**Thread-Runnable**

*В русской терминологии за термином****Thread****укрепился перевод "Поток". Хотя это слово также можно перевести как "Нить".*В Java многопоточность программы организуется с помощью интерфейса Runnable и класса Thread, который наследуется от Runnable. Первый способ более гибкий, второй – проще.

Thread - это абстракция над физическим потоком.

Runnable - это абстракция над выполняемой задачей.

Та часть кода, которая должна выполняться в отдельном потоке, выносится в свой класс, имеющий переопределенный метод run(). Код метода run() выполняется, когда к объекту типа Thread применяется метод start(). Непосредственный вызов run() новый поток не создает.

**public** **class** ThreadTest {

**public** **static** **void** main(String[] args)

**throws** InterruptedException {

AnotherRun anotherRun = **new** AnotherRun();

Thread childTread = **new** Thread(anotherRun);

childTread.start();

**for** (**int** i = 0; i < 3; i++) {

System.out.println("m" + i);

Thread.sleep(1000);

}

childTread.join();

System.out.println("End");

}

}

**class** AnotherRun **implements** Runnable {

@Override

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.out.println("r" + i);

**try** {

Thread.sleep(1000);

}

**catch** (InterruptedException e) {

System.out.println("Interrupt");

}

}

}

}

Результат

m0

r0

m1

r1

m2

r2

r3

r4

End

Здесь обработка исключений необходима из-за статического метода sleep(), который приостанавливает выполнение **текущего** потока. Данный метод часто используют в дочерних потоках, когда они должны выполнять какое-либо действие постоянно, но не бесперебойно. Например, периодически проверять доступность ресурса.

Метод join() заставляет текущий поток ждать завершения нити, к которой применяется. Только после этого текущий поток может продолжить выполнение своего кода.

В данном случае мы создаем класс-наследник от Runnable. Объект типа Runnable или его производное передается в конструктор объекта типа Thread. После этого поток запускается

*Щас будет пример когда пользовательский класс являеться наследником Thread:*

**public** **class** ThreadTest1 {

**public** **static** **void** main(String[] args)

**throws** InterruptedException {

AnotherTask thread = **new** AnotherTask();

thread.start();

**for** (**int** i = 0; i < 3; i++) {

System.out.println("m" + i);

Thread.sleep(1000);

}

thread.join();

System.out.println("End");

}

}

**class** AnotherTask **extends** Thread {

@Override

**public** **void** run(){

**for** (**int** i = 0; i < 5; i++) {

System.out.println("r" + i);

**try** {

Thread.sleep(1000);

}

**catch** (InterruptedException e) {

System.out.println("Interrupt");

}

}

}

}

Этот вариант не подходит, если класс для организации отдельного потока должен наследоваться от другого класса (не Thread). Поскольку в Java нет множественного наследования классов, приходится использовать наследование от интерфейса Runnable. Также данный подход не дает возможности запускать несколько потоков на основе одного объекта. Так в первом примере мы могли бы передать единственный объект anotherRun в несколько объектов типа Thread.

**Напомним, библиотечный класс Thread сам является наследником Runnable.**

Если в отдельный поток обособляется небольшая подзадача, можно использовать неименованный класс:

**public** **class** ThreadNoName {

**public** **static** **void** main(String[] args) {

Thread thread = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

System.out.println("hi");

}

});

thread.start();

Thread thread1 = **new** Thread() {

@Override

**public** **void** run() {

System.out.println("hello");

}

};

thread1.start();

}

}

**Прерывание потоков**

Для прерывания выполнения нити, если это необходимо, используется метод interrupt(), который устанавливает переменную isInterrupt в значение true. К коде пользовательского класса, унаследованного от Runnable/Thread, это переменная должна проверяться. Отсюда следует, что на самом деле в ***Java нет возможности прервать поток извне, поток может остановиться только сам.***

С другой стороны, в метод sleep() уже встроена проверка переменной isInterrupt, поэтому проверку вручную опускают. Если sleep() считывает наличие прерывания, то генерирует исключение.