “*void action*()” и “*boolean done*()”. Первый вызывается, когда в рамках «поведения» агента должно выполниться некоторое действие. Второй метод вызывается после каждого вызова “*action*()”, чтобы определить, следует ли повторять это поведение или нет. Стоит создать ряд удобных классов, которые уже частично реализуют эти методы. Например, в классе “*OneShotBehaviour*” (*Java*-фреймворка *Jade*) метод “*done*()”всегда возвращает *true*, так что поведение «запускается» в первый раз и сразу после этого помечается как выполненное. Его противоположностью является класс “*CyclicBehaviour*”, который просто запускает метод “*action*” в цикле вечно. Это достигается за счет того, что в методе “*done*()” всегда возвращается значение *false*, из-за чего поведение не может завершиться. Все агенты должны предварительно выполнять все свои задачи с помощью моделей поведения, например:

```
public class ReserveSimpleResourceBehaviour extends OneShotBehaviour {
   private SimpleResource resource;
   private Action message;

   public ReserveSimpleResourceBehaviour(Agent agent, SimpleResource resource, Action message) {
      super(agent);
      this.resource = resource;
      this.message = message;
   }
```

```
    @Override
    public void action() {
        resource.reserve(message.getActor(), ((ReserveSimpleResource) message.getAction()).getAmount());
    }
}
```

Это поведение, которое срабатывает, когда агент ресурсов получает команду зарезервировать указанный объем ресурса, который он представляет. Очевидно, что он расширяет «*OneShotBehaviour*», и после добавления его в расписание агента он срабатывает только один раз. Это корректно, если необходимо выполнить команду только один раз. Метод “*action*()” переопределяется так, чтобы выполнить заданную задачу.

Агент Directory Facilitator

Главной особенностью агентов является то, что они способны находить друг друга и взаимодействовать друг с другом. Для этих целей, как правило, создается сервис *Directory Facilitator* (*DF*). Он хранит информацию обо всех зарегистрированных в нем агентах, чтобы другие агенты могли их найти. Агенту полезно зарегистрироваться в *DF* сразу после создания. Для этого можно создать специализированное поведение:

```
public class RegisterInDFBehaviour extends OneShotBehaviour {
        private String serviceType;
        private String ownership;
        public RegisterInDFBehaviour(Agent agent, String serviceType, String ownership) {
                super(agent);
                this.serviceType = serviceType;
                this.ownership = ownership;
        }

        @Override
        public void action() {
                ServiceDescription sd = new ServiceDescription();
                sd.setType(serviceType);
                sd.setOwnership(ownership);
                sd.setName(myAgent.getName());
                DFAgentDescription dfd = new DFAgentDescription();
                dfd.setName(myAgent.getAID());
                dfd.addServices(sd);
                try {
                        DFAgentDescription[] dfds = DFService.search(myAgent, dfd);
                        if (dfds.length > 0) {
                                DFService.deregister(myAgent, dfd);
                        }
                        DFService.register(myAgent, dfd);
                        System.out.println(myAgent.getLocalName() + " is ready.");
                } catch (Exception ex) {
                        System.out.println("Failed registering in DF! Shutting down...");
                        ex.printStackTrace();
                        myAgent.doDelete();
                }
        }
}
```

Это поведение назначается агенту. После регистрации другие агенты могут найти зарегистрированных агентов, используя метод «поиска» в «*DFService*».

Это универсальное поведение, и его можно использовать для регистрации любого агента независимо от его типа.

Сообщения и онтологии

Сообщения, которые агенты отправляют друг другу, должны быть представлены классом.

Например, простое сообщение от агента заказа агенту ресурса может выглядеть следующим образом:

```
public class ReserveSimpleResource implements AgentAction {
	private double amount;
	public double getAmount() {
		return amount;
	}

	public void setAmount(double amount) {
		this.amount = amount;
	}
}
```

Это простой класс с одним свойством, представляющим объем ресурса, который должен быть зарезервирован. Но, имея такой простой класс, возникает проблема – как, например, передавать эти данные по сети?

