Многопоточность

**Многопоточность (multithreading) –** это принцип построения программы, при котором несколько блоков кода могут выполняться одновременно.

***Синхронное программирование*** – это когда процессы идут последовательно друг за другом.

***Асинхронное программирование*** – это когда возможно переключение между процессами, или выполнение процесса во время того, как запущен другой. Это и есть multithreading, concurrency etc.

*Например, задачу по подсчету суммы чисел от 1 до 1 млрд можно выполнить за цикл одним потоком, а можно разделить по 250 млн и создать 4 потока, каждый из которых будет считать свою часть, а потом просуммировать результат.*

**Цели многопоточности:**

1. Выполнять одновременно (или псевдопараллельно) несколько действий
2. Ускорить вычисления

**Concurrency** (согласованность) – общий термин, включающий в себя **parallelism** и **context switch** (псевдопараллельности). По сути, это выполнение сразу нескольких задач.

***Parallelism*** – выполнение 2-х и более процессов одновременно. В компьютерах с несколькими ядрами Concurrency может быть реализовано через Parallelism

***Context Switch*** – это быстрое переключение между процессами, иллюзия параллельности. В скорости так не выиграть, зато можно обеспечивать процессы, такие как msWord, например – где человеку кажется, что процессы происходят одновременно, хотя на самом деле идет быстрое переключение между потоками-задачами.

*Создание потоков*

**Существует 2 способа создания потоков:**

- наследуясь от класса **Thread** (который имплементит интерфейс Runnable)

Class MyThread

- напрямую имплементируя интерфейс **Runnable**

**Runnable** – функциональный интерфейс, метод **run()** которого легко перезаписывается через анонимные классы и лямбды.

**1 способ – через класс:**

**public class** ThreadExample {  
  
 **public static class** MyThread **extends** Thread {  
  
 @Override  
 **public void** run() {  
 **for** (**int** i = 1; i <= 10; i++) {  
 System.***out***.printf(**"%s: %d%n"**, Thread.*currentThread*().getName(), i);  
 }  
 }  
 }  
  
 **public static void** main(String[] args) {  
 MyThread mt1 = **new** MyThread();  
 mt1.start();  
 **new** MyThread().start(); // можно быстрое создание через аноним класс

**2 способ – через интерфейс:**

**public class** ThreadExample {  
  
 **public static class** MyThread **implements** Runnable {  
  
 @Override  
 **public void** run() {  
 **for** (**int** i = 1; i <= 10; i++) {  
 System.***out***.printf(**"%s: %d%n"**, Thread.*currentThread*().getName(), i);  
 }  
 }  
 }  
  
 **public static void** main(String[] args) {  
 Thread t1 = **new** Thread(new MyThread());  
 t1.start();  
 **new** Thread(new Mythread()).start(); // тоже можно быстрое создание через аноним класс

или

new Thread (()-> {

for (int i = 1; i <= n1; i++) {

System.out.println(Thread.currentThread().getName()+ " " + i);

}

} ).start();

**или еще по-модному:**

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

Runnable r = () -> {  
 **for** (**int** i = 1; i <= 10; i++) {  
 System.***out***.printf(**"%s: %d%n"**, Thread.*currentThread*().getName(), i);  
 }  
};  
**new** Thread(r).start();

}

**+ если нам нужно сделать так, чтобы в run() потока были входящие параметры:**

public class ThreadExwithConstrandInter {

public static void main(String[] args) {

try (Scanner scan = new Scanner(System.in)) {

System.out.println("Введите число 2");

int n1 = Integer.parseInt(scan.nextLine());

System.out.println("Введите число 2");

int n2 = Integer.parseInt(scan.nextLine());

new Thread(getrun(n1)).start();

new Thread(getrun(n2)).start();

new Thread1(n1).start();

new Thread1(n2).start();

}

}

private static Runnable getrun(int n) { //тут у нас return type Runnable – значит, можно хреначить Л-ду

return () -> {

for (int i = 1; i <= n; i++) {

System.out.println(Thread.currentThread().getName() + " " + i);

}

};

}

}

class Thread1 extends Thread {

int n;

public Thread1 (int n) { //конструктор с параметром, если создаем как класс

this.n=n;

}

@Override

public void run() {

for (int i = 1; i <= n; i++) {

System.out.println(Thread.currentThread().getName()+ " " + i);

