**Нет независимости нитей! Даешь synchronized!**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 7](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=7), Лекция 3

— Привет, Амиго! У нас есть панацея – лекарство от всех болезней. Как мы уже успели убедиться – неконтролируемое переключение нитей – это проблема.

— А почему бы нитям самим не решать, когда переключиться на следующую? Сделала все важные дела и «маякует», я – все!

— Разрешать нитям самим управлять своим переключением – еще большая проблема. Вдруг какой-то код не очень красиво написан, и нить никогда сама не отдаст свое «процессорное время». Давным-давно так и было – и это был тихий ужас.

— Ладно. И какое же решение есть?

—**Блокировка нитей.** И вот как это работает.

Было выяснено, что нити мешают друг другу, когда пытаются сообща работать с общими объектами и/или ресурсами. Как в примере с выводом на консоль: консоль одна, а выводят на нее все нити. Непорядок.

Поэтому был придуман специальный объект – мютекс. Это как табличка на двери туалета «свободно» «занято». Он имеет два состояния – объект свободен и объект занят, или их еще называют заблокирован и разблокирован.

Когда какой-то нити нужен общий для всех нитей объект, она проверяет мютекс, связанный с этим объектом. Если мютекс свободен, то нить блокирует его (помечает как занятый) и начинает использование общего ресурса. После того, как она сделала свои дела, мютекс разблокируется (помечается как свободен).

Если же нить хочет использовать объект, а мютекс заблокирован, то нить засыпает в ожидании. Когда мютекс, наконец, освободится занятой нитью, наша нить тут же заблокирует его и приступит к работе. Аналогия с табличками для туалета один в один.

— А как работать с этим мютексом. Надо создавать специальные объекты?

— Все намного проще. Разработчики Java встроили этот мютекс в класс Object. Тебе даже создавать его не придется. Он есть у каждого объекта. Вот как это все работает:

|  |  |  |  |
| --- | --- | --- | --- |
| **Код** | | **Описание** | |
| class MyClass  {  private String name1 = "Оля";  private String name2 = "Лена";  public void swap()  {  synchronized (this)  {  String s = name1;  name1 = name2;  name2 = s;  }  }  } | | Метод swap меняет местами значения переменных name1 & name2.  Что же будет если его вызывать из двух нитей одновременно? | |
| **Итоговый порядок** | **Код первой нити** | | **Код второй нити** |
| String s1 = name1; //Оля  name1 = name2; //Лена  name2 = s1; //Оля  String s2 = name1; //Лена  name1 = name2; //Оля  name2 = s2; //Лена | String s1 = name1;  name1 = name2;  name2 = s1;  //исполняется другая нить  //исполняется другая нить  //исполняется другая нить | | //нить ждет, пока освободится мютекс  //нить ждет, пока освободится мютекс  //нить ждет, пока освободится мютекс  String s2 = name1;  name1 = name2;  name2 = s2; |

|  |
| --- |
| **Итог** |
| Значения переменных были дважды обменяны местами и вернулись на первоначальное место. |

Обрати внимание на ключевое слово **synchronized**.

— Да, а что оно значит?

**— Когда одна нить заходит внутрь блока кода, помеченного словом synchronized, то Java-машина тут же блокирует мютекс у объекта, который указан в круглых скобках после слова synchronized. Больше ни одна нить не сможет зайти в этот блок, пока наша нить его не покинет. Как только наша нить выйдет из блока, помеченного synchronized, то мютекс тут же автоматически разблокируется и будет свободен для захвата другой нитью.**

Если же мютекс был занят, то наша нить будет стоять на месте и ждать когда он освободится.

— Так просто и так элегантно. Красивое решение.

— Ага. А как ты думаешь, что будет в этом случае?

|  |  |
| --- | --- |
| **Код** | **Описание** |
| class MyClass  {  private String name1 = "Оля";  private String name2 = "Лена";  public void swap()  {  synchronized (this)  {  String s = name1;  name1 = name2;  name2 = s;  }  }  public void swap2()  {  synchronized (this)  {  String s = name1;  name1 = name2;  name2 = s;  }  }  } | Методы swap и swap2 имеют один и тот же **мютекс** – объект **this**. |

Что будет, если одна нить вызовет метод swap, а другая – метод swap2?

— Т.к. мютекс у них один, то второй нити придется ждать, пока первая нить выйдет из блока synchronized, поэтому проблем с одновременным доступом тут не будет.

— Молодец, Амиго! Верное решение!

Хотелось бы обратить твое внимание на то, что словом synchronized может быть помечен как кусок кода, так и метод. Вот что это значит:

|  |  |
| --- | --- |
| **Код** | **Что происходит на самом деле** |
| class MyClass  {  private static String name1 = "Оля";  private static String name2 = "Лена";  public synchronized void swap()  {  String s = name1;  name1 = name2;  name2 = s;  }  public static synchronized void swap2()  {  String s = name1;  name1 = name2;  name2 = s;  }  } | class MyClass  {  private static String name1 = "Оля";  private static String name2 = "Лена";  public void swap()  {  synchronized (this)  {  String s = name1;  name1 = name2;  name2 = s;  }  }  public static void swap2()  {  synchronized (MyClass.class)  {  String s = name1;  name1 = name2;  name2 = s;  }  }  } |