Можно выбрать один из трех способов передачи данных между агентами:

1. Напрямую передавать текстовые данные, представленные с помощью простых строк. Агенты могут использовать дополнительные методы для кодирования данных в строковый формат. Но в этом случае формат самой строки должен быть задан.
2. Второй способ – использовать технологию сериализации, предоставляемую языком *Java*. Но этот формат не удобочитаем для человека.
3. Третий способ заключается в предоставлении специальных определений объектов.

Для третьего случая совокупность таких определений объектов называется “онтологиями”.

Фрагмент кода, который регистрирует простой класс для операции резервирования в онтологии, выглядит следующим образом:

```
public static void registerOntology(Ontology ontology) throws OntologyException {
	AgentActionSchema as = new AgentActionSchema(SimpleResourceVocabulary.RESERVE_RESOURCE);
	ontology.add(as, ReserveSimpleResource.class);
	as.add(SimpleResourceVocabulary.AMOUNT,
	(PrimitiveSchema) ontology.getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);
}
```

Здесь определяется новая схема действия агента и в нее добавляется новое поле, представляющее собой объем ресурсов, который необходимо зарезервировать.

Требования к МАС управления рестораном
Любой агент должен иметь методы: «отправить сообщение другому агенту», «прочитать входящее сообщение от другого агента», «самоуничтожиться» (например, после выполнения своей задачи), «приостановить свое выполнение» (для последующего возобновления работы требуется соответствующее сообщение от агента, который выше его по рангу в иерархии). Все агенты создаются управляющим агентом.

Сценарий автоматизированного обслуживания посетителей ресторана
Посетитель приходит в ресторан. Садится за свободный столик и делает заказ через мобильное приложение и сразу оплачивает свой заказ. Или сначала он делает заказ на терминале самообслуживания, сразу оплачивает его и садится за свободный столик. Далее ожидает готовности блюд и напитков. Получает уведомление о готовности через мобильное приложение и / или на экране (информационной панели) в ресторане с указанием номера ячейки, которая автоматически открывается (аналог постамата). Посетитель забирает заказ из этой ячейки. После обслуживания посетитель уходит, а столик убирают уборщики ресторана (этот процесс не рассматривается в данной системе с точки зрения автоматизации).

В разрабатываемой МАС необходимо предусмотреть ряд агентов, среди которых, в частности, должны присутствовать агент посетителя, агент заказа, управляющий агент (супервизор), агент склада, агент продукта, агент процесса, агент операции, агент повара, агент оборудования. Необходимо описать поведение каждого типа агента и сообщения, отправляемые агентами друг другу.

Агент посетителя – запрашивает у управляющего агента текущее актуальное меню, содержащее исключительно блюда и напитки, которые могут быть приготовлены за заданное нормативное время (например, максимум за 20 мин) с указанием необходимого для приготовления каждого блюда / напитка времени. Обращается к управляющему агенту с запросом на формирование заказа из выбранных посетителем блюд и / или напитков.
Это агент, который взаимодействует непосредственно с клиентом. Он может быть оформлен в виде приложения для мобильного телефона, сайта, веб-страницы терминала ресторана и т. д. (эта часть системы не автоматизируется). Также он просит управляющего агента создать агентов заказа и взаимодействует с ними, чтобы получить и затем предоставить посетителю время ожидания заказа. Чтобы иметь возможность отключить недоступные пункты меню, этот агент может, например, пассивно ожидать соответствующих уведомлений от других агентов.
Варианты возможного поведения агента:
1. Добавить выбранный элемент меню (блюдо или напиток) в заказ.
2. Удалить выбранный элемент меню (блюдо или напиток) из заказа.

3. Отключить (сделать неактивным) пункт меню из-за недоступности необходимого ресурса.
4. Включить (сделать активным) ранее отключенный пункт меню из-за появления необходимого ресурса.
5. «Попросить» управляющего агента создать экземпляр агента заказа.
6. Отменить заказ, при этом уничтожается соответствующий агент заказа.
7. Получить обновленную информацию о предполагаемом времени ожидания заказа от агента заказа.

