# Лабораторная работа №3: Многопоточность

Проводим моделирование процесса с общими ресурсами, которые необходимо использовать по очереди.

Моделируем посетителей, приходящих в бизнес-центр.

Каждый посетитель заходит в здание, проходит через проходную (одновременно 1 человек), вызывает лифт, едет на свой этаж, некоторое время там находится, потом снова вызывает лифт и выходит.

1. Создайте класс «BusinessCenter». Он отвечает за всё «обслуживание» посетителей – лифт, проходную и т.д.

|  |
| --- |
| **BusinessCenter** |
| - liftFree : boolean  - controlFree : boolean  - liftFloor : int |
| **<<constructors>>**  + BusinessCenter()  **<<methods>>**  + enterLift(v: Visitor )  + exitFromLift(v: Visitor)  + moveLift(targetFloor: int)  + enterControl(v: Visitor)  + exitFromControl(v: Visitor) |

Для простоты считаем, что в лифте и проходной одновременно может находиться один человек. На этаже посетители могут находиться одновременно.

Флаги liftFree и controlFree ограничивают доступ к общим ресурсам из различных потоком. При флаге, установленном в true, другие посетители будут ожидать освобождения при вызове ими enterLift/enterControl (используйте wait()).

moveLift обеспечивает пропорциональную расстоянию задержку, и отладочную печать

Весь общий доступ контролируется только в этом классе.

1. Создайте класс «Visitor» (не путать с шаблоном проектирования)

|  |
| --- |
| **Visitor** |
| - place : BusinessCenter  - totalCount : int  - num : int  - floor: int |
| **<<constructors>>**  + V**isitor**( p : BusinessCenter)  **<<methods>>**  + run()  - passControl()  - goUp()  - doSomeWork()  - goDown()  + toString() : String |

totalCount - счётчик посетителей, статическое поле, используемое в конструкторе для при присвоении номера

floor - на какой этаж идёт посетитель (случайное число 1...10)

num - номер посетителя, инициализируется в конструкторе, используется в toString()

Реализуйте интерфейс Runnable в классе Visitor. Метод run просто по очереди вызывает остальные методы.

Методы passControl() goUp() goDown()вызывают методы «занять» класса **BusinessCenter,** печатают соответствующие сообщения, с помощью Thread.sleep имитируют работу (например, проезд на n-й этаж за n\*100+200 мс), вызывают методы «освободить» класса **BusinessCenter.** Эти методы сбрасывают флаги в состояние «свободно» и уведомляют все ожидающие потоки (notify/notifyAll).

Класс Main cоздаёт 1 бизнес-центр и n посетителей с некоторой задержкой, запускает для каждого из посетителей отдельный поток и ждёт их завершения.

Очерёдность можно не соблюдать (первым проходит любой из ожидавших)

Пример вывода:

Пришёл Посетитель1.  
Пришёл Посетитель2.  
Посетитель1 показывает документы.  
Посетитель1 показал документы.  
Посетитель1 вызывает лифт.  
Посетитель2 показывает документы.  
Посетитель2 показал документы.  
Посетитель1 едет на этаж 7.  
Посетитель2 ждёт лифт.   
Посетитель1 вышел из лифта.  
Посетитель2 ждёт лифт с этажа 5.  
Посетитель1 что-то делает.  
Посетитель2 едет на этаж 7.  
Посетитель2 что-то делает.  
Посетитель1 закончил дела.  
Посетитель1 вызывает лифт.  
Посетитель2 закончил дела.  
Посетитель2 вызывает лифт.  
Посетитель2 ждёт лифт с этажа 7.  
Посетитель1 едет на этаж 1.  
Посетитель1 вышел из лифта.  
Посетитель2 едет на этаж 7.  
Посетитель2 едет на этаж 1.  
Посетитель1 вышел.  
Посетитель2 вышел.