}

}

}

**Выводы:**

**- *Можно создавать разные потоки от одного класса или интерфейса*** *и запускать их, и они будут работать параллельно:*

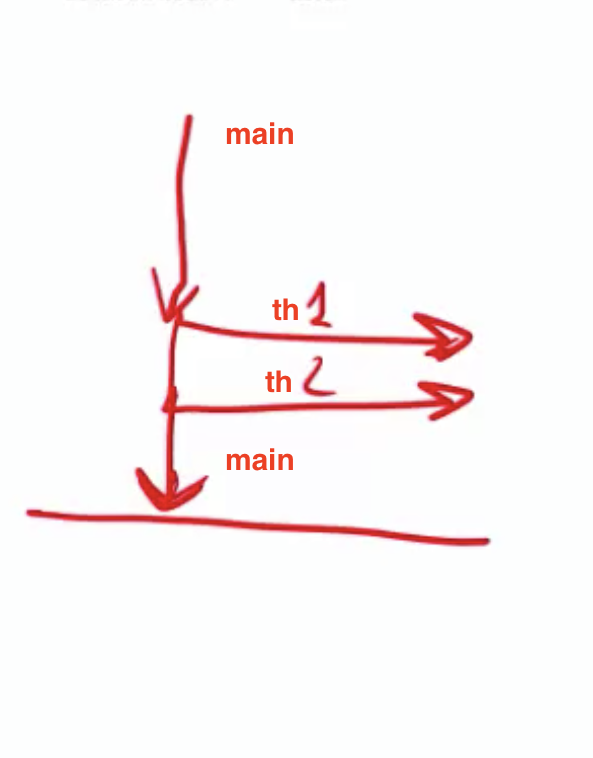
**MyThread1 th1=new MyThread1 ();**

**MyThread1 th2=new MyThread1();**

***- Второй способ создания потоков (через интерфейс) предпочтительнее,*** *т.к. обычно с ним создание потоков проще и короче, а также остается возможность для наследования этого класса от какого-нибудь другого (кроме Thread).*

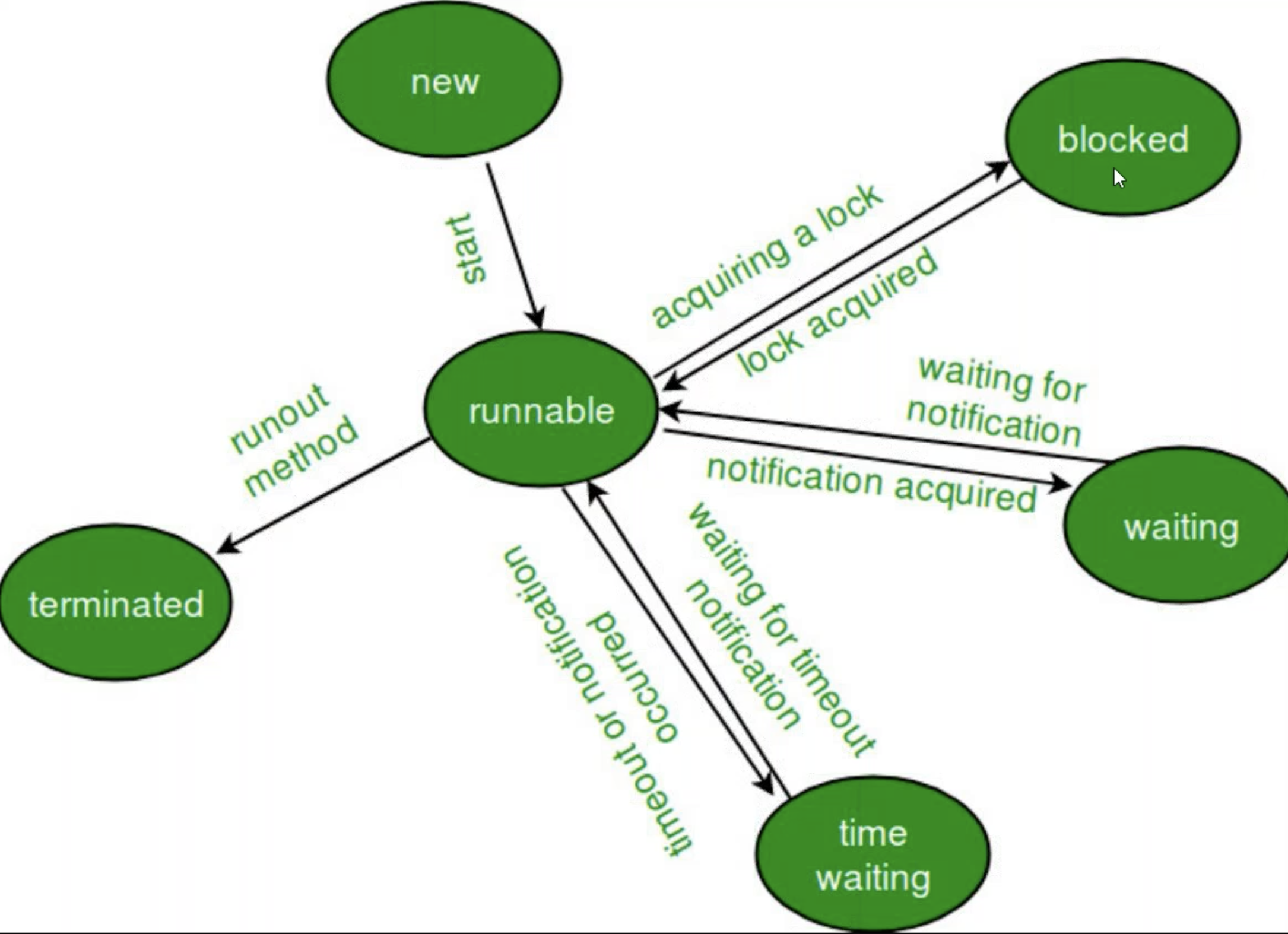
**- *В методе Main – свой поток.*** *Если в нем создать задачу, помимо запуска потоков th1 и th2, она отработает с ними параллельно*