Управляющий агент (супервизор) управляет другими агентами для выполнения заказов посетителей. Запускает процесс создания нового заказа. На основании запроса от агента посетителя создает агента заказа, а после выполнения заказа контролирует уничтожение агента заказа. Взаимодействует с агентом склада. «Приказывает» ему зарезервировать для каждого экземпляра заказанного блюда (напитка) из определенного заказа заданный объем конкретного продукта. Создает и уничтожает всех прочих агентов.

Агент заказа – подчиняется управляющему агенту, хранит список заказанных посетителем блюд и / или напитков в виде списка созданных управляющим агентом агентов блюд / напитков.
Эти агенты взаимодействуют с другими типами агентов. Они отправляют агентам посетителей приблизительное время ожидания заказов; взаимодействуют с управляющим агентом и с агентами продуктов и процессов, чтобы получить необходимые ресурсы и выполнить требуемые процессы для выполнения заказа.
Типовое поведение агента:
1. Принимает и обрабатывает сообщения от управляющего агента.
2. Отправляет агенту посетителя информацию о времени ожидания его заказа.
3. «Просит» агентов процесса предоставить оценку времени ожидания.
4. Обрабатывает ответ от агентов процессов о времени ожидания готовности блюд / напитков.
5. Резервирует необходимые ресурсы для выполнения заказа.
6. Отменяет резервирование определенного ресурса (при отмене заказа).
7. Обрабатывает ответ от агентов продуктов о результате резервирования.
8. Продолжает процесс выполнения заказа; заказ может не быть обработанным после первого завершения такого поведения из-за информации об очереди, предоставленной управляющим агентом (опционально).

Агент блюда / напитка – содержит списки созданных управляющим агентом агентов процесса, операций и агентов продуктов для приготовления конкретного блюда / напитка из заказа посетителя. Уничтожается, когда данное блюдо / напиток приготовлено, а заказ выполнен.

Агент процесса. Агенты, представляющие процесс, выполняемый либо непосредственно человеком (поваром), либо устройством (кухонным оборудованием) под контролем человека. Фактически контролирует выполнение процесса, состоящего из технологических операций. Может предоставить информацию о примерном времени ожидания выполнения процесса.
Эти агенты фактически выполняют подготовку заказа. Они могут обращаться к агентам, представляющим человека (повара, готовящего мясо) или устройство (кофеварку, которая готовит эспрессо). Их цель – обеспечить выполнение заказов, созданных агентами посетителей с помощью управляющего агента (супервизора), чтобы фактически приготовить блюда / напитки с использованием продуктов, которые представляют соответствующие агенты продуктов.

Агент операции запрашивает управляющего агента зарезервировать агента повара и агента оборудования для выполнения операции процесса.

Агент склада – содержит список активных агентов продуктов, которые ему «подчиняются». Содержит таблицу имеющихся на складе продуктов с объемами остатков по каждому из них. «Подчиняется» управляющему агенту. «Проверяет», имеется ли в наличии на складе достаточный объем конкретного продукта для резервирования под экземпляр блюда / напитка для выполнения определенного заказа. Если такая проверка проходит успешно при получении приказа о резервировании от управляющего агента, создает агент продукта.

Агент продукта – представляет некоторый объем продукта, необходимый для приготовления определенного блюда / напитка. Создается агентом склада (или по его «просьбе» управляющим агентом) при формировании заказа, содержащего блюдо / напиток, для приготовления которого необходим этот продукт. При создании агента продукта необходимый объем продукта, физически находящегося на складе, резервируется. Под каждый экземпляр блюда / напитка определенного заказа создается свой агент продукта. Когда блюдо / напиток будет приготовлено, агент продукта уничтожается агентом склада, а ранее зарезервированный и физически потраченный поваром объем этого продукта списывается со склада.
Это агенты, представляющие определенные типы ресурсов – ингредиенты для блюд и напитков (мука, кофе и т. д.), могут предоставлять информацию о зарезервированных ресурсах.

