Лекция 4

МНОГОПОТОЧНОЕ ПРОГРАММИРОВАНИЕ

Модель потоков в Java

- Java использует потоки (threads), чтобы была возможность написания асинхронных программ.
- Однопоточные системы используют подход, называемый цикл событий (*event loop)* с голосованием (*polling)*.

Приоритеты в потоках

- Приоритеты потоков представляют собой целые числа, которые определяют относительный приоритет одного потока к другому.
- Поток может добровольно передать управление.
- Поток может быть вытеснен более приоритетным потоком.

Синхронизация

- Для обеспечения синхронизации потоков в Java есть средство, которое называется *монитор*.
- Когда поток находится внутри синхронизирующего метода ни один другой поток не может вызвать какой-либо другой синхронизирующий метод на том же самом объекте.

Сообщения

- Когда в разрабатываемом многопоточном ПО потоки пишутся на разных языках программирования, то необходимо обеспечить между ними коммуникацию.
- В Java существует способ для общения между собой двух и более потоков через вызовы предопределённых методов, которые имеются у всех объектов.

Класс Thread и интерфейс Runnable

- Для создания нового потока, в программе необходимо либо создать класс-наследник **Thread,** либо реализовать интерфейс **Runnable**.
- Наиболее часто используемые методы класса Thread: getName(), getPriority(), isAlive(), join(), run(), sleep(), start().

Главный поток

- От главного потока порождаются все “дочерние”.
- Часто главный поток бывает последним потоком завершающим выполнение, т.к. он выполняет различные завершающие действия.

Реализация интерфейса Runnable

- Для реализации интерфейса **Runnable**, необходимо написать реализацию единственного метода **run().**
- public void run()

Создание экземпляра Thread

- Класс-наследник должен переопределять метод **run()**, который является точкой входа для нового потока.
- Необходимо использовать следующую форму конструктора **Thread**:
- public Thread(String *threadName*)

Какой выбрать способ?

- Класс **Thread** содержит несколько методов, которые могут быть переопределены в наследуемом классе.
- Многие программисты говорят, что классы необходимо наследовать только тогда, когда они расширяют или переопределяют существующий класс.

Создание нескольких потоков

- Как правило, в промышленном ПО используются не 2 потока, а гораздо больше.

Использование isAlive() и join()

- Как один поток может узнать, когда завершится другой?
  - Во-первых, через вызов метода **isAlive()** в потоке.
  - final boolean isAlive()

Приоритеты потоков

- В теории через определённый период времени потоки с высоким приоритетом получают больше времени CPU, чем потоки с низким приоритетом.
- В целях безопасности потоки с одинаковым приоритетами должны иногда уступать управление.

Синхронизация

- Когда 2 или более потокам необходим доступ к общему ресурсу, то им нужен способ, чтобы узнать, когда ресурс будет использоваться только одним потоком.
- Монитор - это объект, который используется как взаимоисключающая блокировка.

Использование синхронизирующих методов

- Для включения монитора объекта необходимо вызвать метод, который будет помечен ключевым словом **synchronized**.
- До тех пор, пока поток находится внутри синхронизирующего метода все другие потоки, пытающиеся вызвать его (или любой другой синхронизирующий метод) переводятся в состояние ожидания.

Блоки синхронизации

- Для использования синхронизации в классах, не использующих её, необходимо вызывать методы класса внутри блока синхронизации.
- synchronized(objRef) {

// выражения, которые необходимо синхронизировать

}

Межпотоковое взаимодействие

- Использование неявных мониторов в объектах Java эффективно, но можно достичь более высокого уровня управления через межпроцессное взаимодействие.
- У потоков есть интересная особенность: они начинаются после «голосования».

Взаимоблокировка

- В основном, взаимоблокировка происходит редко, например, когда два потока выполняются одновременно.
- Взаимоблокировка может включать более, чем два потока и два синхронизируемых объекта.

Приостановка, возобновление и остановка потоков

- Для показа текущего времени в операционной системе используется отдельный поток.

Получение состояния потока

- Для получения текущего состояния потока используется метод **getState()**, определённый в классе **Thread**:

```
Thread.State getState()
```

Использование фабричного метода для создания и запуска потока

- В некоторых случаях бывает удобно одновременно создавать и запускать поток.

Применение многопоточности

- Когда имеются 2 подсистемы внутри программы, которые могут выполняться одновременно, необходимо запускать их в виде индивидуальных потоков.
- With the careful use of multithreading, you can create very efficient programs.
- A word of caution is in order, however: If you create too many threads, you can actually degrade the performance of your program rather than enhance it.