***-Main не обязан ждать завершения потоков, которые запущены в нем.*** *Программа закончится, когда они доработают, а сам main может не ждать, а завершиться.*

**

***МЕТОДЫ THREAD***

* **Для запуска потоков необходимо использовать метод th.*start()* -** Через run() ничего не произойдет, тк запустится метод run() потока main, в котором он не перезаписан.
* ***Thread.currentThread***() – иногда нужно указывать перед тем, как след. метод писать
* ***Thread.setName() / Thread.getName()*** – можно назначить имя потоку или запросить его имя
* ***Thread.setPriority() / Thread.getPriority()*** – можно назначить приоритет потоку или запросить его приоритет
* ***Thread.sleep(t);*** - усыпить поток – в параметрах указывается t сна в мс !Бросает InterruptedException!
* ***th.join();*** - это значит, что тот поток, в котором вызван джоин должен дождаться окончания потока на котором вызван метод и только после этого продолжиться или завешиться. !Бросает InterruptedException!
* ***th.getState()*** – выведет состояние потока в данный момент времени. Нужно быть внимательнее, тк пока выведет, поток может быть уже в др состоянии.



**public class** ThreadMethods {  
  
 **public static void** main(String[] args) **throws** InterruptedException {  
 System.***out***.println(Thread.*currentThread*().getName());  
 System.***out***.println(Thread.*currentThread*().getId());  
 System.***out***.println(Thread.*currentThread*().getState());  
  
 Thread t = **new** Thread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **for** (**int** i = 1; i <= 10; i++) {  
 System.***out***.printf(**"%s: %d%n"**, Thread.*currentThread*().getName(), i);  
 }  
  
 **try** {  
 Thread.*sleep*(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 **for** (**int** i = 11; i <= 20; i++) {  
 System.***out***.printf(**"%s: %d%n"**, Thread.*currentThread*().getName(), i);  
 }  
  
 }  
 });  
  
 System.***out***.println(t.getState());  
 t.start();  
  
 Thread.*sleep*(100);  
 System.***out***.println(t.getState());  
  
 Thread.*sleep*(1000);  
 System.***out***.println(t.getState());  
 }  
}

Вывод:

main

1

RUNNABLE

NEW

Thread-0: 1

Thread-0: 2

Thread-0: 3

Thread-0: 4

Thread-0: 5

Thread-0: 6

Thread-0: 7

Thread-0: 8

Thread-0: 9

Thread-0: 10

TIMED\_WAITING

Thread-0: 11

Thread-0: 12

Thread-0: 13

Thread-0: 14

Thread-0: 15

Thread-0: 16

Thread-0: 17

Thread-0: 18

Thread-0: 19

Thread-0: 20

TERMINATED