Типовое поведение:

1. Зарезервировать указанный объем ресурса, который он представляет (например, 0.3 кг картофеля).
2. Отменить резервирование ресурса.
3. «Захватить» с помощью агента склада указанный объем ресурса со склада для использования в процессе приготовления блюда / напитка.

Агент повара – представляет конкретного человека – повара ресторана, взаимодействует с ним через кухонный сенсорный терминал, управляет его работой (назначает ему выполнение определенной операции, отменяет / приостанавливает (опционально) ранее назначенную ему операцию, возобновляет ранее приостановленную им операцию), принимает от него события, связанные с выполнением им операций («приступил к выполнению операции», «выполнил операцию», «отменил выполнение операции» (испорчен продукт в процессе приготовления)).

Агент оборудования – представляет конкретную единицу кухонного оборудования (печь, микроволновку и т. д.), подчиняется управляющему агенту, который управляет его использованием посредством воздействия на агента повара, соответствующего конкретному человеку (повару), использующему это оборудование в рамках выполнения назначенной ему определенной операции процесса приготовления блюда. / напитка.

Допущения

1. Пусть в проекте функции администратора (по контролю работы официантов и т. д.) возьмет на себя некий алгоритм (агент), реализация которого опциональна.
2. Пусть в проекте функции шеф-повара и су-шефов (по управлению поварами и т. д.) также возьмет на себя определенный алгоритм (агент), реализация которого опциональна.
3. Пусть в проекте возможность посетителем бронировать столик заранее и затем приходить в ресторан будет опциональной.
4. Пусть наличие в системе менеджера-планировщика – агента, который принимает и снимает бронь со столиков и размещает посетителей с учетом формируемого им расписания загрузки ресторана, будет опциональной.
5. Пусть реализация в системе агента официанта, соответствующего человеку, который принимает заказы от посетителей, размещенных за столиками, приносит блюда и напитки, принимает оплату и убирает столики, будет опциональной.

Задачи к реализации

Необходимо сымитировать работу ресторана за одну смену в модельном времени с ускорением в *n*-раз (например, в 100 раз) относительно реального. Опционально можно предусмотреть возможностью задавать скорость имитации (от реального времени до ускорения в сотни раз).

Для этого необходимо смоделировать поток посетителей, включая, например, ситуации, когда ресторан переполнен в определенные часы (обеденное и вечернее время) или когда посетителей относительно мало (утренние часы).

Возможная ситуация (опциональная): определенный продукт закончился на складе в середине дня, и все блюда, в которых он содержится в качестве

ингредиента, должны быть автоматически удалены из меню и недоступны для заказа посетителями до пополнения запаса.

Автоматизируемый сценарий обслуживания посетителей

Клиент пришел в ресторан, сел за столик и сделал заказ через приложение, для каждого блюда и / или напитка из этого заказа согласно соответствующей рецептуре запущен экземпляр процесса приготовления, состоящий из элементарных технологических операций (для каждой из них выделены или запланированы ресурсы), по готовности блюда и / или напитки доставляются посетителю через ячейку самообслуживания (аналог постамата).

Время выполнения каждого действия (операции) задается согласно нормативам, опционально можно предусмотреть небольшие случайные отклонения от них в ту или другую сторону.

Управляющему агенту необходимо распределять задачи таким образом, чтобы каждый повар был максимально загружен в операциях с минимумом простоев из-за ожидания высвобождения единиц кухонного оборудования (техники). То есть каждый повар должен быть задействован в различных операциях для приготовления разных блюд, а не готовить определенное блюдо полностью самостоятельно.

Каждое блюдо характеризуется себестоимостью, нормативным временем приготовления (когда все необходимые ресурсы доступны без ожидания) и ценой.

В алгоритме распределения операций при выполнении имеющихся заказов из текущего пула следует руководствоваться критерием получения максимальной выручки ресторана, которая зависит от количества и стоимости проданных блюд и напитков.

