Текстовый вариант видеоурока из предыдущего шага

Мьютексы в языке Go позволяют нам синхронизировать доступ к данным путем явной блокировки. То есть, как мы видели до этого, когда использовали каналы, мы также могли блокироваться, используя каналы, но отличие мьютекса в том, что он каналы не использует. Для того, чтобы создать мьютекс, можно воспользоваться пакетом sync, в котором есть структура мьютекс. Как мы видим на слайде, мы создаем переменную mu, в которую присваиваем созданную нами новую структуру мьютекс. Дальше с этой структурой уже мы можем работать, потому что у нее есть различные методы, которые мы рассмотрим на примере кода.

Для этого давайте сначала создадим ситуацию, в которой мы попадем в состояние гонки, так называемый race condition. То есть у нас доступ к какому-то одному ресурсу будет осуществляться из разных горутин одновременно. Для этих целей я написал функцию addAge, которая принимает уже знакомую нами структуру Client и число, которое нужно добавить к возрасту. И дальше просто мы добавляем переданное число к возрасту клиента. Здесь нюанс, что мы передаем ссылку, чтобы каждый раз не создавать нового клиента, а мы просто передаем ссылку на уже созданного.

В функции main мы как раз создаем клиента, берем ссылку. И в цикле вызываем в горутине функцию addAge, которая к переданному клиенту прибавит к возрасту 1. Смотрите, что мы будем ожидать. У нас цикл от 1 до 10, то есть мы вызовем 10 раз в 10 разных горутинов функцию addAge. И по нашей идеи, мы хотим, чтобы в итоге у клиентов возраст стал плюс 10, потому что мы каждую операцию будем прибавлять 1. Здесь по традиции поставим sleep, чтобы дождались завершения всех горутин. И выведем в итоге результат нашей структуры. Давайте запустим и посмотрим, что мы получили. В итоге мы получили 10. Смотрите, здесь есть такая

ловушка, что на маленьких данных нам может казаться, что все в порядке, мы запускаем и получаем всегда 10. И вроде бы все хорошо, но как только у нас увеличивается количество горутин, которые получают доступ к одним и тем же данным, появляется проблема.

Давайте проверим, сможем ли мы поймать эту проблему на 100 горутинах. Ну она достаточно редко сейчас у нас выстреливает, вот видите, мы получили 99. Но, если увеличим до 1000, здесь уже явно будем видеть проблему. Вот, как мы видим, практически ни один вызов нам не дал в результате 1000. Происходит это как раз из-за того, что мы хотим обновить одну и тут же переменную из 1000 разных горутин практически одновременно. Для этих целей как раз можно использовать мьютекс.

Давайте попробуем это сделать. Как мы видели на слайде мьютекс – это структура. Создадим эту структуру, и не забывайте о том, что структуру надо создавать и брать ссылку на нее, чтобы использовать везде в наших функциях именно ссылку. Это делается для того, чтобы мы использовали один и тот же мьютекс и не создавали каждый раз новый. Теперь в цикле мы вызываем нашу горутину с функцией addAge.

Что нам здесь нужно сделать? Нам нужно при входе в эту функцию заблокироваться и сказать, что раз уж горутина какая-то начала выполнять эту функцию, мы блокируем и не даем доступ остальным горутинам в эту функцию. То есть остальные должны подождать, пока первая горутина не закончит работу с этой функцией. Для этого давайте этот мьютекс передавать в саму горутину. И здесь нам нужно объявить, что мы ожидаем ссылку на структуру Mutex. Таким образом, здесь мы сделаем блокировку? путем вызова метода lock. После того, как мы заблокировались, мы обновим нашу переменную, и обязательно в конце

нам нужно разблокировать Mutex, потому что, если мы оставим его заблокированным, то в эту функцию уже не смогут попасть остальные горутины.

Давайте запустим сейчас и посмотрим, как поменялись наши данные. Смотрите, теперь каждый запуск наших горутин приводит к одному и тому же результату. Это происходит, потому что, используя мьютекс, у нас не возникает состояния гонки и горутины выстраиваются в очередь и ждут, пока отработает та горутина, которая уже начала выполнять эту функцию, закончит эту работу, выйдет, разблокирует мьютекс и после этого, мьютекс возьмет следующая горутина. Но мьютекс, на самом деле, представляет собой достаточно большую и тяжелую структуру. Им очень удобно пользоваться, когда у нас есть большие куски кодов в функциях, которые мы хотим заблокировать. То есть у нас начинается, например, функция, мы берем lock потом идет много разной логики, много разных вызовов, и в конце, например, мы делаем unlock. А для наших целей, которые в примере, по сути, здесь просто происходит одна операция инкремента, мы берем значение и прибавляем какое-то другое значение.

В этом случае удобно будет использовать другой инструмент, который называется atomic. Давайте напишем здесь. И у atomic-а, у пакета atomic есть метод addInt64, который атомарно выполнит нашу операцию. Мы передаем число, к которому хотим прибавить, и следующим параметрам передаем число, которое хотим прибавить в предыдущее. И здесь вы видите, что метод ругается, потому что нам нужны типы данных Int64, а у клиента у нас тип данных Int. Поэтому здесь я немножко изменю и укажу, что это Int64, а для следующего параметра я сделаю конвертирование типов Int64 вот таким вот образом. Ну и здесь, конечно же, нам нужна ссылка. Теперь мы можем не создавать мьютекс. И выполнить эту операцию атомарно. Давайте посмотрим на наш результат. И как мы видим, снова каждый запуск нашей программы выдает всегда одно и то же – 1000, и это корректное значение.