Fork – Вилка, развилка, ответвление, разветвление, распутье

Join – присоединиться, вступить

Pool – пул, объединение, общий кател, омут

fork join pool - пул соединения ветвления(присоединение ветвления к пулу)

**О многопоточной обработке**

Различаю 2 разновидности многопоточности:

* На основе процессов;
* На основе потоков;

*Процесс* представляет собой исполняемую программу. Поэтому *многозадачеость на основе процессов* – это средство, обеспечивающее возможность выполнения на компьютере одновременно нескольких программ. При организации многозадачности на основе процессов программа является наименьшей единицей кода, выполнение которой может координировать планировщик задач.

При организации *многозадачности на основе потоков* наименьшей единицей диспетчеризуемого кода является *поток*. Это означает, что в рамках одной программы, может выполняться одновременно несколько задач. Несмотря на то что программы на Java выполняются в среде, поддерживающей многозадачность на основе процессов, в самих программах управлять процессами нельзя. Доступной остается *только* *многозадачность на основе потоков*.

Благодаря многопоточной обработке программа может решать какую-нибудь другую задачу во время вынужденного простоя процессора. Например, в то время как одна часть программы отправляет файл через соединение с Интернетом, другая ее часть может выполнять чтение текстовой информации, вводимой с клавиатуры, а третья - осуществлять буферизацию очередного блока отправляемых данных.

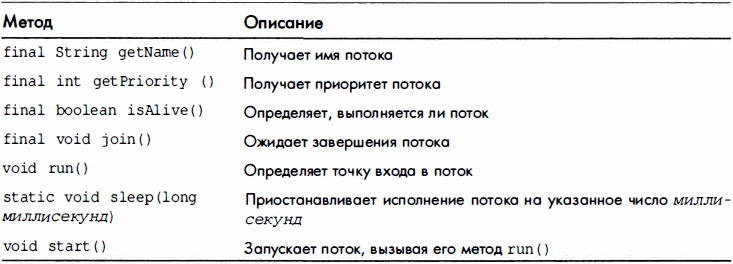
Поток может находиться в одном из нескольких состояний. В целом поток может быть *выполняющимся; готовым к выполнению*, как только он получит время и ресурсы ЦП; *приостановленным*, т.е. временно не выполняющимся; *возобновленным* в дальнейшем; *заблокированным* в ожидании ресурсов для своего выполнения; а также *завершенным*, когда его выполнение закончено и не может быть возобновлено.

В связи с организацией многозадачности на основе потоков возникает потребность в особого рода режиме, который называется синхронизацией и позволяет координировать выполнение потоков строго определенным образом. Для такой синхронизации в Java предусмотрена отдельная подсистема, основные средства которой мы рассмотрим далее.

**Класс Тhread и интерфейс Runnable**

В основу системы многопоточной обработки в Java положены класс Thread и интерфейс Runnable, входящие в пакет java.lang. Класс Thread инкапсулирует поток исполнения. Для того чтобы образовать новый поток, нужно создать класс, являющийся подклассом Thread или реализующий интерфейс Runnable.

В классе Thread определен ряд методов, позволяющих управлять потоками. Некоторые из этих наиболее употребительных методов описаны ниже. По мере их представления в последующих примерах программ мы ознакомся с ними поближе.



В каждом процессе имеется как минимум один поток исполнения, который называется *основным потоком*. Он получает управление уже при запуске программы. Следовательно, во всех рассмотренных до сих пор примерах использовался основной поток. От основного потока могут быть порождены другие, подчиненные потоки.

**Создание потока**

Для того чтобы создать поток, нужно построить объект типа Thread. Класс Thread инкапсулирует объект, который может стать исполняемым. Как уже отмечалось, в Java пригодные для исполнения объекты можно создавать двумя способами:

* Реализуя интерфейс Runnable;
* Создавая подкласс Thread;

Создание экземпляра потока, организация доступа к нему и управление потоком осуществляются средствами класса Thread. Единственное отличие обоих способов состоит в том, как создается класс, активизирующий поток.

Интерфейс Runnable дает абстрактное описание единицы исполняемого кода. Для

формирования потока подходит любой объект, реализующий этот интерфейс. В интерфейсе

Runnble объявлен только один метод - run ():

**public void** run()

**Расширение класса Thread.**