Необходимо предусмотреть загрузку данных в систему из текстовых JSON-файлов, например:

1. **Меню ресторана** (список блюд / напитков) – загружается в JSON-формате (menu_dishes.txt) при старте работы программы, например:

```
{
    "menu_dishes": [{
                "menu_dish_id": 28,
                "menu_dish_card": 518,
                "menu_dish_price": 59,
                "menu_dish_active": true
            }
    ]
}
```

2. **Справочник технологических карт** приготовления блюд / напитков (время, мин), загружается в JSON-формате (dish_cards.txt) при старте работы программы, например:

```
{
	"dish_cards": [{
		"card_id": 518,
		 "dish_name": "Princess Nuri tea bag in a paper cup",
		"card_descr": "pouring boiled water into a paper cup + 2 bags of sugar",
		"card_time": 0.15,
		~~"equip_type": 25,~~
		"operations": [{
			"oper_type": 17,
			***"equip_type": 25,***
			"oper_time": 0.15,
			"oper_async_point": 0,
			"oper_products": [{
					"prod_type": 18,
					"prod_quantity": 1
				},
				{
					"prod_type": 23,
					"prod_quantity": 2

				},
				{
					"prod_type": 24,
					"prod_quantity": 0.2
				},
				{
					"prod_type": 12,
					"prod_quantity": 1
				},
				{
					"prod_type": 19,
					"prod_quantity": 1
				}
			]
		}]
	}]
}
```


3. Справочник типов продуктов / расходников, загружается в JSON-формате (product_types.txt) при старте работы программы, например:


```
{
	"product_types": [{
			"prod_type_id": 11,
			"prod_type_name": "Frozen chicken",
```

```
                "prod_is_food": true
            },
            {
                "prod_type_id": 18,
                "prod_type_name": "Black tea in bags",
                "prod_is_food": true
            },
            {
                "prod_type_id": 22,
                "prod_type_name": "Can of cola 0.33",
                "prod_is_food": true
            } ,
            {
                "prod_type_id": 23,
                "prod_type_name": "Sugar in sticks",
                "prod_is_food": true
            } ,
            {
                "prod_type_id": 24,
                "prod_type_name": "Drinking water",
                "prod_is_food": true
            } ,
            {
                "prod_type_id": 12,
                "prod_type_name": "Paper cup 0.2",
              "prod_is_food": false
            },
            {
                "prod_type_id": 19,
                "prod_type_name": "Plastic stirrer",
              "prod_is_food": false
            }
        ]
}
```


