Pro Java №24

До этого времени все наши программы выполнялись в однопоточном режиме.

Многопоточность – это свойство платформы, при котором процесс может состоять из нескольких параллельных потоков.

Метод `main` является точкой входа для любой Java-программы. Когда программа запускается, JVM (Java Virtual Machine) создает главный поток, который начинает выполнение кода внутри метода `main`.

Процесс – это исполняемая программа, которая может состоять из одного или нескольких потоков.

Класс `Thread` является родительским классом для потоков в Java.

Метод `run()` представляет собой точку входа для выполнения кода в новом потоке. При создании потока необходимо переопределить метод `run()` и написать в нем код, который должен выполняться в этом потоке.

Метод `start()` класса `Thread` запускает новый поток выполнения. При этом создается новый поток, и метод `run()` выполняется в этом новом потоке.

Порядок выполнения потоков определяется внутренней системой компьютера и поэтому может быть не особо важен для нас.

Дочерние потоки – это дополнительные потоки, которые создаются и запускаются внутри главного потока.

1. Наследование от класса `Thread`

При наследовании от класса `Thread`, необходимо переопределить метод `run()` и реализовать в нем логику выполнения потока. Затем создаем экземпляр класса и вызываем метод `start()` для запуска нового потока.

2. Реализация интерфейса `Runnable`

При реализации интерфейса `Runnable`, необходимо реализовать метод `run()`. Для запуска потока создаем экземпляр класса, реализующего `Runnable`, и передаем его в конструктор объекта `Thread`. Затем вызываем метод `start()` для запуска потока.

Реализация интерфейса `Runnable` позволяет классу реализовывать несколько интерфейсов, что обеспечивает большую гибкость и возможность использовать дополнительные функциональности.

Если класс реализует `Runnable` и не предполагает наличие полей с переменными, то можно упростить его запись до лямбда-выражения.

В многопоточности выполнение кода в дочерних потоках начинается одновременно с выполнением кода в главном потоке, как только они запущены.

В многопоточном приложении каждый поток выполняет свою часть работы, и завершение работы главного потока не обязательно означает завершение работы всего приложения. Приложение может продолжать работать, пока все дочерние потоки не завершат свою работу.

Метод `sleep(long millis)` из класса `Thread` используется для приостановки выполнения потока на указанный промежуток времени в миллисекундах. Этот метод может бросить исключение `InterruptedException`, если поток прерывается во время сна.

Метод `join()` из класса `Thread` используется для приостановки выполнения текущего потока до тех пор, пока поток, на котором вызван метод `join()`, не завершит выполнение. Это полезно, когда необходимо дождаться завершения выполнения дочернего потока перед продолжением выполнения основного потока.