4. Список продуктов / расходников на складе ресторана (тип продукта (из справочника), единица измерения, объем (остаток данной партии) в наличии, дата поставки (партии), цена покупки за единицу (партии), годен до (для этой партии)). Загружается в JSON-формате (products.txt) при старте работы программы, например:


```
{
    "products": [{
                "prod_item_id": 191,
                "prod_item_type": 11,
```

```
        "prod_item_name": "Frozen chicken",
        "prod_item_company": "AO OMPK",
        "prod_item_unit": "kilo",
        "prod_item_quantity": 15.3,
        "prod_item_cost": 190,
        "prod_item_delivered": "2023-01-12T08:12:36",
        "prod_item_valid_until": "2023-12-31T23:59:59"
},
{
        "prod_item_id": 172,
        "prod_item_type ": 18,
        "prod_item_name": "Princess Nuri tea in bags",
        "prod_item_company": "ORIMI",
        "prod_item_unit": "pc.",
        "prod_item_quantity": 874,
        "prod_item_cost": 1.5,
        "prod_item_delivered": "2023-01-15T08:10:36",
        "prod_item_valid_until": "2024-12-31T23:59:59"
},
{
        "prod_item_id": 132,
        "prod_item_type ": 22,
        "prod_item_name": "Can of cola 0.33",
        "prod_item_company": "Vimm-Bill-Dann",
        "prod_item_unit": "pc.",
        "prod_item_quantity": 0,
        "prod_item_cost": 39,
        "prod_item_delivered": "2023-01-11T09:22:36",
        "prod_item_valid_until": "2025-12-31T23:59:59"
} ,
{
        "prod_item_id": 213,
        "prod_item_type ": 23,
        "prod_item_name": "Russian sugar in sticks",
        "prod_item_company": "Russian sugar",
        "prod_item_unit": "pc.",
        "prod_item_quantity": 952,
        "prod_item_cost": 0.4,
        "prod_item_delivered": "2023-01-11T09:19:31",
        "prod_item_valid_until": "2025-12-31T23:59:59"
} ,
{
        "prod_item_id": 251,
        "prod_item_type": 24,
```

```
                "prod_item_name": "Drinking water Suzdalskaya",
                "prod_item_company": "Suzdalskie napitki",
                "prod_item_unit": "L",
                "prod_item_quantity": 174.22,
                "prod_item_cost": 11.45,
                "prod_item_delivered": "2023-01-11T09:19:31",
                "prod_item_valid_until": "2025-12-31T23:59:59"
        },
        {
                "prod_item_id": 133,
                "prod_item_type": 12,
                "prod_item_name": "Paper cup 0.2",
                "prod_item_company": "OOO KartonEco",
                "prod_item_unit": "pc.",
                "prod_item_quantity": 1032,
                "prod_item_cost": 0.45,
                "prod_item_delivered": "2023-01-12T08:12:36",
                "prod_item_valid_until": "2026-12-31T23:59:59"
        },
        {
                "prod_item _id": 139,
                "prod_item_type": 19,
                "prod_item_name": "Plastic stirrer",
                "prod_item_company": "Monplast EOOD",
                "prod_item_unit": "pc.",
                "prod_item_quantity": 12874,
                "prod_item_cost": 0.19,
                "prod_item_delivered": "2023-01-12T08:10:36",
                "prod_item_valid_until": "2026-12-31T23:59:59"
        }
    ]
}
```


5. Справочник типов кухонного оборудования. Загружается в JSON-формате (equipment_type.txt) при старте работы программы, например:


```
{
    "equipment_type": [{
                "equip_type_id": 2,
                "equip_type_name": "rotary oven"
        },
        {
                "equip_type_id": 25,
                "equip_type_name": "thermopot"
        }
```

```
	]
}
```

6. Перечень кухонного оборудования. Загружается в JSON-формате (equipment.txt) при старте работы программы, например:

```
{
	"equipment": [{
			"equip_id": 12,
			"equip_type": 2,
			"equip_name": "LIDER 250",
			"equip_active": true
		},
		{
			"equip_id": 13,
			"equip_type": 2,
			"equip_name": "LIDER 150",
			"equip_active": false
		},
		{
			"equip_id": 14,
			"equip_type": 25,
			"equip_name": "DAZHENG LOOKYAMI",
			"equip_active": true
		}
	]
}
```

7. Список поваров. Загружается в JSON-формате (cookers.txt) при старте работы программы, например:

```
{
	"cookers": [{
			"cook_id": 15,
			"cook_name": "Ivanov A. S.",
			"cook_active": true
		},
		{
			"cook_id": 16,
			"cook_name": "Petrov I. V.",
			"cook_active": true
		},
		{
			"cook_id": 14,
			"cook_name": "Sidorov G. S.",
			"cook_active": false
```

```
                }
        ]
}
```

8. Справочник типов операций на кухне. Загружается в JSON-формате (operation_types.txt) при старте работы программы, например:

```
{
        "operation_types": [{
                        "oper_type_id": 1,
                        "oper_type_name": "frying"
                },
                {
                        "oper_type_id": 17,
                        "oper_type_name": "pouring boiled water into a paper cup"
                }
        ]
}
```

9. Агент процесса приготовления блюда / напитка (создается при создании заказа для каждого блюда или напитка автоматически). Цель агента – выполнить конкретный процесс по приготовлению блюда / напитка. Процесс может состоять из одной или нескольких операций. После его выполнения агент уничтожается.

В процессе жизненного цикла агента формируется объект процесса, например приготовления чая (состоит из единственной операции):

```
{
        "proc_id": 434,
        "ord_dish": 625,
        "proc_started": "2023-02-28T10:12:36",
        "proc_ended": "2023-02-28T10:12:50",
        "proc_active": false,
        "proc_operations": [{
                "proc_oper": 82325
        }]
}
```

Объекты хранятся в оперативной памяти. Выгружаются как выходные данные перед завершением работы программы в JSON-формате (process_log.txt):

```
{
        "process_log": [{
                "proc_id": 434,
                "ord_dish": 625,
                "proc_started": "2023-02-28T10:12:36",
                "proc_ended": "2023-02-28T10:12:50",
```

```
                "proc_active": false,
                "proc_operations": [{
                        "proc_oper": 82325
                }]
        },
        {
                "proc_id": 435,
                "ord_dish": 626,
                "proc_started": "2023-02-28T10:12:51",
                "proc_ended": "2023-02-28T10:13:05",
                "proc_active": false,
                "proc_operations": [{
                        "proc_oper": 82326
                }]
        },
        {
                "proc_id": 436,
                "ord_dish": 627,
                "proc_started": "2023-02-28T11:12:36",
                "proc_ended": "2023-02-28T11:12:50",
                "proc_active": false,
                "proc_operations": [{
                        "proc_oper": 82326
                }]
        }]
}
```


10. Агент операции (создается при создании процесса приготовления блюда / напитка автоматически). Цель агента – выполнить конкретную технологическую операцию. После ее выполнения агент уничтожается.

В процессе жизненного цикла агента формируется объект операции (при этом, в частности, назначается повар для ее выполнения) ***и единица оборудования*** в соответствии с технологической картой, например приготовления чая:


```
{
        "oper_id": 82325,
        "oper_proc": 434,
        "oper_card": 518,
        "oper_started": "2023-02-28T10:12:37",
        "oper_ended": "2023-02-28T10:12:49",
        "oper_equip_id": 14,
        "oper_coocker_id": 15,
        "oper_active": false
}
```

Объекты хранятся в оперативной памяти. Выгружаются как выходные данные перед завершением работы программы в JSON-формате (operation_log.txt):


```
{
	"operation_log": [{
			"oper_id": 82325,
			"oper_proc": 434,
			"oper_card": 518,
			"oper_started": "2023-02-28T10:12:37",
			"oper_ended": "2023-02-28T10:12:49",
			"oper_equip_id": 14,
			"oper_coocker_id": 15,
			"oper_active": false
		},
		{
			"oper_id": 82326,
			"oper_proc": 435,
			"oper_card": 518,
			"oper_started": "2023-02-28T10:12:51",
			"oper_ended": "2023-02-28T10:13:04",
			"oper_equip_id": 14,
			"oper_coocker_id": 15,
			"oper_active": false
		},
		{
			"oper_id": 82327,
			"oper_proc": 436,
			"oper_card": 518,
			"oper_started": "2023-02-28T11:12:37",
			"oper_ended": "2023-02-28T11:12:49",
			"oper_equip_id": 14,
			"oper_coocker_id": 16,
			"oper_active": false
		}]
}
```


11. Список посетителей с заказами. Эти входные данные загружается в JSON-формате (visitors_orders.txt) при старте работы программы, например:


```
{
	"visitors_orders": [{
			"vis_name": " Visitor123",
			"vis_ord_started": "2023-02-28T10:12:37",
			"vis_ord_ended": "2023-02-28T10:13:05",
			"vis_ord_total": 118,
```

```
                "vis_ord_dishes": [{
                        "ord_dish_id": 625,
                        "menu_dish": 28
                    },
                    {
                        "ord_dish_id": 626,
                        "menu_dish": 28
                    }
                ]
            },
            {
                "vis_name": " Visitor124",
                "vis_ord_started": "2023-02-28T11:09:07",
                "vis_ord_ended": "2023-02-28T11:12:51",
                "vis_ord_total": 59,
                "vis_ord_dishes": [{
                    "ord_dish_id": 627,
                    "menu_dish": 28
                }]
            }
        ]
}
```


Поле "vis_ord_ended" во входном JSON-файле должно быть пустым (пустой строкой – ""), оно заполняется в соответствующем объекте по факту выполнения заказа (необходимо для удобного сбора статистики) и содержит дату и время завершения выполнения всего заказа, включая отказ.

Особенности реализации параллелизма процессов:

1. Если процесс состоит из более одной операции, то предполагается, что они могут выполняться параллельно (процесс с разветвлением(ями), асинхронно) или последовательно (синхронно).

Пример процесса из шести операций: [A*, B**, C**, D, E*, F].

A стартует одновременно с E, после выполнения A одновременно стартуют B и C, а D запустится после того, как обе операции B и C завершатся (они могут завершиться не одновременно), после завершения D и E начнется выполнение операции F, а когда она завершится, завершится и весь процесс с фиксацией времени.

Символ “ * ” (звездочка) в примере выше указывает на наличие параллелизма в процессе, в программе точки параллелизма задаются в поле «oper_async_point» (числом, большим 0), а если это поле содержит 0, то для такой операции отсутствуют другие операции, начинающиеся с ней одновременно. Поле, по сути, может содержать несколько «звездочек» (число больше 0), в зависимости от числа параллельных фрагментов процесса (точек

параллелизма). Если только у одной операции в поле «oper_async_point» присутствует, например число 3, то следует считать, что данные введены некорректно (должно быть как минимум две операции с тройками в этом поле). При этом необходимо вывести в лог-файл (errors.txt) ошибку в *JSON*-формате:

```
{
        "error": {
                "err_type": "input_error",
                "err_entity": "operation",
                "err_field": "oper_async_point"
        }
}
```

Если при старте программы обнаружена ошибка входных данных, то программы должна быть завершена с выводом errors.txt.

2. Операции в рамках одного процесса приготовления блюда / напитка могут выполняться различными поварами в зависимости от их загруженности, что позволит сократить время их простоя (если несколько поваров свободны, следует выбирать, того, кто был наименее загруженным в течение дня).

Требования к реализации программы

Язык реализации – *Java*. Приложение – консольное. Входные данные: текстовые файлы в *JSON*-формате. Выходные данные: текстовые файлы в *JSON*-формате, содержащие лог (статистику) обслуживания посетителей ресторана, а также работы агентов в процессе их обслуживания (запуск, действие и его результат, уничтожение).

Оценивание результатов:

1. Корректная ООП-реализация программы без поддержки параллелизма в процессах и реализации функционала оценки времени ожидания заказов, а также без проверки корректности входных данных: 4 балла.

2. Реализация дополнительного функционала поддержки параллелизма в процессах: + 2 балла.

3. Реализация дополнительного функционала оценки времени ожидания заказов (только в момент старта): + 1 балл.

3. Реализация дополнительного функционала оценки времени ожидания заказов (в момент старта и в любое другое время на любом этапе выполнения операций процессов): + 1 балл.

4. Реализация дополнительного функционала проверки корректности (обработки ошибок) входных данных ***(если на вход будет подан какой-либо JSON-файл в формате, отличном от заданного, нужно вывести лог с ошибкой, а также нужно проверять, заполнены ли обязательные поля в JSON-файлах)***: + 1 балл.

5. Корректное и обоснованное применение паттерн(а/ов) при проектировании МАС: + 1 балл.

Штрафы:

1. Отсутствие в папке проекта *jar*-файла: – 1 балл.
2. Отсутствие поддержки загрузки входных данных в *JSON*-формате: – 1 балл.
3. Отсутствие поддержки выгрузки выходных данных (логов) в *JSON*-формате: – 1 балл.

Бонусы:

1. Бонус за реализацию МАС на платформе *Jade* (без использования онтологий): + 1 балл.
2. Бонус за реализацию МАС на платформе *Jade* (с использованием онтологий): + 2 балла.

Платформа доступна на сайте: https://jade.tilab.com

Дедлайн 27 марта в 05:59. Удачи!

Зеленым цветом отмечены правки.